FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Scalability for Volume Rendering and Information Visualization Approaches in the Context of Scientific Data

DISSERTATION

zur Erlangung des akademischen Grades

Doktor/in der technischen Wissenschaften

eingereicht von

Dipl.-Ing. Philipp Muigg
Matrikelnummer 0125958

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

Diese Dissertation haben begutachtet:

(Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller)	(Assistant Prof. Dipl.-Ing. Dr.techn. Markus Hadwiger)

Wien, 24.05.2012

(Dipl.-Ing. Philipp Muigg)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Scalability for Volume Rendering and Information Visualization Approaches in the Context of Scientific Data

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Dipl.-Ing. Philipp Muigg
Registration Number 0125958

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

The dissertation has been reviewed by:

(Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller)	(Assistant Prof. Dipl.-Ing. Dr.techn. Markus Hadwiger)

Wien, 24.05.2012

(Dipl.-Ing. Philipp Muigg)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Philipp Muigg
Breitenfurterstraße 376/6/13, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Dipl.-Ing. Philipp Muigg)

Abstract

Data from numerical simulations that model physical processes has to be explored and analyzed in a broad range of different fields of research and development. Besides data mining and statistics, visualization is among the most important methods that grant domain experts insight into their complex simulation results. In order to keep up with ongoing improvements of simulation methods as well as ever increasing amounts of data, state-of-the-art visualization techniques have to be scalable with respect to many different properties. Many numerical models rely on a domain decomposition defined by a volumetric grid. Finer grids yield more accurate simulation results at the cost of longer computing times. The wide availability of high-performance computing resources has resulted in increasingly detailed data sets. The first volume rendering approach that is presented in this thesis uses bricking and resampling to cope with such high resolution data. Important regions of the simulated volume are visualized in as much detail as possible whereas lower resolution representations are used for less important portions of a data set. This allows for interactive frame rates even when dealing with the highly detailed grids that are used by state-of-the-art simulation models. Grid resolution, however, is only one aspect that has increased due to the ongoing development of numerical methods. Grid complexity has increased as well. While initial simulation techniques have required simple tetrahedral meshes current methods can cope with polyhedral cells that allow for increased solver efficiency and simulation accuracy. The second volume visualization algorithm that is presented in this thesis is scalable with respect to grid complexity since it is capable of directly visualizing data defined on grids which comprise polyhedral cells. Raycasting is performed by using a novel data structure that allows for easy grid traversal while retaining a very compact memory footprint. Both aforementioned volume rendering techniques utilize the massively parallel computing resources that are provided by modern graphics processing units.

Many information visualization methods are designed to explore and analyze abstract data that is often high dimensional. Since improvements in the field of numerical modelling have led to simulation data sets that contain a large number of physical attributes the application of techniques from the field of information visualization can provide additional important information to domain experts. However, in order to apply information visualization methods to scientific data such as numerical simulation results, additional scalability issues have to be addressed. This thesis introduces multiple methods that can be used to reduce cluttering and overdrawing problems for line-based techniques such as phase-space diagrams, parallel coordinates and a novel time-series visualization. The trajectories of important trends in the data are illustrated by blurring a noise texture along them. A novel coloring scheme is used to provide visual linking-

information across multiple visualizations in a multi-view framework. The proposed approaches are primarily image-based which makes them very scalable with respect to data set sizes.

The usefulness and real-world applicability of the techniques that are introduced in this thesis is demonstrated in a case study. A complex computational fluid dynamics data set, which contains several simulated breathing cycles within the human upper respiratory tract, is analyzed. The exploration of the data has yielded several hypothesis that are of importance to an ENT specialist. Many of the techniques presented in this work have also been used in the context of additional collaborations in a multitude of fields such as medicine, climatology, meteorology, and engineering.

Kurzfassung

Sowohl in vielen Forschungsbereichen, als auch im Ingenieurwesen werden numerische Simulationen physikalischer Prozessen eingesetzt, um experimentelle Untersuchungen zu ergänzen oder gar zu ersetzen. Visualisierung gehört neben Datamining und Statistik zu den wichtigsten Werkzeugen, um die resultierenden Datenmengen zu analysieren und zu explorieren. Um mit den ständigen Weiterentwicklungen im Simulations und Modellierungsbereich mitzuhalten, müssen moderne Visualisierungstechniken flexibel und skalierbar sein. Viele numerische Methoden zerlegen den zu simulierdenden Bereich durch ein Gitter in einzelne Zellen. Durch den Einsatz von zusätzlicher Rechenleistung können feiner aufgelöste Gitter eingesetzt werden, um genauere Simulationsergebnisse zu erzielen. Nachdem nun Hochleistungsrechner eine immer weitere Verbreitetung finden, werden in zunehmendem Ausmaß sehr hoch aufgelöste Gitter eingesetzt. Der erste Volumsvisualisierungsansatz, welcher in dieser Arbeit beschrieben wird, benützt eine Zerlegung der Simulationsdaten in einzelne Blöcke und eine Resamplingmethode, um solch hoch detaillierte Simulationsgitter verarbeiten zu können. Für den Benutzer wichtige Datenregionen werden sehr genau dargestellt, während Teile geringerer Relevanz durch eine weniger detaillierte Repräsentation visualisiert werden. Durch diesen Ansatz ist eine interaktive Darstellung von großen und detaillierten Volumsdatensätzen möglich. Entwicklungen im Simulationsbereich haben neben der Erhöhung der Auflösung von Simulationsgittern auch zu einer Steigerung der Gitterkomplexität geführt. Während erste numerische Modelle noch auf rein tetrahedrische Gitter angewiesen waren, sind aktuelle Simulationstechniken in der Lage auf einer Volumszerlegung in beliebige polyhedrische Zellen zu arbeiten. Der zweite, in dieser Arbeit vorgestellte Volumsvisualisierungsansatz ermöglicht es direkt auf solch komplexen Gitterstrukturen definierte Daten darzustellen. Hierzu wird Raycasting eingesetzt, welches auf einer neuen hoch effizienten und sehr kompakten Datenstruktur arbeitet. Beide zuvor erwähnten Volumsvisualisierungsmethoden setzen die parallelen Rechenresourcen moderner Grafikhardware ein, um eine interaktive Darstellungsgeschwindigkeit zu erreichen.

Viele Informationsvisualisierungstechniken sind zur Analyse und Exploration von abstrakten, meist hoch dimensionalen Daten konzipiert. Entwicklungen im Bereich der numerischen Modellierung haben zu Simulationsmethoden geführt, welche eine Vielzahl an physikalischen Attributen berücksichtigen. Informationsvisualisierung kann nun eingesetzt werden, um tiefere Einblicke in solch komplexe Simulationsresultate zu gewinnen. Allerdings müssen hierzu einige Probleme hinsichtlich der Skalierbarkeit von Informationsvisualisierungsmethoden gelöst werden. Die Ansätze, welche zu diesem Zweck in dieser Arbeit beschrieben werden, helfen dabei, Visualisierungen übersichtlich zu halten, auch wenn eine große Anzahl an Datenpunkten dargestellt werden muss. Der Verlauf von wichtigen Trends wird durch das Verwischen einer Textur,

welche weißes Rauschen enthält, verdeutlicht, während ein neuartiges Farbschema Zusammenhänge zwischen verschiedenen Visualisierungsmethoden darstellt. Die besprochenen Ansätze sind sehr skalierbar was die Anzahl der Datenelemente betrifft, da sie größtenteils bildbasiert sind.

Um die Effektivität der vorgestellten Methoden zu demonstrieren, enthält diese Arbeit ein ausführliches Anwendungsbeispiel. Ein komplexer Computational Fluid Dynamics Datensatz, welcher mehrere simulierte Atemzyklen innerhalb der oberen Atemwege eines Menschen enthält wird mit Hilfe der zuvor besprochenen Visualisierungsansätzen analysiert und exploriert. Hierbei sind für einen HNO Spezialisten wertvolle Hypothesen entstanden. Neben diesem Anwendungsbeispiel sind die im Zuge dieser Dissertation entstandenen Techniken in weiteren Bereichen wie Medizin, Klimatologie, Meteorologie und Maschinenbau erfolgreich eingesetzt worden.

Contents

CHAPTER 1

Introduction

Steady improvements of CPU architectures and manufacturing techniques have led to an abundance of cheap and powerful processing resources that are available to many different fields. Therefore numerical simulation methods that can be used to model physical processes have gained high importance in the context of research and development. This is especially true for cases where relying on experimentation and measurements is either infeasible or prohibitively expensive. Biologists, for example, utilize massive processing resources to analyze molecular interactions, cosmologists model the evolution of the universe and engineers test their newest engine designs. All these simulation techniques generate increasing amounts of data that has to be analyzed with various tools. Besides statistics and data mining techniques visualization is crucial for gaining insight into such data sets. Especially interactive techniques that involve the user in data exploration and analysis tasks have gained increasing importance. While systems that rely on batch processing and scripting are still important visualization tools, many state-of-the-art visualization methods have to be able to provide immediate visual feedback to the user input. Achieving this interactivity for data sets whose size and complexity are steadily growing is the central motivation that connects all approaches introduced in this work.

The visualization techniques presented in this thesis are primarily applicable to the results generated by computational fluid dynamics (CFD) simulations that are used to predict the physical behavior of fluids and gases. CFD methods rely on a decomposition of the simulated volume into cells. For each cell, physical properties are simulated by defining initial boundary conditions and solving partial differential equations that represent basic physical laws such as conservation of mass or energy. Visualization techniques used to explore/analyse/present CFD simulation results have to be scalable with respect to different data set properties in order to keep up with the growing amounts of data. On the one hand increasingly powerful CPUs allow for higher-resolution meshes which in turn increase the accuracy of a simulation. This results in data sets containing an ever increasing number of cells that have to be considered during visualization. On the other hand simulation methods themselves are steadily evolving as well. Mesh data structures have become increasingly flexible by allowing more general cell geometries. Therefore visualization methods have to cope with higher geometric complexity when dealing with

grids containing such cells. Increasingly complex physical phenomena can be simulated due to the development of new modeling methods. The resulting data sets contain solutions not only for fluid flow but also for different kinds of chemical reactions. Again CFD data visualization methods have to be scalable, this time with respect to the number of physical attributes that are simulated for each grid cell.

This thesis covers different visualization techniques and extensions that each address some of the aforementioned scalability issues in a different way. The first part of this work will cover two volume rendering approaches that can be used to visualize CFD data in a spatial context. One approach is highly scalable with respect to the number of cells whereas the second technique is able to deal with highly complex cell and mesh geometries. Contrary to scientific visualization methods, visualization techniques that represent data without relying on the original spatial data organization are commonly classified as information visualization methods. In the second part of this work extensions to different kinds of information visualization techniques will be introduced. These include a coloring scheme that can be used to indicate user interest in a data subset across multiple visualizations and a texture-based technique that improves the scalability of line-based visualizations with respect to data size.

1.1 Contributions

The contributions of this thesis can be divided into two distinct groups, both of which extend the state of the art in their respective sub field of visualization research:

Unstructured Grid Volume Rendering: Two unstructured grid volume rendering methods are introduced that tackle scalability issues with respect to mesh size and mesh complexity. The individual contributions are as follows:

- A *hybrid raycasting method* that decomposes the simulated volume into individual bricks. Depending on available memory resources each of these bricks is either rendered in its original unstructured mesh representation or as a lower quality structured representation that is created via resampling.
- Resampling controlled by a *memory management system* that utilizes statistical data characteristics and a user defined degree of interest function.
- An explicit and exact *mesh boundary visualization* that is consistent across all brick representations and can be combined with different volume rendering styles.
- Direct volume rendering of *different cell types* without the need for tetrahedralization. Mean value interpolation is used to sample data values inside a cell. The method proposed in Section 2.3 is still limited to a small number of convex cell types. The extensions described in Section 2.4 allow for general polyhedral cells.
- A very compact, purely *face based mesh representation* suitable for unstructured grids that comprise general polyhedral cells.

Information Visualization for Simulation Data: Directly applying well-known information visualization techniques such as parallel coordinates [62] to scientific data such as CFD simulation solutions often results in cluttered images that are difficult to read and interpret. The following contributions extend the state of the art in information visualization with respect to scientific data sources such as CFD simulations:

- A consistent coloring scheme that is used throughout a visual analysis framework for CFD simulation data, which combines information visualization and scientific visualization techniques via a multi-view setup. Color is used to relate user defined selections between various views.
- A scalable time series visualization that is capable of displaying large numbers of graphs. Different image-space based techniques are introduced to reduce cluttering.
- A general image-based method for enhancing visualization techniques that require the rendering of a large number of overlapping lines. Anisotropic diffusion of a noise texture is used to indicate general line directions.

In order to illustrate the usefulness of the proposed techniques an elaborate application example is included in Chapter 4.

1.2 Organization

This thesis is subdivided into three major parts. Chapter 2 will introduce novel highly scalable volume visualization techniques. Chapter 3 will focus on different extensions to information visualization approaches. The first sections of both chapters provide a short overview on concepts that are relevant to the respective field of research. The scientific contributions to volume rendering can be found in Sections 2.3 and 2.4. Novel contributions to the field of information visualization are presented in Sections 3.3 and 3.4. Chapter 4 presents a case study that exemplifies how all the techniques that have been introduced in the previous chapters can be combined to explore and analyze a complex CFD simulation data set. The thesis ends with a short summary and conclusions.

CHAPTER 2

Scalable Volume Visualization for Unstructured Grids

2.1 Introduction

Volumetric data visualization has become increasingly important since data acquisition and simulation methods nowadays frequently generate 3D or even 4D (space + time) results. Volumetric data can be defined as a function $f(x) : \mathbb{R}^3 \Rightarrow \mathbb{R}^n$ that assigns a vector of attributes to each position in $\mathbb{R}^3$. Depending on the actual data source n can vary greatly. Medical imaging techniques such as computed tomography scans generate only one scalar attribute whereas complex simulation models often create high-dimensional result-vectors that contain information about fluid flow, physical properties and chemical reactions within a volume. The following section will provide an overview on the different ways volumetric data can be stored, organized, and visualized.

2.1.1 Volumetric Grids

Since $f(x)$ cannot be expressed analytically for most real-world cases, volumetric data is typically represented by using *sampling*. That is, the volume attributes are stored only for a finite number of sample points. Data in between sample points is calculated by using *interpolation*. Sample points can be organized either in a volumetric grid or gridless. Volumetric grids define a decomposition of the spatial domain into non-overlapping cells. Each cell comprises a number of faces, each of which contains a number of edges that are defined by two vertices. Even though data samples can be assigned to all of these objects (i.e., vertices, edges, faces, cells) typically data is either specified per cell (cell-centered data) or per vertex (vertex-centered data). For gridless data representations only the spatial position and sometimes a region of influence are stored per sample point. The volume-rendering techniques presented in this thesis primarily deal with data defined on volumetric grids. In order to visualize gridless data (e.g., from spherical particle

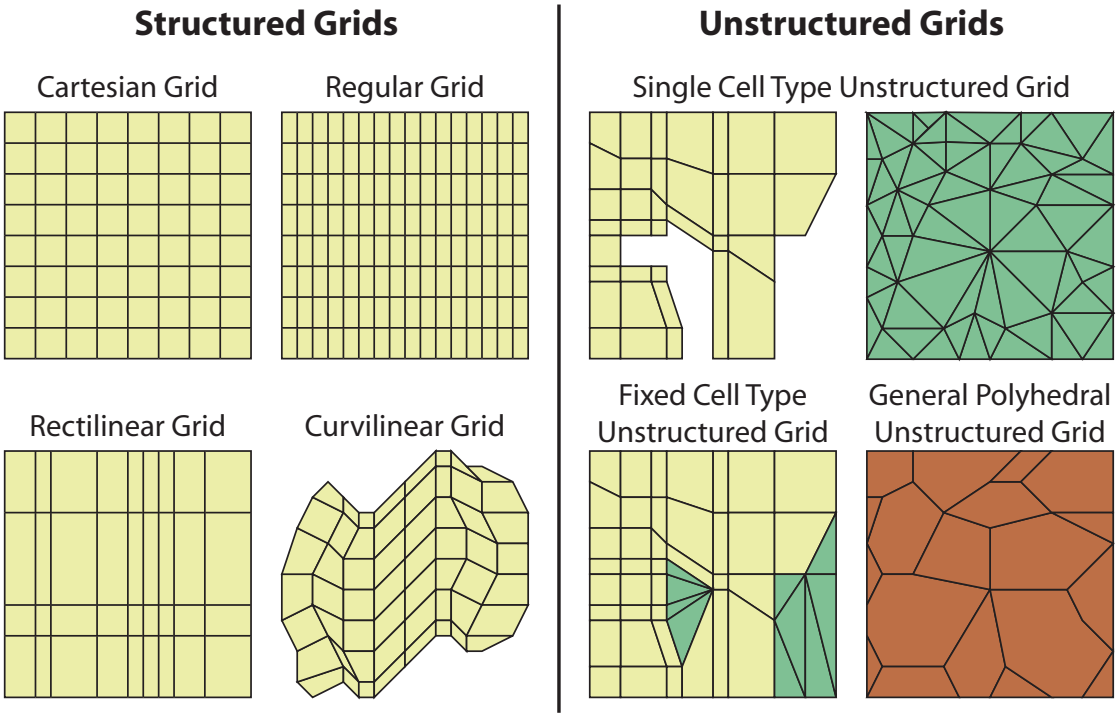


Figure 2.1: Overview of different types of grids that can be used to organize volume data. Note that two dimensional versions of the grids are used to illustrate the mesh topology.

hydrodynamics simulations) a grid has to be created by applying meshing approaches such as Delauney tetrahedralization [39].

Volumetric grids can be classified based on their topological structure and the spatial location of the sample points. As illustrated in Figure 2.1 the simplest grids are *structured* arrangements of hexahedral cells. Each cell can be identified by a three-dimensional indexing-vector that specifies the column, row and layer within the grid. The entire topology of a structured grid is given implicitly. No data structure is required to store the grid connectivity. For example queries such as finding a cell d that is connected by a common face to cell c can be resolved by looking at the respective indexing vectors $c_i, d_i \in \mathbb{N}^3$. If and only if c_i and d_i differ by one in exactly one component, then c and d share a common face. Similarly if and only if c_i and d_i differ in all components by one, then c and d share exactly one common vertex. The classification of structured grids is solely based on grid topology. Further sub groups can be identified when taking the spatial placement of sample points into account. In a *cartesian grid* all grid cells are equally sized cubes. The locations of sample points can be defined by specifying the spatial extent and resolution (i.e., the number of cells along each axis) of the grid. This is also true for *regular* grids. Instead of restricting cells to cubes regular grids contain equally sized cuboids. In both cases the important problem of finding a cell that contains an arbitrary point is straightforward. Solving this so called *point location problem* for the following structured grid types is significantly more challenging and often involves search operations. Similarly to

regular grids *rectilinear grids* comprise only cuboids. Contrary to the regular case cuboids can be of different sizes (see Figure 2.1). In order to define the vertex positions for a rectilinear grid it is sufficient to store the spacing of the vertices along the x, y, and z-axis of the grid. *Curvilinear* grids are even more general than rectilinear grids in that cells are not limited to cuboids anymore. In many application scenarios regular or even cartesian grids are sufficiently flexible to properly represent volumetric data. Prominent examples are data from medical scanning devices such as computed tomography (CT) or magnet resonance tomography (MRT). The primary motivation for choosing a more general grid structure such as a curvilinear grid rather than a regular or cartesian one is the added flexibility with respect to the grids overall shape and the distribution of its cells. This is especially important for numerical simulation techniques such as computational fluid dynamics (CFD). Fluid behavior in the vicinity of a solid surface differs greatly from flow characteristics in an unobstructed region. By adapting the grid shape to the surface of an object and adjusting cell sizes to the expected local flow behavior the overall number of cells can be reduced while still retaining the desired accuracy of a simulation.

Structured grids have many advantages such as their implicit topology representation and the possibility to use central differences to approximate gradients. However the restriction of using only a fixed arrangement of hexahedral cells is still limiting the flexibility of this grid type. This is especially problematic in the context of engineering applications where the physical properties of highly complex components have to be simulated. Adapting cell sizes to the scale of expected physical phenomena is crucial for numerical simulation techniques. For example small cells have to be used in order to properly model turbulent flow in the vicinity of a solid surface. Filling the remaining volume using equally small cells would be prohibitively wasteful since here features are generally larger in scale. In order to allow for the required flexibility many state-of-the-art numerical simulation methods rely on *unstructured grids*. As implied by the name, cells are not organized in a structured way. Cell indexing is performed by using a single integer number that is not related to geometrical properties. Therefore the grid topology and geometry has to be specified explicitly. Chapter 2.4 will provide further details on how grid geometry and topology can be stored efficiently. As with structured grids, the class of unstructured grids encompasses several sub groups. The simplest form of unstructured grid contains only a single type of cell. Typically such grids are composed of either tetrahedral or, less frequently, hexahedral cells. Tetrahedra are selected since every possible volumetric shape can be decomposed into these 3D simplices whereas hexahedra are chosen for their properties with respect to numerical simulations. The restriction to allow only one cell type within a grid primarily allows for memory savings when storing the grid topology. In order to exploit the advantages of tetrahedral and hexahedral grids several state of the art simulation tools can use grids that contain a predefined set of cell types. These include quadratic pyramids and wedges, which are used in transition regions between tetrahedral and hexahedral parts of a grid or close to object surfaces. As long as only predefined cell types are used the topology for each type can be stored in a lookup table. Again this allows for considerable memory savings. The main drawback of using such hybrid tetrahedral/hexahedral grids is that they represent a tradeoff between simulation quality and grid flexibility. Hexahedra exhibit desirable properties with respect to numerical simulations while tetrahedra are far more flexible when grids have to be defined around complex geometrical shapes. In order to alleviate this situation recent developments have led

to the introduction of general polyhedral grids. Two of the leading simulation tools [1, 3] now use grids consisting of cells that may contain an arbitrary number of faces, each of which can be defined by an arbitrary number of vertices. This allows for great flexibility in the vicinity of complex object surfaces and cell shapes that retain geometric properties which are well suited for the simulation process. While memory requirements for storing these grids are typically higher when compared on a cell-by-cell basis (one polyhedral cell requires more memory than a tetrehedral one), proponents of this grid type argue that fewer cells have to be used to achieve the desired simulation accuracy [1, 3]. As with rectilinearly and curvilinearly structured grids, point location for all types of unstructured grids is not possible without resorting to searching. Different kinds of auxiliary data structures can be used to speed up this process.

2.1.2 Volume Data Sources

Volume data can originate from a plethora of different sources. This section will provide a short overview over a selected few of these volume data sources. The first distinguishing characteristic for any data source is whether data has been acquired by measuring the physical world or by a simulation process.

2.1.2.1 Measured Volume Data

Biology and medicine are the primary fields in which large amounts of volumetric data is generated via measurements. Medical patients are scanned by different kinds of 3D imaging devices on a regular basis. Each scanning device generates data with distinct characteristics that frequently require complex pre-processing steps such as noise reduction, filtering or resampling.

Computed Tomography (CT) scans are acquired by combining multiple X-Ray images that have been taken from multiple positions around a patient. X-Rays easily penetrate different kinds of soft tissue in an equal way. Bones, however, absorb a significantly larger percentage of X-Ray radiation. Therefore CT volumes are preferably used to image the bone structure of a patient. CT is also often used in the field of material science to perform non-destructive testing. The mesh of the data set that is analyzed in Chapter 4 is based on a CT scan of the head and upper torso of a patient.

Magnetic Resonance Imaging (MRI) can be used to acquire information about soft tissue. Instead of using X-Rays MRI is based on strong magnetic fields that affect the spin of protons. By using different kinds of MRI modes different tissue types can be identified. CT is primarily used for structural imaging that is acquiring data about a patient's anatomy. MRI is frequently used for both, structural as well as functional data acquisition. MRI perfusion data is based on measuring the accumulation and subsequent flushing of a contrast agent within a certain tissue. Therefore it can be used to assess blood flow through different organs of the body. Section 3.3 provides two application examples that focus on MRI perfusion data acquired from brain and female breast tissue. Functional MRI is used to monitor brain activity by detecting changes in the blood flow and oxygen consumption during cognitive tasks. If medical imaging is used to capture the temporal evolution of a physiological process, individual scans have to be aligned with each other. This task is called *image registration*.

In the following a short list of additional examples for measured three (or even four) dimensional volume data will be given. *Ultrasound* scans measure the run time of ultraound waves that are generated at a probe and are reflected at tissue boundaries to image soft tissue. *Positron Emission Tomography* is based on injecting weakly radioactive substances that accumulate in specific organs. The radiation can be detected and located. In biology different kinds of microscopy-based acquisition-techniques can be used to gather structural and functional data at a cellular level. None of these imaging modalities are used directly in the remainder of this thesis. However many of the techniques presented in subsequent chapters are very general and therefore still widely applicable. With the exception of ultra sound scanning all aforementioned imaging devices generate volumetric data that is specified on a regular grid. Data generated by ultra sound probes is typically defined on a curvilinear grid. However, in order to simplify post processing, the volume is often resampled onto a regular grid.

2.1.2.2 Simulated Volume Data

Even though the following chapters will give some short examples on how to apply the proposed techniques to measured data, the primary application domain for the contributions of this thesis is volume data that has been generated by a simulation process. Simulated data poses different kinds of challenges when compared to data acquired by measurements. First of all, simulation results are often not as noisy as measured data. Therefore noise reduction and/or filtering is frequently not required or even desired. Simulation experts are actually interested in the exact data that has been created by their simulation models. Furthermore transient phenomena can be captured without the need for any kind of image registration since the underlying geometry is a mesh. Both of these aspects simplify pre processing and visualization tasks for simulation data as compared to measured data. As outlined in the introduction, the primary challenge related to simulation data is scalability with respect to different data aspects. Scalability with respect to the size of data that can be processed has become increasingly important since super-computing resources have become widely available. While there are some examples for extremely large measured volumetric data sets (e.g. in electron microscopy based imaging [14]), simulation processes typically generate larger amounts of data on a regular basis. This is especially true for time dependent simulations (e.g. in climate research). Measured data is often defined on regular grids that lend themselves very well to many operations that are required for visualization, such as point location. This is not the case for simulation data that is often specified on increasingly complex unstructured meshes. Additionally complex simulation models often compute a multitude of physical properties per volume cell.

The foundation for most simulation models is the formulation of the physical laws that govern the simulated domain as partial differential equations. *Boundary conditions* are used to define fixed physical properties of parts of the simulation domain such as the enclosing surfaces. For example when simulating fluid flow the boundary condition at an object surface could be the requirement that flow velocity is zero there. Numerical methods are used to solve the governing differential equations with respect to the boundary conditions. Eulerian solution techniques use a frame of reference that is spatially fixed. The simulation domain is decomposed into a large number of cells, for each of which physical properties are simulated. Finite volume (FV) and finite element (FE) methods are some of the most prominent members of this group. Both

can work on unstructured discretizations of the simulated domain. In both cases this discretization has to fulfill several conditions in order to guarantee high accuracy and fast simulation times. Using a large number of small cells increases simulation accuracy while at the same time increasing processing times and memory requirements. The flexibility of unstructured grids allows for using small cells only in regions where small scale features have to be resolved. The remaining portions of the simulated domain are then discretized using larger cells. FE models compute physical properties for the vertices of a mesh whereas FV models generate results that are defined per cell. For FE models the simulation results are vertex centered while FV data sets contain cell centered data.

Contrary to Eulerian techniques, Lagrangian solution methods decompose the substance that should be simulated into discrete particles that interact with each other and the boundaries. In order to query physical properties at a specific position within the simulated volume a search operation has to be used. A small number of the close-by particles is used to interpolate the physical properties at a point. The volume-visualization techniques presented in this chapter are only applicable to data from Eulerian models.

2.1.3 Volume Visualization

The general goal for any visualization technique is to give a user insight into different kinds of data via a visual representation. This goal can be subdivided further into three different scenarios. Each case requires different amounts of apriori knowledge about the data and demands different qualities from a visualization system. When no prior knowledge about a data set is available visualization can be used to *explore*. In this case interactivity as well as providing good summary representations of a data set is important. Visualization techniques suitable for exploratory purposes typically do not generate "high-quality" visuals. Instead a high rendering performance, even when showing large portions or even all of the data, is a core requirement for exploratory visualization tools. After a user has accumulated enough knowledge to form theories and hypotheses of his own during the initial data-exploration, visualization can also be used to perform data *analysis*. For this scenario visualization systems again should be interactive. Additionally providing querying and data-derivation functionality becomes increasingly important. If a user has formed a hypothesis about a specific subset of the overall data, querying techniques should allow for the selection and subsequent visualization of this interesting portions of the data set. Data derivation is used to compute additional measures that can help to confirm or disprove a hypothesis. When the data analysis has been completed a domain expert frequently needs to communicate findings to non-domain experts. In this context visualization is used as a *presentation* tool. Interactivity is less important in this case since the user wants to create images or videos that convey his or her findings. Often complex rendering styles and lighting models are employed in order to make such visualizations more intuitive and expressive.

All subsequently discussed techniques can be used in most of the aforementioned scenarios. The central task for any volume-visualization method is the projection of inherently three dimensional data onto a two-dimensional image plane. Each method can be classified to be either direct or indirect. In the direct case the whole volume is processed during the image creation. Contrary to this indirect approaches rely on an intermediate representation that is extracted from the volumetric data.

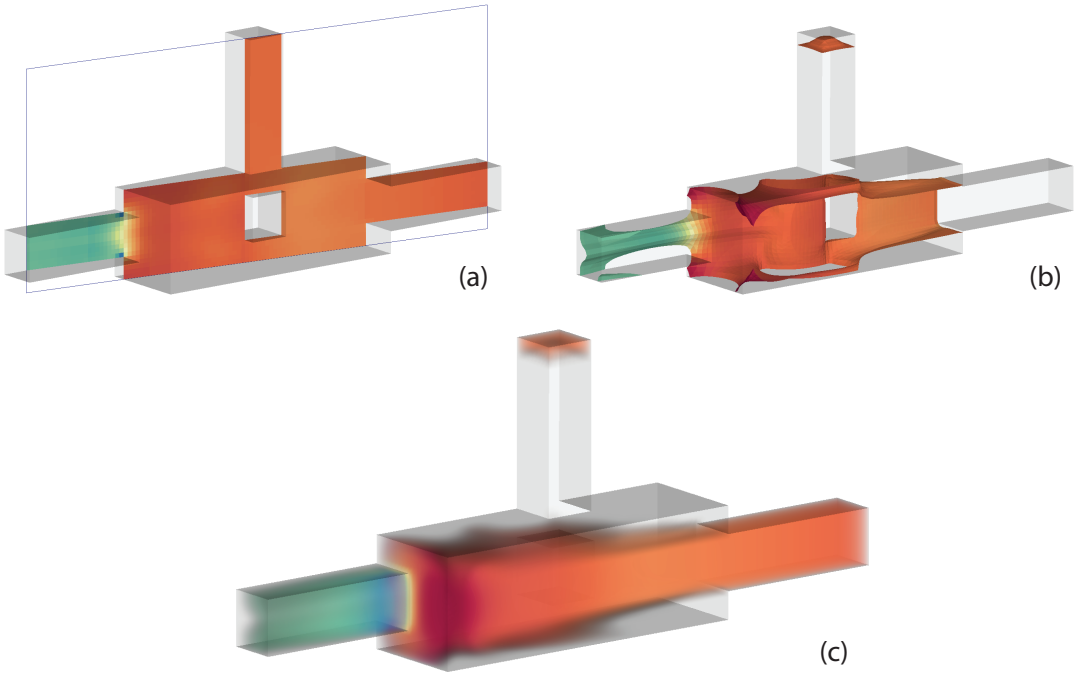


Figure 2.2: Three different types of volume visualization. The data set shown in all three figures contains simulation results for a t-junction into which warm fluid is injected through the right and top inlets. Color encodes pressure. In (a) a single intersection plane is shown whereas an isosurface of the temperature attribute is displayed in (b). (c) has been created by direct volume rendering. Opacity is high for high temperature regions and low otherwise. The t-junction data set is courtesy of AVL List GmbH, Graz, Austria.

2.1.3.1 Indirect Volume-Visualization

The simplest indirect technique is using *slicing planes* (see Figure 2.2 (a)) through the data volume. A single scalar data-attribute can be visualized on the plane by using a *transfer function* that maps data values to colors. Slice planes avoid cluttering and occlusion problems that are often inherent to more complex volume-visualization techniques by selectively showing only small parts of the overall data set. *Isosurfaces* are another way of selectively visualizing volumetric scalar data. An isosurface for isovalue $i \in \mathbb{R}$ is defined as the set of points x for which $f(x) = i$. Techniques such as marching cubes [92] for structured or marching tetrahedra [105] for unstructured grids are used to extract a triangle-mesh representation of an isosurface. The surface depicted in Figure 2.2 (b) has been created by applying marching tetrahedra to a tetrahedral decomposition of the original unstructured mesh. Slice planes as well as isosurfaces are considered to be indirect volume-visualization techniques since the volumetric data itself is not displayed directly. Instead an intermediate two-dimensional surface representation is extracted and projected onto the image plane.

2.1.3.2 Direct Volume-Visualization

Direct volume-visualization methods do not rely on an intermediate representation. Instead every portion of the data volume is considered during image generation. This is achieved by sampling the data volume along viewing rays that originate at the view point and pass through each pixel of the image plane. The way pixel colors are computed based on data values along corresponding viewing rays differs between different direct volume-visualization techniques. In the case of *maximum-intensity projection* (MIP) for example only the highest data value along a viewing ray influences the final image. Contrary to MIP *direct volume-rendering* (DVR) uses a transfer function to assign colors and opacities to data values. An optical model is used to simulate light propagation throughout the resulting colored and semi-transparent volume (see section 2.1.3.3). Figure 2.2 (c) has been created by using direct volume rendering.

Most direct volume-rendering algorithms can be classified either as object-order or image-order methods. This distinction is based on the order in which the cells of a data volume contribute to the final output image (see Figure 2.3 for an illustration). Note that image-order and object-order methods generate very similar results if the same optical model is used.

Object-Order methods generate an output image by iterating over all cells of a volumetric grid. During this iteration process the contribution of each cell to the output image is computed. The order in which cells have to be processed depends on the optical model that is used to simulate light absorption and emission within the data volume. Often cells have to be sorted from front-to-back or back-to-front with respect to the viewing direction. Only the most basic optical models do not require any kind of sorting.

Image-Order techniques iterate over all pixels of the result image. All cells that cover a pixel are queried and their color contributions are accumulated as defined by an optical model. Per-pixel cell-query operations either rely on fast point location or grid-topology data. The former is primarily used if the data volume is sampled on a regular grid. For unstructured grids however, point location frequently requires time consuming search procedures. Each cell contributing to an image pixel intersects a viewing ray that originates at the view point. After the first cell

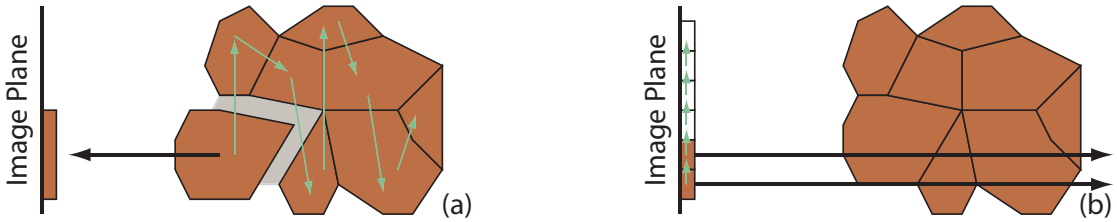


Figure 2.3: Object-order volume rendering is shown in whereas (b) depicts image-order rendering. Object-order approaches iterate over the discretisation of the volume while image-order methods iterate over image pixels. The respective iteration process is depicted as a sequence of green arrows. Object order methods typically require a view point dependent traversal sequence (e.g., front-to-back as shown in (a)).

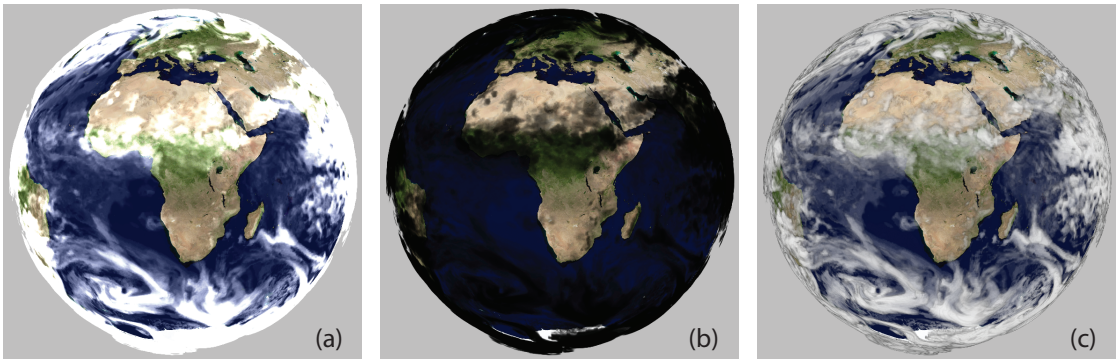


Figure 2.4: Rendering of clouds using three different optical models. In (a) only light emission is considered whereas (b) is based on an absorption-only model. (c) combines absorption and emission. The cloud data set is courtesy of the DKRZ, Hamburg, Germany.

along such a ray has been determined (e.g., by standard point location) subsequent cells can be identified by traversing the mesh topology from cell to cell through connecting faces. This approach avoids time consuming search operations, but limits the cell traversal to either be front-to-back or back-to-front along a viewing ray. This traversal order is actually required by many optical models.

2.1.3.3 Optical Models for Direct Volume-Rendering

An optical volume model describes how light interacts with a semi transparent colored volume. For DVR this volume is traversed along viewing rays for which a final color has to be computed. This traversal process is often formulated as an integral equation that is denoted as the *volume rendering integral*. Max [96] proposes five optical models that differ with respect to how different volume/light interactions are included. In the most complex model light can be scattered multiple times within the volume as participating medium. A phase function is used to describe the local optical properties of the volume. The three simplest models that are relevant for this thesis are simplifications that do not include any kind of light scattering. They are outlined in the following.

Emission Only: This model treats the volume as a light source. Light is only emitted without being absorbed. Figure 2.4 (b) shows a cloud rendering that has been generated by such an emissive volume. For direct volume rendering the equation

$$I(s) = I_0 + \int_0^s g(t)\mathrm{d}t \tag{2.1}$$

has to be evaluated for each viewing ray. I_0 denotes the background brightness. $g(t)$ represents the emitted light at position t within the volume. The integral is evaluated from the background toward the position s on the viewing plane.

Absorption Only: In this model the volume is treated as a semi transparent medium that is only absorbing light. Figure 2.4 (a) shows a light absorbing cloud in front of a textured background. The following equation along a viewing ray has to be solved for direct volume rendering.

$$I(s) = I_0 e^{-\int_0^s \tau(t)\mathrm{d}t} \tag{2.2}$$

This time the background brightness I_0 is attenuated by the accumulated transparency of the volume between positions 0 and s along the viewing ray. $\tau(t)$ represents the extinction coefficient at position t within the volume.

Emission and Absorption: The drawback of the previous two simple models is that they provide only limited depth-cues since the order of features along a viewing ray does not influence the result image. This is not the case when emission and absorption are combined into the optical volume model represented by the following equation.

$$I(D) = I_0 e^{-\int_0^D \tau(t)\mathrm{d}t} + \int_0^D g(s) e^{-\int_s^D \tau(t)\mathrm{d}t} \mathrm{d}s \tag{2.3}$$

The luminance contribution of a portion of the volume at position s is attenuated by the transparency between the view plane at position D and s. Most of the volume-rendering images contained in this thesis have been created by applying this emission and absorption optical model. Therefore a more in depth discussion is provided. In order to evaluate equation 2.3 a numerical approximation has to be used. By replacing the integral in the term $e^{-\int_0^a \tau(t)\mathrm{d}t}$ by a Riemann sum we end up with $e^{-\sum_{i=1}^n \tau(i\Delta x)\Delta x}$. n and Δx denote the number of samples and the corresponding sampling distance respectively. Further simplification yields $\prod_{i=1}^n e^{-\tau(i\Delta x)\Delta x}$. Replacing the integral which accumulates luminance along a ray by another Riemann sum results in the following approximation for $I(D)$:

$$I(D) \approx I_0 \prod_{i=1}^{n} e^{-\tau(i\Delta x)\Delta x} + \sum_{i=1}^{n} g(i\Delta x)\Delta x \prod_{j=i+1}^{n} e^{-\tau(j\Delta x)\Delta x} \tag{2.4}$$

The exponential terms $e^{-\tau(i\Delta x)\Delta x}$ can be interpreted as the overall transparency of ray segment i that has length Δx. Similarly $g(i\Delta x)\Delta x$ is the amount of light emitted by ray segment i. Substituting t_i and g_i for the ray segment transparency and light contribution results in the following simplified equation.

$$I(D) \approx I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j \tag{2.5}$$

The sum in this equation can be written as

$$g_n + t_n(g_{n-1} + t_{n-1}(g_{n-2} + t_{n-2}(g_{n-3} + t_{n-3}(g_{n-4} + ...))))). \tag{2.6}$$

This expression is equivalent to standard alpha compositing that is used in computer graphics to combine semi-transparent surfaces in a back-to-front order:

$$I_i = \alpha_i c_i + (1 - \alpha_i) I_{i-1} \tag{2.7}$$

Alpha values α_i represent voxel opacities and can be written as $\alpha_i = 1 - t_i$. Note that g_i is considered to be an opacity-weighted color in this context, i.e., $g_i = \alpha_i c_i$. The volume-rendering techniques described in sections 2.3 and 2.4 use a front-to-back compositing approach that requires an additional variable T_i which keeps track of the remaining transparency.

$$I_i = T_{i-1}\alpha_i c_i + I_{i-1} \tag{2.8}$$

$$T_i = (1 - \alpha_i)T_{i-1} \tag{2.9}$$

The advantage of this formulation is that after the accumulated transparency T_i has reached a sufficiently small value the summation can be stopped because subsequent samples will barely contribute to the resulting color. This *early ray-termination* technique is used by all volume-rendering approaches that are presented in this thesis.

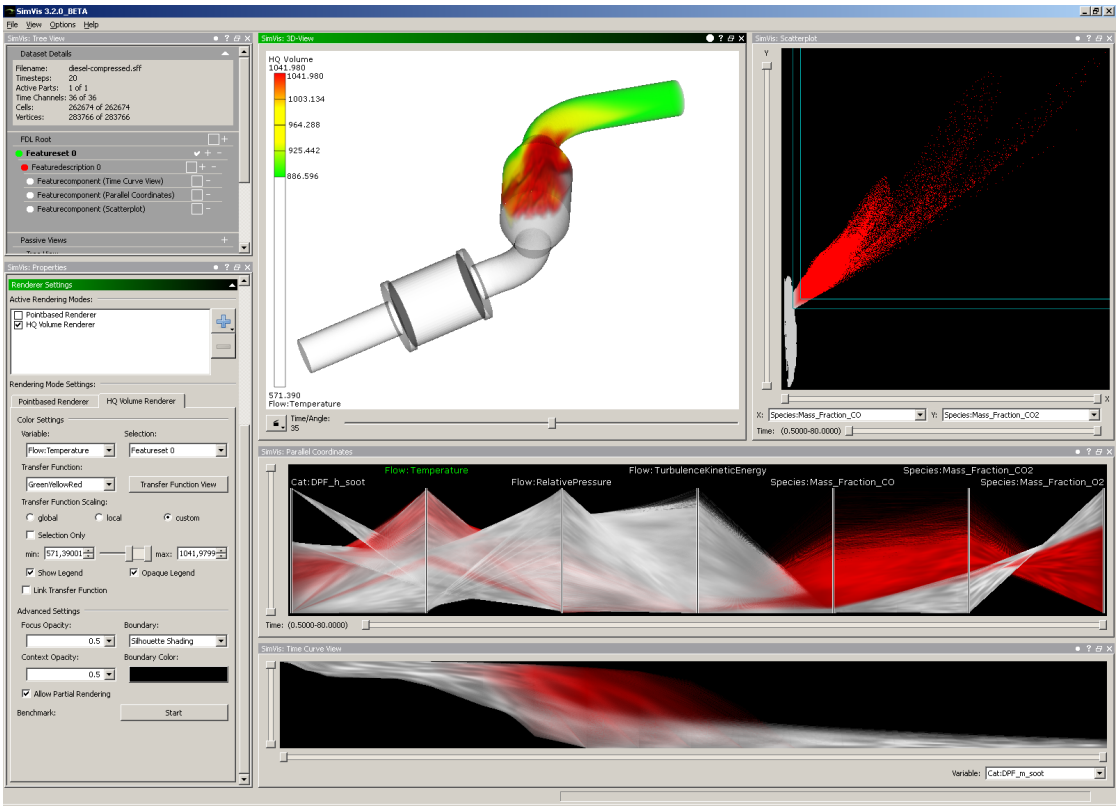


Figure 2.5: The SimVis system into which the subsequently presented volume-rendering techniques have been incorporated. Different views provide an overview on different aspects of the data. While a 3D view can be used to explore the spatial location of features, attribute-space visualization-techniques such as scatterplots or parallel coordinates allow for user driven feature definition and more in-depth data analysis. The diesel particulate filter data set that is shown in this figure is courtesy of AVL List GmbH, Graz, Austria.

2.2 Related Work

The volume rendering techniques discussed in this chapter have been implemented as a plugin for the SimVis system [31]. SimVis employs multiple linked views (Figure 2.5) for concurrently showing, exploring, and analyzing different aspects of multi-variate data. 3D views can be used to visualize features that are specified interactively in several types of attribute views, e.g., scatterplots or histograms. For this feature specification, the user visually inspects selected attributes in order to gain insight into the selected relations in the data. The interesting subsets are then brushed interactively, the result of which is integrated with the data volume in the form of an additional degree-of-interest (DOI) volume. This DOI attribution is used throughout the analysis setup to visually discriminate the specified features from the rest of the data in a focus+context visualization style that is used consistently in all linked views. Focus+context approaches are very well suited for showing the user the actual parts of interest in large data sets [78, 131]. The SimVis system and both rendering techniques that are introduced in this chapter support smooth brushing [32], i.e., fractional DOI values. Selections can be combined using fuzzy logic operators in order to specify more complex features. Chapter 3 provides additional information about the SimVis framework.

The related work that is discussed in this section is split into two distinct fields of research. First, our novel volume rendering approaches for unstructured grids are related to a multitude of volume visualization methods. The method proposed in section 2.4 is based on a novel grid representation, which in turn is related to work conducted in the corresponding research areas.

Unstructured-grid volume-rendering: Unstructured-grid rendering-approaches can be categorized as either object-order methods that iterate over the cells of a mesh, or image-order methods that accumulate data for each image pixel. Both classes require different mesh representations. The Projected Tetrahedra algorithm [119], which is the basis of many object-order techniques, does not require connectivity information between neighboring cells. However, the sorting step necessary to composite individual cells in visibility order is very costly. This has resulted in several approaches that improve sorting performance by utilizing cell-to-cell connectivity information [121, 124, 144]. Object-order methods such as HAVS [22] instead utilize a hybrid CPU/GPU sorting scheme, or avoid sorting by using special order-independent optical models [140, 152], and thus do not require to store cell connectivity. With the exception of complex approximation techniques [95], all these methods are limited to tetrahedral grids. Image-order approaches such as raycasting [48, 53, 139] compute the final image pixel by pixel, by casting viewing rays through the mesh. For rendering performance, cell-to-cell connectivity information is used to traverse the grid. Thus, no sorting step is necessary, which compensates for the slower rendering performance of raycasting. Raycasting methods are also very flexible, e.g., with respect to adaptive sampling, and are a natural choice when complex non-linear interpolation techniques such as mean value coordinates [67] are desired. Since cell-to-cell connectivity is required to traverse the mesh, memory consumption has always been a limiting factor of these methods. Therefore, different data-organization schemes have been developed to minimize the memory footprint [41, 97, 141]. The accuracy of the ray-integration process can be improved by utilizing pre-integration tables [114]. Since some raycasting and cell-projection approaches require a convex volume boundary, convexification methods have been proposed [75, 113]. We

avoid the need for this by employing a method similar to depth peeling [11, 42, 141]. Marmitt et al. [93] provide a more detailed overview of raycasting techniques.

All aforementioned rendering methods target tetrahedral grids and cannot easily be generalized to polyhedral cells, which have become increasingly relevant in state-of-the-art simulation packages [1, 3]. Thus, several approaches have been developed to deal with more complex cell types. Callahan et al. [21] propose a level-of-detail extension to HAVS that selectively removes triangles from a tetrahedral grid. The resulting mesh comprises polyhedral cells and can be rendered by using piecewise linear interpolation. Contrary to the approaches proposed in the following sections, HAVS does not support more complex interpolation methods because only face-to-vertex connectivity information is stored. Lévy et al. [89] extend the half-edge data structure by incorporating additional links for representing general polyhedral cell complexes in a *Circular Incident Edge List (CIEL)*. Figure 2.6 illustrates these extensions. They propose isosurface extraction, as well as slice-based volume rendering algorithms, using CIEL. The main drawback of the CIEL data structure is the high memory consumption of storing half-edges with additional links, as stated by the authors. Additionally, parallelizing CIEL-based volume rendering is difficult, because a single global active edge list has to be maintained during rendering. Adaptive sampling of grid cells of different sizes is also not possible. This is a significant limitation, since the cell sizes in unstructured grids used for simulation can vary by several orders of magnitude. Space-time discontinuous Galerkin simulations are based on even more complex grids containing non-convex curvilinear cells. Üffinger et al. [129] have proposed a volume rendering algorithm for such grids and the corresponding simulations, which not only contain a simple scalar value but a polynomial of varying degree per cell. This polynomial representation allows for using fewer cells to resolve spatially small features. The work focuses mainly on solving problems related to the polynomial representation of the scalar data volume and uses a very basic grid representation.

One way to tackle large data volumes is using resampling or mesh simplification. Many such methods are especially tailored to tetrahedral grids. Mesh simplification uses dual operations such as edge collapses/vertex splits to reduce/increase mesh resolution, respectively [77]. To retain the iso-surface topology for a scalar data attribute, methods developed for structured grids [76] have been applied to unstructured tetrahedral grids as well [23]. In order to selectively control the refinement process and to efficiently store a multi-resolution data set, different methods based on dependency graphs between vertex-split/edge-collapse operations have been proposed [25, 146]. Mesh modification operations depend on each other if the application of one requires the prior application of the other. For example a vertex split that creates vertices A and B from vertex C depends on another vertex split operation that generates vertices C and D from E. Contrary to these methods, sampling only selected vertices or triangles of a mesh has also been proposed [21, 128]. This allows for much more aggressive down-sampling without having to perform individual simplification steps. Resampling techniques create a regular [138, 143], or semi-regular [118], grid representation from the original unstructured volume that can be rendered efficiently by techniques such as raycasting [79]. Resampling can also be performed hierarchically [88]. A recent approach uses a resampling strategy on-the-fly during rendering [49].

In order to tackle large data volumes, and for parallel rendering, many approaches developed

for structured grids employ a spatial subdivision scheme such as octrees [12, 83, 142]. These bricking methods have also been applied in the context of unstructured meshes [109, 132]. Progressive rendering and streaming is also used to cope with large grids [20].

Besides GPU-based unstructured-grid rendering-approaches, highly optimized CPU algorithms have also been developed for volume rendering of scalar data defined on tetrahedral [51] and hexahedral grids [94], as well as point clouds [24].

Unstructured-grid representations: Data structures for representing 2D regular complexes, such as the winged-edge data-structure [9], and the half-edge data-structure [137], are the foundation of many volumetric mesh representations (see Kettner [72] for an overview). Both CIEL [89] and the data structure proposed by Bru and Teillaud [15] are direct extensions of half-edges to 3D grids. The former is highly optimized for isosurface extraction and slice-based volume rendering. It stores redundant linking information in order to speed up traversal. The latter is an extensible general purpose data structure, which stores only one additional link per half edge in a minimal configuration. If necessary, additional linking information can be stored per face, vertex, cell, and half facet. There are two major drawbacks of using such data structures for GPU raycasting. First, the use of an explicit half-edge representation increases the memory footprint, because $2n + m$ half-edges per edge have to be stored and addressed (with n and m being the number of internal and external faces incident to an edge, respectively). This increased memory consumption does allow for greater flexibility with regard to edge-based queries. However, these are not required for raycasting methods. For raycasting one needs fast access to all faces of each cell. This is the second drawback of half-edge representations: In order to sequentially access all faces of one cell, without including additional data, all half edges of that cell must be traversed. This requires using a stack or queue, as well as a way of marking already visited half-edges (e.g., in a hash table), since the underlying half-edge graph contains an arbitrary number of cycles. This is especially problematic for GPU-based visualization. There are

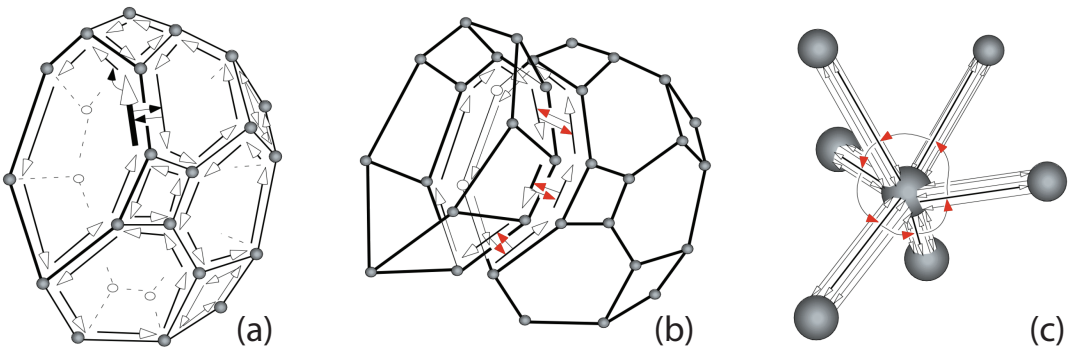


Figure 2.6: The half-edge data-structure [137] has been extended in order to represent the connectivity between polyhedra in an unstructured volumetric mesh. (a) depicts the original winged-edge representation whereas (b) and (c) show the two links that have to be stored in the CIEL representation for each half edge. These additional links are indicated as red-tipped arrows. Image adopted from Lévy et al [89]

also grid representations that are specifically targeting meshes from numerical simulations (e.g., CFD or FEM), which often contain only a limited number of cell types. For example, Alumbaugh and Jiao [6] propose an array-based half-face data structure (AHF), which is capable of representing such volumetric meshes. Similarly to our work, half faces corresponding to one cell are packed in memory in order to reduce the number of necessary links. The lack of flexibility with respect to cell types (only a predefined set with small variation in face count should be used), and the need to utilize additional cell-to-vertex connectivity-data, makes the AHF mesh representation unsuitable for coping with more general polyhedral grids. The simplest type of unstructured mesh contains only tetrahedral cells. Mesh data structures optimized for this case are highly efficient with respect to memory used per tetrahedron. Examples include tetrahedral strips [141], the Compact Half Face (CHF) data structure [82], and its extension proposed by Gurung and Rossignac [52]. Since such algorithms exploit topological properties of tetrahedral cells, a straightforward generalization to arbitrary polyhedra is not possible.

2.3 Scalable Hybrid Unstructured and Structured Grid Raycasting

This section is based on the paper *Scalable Hybrid Unstructured and Strucuterd Grid Raycasting* [103], which has used work conducted by Muigg [100] as a basic foundation. Unstructured grids are an important volumetric representation that is especially common in the field of computational fluid dynamics (CFD), e.g., simulations of engineering problems computed with finite volume methods. Real-world simulation grids often comprise a variety of cell types, such as tetrahedra, hexahedra, octahedra, and prisms. Most rendering approaches convert general unstructured grids into tetrahedral grids in a pre-processing stage and only handle tetrahedra during actual rendering. This subdivision of more complex cells into linear subcells, however, prevents interpolation of a C^1-continuous function within non-tetrahedral cells. Thus, it is desirable for many real-world applications to avoid tetrahedralization for rendering. Another challenge for rendering is the number of cells, which can easily be several hundred thousands to millions of cells. Avoiding tetrahedralization reduces the number of cells that need to be rendered significantly. Figure 2.7 shows a data set of a generator that contains more than six million cells of mixed type (for one time step), which can be rendered interactively by our system.

The four main approaches for GPU-based rendering of unstructured grids are *cell projection* via the Projected Tetrahedra algorithm [119], *raycasting* [139], *resampling* into a structured grid and rendering this grid instead [143], and *point-based approaches* [152]. One of the most important problems when rendering unstructured grids is obtaining a correct visibility order. Cell projection and point-based approaches require explicit visibility sorting, which is a major bottleneck of these methods. Raycasting implicitly establishes the correct visibility without explicit cell sorting. This property is a huge advantage of raycasting approaches and makes them competitive with cell projection when for the latter not only the time for projection and rendering but also for sorting is considered. Our framework employs raycasting as basic rendering method. Iin contrast to similar approaches it does not require the entire grid to be resident in GPU memory due to bricking.

Another important problem is how data interpolation is performed. Although CFD simulations are commonly computed on a per-cell basis (*cell-centered data*), visualization with interpolation usually builds on data values given at the vertices of the grid (*vertex-centered data*). If necessary, conversion between these different representations can be performed. The main problem, is how to perform interpolation when rendering general unstructured cell types. Cell projection is usually restricted to tetrahedral cells and thus linear interpolation, i.e., barycentric interpolation within individual tetrahedra. An important goal of our work is to be able to render the original cells, and thus also to perform consistent interpolation in these cells. A powerful method for interpolating in general polygons are *mean value coordinates* [43], which have been extended for closed triangular meshes [67], and for general polyhedra [84]. Our method offers different interpolation options and builds on mean value coordinates for high-quality interpolation in general cells.

The approach presented in this section renders large, and possibly time-dependent, unstructured grids in real-time by performing raycasting through a hybrid unstructured and structured

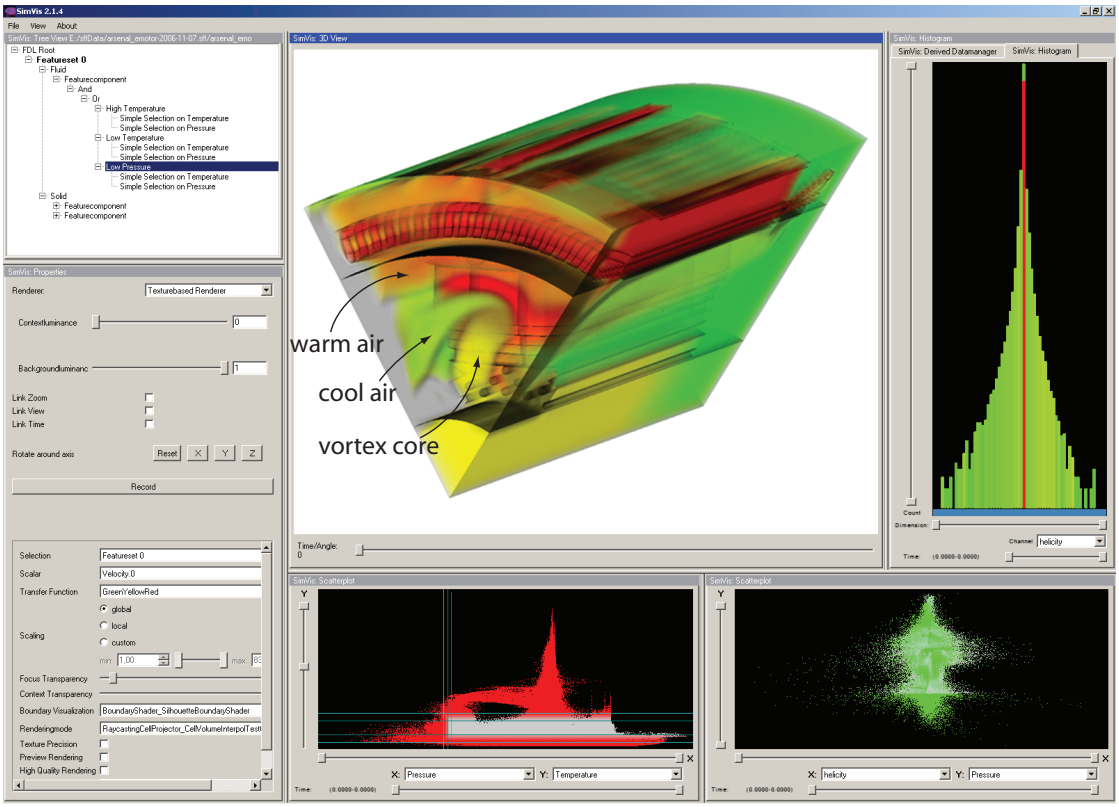


Figure 2.7: The SimVis system with our raycasting view showing high/low temperature (red/green) regions, and low pressure (yellow) regions within the *generator* data set containing more than six million cells. The data has been provided by Arsenal Research, Vienna, Austria and Traktionssysteme Austria (contact: DDr Neudorfer).

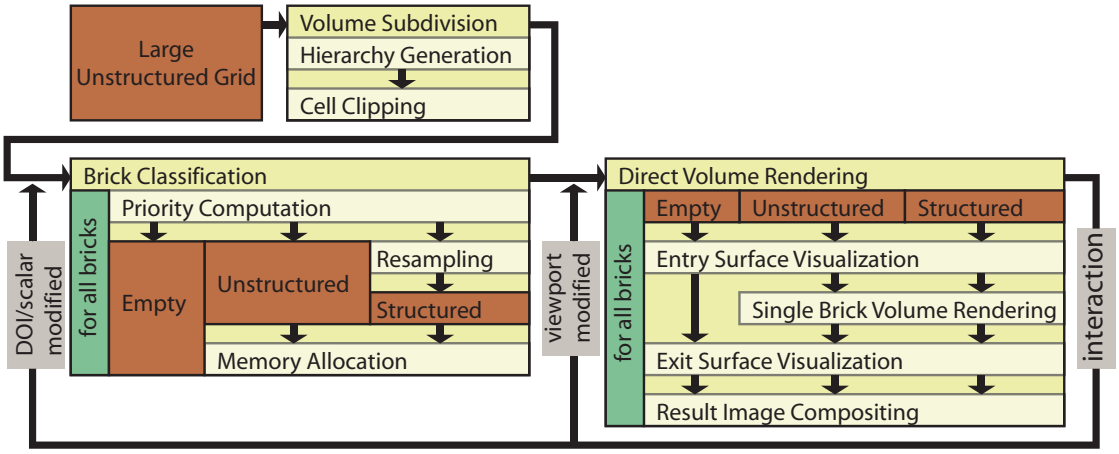


Figure 2.8: Overview of our hybrid raycasting pipeline. In a pre-processing step, the grid is subdivided into a kD-tree hierarchy of bricks. Given the current degree of interest (DOI) function, bricks are classified for rendering as unstructured bricks, resampled into structured bricks, or "empty" bricks that will be rendered as grid boundary only. Dynamic memory allocation is performed depending on brick type. Direct volume rendering proceeds in front to back order, rendering each brick according to its type, and also rendering the geometric grid boundary in correct visibility order.

brick subdivision of the original unstructured grid. We employ a *focus+context* approach where the goal is to render as many bricks as possible using the original cells, especially bricks in the *focus*, and render less important bricks in the *context* using a structured grid obtained via on-the-fly resampling. Focus and context regions are selected via interactive specification of a degree of interest (DOI) function [31]. DOI values in $[0, 1]$ are specified for each grid vertex or cell. They constitute an additional volume that is always used during rendering in addition to measured or simulated data values such as temperature or pressure. Our system handles multi-variate scalar data and at any point in time renders one selected scalar channel such as temperature in conjunction with the scalar DOI function. The original grid cells are never subdivided into tetrahedra, but are rendered directly using one of several options for data interpolation. Scalability with respect to the number of cells is mainly achieved by combining:

- Unstructured and structured bricks (hybrid rendering)
- Different structured brick resolutions
- Different interpolation options and adaptive sampling rates
- Distinguishing between focus and context regions
- Custom dynamic texture memory management

Furthermore, although we are focusing on rendering on a single GPU, our method would also be easily parallelizable by distributing individual bricks to multiple GPUs [125]. To summarize, the major contributions presented in this section are:

- Hybrid raycasting through unstructured and structured bricks
- On-demand resampling steered by interactive focus+context specification
- Rendering of original cell types with interpolation
- Exact grid boundary visualization
- Scalability in image quality and memory consumption

2.3.1 Hybrid Raycasting

Figure 2.8 shows an overview of our hybrid raycasting pipeline. It consists of three major stages. The first stage (Section 2.3.1.1) is a pre-processing step that subdivides the entire unstructured grid into a kD-tree hierarchy by clipping grid cells against kD-tree leaf boundaries. This determines the bricks used in the interactive stages.

The second stage (Section 2.3.1.2) must be invoked whenever the user changes the DOI function or selects a different scalar data channel such as temperature. According to the DOI, bricks are classified in order to determine their importance and corresponding rendering quality and style. Furthermore, the scalar data in a brick also influence the choice of rendering quality. The major choice here is whether a brick should be rendered in its original unstructured form, or as a lower-quality structured brick. The latter are obtained by on-demand resampling whenever a brick that has not been resampled before is required in a structured form. Finally, when the DOI for all values in a brick is zero, it is classified as an "empty" brick for which only the contained geometric grid boundary is rendered.

The final stage is invoked whenever a new view needs to be generated. In this case the kD-tree is traversed in front-to-back visibility order, rendering the contained mesh boundary and raycasting each encountered brick in turn. The results are composited in the output image buffer. As illustrated in Figure 2.9, viewing rays can traverse different brick types. The sampling pattern and type of interpolation along each ray is adapted to the current underlying brick. Especially due to the hybrid nature of our pipeline, the transition between individual bricks must be handled with care (Section 2.3.1.3). Raycasting of individual bricks also depends on their type, such as unstructured bricks (Section 2.3.1.4) or structured bricks (Section 2.3.1.5). For each brick, the front-most mesh surface is rasterized first in order to obtain ray start positions. The mesh surface itself is rendered as well, and direct volume rendering with one of several user-selectable optical models is performed in-between. This ensures that the original mesh boundary is rendered accurately and consistently everywhere and is independent of actual brick types and quality settings. Non-convex parts of the mesh are handled using a depth peeling approach. For "empty" bricks, the mesh boundary is rendered using regular depth peeling [42]. For all other bricks each depth layer is ray-cast individually with proper compositing of layers in front-to-back order [141].

2.3.1.1 Spatial Subdivision

An important advantage of our bricking scheme is that it allows to apply optimizations from structured volume rendering to all brick types uniformly, including unstructured bricks. Each brick is a leaf node of a kD-tree, which is used because it provides better control of the number of entries in its child nodes than similar data structures such as octrees. For each brick, view frustum culling is performed, and empty bricks are skipped entirely (by the volume renderer). This helps with distributing rendering and memory resources only among visible parts of the data. Besides this, rendering individual bricks enables efficiently dealing with highly non-convex meshes. Many data sets from the engineering field have very complex surface geometries (e.g., Figure 2.13), which would require a high number of rendering passes when treated as one entity. Our bricking scheme greatly reduces the overall number of depth peels, because each brick is peeled independently.

The contents of each brick are clipped against the brick boundary in order to guarantee correct compositing of brick contributions during raycasting. Depending on the brick type one of three different clipping methods is used: The intersection polygon for cells in structured bricks is obtained via cell/brick intersection on the CPU. For cells in unstructured bricks, a quad is clipped to the cell interior on the GPU. The boundary of "empty" bricks is clipped using simple clipping planes. The depth of the kD-tree is determined by limiting the number of cells and vertices within each brick to 64 K, which allows for memory savings when storing unstructured grid topology. If time-varying meshes have to be visualized, the brick decomposition is performed for every timestep, except for time-dependent data specified on static grids.

2.3.1.2 Brick Classification

The bricking scheme we employ facilitates distribution of processing and memory resources so that salient portions of the data set are favored over less interesting regions, which is done based on each brick's *importance*. In order to determine the importance of a brick, we combine three simple measures that can be evaluated efficiently. The first measure is the average DOI value within the brick, which represents the user's interest in its contents. The second and third measure are the entropy of the histograms of the DOI and the scalar field within the brick, in order to reflect its information content. All three measures result in values in $[0, 1]$ and are weighted equally. The individual bricks are then sorted based on their importance, and texture memory resources are distributed in this order. The original unstructured representation (Section 2.3.1.4) of a brick is used as long as enough memory is available, while for all other bricks a small 3D texture is allocated and filled on-demand by resampling the unstructured data (Section 2.3.1.5). Since empty bricks do not need any volumetric representation, they are ignored by the memory management system.

2.3.1.3 Inter-Brick Ray Propagation

An important issue in bricked volume rendering is that the transition between individual bricks should not introduce visual artifacts. This issue is especially important in a hybrid bricking scheme such as the one presented here. Specifics of raycasting unstructured and structured

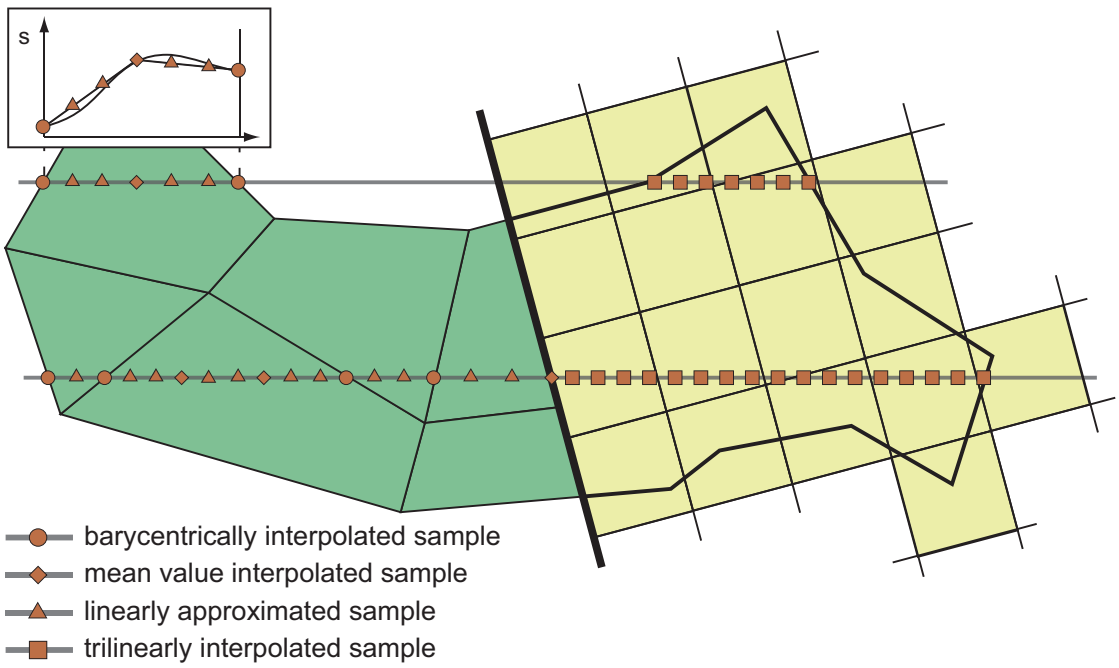


Figure 2.9: Two different viewing rays traversing an unstructured (left), followed by a structured (right) brick. The different sampling types are represented by the colored dots along the rays.

grids, are described in the two following sections. This section highlights common details and describes how rays are propagated from brick to brick.

Each ray cast through a brick starts either at the front-most mesh boundary contained in the brick, or at the intersection of the brick boundary with the interior of an unstructured cell. This is true for both unstructured and structured bricks, and ray start positions are obtained via rasterization of either mesh boundary faces or cell/brick-boundary intersection geometry. Compositing is performed employing two compositing buffers that are used alternately (*ping-pong blending*).

Figure 2.9 shows the two most important cases for ray traversal from one brick to the next. In a single rendering pass, each ray stops either when the mesh interior is exited, which may happen multiple times in a single brick due to non-convex mesh geometry and depth peeling, or when a back face of the brick bounding box is hit. Depending on these cases, raycasting in the next brick continues either from the last location of the previous ray segment (lower ray in Figure 2.9), or continues where the ray re-enters the mesh interior (upper ray in Figure 2.9). Within a brick, non-convex mesh geometry is handled via depth peeling, and thus raycasting may be performed in multiple passes. Each subsequent pass continues at the next ray-entry position beyond the last depth of the previous pass, until the final depth layer has been traversed. The number of raycasting passes is determined via OpenGL occlusion queries, which detect whether an additional depth layer within the current brick exists. Bricking significantly reduces the number of passes due to depth peeling which alleviates its potential performance impact,

Texture Set	r	s Texture Coordinates
Cell	i_x	$i_y + \mathrm{o}_{cell}$
Vertex	v_x	$v_y + \mathrm{o}_{vtx}$
Face*	i_x	$i_y \cdot f_{max} + f + \mathrm{o}_{face}$
Tri Face**	i_x	$i_y \cdot n_{i,tri} + f_{tri} + \mathrm{o}_{tri}(n_{i,tri}, n_{i,quad})$
Quad Face**	i_x	$i_y \cdot n_{i,quad} + f_{quad} + \mathrm{o}_{quad}(n_{i,tri}, n_{i,quad})$

Table 2.1: Addressing in texture sets (Figure 2.11). Each texture set (see Figure 2.11) is stored within multiple 2D textures that are addressed by the two texture coordinates denoted as r and s. Cell and vertex indices are represented as a tuple of two 8-bit values: (i_x, i_y) and (v_x, v_y) respectively. Two different addressing schemes can be used to organize face data. The simple scheme is marked with * whereas the packed scheme is indicated by **. Face indices within a cell are denoted as f in the simple and f_{tri} and f_{quad} in the packed data layouts. Each brick stores data in consecutive rows in a texture set starting at offsets o_{cell}, o_{vtx}, and o_{face}. o_{tri} and o_{quad} represent lookup tables that store one offset per celltype. $n_{i,tri}$ and $n_{i,quad}$ denote the number of triangles and quads of cell i. For the simple memory layout f_{max} stores the highest number of faces per cell within a brick.

because the maximum number of depth layers within a single brick is much lower than for the entire grid.

2.3.1.4 Unstructured Bricks

Our raycasting algorithm for unstructured bricks is based on the work of Garrity [48] and Weiler et al. [139, 141] for tetrahedral meshes. We perform direct raycasting of non-tetrahedral cells without an often ambiguous tetrahedral decomposition, which usually introduces C^1 discontinuities of the interpolated scalar within a cell. Figures 2.10 (a) and (b) show volume renderings of two different tetrahedral subdivisions of an octahedron, with color representing the influence of the top-most vertex. In contrast, Figure 2.10 (c) uses our algorithm without decomposition, and interpolation through mean value coordinates.

Data and Memory Management: Raycasting of one brick is performed for small submeshes of the whole data set, which have to be represented efficiently in texture memory. We employ a custom memory manager to distribute texture resources between bricks. Figure 2.11 shows the four different types of *texture sets* we are using to store the data of unstructured bricks: the *cell texture set*, *triangle face texture set*, *quad face texture set*, and *cell vertex texture set*. The cell texture set stores the number of faces of each cell. For cell-centered data, it also stores the per-cell DOI and the data scalar. The triangle and quad face texture sets store the neighboring cells' indices, face plane equations, and vertex indices corresponding to a face. The cell vertex texture set stores vertex (x, y, z) positions, and per-vertex DOI and scalar data. Each texture set comprises one or more 2D textures of size $256 \times s_{max}$, where s_{max} is the maximally allowed size of a 2D texture. The information of several consecutive bricks is stored back-to-back as long as there is space, and a new texture is only allocated when the previous one is full. For each

brick as many consecutive rows as necessary are allocated, forcing an integral number of rows to make addressing simpler. Thus, a brick containing n_{cell} cells and n_{vtx} vertices uses $\lceil n_{cell}/256 \rceil$ and $\lceil n_{vtx}/256 \rceil$ rows in the cell and vertex texture sets, respectively. In our implementation, all indices are 16-bit integers that are represented by two separate 8-bit components that can directly be stored in two 8-bit color channels, e.g., the red and green channel. Thus, we also create bricks such that $n_{cell} < 64\text{K}$ and $n_{vtx} < 64\text{K}$. Table 2.1 illustrates addressing in texture sets. Every data item is addressed with two 8-bit values i_x and i_y for the index within a row and the address of the row, respectively. Additionally, row base offsets o_{cell} and o_{vtx} must be added in order to perform addressing relative to where the brick's data start in the texture set.

Addressing cell face information is done using indices i_x, i_y, and f, where f is the face index (Table 2.1). There are two alternative schemes for this memory layout and addressing with different trade-offs between access speed and memory consumption: The first method (marked with $*$ in Table 2.1) simplifies address computations, and assumes that all cells contain an equal number of f_{max} faces. Both triangle and quad faces occupy the same amount of space, and for the former the fourth vertex index is simply flagged as invalid. In this case, $\lceil n_{cell}/256 \rceil f_{max}$ rows have to be allocated. The second addressing scheme (marked with $**$ in Table 2.1) allows for a more compact memory footprint. Here, triangle and quad faces are stored in separate texture sets which are addressed independently. All cells are grouped by the number of triangle faces ($n_{i,tri}$) and quad faces ($n_{i,quad}$), also using corresponding row offsets o_{tri} and o_{quad}, respectively. This enables allocating only as many texture rows as necessary for each group, which allows for a tighter packing of face data. Both of these memory layouts have been tested on real world data sets, and are denoted as *simple* and *packed* layout in Section 2.3.3.2.

Raycasting: The raycasting process itself consists of two major parts: ray propagation through the grid, and sampling the scalar function within a cell in order to evaluate the volume rendering integral. Rays are started by rasterizing the surface of the mesh in order to write encoded cell indices into the red and green color channels. Since we limit the maximum number of cells per

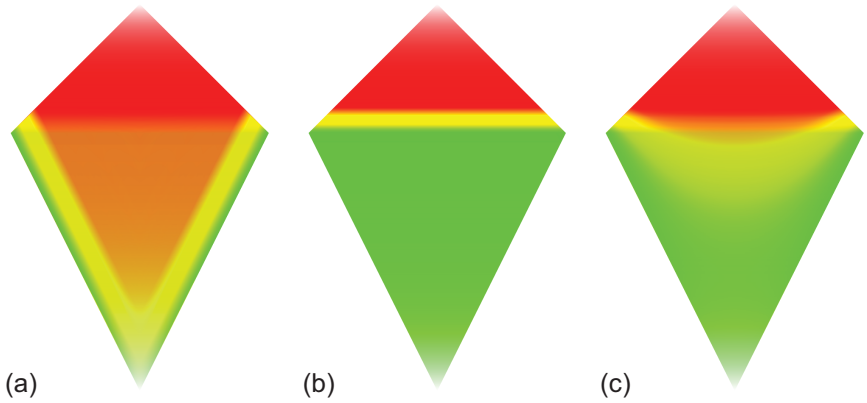


Figure 2.10: Comparison of the influence of the top-most vertex of an octahedron in two different tetrahedralizations (a) and (b), and the original cell using mean value interpolation (c).

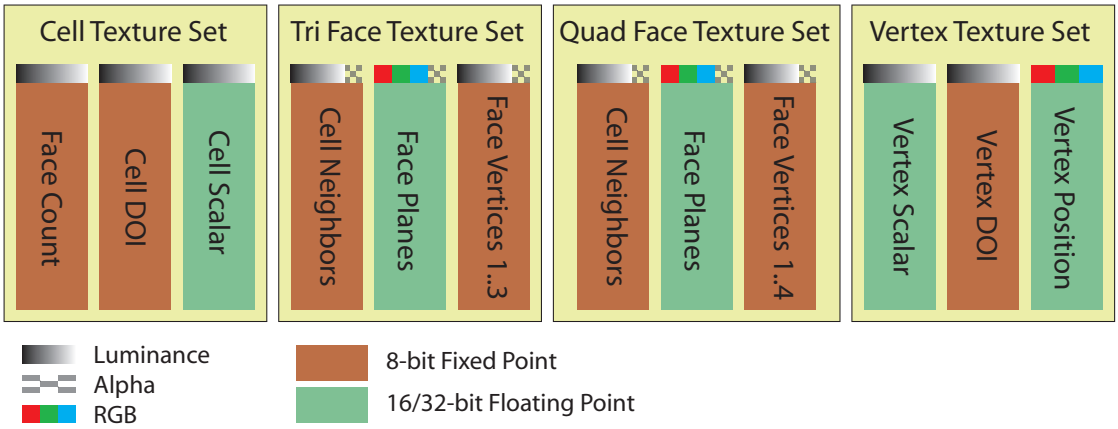


Figure 2.11: Unstructured bricks are stored in *texture sets* consisting of several different 2D textures each. Indices are 16-bit in (8-bit, 8-bit)-tuples.

brick to 64 K, two channels with 8-bits each are sufficient. Additionally, the face through which a ray enters a cell is encoded in the blue channel. If a cell is clipped by the brick boundary, this is indicated in the blue channel as well. During ray propagation, only cell-centered and cell face information is used, which can be addressed directly. This allows for rapid evaluation of the face through which a ray exits the current cell. This is achieved by intersecting the ray with all face planes and choosing the intersection closest to the entry point in the ray's direction [48].

Sampling and Interpolation: The volume rendering integral is approximated by sampling the DOI and scalar data volumes within each cell and compositing the corresponding opacities and colors according to one of several optical models (Section 2.3.2). In contrast to approaches for tetrahedral cells, sampling a single cell multiple times is necessary in order to handle complex cell types. In order to obtain scalar values at arbitrary positions within a cell, we employ mean value coordinates for interpolation as introduced by Floater [43], and generalized to triangular meshes by Ju et al. [67]. An important property of mean value interpolation is that it can be evaluated locally for each face. Thus each vertex weight can be split up into components w_{j_k} that are contributed by the faces j that contain the vertex at index k. Now w_{j_k} can be evaluated for position $\mathbf{x}$ as

$$w_{j_k}(\mathbf{x}) = \frac{\mathbf{n}_{j_k} \cdot \mathbf{m}_j}{\mathbf{n}_{j_k} \cdot (\mathbf{p}_{j_k} - \mathbf{x})}. \tag{2.10}$$

Here, $\mathbf{p}_{j_k}$ is the position of vertex k in face j, and $\mathbf{n}_{j_k}$ the normal of the plane defined by points $\mathbf{x}$, $\mathbf{p}_{j_{k+1}}$ and $\mathbf{p}_{j_{k+2}}$. The vector $\mathbf{m}_j$ is the so called mean vector of face j which can be computed as

$$\mathbf{m}_j = \sum_{k=0}^{2} \theta_{j_k} \mathbf{n}_{j_k} \tag{2.11}$$

θ_{j_k} denotes the angle between $\mathbf{p}_{j_{k+1}} - \mathbf{x}$ and $\mathbf{p}_{j_{k+2}} - \mathbf{x}$. For a more elaborate description of mean value coordinates for meshes we refer to Ju et al. [67], who additionally propose several opti-

mizations to enhance the numerical robustness of their approach in the vicinity of the polyhedron's faces and vertices.

Evaluating all contributions of all cell faces to one sample position means iterating over all faces. This implies that the face data used to perform the interpolation (vertex positions, scalar and DOI data) all have to be available for the evaluation of each sample. The drawback of this approach is that either a large number of temporary registers in the shader has to be used to store all face information or that fetching the data multiple times from the textures would be necessary. We avoid this by computing the scalar and DOI contributions from one face j to all sampling positions before the next face is considered:

$$S_i = \sum_{k=0}^{2} s_{j_k} w_{j_k} \quad DOI_i = \sum_{k=0}^{2} doi_{j_k} w_{j_k} \quad W_i = \sum_{k=0}^{2} w_{j_k} \tag{2.12}$$

Only three values per sample i have to be stored: the current scalar value S_i, DOI value DOI_i, and the accumulated weight W_i which is used to homogenize the samples after all faces of a cell have made their contributions. s_{j_k} and doi_{j_k} denote scalar and DOI data sampled at the face's vertices. The main advantage of this approach is that only the data from one face and one small array storing three values per sample has to be used by the raycasting shader. The size of the array is bounded by the maximum number of mean value samples per cell, which can be specified by the user.

Sample Distribution: The computation of mean value coordinates for a sample is expensive. It involves several cross products, normalizations, and an inverse trigonometric function to compute the angle θ_{j_k}. Therefore it is desirable to distribute mean value samples carefully. Thus, mean value interpolation is only used in a cell's interior, distributing the samples equally along the ray segment within it. Simple barycentric interpolation is applied to compute scalar values at the ray-face intersections. In order to further improve the image quality without sacrificing much performance, additional samples can be computed between the positions of the already computed samples by linearly interpolating between them. If only one scalar volume is used, pre-integration methods from tetrahedral grid raycasting could be applied [114] instead of computing multiple linearly interpolated samples.

In order to steer the performance/image quality of the visualization, two parameters are used by the raycaster for unstructured bricks: The maximum length of a ray segment which can be approximated linearly (l_l), and the maximum length of a ray segment along which a scalar is considered to be constant (l_c). Thus $\lfloor s_l/l_l \rfloor$ mean value samples are computed for a ray segment within a cell, where s_l is the length of the ray segment. We have chosen to distribute the sampling positions evenly to avoid mean value coordinate computations near the face through which the viewing ray exits, which degenerates toward barycentric interpolation anyway. The parameter l_c is used to determine how many samples should be taken in-between two mean value/barycentrically interpolated values by interpolating linearly, as shown in Figure 2.9. Again $\lfloor s_c/l_c \rfloor$ equally-distributed samples are computed and evaluated, where $s_c = s_l / (\lfloor s_l/l_l \rfloor + 1)$.

Both parameters can be used to specify several special cases that can either be used to render preview images or to produce high-quality presentation visualizations. By setting s_l larger than the diameter of the largest cell's bounding sphere, only barycentric samples are computed at the

cell's faces. This reduces rendering complexity tremendously and results in highly interactive framerates even for large data sets. In order to create very high-quality images, s_l can be specified far smaller than the largest cell, and s_c larger than s_l, which forces the raycaster to perform only mean value interpolation.

2.3.1.5 Structured Bricks

If the brick classification step (Section 2.3.1.2) determines that a brick should be rendered in structured form, all unstructured cells contained in or intersected by the brick will be resampled into a single 3D texture. The resampling resolution depends on the number of unstructured cells n in the brick instead of its volume. For this purpose, a uniform distribution within the brick is assumed in order to guarantee memory savings over the unstructured representation. To account for the brick's aspect ratio we first approximate the structured brick's resolution as

$$x = \sqrt[3]{nr_1^2 r_2} \qquad y = \sqrt{(nr_2)/x} \qquad z = n/(xy) \tag{2.13}$$

where $r_1 = \mathrm{width}/\mathrm{height}$ and $r_2 = \mathrm{height}/\mathrm{depth}$ denote the aspect ratio of a brick bounding box. The actual texture resolution along each axis is then selected as the first power-of-two that is not smaller than x, y, and z, respectively.

In this work, we are not focusing on the actual algorithm used for resampling, as any existing method could be used for this purpose, e.g., the algorithm by Westermann [143]. Currently, we are using a very simple and fast CPU-based resampling algorithm that still achieves good results. Basically, the scalar values at the resampling locations are determined by Gaussian splats positioned at the cell centers. The size of these splats is chosen according to the cell size, and modified if necessary such that even cells that are much smaller than a single voxel still contribute to the eight surrounding voxels.

Raycasting of the resulting structured bricks is performed by rasterizing the mesh boundary in order to obtain ray start and end positions, respectively. In order to handle non-convex grids and skip all empty space, we use a modified depth peeling approach [141], where the number of raycasting passes is determined by the number of depth layers. Depth peeling also makes rendering the geometric grid boundary in correct visibility order straightforward. The main difference to regular volume raycasting [79] is that rays are started and stopped at exactly those positions where a ray enters and exits the unstructured mesh, respectively. The mesh boundary is rendered into two floating point textures: The first one for the position of the ray entry point, and the second one for the ray exit point. Depth peeling is also performed separately for ray entry and ray exit positions, respectively. The main reason for this is that common simulation approaches such as coupled heat transport and flow simulation result in meshes where interface faces are duplicated, i.e., both the front-face and the back-face are present in the mesh. Section 2.3.3 describes an example. The sampling rate for structured bricks is chosen to be consistent with unstructured brick rendering. The parameter l_c described in the previous section that determines the maximum distance between two linearly interpolated samples in unstructured bricks also determines the sampling rate of structured bricks. As shown in Figure 2.9 and described in Section 2.3.1.3, the actual positions of samples are chosen such that the visibility of the brick boundary between different brick types is minimized.

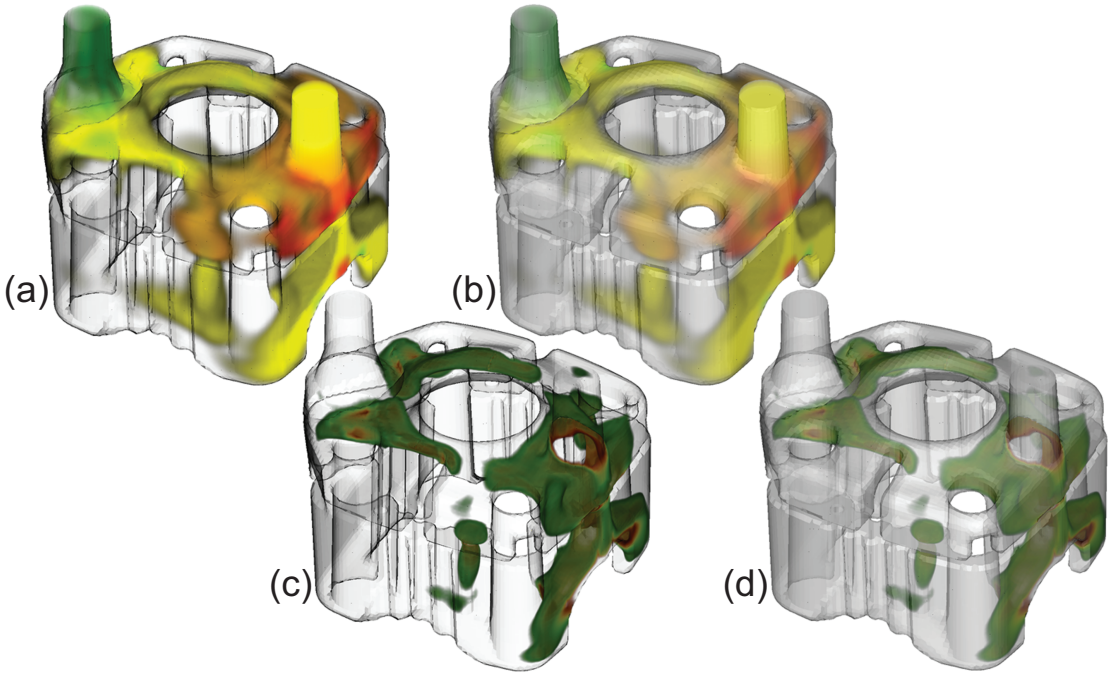


Figure 2.12: Two different boundary rendering techniques combined with two different optical volume models to four different rendering modes. Silhouette enhancement is used in (a) and (c) and standard surface rendering is shown in (b) and (d). Standard DVR is applied in (a) and (b) whereas (c) and (d) display smooth isosurfaces. High turbulent kinetic energy (TKE) is brushed and pressure mapped to color in the DVR images and used as parameter for the isosurface visualizations. This small cooling jacket data set is courtesy of the institute for Internal Combustion Engines and Thermodynamics, University of Technology Graz, Austria.

2.3.2 Rendering Modes

Our hybrid raycaster has to deal with two different data volumes simultaneously: the scalar field of the underlying data set (e.g., temperature), and the scalar DOI function specified by the user. In order to visualize this multi-volume, several rendering modes can be selected by specifying a boundary visualization technique, and an optical model for direct volume rendering. Figure 2.12 illustrates all four combinations of two different optical models and boundary visualization techniques. Table 2.2 summarizes the major parameters and the underlying data required. Color and opacity at position $\mathbf{x}$ are computed according to the following equations, using a local blending factor $\lambda(\mathbf{x})$ that depends on the optical model:

$$\mathrm{c}(\mathbf{x}) = \lambda(\mathbf{x})\mathrm{tf}\big(\mathrm{s}_{\mathrm{win}}(\mathbf{x})\big) + \big(1-\lambda(\mathbf{x})\big)c_l, \tag{2.14}$$

$$\alpha(\mathbf{x}) = \lambda(\mathbf{x})f_\alpha, \tag{2.15}$$

where c_l is the luminance of the context, f_α is the opacity of the focus, and $\mathrm{s}_{\mathrm{win}}(\mathbf{x})$ is the scalar $\mathrm{s}(\mathbf{x})$ mapped from $[s_{min}, s_{max}]$ to $[0,1]$ with a windowing function (Table 2.2).

The first optical model is based on emission/absorption without shading and distinguishes between context and focus according to the DOI function:

$$\lambda(\mathbf{x}) = \mathrm{DOI}(\mathbf{x}) \tag{2.16}$$

This choice of λ smoothly blends between the context luminance and the color transfer function, and derives the opacity linearly from the DOI function. Figures 2.12 (a) and (b) show this optical model applied to the *Small Cooling Jacket* data set in which regions of high turbulent kinetic energy have been selected smoothly. Color values represent the corresponding pressure within the volume. The context luminance c_l is set to zero to emphasize the structure of the selected data region.

The second optical model shown in Figures 2.12 (c) and (d) creates visualizations that resemble smooth iso-surfaces. In this case, both the DOI function and the windowed scalar value are used to determine the blending factor λ:

$$\lambda(\mathbf{x}) = \mathrm{DOI}(\mathbf{x})\mathrm{t}_{01}\big(\mathrm{s}_{\mathrm{win}}(\mathbf{x})\big) \tag{2.17}$$

where $\mathrm{t}_{01}(\cdot)$ is a tent function centered at 0.5 in the interval $[0, 1]$, which smoothly fades out λ around the window center. The DOI function is incorporated in order to ensure that bricks with $\mathrm{DOI}(\mathbf{x}) = 0$ for all $\mathbf{x}$ also use a constant opacity of zero. For such bricks only the boundary will be rendered. Thus, the smooth iso-surface created by this optical model can additionally be clipped smoothly by specifying an adequate DOI function. By using $\lambda(\mathbf{x})$ to modulate the resulting color the lack of proper shading of the smooth surface can be somewhat compensated. Rays only touching the surface will gather more dark samples (if c_l is set to zero) than rays intersecting it, which results in dark silhouettes. This "shading" approach is similar to limb darkening [57].

It is important to note that besides the optical model for volume rendering the boundary visualization is of very high importance to the overall visualization approach presented in this section. Data sets from the engineering field very often contain complex surfaces which can be of high interest to the user if they interact with the simulated phenomena. Therefore, we have decided to treat the mesh surface as context information which can be visualized using different

Rendering Mode Parameters		**Available Data**	
f_α	global opacity of focus	$\mathrm{DOI}(\mathbf{x})$	DOI function
c_α	global opacity of context	$\mathrm{s}(\mathbf{x})$	scalar field
c_l	global luminance of context	$\mathbf{v}$	view direction
b_c	boundary color	$\mathbf{n}_s$	surface normal
$\mathrm{tf}(s)$	user-defined transfer func.		
s_{min}	lower windowing bound		
s_{max}	upper windowing bound		
$\mathrm{s}_{\mathrm{win}}(\mathbf{x})$	$= \mathrm{clamp}_{01}\left(\frac{\mathrm{s}(\mathbf{x})-s_{min}}{s_{max}-s_{min}}\right)$		

Table 2.2: User-definable parameters, and the data available to the different boundary rendering and direct volume rendering models.

Data Set	Tet	Pyram.	Prism	Hexa	# Cells	Tet	Bricks
Generator	55.4%	0.6%	23.9%	20.1%	6,730 K	≈15.4 M	282(222)
Large Cooling Jacket	0.2%	5.2%	9.3%	85.3%	1,538 K	≈7.1 M	60(10)
Small Cooling Jacket	0.2%	1.3%	2.1%	98.4%	77 K	≈377 K	2(0)

Table 2.3: Various data sets used throughout this section. In addition to the cell type percentage (tetrahedra, pyramids, triangular prisms and hexahedra), the equivalent number of tetrahedral cells using a minimal subdivision is given for comparison. The overall number of bricks is shown along the number of resampled bricks (in brackets).

Data Set	Interact.	Static	Pres.	HAVS
Generator	375ms	2.6s	22s	-
Large Cooling Jacket	234ms	1.0s	2.4s	6.0s
Small Cooling Jacket	53ms	784ms	784ms	156ms

Table 2.4: The performance measures are based on the *simple* memory layout presented in Section 2.3.1.4. The three columns "Interact.", "Static.", and "Pres." show rendering times for different image-quality settings (see Section 2.3.3.2). For comparison purposes, performance numbers using the public version of HAVS [22] are also given for the tetrahedralized data sets (using the command line options "-none -p").

boundary visualization techniques. Figures 2.12 (a) and (c) show silhouette boundary visualizations, whereas Figures 2.12 (b) and (d) have been created using simple shaded semi-transparent surface rendering. Silhouette rendering is realized by assigning a color b_c to boundary samples and computing the corresponding opacity as $\alpha = (\mathbf{n}_s \cdot \mathbf{v})^p c_\alpha$. The exponent p has been set to 4 for all result images that utilize this boundary-visualization method. With this approach only a minimal amount of the data volume is obstructed by the boundary visualization, which, however, still provides proper contextual cues to relate the selected flow features to the overall geometry of the data set.

2.3.3 Application Results

In this section an example application of the hybrid raycasting algorithm in conjunction with the SimVis system is presented. Additionally the memory requirement, rendering performance and image quality of our approach is discussed.

2.3.3.1 Large Data

The *Generator* data set contains one sixth of the geometry of a generator (the remaining five sixth are symmetrical). The data volume itself consists of two parts: the solid portion of the generator and the air surrounding it. Both parts are separate meshes which touch each other on a common surface. The problem that has been modeled within the data set, is the cooling of the generator. Thus heat transport and dissipation has been simulated for the solid portion of the data set, while air temperature and flow has been computed for the surrounding air.

The grid itself poses multiple challenges to the visualization system. First of all the mesh surface has a very high depth complexity, which requires many depth layers to be rendered for the final visualization. Even when decomposed into 282 bricks every brick contains between two to eight depth layers (depending on the view port). On average five depth layers have to be rendered. Since nearly all state-of-the-art methods can only handle tetrahedral volume cells, the *Generator* data set would have to be tetrahedralized to be visualized with them, which would result in a cellcount of over 15 million as shown in Table 2.3.

Figure 2.7 shows a result image from a complete visualization session using the SimVis system and the hybrid raycasting algorithm. It comprises four different selections which are combined by using a fuzzy OR operation, and maps temperature to color. The first selection specifies the solid portions of the data set which are used to provide additional context to the user. Additionally cold and warm portions of air have been selected. It can be observed that cool air which is sucked through the cooling ducts at the top of the data set is heated up rather quickly and stagnates in the space above the winding heads. In order to visualize the vortex, which is located behind the air outlet the fourth selection marks low pressure as important. As concluded by Trenker et al. [127], it is desirable to deflect this vortex to pass through the winding heads at the air outlet to avoid the stagnation of hot air and thus provide better cooling for the overall machine. This can be accomplished by moving the air outlet. Trenker et al. relied heavily on a prototype implementation of the hybrid raycasting method in order to come to this conclusion.

2.3.3.2 Quality and Performance Considerations

Several parameters can be used to steer the memory consumption, the speed, and the image quality of the presented visualization approach. In order to guarantee responsiveness during an analysis and exploration session two different sets of parameter settings can be chosen: one is applied during interaction, and one is used for generating static images. Additionally, it is possible to progressively update the display during the generation of a static image. Every n'th brick the accumulated color and opacity information is copied to the front buffer. In Table 2.4 some performance measures are presented for the different data sets shown in this section. Different quality settings have been used to measure rendering performance during interaction, for a static image and for the data set without using resampling. Here, an unstructured brick always forces the memory manager to allocate texture memory, even if data from another brick has to be overwritten. During interaction only one depth peel for every brick is rendered, and the unstructured brick sampling settings are configured to perform only barycentric interpolation at the cell boundaries. Additionally the result image resolution is reduced to 256^2 (instead of 512^2 for the static measurements). In the static case the sampling settings for the unstructured bricks are set to $_l = c_{max}/4$ and $_c = c_{max}/16$ with c_{max} being the maximum diameter of a cell's bounding sphere. Thus a maximum of four mean value samples are computed per cell per ray. These settings are equal to those used to create the "Presentation" column of Table 2.4 resulting in the same measurements for data sets which completely fit into the texture memory allocated for unstructured data.

For comparison with a fast state-of-the-art tetrahedral volume renderer, the last column of Table 2.4 shows the performance of the publicly available implementation of the HAVS algorithm [22] for our data sets converted to tetrahedral grids. It can be observed that HAVS is

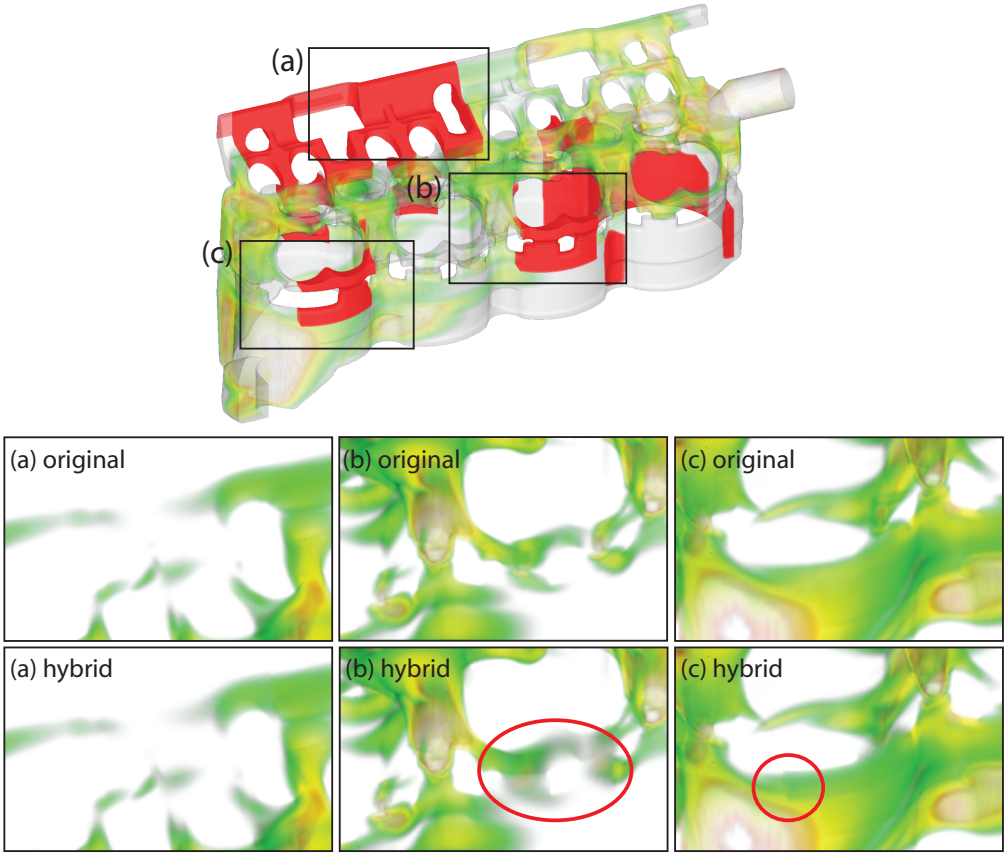


Figure 2.13: Overview and magnifications of the *Large Cooling Jacket* data set. The original unstructured mesh has been used when rendering the images labeled "original". The row of images labeled "hybrid" has been created by using a combination of unstructured and structured bricks. Red color indicates the resampled bricks in the overview rendering. The data set is courtesy of AVL List GmbH, Graz, Austria.

more than five times faster than our approach when generating high-quality images for the *Small Cooling Jacket*. However, the time required by visibility sorting increases at least linearly with the number of cells (2.5s for the *Large Cooling Jacket*), which is probably the main factor why our raycasting method outperforms HAVS for this data set even under "Presentation" quality settings that cause streaming of texture data onto the GPU during rendering. For loading the *Generator* data set, the HAVS implementation that we used exceeded the memory limits on our machine. However, this implementation is probably not optimized for data sets of this size, and the client/server version [20] would have to be used for comparison purposes.

A more in-depth analysis of memory consumption and performance characteristics of our approach is presented in Table 2.5. Here the *Large Cooling Jacket* and the *Generator* are used to compare the *simple* and *packed* memory layouts. The table clearly shows that the effectiveness of the two layouts highly depends on the distribution of different cell types within the data set.

The percentage of tetrahedral cells is far higher in the *Generator* which leads to a less efficient representation if compared to the *Large Cooling Jacket*. This can be observed when comparing the memory requirements per tetrahedron. As comparison the tetrahedral strip encoding proposed by Weiler et al. [141] uses 76 byte/tet which can be reduced to 15 byte/tet if additional rendering speed is sacrificed for a less redundant data representation. Furthermore, the *packed* layout is only more efficient than the *simple* layout if the mesh contains a large variety of cell types, e.g., in the case of the *Generator* data set. If most cells are of the same type, such as in the *Large Cooling Jacket*, the simple layout uses less memory (Table 2.5). This is also reflected in the percentage of resampled cells, since a higher memory consumption requires resampling more cells. Additionally, the increased cost for fetching face data in the packed layout is reflected in the performance measures (using the same settings as Table 2.4).

Besides the rendering performance itself, the brick boundaries are of high importance for image quality (especially between structured and unstructured bricks). Figure 2.13 shows some comparisons between a high-quality visualization using only unstructured bricks and the default still image settings mentioned above. In order to reduce obstructions the boundary visualization is set to completely transparent. The overview on the left-hand side indicates resampled bricks in red. In Figure 2.13 (a) barely any difference between resampled and original mesh can be seen, whereas Figure 2.13 (b) shows the low-pass filtering effect of the resampling process. Additionally Figure 2.13 (c) shows a slight discontinuity between the resampled and original mesh. The previously mentioned effects mostly occur if high frequency transfer functions and DOI specifications are used. Since the SimVis framework emphasizes the smoothness of flow features (through smooth brushing) and the resampling process can be steered by interactively modifying the DOI specification, the application of resampling to unimportant regions is feasible. All tests have been carried out on an AthlonX2 4400+ with 4GB of RAM, and a Geforce 8800GTX with 768MB video memory.

2.3.4 Summary and Discussion

We have presented a scalable hybrid GPU raycasting algorithm for unstructured grids. Our method directly renders complex cell types without tetrahedralization. Non-tetrahedral cells

	Generator		Large Cooling J.	
Data Layout	Packed	Simple	Packed	Simple
Without Resampling				
Mem.	1,094 MB	1,330 MB	332 MB	318 MB
Per “Tet”	71 Byte	86 Byte	46 Byte	44 Byte
With Resampling				
Mem.	293 MB	310 MB	254 MB	266 MB
# Resampled	76%	82%	23%	18%
Interact.	590ms	375ms	390ms	234ms

Table 2.5: Comparison of the two different layouting schemes (*packed* and *simple*) for the face texture sets described in Section 2.3.1.4.

employ mean value interpolation for selected samples, and linear interpolation in-between. We use bricking, resampling, and custom texture memory management in order to sustain interactive performance and make optimal use of the available amount of texture memory. It is a hybrid approach in the sense that it combines unstructured and structured grid raycasting, as well as image order methods (raycasting) and object order approaches (bricking). Different volume rendering styles are combined with surface rendering methods to create highly parameterizable visualizations that are based on two concurrent data volumes: a degree of interest (DOI) function specified in the SimVis system, and the scalar data volume. We also pay special attention to accurately rendering the surface mesh of the original grid at all times. Apart from better interpolation quality, avoiding tetrahedralization also balances the additional amount of work required by mean value interpolation as well as the additional memory for storing a variety of cell types. Converted to tetrahedral meshes, our data sets would contain twice to four times as many cells. The effectiveness and scalability of our approach has been demonstrated by applying it to real-world data sets of different sizes containing large and highly complex meshes.

While the proposed technique has been the first GPU-based raycaster capable of directly handling non-tetrahedral cells, for reasons of memory addressing and alignment, the overall data layout targets grids with only a small number of different cell types. Cell faces are also limited to triangles or planar quadrangles, and only convex cells are supported. While memory consumption of the proposed technique is low in comparison to other GPU-based raycasting methods it is still significantly higher than for object order methods that do not require mesh connectivity data. The data structure and the corresponding raycasting algorithm discussed in Section 2.4 directly adresses all these limitations.

2.4 Interactive Volume Visualization of General Polyhedral Grids

This section is based on the paper *Interactive Volume Visualization of General Polyhedral Grids* [101]. Over the years, the complexity of the grids produced by state-of-the-art meshers and simulation packages, such as OpenFOAM [2] or STAR-CCM+ [3], has increased tremendously. The volume meshes used nowadays in complex multi-physics simulations, for example, consist not only of an arbitrary combination of fixed cell types, such as tetrahedra, hexahedra, and octahedra, but contain a significant number of essentially arbitrary polyhedral cells. These cells can have an arbitrary number of faces, each of which can consist of an arbitrary number of vertices. Cells can be non-convex and even degenerate, while their faces can be non-convex and non-planar. Moreover, different regions of a mesh are often generated using different meshing strategies. Therefore state-of-the-art visualization systems have to be highly scalable with respect to cell/mesh complexity. That is, volume rendering methods have to cope with meshes that potentially contain different regions that might comprise cells of hugely different complexity. For example one region might contain only tetrahedral cells whereas another one contains arbitrary polyhedra. However, most existing unstructured-grid visualization-methods are constrained to tetrahedral meshes for performance reasons and simplicity of implementation. Thus, these approaches have to tetrahedralize more complex grids before visualization. However, for meshes with complex cells, the increased number of cells resulting from tetrahedralization is a significant burden for visualization performance and memory usage. Moreover, a given tetrahe-

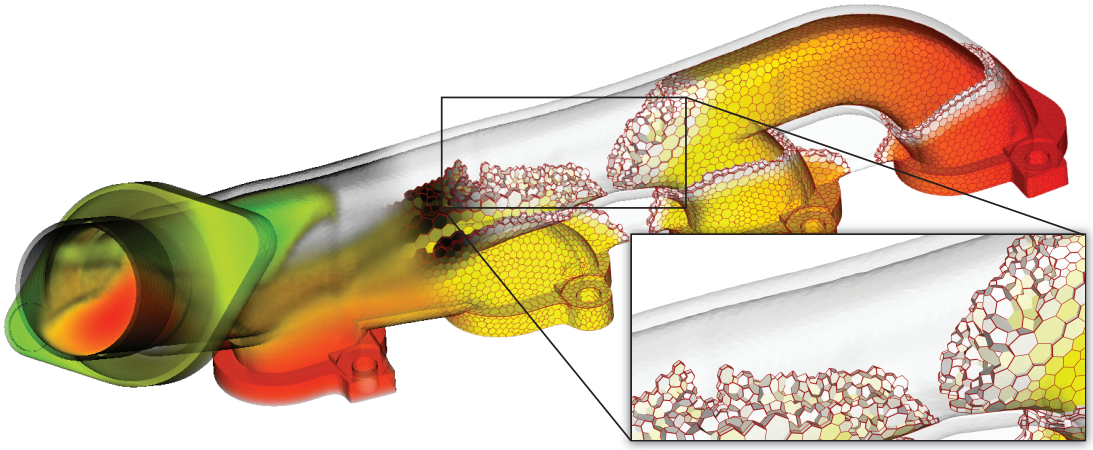


Figure 2.14: Interactive raycasting of the temperature distribution in an exhaust manifold that was simulated using a state-of-the-art CFD solver on a complex grid composed of general polyhedral cells. Red color indicates warm, green cool regions. The complex structure of the underlying mesh is illustrated through cell faces. The cells in this mesh are not only tetrahedra or other predefined cell types, but are also very general, often non-convex, polyhedra with non-planar faces, which our approach can nevertheless visualize directly. The data set has been provided by CD-Adapco.

dralization is not unique and introduces linearization artifacts when interpolation is used, since the original cells are split up into separate, piecewise linear constituents. This section introduces the first interactive visualization approach for extremely complex unstructured grids, which takes into account all of the properties of state-of-the-art volume meshes outlined above.

A major practical problem of complex meshes composed of general polyhedral cells is that their representation requires very flexible data structures. In order to be able to traverse these data structures for interactive visualization, e.g., during raycasting, a lot of additional information is usually stored. Commonly, efficient traversal of mesh topology is facilitated by using both per-cell information, e.g., a list of faces a cell is composed of, as well as per-face information, e.g., pointers to the cells a given face connects. All this information consumes a significant amount of memory and book-keeping overhead for meshes with general polyhedral cells. In principle, either one of these two types of information is redundant. Nevertheless, it is usually incorporated for performance reasons. In contrast, as a basis for a variety of interactive visualization algorithms, we propose a very compact representation for such grids, which is purely face-based while still allowing for efficient traversal. Building on this data structure, we have developed a very flexible, interactive GPU raycasting method. We note that, for similar reasons, state-of-the-art volume meshers have also switched to purely face-based grid representations [2, 3]. In this regard, our contribution is the development of an augmented face-based data structure that is almost as compact as these original mesh representations, but whose goal is efficient visualization instead of raw data storage. In summary, the main contributions presented in this section are:

- A very compact, purely face-based data structure for complex unstructured grids composed of general polyhedral cells. This *TSFSL representation* specifically targets direct visualization algorithms, such as raycasting, not only raw data storage.
- A very efficient GPU raycasting approach that operates directly on the proposed TSFSL representation, with full support for domain decomposition (in this work, using a kD-tree and bricking).

2.4.1 Polyhedral Grid Representation

In order to develop an efficient representation and corresponding data structure for arbitrarily complex polyhedral grids, we first consider the basic requirements that such a representation has to support. The most important operation is efficient traversal of the grid topology, for example marching from cell to cell along a given ray in raycasting. For this, one has to be able to determine *which cells* are intersected by a ray in *what order*. To achieve this efficiently, all state-of-the-art unstructured grid raycasters rely on an adjacency graph of *cell-face-cell connectivity* (see Figure 2.15 (a)). One must also be able to obtain sample values at arbitrary positions within each cell via *interpolation*. In tetrahedral meshes, this can naturally be done via barycentric interpolation. For more complex polyhedra, mean value interpolation [43,67] can be used. This, however, requires data from all faces of the enclosing cell for every sample. Finally, determining the *entry face* through which a ray enters the mesh must be efficient. Current unstructured-grid raycasting-methods usually achieve this by using GPU rasterization of the mesh boundary, storing a face ID in each pixel.

Thus, our representation must support the following operations:

1. *Query all faces of an individual cell:* Provide efficient means for enumerating all faces of a given cell.
2. *Query neighboring cell across a face:* Given a cell and one of its faces, enable fast access to the neighboring cell across this face.

These fundamental operations are not only sufficient for raycasting, but for all algorithms that require traversing the grid from cell to cell in some order. Examples include the computation of stream, path, and streak lines or surfaces, vortex-core extraction-techniques, and many other visualization algorithms. With the exception of tetrahedral strips [141] and CIEL, [89] all data structures used by previous raycasting approaches employ an explicit cell representation. This is simple and efficient for tetrahedral grids, where each cell has four vertices and four faces. When the number of faces per cell is not constant, however, a more complex representation must be used in order to allow each cell to contain an arbitrary number of links to faces comprising its boundary.

2.4.1.1 Standard Unstructured-Grid Representation

Figure 2.15(a) illustrates the most common data structure for representing unstructured grid topology. This example depicts three cells composed of six faces in total, where each face knows

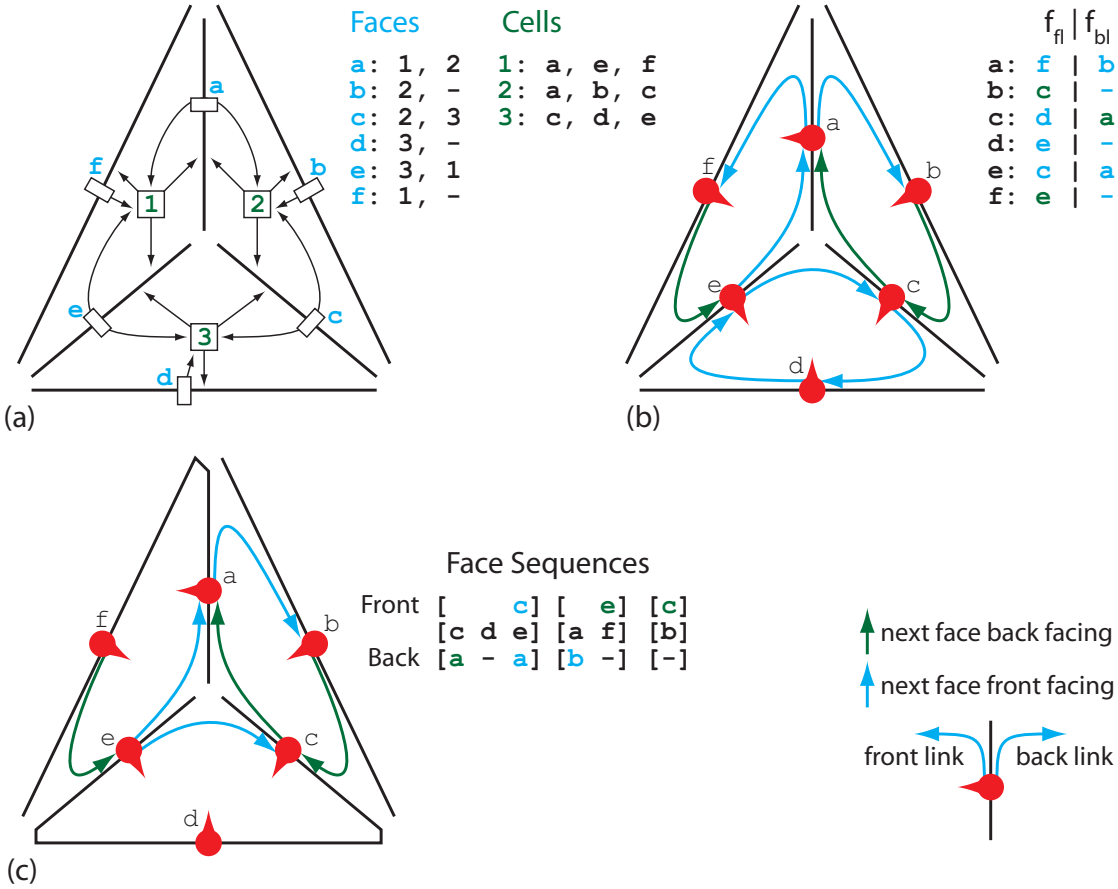


Figure 2.15: Comparison of the most common representation of unstructured grids (a), with our basic TSFL (b) and extended TSFSL (c) data structures. In (a), for every cell a list of faces is stored, and every face stores which two cells it connects. In contrast, our data structure (b) stores only faces and two links per face, completely avoiding explicit per-cell storage. These two links reference the next face of the cell in front of the face (f_{fl}), and behind the face (f_{bl}), respectively. The face normals are indicated by red arrows in the figure. Additionally, each link requires a boolean flag (f_{ff}, and f_{bf}, respectively) to indicate whether the cell is in front or behind of the next face that it is linked to. Here, blue and green colors indicate the front and back half spaces, respectively. Figure 2.19 gives pseudo code for efficient traversal of the TSFSL data structure, shown in (c), during raycasting.

which two cells it connects (*face-to-cell connectivity*, column "Faces"), and each cell knows all of its comprising faces (*cell-to-face connectivity*, column "Cells"). With this information, querying all faces of an individual cell, as well as traversing from one cell through one of its faces to its adjacent cell, is trivial. Note, however, that only one of the two types of connectivity is really required to represent the entire mesh and to allow reconstruction of the other type. In principle, it is trivial to reconstruct face-to-cell information from cell-to-face information, and

vice versa. The advantage of storing redundant connectivity is that it allows for fast *cell-face-cell traversal*, which is crucial for many visualization algorithms. Thus, the trade-off is sacrificing memory for higher traversal performance. Naturally, for complex cells the memory footprint of redundant topology information rises rapidly.

2.4.1.2 TSFL/TSFSL Unstructured-Grid Representation

In contrast to the standard representation outlined above, the flexible data structure we introduce in this section does not store any redundant connectivity information at all. However, it still allows for very efficient grid traversal and supports the two required fundamental operations described above. This is achieved by representing all connectivity through lists of faces comprising the mesh, without any explicit cell representation. Furthermore, we have separated the actual face-geometry data, such as how many and which vertices make up a face, from the representation of mesh topology. This enables a very compact representation of polyhedral meshes generated by different meshing approaches that employ a wide variety of cell characteristics.

Two-sided face lists (TSFL): We only store mesh faces, and exploit the fact that each face has at most two adjacent cells. Therefore, if we think of linking all faces of a given cell together, then any given face can be a part of at most two separate face lists. That is, a face is part of two face lists if it connects two cells, or is part of only a single face list if it is a boundary face. Therefore, it is sufficient to reserve only two link fields per face, including some additional information per link as described below. Because of the two sides of each face and their respective lists, we call the resulting data structure *Two-Sided Face Lists (TSFL)*.

Face links: Consider Figure 2.15(b), which depicts six faces, each of which has two links to other faces. The *front link* f_{fl} links to another face of the cell located in its front half space; the *back link* f_{bl} links to another face of the cell in its back half space. Each cell is then only represented implicitly as one cycle in the directed graph whose vertices are the cell faces, and whose edges are these front and back links, respectively. This representation has obvious similarities to the winged edge data structure for surface meshes [9]. However, instead of linking edges in adjacency order, we link the faces comprising a cell in *arbitrary order*, storing only minimal linking information that is sufficient to support the two queries required above. This arbitrary linking differentiates TSFL from other half-edge/face data structures, where each link represents a specific geometric relation.

Face link flags: In order to enumerate all faces of a cell, the front and back links alone are not sufficient. For example, consider enumerating the faces of the cell in front of face f in Figure 2.15(b). At every step of following a face link, one has to know whether to follow the front link or the back link, respectively. It is not possible to orient face normals consistently with respect to every cell. This problem can be solved by augmenting every link field with a boolean flag that indicates whether the cell whose faces are being enumerated is in the front or back half space, respectively, of the face that the link refers to.

Cell-face-cell traversal: The TSFL representation, consisting of the per-face front link (f_{fl}) with its boolean flag (f_{ff}), and the per-face back link (f_{bl}) with its boolean flag (f_{bf}), is sufficient for complete enumeration of all faces of any cell in the entire mesh, and for traversing from any cell across any of its faces to the respective adjacent cell. That is, cell-face-cell traversal is

performed by intentionally following the link corresponding to the opposite half space of a face, thus stepping from the cell on one side of the face to the cell on its other side. In the remainder of this section, we will further use the notation that the two values which the two boolean flags (f_{ff}, f_{bf}) can assume are $+$ and $-$, to indicate the front and back half space of a face, respectively.

Two-sided face sequence lists (TSFSL): We now introduce an important refinement of the TSFL representation, which results in the construction of the *Two-Sided Face Sequence Lists* (TSFSL) data structure that we use in the remainder of this work. A TSFSL is constructed from a TSFL by storing selected sequences of linked faces in adjacent memory locations without storing actual links. This significantly optimizes access and traversal speed, and reduces the overall memory footprint. One of the main advantages of storing a mesh via both cell and face arrays, as in Figure 2.15(a), is that references can be stored sequentially in memory and thus allow for fast access. For example, enumerating all faces of cell 3 in the figure simply scans the array $[c, d, e]$. The same enumeration in the TSFL data structure involves more effort, because for each face additional linking information must be read from memory. This introduces a large number of random memory accesses that reduce performance on architectures which prefer coherent memory accesses, such as GPUs. It also inherently incurs memory access latency, because the next face can only be visited after the corresponding link has been fetched from memory.

The TSFSL data structure overcomes these problems to a large extent. Considering the mesh shown in Figure 2.15(b), we can observe sequences of faces that are connected by arrows of the same color. Note especially the cell with faces (c, d, e). Here, within the cell only front links have to be followed since all faces are facing toward the cell. The occurrence of such *face sequences* can be exploited by storing their faces sequentially in an array without links. If this sequence is stored consecutively in memory as $[c, d, e]$, we can drop the *front* links and flags entirely. This does not result in any loss of information, since the storage location itself now implicitly encodes these links. If face sequences are constructed using front links, which we have chosen to do, this immediately implies that the *back* links and flags have to be retained, because they are then required to connect to adjacent cells. This also implies that all face sequences can be constructed by collecting all the faces of each cell that are facing toward it.

We guarantee *at most one* face sequence per cell, by putting all faces that are facing toward it into the same sequence. This is possible because the order in which the faces of a cell are linked is arbitrary. Guaranteeing *at least one* face sequence per cell is also possible, because simply flipping a face's normal allows removing it from one sequence and adding it to the sequence of the cell on the opposite side. Together, this allows to guarantee *exactly one* face sequence per cell. We use the following two-phase *face sequence generation algorithm*: The first phase starts with an initially empty face sequence s_i for each cell c_i. Then, the number of faces n_i assigned to each s_i is tracked, while assigning each face to the sequence with the lower n_i (outside the mesh, $n_i = \infty$). For all meshes that we tested, creating sequences this way was already sufficient. However, in general this may produce empty sequences with $n_i = 0$. In order to guarantee $n_i > 0$ everywhere, we process each empty s_i in a second phase. For each such s_i, we pick a neighboring cell c_j based on two conditions: c_i has to be connected to c_j through a face that is not flagged as *used*, and n_j has to be the maximum among the neighbors

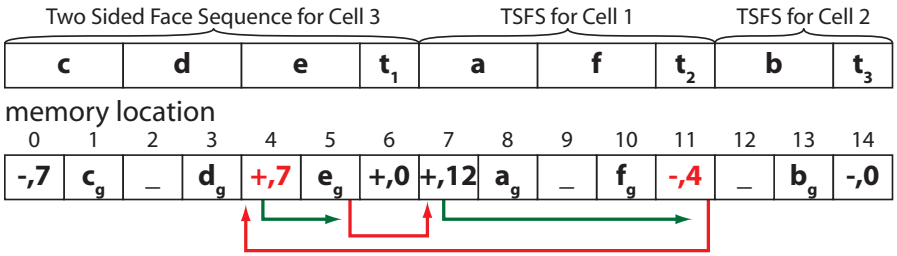


Figure 2.16: All face sequence lists of the mesh from Figure 2.15 are stored in a single linear 1D array. The colored arrows illustrate the traversal of the faces of cell 1. Green arrows indicate sequential memory reads, whereas red arrows denote jumps to different memory locations. The $+$ and $-$ signs indicate flag values of front- and back-facing, respectively.

that fulfill the first condition. Because $n_i = 0$, the face connecting c_i to c_j is assigned to s_j. By re-assigning this connecting face to s_i, n_i is increased from zero to one, and n_j is decreased by one. The face is then flagged as *used*. This face re-assignment is repeated as long as $n_j = 0$. If $n_j \neq 0$, all flags that identify faces as *used* are cleared. After each iteration of this algorithm, the number of empty face sequences is reduced by one.

See Figure 2.15(c) for an illustration of how face sequences can be defined in this example. Faces within a sequence are shown connected without arrows. Note that the last link e_{fl} and flag e_{ff} have to be retained, in order to connect the end of the face sequence with subsequent faces of the cell. In the example of sequence $[c, d, e]$, all faces of the cell are in one face sequence. This is not true in general, as can be seen in the face sequence $[a, f]$. Also shown in Figure 2.15(c) is the face b, which is considered as a face sequence of length one, since it cannot be connected to a longer face sequence.

In the following, we summarize four important properties of the TSFSL representation that result from its construction:

1. The face list corresponding to any given cell contains *exactly one* face sequence, which contains all the faces that are facing *toward* the cell. This sequence can, however, be of length one.

2. The *front* links/flags for faces in the sequence are *not stored*, with the exception of the last face in the sequence.

3. The *back* links/flags must be stored for *all* faces in the sequence.

4. All faces facing *away* from a cell are not part of this cell's face sequence, but of the sequence of the cell they are facing toward.

A crucial consequence of this construction is that *all* face sequence lists of an entire mesh can be stored back-to-back in a single 1D array.

Memory savings of the TSFSL representation: In order to quantify the memory savings resulting from face sequences, we count the number of links that can be discarded, in comparison with the TSFL structure. For a sequence containing n faces, $n - 1$ links can be removed. For a

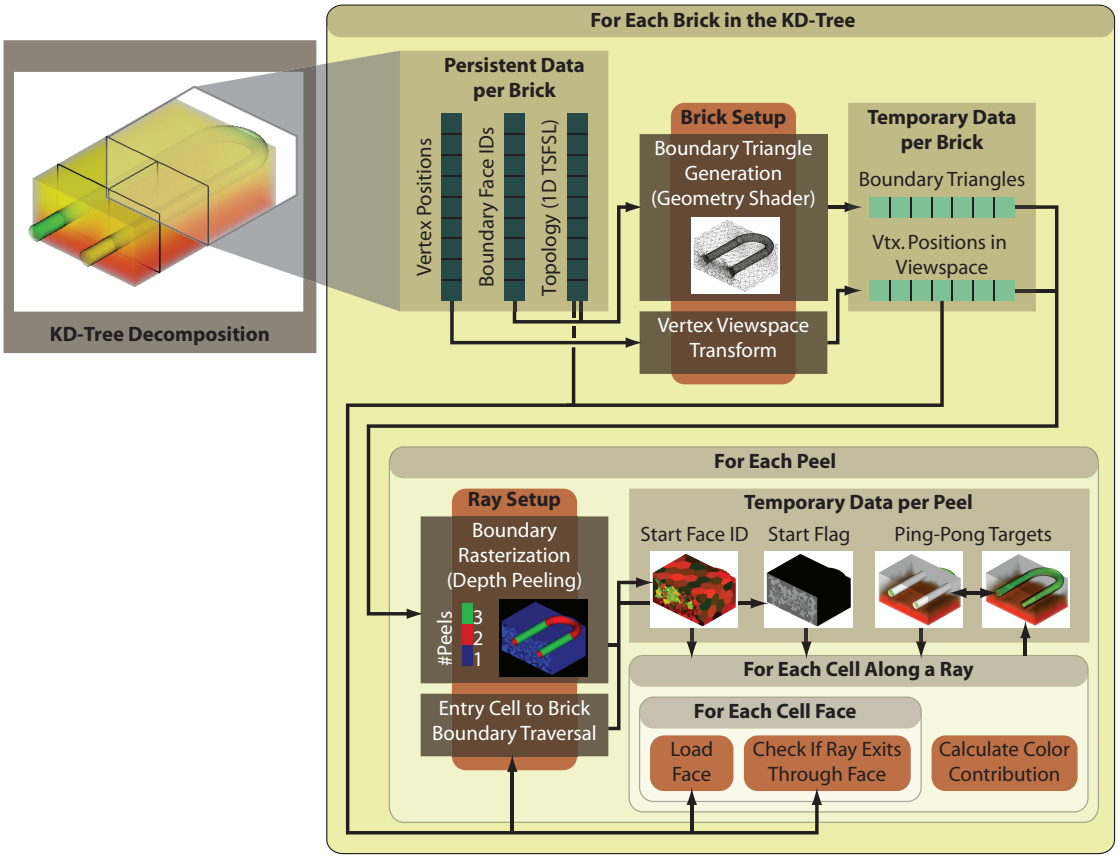


Figure 2.17: The pipeline for raycasting unstructured meshes given in the TSFSL representation. A large mesh is subdivided into bricks that correspond to the leaves of a kD-tree. Rendering proceeds from brick to brick in front to back order. Each brick contains the corresponding 1D TSFSL array, as well as vertex positions and IDs of the faces comprising the mesh boundary. In order to obtain ray-start positions, the mesh boundary is rasterized. Non-convex cells are treated implicitly in each rendering pass, whereas non-convex areas of the mesh are handled using depth peeling.

mesh containing c cells, with f internal and b mesh boundary faces, the average face sequence length is $(f + b)/c$, because each face belongs to exactly one face sequence, which in turn corresponds to exactly one cell. Thus, on average $(f + b)/c - 1$ links can be omitted per cell, which amounts to omitting $f + b - c$ links in total over the entire mesh. Note that with an increasing average number of faces per cell, $(b + 2f)/c$, the number of links that can be discarded increases. Overall, the number of links stored in the TSFL representation is equal to $b + 2f$, whereas the TSFSL representation thus contains only $f + c$ links. For comparison, the standard approach depicted in Figure 2.15(a) must store $2(f + b)$ links from faces to cells, and $b+2f$ links from cells to faces. It is interesting to note that the number of links stored in our data structure depends on the number of cells, whereas this is not the case in the standard approach.

In order to directly compare both numbers, we can use the property that the number of faces per cell is at least 4 (i.e., cells are at least tetrahedra), and thus: $(b + 2f)/c \geq 4$. From this follows that $(b + 2f)/4 \geq c$. An upper bound for TSFSL can be defined as $f + c \leq 3f/2 + b/4$. For the "worst" case of a tetrahedral mesh this is still smaller than either the number of face-to-cell or cell-to-face links required in the standard representation. Relative memory savings increase as the average number of faces per cell goes up. Actual memory consumption numbers and comparisons for real-world meshes are reported in Section 2.4.3.3.

2.4.1.3 Face Geometry and TSFSL Storage

In addition to storing mesh topology in the TSFSL data structure, the geometry of each face must also be stored. Conceptually, we completely separate mesh topology from face geometry. This enables adapting the geometry representation to a variety of characteristics of real-world meshes and the corresponding meshing algorithms and tools, which can also vary between different mesh regions or zones. Despite the important conceptual separation, we can interleave the actual storage of face geometry in memory (denoted below abstractly as f_g for face f) with the TSFSL records of the corresponding faces. This improves memory-access coherency. The TSFSL representation stores $n + 1$ face links/flags for a face sequence of n faces. Interleaving this storage with actual face geometry, a face sequence of faces $[a, b, c]$ is stored in memory as $[(a_{bf}, a_{bl}), a_g, (b_{bf}, b_{bl}), b_g, (c_{bf}, c_{bl}), c_g, (c_{ff}, c_{fl})]$. Note that the front link/flag of the last face in the sequence is stored after the corresponding face geometry. We will call it the *face sequence terminator* (denoted as t_i in Figure 2.16). Given this memory layout, the face sequence can be traversed by alternately reading a link/flag tuple and face geometry, until the sequence terminator is read. As above, one of the most important observations to make here is that *all* faces of a mesh are stored as part of a face sequence. This is possible because every face of the mesh is part of exactly one face sequence. This sequence is the one corresponding to the cell in the front half space of the face. The faces' back links, also stored in the sequence, reference faces at non-sequential storage locations, which in turn are part of other face sequences. Therefore, the entire mesh can be stored in a single 1D array, as illustrated in Figure 2.16.

Face geometry records: The actual geometry storage format of f_g can be arbitrary, with the restriction that its size can either be inferred implicitly (e.g., if all faces use an equal amount of memory), or can be derived from a small header at the start of f_g. Our current implementation supports two fixed-size face geometry representations, which can store three or four vertex indices. A variable-sized version stores the number of vertices of each face in addition to its vertex IDs. The former are used for meshes (or bricks, see below) that only contain triangle and quad faces. The latter is used for more complex polyhedral meshes, for example the mesh shown in Figure 2.14. A more detailed description is given in the context of ray-face intersection in Section 2.4.2.3.

2.4.2 Raycasting of TSFSL Meshes

For efficient raycasting of large TSFSL meshes, the grid is subdivided into bricks that are the leaves of a kD-tree (Section 2.4.2.1). Bricks are rendered individually, and composited in front

to back order (Sections 2.4.2.2 and 2.4.2.3). The mesh in each brick is represented by a single 1D TSFSL array. Figure 2.17 gives an overview of the resulting raycasting pipeline.

2.4.2.1 Bricking

We employ a domain decomposition of the entire mesh into bricks. Reasons are: scalability, memory management, adaptivity with respect to characteristics of the contained cells, coarse-grained culling, and the potential for direct parallelization across multiple GPUs. The brick geometry corresponds to the leaves of a kD tree, which is built using criteria similar to the work presented in Section 2.3. However, we avoid explicitly clipping cells against brick boundaries. Instead, each brick is assigned a *submesh* that comprises all cells that are entirely or *partially* inside the brick boundary. Raycasting is performed brick by brick in front-to-back visibility order, determined using the kD tree. In the following, the raycasting procedure is described considering a single brick.

2.4.2.2 Ray-Setup Stage

The first step of raycasting is to determine the positions where rays enter each brick. The most important part of this step is determining the ID of the face through which a given ray enters the first cell. This can be done very efficiently by encoding IDs in colors and rasterizing the grid boundary accordingly. In previous GPU raycasting approaches [41, 139, 141], the triangle mesh representing the volume boundary had to be stored explicitly in a format suitable for fast rasterization (e.g., in OpenGL vertex buffer objects). We avoid permanently storing this redundant representation for the whole mesh. Instead, each brick's submesh boundary is extracted on-the-fly from the TSFSL data structure before a brick is rendered, and immediately discarded afterward. The size of the buffer for storing this temporary boundary geometry needs only be large enough to accommodate the largest brick of the mesh. The submesh boundary geometry is

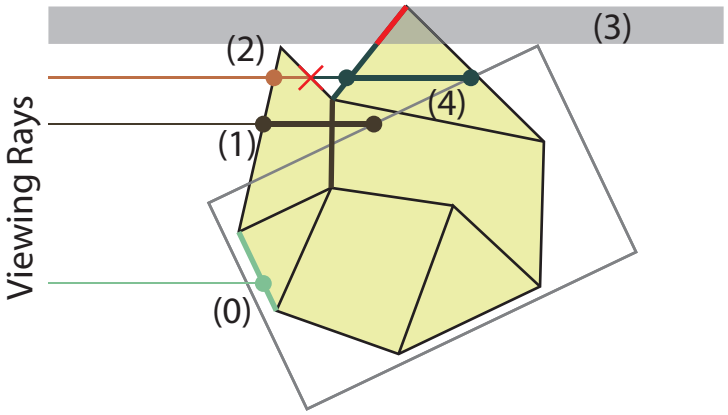


Figure 2.18: Instead of explicitly clipping cells against brick boundaries (grey rectangle), the ray-setup stage of each brick determines ray-start positions by traversing rays from boundary-cell faces to the brick boundary.

generated on-the-fly using a GPU geometry shader. The faces of the triangulated boundary are computed from an array of boundary face IDs stored with the brick data (see Figure 2.17). Details on using a geometry shader for computing this triangulation are provided in Section 2.4.3.1.

Special care has to be taken when bricking is used. The spatial subdivision requires that the submesh in each brick must only contribute to the volume rendering integral inside the brick boundary for correct compositing. Previous options for doing this include either explicitly clipping cells against the brick boundary, or rasterizing one quad per cell that is large enough to cover the projection of the cell/clipping plane intersection in screen space. The former approach requires storing explicit geometry for the resulting cuts, which is very costly for general polyhedral cells and usually involves triangulating the cut. The latter approach, which is used by the technique presented in Section 2.3, requires discarding fragments outside of the cell using face geometry information. However, for a complex cell and face geometry, the inside/outside test for discarding fragments becomes very expensive. Given these drawbacks of previous approaches for complex polyhedral cells, we use a different approach. Before the actual rendering passes for each brick, we perform a setup-rasterization step that traverses rays in order to find the entry positions of rays into cells inside the brick. This setup step is illustrated in Figure 2.18. Instead of rasterizing the volume boundary and the cell/brick-boundary intersections separately, we rasterize the unmodified volume boundary of the submesh that intersects a given brick, without clipping it. This rasterizes faces completely within the brick boundary, faces intersected by the brick boundary, and faces completely outside the brick boundary. During this rasterization, a setup ray is cast for each fragment. If the start position is already in the brick (ray (0) in Figure 2.18), the fragment immediately writes out the ID of the face that is being rasterized. For all other rays, the ray start position is outside the brick boundary, where we distinguish the following cases:

1. Rays may enter the brick without first leaving the corresponding submesh (ray (1) in Figure 2.18). At that point, these rays terminate and write out the face ID of the last face they intersected.

2. Rays may leave the submesh before they enter the brick (ray (2) in Figure 2.18). These rays terminate without writing a face ID.

3. Rays that do not intersect the brick boundary at all (in the area marked with (3) in Figure 2.18). These rays terminate immediately.

Note that since the entire submesh boundary is rasterized, this procedure also determines correct entry positions in non-convex areas, with respect to both non-convex mesh and cell boundaries. For example, ray (4) in Figure 2.18 determines and writes out the correct intersection and start position at the brick boundary, whereas ray (2) is correctly discarded, as described above.

The output of the ray-setup stage is a *ray-setup image* that for every pixel stores a face ID and flag that identify the cell where the corresponding ray must start for volume raycasting (Section 2.4.2.3). The face ID is encoded as a color, and the alpha channel is used for the flag. We deal with rays exiting and re-entering non-convex meshes by utilizing an approach that is similar

to depth peeling [141]. The grid is processed in depth layers corresponding to successive front-facing mesh boundary faces. For each depth layer, a ray-setup image is generated. However, the full ray-setup stage is only required for the first depth layer, since for all subsequent layers the ray start positions are already guaranteed to lie inside the brick. For these layers, simply clipping the submesh boundary to the brick's bounding box is sufficient.

2.4.2.3 Raycasting

After the ray-setup stage described above, the main volume raycasting pass is executed. In this pass, a ray is generated for every pixel where the flag in the ray-setup image is $\neq 0$, i.e., where the ray has been determined to intersect the mesh, and the pixel's face ID is valid. Volume raycasting proceeds in a loop from sample to sample along each ray, repeatedly carrying out the following three main tasks:

1. *Mesh traversal*, for traversing the mesh topology along a ray cell by cell, using the TSFSL data structure.
2. *Ray-face intersection*, for determining the next face intersected by the ray, also for non-convex faces and cells.
3. *Sampling and interpolation*, for determining sample values via interpolation within the current cell.

Mesh traversal: The algorithm is illustrated in pseudo code in Figure 2.19. The outer loop (lines 3-27) traverses the mesh along a ray by stepping from cell to cell through shared faces. The inner loop (lines 6-23) iterates over all faces of the current cell using the TSFSL data structure, and computes the next intersection of a ray with a cell face. Because we do not store explicit cell information, both of these tasks are performed by following face links: Either by stepping from one side of a face to the other side by following the opposite-side front/back link, or by iterating over all faces of a cell, in turn invoking the ray-face intersection for every encountered face. Mesh traversal is started at the face specified in the color channel of the ray-setup image: $startFace$. The flag stored in the alpha channel is used to decide whether to initially follow front or back links: $cellInFrontOfStartFace$.

The result of the inner loop is the face through which the ray leaves the cell, as well as the flag indicating whether the next cell is in front or behind this face. Note that non-convex cells can be entered and exited multiple times.

Before traversal is continued in the outer loop, the entire ray segment between the entry and exit faces of the inner loop is sampled and composited. The exit face and negated flag resulting from the inner loop completion then become the input to the next iteration of the outer loop. The simple operation of logically negating the flag before the next loop iteration is what performs the step from one cell to the next. The outer loop terminates when the ray has exited the brick.

Ray-face intersection: The inputs to the ray-face intersection algorithm are: a face ID, a ray direction $\mathbf{v}$ and origin $\mathbf{o}$, and a boolean flag s that indicates whether the face normal should be flipped for the intersection test as described below. The output is the distance to the intersection,

or ∞ if there is no intersection. An intersection with a face is only reported when the local face normal $\mathbf{n}_x$, at an intersection point x, points in the same direction as $\mathbf{v}$, i.e., $\mathbf{n}_x \cdot \mathbf{v} > 0$. If s is *true*, the face normal is flipped for this comparison. This test, together with a check whether the intersection is further down the ray than the cell entry position, correctly handles faces that are viewed edge-on. It also avoids erroneous detection of an intersection with the face through which the ray has entered the cell. In the case of orthogonal projection, we further simplify the computations for ray-face intersection by transforming all vertices of a brick from world space to view space, and performing intersection in the latter. The ray direction then is $(0, 0, -1)$, which reduces the required 3D ray-triangle intersections to computing 2D barycentric coordinates for $\mathbf{o}$ transformed and projected into view space. The involved dot product reduces to a single multiplication. For further optimization, our system integrates three different types of face geometry representations at a per-brick granularity (see also Section 2.4.1.3). Each type employs its own optimized ray-face intersection:

1. For purely tetrahedral bricks, the three vertex indices of each triangle are stored, which makes the ray-face intersection trivial.

```
 1: function MESHTRAVERSAL
 2:     Read startFace and cellInFrontOfStartFace from ray-setup images
 3:     repeat                                   ▷ This loop traverses all cells along a view ray
 4:         currentFace ← startFace, d_min ← ∞
 5:         cellInFrontOfFace ← cellInFrontOfStartFace
 6:         repeat                               ▷ This loop iterates over all faces of a cell
 7:             d ← RayFaceIntersection(currentFace, cellInFrontOfFace)
 8:             if d < d_min then
 9:                 d_min ← d
10:                 minFace ← currentFace
11:                 cellInFrontOfMinFace ← cellInFrontOfFace
12:             end if
13:             if cellInFrontOfFace then                    ▷ Go to next face of cell
14:                 advance currentFace along face sequence
15:                 if sequence terminator reached then
16:                     cellInFrontOfFace ←GETFRONTFLAG(currentFace)
17:                     currentFace ←GETFRONTLINK(currentFace)
18:                 end if
19:             else
20:                 cellInFrontOfFace ←GETBACKFLAG(currentFace)
21:                 currentFace ←GETBACKLINK(currentFace)
22:             end if
23:         until currentFace = startFace
24:         Perform sampling between startFace and minFace
25:         startFace ← minFace
26:         cellInFrontOfStartFace ← ¬cellInFrontOfMinFace
27:     until startFace is boundary face or ray exits brick
28: end function
```

Figure 2.19: Pseudo code for mesh traversal during raycasting using the TSFSL data structure. Note that a face in this context simply denotes a face reference (ID), not actual face geometry data.

2. For bricks with cells where all faces are either triangles or quads (e.g., a mixture of tetrahedral, hexahedral, and octrahedral cells), quads are decomposed on-the-fly into two triangles. The ray-face intersection is then carried out separately for these two triangles. Note that this case includes non-planar faces.

3. For bricks containing cell faces with more than four vertices, i.e., an arbitrary number of vertices describing possibly highly non-planar faces, we use the general strategy described below.

For efficiency reasons, our system triangulates cell faces with more than four vertices as soon as a mesh is loaded, instead of triangulating them on-the-fly during raycasting. For this purpose, arbitrary triangulations can easily be integrated into our system. We have done extensive experiments with triangulations based on either a triangle strip or a triangle fan. Only the latter has turned out to work very well in practice. A very important consideration for face triangulations is that in complex real-world meshes the faces are often non-planar, ranging from slightly non-planar to extremely non-planar. In the meshes that we have examined, a triangle strip decomposition of each face has turned out to be infeasible. These meshes contain at least some faces that are so highly deformed that they cannot be decomposed into triangle strips without introducing mesh inconsistencies. Our current approach therefore decomposes each face into a triangle fan around the centroid of all face vertices. Such a triangulation is symmetric, while the triangle strip decomposition is not. Ray-face intersection during raycasting is then performed by intersecting the ray with each triangle of the fan individually. A drawback of triangle fans around the centroid is that an additional centroid vertex has to be stored per face. Naturally, general ray-face intersections are costly. Therefore, for each face we also store the squared radius r^2 of a bounding sphere centered at the centroid. Full ray-face intersection is only performed when the ray's intersection with the face's bounding sphere is non-empty. This test is also performed in screen space which reduces the ray-sphere intersection to simply computing the squared distance between the ray position and face centroid in 2D screen space. The proposed triangle-fan face-decomposition is not feasible for a cell if the resulting triangles intersect each other. To cope with such cells, a more general face-geometry representation, such as an actual triangle mesh, could be used. However, none of the real-world meshes that we have evaluated actually contained such cells.

Sampling and interpolation: The final component of raycasting is obtaining values at sample positions by using different interpolation schemes. We switch between two different interpolation methods on a per-brick basis:

1. For vertex-centered data, we employ mean value interpolation in the interior of complex cells (see Section 2.3), and barycentric interpolation in purely tetrahedral cells, as well as on all triangular faces.

2. For cell-centered data with a constant scalar value per cell, we employ piecewise constant interpolation. This is very useful in practice, because in many cases real simulation data from state-of-the-art packages [1–3] can be cell-centered. Thus, this simple approach generates visualizations that are an adequate representation of the simulation results.

For mean value interpolation, we allow the user to enable two different optimizations during interaction that slightly reduce image quality while increasing rendering performance significantly. First, we can fall back to *linear interpolation* along a ray segment between cell-ray entry and exit positions. Second, we can use an *adaptive scheme*, which alternatively uses either linear interpolation along a ray segment or inserts one mean-value sample at the segment's center. Subsequently, linear interpolation between entry, center, and exit points is used. The additional mean-value sample is only used when the ray-segment length exceeds a user defined threshold. Even for high quality visualizations, we found it to be sufficient to use one mean-value sample at the center of each ray segment within a cell. This can be attributed to the nature of mean value interpolation, which deviates most strongly from linear behavior in the cell interior, whereas it quickly converges toward barycentric interpolation at the cell faces. Note that in the case of non-convex polyhedra, mean value interpolation can generate negative weights for faces whose exterior side is facing toward the sample position. In order to solve this, positive mean value interpolation could be used instead [81]. However, this would come at the expense of smoothness, since positive mean value coordinates are not guaranteed to remain smooth within non-convex polytopes.

As an additional consideration, using complex interpolation methods for cells that are smaller than one image pixel in the final visualization is a waste of processing resources. Thus, we automatically enable piecewise constant interpolation for bricks where all cells project to one image pixel or less. These bricks are determined by storing their maximum cell size and calculating the corresponding projected size at the distance of the brick corner closest to the view plane.

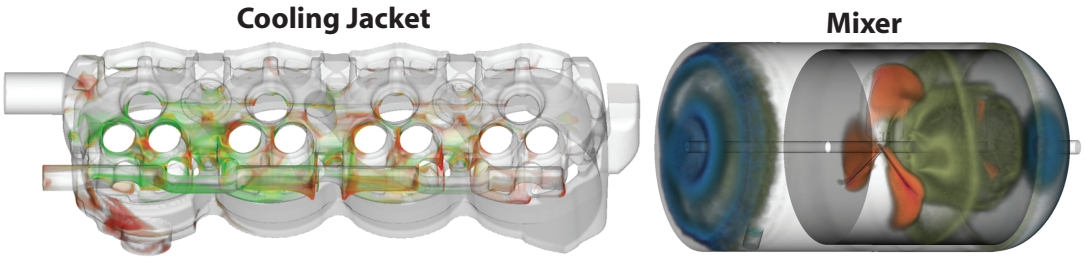
2.4.3 Results

We discuss important implementation details, give results for complex real-world meshes, and compare our approach with previous work.

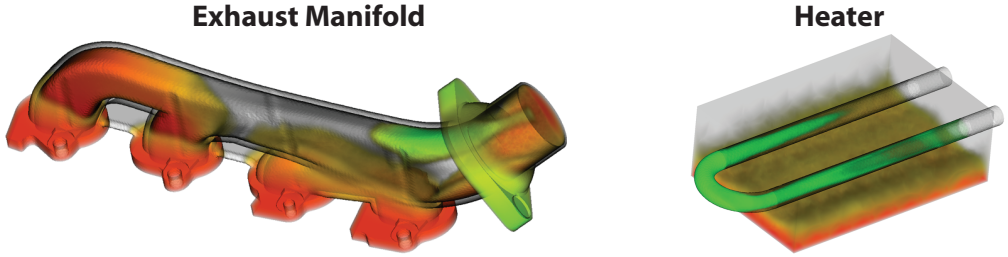
2.4.3.1 Implementation

This section discusses details of how the TSFSL representation and additional mesh data are stored in memory, as well as our TSFSL raycasting implementation.

Memory layout: Our raycasting system is implemented in OpenGL and GLSL. The entire TSFSL data structure for each brick is stored in a single 1D 32-bit integer texture. Each face is represented by a 32-bit int header followed by the vertex index data. The header contains a 24-bit address for the face's back link, 6 bits for the size of the following vertex data, and two additional flag bits. One is the back flag for the back link, and the second flag indicates whether the face is the last in the sequence. If the latter bit is set, the integer value following the face vertex data is the face sequence terminator. As in the face header, the int of the face-sequence terminator is split into a 24-bit address for the front link, and one bit front flag. In the case of cell-centered data, the terminator is followed by a 32-bit int addressing the cell corresponding to the face sequence that is terminated. If triangle-fan face-decomposition is used, we store all vertex indices of a face, starting with the center vertex, as 32-bit int indices. For triangle and quad faces, each vertex is a single 32-bit int.



Cells/Vertices/Faces:	1,538 K / 1,631 K / 4,707 K	1,362 K / 7,432 K / 8,869 K
Tetrahedra:	17,044 K (~8.5 byte/tet)	89,417 K (~7.5 byte/tet)
Celltypes:	tets/pyramids/wedges/hexas	general (non-convex) polyhedra
Bricks/Cell Overhead:	4/1.7%	10/8.6%
TSFSL Creation Time:	4.0s	9.0s



Cells/Vertices/Faces:	82 K / 324 K / 441 K	17 K / 68 K / 91 K
Tetrahedra:	4,095 K (~7.0 byte/tet)	851 K (~7.0 byte/tet)
Celltypes:	general (non-convex) polyhedra	general (non-conv.) polyh.
Bricks/Cell Overhead:	1/0%	1/0%
TSFSL Creation Time:	1.7s	1.0s

Figure 2.20: The data sets used to evaluate our TSFSL raycasting approach, with mesh statistics. Timings for raycasting are given in Table 2.7. The cooling jacket is courtesy of AVL List GmbH, Graz, Austria whereas the remaining three data sets have been supplied by CD-Adapco.

Raycasting: Two different kinds of temporarily derived data are generated before raycasting each brick (see Section 2.4.2): First, all vertex positions are transformed into view space by utilizing the transform feedback capabilities of current GPUs. Then, the mesh boundary containing b faces is triangulated on-the-fly by a geometry shader. The input primitives for this shader are b points, for each of which only a single integer index is stored. This index is the address of a face in the TSFSL data structure that is subsequently triangulated according to the current face-triangulation settings. On current GPUs, we are limited to generate triangle strips in the geometry shader. We therefore generate the triangle-fan decompositions of complex cell faces by inserting degenerate triangles. This approach is preferable to restarting a new strip after each triangle since fewer vertices have to be emitted by the shader. After depth peeling has been performed for a brick, the triangulation data as well as the transformed vertex positions are discarded. This approach is feasible because the overall mesh is processed brick by brick, each of which contains only a fixed maximum number of cells/faces/vertices. The shading effects

visible in Figure 2.20 are generated by applying limb darkening [57].

2.4.3.2 Data Sets

In order to demonstrate the usefulness of our new volume-visualization algorithms we have selected the four representative data sets depicted in Figure 2.20, which range from fairly small to reasonably large in size. Among these four data sets, two different configurations of our raycasting approach have to be employed in order to cope with different cell characteristics. In order to compare memory requirements and rendering performance, we provide the number of tetrahedra into which the mesh can be decomposed. The tetrahedralization scheme taken to derive this number is selected based on the face-triangulation approach used for the respective mesh. Each sub-triangle of an interior face results in two tetrahedra made up of the triangle vertices and the cell centroid of either the cell in front or behind the triangle. Sub-triangles of boundary faces only generate one tetrahedron. Although this is not the tetrahedralization resulting in the minimum number of cells, we have used it to obtain result images that are comparable with our results, which use mean value interpolation.

The *heater*, *manifold*, and *mixer* data sets have been created by the same simulation package [3] and therefore share common mesh characteristics. All meshes comprise general polyhedral cells with often highly non-planar faces. We cope with this by utilizing a triangle fan decomposition of each face. Note that as soon as more complex cells are present, the number of tetrahedra generated by tetrahedralization can increase between one to two orders of magnitude when compared to the original cell decomposition. The *cooling jacket* data set [85] contains only a fixed set of cell types (tetrahedra, quadratic pyramids, triangular prisms, and hexahedra), which again can be non-convex. The mixer and the cooling jacket data sets are decomposed into four and ten bricks, respectively, due to their size. The increase of the number of cells that have to be stored due to bricking is shown in Figure 2.20.

2.4.3.3 Evaluation and Discussion

The evaluation of the methods proposed in this work is complex and will be performed in multiple parts. First, we compare the memory requirements of the TSFSL mesh representation to other data structures that have been used to perform volume rendering. Next, we evaluate the rendering performance of different quality and sampling settings, and discuss their impact on the resulting image quality.

Memory Consumption: This comparison is difficult since to our knowledge there exist only three other GPU volume renderers capable of directly rendering convex polyhedral cells [21, 89, 129]. All of those can process non-convex cells without requiring subdivision. The HAVS level-of-detail extension has been included in this list since it can cope with polyhedral cells on a basic level. It is, however, limited to piecewise linear interpolation along ray segments within a cell because only face-to-vertex connectivity data is available to the GPU. We compare the memory requirements of our TSFSL structure with different state-of-the-art representations by giving the number of bytes per tetrahedron in Table 2.6. For non-tetrahedral meshes, this is computed by dividing the overall memory consumption by the number of tetrahedra in an

equivalent tetrahedralization. The memory overhead that is introduced by our technique due to bricking is included in these figures.

The hybrid raycasting algorithm presented in Section 2.3 is optimized for meshes containing only few cell types and requires convex cells composed of triangular or planar quadrangular faces. Slicing-based volume rendering on CIEL is mainly executed on the CPU, because only the slice rasterization is performed on the GPU, and thus requires basically no GPU memory. The per-tetrahedra memory consumption for CIEL has been estimated based on the overall memory consumption stated in the corresponding publication. Scalar data are represented as polynomials in the case of the higher-order finite-elements visualization technique (HOFEV) [129]. Therefore a direct comparison to the other techniques in Table 2.6 would be biased toward techniques dealing with conventional CFD data. In order to avoid this biasing, we have compared the memory required to store only topology information. In the case of data sets such as the heater or the manifold, the TSFSL representation uses 34% less memory than the topology data structure employed by HOFEV. For simpler data sets, such as the cooling jacket, this difference is 24%. Similarly to TSFSL, HAVS uses face data to represent a volume mesh. Only three vertex indices per face as well as vertex positions and scalar data are stored in GPU memory. This results in very competitive memory requirements on the GPU side. However, a significant amount of memory is used to perform efficient face sorting on the CPU. The per-tetrahedra memory figures are based on our four test data sets. In summary, Table 2.6 clearly shows that our GPU-based data structure is superior with respect to memory consumption to most state-of-the-art grid representations for volume rendering. Only HAVS utilizes a data representation on the GPU that is equally efficient.

Rendering Performance: Besides the memory footprint, the actual rendering performance of our raycasting approach has to be evaluated. However, we note that the main goal of this work is to reduce the memory that is necessary to represent an unstructured mesh, while still allowing for efficient raycasting. A major reason for this rationale is that recent developments in graphics hardware indicate that computing speed is growing far more rapidly than on board memory size. Table 2.7 compares the performance of an object-order volume-rendering-method (HAVS) and a state-of-the-art GPU-based tiled raycasting method (TRC) to TSFSL volume rendering. All tests have been performed by rendering into a 1024x768 view port on a Core 2 Quad CPU at 2.8 GHz with 8 GB of RAM and an NVIDIA Geforce 480 GTX with 1.5 GB of memory. The different results listed for raycasting with TSFSL are due to different quality and sampling settings. The rows labeled high quality (HQ) represent results generated by using the full 1024x768 render target. HAVS and TRC create output images with the same resolution. The low quality (LQ) figures were measured when rendering only 512x384 images that were subsequently scaled to 1024x768 while also only performing raycasting for one depth peel per brick. The disproportionately strong performance increase between the HQ and the LQ modes for the manifold data set is caused by the fact that the data set consists of a thin layer of geometry representing solid parts surrounding fluid cells. Thus, terminating the raycasting after one depth peel skips a considerable part of the overall data volume. When working with the mixer data set, the memory requirements for HAVS exceeded the 8 GB of RAM installed in the workstation used for benchmarking. Therefore, no conclusive results were measured due to constant swapping oper-

Algorithm	Cell Types	Byte/Tet
TSFSL	general (non-convex) cells	$\sim$ 7 - 9
HARC [139]	tetrahedra only	160
TRC [11]	tetrahedra only	144
HARC-Partial [41]	tetrahedra only	96
Hybrid Rayc. (see Section 2.3)	convex cells (few celltypes)	$\sim$ 45 - 80
CIEL [89]	general (non-convex) cells	$\sim$50 CPU
VF-Ray-GPU [97]	tetrahedra only	38
Tet. Strips [141]	tetrahedra only	$\sim$15
HAVS [22]	tetrahedra only	$\sim$ 7 - 9 GPU $\sim$118-149 CPU

Table 2.6: Memory footprint comparison of TSFSL with state-of-the-art unstructured grid rendering approaches (measured in bytes/tetrahedron).

ations by the operating system. When assuming linear scaling in the number of grid triangles, rendering the mixer data set with HAVS should require around 16 seconds.

Interpolation and Image Quality: We have evaluated four different sampling strategies. The fastest is piecewise-constant sampling using cell-centered data. We note that all four test data sets originally contain cell-centered data, and therefore this sampling approach properly represents the original simulation results. However, if smoothed results are desired for presentation purposes, vertex-centered data can be visualized. Our rendering system uses barycentric interpolation at cell faces. Results stated in the rows labeled as "linear" have been generated by using linear interpolation between ray entry and exit positions of a cell. In order to generate higher quality images, one mean-value sample is used at the midpoint between ray entry and exit position. Performance figures for this sampling strategy are given in the rows denoted as "mean value". As can be expected, the complex computations necessary per cell face incur a severe performance hit on the overall rendering time. This is especially prominent for data sets containing general polyhedral cells (heater, manifold, and mixer). The main differences between images generated by the fast "linear" quality setting and the "mean value" option lie in regions where large cells are located (see image quality comparison in Figure 2.21). Therefore, we propose a fourth "adaptive" sampling strategy that is controlled by a user defined threshold θ. Reconstruction using mean value interpolation is only used if the length of a viewing-ray segment within a cell is greater than θ. Otherwise the much faster linear reconstruction is used within a cell. The user has the option to choose two out of the four sampling strategies in their HQ or LQ configurations for rendering preview images during interaction, and presentation quality visualizations when interaction has stopped.

When compared to HAVS, our high-quality TSFSL methods are only faster for reasonably large data sets such as the cooling jacket. Here, the main limiting factor for HAVS is the number of triangles ($\sim$53 M), which have to be sorted by the CPU. Also note that the HAVS algorithm generates rendering artifacts on current generation graphics hardware since it performs reading from and writing to the same texture target within one GPU program. This operation results in

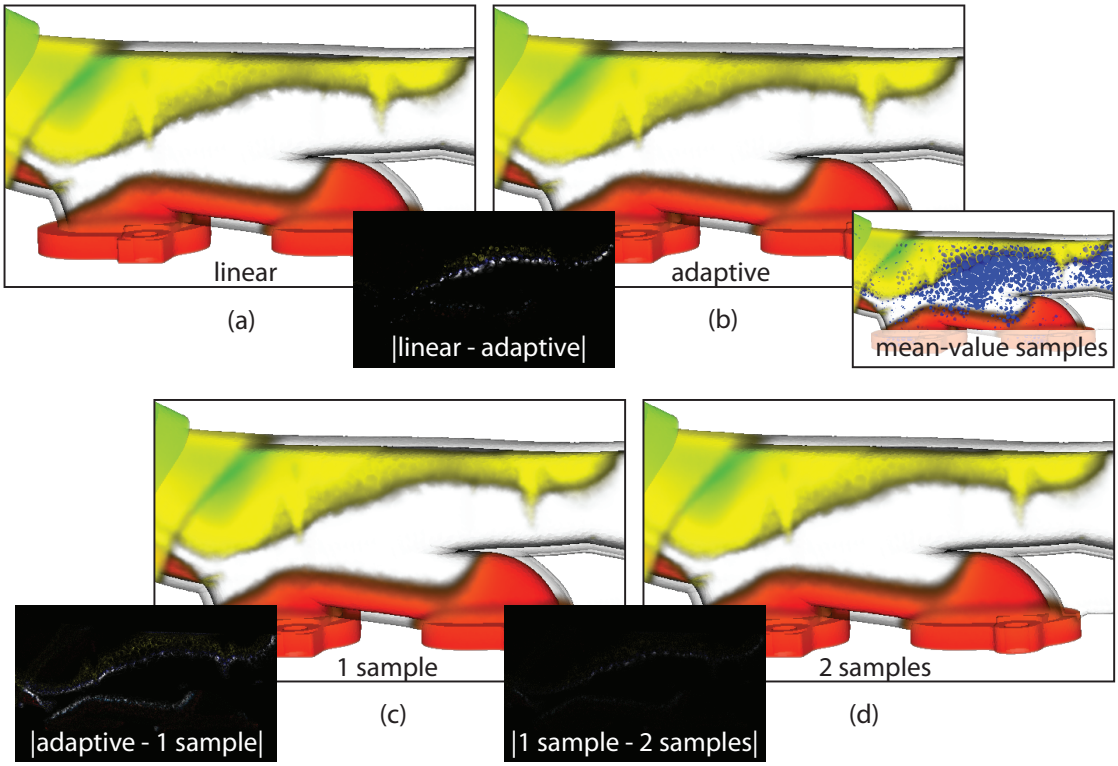


Figure 2.21: Comparison of the influence of different sampling strategies on image quality. The three inlays labeled "|linear-adaptive|", "|adaptive - 1 sample|", and "|1 sample - 2 samples|" depict the differences between (a) and (b), (b) and (c), and (c) and (d), respectively. Difference-color values have been scaled by a factor of 16. The additional inset denoted as "mean-value samples" indicates regions in (b) for which mean value interpolation has been used (colored in blue).

undefined behavior in OpenGL and in DirectX. The tiled raycasting approach scales similarly to our low quality volume rendering implementations. However, it requires much more GPU memory (see Table 2.6), and the publicly available open source implementation has not been able to load the cooling jacket nor the mixer data set (even though a 64-bit binary has been used).

Image Quality: Figure 2.21 compares the image quality of the different sampling strategies that are supported by our rendering framework. The transfer function used throughout the comparison represents a worst case scenario, which highlights the differences between the sampling methods. This is because it comprises a nearly opaque region and a completely transparent region without a smooth transition in between. Figure 2.21(a) depicts the fastest sampling strategy for vertex-centered data, which interpolates linearly between ray-cell entry and exit positions. The adaptive sampling method is used in Figure 2.21(b). The linearization artifacts introduced by the first approach are reduced by incorporating one mean-value sample for ray segments that are longer than a user defined threshold θ. If θ is chosen to be larger than the diameter of the

Algorithm		Heater	Manifold	Cooling J.	Mixer
TSFSL	HQ	$124ms$	$156ms$	$172ms$	$1.8s$
(cell-centered)	LQ	$48ms$	$49ms$	$65ms$	$512ms$
TSFSL	HQ	$145ms$	$181ms$	$192ms$	$2.2s$
(linear)	LQ	$50ms$	$49ms$	$68ms$	$579ms$
TSFSL	HQ	$1.3s$	$1.7s$	$1.2s$	$28.1s$
(mean value)	LQ	$343ms$	$226ms$	$325ms$	$6.4s$
TSFSL	HQ	$261ms$	$303ms$	$222ms$	$2.9s$
(adaptive)	LQ	$91ms$	$56ms$	$81ms$	$742ms$
HAVS [22]		$125ms$	$604ms$	$2.8s$	-
TRC [11]		$150ms$	$170ms$	-	-

Table 2.7: Performance measures and comparison with state-of-the-art unstructured grid rendering approaches. Rows labeled HQ correspond to high rendering quality (1024x768), and rows labeled LQ correspond to low rendering quality (512x384).

largest cell, then only the linear approximation is used, whereas if θ is set to zero, a mean-value sample is selected for every ray segment. The viewing rays for which mean-value samples have been computed are highlighted in the inset as blue regions. Figure 2.21(c) shows a result image where one mean-value sample has been used per ray segment within a cell. In order to qualify differences between visualizations we have included difference images that have been scaled up by a factor of 16. They clearly show how linear interpolation and the adaptive sampling strategy differ in regions where mean-value samples have been included. Naturally, the largest differences between the adaptive sampling scheme and always using one mean-value sample show up in regions composed of small cells. The number of mean-value samples per ray segment within a cell is limited to one in all our sampling strategies. Additional samples only marginally add to the overall image quality. This is illustrated in Figure 2.21(d). Here, two mean-value samples have been computed per ray segment. The difference image between Figures 2.21(c) and (d) barely shows any distinguishable features.

2.4.4 Summary and Conclusions

The raycasting method presented in this section is a direct extension to the hybrid raycasting system discussed in Section 2.3. It can be used instead of the original unstructured grid rendering technique that has been limited to simpler cells and required significantly more memory. The new face-based representation for general unstructured grids is highly scalable with respect to grid/cell complexity and does not store redundant information. Our *two-sided face sequence list* (TSFSL) representation is flexible enough to support the efficient traversal of rays from cell to cell, as well as enabling efficient access to all information required for high-quality interpolation, such as mean value schemes. These are the two crucial components required for volume visualization via raycasting, as well as other important visualization algorithms. In order to demonstrate the feasibility of the TSFSL representation in practice, we have presented a complete GPU-based unstructured-grid raycasting pipeline. Our system can handle large data

sets via bricking, and is able to efficiently cope with both non-convex mesh boundaries and non-convex polyhedral cells, even in the presence of highly non-planar faces.

CHAPTER 3

Information Visualization for Scientific Data

3.1 Introduction

In the previous chapter scientific visualization methods have been used to display simulation data. Such techniques rely on the inherent connection between data items and their corresponding volumetric cells in order to represent the data in its original spatial context. That is, the data volume is projected directly onto an image plane. Information visualization methods are frequently associated with data that is either loosely or not at all related to a spatial location. Therefore a direct projection into image space is typically not possible. Instead different data attributes are used to define the visual representations and image-space positions of data items. Another important distinction is that information visualization methods often do not treat data as continuous. As with Lagrangian volume data, scattered data interpolation based on a fixed number of nearest neighbors could be used to calculate additional data samples. This, however, is only feasible if the data attributes themselves are continuous. Interpolation for discrete or categorical data, which are common in information visualization applications, does not generate meaningful results. Relational data, such as graphs, are also associated with information visualization techniques. Besides these fundamental differences, data associated with scientific visualization is frequently larger with respect to the number of data samples whereas information visualization deals with data that contains a large number of attributes and is often more complex on a conceptional level.

The primary motivation for using information visualization techniques in the context of simulation data is that state-of-the-art numerical models generate data that is increasingly complex. Instead of just accounting for the physical properties of fluid flow, additional processes are modeled more and more frequently. This greatly increases the number of volume attributes that have to be explored and analyzed. Combustion simulations, for example, track chemical reactions and their reagents while climate models track a multitude of variables such as water vapor, precipitation, temperature, and different species of green house gases. Information visualization

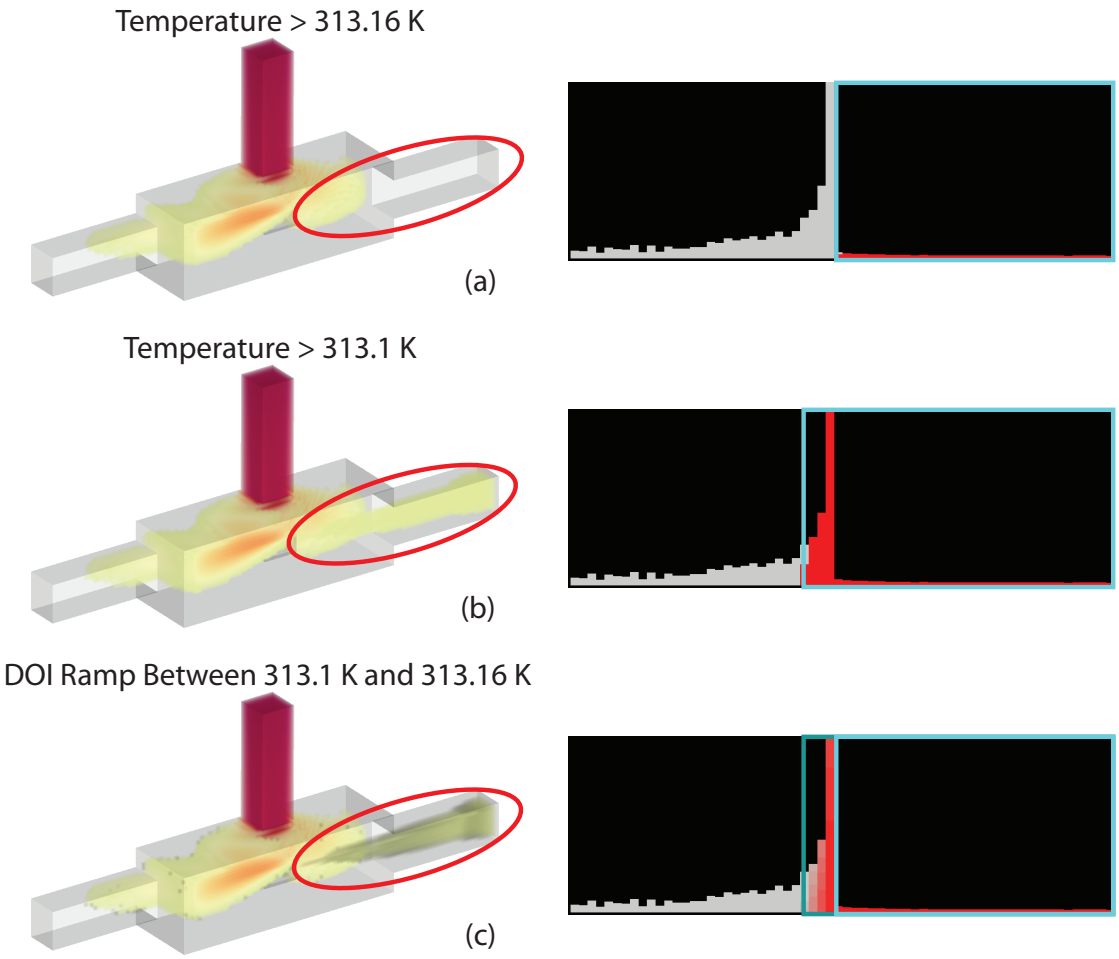


Figure 3.1: The left column of images shows the features that have been selected by thresholding in the histograms in the lower row. The DOI is mapped to opacity and luminance. Regions of high DOI are visualized bright and opaque whereas low DOI portions of the data set are indicated by darker colors and more transparency. A slight variation in the threshold that is used to define the feature can result in large changes of its spatial extent (compare (a) and (b)). Regions that are sensitive to changes of the threshold parameters can be visualized if a smooth DOI function is used (see (c)).

methods tailored toward exploring and analyzing relations between different attributes of high dimensional data are therefore well suited for this kind of high-dimensional simulation results. The following chapter will focus on enhancing standard information visualization techniques with respect to data typically associated with scientific visualization methods.

As in Chapter 2 different types of scalability will be addressed. The proposed approaches mostly work on an image-order basis. This means that only a minimal computational overhead per data item is introduced. The desired visualization effects are created via image processing.

This is especially important for achieving scalability in the number of data items that is required when dealing with large data sets. Contrary to this computational scalability, visual scalability [112] is related to the capability of a visualization technique to convey information about increasingly large numbers of data items. This kind of scalability is crucial for information visualization methods that are applied to data from numerical simulations since often hundreds of millions or even billions of data items have to be displayed. Explicitly representing each data item in one image becomes infeasible for such data sets. In order to address this challenging problem we introduce techniques for a class of visualization methods that provide an overview on prominent trends in the data. A novel data highlighting scheme is used to emphasize individual data items that are of interest to the user.

3.2 The SimVis Framework

All methods that are presented in this thesis have been integrated into the SimVis framework [31] and have therefore been used in a large number of scientific collaborations and publications [45, 68, 107]. The system is designed to support the interactive exploration and analysis of data that has been generated by different kinds of numerical models. Such simulation results are typically large in the number of data items and frequently contain a large number of, often time dependent, data attributes. Different concepts that have been introduced in information visualization are used to cope with challenges posed by this kind of data. In order to provide a proper overview on different aspects of a data set the SimVis framework provides *multiple linked views* (see Figure 2.5). Each view can use a different visualization method that either shows the data in its original three dimensional context by using a scientific visualization technique or as an attribute-space based information visualization representation. Supported scientific visualization methods include the direct volume-rendering algorithms presented in Chapter 2 as well as indirect volume-visualization techniques such as slicing and isosurfacing, all of which can be combined into one integrated visualization. Scatterplots, histograms and parallel coordinates are some of the information visualization techniques that have been incorporated into the SimVis framework.

Feature-Based Visualization & Smooth Brushing: All views use *feature-based* visualization-techniques. Portions of a data set, that have been identified by the user as important are considered to be features. Each feature is represented by a degree of interest function (DOI) that is one for data items inside the feature and zero otherwise. This binary distinction is sufficient for categorical or discrete data that is typically associated with classical information visualization applications. However numerical simulations of physical processes generate results that are smoothly distributed across the modeled volume. In this case a binary feature definition introduces problems that are illustrated by the following example. Let a feature be defined by a simple threshold t with respect to one scalar data attribute $s(x)$. A data item at position x is part of the feature if $s(x) > t$. This binary classification is highly unstable with respect to regions for which the gradient magnitude of $s(x)$, $|\nabla s(x)|$, is low and $s(x)$ is close to t. That is, modifying t only a little bit will change the feature's spatial dimensions in a significant way. This is illustrated in Figures 3.1(a) and (b). The slight change of the feature-defining temperature-threshold

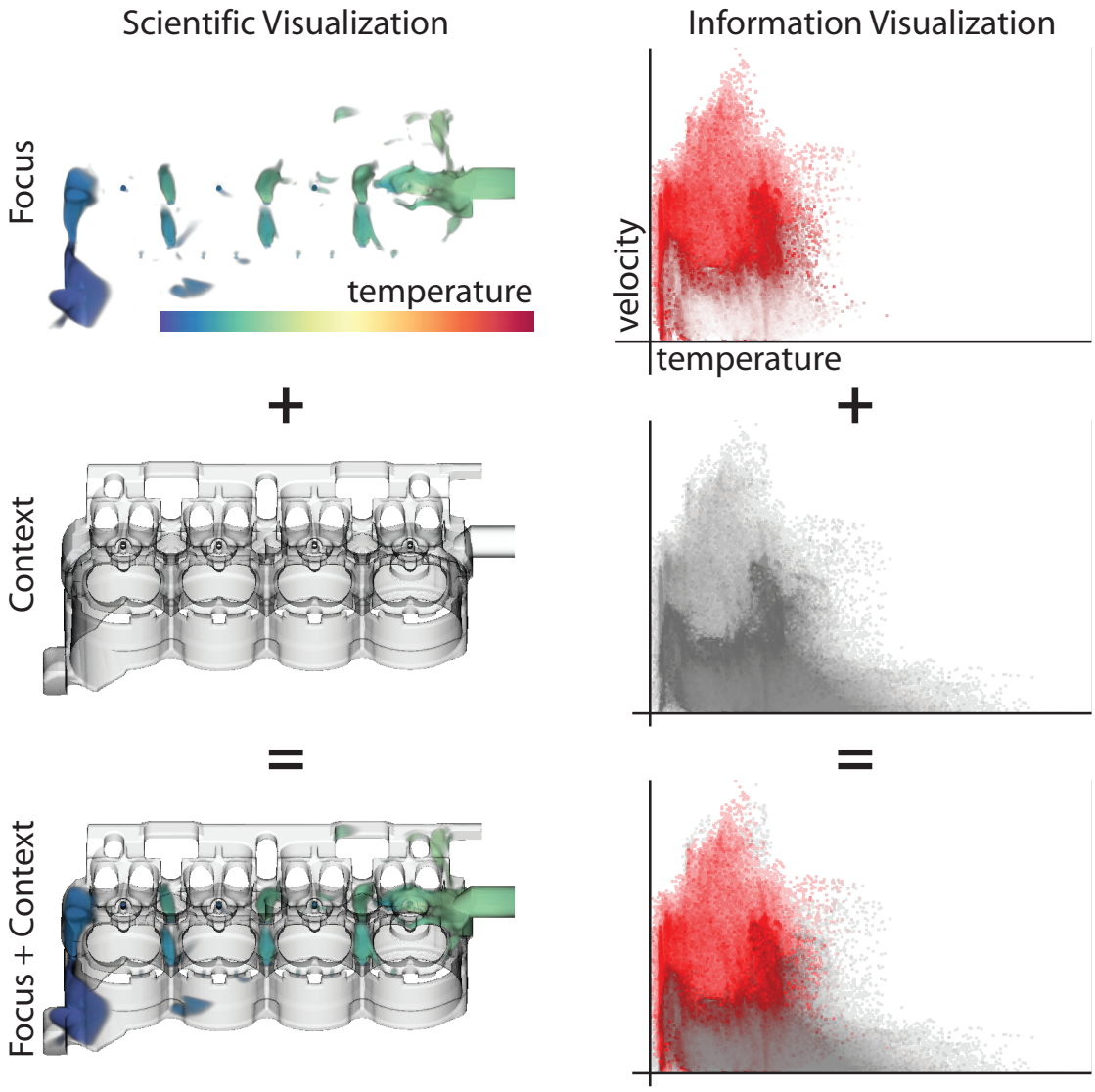


Figure 3.2: The SimVis framework uses focus+context rendering in all views. The left column of this figure illustrates this for volume visualization whereas the right column shows a simple scatterplot. Both columns display data from a simulation of a cooling jacket within a diesel engine. The feature that is displayed as focus has been defined by selecting regions of high turbulent kinetic energy. The transfer function indicated in the top-left image is used to map temperature to color in the volume rendering examples.

results in large changes in the feature's spatial extent. A continous DOI function that assigns values from the unit interval to each data item alleviates this problem. Now, instead of one cutoff threshold t, a transition region between t_l and t_u can be used. Spatial regions that exhibit small $|\nabla s(x)|$ and for which $t_l < s(x) < t_u$ are represented as large regions that are only partially

selected (see the region indicated by the red ellipse in Figure 3.1(c)). This allows the user to get an overview on the spatial gradient magnitude of $s(x)$ for $t_l < s(x) < t_u$ and therefore gain information about the sensitivity of the feature with respect to changes of the defining thresholds. SimVis relies on a smooth feature definition scheme [32].

The actual feature definition is performed by interactive *brushing* methods that are integrated into all information visualization views. The user can employ different selection tools within each view to mark portions of a data set as interesting features. Transition regions can be used to partially select data items. The lower row of images in Figure 3.1 shows how a simple histogram can be used to brush data either binarily (a, b) or smoothly (c). In the binary case the user specifies a rectangle that encloses all data items that should be selected. A smooth selection is defined by a second rectangle that encloses the first one. The brushing information is translated into the smooth DOI function that is stored per feature. SimVis utilizes a *feature definition language* (FDL) to organize individual features into a tree structure that can be used to recombine multiple features via logical operations. Since the DOI functions are continuous, fuzzy logic norms are employed to compute these combinations. Section 3.3.2 will provide additional details related to the organization and definition of individual features.

Focus+Context Visualization: Focus+context rendering techniques [54] are used to distinguish important features (the focus) from the remainder of the data set (the context). Important data is represented by very prominent rendering styles (e.g., high opacity, bright colors) whereas the context is visualized less obtrusively. This enables the user to focus on important features while at the same time retaining an overview of the whole data set. Figure 3.2 illustrates this by showing how simple focus+context visualization techniques can be applied to scientific visualization and information visualization methods. Section 3.3.2 shows how this simple color-based highlighting for information visualization views can be extended in order to cope with multiple features.

Derived Data: In order to define complex application-context dependent features within the SimVis framework data derivation can be used. Data attributes that are computed by this functionality can be as simple as differences between two already existing data attributes or as complex as quantities that are related to flow properties in a CFD simulation. In Chapter 4 for example a vortex detection measure is computed from a flow field. Buerger et al. [17] use the data derivation support of the SimVis framework to compute and compare several different such vortex detectors. Instead of deriving quantities related to flow properties, Kehrer et al. [69] compute different statistical moments for ensemble simulations in order to visualize the statistical properties of a large number of simulation runs.

3.3 A Four-level Focus+Context Approach to Interactive Visual Analysis of Temporal Features in Large Scientific Data

This section is based on the paper *A Four-level Focus+Context Approach to Interactive Visual Analysis of Temporal Features in Large Scientific Data* [104]. One interesting recent devel-

opment is the increased share of time-varying scientific measurements and computations. In numerical simulation, time-dependent representations of dynamic phenomena are commonly used to solve real world problems. Additionally, fast imaging devices and improvements in the fields of registration and motion correction allow for time-dependent data also in the medical domain. Traditionally, time-dependent data is either visualized by displaying separate individual time steps or by specifying a few positions within the data set for which the evolution of a data attribute is visualized over time (by drawing function graphs). Both approaches provide only a selective view of the data set, either with respect to time, when showing just one timestep at a time, or with respect to space, when showing the evolution of an attribute for a few sample positions, only. Users interested in temporal features in a time-dependent data set either have to mentally integrate multiple images from different time steps or search for interesting spatial regions by choosing different sample locations and looking at the resulting function graphs. Especially if the temporal evolution of corresponding data items, e.g., the change of temperature within a Computational Fluid Dynamics (CFD) cell over time, is investigated, the latter, curve-based approach is far more effective than looking at multiple snapshots or animations. Still the selection of regions of interest, especially during data exploration and analysis, is a tedious task if no proper context information about the overall data set is provided.

We propose a new dense visualization approach by displaying function graphs for all cells/voxels of a volumetric, time-dependent data set. This leads to visualizations consisting of hundreds of thousands or even of millions of function graphs, sampled at a relatively low number of time steps. Existing visualization technology for time series data has not yet addressed such large numbers of function graphs concurrently. The following section will discuss a solution to this challenging problem that is visually and computationally scalable.

In order to improve on occlusion and cluttering problems, resulting from the huge number of function graphs to draw, we utilize a highly interactive four-level focus+context method. We use data aggregation and image space methods to retain responsiveness even if interacting with large data sets. Our approach is integrated with an interactive visual analysis technique of coordinated multiple views [31]. Features are specified by the user in a hierarchical scheme which is based on individual selections and logical combinations of them on higher levels. We allow for smooth transitions between focus and context, based on the concept of smooth brushing [32], which are represented by degree of interest (DOI) attributions of the data. We consider three different DOI values per data item and present how this is integrated with our visual analysis approach.

In order to account for the temporal nature of the data, we extend the brushing functionality of our framework by *function-similarity brushes* which are based on a user definable pattern. These patterns, which can be sketched and modified interactively, are matched to the data curves, also resulting in smooth DOI attributions. Graphical context information is provided by using line integral convolution [18] (LIC) on a synthetic flow field which represents the temporal evolution of each data item. We also propose to use a binned data representation to increase the rendering performance as well as additional image-based techniques to further enhance the visualization.

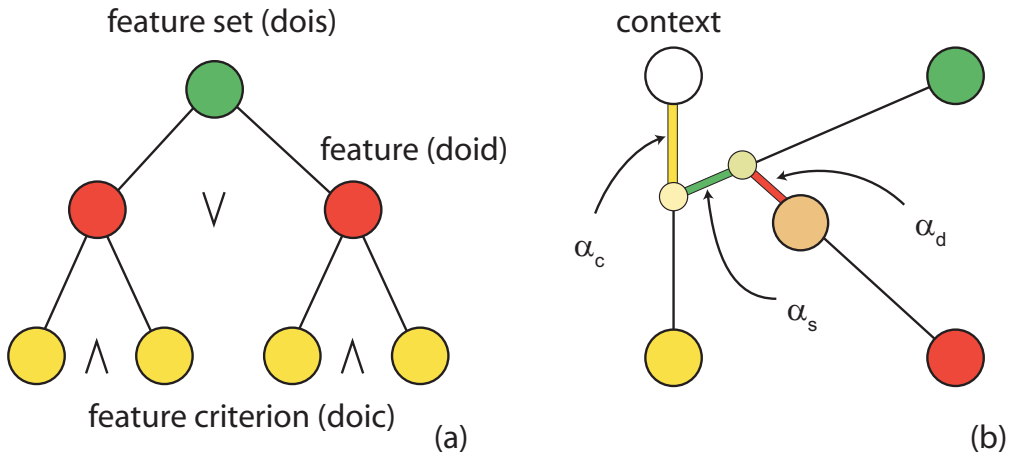


Figure 3.3: The structure of the FDL tree (a) and the optimized blending scheme for color coding (b). The three colors represent different levels of the FDL tree.

3.3.1 Related Work

A large variety of publications deal with the visualization and analysis of *time-oriented information*. See the work of Aigner et al. [4] for an overview. Several applications exist that allow the analysis of time-dependent data using interactive brushing or querying techniques. Feature visualization and specification via linking and brushing in multiple views is an integral part of the *SimVis* framework [29]. Scalar degree of interest (DOI) values from the unit interval represent the membership with respect to features for all data items. The values are assigned, for example, using *smooth brushing* [32], where a linear border region is assumed between focus (DOI $= 1$) and context (DOI $= 0$), and focus+context visualization is applied to represent features.

In order to reduce visual cluttering, Johansson et al. [66] use high-precision *density maps* [134] in their work on parallel coordinates that count the number of primitives (*density*) that pass through each screen pixel. User-defined *transfer functions* are applied that map density to opacity values in the final output. However, the rendering of the individual primitives into the density map can take up to a few seconds when visualizing large data sets. Novotný and Hauser [106] apply 2D *bin maps* as data representation mechanism in parallel coordinates in combination with outlier detection and clustering within the maps. Depicting the binned information allows the focus+context visualization of large data sets showing data trends while preserving outliers at interactive frame rates.

Konyha et al. [74] introduce *line brushes* to select a subset of function graphs out of a large number of graphs, which intersect with a simple line segment drawn in the view. The *TimeSearcher* [59] is an application especially designed for the visual analysis of time series using *Time Boxes* or *angular query widgets*. The latter are used to select time series which have a similar slope on a sequence of time steps (compare to *angular brushing* [55]). Further extensions of the TimeSearcher [16] allow for *similarity-based querying* of temporal patterns. *QuerySketch* [133] enables the user to outline the shape of a pattern used for querying directly in the view. Inspired by this technique, *QueryLines* [115] allows the graphical specification of approximate

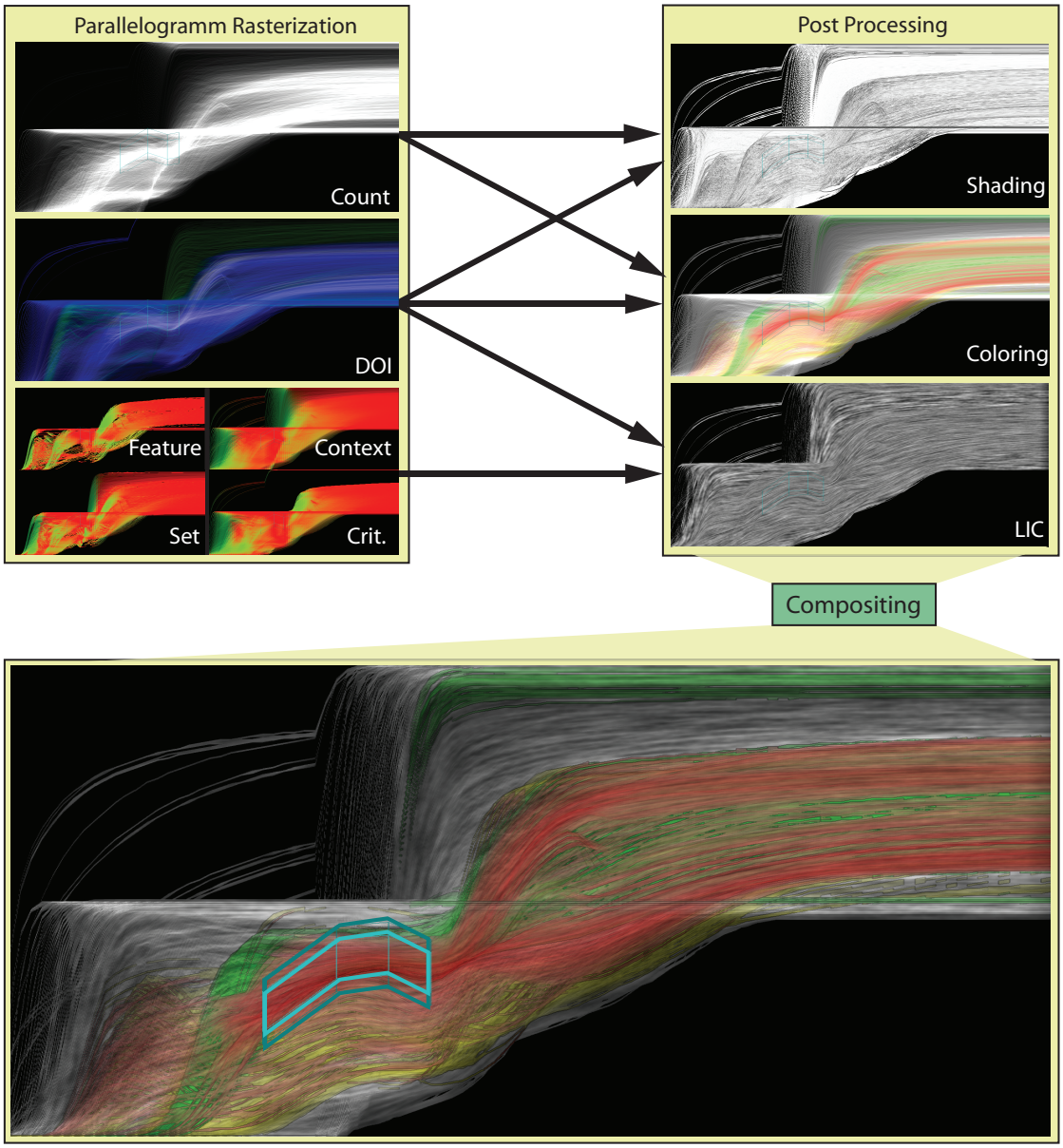


Figure 3.4: The first stage of our rendering pipeline rasterizes parallelograms (each representing multiple function graph segments) and generates multiple output images which are used by the post processing stage. Here different image based algorithms are applied in order to create three different effects (shading, coloring, and LIC), which are combined in the compositing stage.

queries, where soft constraints and preferences are used for fuzzy pattern description and ranking of the query results, respectively.

Our approach combines several of the previously mentioned methods in a novel way and

introduces additional functionality for the definition and visualization of temporal features using multiple linked views.

3.3.2 Four-Levels of Focus+Context

Generally a *feature* within a data set can be defined as a subset of data items which meet multiple criteria specified by the user. The framework, into which our approach has been incorporated, provides different linked views (such as histograms, scatterplots or the proposed function graph visualization) that can be used to specify a DOI function representing one such *feature criterion*. In order to specify multiple features, which themselves are defined by multiple criteria, we use a Feature Definition Language (FDL) tree [31] (see Figure 3.3 (right)). It consists of three levels and is organized as disjunctive normal form. The DOI functions of leaf nodes, the *feature criteria*, are combined into DOI functions representing *features* by applying a fuzzy "AND" operation (all criteria have to be valid for a data item in order to belong to a feature). *Features* themselves are combined into *feature sets* by using a fuzzy "OR" operation.

In order to analyze the influence of the criterion specified within one view on its containing feature and feature set we propose to visualize the respective DOI functions. Figure 3.4 (right) shows an example where CFD simulation data has been used. For each cell temperature is plotted over time. Coloring is used to visualize whether a data item is selected only in the current view (yellow), in the containing feature (red), in the containing feature set (green) or not selected at all (grey). The problem of properly blending these three smooth DOI functions with a context representation is tackled by our four-level compositing scheme (i.e., three focus plus one context level) which is discussed in the context of the proposed function graph visualization in the remainder of this section.

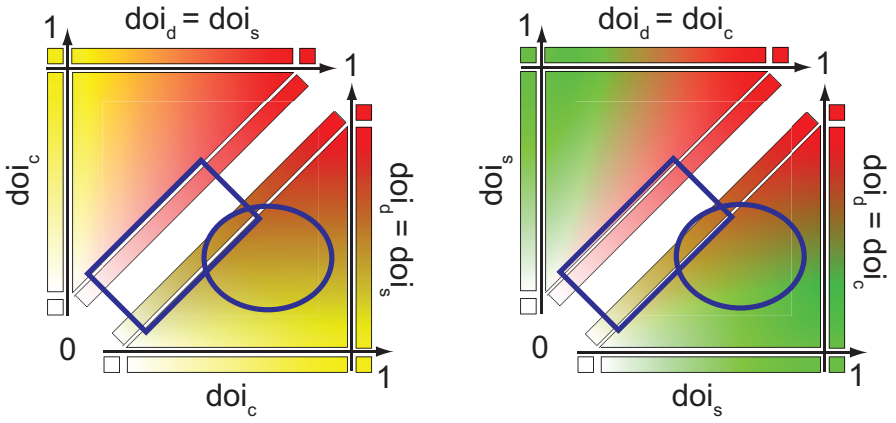


Figure 3.5: Two different color blending approaches (combining red representing doi_d, green doi_s, and yellow doi_c) are illustrated. The top-left triangle of each side shows our compositing scheme using α_s, α_d, and α_c whereas the lower-right parts apply standard alpha compositing using doi_s, doi_d, and doi_c as weights. Note that especially along the diagonal and the borders a brownish tint is introduced by standard alpha compositing because of color mixing.

3.3.2.1 The Four-Level Compositing Scheme

The three different DOI functions which have to be visualized for each data item in the context of a current view are semantically related to each other: The DOI representing a feature criterion (doi_c) directly reflects selections within the current view. No other parts of the FDL tree (related to other views) are considered. The DOI within a feature (doi_d) is always less than or equal to doi_c (since doi_d is an AND combination of doi_c values as shown in Figure 3.3). Contrarily, the DOI of the feature set (doi_s) is always greater or equal than every contained doi_d (due to the OR combination). In order to properly express the contribution of each DOI, we order them based on their influence. The doi_s is the top level collection of individual features defined in their respective doi_d. Thus doi_d is visualized above doi_s. This allows the user to discriminate between the feature containing the current view and all remaining features in the common feature set (compare red and green regions in Figure 3.4 (right)). In order to provide additional information about the feature criterion defined in the current view doi_c is visualized behind doi_d and doi_s.

If colors are blended together to represent the different smooth DOIs it is crucial that as few color mixing as possible occurs in order to allow for straightforward interpretation [13]. Thus simple alpha blending, using the different DOIs as coefficients, has not been used (the color mixing resulting from directly using the DOIs as weights is illustrated in the two lower-right triangles in Figure 3.5). Instead we use repeated alpha blending with the following weights:

$$\alpha_c = \max(\mathrm{doi}_c - \mathrm{doi}_s, 0) \quad \alpha_s = \mathrm{doi}_s - \mathrm{doi}_d \quad \alpha_d = \mathrm{doi}_d \tag{3.1}$$

These weights avoid the introduction of additional tints as shown in the upper-left triangles in Figure 3.5 because they avoid adding colors if they represent only redundant information. For example adding the color assigned to the containing feature set to the color assigned to the current feature is redundant if both DOI functions are equal (no other features contribute to the definition of the feature set). Thus we focus on the difference between doi_s and doi_d which indicates that other features than the current one contribute to the containing feature set.

The order in which the alpha blending is performed is based on the already mentioned influence of each DOI resulting in blending doi_d over doi_s over doi_c over a context representation, each using their respective alpha value (α_c, α_s, α_d). This is illustrated in Figure 3.3 (left) using colors which are applied throughout the remainder of this section.

3.3.2.2 Visualizing Binned Curves

Since the SimVis framework is dealing with relatively large data sets (e.g., 500 K time series) we are using an aggregated data representation to interactively visualize a function over time for each sample in our function graph visualization. As proposed by Novotný and Hauser [106] in the context of parallel coordinates we split the time series data into segments between two consecutive time steps. Each of these segments (see Figure 3.6 left) can be represented by a *frequency binmap* (see Figure 3.6 right) which basically is a 2D histogram. The data values of the two adjacent time steps are used as axes and each bin stores the number of function graphs passing through. It is important to note that the approach proposed by Novotný and Hauser did only use one binmap to store frequency information. Since we have to visualize three additional scalar values we introduce additional *DOI binmaps*. Because the DOI functions can vary over

time it is necessary to use six additional DOI binmaps per segment: two for each of the three DOI functions, representing their aggregated values at the left and the right timestep of the segment.

In order to properly reconstruct the original curves and to apply the proposed four-level focus+context compositing method to coloring and an adapted line integral convolution (LIC) approach, we use the rendering pipeline presented in Figure 3.4.

Parallelogram Rasterization: In this first pipeline stage the frequency representation stored in the individual binmaps is translated back into the time/attribute domain. We rasterize one parallelogram for each bin of the binmaps which contains at least one entry (see Figure 3.6 for the underlying geometry). Since the data from two consecutive time steps is often highly correlated only a small fraction of the bins has to be visualized (for example only 2.1% of the 256×256 bins per time segment have to be displayed for the simulation data set shown in Figure 3.4).

Each such parallelogram has to represent different properties of the underlying frequency data which are accumulated per image pixel by additive blending and multiple floating point precision render targets. These properties include the number of function graphs c within a bin as shown in the sub-figure labeled "count" of Figure 3.4. High luminance indicates a large number of function graphs passing through a pixel. Additional properties are the averaged DOI values multiplied by c in order to account for the fact that each parallelogram represents a different number of function graph segments. The averaged DOI values are fetched from the DOI binmaps and interpolated linearly across the parallelograms. This enables us to visualize the DOI (which often varies over time) for individual sample points along the function graph. Sub-figure "DOI" of Figure 3.4 shows the accumulated DOIs (each color channel representing one of doi_c, doi_d or doi_s).

Besides the frequency and DOI information we propose to include the temporal evolution of the data into our visualization by applying a flow visualization technique (LIC) to a vector field which represents temporal trends. This vector field is created by accumulating the normalized direction vectors $\mathbf{d} = (y_{i+1} - y_i, t_{i+1} - t_i)/\left|(y_{i+1} - y_i, t_{i+1} - t_i)\right|$ of the parallelograms per image pixel. Here y_i and y_{i+1} are the scalar data values represented by the parallelogram at

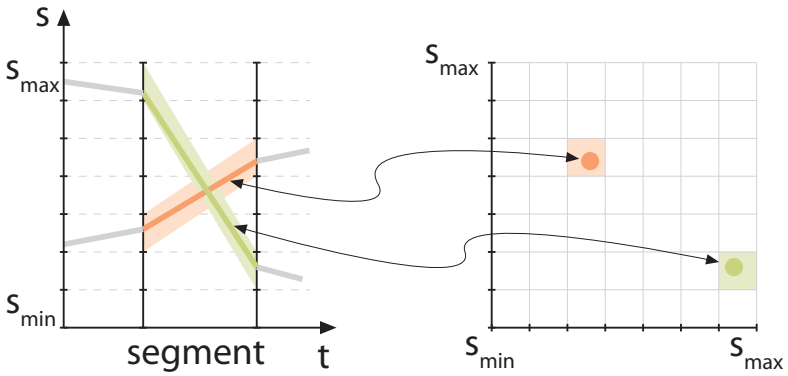


Figure 3.6: This figure illustrates the duality between bins in a binmap and function graph segments in the corresponding function graph space.

timestep i and $i+1$ and t_i and t_{i+1} are the corresponding points in time. As mentioned previously we accumulate $\mathbf{d} \cdot c$ in order to account for the number of function graphs represented by a parallelogram. Since we treat the three different DOI attributes and the context separately we propose to additionally accumulate $\mathbf{d} \cdot \mathrm{doi}_s \cdot c$, $\mathbf{d} \cdot \mathrm{doi}_d \cdot c$, and $\mathbf{d} \cdot \mathrm{doi}_c \cdot c$ as shown in sub-figures "Context", "Set", "Feature", and "Crit." of Figure 3.4. Here the direction is mapped to the red and green color channels. This allows us to blend between these four different vector fields if applying LIC in order to visualize the temporal evolution of the overall data and the selected subsets represented by the DOI functions separately.

The results of this first stage are eight images containing accumulated information about all function graphs: $\mathrm{C}(x, y)$ the number of function graphs through pixel (x, y), $\mathrm{DOI}_s(x, y)$, $\mathrm{DOI}_d(x, y)$, and $\mathrm{DOI}_c(x, y)$ the average DOI of all function graphs, and $\mathbf{D}(x, y)$, $\mathbf{D}_s(x, y)$, $\mathbf{D}_d(x, y)$, and $\mathbf{D}_c(x, y)$ storing the average direction of all function graphs through pixel (x, y) weighted by the respective DOI.

Post Processing: The post-processing stage first performs simple transformations on some images: $\overline{\mathrm{C}}(x, y) = \mathrm{C}(x, y) \cdot s + t$, $\overline{\mathrm{DOI}}_s(x, y) = \mathrm{DOI}_s(x, y)^\gamma$, $\overline{\mathrm{DOI}}_d(x, y) = \mathrm{DOI}_d(x, y)^\gamma$, and $\overline{\mathrm{DOI}}_c(x, y) = \mathrm{DOI}_c(x, y)^\gamma$. The scale factor s and offset t can be used to linearly modify the brightness of the result visualizations whereas γ is used to enhance the contrast of the DOI attributations in order to highlight selected outliers.

Based on these transformed images three different outputs are computed which are weighted and combined in the compositing stage. First the images $\overline{\mathrm{C}}(x, y)$, $\overline{\mathrm{DOI}}_s(x, y)$, $\overline{\mathrm{DOI}}_d(x, y)$, and $\overline{\mathrm{DOI}}_c(x, y)$ are interpreted as height fields. Thus a diffuse lighting contribution can be computed using a directional light source above the height fields. The first output image, as depicted in Figure 3.7(a), represents the minimum of all four lighting contributions. This is similar to basic edge enhancement which allows the user to recognize sharp boundaries of trends in the four underlying images.

The consecutive outputs are based on our four-level focus+context compositing scheme. Figure 3.7(b) shows the result of applying the compositing to four color values as already indicated in Figure 3.3. Additionally the result color is modulated by $\overline{\mathrm{C}}(x, y)$ in order to account for function graph density. The final output shown in Figure 3.7(c) is also based on our compositing approach. But instead of compositing colors on a pixel per pixel basis we blend the different direction fields $\mathbf{D}(x, y)$, $\mathbf{D}_s(x, y)$, $\mathbf{D}_d(x, y)$, and $\mathbf{D}_c(x, y)$ using the corresponding (transformed) DOI images. This synthetic vector field represents the overall evolution of all function graphs over time with special focus on the three DOI layers, thus trends in selected regions always override trends visible in the context. In order to visualize this field we apply line integral convolution by sampling from a low and a high frequency noise texture along Euler interpolated streamlets. The final values which are accumulated along the streamlets are a linear combination of the high and low frequency samples (s_h and s_l) based on $\lambda = \max(\overline{\mathrm{DOI}}_s(x, y), \overline{\mathrm{DOI}}_d(x, y), \overline{\mathrm{DOI}}_c(x, y))$:

$$s = s_h \cdot \lambda + s_l(1 - \lambda) \tag{3.2}$$

Thus regions with high DOI values are displayed using finer noise whereas low DOI regions are represented by coarser noise as shown in Figure 3.7(c).

Compositing: As implied by its name this step combines the three images (shading i_s, coloring i_c, and LIC i_l) based on three weights (w_s, w_c, and w_l respectively) resulting in a visualization as shown in Figure 3.4. The blending is performed according to this equation:

$$(i_s w_s + (1 - w_s)) \cdot (i_c w_c + (1 - w_c)) \cdot (i_l w_l + (1 - w_l)) \tag{3.3}$$

We chose to modulate all images with each other in order to retain a maximum amount of contrast whereas the different weights allow for an easy adjustment of the influence of single post processing effects. Since only the contribution from i_c contains color information (i_s and i_l contain luminance values) the resulting color is just attenuated (not distorted). The result images in this section have been generated with settings of $w_s = 0.7$, $w_c = 1$, and $w_l = 0.7$.

3.3.2.3 Brushing

When analyzing a large number of function graphs it is often necessary to compare them against a certain pattern. For example, the user may be interested in functions showing a sudden rise at some point in time with subsequent stabilization. Such queries can to some degree be approximated by combining multiple vertical interval selections at different positions within the domain (as also supported by our framework). However, brushing function graphs by roughly outlining a target function and evaluating the distances proved to be faster and more intuitive for many tasks and offers more options like considering average distances or ignoring vertical offsets. Basically, various ways of defining the target function are conceivable (e.g., analytical). In our case, the target is specified by linearly interpolating between an arbitrary number of control points (t_i, y_i) with t_i being a discrete time-step and y_i the corresponding value in the data domain. After defining the target function, the basic steps generally necessary for evaluating the brush are a transformation of the sampled raw data, an aggregation of differences, and finally applying one (or two for smooth brushing) threshold(s) to obtain the degree-of-interest values.

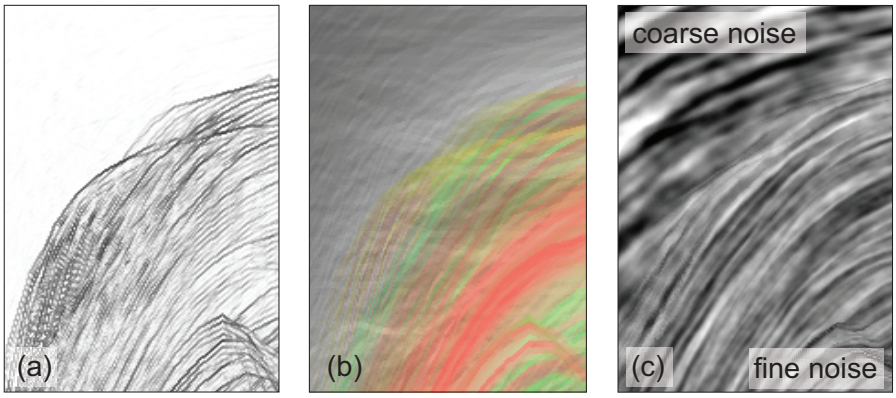


Figure 3.7: Cutout images from Figure 3.4. Shading (a) is used to emphasize "edges" in the DOI and frequency images whereas the application of our compositing scheme translates the three different DOI attributations into easily recognizable colors (b). Additionally LIC is applied to a flow field representing the average function graph directions (c).

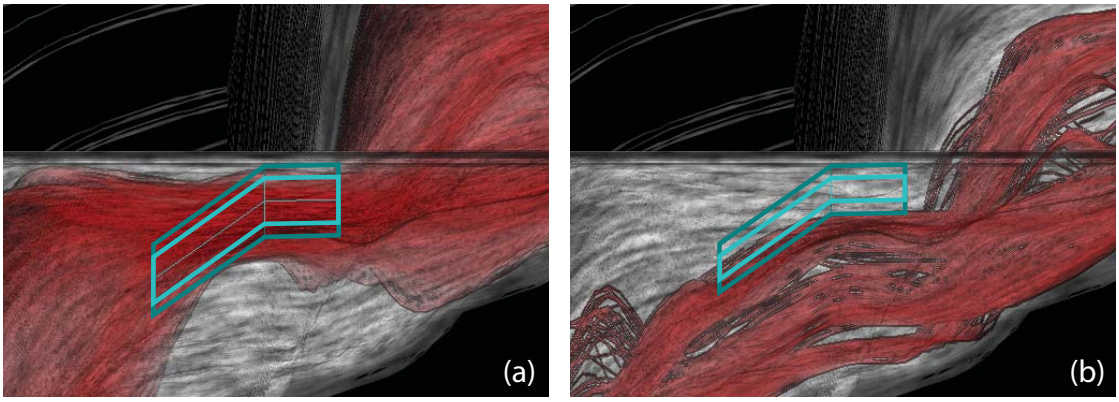


Figure 3.8: This figure shows brushing results based on two different similarity measures. In (a) no data transformation is used whereas in (b) a gradient based distance is evaluated which is invariant to vertical translation.

As an (optional) first step, different transformations may be applied prior to evaluation. Without transformation, similarities are computed on the raw data values with respect to the absolute position of the target function (see Figure 3.8(a)). However, it is often desirable to account for task- and data-specific issues like offset translation, magnitude scaling, linear trends removal or noise reduction [71]. Disregarding constant offsets is particularly necessary in many cases, as the computed distances can otherwise become meaningless for measuring (subjective) similarity [71]. We propose two different ways of accounting for vertical offset translation (horizontal offsets are prohibitively expensive to determine for interactively brushing up to millions of function graphs): The first approach, shown in Figure 3.8(b), is based on computing the first derivative of the target function and all function graphs of the data set, i.e., considering gradients rather than the raw data values themselves (e.g., estimated by forward differences or central differences). While this is comparatively fast because the transformation and the subsequent aggregation can be done within one pass through the data, it is susceptible to noise. To account for this, the second approach seeks to align two functions on each other before comparing them by subtracting the average Y-value within the common interval $[T_1, T_2]$. For a function $f(t)$, this value is given by $(\int_{T_1}^{T_2} f(t)dt)/(T_2 - T_1)$. This approach is more stable, but may still be distorted by very strong "outliers" within the sample points and needs an own pass for computing the average value per function graph.

As the next step, the difference function $d(t) = |\hat{f}(t) - \hat{g}(t)|$ is computed for each transformed function graph $\hat{f}(t)$ with $\hat{g}(t)$ denoting the transformed target function and aggregated within the common interval $[T_1, T_2]$. Among others, useful aggregations are the absolute maximum of $d(t)$ and the average distance $(\int_{T_1}^{T_2} d(t)dt)/(T_2 - T_1)$. Note that distances computed this way have the same scaling as the Y range and are independent of the scaling of the time domain. For function graphs given by discrete, unequally sampled points, it is important to weight the distances by the size of the corresponding sample intervals when summing them up.

The final step is computing degree of interest (DOI) values from the distances by applying one (for binary brushing) or two (for smooth brushing) threshold(s). The latter case as used in

our approach is illustrated in Figure 3.9. As for the distance values, thresholds are specified in the value range of the function.

Interaction is very important for function-similarity brushes: After defining a target function by setting initial control points, further points can be added, existing ones can be modified or removed, and it is also possible to move the whole function. Adapting the thresholds is usually necessary and conveys a feeling at which gradient the similarity is changing for various regions. The evaluation is executed in parallel to interaction (i.e., threaded), which prevents disturbing delays and provides visual feedback as soon as possible.

3.3.3 Case Study Perfusion Data

Perfusion data are dynamic medical image data which characterize the regional blood flow in human tissue. They are acquired for example for tumor detection and classification in the female breast or to diagnose an acute ischemic stroke. We discuss two application examples based on data sets from Magnetic Resonance (MR) perfusion imaging. The distribution of contrast agents (CA) is registered to assess tissue kinetics (breast tumor diagnosis) and blood flow (ischemic stroke diagnosis). The CA is injected intravenously and its distribution is measured by a repeated acquisition of subsequent images covering the volume of interest. The CA provides signal changes in the acquired 4D data (3D+time). For each image voxel, a time-intensity curve (TIC) characterizes the CA enhancement (see Figure 3.10). Clinicians are trained to infer tissue characteristics from the shape of these curves. The function graph visualization presented here allows the clinician to define a curve pattern and a similarity measure. Latter is applied to match the original TICs with this pattern. To our knowledge, this represents a new approach for the analysis of perfusion data. Existing approaches base the feature detection either on percental enhancement between two time-steps [26], or on so-called perfusion parameters derived from the TICs [108]. For more detailed information regarding this application examples and perfusion imaging in general see work by Oeltze [107].

3.3.3.1 Breast Tumor Diagnosis

The process of CA enhancement in a tumor can be described by the diffusion of tracer particles from the inside of blood vessels into the extravascular space and vice versa before it becomes ex-

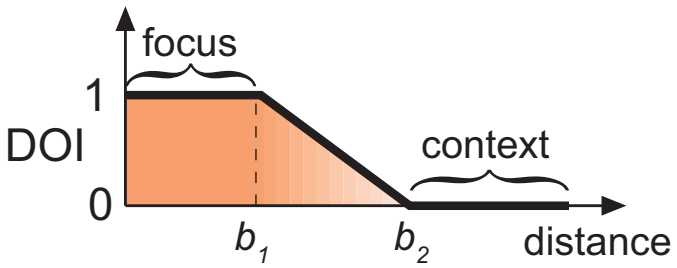


Figure 3.9: Computing smooth DOI values based on distance values by linearly interpolating between two thresholds.

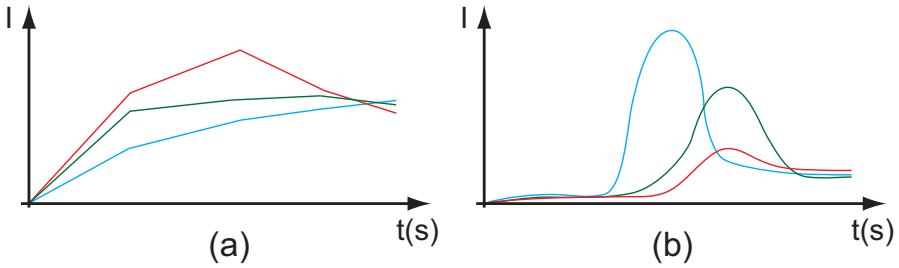


Figure 3.10: (a) Time-intensity curves (TICs) typical for contrast agent accumulation in breast tissue. The continuous gentle inclination of the blue curve is characteristic for healthy tissue. The green and red curves are suspicious due to their high early enhancement. The red curve is especially suspicious because of its subsequent rapid wash-out, which is typical for malignant tumors. (b) TICs typical for contrast agent accumulation in the gray matter of the brain. The blue curve shows normal brain perfusion whereas the green and red curves show decreased and delayed perfusion around an infarction.

creted in the kidneys. The permeability of the vessel walls and the extracellular volume fraction determine the amplitude and the shape of the TIC. TICs which show a high early enhancement (wash-in) followed by a rapid wash-out, i.e., a significant decrease of signal intensity afterwards, are especially suspicious (see Figure 3.10(a), red curve). Less suspicious are curves showing a plateau later on (green curve), or regions that continue to enhance (blue curve). This is typically observed in benign tumors. The major diagnostic task is to confirm or reject the hypothesis of a tumor being malignant. For more details on tumor perfusion, see Heywang-Köbrunner et al. [58].

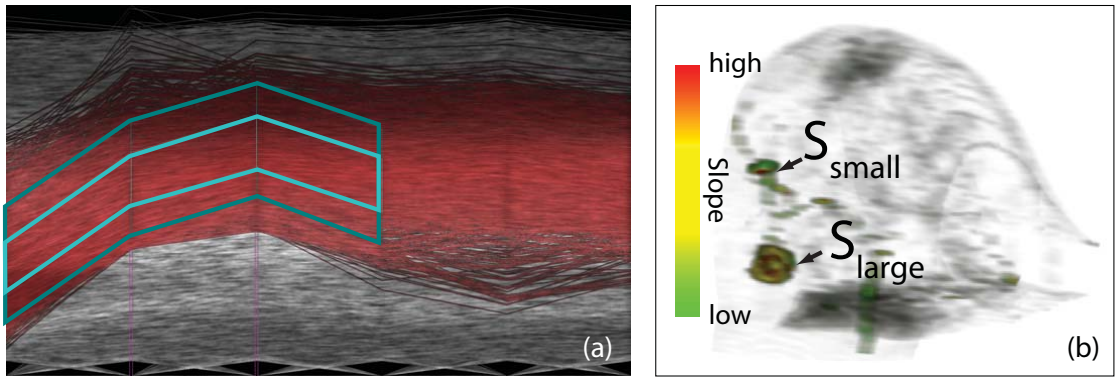


Figure 3.11: Detection of suspicious structures in breast tumor diagnosis. In our function graph visualization (a), a smooth similarity brush (blue) has been defined that resembles a curve pattern which is typical for malignant tumors. The resulting selection is visualized in the context of the entire right mamma in (b). Two suspicious structures S_{large} and S_{small} are revealed. The color indicates the accumulation speed during the early phase of the contrast agent passage (Slope).

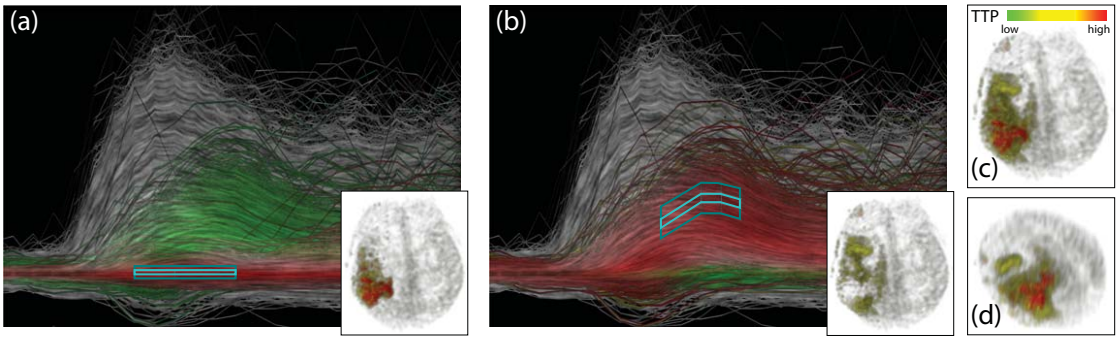


Figure 3.12: A feature set containing two different features has been defined in two function graph visualizations. In (a), a smooth similarity brush (blue) has been defined such that tissue with no significant signal enhancement is selected in the first feature (shown in its 3D context in the inlet). In (b), a similarity brush has been used to specify a second feature representing tissue exhibiting reduced and delayed perfusion. Candidate areas of this tissue at risk are revealed around the infarction core (inset). In (c) and (d) the overall feature set is visualized. The color is modified according to the time it takes until the maximum amount of contrast agent is delivered for a particular voxel (TTP, time to peak).

The data set analyzed in the following was acquired to examine a suspicious region in the right mamma which had been detected during conventional mammography. As a pre-processing step, the original data has been cropped to restrict the analysis to relevant breast tissue. The resulting data set matrix is: 458 x 204, slice distance: 3 mm, number of slices: 26 (overall 2.429 K voxels), temporal resolution: 6 measurements in 10 min. According to the accompanying diagnostic report, the patient suffered from a probably malignant structure (S_{large}) close to the thoracic wall and a smaller satellite lesion (S_{small}). The detection of the two structures and the verification of the reported findings are illustrated in Figure 3.11. In order to select data items exhibiting a temporal pattern typical for malignant tissue a similarity brush using our gradient based measure is defined in a function graph view (a). As a result of this feature definition, S_{large} and S_{small} are revealed in a linked 3D visualization (b). The shape of the breast is indicated as context information.

3.3.3.2 Ischemic Stroke Diagnosis

In contrast to leaky vessels in malignant tumors, micro-vessels in normal brain tissue do not leak as a result of the blood brain barrier. Consequently, there is no enhancement in the extracellular volume. Instead, we observe the "first-pass" of the CA through the vessel components. In case of an ischemic stroke, the existence and the extent of "tissue at risk" surrounding the core of the stroke has to be evaluated immediately after the infarction. "Tissue at risk" is characterized by reduced and delayed perfusion (Figure 3.10(b)). Surgical and chemical interventions may salvage at least parts of the "tissue at risk" [27].

The second data set has been acquired immediately after an acute stroke. The data set matrix is: 128 x 128, slice distance: 7 mm, number of slices: 12, temporal resolution: 40 measurements

in 40 sec. According to the accompanying diagnostic report, the patient suffered from an infarction in the right hemisphere (which appears left in each view of Figure 3.12). The detection of the infarction core and the surrounding "tissue at risk" is illustrated in Figure 3.12. We have specified a feature set containing two features, the infarction core (a) and "tissue at risk" (b), by using two different function graph views and similarity brushes. Each view shows locally selected data (red) in combination with curves selected in the overall feature set (green). This allows for easy comparison of both features. Our LIC based approach helps to convey general temporal trends in the context (grey) as well as in the selected regions (colored). The insets of (a) and (b) each show the respective feature in their spatial context. The color is modified according to the time it takes until the maximum amount of CA is delivered for a particular voxel, the so-called Time to Peak (TTP). Candidate areas for "tissue at risk" appear yellowish (medium TTP values) whereas the infarction core is colored red. In order to visualize the extent of the infarction, the overall feature set is shown in (c) and (d).

Our observations in this section could be successfully validated with results from Oeltze et al. [108] where the same data sets have been examined.

3.3.4 Summary and Future Work

We have presented a highly scalable approach which effectively handles hundreds of thousands or even millions of function graphs interactively. A novel four-level focus+context method is used to blend both color and directional information according to the degree of interest specified in the various layers. The directional information is derived from the average orientation of function graphs through each pixel allowing for an effective visualization (by applying LIC) of large trends for the whole data set as well as in regions selected by the user. Note that the averaging process that is used to create the vector field can result in misleading visualizations in regions where many lines are crossing at an obtuse angle. The technique presented in Chapter 3.4 can be used to address this problem. The four-level coloring-scheme has been applied to all information visualization views that are provided by the SimVis framework and therefore provides a consistent linking mechanism. This allows the user to relate features across multiple views such as scatterplots, histograms, parallel coordinates, and the time series visualization discussed in this section.

We propose different smooth brushing techniques allowing for fast and intuitive selection of temporal trends based on user definable target functions. Additionally shading is used to indicate strong variation in the function graph density. In order to show the usefulness of our method, it has been successfully applied to medical perfusion data.

For future work we would like to abandon the necessity to explicitly rasterize a parallelogram for each bin within a binmap. Instead we plan to utilize the duality between a binmap and its corresponding segment by sampling along a line in the binmap to accumulate all bins which affect one pixel.

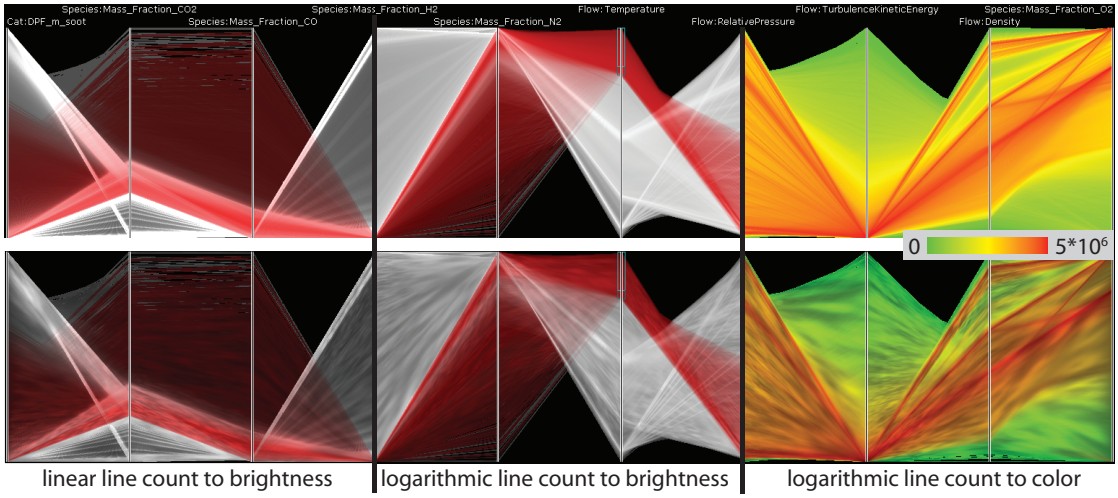


Figure 3.13: Comparison between parallel-coordinates plots where line density is linearly/logarithmically mapped to brightness (first/second column), and color (third column). Each bottom-row image has been generated by applying our technique to the corresponding top-row plot. In the first and second columns, color is used for distinguishing between focus (red) and context (grey-scale). Regions where line density does not vary strongly are improved most visibly. The coherent noise texture employed by our method provides important information on line orientations that is otherwise lost.

3.4 Visual Coherence for Large-Scale Line-Plot Visualizations

This section is based on the paper *Visual Coherence for Large-Scale Line-Plot Visualizations* [102]. Lines are an integral part of many visualization methods. However, in most cases the number of lines that is rendered is directly proportional to the number of displayed data items. For this reason, such line-based approaches quickly suffer from cluttering and overdraw when large data sets have to be visualized. Standard line rasterization decomposes each line into individual pixels without orientation information. Thus, it can at most yield information about the line density per pixel, e.g., by using additive blending. We propose to generate additional data about line orientation on a per-fragment basis, which can be used to enhance line-based visualizations. In Section 3.3 a vector field has been used to store averaged line orientations per pixel. However, this is not sufficient for many visualization applications, as a vector field does not contain information about the overall distribution of line orientations. For example, distinguishing image regions where all lines are parallel from regions where many lines are crossing is impossible when only a vector field is used. In contrast, the technique discussed in this section relies on a 2x2 tensor field that is similar to structure tensor fields used in image processing. In both cases, tensors are symmetric positive definite, and the eigenvector corresponding to the largest eigenvalue encodes directional information per pixel. In our approach, this eigenvector corresponds to the most dominant line orientation through a pixel, whereas in a structure tensor it represents the dominant image-gradient direction. Additionally, the relation between eigenvalues λ_1 and

λ_2 provides further information that is complementary to the dominant line orientation. We exploit this property to solve the problem of distinguishing regions comprising parallel lines from regions with more evenly distributed line orientations. Furthermore, we construct the tensors representing the line orientations in such a way that they can be used directly as diffusion tensors. We then apply anisotropic diffusion to a white noise texture, driven by the directional information encoded in the tensor field. The resulting texture is spatially coherent in regions where the orientations of many lines are well aligned. Complementarily, the more an area comprises isotropically distributed line orientations, the more the resulting texture approaches isotropically blurred noise. The latter would also result from filtering with a Gaussian kernel. However, in isotropic regions the diffused noise texture does not convey additional information. We therefore adapt the influence of the noise texture on the visualization according to a local measure of anisotropy. Figure 3.13 compares parallel-coordinates without (top) and with (bottom) the technique proposed in this chapter. Line density is indicated by brightness/color. Especially in regions of uniform line density our approach provides pronounced visual cues that convey general line orientation very well.

Furthermore, many visualization techniques rely on a focus+context rendering approach. Data selected/highlighted by the user are displayed prominently as focus, whereas the remainder of the data set is represented in a less obtrusive style to provide context. In order to properly support this differentiation between focus and context, we propose to use two separate tensor fields. One field represents line orientations for the overall data set (i.e., the context), and one field stores line orientations weighted by a selection function in $[0, 1]$ (i.e., the focus). This function is zero for unselected, and one for selected data, respectively. Anisotropic diffusion is either performed for a tensor field that is generated by combining the respective fields for focus and context, or for both tensor fields independently (see Section 3.4.3.5).

Since individual features can exist at different spatial scales, choosing just one noise frequency for the whole image plane is often not sufficient. Therefore, we propose to utilize a multi-resolution approach which builds on an image pyramid of the tensor field. Anisotropic diffusion is performed for each level of this pyramid. The input-noise frequency is adapted to the local scale of the tensor field. This results in a second image pyramid containing smeared noise textures which correspond to the different resolution levels of the original tensor field. A final image compositing step is used to blend multiple noise levels. Coarser noise is applied to homogeneous regions, whereas fine-grained noise is used in image areas exhibiting small-scale detail.

To summarize, the contributions of this section are:

- A generic image-space method that can enhance any visualization that draws many overlapping lines in 2D.
- Visual coherence via anisotropic diffusion of noise, driven by a 2x2 tensor field that subsumes line orientations.
- Combined visualization of features at different scales.
- Support for focus+context rendering.
- Interactive performance with straightforward implementation and integration into existing code.

3.4.1 Related Work

The technique proposed in this work is quite general and thus can be applied to a large class of visualization problems that render many lines. One of the most prominent examples is the parallel coordinates technique, which represents each data item by one polyline [62]. For large data sets, the main problem of this technique is cluttering in image space due to line overdraw. Fua et al. [44] propose to perform hierarchical clustering on the data in order to solve this problem. A different approach employing image-based clustering has been proposed by Novotny and Hauser [106]. Instead of clustering entire polylines, only line segments between coordinate axes are grouped together in order to greatly speed up rendering. Outlier detection is also performed for line segments and not for entire polylines. McDonnell and Mueller [98] utilize different illustrative techniques in combination with edge-bundling and clustering. Edge-bundling is also used in the context of graph visualization via link-node diagrams [60, 61], which could also be enhanced by our technique. Contrary to these approaches, Johansson et al. [65, 66] do not rely on a priori clustering of the data. Instead, a floating point texture is used to accumulate line density data. Transfer functions are then applied in order to reduce cluttering. Figure 3.13 shows how our technique can be combined with this approach. Sophisticated sampling strategies have also been used to reduce overdrawing [38]. Ellis and Dix [40] provide a more comprehensive classification of clutter reduction techniques.

Tensor fields are most commonly visualized by mapping tensor properties to glyph shapes such as super quadrics [73]. Depending on the properties of the tensor field, more complex shapes/approaches can be used [136]. Directionally-blurred noise has first been used in the context of flow visualization. Van Wijk uses locally deformed spots [130], whereas Cabral and Leedom [18] integrate white noise along integral lines of the flow field. Non-linear anisotropic diffusion is most commonly applied in image processing [135], but has also been utilized for dense flow visualization [28].

The notion of a continuous scale space has been introduced by Witkin [145]. This concept of analyzing data on different scales has been used in many different fields in image processing as well as in visualization. Examples are information-visualization approaches relying on hierarchical clustering [44], or flow-visualization methods based on a multi-resolution representation of the underlying flow field [126]. Rasterizing discrete lines for data items which are samples of a continuous data domain is not always sufficient. Thus, Bachthaler and Weiskopf [7] propose a continuous data representation for scatterplots, which has also been extended to parallel coordinates [56]. Even though no actual lines are drawn by this technique, data-density information is still transferred between coordinate axes along certain trajectories. For example, for two perfectly correlated attributes, these trajectories would all be horizontal lines between the two corresponding axes. Therefore, the tensor field driving our method could also be generated for the continuous case.

3.4.2 Dense Encoding of Line Orientations

Standard line rasterization can at most provide line-density information per pixel. Let us consider computing a line density function $C(x, y) \in \mathbb{N}$ during rasterization. This gives the number of lines crossing the pixel at position (x, y), but discards all information on line orientation. In

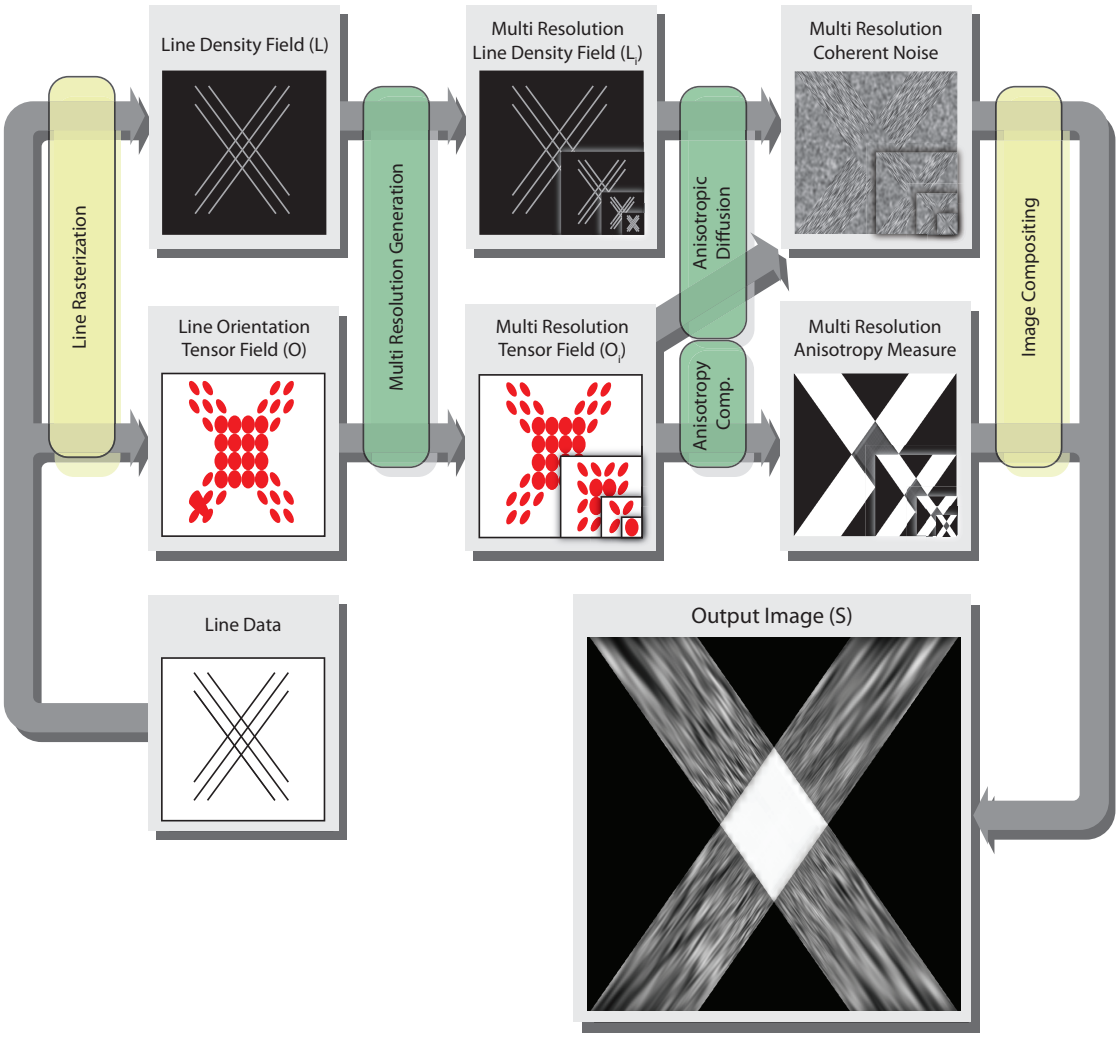


Figure 3.14: The rendering pipeline that we use to synthesize the final coherent noise texture. Except for the line rasterization stage itself, all operations are image-based. This makes the technique scalable to very large data sizes. The steps indicated in yellow can be used to merge input from multiple subsets of the data set in order to accommodate focus+context visualizations.

principle, the gradient ∇C can be used to reconstruct some directional information in image space. Consider an image with a single rasterized line. The gradient at the edge of the line is always perpendicular to the line, and can thus be used to obtain information about line orientation without actually requiring the original line. As soon as line density increases, however, overdraw resulting from crossing or overlapping lines rapidly reduces the accuracy of image-space gradients. In regions of constant $C(x, y)$, no orientation can be reconstructed at all. In order to solve these problems, we generate additional data during rasterization. A 2D vector field $V(x, y) \in \mathbb{R}^2$ can be used to store the averaged line orientation for the pixel at position (x, y). On GPUs, such a field can be generated easily by calculating multiple output values per fragment

during rasterization. These can be accumulated per pixel and renormalized in a subsequent rendering pass. However, a vector field lacks information about the coherence of line orientations through the pixel. No information about the actual distribution of line orientations is retained. Averaging different orientations may result in the same average. In this case, it is impossible to distinguish a pixel through which parallel lines are passing from one that is crossed by lines of varying orientations. Treating both cases equally can lead to misleading visualizations. Instead of averaging vectors, we propose to employ symmetric positive-definite 2x2 tensors. This is somewhat similar to the use of structure tensors in image processing. The structure tensor for a 2D image I and a 2D windowing function w is defined as

$$S_w(\mathbf{p}) = \sum_{\mathbf{u}} \left(w[\mathbf{p}-\mathbf{u}] \begin{bmatrix} I_x(\mathbf{u})^2 & I_x(\mathbf{u})I_y(\mathbf{u}) \\ I_x(\mathbf{u})I_y(\mathbf{u}) & I_y(\mathbf{u})^2 \end{bmatrix} \right), \tag{3.4}$$

where $\mathbf{u}$ covers a neighborhood of $\mathbf{p}$. I_x and I_y denote the partial derivatives of the image in x and y, respectively. The eigenvector of $S_w(\mathbf{p})$ corresponding to the largest eigenvalue λ_1 represents the predominant gradient orientation in a local neighborhood of $\mathbf{p}$. This neighborhood is defined by the windowing function w. The degree of anisotropy of the gradient in a region (i.e., how unevenly gradient orientations are distributed) can be expressed by the relative anisotropy:

$$f_r = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}. \tag{3.5}$$

If the gradient within the windowed region is constant, then the smaller eigenvalue $\lambda_2 = 0$, and hence $f_r = 1$. If the gradient orientations are distributed uniformly (isotropically), $\lambda_1 = \lambda_2$, and thus $f_r = 0$.

We exploit these properties to encode the orientations of lines passing through a pixel $\mathbf{p}$, instead of the gradient orientations in the neighborhood of the pixel. Instead of using a windowing function to define a region for which directional information is accumulated, we sum over all normalized orientations $\mathbf{v} = (v_x, v_y)$ of lines passing through a pixel $\mathbf{p}$, with

$$O(\mathbf{p}) := \sum \mathbf{v}^{\mathrm{T}}\mathbf{v} = \begin{bmatrix} \sum v_x^2 & \sum v_x v_y \\ \sum v_x v_y & \sum v_y^2 \end{bmatrix} \tag{3.6}$$

defining the line-orientation tensor-field O. Analogously to structure tensors, the eigenvector corresponding to λ_1 represents the predominant orientation of all lines passing through $\mathbf{p}$. Likewise, the anisotropy measure f_r can be used to obtain information about the overall distribution of orientations. The tensor field O can be computed efficiently in a GPU fragment shader during line rasterization, computing $O(\mathbf{p})$ for each fragment covering a pixel $\mathbf{p}$. This approach requires almost no change of the underlying visualization method determining what and where lines should be drawn.

3.4.3 Visually Coherent Noise for Line Plots

Our method builds on the construction of the dense line-orientation tensor-field O described above, in order to produce coherent line visualizations in image space. Figure 3.14 provides an overview of the overall rendering pipeline.

During line rasterization, we compute the line orientation tensor $O(\mathbf{p})$ for each pixel $\mathbf{p}$, as well as a scalar line density $L(\mathbf{p})$, by summing up the contribution of each line passing through the pixel. Additionally, in order to allow for focus+context techniques, multiple tensor fields representing different subsets of the data can be generated. The two stages highlighted in yellow in Figure 3.14 can be used to integrate the data in the focus with the data in the context for a cohesive focus+context visualization (Section 3.4.3.5). The coherent noise texture which is used to indicate predominant line orientations in a dense way is synthesized via non-linear anisotropic diffusion. In order to do this, $O(\mathbf{p})/L(\mathbf{p})$ is directly used as the diffusion tensor (Section 3.4.3.2). In order to represent features at different spatial scales this diffusion process is performed at multiple image resolutions (Section 3.4.3.3). The output image is generated in a final compositing step that uses the coherent noise textures and an anisotropy measure of $O(\mathbf{p})/L(\mathbf{p})$ as input (Section 3.4.3.4).

3.4.3.1 Line Rasterization

The primary goal of the line-rasterization stage is the computation of $O(\mathbf{p})$ and $L(\mathbf{p})$ for each pixel $\mathbf{p}$. The contribution to all pixels covered by a single line between points $\mathbf{p_0}$ and $\mathbf{p_1}$ is equal to $\mathbf{v}^{\mathrm{T}}\mathbf{v}$ with $\mathbf{v} = (\mathbf{p_1} - \mathbf{p_0})/|\mathbf{p_1} - \mathbf{p_0}|$. Additive blending into a floating point render target can be used to perform the component-wise summation of each line's contribution to O and L. In total, only four floating point values per pixel are required to store both fields: three floats for the symmetric 2x2 tensor $O(\mathbf{p})$, and one float for the number of lines $L(\mathbf{p})$ that contribute to the pixel. A single four-component floating point render target is sufficient to store all information required for coherent line visualization. Current GPUs support simultaneously writing to multiple render targets. This makes it straightforward to adapt existing visualization methods to generate both fields without requiring additional rendering passes. Furthermore, this approach can easily be combined with techniques that cluster lines and draw only one representative line per cluster. In this case, the contribution of each representative line must simply be weighted according to the number of lines in the cluster. For n lines in a cluster, $n\mathbf{v}^{\mathrm{T}}\mathbf{v}$ has to be added to $O(\mathbf{p})$ at every pixel, and $L(\mathbf{p})$ must be increased by n instead of by 1.

3.4.3.2 Line-Orientation Driven Diffusion

After the fields O and L have been generated as described above, they can be used to visualize the encoded orientations in a coherent manner. It is important that this is done in a non-intrusive way that can be added easily to existing visualization techniques, while retaining their overall properties. For this reason, instead of using tensor visualization techniques such as glyphs, we employ dense noise textures that result from performing anisotropic diffusion of white noise. The resulting noise textures are straightforward to integrate into existing visualizations in image space by using them to modulate brightness per pixel.

Diffusion is the process of equilibrating concentrations within a spatial domain without generating or destroying mass. The flux j that corresponds to the direction along which mass is transported by the diffusion process is defined by Fick's Law:

$$j = -D \cdot \nabla u \tag{3.7}$$

Here, $u = u(\mathbf{x}, t)$ is a scalar function that defines the temporal evolution of concentrations over the spatial domain. In our case, $u \in [0, 1]$ is the pixel intensity. D denotes the symmetric positive definite diffusion tensor, which is used to steer the strength and direction of the flux j. When the eigenvalues of D are equal (i.e., $\lambda_1 = \lambda_2$), the diffusion that results from the flux is isotropic, because then j is parallel to ∇u. Anisotropic diffusion results when $\lambda_1 \neq \lambda_2$. In the extreme case of $\lambda_2 = 0$, flux parallel to the corresponding eigenvector is not permitted at all. Thus, the diffusion tensor locally steers the directional properties of the diffusion process. The diffusion can be formulated as a partial differential equation over time [135]:

$$\partial_t u = \mathrm{div}(D \cdot \nabla u) \tag{3.8}$$

This means that the change of concentration over time is equal to the negative divergence of the flux. As proposed in the context of flow visualization [28], this formulation can be used to create a dense visualization of directional information by diffusing a white noise texture and carefully selecting the diffusion tensor D. For a vector field V, D can be constructed such that the eigenvector corresponding to the largest eigenvalue is always parallel to V. The remaining degrees of freedom can be used to encode $|V|$. In our method, the normalized tensor field O/L has already been created with diffusion in mind, and can thus be used directly as the diffusion tensor, i.e., we use $D = O/L$. The normalization by L decouples diffusion strength from line density. It restricts the values of λ_1 and λ_2 to the unit interval $[0, 1]$, regardless of the number of lines contributing to a given pixel. Without this normalization, the diffusion in regions with a higher line density would be stronger than in sparse image regions. Even if this effect would be desirable, a diffusion tensor with large eigenvalues generates large fluxes, which would require very small time steps for solving Equation 3.8.

Another important consideration is that diffusion generally reduces image contrast over time. Diewald et al. [28] proposed to introduce a contrast enhancement function f. This function pushes pixels in the range $[0.0, 0.5)$ toward 0.0, and pixels in the range $[0.5, 1.0]$ toward 1.0. This can be incorporated into the diffusion equation as

$$\partial_t u = f(u) + \mathrm{div}(D \cdot \nabla u) \tag{3.9}$$

The contrast function that we use in our method is

$$f(u) := \beta \cdot (-(2u - 1)^3 + 2u - 1), \tag{3.10}$$

where $\beta \in [0, 1]$ represents a scalar that is defined by the user and controls the strength of contrast enhancement. Figure 3.15 shows how different settings for β influence the diffusion.

The coherent noise texture $u(\mathbf{x}, t)$ is computed for time t by performing explicit integration of Equation 3.9, evaluating

$$u_i = u_{i-1} + \Delta_t \left(f(u_{i-1}) + \mathrm{div}(D \cdot \nabla u_{i-1})\right) \tag{3.11}$$

for a desired number of iterations. The initial image u_0 contains black-and-white noise. The two major parameters that influence the result of the diffusion are the number of iterations, and the contrast enhancement parameter β. More iterations result in longer coherent structures, which is illustrated in Figure 3.15. All results in this section have been created using between 128 to 256 iterations of Equation 3.11. We use a second-order discretization of $\mathrm{div}(D \cdot \nabla u_{i-1})$, [135] (page 95), with $\Delta_t = 10^{-3}$ and pixel sizes of 1x1.

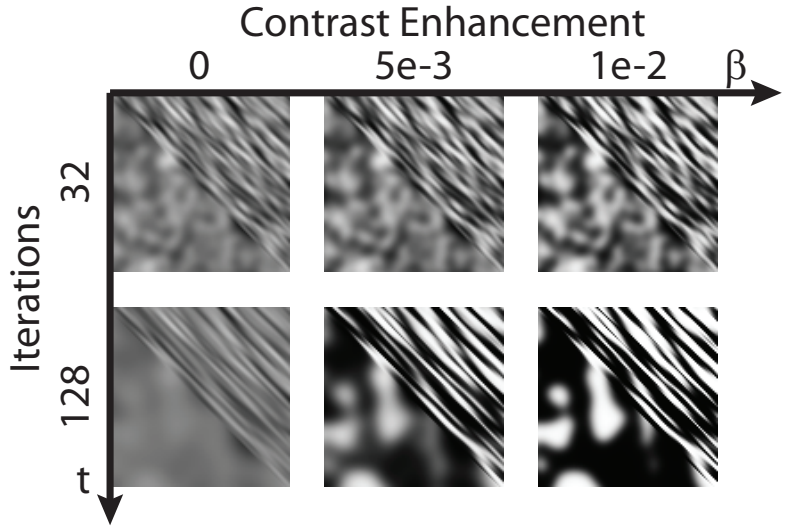


Figure 3.15: Comparison of the influence of the contrast enhancement parameter β, and the number of iterations for diffusion, respectively. A small number of iterations requires higher values of β in order to generate comparable image contrast. The reason for this is that the influence of the contrast enhancement function f increases over time.

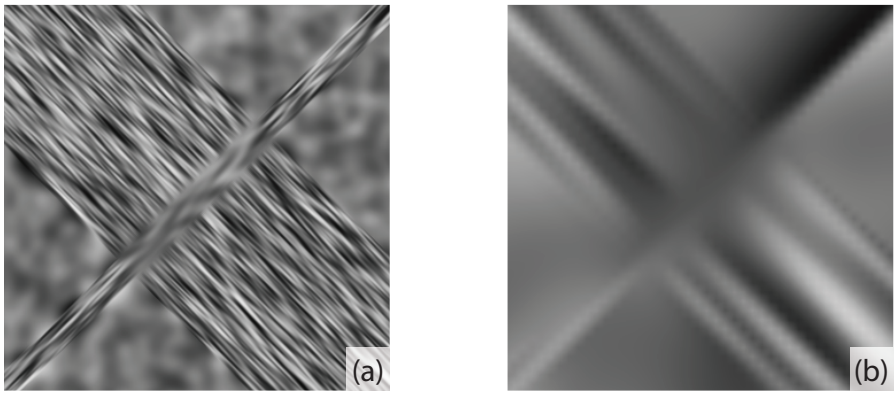


Figure 3.16: Two different resolution levels of a coherent noise texture synthesized with our technique. The input were two sets of crossing lines. The set of lines that are packed tightly together is represented well by the high-frequency noise (a), whereas the set of lines spread out more broadly is also represented by low-frequency noise (b).

3.4.3.3 Visualizing Features of Different Scales

Multiple directionally-blurred noise textures to visualize features of different spatial scale have been used in the context of 2D vector field visualization [126]. Depending on the actual underlying line-based visualization approach, the tensor field O can also exhibit features of different spatial scales. Figure 3.16 shows an illustrative example where two groups of lines are crossing.

The group of lines from the top-left to the bottom-right is distributed broadly and represents a large-scale feature. Lines from the bottom-left to the top-right are packed tightly, and thus can be considered to be a small-scale feature. The region where both line groups are crossing can also be regarded as a small-scale feature.

In Figure 3.16(a), high-frequency noise has been used for the diffusion process. The resulting coherent noise texture shows both, the small-scale, and the large-scale features. However, the small length of the coherent structures and the high noise frequency result in a cluttered image. This specifically affects the region of the large-scale feature, because in that area a large number of short, narrow "blobs" is used to represent orientations. The length of these structures is limited by the number of iterations in the diffusion process. This number is in turn limited by the requirement that interactive frame rates should be retained during user interaction. However, the length of the coherent noise structures can be greatly increased by reducing the resolution of the domain discretization on which the diffusion process is performed. The number of "blobs" can be decreased by reducing the spatial frequency of the input noise. To summarize, in order to create fewer, longer coherent noise structures which better represent large-scale features, lower-frequency input noise and lower-resolution representations of O/L can be used. Figure 3.16(b) also shows the small- and large-scale feature. This time, a low-resolution version of O/L, as well as low-frequency noise are used as input to the diffusion process, which is likewise performed on a domain discretized with lower resolution. This results in an image that is less cluttered and contains much longer coherent structures along the large-scale feature. However, the small-scale features are less pronounced. Section 3.4.3.4 describes the blending technique that we employ to combine multiple noise resolutions in order to capture both small-scale and large-scale features.

We exploit the automatic mip-map generation on GPUs for the creation of multi-resolution image pyramids. The averaging filter that is utilized by most GPUs per default is sufficient to generate all mip-map levels of the four-component floating point texture that contains the fields O and L. We use O_i and L_i to denote the i-th mip-map level, where $i = 1$ represents the original field. The resolution is reduced by one half along each axis between subsequent levels. Here, the field O is used instead of the field O/L since the latter does not account for line density. Using O/L to generate the mip-map pyramid would overemphasize directional information from regions with low line densities. After mip-map generation, the diffusion step of the rendering pipeline is applied to each renormalized tensor field O_i/L_i separately. This creates coherent noise images N_i in the noise pyramid N.

3.4.3.4 Final Image Compositing

In the final image compositing step, the results of previous pipeline stages are combined into a single, grey-scale output image S. This image is then used to enhance the underlying line-based visualization technique by, for example, modulating image brightness per pixel. All of the presented examples (Section 3.4.4) use simple brightness modulation in order to combine a computed visualization result with the coherent noise image S.

The primary task of this pipeline step is the blending of the different coherent noise textures N_i described above. These textures are weighted so that noise frequency corresponds to local feature scale. Regions where small-scale features are present use higher-frequency noise,

whereas homogeneous regions use lower-frequency noise. In the following, the salient features of the tensor fields O_i/L_i are considered to be regions of high anisotropy and similarly oriented eigenvectors. We have chosen anisotropy because the diffusion process only generates coherent information on line-orientation in regions of strong anisotropy. The scale of a feature at pixel position $\mathbf{p}$ can be determined by considering how the anisotropy of $O_i(\mathbf{p})/L_i(\mathbf{p})$ changes with increasing i. Large-scale features retain higher anisotropy values for larger i than small-scale features. The averaging process that is applied during mip-map generation reduces the anisotropy of image regions corresponding to smaller features faster. Therefore, the following weighted sum over a user-defined number $n > 1$ of noise resolutions can be used to accumulate the output image S:

$$S = (1 - A_1) + A_1 \left(\sum_{i=1}^{n} w_i N_i \right) / \left(\sum_{i=1}^{n} w_i \right) \tag{3.12}$$

$$w_i = \begin{cases} \sum_{j=1}^{d} \max(-A''_j, 0), & \text{if } i = 1 \\ \sum_{j=n+d-1}^{m} \max(-A''_j, 0), & \text{if } i = n \\ \max(-A''_{i+d-1}, 0), & \text{else} \end{cases} \tag{3.13}$$

A_i denotes fractional anisotropy for resolution level i and is derived from the tensor-field image-pyramid by applying Equation 3.5. It is not desirable to show the noise texture in regions of low anisotropy, because these regions do not contain well-defined coherent structures. Therefore A_1 is used to blend between the noise representation and a uniform white background. While n denotes the lowest resolution mip-map level from which noise is sampled, m is the smallest overall mip-map level. A''_i represents a numerical approximation to the second-order derivative of A_i with respect to i. It is computed by using central differences:

$$A'_i = A_{i+1} - A_{i-1}, \qquad A''_i = A'_{i+1} - A'_{i-1} \tag{3.14}$$

The noise weight w_i for mip-map level i is determined by this second-order derivative since it indicates the resolution at which most of the anisotropy is lost due to the averaging process. Large negative values in a region of A''_i indicate that isotropy will start to decrease quickly in the following lower-resolution levels. A noise level j, which can represent features at level i, has to be selected. The level j is not equal to level i, because it is not possible to represent one pixel of directional information with just one pixel of a coherent noise texture. The integer parameter d is used to account for this, where $2^d \times 2^d$ noise pixels are used to represent one pixel of directional data. Typically, $d = 5$ in our examples. Since N_1 is the highest frequency noise, it is used to represent all levels between 1 and d. This is accounted for by the sum in row $i = 1$ in Equation 3.13. The case $i = n$ handles the weight computation for the noise of lowest resolution.

3.4.3.5 Focus + Context Visualization

The basis of all focus+context visualization techniques is the definition of one or multiple subsets of the data which are of interest to the user. These subsets are called the focus, which have to

be visualized in a prominent way. A context representation is used in order to provide the user with information on how the focus relates to the remainder of the data set. Many line-based visualization techniques rely on focus+context approaches, e.g., via different brushing/selection techniques. Our method facilitates easy integration of focus+context rendering. Primarily, a way to represent subsets of the data has to be established. Since the proposed technique is image-based, this can be achieved by supporting multiple input fields O^i and L^i, where O^1 and L^1 denote fields representing the entire data set. In order to create a proper focus+context representation, these fields have to be combined to form a single output image. This is somewhat similar to multi-volume rendering [19]. Combination can be performed at different stages in the rendering pipeline. A fast way to integrate l focus and context representations is to blend all input fields directly after the line rasterization stage:

$$O = \left(\sum_{i=1}^{l} b_i O^i\right) / \left(\sum_{i=1}^{l} b_i\right), \quad L = \left(\sum_{i=1}^{l} b_i L^i\right) / \left(\sum_{i=1}^{l} b_i\right) \tag{3.15}$$

The weights b_i can be used to emphasize data in the focus over data in the context. After this blending step, the rendering pipeline is executed as described before.

Another alternative is to integrate focus and context in the image compositing step. The entire rendering pipeline can be executed for all image pairs O^i, L^i separately. This results in coherent noise image pyramids N^i, and subsequently in multiple blended images S^i. These images can then either be combined by using the weights b_i, just as in the previous case, or the underlying visualization can use them individually to enhance the focus and context layers separately.

Both of these blending approaches have very distinctive advantages and disadvantages. Blending all input fields after line rasterization requires only a single diffusion step, which is the most time-consuming operation in our rendering pipeline. Furthermore, no additional texture memory is required to represent the intermediate information that is generated throughout the pipeline. The main drawback of this approach is that prior blending of the orientation tensor fields results in a field that does not exclusively represent the line orientations of the focus. This is illustrated in Figure 3.17(a). Here, the line orientations of the focus (indicated in red) are only barely recognizable in the regions where context lines are crossed. Figure 3.17(b) clearly indicates the orientations of lines in the focus throughout the whole visualization. Additionally, the generation of just one output image S restricts the way the underlying visualization method can integrate our technique. This is not the case when multiple output images S^i are generated by executing all pipeline steps multiple times for each data subset. Here, the main drawbacks are the increased memory requirements, and the necessity to compute multiple separate diffusions.

3.4.4 Evaluation and Discussion

Our visualization technique generates grey-scale images of coherent noise which illustrate general line orientation. This grey-scale output can be integrated into a multitude of visualization techniques which rely on lines as rendering primitives. We demonstrate this in the following by extending three exemplary visualization methods accordingly. In all three cases, the blended coherent noise texture S is combined with the unmodified visualization V by means of opacity

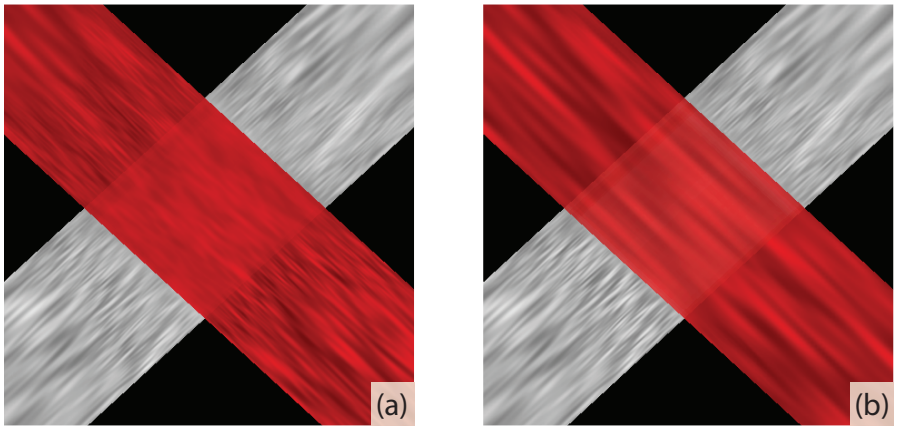


Figure 3.17: Difference of blending the focus and the context, at different stages in our rendering pipeline. (a) the two tensor fields have been created separately for focus and context during line rasterization, but are then immediately combined to form a single composite field. (b) the entire rendering pipeline is executed for both tensor fields separately, until the final image compositing step merges their coherent noise textures into the result image.

modulation:

$$\gamma SV + (1 - \gamma)V \tag{3.16}$$

The parameter γ is used to adjust how prominently the coherent noise should influence the original image. This approach retains the color information of the original visualization, which is used to distinguish focus from context. If V is a grey-scale image, image brightness instead of color has to be retained. In this case a color transfer-function f_{tf} can be applied to S before the blending step. To prevent $f_{\mathrm{tf}}(S)$ from influencing image brightness, only colors of equal brightness have to be selected for f_{tf}.

The diffusion process has been applied to the focus and the context portions of the data separately. For diffusion, 128 (Figures 3.13 and 3.19), or 256 iterations (Figure 3.18) have been computed, using a contrast enhancement parameter of $\beta = 1.28e - 3$. All application examples show data from the same time-dependent and high-dimensional computational fluid dynamics (CFD) data set. This data set contains simulation results from a model of the regeneration process within a diesel particulate filter [34]. The combustion and dissipation of soot within the filter has been simulated. The data set comprises 20 time steps and 260 K cells per time step. Overall, 5.2 million data items, each consisting of 32 scalar values, are stored.

3.4.4.1 Parallel Coordinates

The parallel coordinates implementation that we have extended with our technique is based on an approach presented by Novotny and Hauser [106], and thus scales well with data size. Figure 3.13 shows two result images which have been created by visualizing 10 different dimensions of the diesel particulate filter data set while using three different line density transfer functions. A linear/logarithmic mapping is used in the left/center images respectively whereas a

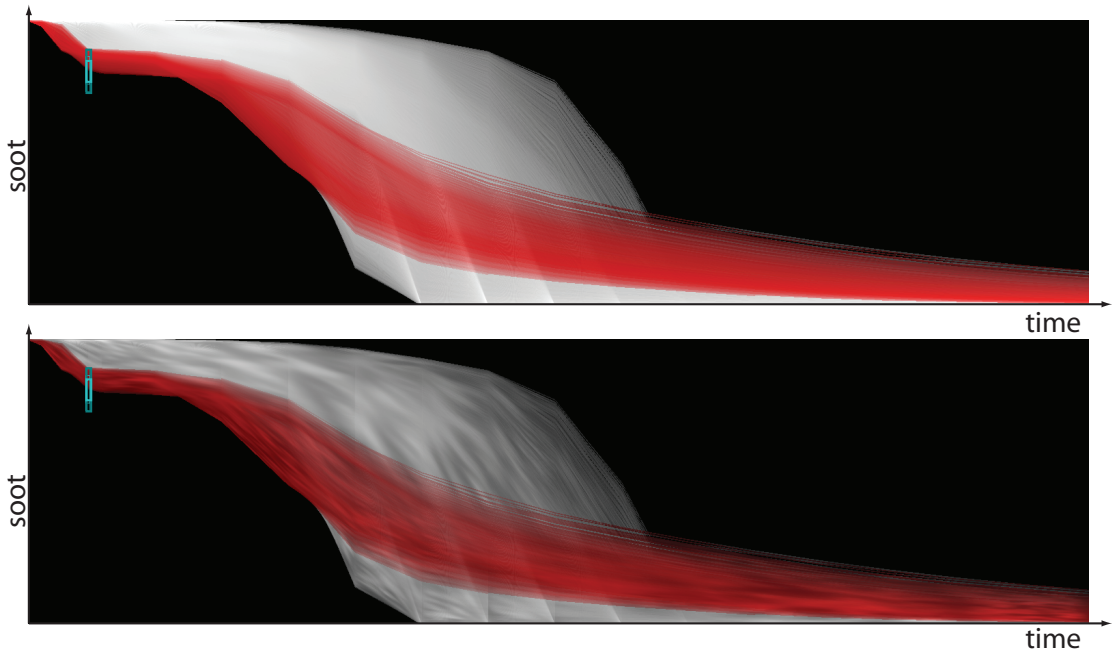


Figure 3.18: Comparison between an original time series visualization (top), and our enhanced version (bottom). The structure that is added by our technique helps the user to discern general line orientation, especially in regions of low image contrast. Here, the scalar attribute represents the amount of soot within a CFD cell. The user-defined selection highlights portions of soot that initially burns at a fast rate.

color transfer function is applied in the right-most visualizations. All 20 time steps are shown, which implies that 5.2 M polylines have to be drawn to create these results. Each polyline is made up of nine line segments. The right-most visualizations indicate that some pixels are crossed by nearly all of the lines. The top-row images show the original visualization results. Color is used to differentiate focus from context in the first two visualizations. Red indicates selected data items, whereas the context is visualized using a grey-scale transfer-function. Our technique has been applied to the top-row images in order to create the bottom-row images. Especially evenly populated regions benefit from the addition of the coherent noise image, as it adds visual cues that indicate line orientations. Furthermore, regions of parallel lines can be clearly distinguished from regions where lines are crossing, since anisotropy is used to attenuate the influence of the coherent noise texture. Here, the parameters in Equation 3.12 have been set to $n = 3$ and $d = 5$, respectively.

3.4.4.2 Time-Series Visualization

The time-series visualization presented in this section is based on the technique presented in Section 3.3. For each data item, i.e., cell in the case of the CFD data set, one curve is drawn which represents the development of a scalar attribute over time. Therefore, 260 K polylines,

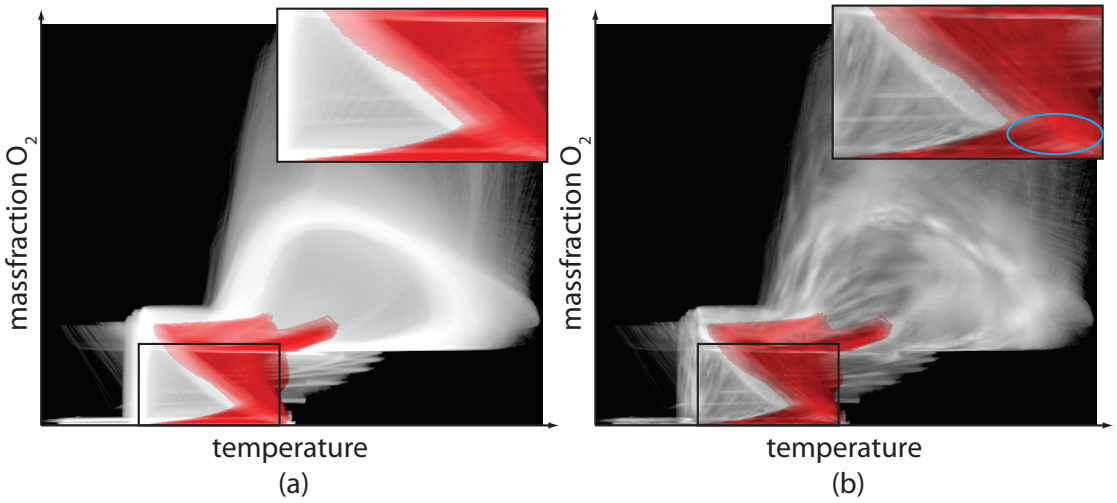


Figure 3.19: Enhancing phase-space diagrams with our method (b). The focus data is the same as in Figure 3.18. For comparison, the original visualization is shown in (b).

each comprising 19 line segments, are displayed. The bottom image in Figure 3.18 has been created by modulating the top image with the coherent noise texture. Structure is added in regions where the line-density to brightness mapping is creating uniform regions which lack contrast. Similarly to the parallel-coordinates implementation individual lines are clustered which results in a highly scalable visualization approach. Selected data is indicated in red, whereas the context is again visualized using a grey-scale representation. The visualizations in Figure 3.18 show plots of the amount of soot per cell. Since soot is burning up over time, all curves are decreasing monotonously. The user-defined selection represents portions of the data set where soot is initially burning at a fast rate. Although the dissipation of soot starts early in the selected regions, it takes longer to vanish completely compared to the remainder of the data set. Here, the parameters in Equation 3.12 have been set to $n = 3$ and $d = 5$, respectively.

3.4.4.3 Phase-Space Diagram

A phase-space diagram is a generalization of the time series visualization presented in the previous section. Instead of plotting the evolution of one scalar value over time, curves representing the evolution of each data item with respect to two arbitrarily selectable data attributes are rendered. Again 260 K polylines, each containing 19 line segments, are displayed in Figure 3.19. The x-axis is mapped to the temperature within a cell, and the y-axis represents the amount of contained oxygen O_2. The focus represents the same subset of the data as in Figure 3.18. In Figure 3.19(b), our technique has been used to enhance the phase-space plot shown in (a). The coherent structures added in (b) nicely depict regions where data items evolve over time in a similar fashion. The combustion process of the soot in the selected regions can be tracked easily. The combustion starts after a short heating period at the region indicated by the blue ellipse. After this, temperature and oxygen concentration drop off significantly. Here, the parameters in

Equation 3.12 have been set to $n = 3$ and $d = 4$, respectively.

3.4.4.4 Impact on Rendering Performance

We evaluate the performance for the parallel-coordinates visualization rendered to a 1024×768 view port on an NVIDIA Geforce GTX 480 graphics card. Basic line rasterization has to be performed for most visualization approaches which employ lines as rendering primitives. The line rasterization stage introduced in Section 3.4.3.1 can be performed as part of the basic line rasterization by exploiting multiple render targets. Therefore, no separate rendering pass has to be used to generate the fields O and L. Writing to additional render targets increases the fill-rate requirements of the original algorithm. However, we found that this does not strongly impact performance since the line rasterization process seems to be limited by geometry throughput. We also found the difference between rendering to a single or to three floating-point render targets to be also barely measurable at 1 ms. If the original visualization method uses only a single 8-bit fixed-point render target, the overhead incurred by the additional two floating-point targets increases to 50 ms.

However, the most important property of the proposed technique with respect to rendering performance is that the remainder of the rendering pipeline is purely image-based. For these stages, the data size does not influence rendering speed at all. Instead, the output image resolution is the limiting factor. In image space, computing the anisotropic diffusion requires the most time, since a high number of iterations has to be computed, where each iteration performs a large number of texture fetches. For 128 iterations for both focus and context data, i.e., 256 iterations in total, our system needed 104 ms. Since mip-map generation is a hardware feature of current GPUs, the impact on the overall rendering time for the generation of the multi-resolution image pyramid is negligible (below 1 ms). The evaluation of an anisotropy measure also barely influences performance. The final image-compositing step requires only 1 ms to blend multiple noise levels for the final coherent noise texture.

3.4.5 Conclusions and Future Work

This work introduces a novel, generic image-based technique that can be used to enhance many existing visualization methods which depict a large number of lines in 2D. A 2D tensor field, subsuming line orientations per pixel, is used to drive anisotropic non-linear diffusion of white noise. The resulting coherent noise texture improves the perception of overall line orientations even if a large number of overlapping lines have to be displayed. Multiple noise frequencies are combined, with blending weights determined by a local feature-scale measure. Using this approach, line orientations in large-scale features can be illustrated with low-frequency noise, whereas high-frequency noise is used for small-scale features. We presented two different ways of how focus+context rendering can be integrated. Since our approach is image based it is highly scalable with respect to the number of lines that have to be visualized. The general applicability and usefulness of our method has been demonstrated on three different line-based visualization methods.

Future work will deal with extending continuous visualization techniques, such as continuous parallel coordinates [56], or continuous phase space diagrams. The latter would be a

time-dependent extension to continuous scatter plots [7, 87]. Furthermore, additional blending strategies for the combination of different noise levels could be explored.

CHAPTER 4

Visual Exploration of Nasal Airflow

This chapter is based on the publication *Visual Exploration of Nasal Airflow* by Zachow et al. [148] and will demonstrate how the visualization techniques that have been discussed in previous chapters can be used to explore and analyze a complex real world CFD data set.

In otorhinolaryngology (a medical specialty concerned with diseases of the Ear, Nose, and Throat), ENT specialists (esp. rhinosurgeons) are often faced with the problem of assessing nasal breathing from a functional point of view. Due to the anatomical situation, direct measurements within the nose are not possible without influencing the airflow itself. Besides a visual inspection of the nasal airways with regard to obstruction and deformities, as well as medical imaging as a diagnostic means, the only objective measurement of nasal airflow is rhinomanometry which, however, delivers rather unspecific integral information on the pressure gradient as well as on total flow respectively flow resistance. Thus, physiological function of the nose cannot be assessed sufficiently and therapeutic concepts are difficult to derive.

In cases of pathological nose breathing due to any persistent obstruction or deformity, a surgical correction might be indicated. Rhinosurgery is a reconstructive surgical approach that reshapes the external nose and/or inner structures of the nasal cavities with the objective to re-establish *normal nasal breathing*. Though treatment methods in ENT surgery have constantly improved, the appraisal of a promising individual therapy under consideration of *regular nasal airflow* still is a challenging task. Very often too radical resection of turbinate mucosal tissue results in a so called “empty nose syndrome” (ENS). There is no standard, however, of how much turbinate tissue has to be removed to re-establish physiological breathing. Neither is it known how much tissue can be removed before it causes other damage to the physiology of the nose. An in-depth knowledge of normal nose breathing, as well as an understanding of the relationship between morphology and physiology of the nose will be the foundation for an improved rehabilitation in functional rhinosurgery.

Our objective is to analyze and to understand the physiology and pathophysiology of nasal breathing. To this end, airflow simulations based on computational fluid dynamics (CFD) have been conducted for a highly detailed anatomy of the upper respiratory tract from the external nose deep into the trachea, including frontal and maxillary sinuses, as well as the ethmoid [149,

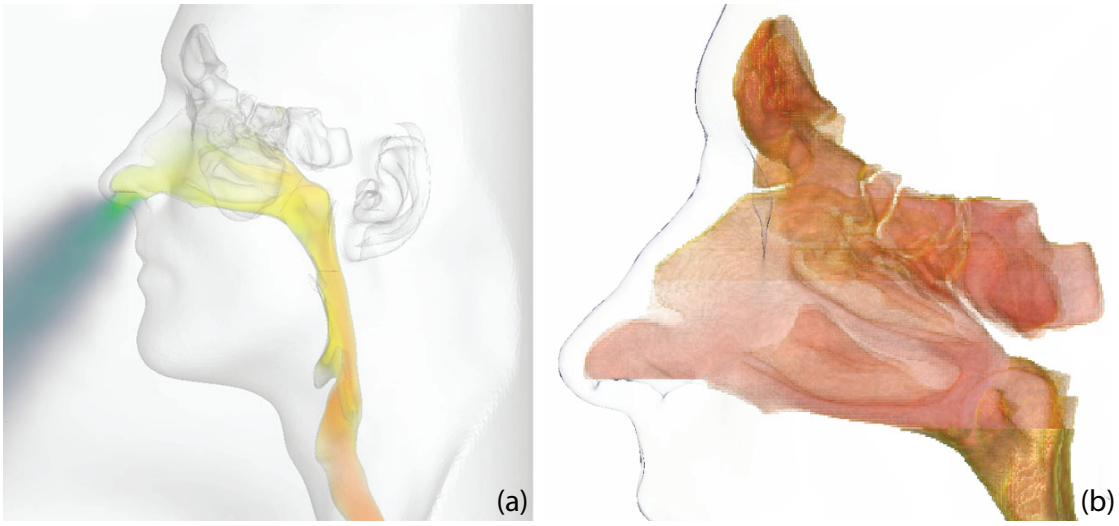


Figure 4.1: Model of the upper respiratory tract (a), nasal cavities (b).

150] (see Figure 4.1). The following sections show how the different scientific visualization and information visualization approaches that have been introduced throughout this thesis can be used to perform a visual exploration of complex transient airflow simulation results.

In particular we demonstrate how highly scalable techniques for information visualization approaches such as parallel coordinates, time series visualizations, as well as unstructured grid multi-volume rendering, all integrated within a multiple linked views framework, can be utilized to gain a deeper understanding of nasal breathing. Evaluation is accomplished by visual exploration of spatiotemporal airflow characteristics that include not only information about the flow features but also on accompanying quantities such as temperature and humidity.

4.1 Related Work

The function of the human nose is manifold, ranging from breathing and smelling to humidification, warming, and cleaning of the air. For patients with a functional impairment of the nose, however, it is neither clear how to assess the individual function of the nose, nor how to re-establish regular nose breathing. In fact, it is even unclear what *regular* nose breathing is. Thus, there is a great demand for understanding the physiology and pathophysiology of nasal breathing in functional rhinosurgery [99]. With constantly increasing computational performance as well as advances in modeling and simulation of complex problems in applied mathematics and engineering, CFD has become an attractive means for predicting fluid flow in a variety of bioengineering problems [80]. In medical applications, for instance, CFD methods have been successfully applied for blood flow simulations [46]. Several research groups also showed that computational fluid dynamics is basically applicable to analyze airflow phenomena in the nose. Investigations vary from general analysis of nasal airflow including experimental work [8,36,63], to simulations with respect to morphology changes [37], odorant transport [151],

temperature distribution [91,110], aerosol distribution [117], up to questions for decision support in rhinosurgery [47, 90, 147].

The visualization of CFD simulation results (flow visualization), is a broad area of research mainly dealing with 2D, 3D or even 4D vector fields. A good survey of commonly used methods can be found in publications by Post et al. [111] and Laramee et al. [86]. Especially when dealing with more complex simulation models, i.e., models incorporating not only flow but also data about processes such as combustion, chemical reactions or meteorological phenomena, interactive exploration and analysis techniques have gained importance. Gresh et al. [50] were among the first using interactive visual analysis methods such as multiple linked views (attribute space as well as 3D visualizations) for the analysis of scientific simulation data. Doleisch et al. extended previous binary brushing approaches by proposing smooth brushing [32], and introducing more complex feature definition concepts in combination with focus+context visualization [30, 31]. Meanwhile, interactive visual analysis frameworks have been successfully introduced in many other application areas such as automotive engineering [33], meteorology [35], and climate research [70].

Since most CFD simulation data is either of spatial (3D) or spatiotemporal (4D) nature and typically large in size, it is important for the visual exploration process to provide highly responsive volume visualization techniques, which must be able to cope with the underlying spatial grid structures. The nasal airflow data presented in this work is specified on a static unstructured grid. Thus, unstructured grid raycasting [48] is heavily used for the visual exploration of the data. In order to retain interactivity the GPU implementation that has been presented in Chapter 2 has been used. In addition, classical information visualization techniques such as parallel coordinates [62] are helpful for the exploration process, whereas different optimizations have been proposed to retain interactivity. Novotny and Hauser [106], for instance, propose a binning approach which is also incorporated into the visual analysis framework of Akibay and Ma [5]. This binning scheme was also used throughout Chapter 3.

Besides the fact that very much work is done in nasal airflow simulation as well as in exploratory visual data analysis, to our knowledge, this is the first in-depth visual exploration of nasal airflow phenomena. We evaluate several simulated breathing cycles within a comprehensive model of the nasal airways under consideration of realistic boundary conditions, and all physically relevant time-varying quantities.

4.2 Visual Exploration of CFD Simulation Results

There are three possibilites to evaluate time-dependent field variables, such as velocity, pressure, temperature, humidity, turbulence quantities, etc.: 1) visualization of individual time steps, 2) visualization of changes between time steps, and 3) visualization of developments within finite time intervals. In this work we focus on the first two techniques. The AMIRA software was used for the visualization of single time steps (stationary flow analysis), and the SimVis [31] software was used for the exploration of flow dynamics (transient flow analysis).

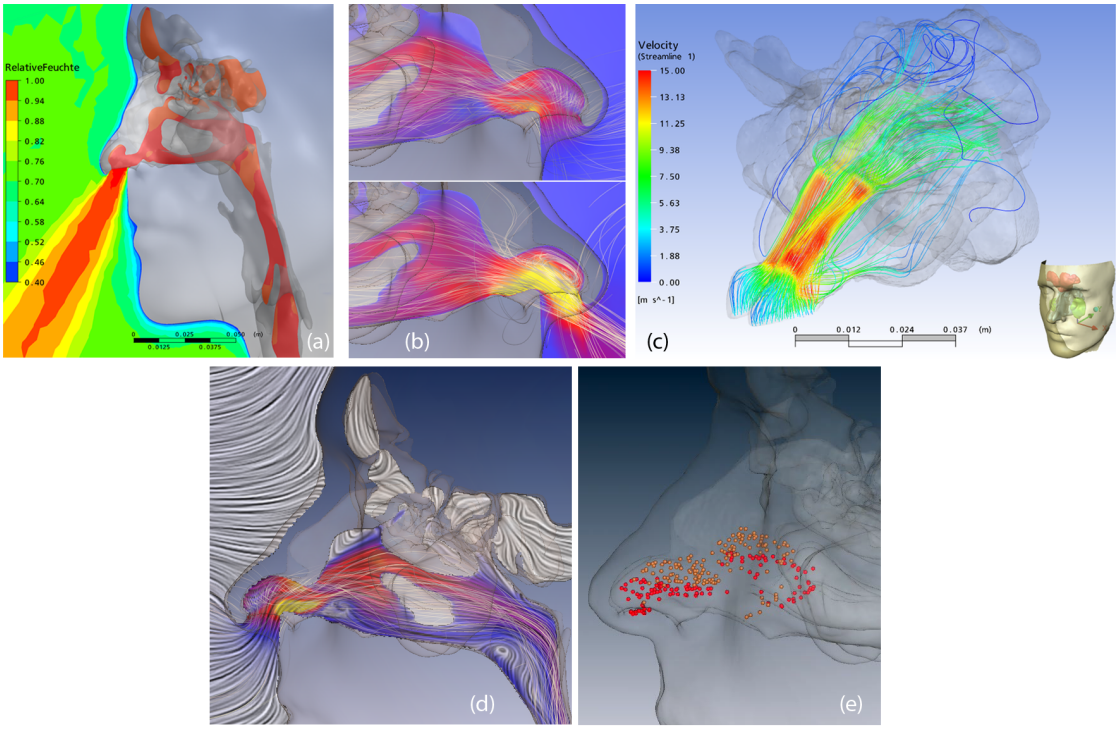


Figure 4.2: Visualization of flow quantities: Relative humidity for an exhalation state (a), flow velocity within the nasal cavity (b) during inhalation (top) and exhalation (bottom), inspiratory flow velocity of up to 15 m/s with color coded stream lines (c), illuminated streamlines with color coded planar LIC visualization (d), and particle visualization along field lines with discrimination of left and right side (e).

4.2.1 Stationary Flow Analysis

To get an overview of the simulation results, individual time steps of the inhalation and exhalation phases were selected in the first place and the respective flow quantities were visualized. Since all data are encoded on volume cells of the unstructured input grid, appropriate visualization methods have to be used. For stationary analysis we used techniques like color coding of scalar or vector valued quantities, contour plots, iso-surfaces, vector visualization, line integral convolution (LIC), streamline and particle visualization with an animation of seeded streamlines or particles along respective field lines (see Figure 4.2). These techniques are provided by the AMIRA software [122] that has already been employed for the geometric modeling in this work.

Due to its ease, we made heavy use of the possibility to visualize any quantity on top of arbitrarily oriented planes with an appropriate color coding scheme. In Figure 4.2(b), for instance, we see on a section of the nasal cavity that for inhalation (top view) and exhalation (bottom) the main path of airflow differs. Note that the lung induces the flow by creating a reduced pressure, causing air flowing into the nose. Thus, outflow is directed due to the shape of the nose, whereas inflow is rather nondirectional. Temperature and humidity are also visualized this way (see Fig-

ure 4.2(a)), thus enabling us to assess inhalation and exhalation while getting a first impression of the scalar distribution and being able to check the correctness of boundary conditions. Isosurfaces were generated to locate pressure minima, and LIC techniques were best suited for the understanding of flow structures [18] as shown in Figure 4.2(d). A visualization of the flow velocity during inhalation and exhalation using color coded streamlines yields a good overview on spatial flow characteristics for individual time steps. Looking at the velocity field, for example, we can easily grasp the flow structure and see that a maximum of up to 16 m/s is reached within the main nasal cavity (Figure 4.2(c)). In Figure 4.2(b) we used illuminated streamlines in combination with color coded planar cuts, mapping the magnitude of velocity to color values. Even for a single time step the flow field yields enough information to visualize flow characteristics in an animated fashion. Illuminated streamlines can be animated with Amira [123], as well as particles that - after being seeded - move along the streamlines [116] to visualize flow direction (Figure 4.2(e)). Seeding of streamlines or particles can be accomplished, for instance, by locating pressure minima in order to visualize vorticity. All these techniques allow for a first visual exploration of the data, where the challenge is to locate important flow features and to choose the appropriate visualization technique. The spatiotemporal relations, however, cannot be easily assessed this way. Therefore a transient flow analysis is needed.

4.2.2 Transient Flow Analysis

Without a priori knowledge of relevant structures within time-dependent 3D data, the exploration and the extraction of meaningful features is a highly challenging task. This explains the need for a highly interactive approach. On the other side, visual analysis and data exploration often turns out to be difficult or even impossible at all, without employing semi-automatic visualization methods that require minimal user interaction. Thus, we have used a highly interactive visualization system with basic built-in data analysis tools to thoroughly explore the nasal breathing data and to form hypotheses about the features which have been identified during this process. The following section gives an overview on visualization techniques which have been employed and extended in order to tackle the challenging task of gaining insight into the process of nasal breathing.

4.2.2.1 Multiple Linked Views for Feature Selection

The framework which has been used to perform the transient flow analysis provides multiple linked views for concurrent visualization, exploration, and analysis of multi-variate data. During our exploration sessions we have used multiple 3D views in combination with several types of attribute views, e.g., scatterplots (Figure 4.5(c)), histograms, parallel coordinates [5, 106] (Figure 4.5(d)) or time series visualizations (Figures 4.5(e, f)) in order to gain an overview on multiple data modalities. The interactive brushing functionality provided by all attribute views has been utilized in order to define data regions which exhibit interesting behavior. This feature definition process resulted in multiple *Degree of Interest* (DOI) volumes each storing a scalar field which equals one in interesting and zero in uninteresting data set regions. Since especially data from CFD simulations is often distributed smoothly, it may be difficult to sharply discriminate between important and less important data regions. Thus non binary, i.e., unsharp

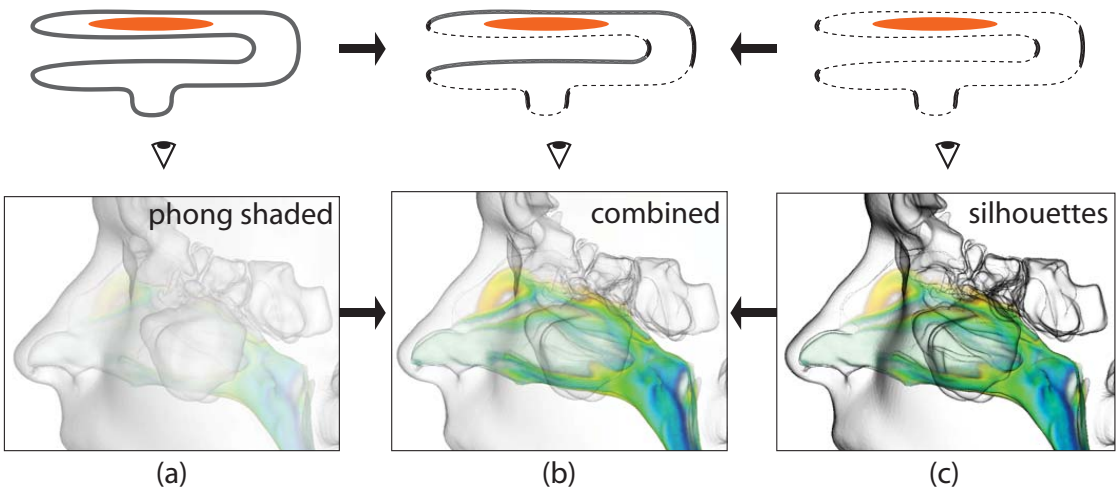


Figure 4.3: Comparison of boundary visualization methods.

$DOI \in [0..1]$ values have been used to convey this uncertainty [32]. Subsequently DOI attributions have been combined by using fuzzy logic operations in order to form more complex features as discussed in the following sections.

4.2.2.2 Volume Rendering Techniques

Immediate feedback on the spatial location of interesting features via scientific visualization methods such as volume rendering is crucial. In this respect the nasal airway data poses two main challenges: On the one hand the underlying unstructured grid is reasonably large (1.3 million tetrahedra and 2.2 million prisms which would equal roughly 8.0 million elements when tetrahedralized) and on the other hand the anatomical structure of the nasal airways with all related cavities is very complex which can lead to cluttered images and occlusion problems. We tackle these challenges by utilizing two different volume rendering techniques: In Figure 4.5(b) a *point-based* overview rendering of the nasal airway data is shown. Here volume cells are represented by Gaussian splats which are blended over each other in back to front order. Data regions which have not been selected by the user (i.e., $DOI = 0$) are rendered with lower opacity and no color, whereas selected features are displayed less transparent and with a color that is derived from a scalar data attribute via a transfer function. The main advantage of this rendering approach is its performance, as long as the viewing direction remains constant (otherwise an expensive visibility sort becomes necessary). Thus, it has been used to interactively explore the spatiotemporal evolution of selected features by allowing the user to rapidly move from timestep to timestep. Main drawbacks of this method are the low rendering quality (circular Gaussian splats are only an approximation of the cells) and the lack of a distinct visualization of object boundaries.

In contrast to the point-based approach, *raycasting-based* volume rendering, equipped with user selectable optical volume models, allows for a visualization of different semi-transparent boundaries. Only regions for which $DOI > 0$ are visualized using volume rendering, whereas

unselected portions of the data set are skipped by the raycasting implementation. Choosing a suitable rendering mode for the volume surface is crucial since it provides context information which enables us to spatially locate selected features within anatomical structures. Figure 4.3(a, c) shows two different boundary surface rendering methods. Figure 4.3(a) uses simple transparent Phong shaded surface rendering, whereas in Figure 4.3(b) a simple silhouette enhancement shading model is chosen. Obviously, for the presented case both approaches are problematic: the surface rendering lacks contrast because the feature selected within the nasal airway is behind multiple irrelevant surfaces which are displayed in front of the nasal cavity. On the other hand the silhouette rendering introduces cluttering and makes it difficult to recognize the distance between viewer and feature of interest. Thus, we chose a combination of both visualization methods by using the surface shading for all faces pointing away from the user, and silhouette enhancement only for front facing faces as shown in Figure 4.3(b). This effectively halves the number of the surfaces occluding volumetric features while still supplying depth cues by rendering back facing surfaces. In addition, silhouette rendering adds detail for front facing surfaces. In order to guarantee correct composition of transparent surface and volume rendering an extended depth peeling method is used that has been introduced in Section 2.3.

4.2.2.3 Parallel Coordinates

A very useful tool for visual exploration of the high dimensional attribute space has been a scalable implementation of parallel coordinates (Figure 4.5(d)). It is based on techniques proposed by Novotny et al. [106] and also used by Akibay and Ma [5]. In order to further reduce cluttering and overdraw issues inherent to parallel coordinates when visualizing millions of data entries we use a noise blurring approach that is a precursor to the technique discussed in Section 3.4. Similarly to this method a structure tensor field is constructed by computing the following sum per pixel, $\frac{1}{N}\sum_{i=1}^{N}\mathbf{v}_i^T\mathbf{v}_i$, with N being the number of lines covering the image pixel and $\mathbf{v}_i$ the direction of the i^{th} line in image space. The resulting tensor per pixel encodes directional information about all lines passing through that pixel. This is illustrated in Figure 4.4: In (a) the directions of the lines through a pixel are not distributed uniformly around that pixel which results in an anisotropic tensor visualized by the red ellipse (the ellipse is constructed from the eigenvectors of the tensor). On the other hand lines with uniformly distributed directions passing through an image pixel result in an isotropic tensor as illustrated in Figure 4.4(b). Now simple diffusion is performed on a noise texture via the tensor field in order to create a pattern which is

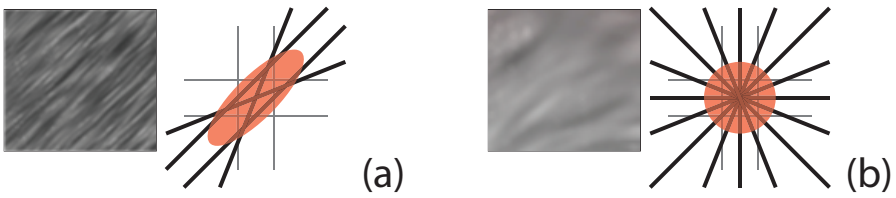


Figure 4.4: Different line direction distributions through an image pixel result in anisotropic (a) or isotropic (b) tensor representations which can be used to blur a noise texture. This allows for the visualization of large trends in our parallel coordinate view.

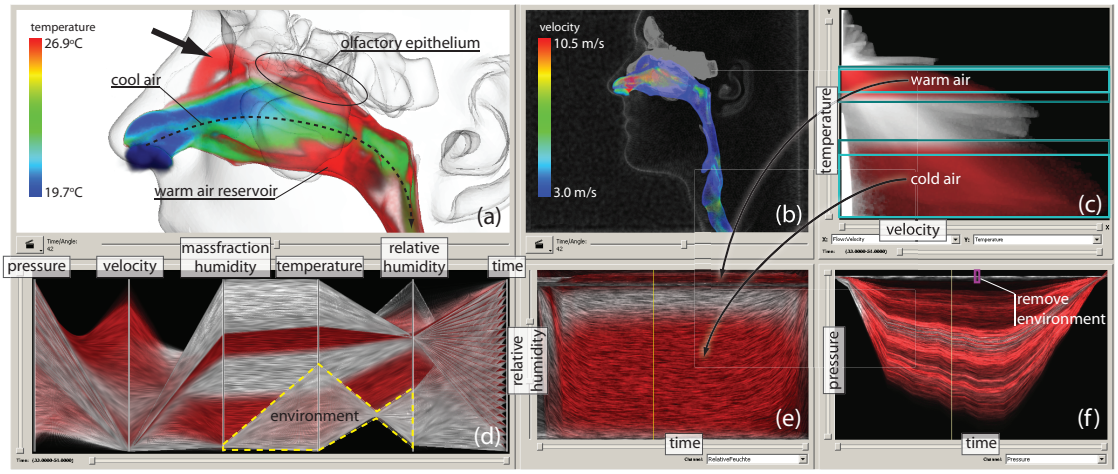


Figure 4.5: Overview of a visual exploration session: Different attribute views (c), (d), (e) and (f) are used to interactively select data regions of interest which can be examined and modified in all other visualizations. The 3D views (a) and (b) display the selections in their spatial context.

blurred along the predominant line direction at each image pixel (see insets in Figure 4.4). We use 16 iterations with a 7×7 Gaussian filter kernel to achieve sufficient image quality retaining interactive frame rates with our current GPU implementation. The main advantage of using this approach is that even when drawing a large number of lines on top of each other, the user is still able to recognize trends which otherwise could have been easily missed. In the example shown in Figure 4.5(d) 21 time steps are visualized resulting in about 75 million lines.

4.2.2.4 Visual Exploration

To gain insight into complex transient 3D simulation data by using an interactive multiple linked views framework a generic workflow can be followed which closely matches the information seeking mantra proposed by Shneiderman [120]: *Overview first, zoom and filter, then details-on-demand*. First the user has to gain an overview on all potentially interesting data attributes and their spatial distribution within the simulated domain. This is achieved by setting up an exploration session as shown in Figure 4.5. As a starting point only simple selections (e.g., selecting only high or low attribute values) are defined by the user which can then be explored in all linked attribute and 3D views.

This first step is illustrated in Figure 4.5 where relatively cold air (up to 23.5° C) is selected by the lower brushing rectangle in Figure 4.5(c) in order to gain an overview of the aerodynamic heating during the inhalation phase of the first breathing cycle. An additional selection of warmer air (between 29° and 31° C) has been defined in order to highlight regions of air which have already been warmed up by the mucous walls, but which are colder than the air within the paranasal cavities. Both selections represent features that can be explored in all other views shown in Figure 4.5. The 3D visualization in Figure 4.5(a) shows this feature at 450 ms after the beginning of the first breathing cycle. In Figure 4.3 only the heated air is shown using a dif-

ferently scaled transfer function. The two parts of the selected data region (the cold air and the slightly warmed air) can be discriminated by their coloring. Blue-green corresponds to colder flow whereas red represents the warmer data selection. It can be observed that the main portion of cool air travels along one distinct path as indicated by the dotted line and is warmed up approximately $4.5° \, C$ during its traversal. Note that a reservoir of warmer air is located below the dotted line as indicated by the label. Further exploration of this feature will be conducted in subsequent sections.

A second notable structure can be seen on the left side behind the nasal valve. Here the geometry of the Isthmus Nasi (the narrowest point between vestibule and the nasal atrium) seems to result in vortical flow which leads to early mixing of warm and cold air (indicated by the black arrow in Figure 4.5(a)). The overview visualization presented in Figure 4.5(b) shows the same features as in (a), but this time flow velocity has been mapped to color. Again the main trajectory of the inhaled fresh air can be discerned by looking for high velocity regions of the selected data set volume. Note that in the regions of the olfactory epithelium the air flow velocity during the inhalation phase remains relatively low. This allows for the preservation of humidity and warmth which is crucial for the performance of the sense of smell. Additionally the high air velocity in the middle nasal passage during the inhalation phase suggests that the frontal and maxillary sinus are ventilated during this breathing phase. This is accomplished through small ducts opening into the middle nasal passage.

The temporal development of the selected features has not yet been investigated in detail. Even though the scatterplot that is used to select the cold and warm air is capable of displaying a selected time-interval of the data which can be changed interactively, the overall evolution of the feature over time is difficult to judge from this basic view. Thus, the time series visualization proposed in Section 3.3 has been used in Figure 4.3(e, f) to plot the selected feature over time within the context of an additional data attribute. Here, for each volume cell the actual relative humidity (e) and the pressure (f) are plotted over time (x-axis). Color indicates to which degree a curve is selected (red equals fully selected and white represents unselected data) whereas brightness represents the number of curves overlapping each other. The vertical yellow line is linked to both time series views in correspondence to the currently shown timestep in the 3D visualizations.

Since the SimVis framework currently does not explicitly support distinguishing between semantically different portions of the simulated volume (e.g., the surrounding environment and the nasal airways) the relative pressure time series view (Figure 4.5(f)) has been used to exclude the anterior inflow region from all selections by employing an inverted *timestep brush* which is represented by the pink rectangle. This kind of brush defines a DOI function which is zero if the underlying scalar attribute lies within a user defined interval (here a small neighborhood around zero relative pressure) at a certain timestep and one otherwise. In this case the timestep of maximal negative relative pressure at the interface to the lung has been selected since it exhibits the strongest pressure differential between the nasal airways and the environment.

One of the important functions of the nasal airways is moistening inhaled air. Figure 4.5(e) gives an overview of the temporal development of the relative humidity within the upper respiratory tract. By investigating the curve progression, which is indicated by a line integral convolution based approach, a steep drop in relative humidity can be observed which stabilizes after

approximately 200 ms. Note that the warm/cold air feature previously defined in the scatterplot is shown as two distinct red regions in the respective visualization. The higher humidity region corresponds to the warm air selection whereas the low humidity region corresponds to the selected air with lower temperature. After 700 ms seconds relative humidity increases again since low pressure from the lung diminishes (see Figure 4.5(f)).

The relative-pressure time-curves shown in Figure 4.5(f) help locating currently investigated timesteps in the context of the whole breathing cycle. When investigating the visualization it becomes apparent that after a short period of time (roughly 150 ms) the basic distribution of pressure values remains constant with respect to scaling (i.e., there are no curves crossing each other). This means that after the initial turbulent conditions between the exhalation and inhalation phases, flow features such as vortices remain fairly stationary in the simulated volume which will be confirmed in consecutive sections.

In Figure 4.5(d) our parallel coordinates implementation is used to gain an overview on the selected data regions in multiple additional data dimensions. Like in all other attribute views currently selected features are highlighted in red whereas the remaining data is shown in white/grey. The axes correspond to the scalar attributes pressure, velocity, massfraction of humidity, temperature, relative humidity and time. At a first glance, a strong correlation between mass fraction of humidity and temperature can be recognized (this assumption is further strengthened by the horizontally blurred noise indicating that most of the lines are parallel to the x-axis). In addition, at least the selected data regions exhibit a correlation between temperature and relative humidity. Furthermore a negative correlation between flow velocity and mass fraction humidity can be observed. As with the scatterplot we have explored the temporal distribution of the data values by interactively selecting time intervals that are to be visualized. The region of the data set marked with "environment" corresponds to the anterior inflow region in front of the face which, as discussed previously, has been excluded from the selections via a *timestep brush*.

4.2.2.5 In Depth Visual Analysis

Now after a first superficial exploration of the simulation data the next step in our analysis workflow corresponds to the *zoom and filter* portion of Shneiderman's mantra. In order to gain additional insight into the nasal breathing data we use the analysis framework to derive additional information such as spatial/temporal gradients or more complex flow parameters such as λ_2. These can then be used to further restrict/modify previously defined features which allows the user to form further hypotheses about the data.

In the following an in-depth analysis of two distinct features will be presented. Figure 4.6 shows an overview of three different timesteps: 100 ms, 600 ms, and 1 s after the beginning of the inhalation phase which lasts for roughly 1.05 s. At 600 ms an additional view from below the head is shown. In the column denoted by (a) a selection is depicted which is based on the warm air feature that was explained in the previous section. This selection has been further restricted to low velocity regions. Temperature is mapped to color. The column labelled (c) shows the parallel coordinates view at corresponding timesteps which has been used to define the selection via two brushing rectangles (indicated in yellow). In column (b) of Figure 4.6 the warm/slow air feature has been further restricted to vortical regions within the flow, as defined by the λ_2 criterion [64]. Color is mapped to the strength of vorticity with green indicating weak,

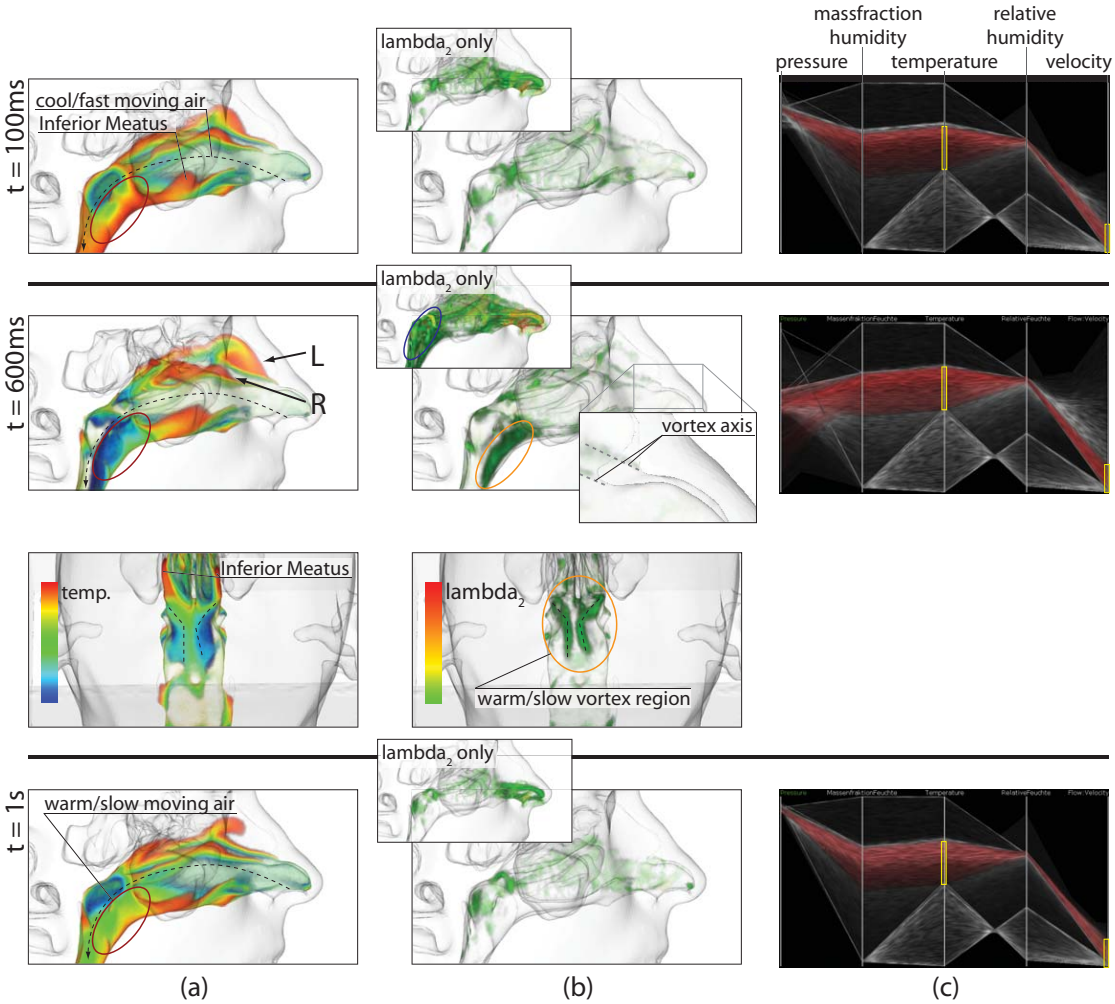


Figure 4.6: Different timesteps of a feature defined by low flow velocity and relatively high temperature are shown in column (a). In column (b) the feature is further restricted to vortical regions as defined by the λ_2 criterion. For $t = 600\text{ms}$ an additional pair of images from below the head illustrate how vortical regions can be related to heat distribution. The parallel coordinate plots show the data at corresponding timesteps (c).

and red indicating strong vortical air motion. The insets labelled as "lambda_2 only" provide an overview of all vortical regions without any restriction to warm/slow flow regions.

The first feature discussed in this section is the early mixing of cold and warm air behind the nasal valve. It is notable that this process is much stronger within the left side of the nose than the right side. This can be seen clearly at 600 ms when comparing the selected regions indicated by "L" and "R". The cause for this is an asymmetry in the geometry of the external nose as shown in the close-up image in row (b). Here, two different (weak) vortices, which are indicated as grey dotted lines, originate at cartilage structures at the top of the Isthmus nasi. On

the left side of the nose the progression of this vortex points slightly up which results in the stronger expression of this feature. This shows how minor deviations in the nose geometry can result in rather different air flow behavior, thus, being a strong indication for further studies on varying nasal geometries. Furthermore this shows that the geometry of the Isthmus nasi is highly important for directing inflowing air and thus has to be considered when deriving diagnoses or performing operative interventions.

The next feature that has already been hinted at in the previous overview is related to the existence of relatively warm air in the area of the nasopharynx, shown in all three timesteps in Figure 4.6 (see red ellipses in column (a)). This is notable since all inhaled air has to travel through this region at a rapid rate which should inhibit the warming process. The selected regions shown in column (a) of Figure 4.6 all represent warm but also slow moving air (as selected in the parallel coordinates view (c)). This indicates that most of the fast moving cold air is traveling along the trajectory already identified in Figure 4.5(a). A main cause for this is a complex system of vortices located in the upper regions of the nasopharynx marked by the blue ellipse in the close-up picture in column (b) at 600 ms. It shows vortical regions without the restriction to warm/slow air with color indicating vortex strength. Below the vortex system transporting fast/cold air the aforementioned slowly moving air can be warmed up more efficiently. This process is further aided by a secondary system of two parallel vortices indicated by the orange ellipses in column (b) of Figure 4.6. They originate at the transition between nasopharynx and the Inferior Meatus on both sides of the nasal airways. Note how the vortices shape the structure of slow moving region of warm air. The vortex axes are indicated as stippled lines.

4.3 Conclusions and Future Work

A visual exploration of nasal airflow for several simulated breathing cycles has been presented. Fluid flow simulations are based on a comprehensive geometric model of the upper respiratory tract as well as realistic boundary conditions under consideration of physically relevant quantities such as pressure, temperature, and humidity. CFD simulations of several breathing cycles lead to 85 GB of time-dependent multi-variate flow data that have been visually analyzed within this work.Visual exploration of the data required combinations of different highly scalable scientific visualization and information visualization methods that have been introduced throughout this thesis. For a stationary analysis of the simulation data we employed techniques for scalar and vector field visualization, such as color coding, contour plots, iso-surfaces, line integral convolution, streamline visualization, and tracing of seeded particles. This already led to interesting observations such as major differences between the progression of inhaled and exhaled air within the nasal cavities.

In order to gain an overview of the temporal characteristics of the flow data we used multiple linked views for concurrent visualization, exploration, and analysis - employing 3-D views in combination with several types of attribute and time series visualizations. Specific challenges imposed by data size and geometric complexity have been addressed by utilizing different volume rendering implementations as well as a scalable parallel coordinates and time series visualization implementation. Interactive brushing and linking techniques enabled us to define and investigate data regions that exhibit interesting behavior such as vortex systems in the lower part

of the nasopharynx which might be related to warming inhaled air, or early mixing of inhaled air with warm air in the nasal valve region. A visual exploration framework enables researchers to understand how their data fits into the context of known conceptions or beliefs and accelerates the process of generating testable hypotheses such as:

- The differing courses of airflow in inspiration and expiration due to the nasal geometry suggest an airway resistance that not only depends on the nasal valve but also varies with flow direction.
- The more cranial course of the inhaled air supports an adequate presentation of odor molecules to the olfactory receptors. During exhalation, the direction of warm and humid air to the olfactory epithelium possibly creates optimal climate conditions for the smelling process, while providing a retrograde transport of odor molecules from the oropharynx to the olfactory slit.
- The nasal isthmus obviously has a strong impact on the distribution of the air stream. Its geometry not only seems to be significantly responsible for the nasal airflow resistance but also relevant for directing air during the inhalation phase.
- The air stream through the middle meatus within the initial inspiration phase has a particularly high velocity. Consequently, the resulting static pressure differences at the sinus ostium may provide an appropriate ventilation of the sinuses.
- The omnidirectional inspirational flow in contrast to the directed expirational flow refutes the widely held belief that a hanging tip automatically induces nasal obstruction.

Finally, it should be noted that the analysis of the flow data was quite laborious due to the lack of initial hypotheses, thus requiring a visual exploration of *all* attributes. Future work might involve research on methods comparable to VisTrails [10]. Furthermore, the unfamiliar multi-view setup initially posed challenges to our domain experts. Still after a training period, the ability to interactively define different queries while immediately being able to assess results was highly appreciated and significantly improved the investigation.

In our future work we shall address the extraction of Lagrangian features and coherent structures. This might allow us to confirm our current findings and investigate further details, as for example the difference between regular and irregular flow.

CHAPTER 5

Summary and Conclusions

The numerical simulation of physical processes has become an ubiquitous tool in many different research and engineering fields. This has been facilitated by the broad availability of processing resources and the ongoing evolution of simulation tools. In order to keep up with these developments interactive visualization techniques that can be used to explore and analyze complex simulation results have to be scalable with respect to different data set properties. This thesis has presented several techniques that address different sets of scalability issues for volumetric, possibly time dependent, simulation data sets. First two interactive GPU-based volume rendering techniques are discussed in Chapter 2. Section 2.3 introduces a hybrid raycasting system that scales very well with respect to the number of cells in an unstructured grid. A level of detail method based on resampling is used to allow for interactive rendering even for data sets that are too large for GPU memory. Volume rendering is combined with different surface rendering techniques in order to provide a focus+context style visualization. This is especially important for simulation results that include objects of high geometric complexity.

The subsequent section presents an unstructured grid representation called two sided face sequence lists (TSFSL) and an accompanying raycasting method that can be used to eliminate most of the restrictions of the original hybrid raycasting approach. The TSFSL data structure is highly memory efficient as well as extremely flexible with respect to cell and mesh geometry. The former allows for greater scalability with respect to mesh size whereas the latter facilitates scalability with respect to cell and mesh complexity. Therefore the proposed visualization methods can directly deal with large data sets from state of the art simulation tools that rely on meshes that comprise general polyhedral cells. Both techniques presented in Chapter 2 are brick based. The original mesh is divided into sub grids, each of which can be handled individually. This property facilitates easy parallelization across multiple graphics cards or even work stations, which in turn again adds another layer of scalability to the proposed techniques.

While Chapter 2 primarily deals with volume rendering, Chapter 3 addresses scalability issues that arise when interactive information visualization techniques are applied to scientific data such as results from numerical simulations. Here the primary challenges are computational as well as visual scalability. Computational scalability, that is the ability to process data sets

which contain a large number of data items, is achieved by using clustering as well as image order algorithms. Section 3.3 introduces a time series visualization that is capable of displaying function graphs of one scalar attribute over time for each cell of a large volumetric data set. Visual scalability is related to the capability of a method to generate meaningful result images even for a large number of data items. This is addressed for line-based visualization methods by two techniques that are introduced in Chapter 3. Line-based visualizations such as parallel coordinates or the time series visualization introduced in Section 3.3 use lines as primary rendering primitive to represent individual data items. This leads to cluttering and overdrawing issues if hundreds of thousands or even millions of lines have to be drawn. Therefore discerning individual lines or even larger trends becomes increasingly difficult. The proposed techniques compute statistical information about line directions passing through each image pixel. This information is then used to steer texture-based flow visualization methods in order to convey the general line orientations in the result images. By using this approach line-based visualization methods can provide a proper overview on general trends even for extremely large data sets. The four level focus+context coloring approach that has been introduced in Section 3.3 is used to highlight smaller subsets of a data set or even individual data items throughout different information visualization views. It therefore allows a user to define complex features in a data set across multiple views.

All aforementioned techniques have been integrated into the SimVis framework that can be use to interactively explore and analyze data generated by complex numerical simulations. Chapter 4 demonstrates how the visualization methods that have been introduced in this thesis can be applied to a large time dependent CFD data set in a concerted effort. The CFD data has been generated by a model of the human breathing cycle for the nasal cavities, the pharynx and the trachea. The time dependent volumetric simulation results contain information about the air flow as well as about different quantities such as temperature and humidity. The underlying unstructured mesh is highly complex due to the geometric complexity of the nasal cavities. The proposed interactive volume rendering techniques have been used to display different features within the data, each of which has been defined in multiple information visualization views. Linking across views is provided by the four level focus+context coloring scheme. Since nasal breathing is not fully understood, application experts have used the SimVis framework for an initial assessment of the data. This has lead to several interesting hypotheses that have been further explored in additional more in-depth analysis sessions. As illustrated by this example, the techniques proposed in this thesis have been an important addition to the state of the art in interactive visualization. This has also been proven in the context of multiple additional research collaborations with scientists from different fields such as medicine, climate research and engineering.

Acknowledgements

All the work that has been condensed into this PhD thesis whould not have been possible without many collegues and friends. First of all I would like to thank my supervisors (Meister) Eduard Gröller and Markus Hadwiger as well as Helmut Doleisch. Thank you Meister, for giving me the opportunity to work at the VisGroup in Vienna and for always providing me with quick feedback and guidance during times when I was in dire need of counsel from a knowledgeable, wise, accomplished, professional, and seasoned person. Markus, I am very greatful for all the support you provided when working on some of the papers that are included in this thesis. Your knack for untangling my confusing/disorganized ideas really helped a lot. I also want to thank Helmut for providing me with a work environment first at the VRVis and later at the SimVis GmbH that allowed me to successfully pursue my research, and Helwig Hauser for introducing me to the field of visualization and getting me involved in scientific work.

Many thanks go to my collegues and friends Wolfgang Freiler, Wolfgang Berger, Johanna Beyer, Rafael Fuchs, Johannes Kehrer, Harald Piringer, Steffen Hadlak, Bernhard Petracek, Michaela Hofer and Michael Kusternig who either directly contributed to this work or provided very welcome distractions in times of frustration or stress. Since I am a somewhat quiet person I probably do not express this often enough, but I have been really greatful for your company during the last few years! I would also like to thank Steffen Oeltze, Bernhard Preim, Stefan Zachow, Thomas Hildebrandt, and Hans Christian Hege who co-authored the publications which have been compiled into this thesis.

Last but not least I would like to extend my gratitude towards my family. Thank you for always supporting me in my decision for pursuing my interests and for not bothering me (too often) about the completion date of my PhD.

This thesis encompasses work that has been carried out at the VRVis Research Center, the Vienna University of Technology as well as SimVis GmbH in the context of different projects. The Austrian Research Funding Agency provided funding via the Severe Weather Explorer, the MulSimVis, and the AutARG projects. Since the presented methods are very general, data from many sources has been used throughout this work. Simulation results from different engineering applications have been provided by the Institute for Internal Combustion Engines and Thermodynamics, University of Technology Graz, Austria, AVL List GmbH, Graz, Austria, Arsenal Research, Vienna, Austria and by Traktionssysteme Austria (contact: DDr. Neudorfer). The polyhedral data that is used throughout Section 2.4 has been supplied by CD-Adapco. I extend my thanks to Hermann Lang, Markus Trenker, Stephan Schmidt, Oliver Schögl, Eberhard Schreck, and Julian Gänz for granting me access to the data as well as for providing information about the corresponding applications. The simulation data, which has been explored in

Chapter 4, has been provided by Alexander Steinmann from CFX Berlin Software GmbH. The medical perfusion data shown in Section 3.3 has been provided by Andreas Fessel from the University Hospital Magdeburg and Jonathan Wiener from Boca Raton Comm. Hospital, Florida.

Bibliography

[1] Fluent by Ansys. See URL: http://www.fluent.com, last visited 2011.

[2] OpenFOAM. See URL: http://www.openfoam.com, last visited 2011.

[3] Star-CCM+ by CD-adapco. See URL: http://www.cd-adapco.com/products/star_ccm_plus, last visited 2011.

[4] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visual methods for analyzing time-oriented data. *IEEE TVCG*, 14(1):47–60, 2008.

[5] H. Akiba and K. L. Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proc. of EuroVis07: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 115–122, 2007.

[6] T. J. Alumbaugh and X. Jiao. Compact array-based mesh data structures. In *Proc. of the 14th International Meshing Roundtable, IMR 2005, September 11-14, 2005, San Diego, CA, USA*, pages 485–503, 2005.

[7] S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE TVCG*, 14(6):1428–1435, 2008.

[8] N. Bailie, H. Brendan, J. Watterson, and G. Gallagher. An overview of numerical modeling of nasal airflow. *Rhinology*, 44:53–57, 2006.

[9] H. B. Baumgardt. A polyhedron representation for computer vision. In *Proc. of AFIPS National Conference*, pages 589–596, 1975.

[10] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, and H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *Proc. of IEEE Visualization 2005*, pages 135–142, 2005.

[11] F. F. Bernardon, C. A. Pagot, J. L. D. Comba, and C. T. Silva. GPU-based tiled ray casting using depth peeling. *J. Graphics Tools*, 11(4):1–16, 2006.

[12] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(3):185–197, 2001.

[13] C. A. Brewer. Color use guidelines for data representation. In *Proc. of Section on Statistical Graphics, American Statistical Association, Alexandria VA.*, pages 55–60, 1999.

[14] K. L. Briggman and W. Denk. Towards neural circuit reconstruction with volume electron microscopy techniques. *Current Opinion in Neurobiology*, 16(5):562–570, 2006.

[15] A. Bru and M. Teillaud. Generic implementation of a data structure for 3d regular complexes. In *Abstracts of 24th European Workshop on Computational Geometry*, pages 95–98, 2008.

[16] P. Buono, A. Aris, C. Plaisant, A. Khella, and B. Shneiderman. Interactive pattern search in time series. In *Proc. IST/SPIE's 17th Ann. Intl. Symp. Electronic Imaging (VDA '05)*, volume 5669, pages 175–186, 2005.

[17] R. Bürger, P. Muigg, Martin Ilcik, Helmut Doleisch, and Helwig Hauser. Integrating local feature detectors in the interactive visual analysis of flow simulation data. In *Proc. of EuroVis07: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 171–178, 2007.

[18] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proc. of ACM SIGGRAPH '93*, pages 263–272, 1993.

[19] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum (Eurographics '99)*, 18(3):359–368, 1999.

[20] S. P. Callahan, L. Bavoil, V. Pascucci, and C. T. Silva. Progressive volume rendering of large unstructured grids. *IEEE TVCG*, 12(5):1307–1314, 2006.

[21] S. P. Callahan, J. L. D. Comba, P. Shirley, and C. T. Silva. Interactive rendering of large unstructured grids using dynamic level-of-detail. In *Proc. of IEEE Visualization 2005*, pages 199 – 206, 2005.

[22] S. P. Callahan, M. Ikits, J. Luiz D. Comba, and C. T. Silva. Hardware-assisted visibility sorting for unstructured volume rendering. *IEEE TVCG*, 11(3):285–295, 2005.

[23] Y. J. Chiang and X. Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. *Computer Graphics Forum*, 22(3):493–504, 2003.

[24] H. Childs, M. Duchaineau, and K. L. Ma. A scalable, hybrid scheme for volume rendering massive data sets. In *Proc. of Eurographics Symposium on Parallel Graphics and Visualization*, pages 153–161, 2006.

[25] P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE TVCG*, 10(1):29–45, 2004.

[26] E. Coto, S. Grimm, S. Bruckner, E. Gröller, A. Kanitsar, and O. Rodriguez. MammoExplorer: An Advanced CAD Application for Breast DCE-MRI. In *Proc. Vision, Modelling, and Visualization (VMV)*, pages 91–98, 2005.

[27] J.A. den Boer and P.J.M. Folkers. MR perfusion and diffusion imaging in ischaemic brain disease. *Medica Mundi*, 41(2):20–35, 1997.

[28] U. Diewald, T. Preußer, and M. Rumpf. Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces. *IEEE TVCG*, 6(2):139–149, 2000.

[29] H. Doleisch. *Visual Analysis of Complex Simulation Data using Multiple Heterogenous Views*. PhD thesis, Vienna University of Technology, Austria, 2004.

[30] H. Doleisch. SIMVIS: interactive visual analysis of large and time-dependent 3D simulation data. In *Proc. of Winter Simulation Conference*, pages 712–720, 2007.

[31] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proc. of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)*, pages 239–248, 2003.

[32] H. Doleisch and H. Hauser. Smooth brushing for focus+context visualization of simulation data in 3D. In *WSCG 2002 Conference Proceedings*, pages 147–154, 2002.

[33] H. Doleisch, M. Mayer, M. Gasser, P. Priesching, and H. Hauser. Interactive feature specification for simulation data on time-varying grids. In *Simulation und Visualisierung 2005 (SimVis 2005), 3-4 März 2005, Magdeburg*, pages 291–304, 2005.

[34] H. Doleisch, M. Mayer, M. Gasser, R. Wanker, and H. Hauser. Case study: Visual analysis of complex, time-dependent simulation results of a diesel exhaust system. In *Proc. of the 6th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2004)*, pages 91–96, 343, 2004.

[35] H. Doleisch, P. Muigg, and H. Hauser. Interactive visual analysis of hurricane isabel with simvis. Technical Report TR-VRVis-2004-058, VRVis Research Center, Vienna Austria, 2004.

[36] D. Doorly, D. J. Taylor, P. Franke, and R. C. Schroter. Experimental investigation of nasal airflow. *Proc. IMechE 2008, Part H, J. Engineering in Medicine*, 222:439–453, 2008.

[37] D. J. Doorly, D. J. Taylor, A. M. Gambaruto, R. C. Schroter, and N. Tolley. Nasal architecture: form and flow. *Phil. Trans. R. Soc. A*, 366:3225–3246, 2008.

[38] E.Bertini and G.Santucci. Give chance a chance: modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.

[39] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.

[40] G. Ellis and A. J. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE TVCG*, 13(6):1216–1223, 2007.

[41] R. Espinha and W. C. Filho. High-quality hardware-based ray-casting volume rendering using partial pre-integration. In *Proc. of SIBGRAPI*, pages 273–280, 2005.

[42] C. Everitt. Interactive order-independent transparency, September 01 2001.

[43] M. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[44] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *Proc. of IEEE Visualization '99*, pages 43–50, 1999.

[45] R. Fuchs. *The Visible Vortex - Interactive Analysis and Extraction of Vortices in Large Time-Dependent Flow Data Sets*. PhD thesis, Vienna University of Technology, Austria, 2008.

[46] G. P. Galdi, R. Rannacher, A. M. Robertson, and Stefan Turek. *Hemodynamical Flows: Modeling, Analysis and Simulation (Oberwolfach Seminars)*. Birkhäuser, 1. edition, 2007.

[47] G. J. M. Garcia, N. Bailie, D. A. Martins, and J. S. Kimbell. Atrophic rhinitis: a CFD study of air conditioning in the nasal cavity. *J Appl Physiol*, 103:1082–1092, 2007.

[48] M. P. Garrity. Raytracing irregular volume data. *ACM Computer Graphics*, 24(5):35–40, 1990.

[49] J. Georgii and R. Westermann. A generic and scalable pipeline for GPU tetrahedral grid rendering. *IEEE TVCG*, 12(5):1345–1352, 2006.

[50] D. L. Gresh, B. E. Rogowitz, R. L. Winslow, D. F. Scollan, and C. K. Yung. Weave: a system for visually linking 3-D and statistical visualizations, applied to cardiac simulation and measurement data. In *Proc. of IEEE Visualization '00*, pages 489–492, 2000.

[51] M. Gross, H. Hagen, and F.-J. Pfreund. Interactive simd ray tracing for large deformable tetrahedral meshes. In *Proc. of IEEE Symposion on Interactive Ray Tracing*, pages 147 –154, 2008.

[52] T. Gurung and J. Rossignac. Sot: compact representation for tetrahedral meshes. In *Proc. of 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 79–88, 2009.

[53] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[54] H. Hauser. Generalizing focus+context visualization. In *Scientific Visualization: The Visual Extraction of Knowledge from Data*, Mathematics and Visualization, pages 305–327. Springer Berlin Heidelberg, 2006.

[55] H. Hauser, F. Ledermann, and H. Doleisch. Angular brushing of extended parallel coordinates. In *Proc. IEEE Symposium on Information Visualization 2002 (InfoVis 2002)*, pages 127–130, 2002.

[56] J. Heinrich and D. Weiskopf. Continuous parallel coordinates. *IEEE TVCG*, 15(6):1531–1538, 2009.

[57] A. Helgeland and O. Andreassen. Visualization of vector fields using seed LIC and volume rendering. *IEEE TVCG*, 10(6):673–682, 2004.

[58] S.H. Heywang-Köbrunner, P. Viehweg, A. Heinig, and C. Kuchler. Contrast-enhanced MRI of the breast: accuracy, value, controversies, solutions. *European Journal of Radiology*, 24(2):94–108, 1997.

[59] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[60] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–748, 2006.

[61] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[62] A. Inselberg. The plane with parallel coordinates. *Visual Computer*, 1(4):69–91, 1985.

[63] S. Ishikawa, T. Nakayama, M. Watanabe, and T. Matsuzawa. Visualization of flow resistance in physiological nasal respiration - analysis of velocity and vorticities using numerical simulation. *Arch Otolarygol Head Neck Surg*, 132:1203–1209, 2006.

[64] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics Digital Archive*, 285(-1):69–94, 1995.

[65] J. Johansson, P. Ljung, and M. Cooper. Depth cues and density in temporal parallel coordinates. In *Proc. of EuroVis07: Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 35–42, 2007.

[66] J. Johansson, P. Ljung, M. Jern, and M. Cooper. Revealing structure in visualizations of dense 2d and 3d parallel coordinates. *Information Visualization*, 5(2):125–136, 2006.

[67] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *Proc. of SIGGRAPH 2005*, pages 561–566, 2005.

[68] J. Kehrer. *Interactive Visual Analysis of Multi-faceted Scientific Data*. PhD thesis, University of Bergen, Norway, 2011.

[69] J. Kehrer, P. Filzmoser, and H. Hauser. Brushing moments in interactive visual analysis. *Computer Graphics Forum*, 29(3):813–822, june 2010.

[70] J. Kehrer, F. Ladstädter, P. Muigg, H. Doleisch, A. Steiner, and H. Hauser. Hypothesis generation in climate research with interactive visual data exploration. *IEEE TVCG*, 14(6):1579–1586, 2008.

[71] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02)*, 2002.

[72] L. Kettner. Designing a data structure for polyhedral surfaces. In *Proc. of Symposium on Computational Geometry*, pages 146–154, 1998.

[73] G. Kindlmann. Superquadric tensor glyphs. In *Proc. of the 6th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2004)*, pages 147–154, 2004.

[74] Z. Konyha, K. Matkovic, D. Gracanin, M. Jelovic, and H. Hauser. Interactive visual analysis of families of function graphs. *IEEE TVCG*, 12(6):1373–1385, 2006.

[75] M. Kraus and T. Ertl. Cell-projection of cyclic meshes. In *Proc. of IEEE Visualization 2001*, pages 215–222, 2001.

[76] M. Kraus and T. Ertl. Topology-Guided downsampling. pages 223–234, 2001.

[77] M. Kraus and T. Ertl. Simplification of nonconvex tetrahedral meshes. In *Hierarchical and Geometrical Methods in Scientific Visualization*, pages 185–196. Springer-Verlag, 2002.

[78] J. Krüger, J. Schneider, and R. Westermann. ClearView: An interactive context preserving hotspot visualization technique. In *Proc. of IEEE Visualization 2006*, pages 941–948, 2006.

[79] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proc. of IEEE Visualization 2003*, pages 287–292, 2003.

[80] V. Kulish and M. Kanoh. *Human Respiration: Anatomy And Physiology, Mathematical Modeling, Numerical Simulation And Applications (Advances in Bioengineering, No 3)*. WIT Press, 2006.

[81] Y. L., J. Kopf, D. Cohen-Or, and D. Levin. GPU-assisted positive mean value coordinates for mesh deformations. In *Proc. of the Fifth Eurographics Symposium on Geometry Processing*, pages 117–123, 2007.

[82] M. Lage, T. Lewiner, H. Lopes, and L. Velho. CHF: A scalable topological data structure for tetrahedral meshes. In *Proc. of SIBGRAPI*, pages 349–356, 2005.

[83] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proc. of IEEE Visualization '99*, pages 355–361, 1999.

[84] T. Langer, A. Belyaev, and H. P. Seidel. Spherical barycentric coordinates. In *Proceedings of Eurographics Symposium on Geometry Processing 2006*, pages 81–88, 2005.

[85] R. S. Laramee, C. Garth, H. Doleisch, J. Schneider, H. Hauser, and H. Hagen. Visual Analysis and Exploration of Fluid Flow in a Cooling Jacket. In *Proc. of IEEE Visualization 2005*, pages 623–630, 2005.

[86] R. S. Laramee, H. Hauser, H. Doleisch, F. H. Post, B. Vrolijk, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[87] D.J. Lehmann and H. Theisel. Discontinuities in continuous scatter plots. *IEEE TVCG*, 16(6):1291–1300, 2010.

[88] J. Leven, J.Corso, J. Cohen, and S. Kumar. Interactive visualization of unstructured grids using hierarchical 3D textures. In *Proc. of VolVis 2002*, pages 37–44, October 28–29 2002.

[89] B. Lévy, G. Caumon, S. Conreaux, and X. Cavin. Circular incident edge lists: a data structure for rendering complex unstructured grids. In *Proc. of IEEE Visualization 2001*, pages 191–198, 2001.

[90] J. Lindemann, H. J. Brambs, T. Keck, K. Wiesmiller, G. Rettinger, and D. Pless. Numerical simulation of intranasal airflow after radical sinus surgery. *American Journal of Otolaryngology–Head and Neck Medicine and Surgery*, 26:175–180, 2005.

[91] J. Lindemann, T. Keck, K. Wiesmiller, B. Sander, H. J. Brambs, G. Rettinger, and D. Pless. Nasal air temperature and airflow during respiration in numerical simulation based on multislice computed tomography scan. *Am J Rhinol*, 20:219–223, 2006.

[92] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169, 1987.

[93] G. Marmitt, H. Friedrich, and P. Slusallek. Interactive Volume Rendering with Ray Tracing. In *Eurographics State of the Art Reports*, 2006.

[94] G. Marmitt and P. Slusallek. Fast ray traversal of tetrahedral and hexahedral meshes for direct volume rendering. In *Proc. of the 8th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2006)*, pages 235–242, 2006.

[95] N. Max, P. Williams, C. T. Silva, and R. Cook. Volume rendering for curvilinear and unstructured grids. In *Proc. of Computer Graphics International*, pages 210–215, 2003.

[96] N. L. Max. Optical models for direct volume rendering. *IEEE TVCG*, 1(2):99–108, 1995.

[97] A. Maximo, S. Ribeiro, C. Bentes, A. Oliveira, and R. Farias. Memory Efficient GPU-Based Ray Casting for Unstructured Volume Rendering. pages 155–162, 2008.

[98] K. T. McDonnell and K. Mueller. Illustrative parallel coordinates. *Computer Graphics Forum*, 27(3):1031–1038, 2008.

[99] G. Mlynski. *Physiology and Pathophysiology of Nasal Breathing*, pages 75–87. Thieme, 2004.

[100] P. Muigg. Hybrid multiresolution unstructured and structured grid volume raycasting. Master's thesis, Vienna University of Technology, Austria, February 2007.

[101] P. Muigg, M. Hadwiger, H. Doleisch, and E. Gröller. Interactive volume visualization of general polyhedral grids. *IEEE TVCG*, 17(12):2115–2124, 2011.

[102] P. Muigg, M. Hadwiger, H. Doleisch, and E. Gröller. Visual coherence for large-scale line-plot visualizations. *Computer Graphics Forum*, 30(3):643–652, 2011.

[103] P. Muigg, M. Hadwiger, H. Doleisch, and H. Hauser. Scalable hybrid unstructured and structured grid raycasting. *IEEE TVCG*, 13(6):1592–1599, 2007.

[104] P. Muigg, J. Kehrer, S. Oeltze, H. Piringer, H. Doleisch, B. Preim, and H. Hauser. A four-level focus+context approach to interactive visual – analysis of temporal features in large scientific data. *Computer Graphics Forum*, 27(3):775–782, 2008.

[105] H. Müller and M. Wehle. Visualization of implicit surfaces using adaptive tetrahedrizations. *Scientific Visualization Conference*, 0:243, 1997.

[106] M. Novotny and H. Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *IEEE TVCG*, 12(5):893–900, 2006.

[107] S. Oeltze. *Visual Exploration and Analysis of Perfusion Data*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Germany, August 2010.

[108] S. Oeltze, H. Doleisch, H. Hauser, P. Muigg, and B. Preim. Interactive visual analysis of perfusion data. *IEEE TVCG*, 13(6):1392–1399, 2007.

[109] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE TVCG*, 5(3):238–250, 1999.

[110] D. Pless, T. Keck, K. Wiesmiller, G. Rettinger, A. J. Aschoff, T. R. Fleiter, and J. Lindemann. Numerical simulation of air temperature and airflow patterns in the human nose during expiration. *Clin. Otolaryngol.*, 29:642–647, 2004.

[111] F. H. Post. The state of the art in flow visualization: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22(4), pages 775–792. 2003.

[112] G. Robertson, D. Ebert, S. Eick, D. Keim, and K. Joy. Scale and complexity in visual analytics. *Information Visualization*, 8(4):247–253, December 2009.

[113] S. Röttger, S. Guthe, A. Schieber, and T. Ertl. Convexification of unstructured grids. In *Proc. of Workshop on Vision, Modeling and Visualization (VMV 2004)*, pages 283–292, 2004.

[114] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proc. of IEEE Visualization 2000*, pages 109–116, 2000.

[115] K. Ryall, N. Lesh, T. Lanning, D. Leigh, H. Miyashita, and S. Makino. Querylines: approximate query for visual browsing. In *Extended abstracts on Human factors in computing systems (CHI '05)*, 2005.

[116] I. A. Sadarjoen, T. van Walsum, A. J. S. Hin, and F. H. Post. Particle tracing algorithms for 3D curvilinear grids. In *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 311–335. IEEE Computer Society, 1997.

[117] T. Shakked, D. Katoshevski, D. M. Broday, and I. Amirav. Administration of aerosolized drugs to infants by a hood. *Journ. of Aerosol Medicine*, 19(4):533–542, 2006.

[118] N. Shareef, T.-Y. Lee, H.-W. Shen, and K. Mueller. An image-based modelling approach to GPU-based rendering of unstructured grids. pages 31–38, 2006.

[119] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990.

[120] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[121] C. T. Silva, J. S. B. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proc. of VolVis '98*, pages 87–94, 1998.

[122] D. Stalling, M. Westerhoff, and H. C. Hege. Amira: A highly interactive system for visual data analysis. In *The Visualization Handbook*, pages 749–767. Elsevier, 2005.

[123] D. Stalling, M. Zöckler, and H. C. Hege. Fast display of illuminated field lines. *IEEE TVCG*, 3(2):118–128, 1997.

[124] C. M. Stein, B. G. Becker, and N. L. Max. Sorting and hardware assisted rendering for volume visualization. In *Proc. of VolVis '94*, pages 83–89, 1994.

[125] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, and T. Ertl. Hierarchical visualization and compression of large volume datasets using GPU clusters. In *5th Eurographics/ACM SIGGRAPH Symposium on Parallel Graphics and Visualization (EGPGV 2004)*, pages 41–48, 2004.

[126] A. Telea and R. Strzodka. Multiscale image based flow visualization. In *Proc. of SPIE-IS&T Electronic Imaging, Visualization and Data Analysis (VDA) 2006*, volume 6060, pages 1–11, 2006.

[127] M. Trenker, H. Lang, P. Muigg, and H. Doleisch. Validation of vortex flow phenomena in electrical machinery using advanced simulation and visualization techniques. In *Proceedings of the NAFEMS Worldcongress*, page full Proceedings on CDROM, 2007.

[128] D. Uesu, L. Bavoil, S. Fleishman, J. Shepherd, and C. T. Silva. Simplification of unstructured tetrahedral meshes by point sampling. In *Proc. of VolVis 2005*, pages 157–165, 2005.

[129] M. Üffinger, S. Frey, and T. Ertl. Interactive high-quality visualization of higher-order finite elements. *Computer Graphics Forum*, 29(2):337–346, 2010.

[130] J. van Wijk. Spot noise: Texture synthesis for data visualization. In *Proceedings SIGGRAPH '91*, pages 309–318, 1991.

[131] I. Viola, M. Feixas, M. Sbert, and M. E. Gröller. Importance-driven focus of attention. In *Proc. of IEEE Visualization 2006*, pages 933–940, 2006.

[132] H. T. Vo, S. P. Callahan, N. Smith, C. T. Silva, W. Martin, D. Owen, and D. Weinstein. iRun: Interactive rendering of large unstructured grids. In *Proc. of Eurographics Symposium on Parallel Graphics and Visualization*, pages 93–100, 2007.

[133] M. Wattenberg. Sketching a graph to query a time-series database. In *Extended abstracts on Human factors in computing systems (CHI '01)*, pages 381–382. ACM Press, 2001.

[134] E. J. Wegman and Q. Luo. High dimensional clustering using parallel coordinates and the grand tour. *Computing Science and Statistics*, 28:352–360, 1997.

[135] J. Weickert. *Anisotropic Diffusion in Image Processing*. Teubner-Verlag, 1998.

[136] J. Weickert and H. Hagen. *Visualization and Processing of Tensor Fields*. Springer, 2006.

[137] K. Weiler. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[138] M. Weiler and T. Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proc. of IEEE Visualization 2001*, pages 199–206, 2001.

[139] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proc. of IEEE Visualization 2003*, pages 333–340, 2003.

[140] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based view-independent cell projection. *IEEE TVCG*, 9(2):163–175, 2003.

[141] M. Weiler, P. N. Mallón, M. Kraus, and T. Ertl. Texture-encoded tetrahedral strips. In *Proc. of VolVis 2004*, pages 71–78, 2004.

[142] M. Weiler, R. Westermann, C. D. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *Proc. of VolVis 2000*, pages 7–13, 2000.

[143] R. Westermann. The rendering of unstructured grids revisited. In *Proc. of the 3rd Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2001)*, pages 65–74, 2001.

[144] P. L. Williams. Visibility-ordering meshed polyhedra. *ACM Trans. Graph.*, 11(2):103–126, 1992.

[145] A. P. Witkin. Scale-space filtering. In *Proc. International Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

[146] C. K. Yang and T. Chiueh. An integrated pipeline of decompression, simplification and rendering for irregular volume data. In *Proc. of VolVis 2005*, pages 147–155, 2005.

[147] S. Yu, Y. Liu, X. Sun, and S. Li. Influence of nasal structure on the distribution of airflow in nasal cavity. *Rhinology*, 46:137–143, 2008.

[148] S. Zachow, P. Muigg, T. Hildebrandt, H. Doleisch, and H. C. Hege. Visual exploration of nasal airflow. *IEEE TVCG*, 15:1407–1414, November 2009.

[149] S. Zachow, A. Steinmann, T. Hildebrandt, and W. Heppt. Understanding nasal airflow via CFD simulation and visualization. In *Proc. Computer Aided Surgery Around the Head*, pages 173–176. Pro Business Verlag, 2007.

[150] S. Zachow, A. Steinmann, Th. Hildebrandt, R. Weber, and W. Heppt. CFD simulation of nasal airflow: treatment planning for functional rhinosurgery. In *Computer Assisted Radiology and Surgery*, pages 165–167, Osaka, Japan, 2006. Elsevier.

[151] K. Zhao, P. Dalton, G. C. Yang, and P. W. Scherer. Numerical modeling of turbulent and laminar airflow and odorant transport during sniffing in the human and rat nose. *Chem. Senses*, 31:107–118, 2006.

[152] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE TVCG*, 12(5):1229–1236, 2006.

Curriculum Vitae

Personal:

Philipp Muigg was born on October 25th, 1981, in Vienna, Austria, as the first son of Mag. Elisabeth Muigg (maiden name Knoll) and Dipl. Ing. Alfred Muigg.

Contact:

Address: Breitenfurterstraße 376/6/13, 1230 Vienna, Autria
E-Mail: muigg@simvis.at
Phone: +43(1)2050167200

Professional Activities:

- 1999, 2000, 2003: **Holiday intern** at PSE, Siemens Austria
- 2004/Oct.-2005/Jan.: **Tutor (teaching assistant)** at the Institute of Computer Graphics, Vienna University of Technology, Austria (refs.: Prof. Helwig Hauser), http://www.cg.tuwien.ac.at/
- 2004/Sept.-2007/March: **Software engineer** (department "Virtual Interactive Analysis") in the VRVis Research Center in Vienna, Austria (refs.: VRVis-CEO DI Georg Stonawski, Prof. Eduard Gröller), http://www.VRVis.at/
- 2007/March-2008/June: **Junior researcher** (department "Virtual Interactive Analysis") in the VRVis Research Center in Vienna, Austria (refs.: VRVis-CEO DI Georg Stonawski, Prof. Eduard Gröller), http://www.VRVis.at/
- Since 2009/April: **Project Assistant** at the Institute of Computer Graphics and Algorithms at the Vienna University of Technology for the FFG project AUTarg under supervision of Associate Professor Eduard Groeller, http://www.cg.tuwien.ac.at
- Since 2009/April: **Software engineer** leading the software development process of the SimVis framework at SimVis GmbH, Vienna, Austria (refs.: SimVis-CEO DI Dr Helmut Doleisch), http://www.simvis.at

Activities related to scientific work:

- **diploma thesis project** during my studies of computer science at theVRVis Research Center in Vienna, Austria, about "Hybrid Multi Resolution Structured and Unstructured Grid Raycasting" [100],
- **Bachelors thesis project** at the VRVis Research Center in Vienna, Austria, about "Feature Extraction and Visualization in the SimVis System"
- **Junior researcher** in the FFG-funded bridge projects "MulSimVis" and "Severe Weather Explorer"
- Co author of the **1st place** entry to the "IEEE Visualization Contest" (held in the course of the annual IEEE Visualization Conference), Austin, Texas, USA, October 2004. An accompanying webpage including the submission is available at http://www.SimVis.at/Isabel/
- **Reviewer for the conferences** IEEE Visualization (2007, 2008, 2009, 2010), IEEE Symposium on Information Visualization (2007, 2008), VMV (2009), Volume Graphics (2008), EuroVis (2009), Eurographics (2010), PacificGraphics (2010), PacificVis (2011), EuroVis (2011), IVAPP (2012)

Activities related to teaching:

- Technical supervisor of many **student projects** and **seminar works** in the studies of computer science, including those of S. Fischer, S. Dumke, M. Braun, S. Tomitsch, M. Brunnhuber, J. Kehrer, and M. Ilcik
- **Teaching assistant (Tutor)** at the Institute of Computer Graphics, Vienna University of Technology in 2004/2005 for the lab "Visualization"

School:

- 1988/Sept.-1992/June: **Volksschule** (primary school) in Perchtoldsdorf, Austria (four years)
- 1992/Sept.-2000/June: **Bundesrealgymnasium** (combined secondary school and high school) in Perchtoldsdorf, Austria (eight years). Graduation (Matura) with highest distinction on June, 2000

Studies:

- 2001/Oct.-2005/March: **Studies of Software and Information Engineering (Bacc.)** at Vienna University of Technology, Austria.
- 2005/March-2007/March: **Studies of Computergraphics and Digital Image Processing (Masters)** at Vienna University of Technology, Austria, with special focus on computer graphics.

- 2006/June-2007/March: Member of the **Computer Graphics Student Club** at the Institute of Computer Graphics, Vienna University of Technology

- 2007/March-2008/April: Member of the **scientific staff** at the VRVis Research Center in Vienna, Austria

- Since 2008/April: PhD student and member of the **scientific staff** at the Institute for Computer Graphics and Algorithms of the Vienna University of Technology, Austria

Reviewed Publications:

- Philipp Muigg, Markus Hadwiger, Helmut Doleisch, Eduard Gröller: Visual Coherence for Large-Scale Line-Plot Visualizations. In *Computer Graphics Forum*, 30(3), 2011, pp. 643-652

- Philipp Muigg, Markus Hadwiger, Helmut Doleisch, Eduard Gröller: Interactive Volume Visualization of General Polyhedral Grids. In *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2011, pp. 2115-2124

- Johannes Kehrer, Philipp Muigg, Helmut Doleisch, Helwig Hauser: Interactive Visual Analysis of Heterogeneous Scientific Data across an Interface. In *IEEE Transactions on Visualization and Computer Graphics*, 17(7), 2011, pp. 934-946

- Florian Ladstädter, Andrea K. Steiner, Bettina C. Lackner, Barbara Pirscher, Gottfried Kirchengast, Johannes Kehrer, Helwig Hauser, Philipp Muigg, Helmut Doleisch: Exploration of Climate Data Using Interactive Visualization. In *Journal of Atmospheric and Oceanic Technology*, 27(4), 2010, pp. 667-679

- Harald Piringer, Christian Tominski, Philipp Muigg, Wolfgang Berger: A Multi-Threading Architecture to Support Interactive Visual Exploration. In *IEEE Transactions on Visualization and Computer Graphics*, 15(6), 2009, pp. 1113-1120

- Stefan Zachow, Philipp Muigg, Thomas Hildebrandt, Helmut Doleisch, Hans-Christian Hege: Visual Exploration of Nasal Airflow. In *IEEE Transactions on Visualization and Computer Graphics*, 15(6), 2009, pp. 1407-1414

- Florian Ladstädter, Andreas Steiner, Bettina Lackner, Gottfried Kirchengast, Philipp Muigg, Johannes Kehrer, and Helmut Doleisch: SimVis: An Interactive Visual Field Exploration Tool Applied to Climate Research. In New Horizons in Occultation Research: Studies in Atmosphere and Climate. In *Proc of OPAC-3*, pp.235-245, 2009

- Andrea Unger, Philipp Muigg, Helmut Doleisch, Heidrun Schumann: Visualizing Statistical Properties of Smoothly Brushed Data Subsets. In *Proc. of 12th International Conference on Information Visualization*, 2008, pp. 233-239

- Johannes Kehrer, Florian Ladstädter, Philipp Muigg, Helmut Doleisch, Andrea Steiner, Helwig Hauser: Hypothesis Generation in Climate Research with Interactive Visual Data Exploration. In *IEEE Transactions on Visualization and Computer Graphics*, 14(6), 2008, pp. 1579-1586

- Raphael Fuchs, Ronald Peikert, Helwig Hauser, Filip Sadlo, Philipp Muigg: Parallel Vectors Criteria for Unsteady Flow Vortices. In *IEEE Transactions on Visualization and Computer Graphics*, 14(3), 2008, pp. 615-626

- Philipp Muigg, Johannes Kehrer, Steffen Oeltze, Harald Piringer, Helmut Doleisch, Bernhard Preim, Helwig Hauser: A Four-level Focus+Context Approach to Interactive Visual Analysis of Temporal Features in Large Scientific Data. In *Computer Graphics Forum*, 27(3), 2008, pp. 775-782

- Philipp Muigg, Markus Hadiwger, Helmut Doleisch, Helwig Hauser: Scalable Hybrid Unstructured and Structured Grid Raycasting. In *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007, pp. 1592-1599

- Steffen Oeltze, Helmut Doleisch, Helwig Hauser, Philipp Muigg, Bernhard Preim: Interactive Visual Analysis of Perfusion Data. In *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007, pp. 1392-1399

- Raphael Bürger, Philipp Muigg, Helmut Doleisch, Helwig Hauser: Interactive cross-detector analysis of vortical flow data. In *Proc. of Coordinated & Multiple Views in Exploratory Visualization 2007 (CMV 2007)*, July 2, 2007, in Zürich, Switzerland, pp. 98-110

- Raphael Bürger, Philipp Muigg, Martin Ilcik, Helmut Doleisch, and Helwig Hauser: Integrating Local Feature Detectors in the Interactive Visual Analysis of Flow Simulation Data. In *Proc. of EUROVIS 2007*, May 23–25, 2007, in Norrköping, Sweden, pp. 171-178

Other Publications:

- Markus Trenker, Hermann Lang, Philipp Muigg, and Helmut Doleisch: Validation of Vortex Flow Phenomena in Electrical Machinery using Advanced Simulation and Visualization Techniques.In *Proc. of NAFEMS World Congress*, May 22–25, 2007, in Vancouver, Canada.

- Stephan Schmidt, Oliver Schögl, Roland Kirchberger, Helmut Doleisch, Philipp Muigg, Helwig Hauser, Markus Grabner, Alexander Bornik, Dieter Schmalstieg: Novel Visualization and Interaction Techniques for Gaining Insight into Fluid Dynamics in Internal Combustion Engines. In *Proc. of the NAFEMS World Congress 2005*, May 17–20, 2005, in Malta.

- Philipp Muigg, Helmut Doleisch, Markus Hadwiger, Eduard Gröller: Novel Volume-Visualization Methods for the Interactive Rendering of CFD Simulation Data. In *Proc.of the NAFEMS Seminar on Simulation of Complex Flows (CFD)*, Mar 10-11, 2008, in Wiesbaden, Germany