M A S T E R A R B E I T

Hybrid Multiresolution Unstructured and Structured Grid Volume Raycasting

Ausgeführt am VRVis Zentrum für
Virtual Reality und Visualisierung Forschungs-GmbH

unter der Anleitung von
Priv.-Doz. Dipl.-Ing. Dr.techn. Helwig Hauser
und der Mitbetreuung von
Dipl.-Ing. Dr. techn. Helmut Doleisch
und
Dipl.-Ing. Dr. techn. Markus Hadwiger

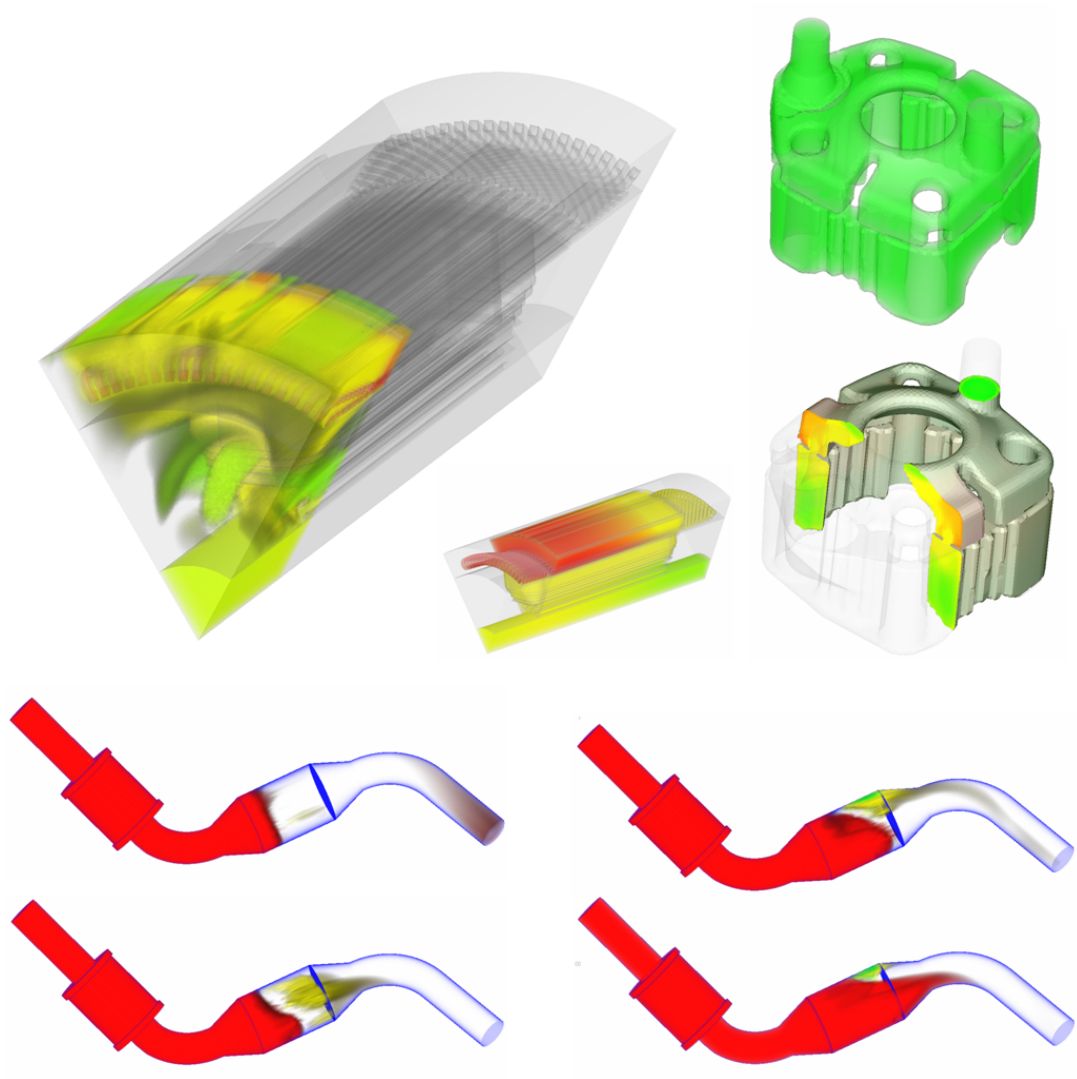
eingereicht an der Technischen Universität Wien,
Fakultät für Informatik
durch

Philipp Muigg
Meister Kliebergasse 26
A-2380 Perchtoldsdorf
Matrikelnummer: 0125958

Wien, im Februar 2007 Philipp Muigg

Hybrid Multiresolution Unstructured and Structured Grid Volume Raycasting

Philipp Muigg



mailto:Muigg@VRVis.at
http://www.VRVis.at/via/resources/masterthesis-PM/

Abstract

Different simulation techniques have gained importance in many sciences. Especially in the engineering field Computational Fluid Dynamics (CFD) has become a valuable tool to test designs even before building expensive prototypes. The results from such simulations are mostly multi modal volumetric datasets. This means, that for a simulated volume, different physical qualities like pressure or temperature are stored. The underlying organization scheme for such datasets are volume meshes or grids. Since the simulation has to be performed as fast as possible these grids are mainly designed to fill the simulated domain efficiently, using higher resolution at interesting and lower resolution at uninteresting regions. These unstructured grids pose several challenges when the corresponding volume data should be visualized. This thesis presents a novel approach to interactively visualize volume data specified on such unstructured grids by subdividing the simulated domain into bricks, which are then rendered using graphics hardware-based raycasting. View rays are shot through all bricks separately accumulating color and transparency information by treating the data volume as a semi transparent medium. Here different optical volume models can be selected making it possible to produce a variety of useful results. Subsequently the visualization results from each brick are combined to form the final visualization. The presented raycasting method is highly flexible and can cope with more complex grids, than many other state of the art approaches. Furthermore the visualization of the boundary of the simulated domain is performed explicitly, which provides the users with more information about their data. Additionally the different volume rendering methods can be freely combined with the boundary visualization methods, resulting in a large number of different selectable rendering modes. Also the volume subdivision approach allows for several optimizations which make interactive visualization on single consumer class PCs possible. If a brick is considered empty only its boundary has to be displayed. Furthermore barely interesting regions of the data volume can be resampled to low resolution representations specified on structured grids, which can be rendered much faster than the original unstructured versions. Memory requirements and memory management issues of the presented hybrid raycasting approach are discussed in the course of this thesis as well. In order to trade quality for speed or vice versa it is possible to control the precision and resolution, which should be used to create a visualization. In order to show the utility of the proposed hybrid raycasting algorithm different, highly complex, real world datasets are used to create high-quality visualization results.

Kurzfassung

Verschiedene Simulationstechniken sind in der letzten Zeit auf vielen Gebieten sehr wichtig geworden. Vor allem im Ingenieurswesen ist Computational Fluid Dynamics (CFD) ein wertvolles Werkzeug geworden, welches es ermöglicht verschiedene Entwürfe zu testen ohne teure Prototypen bauen zu müssen. Die Resultate solcher Simulationen sind meist multi-modale Volumendatensätze. Das bedeutet, dass für ein simuliertes Volumen verschiedene physikalische Größen wie zum Beispiel Druck oder Temperatur gespeichert werden. Die diesen Daten zugrunde liegende Organisationsstruktur wird als volumetrisches Gitter bezeichnet. Da Simulationsläufe so schnell wie möglich absolviert werden müssen sind die zugrundeliegenden volumetrischen Gitter darauf ausgelegt eine hohe Auflösung in wichtigen und eine niedrige Auflösung in unwichtigen Teilen des simulierten Volumens zu haben. Diese unstrukturierten Gitter stellen einige Anforderungen an Visualisierungssysteme, die die dazugehörigen Volumendaten darstellen sollen. Diese Arbeit beschreibt einen neuen Algorithmus um interaktiv Volumendaten die auf unstrukturierten Gittern gegeben sind darzustellen. Das Datenvolumen wird hierzu in einzelne Blöcke zerlegt welche dann mittels Grafikhardware basiertem Raycasting dargestellt werden. Das bedeutet, dass Blickstrahlen durch die einzelnen Blöcke geschossen werden und dabei Farb und Transparenz Informationen aufakkumuliert werden, wobei das Datenvolumen als halb durchsichtiges Medium behandelt wird. Hier können verschiedene optische Modelle gewählt werden, um verschiedene Resultate zu erzeugen. Anschließend werden die Visualisierungen der einzelnen Blöcke kombiniert um die endgültige Visualisierung zu erzeugen. Der beschriebene Raycasting Ansatz ist sehr flexibel und kann mit komplexeren Daten umgehen, als viele andere aktuelle Methoden. Außerdem wird die Oberfläche des simulierten Datenvolumens explizit dargestellt, um den Benutzern einen kompletteren Überblick über ihre Daten zur bietet. Hierzu können verschiedene Oberflächendarstellungsmethoden gewählt werden, welche beliebig mit den Volumendarstellungsmethoden kombiniert werden können, was in einer große Menge an zur Verfügung stehenden Darstellungsarten resultiert. Desweiteren ermöglicht die Unterteilung des Datenvolumens einige Optimierungen die eine interaktive Darstellung auf handelsüblichen PCs möglich macht. Wenn ein Block als leer eingestuft wird braucht nur die enthaltene Oberfläche dargestellt werden. Außerdem können wenig Information enthaltende Blöcke in eine niedriger aufgelöste Repräsentation in einem strukturierten Gitter, konvertiert werden, die dann schneller als die originale unstrukturierte Version dargestellt werden kann. Neben dem eigentlichen Visualisierungsansatz wird in dieser Arbeit auch sowohl der Speicherverbrauch als auch die nötige Speicherverwaltung des präsentierten Algorithmus diskutiert. Um Qualität gegen Darstellungsgeschwindigkeit oder umgekehrt zu tauschen kann sowohl die Präzision als auch die Auflösung der Visualisierung gesteuert werden. Die Nützlichkeit des beschriebenen Algorithmus wird belegt, indem einige Datensätze aus der Industrie verwendet werden um aussagekräftige Visualisierungen zu erzeugen.

Contents

Chapter 1

Introduction

In the last few years the increased availability of huge storage devices and the ever growing processing capabilities of modern computers have lead to large amounts of result data in engineering applications. Engineers are performing complex simulations to test their designs even before the first prototype is constructed. There are two basic techniques to perform such simulations. The first is called Computational Fluid Dynamics (CFD) and deals, as the name already suggests, with the modeling of the flow of fluids or gases through components. The second group of methods is called Finite Element Methods (FEM) which deal with the simulation of stresses of structural elements. Both groups have in common, that a discretization of the simulated components into a finite number of cells is necessary. The data that is generated by the simulation software consists of physical properties which are computed for those volume cells. Most of the time unstructured grids are the basis for this kind of data. In order to explore, analyze and present these datasets they have to be visualized interactively.

Besides the need for visualization techniques in the engineering field the rapid development of medical imaging technologies lead to the need for proper visualization techniques for medical data as well. In contrast to simulation results medical data is mostly specified on structured grids.

This master thesis is focusing on a hybrid rendering approach which allows proper visualization of data specified on structured and unstructured grids. This will be demonstrated by transferring portions of simulation results from an unstructured onto a structured grid, and displaying a hybrid visualization of the whole dataset. This resampling approach allows for memory savings and rendering speedups. Future work may even include the hybrid visualization of simulations for medical applications in the context of measured data (e.g. visualizing the flow through a blood vessel combined with the original CT (Computer Tomography) dataset from which it was extracted). Novel approaches utilizing modern graphics hardware, which make real-time rendering possible, are presented and the integration into a framework for the interactive visual analysis of multi modal data is discussed.

The following sections focus on the motivation, data organization schemes, the basic approaches which can be used to visualize simulation and medical data and the funda-

mentals concerning the architectures and abilities of modern graphics hardware. After this the further structure of this master thesis is introduced.

1.1 Motivation

There are three basic tasks for which visualization is used nowadays:

- Exploration
- Analysis
- Presentation

When a new dataset is explored no a-priori assumptions are made by a specialist. The data is inspected in order to get an overview over and to form ideas about it. In the analysis step the specialist is inspecting the data based on assumptions and theories which have to be confirmed or negated. This process is also supported by appropriate visualization techniques. The final task is the presentation of findings about the data to other people which are not necessarily as knowledgeable as the specialist in his or her domain.

Especially in the first and to a lesser extent also in the second task interactivity during the visualization is necessary and thus a key issue. Furthermore, modern approaches of visual storytelling, as discussed by Wohlfart [88], make interaction also during the presentation very important. Since the already mentioned unstructured grids on which data, produced by modern simulation software, is defined, are rather complex, special care has to be taken when interactive frame rates should be retained during the visualization. Thus there are only very basic visualization techniques integrated into many simulation packages which can provide the engineer interactive access to the data. The approach, onto which the contribution of this thesis is based, is able to give the engineer a more complete overview over the simulation results than previous approaches. This is achieved by employing a visualization scheme probably best described by Schneiderman's information drill-down methodology [69]. The basic incremental feature specification and refinement process of the approach presented in this thesis is realized in the new visualization system SimVis, which has been introduced by Doleisch and Hauser [23, 22, 21]. A very important aspect of this system is the interactive display of selected portions of a dataset in its three-dimensional context giving the user immediate feedback about his or her actions. This thesis introduces a new rendering method for the SimVis system which is able to cope with unstructured and structured meshes in a novel way. The presented method heavily utilizes the parallel processing power of modern graphics hardware and is more flexible than most previously published approaches. The algorithm is able to perform on the fly conversion of data from unstructured to structured grids which makes memory savings and rendering speedups possible. Those characteristics are extremely important since limited resources like main memory or CPU cycles have to be shared between different modules of the visualization system.

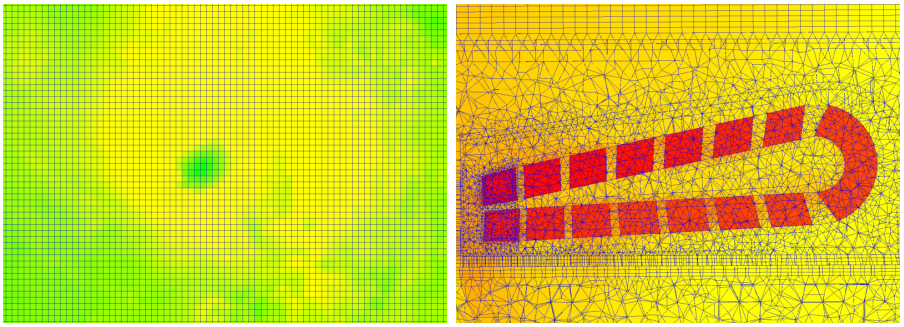


Figure 1.1: A cut through a structured (left) and an unstructured (right) grid.

Furthermore the rendering method has to be able to present a complete overview as well as a detailed in-depth examination of the data volume to the user, which poses additional difficulties to the underlying architecture. Besides the simulated domain itself, its boundary can be of high interest as well. The proposed algorithm has to take special care when treating the mesh boundary of the simulation domain. Thus the main motivation for this thesis is the design and implementation of a flexible and extendible rendering frontend which can be integrated seamlessly into the SimVis system and thus should help the user to gain better insight into the data.

1.2 Data Organization

The data, the techniques presented in this thesis have to deal with, can be described as three dimensional vector function $f : G \Rightarrow R^x$, with x being the number of data values which are calculated and/or measured and $G \subset R^4$ being the spatial volume and time interval. Since it is impossible to solve the equations that are used to model natural phenomena like fluid flow or physical stresses analytically it is necessary to apply numeric approaches to the problem which involve discretizing the simulated domain. The spatial volume is split up into cells and the temporal domain, if it is considered, into timesteps. This is also necessary for measured data since measurements can only be performed at discrete locations/times. The discretization is of crucial importance to the quality of the results. For example parts of a simulation domain where turbulent flow is to be expected have to be split into more, smaller cells than regions with expected laminar flow. This process of creating the discretized volume is called meshing and up to now is still only possible with semi automatic approaches involving heavy user interaction. In the measured case the measurement resolution has to be adapted to the desired accuracy.

The following sub-sections give a short introduction to methods which are used to store volumetric data and discuss the advantages and drawbacks which are introduced. Especially the flexibility and the addressing schemes in the different storage methods are of interest. The first aspect is more important for the simulation while the second one is essential for the visualization techniques which can be used. The data stored in a flexible data structure well suited to fit the needs of simulation is mostly harder to

access for visualization purposes and vice versa.

Structured Grids

The simplest way to organize volumetric data is to insert it into a three dimensional structured grid. This means that a rectangular bounding volume around the data is split up into equally sized cells. The whole data domain has to be filled up with volume elements (cells), even if they are outside the space which is of interest to the user. Furthermore it is impossible to apply local refinement schemes if this data organization has to be used, since all cells have to be equally distributed and sized. They are addressed by a three dimensional indexing vector which specifies the row/line/slice of the cell. Thus querying a cell index at a specific position within the data domain is straight forward. Furthermore the neighborhood information between cells is available implicitly because of the aforementioned addressing scheme. This makes it very well suited to visualization systems since fast traversal of the data volume is possible. Mostly medical imaging data and to a lesser extend simulation results, are organized on this kind of grids. Figure 1.1, left, shows a cut through a structured grid.

Unstructured Grids

Unstructured grids are used very commonly by engineers who need a very flexible data organization scheme. Basically the volume cells are specified on a set of vertices with fixed position within the simulated domain. The cells are addressed with a one-dimensional index vector. This means, that the relation between cell index and cell position, which existed in the structured case, is lost. Thus additional techniques have to be employed if a cell at a specified point within the mesh should be retrieved. Furthermore the connectivity information between the volume cells has to be specified explicitly which increases the storage requirements of this data organization scheme. Structured grids do not have this problem since cell neighbors can be determined from the indexing vector. Besides those drawbacks unstructured grids have the valuable quality that they can be arbitrarily shaped and refined giving the engineers the ability to pack more cells into interesting regions of the simulated domain while maintaining a low cell count in uninteresting areas. Figure 1.1, right, shows a cut through an unstructured grid containing tetrahedra, prisms and hexahedra. Basically unstructured grids, which are used by nowadays simulation packages (like Fluent [3], StarCD [8], Fire [2], ...) can be distinguished by the cell types which are used. The first group of unstructured grids contains only one type of cell. Here either tetrahedra or hexahedra are in use. The second group of grids can contain different cell types. Here mostly a fixed set of volume cell types is used even though advances in the simulation packages have emerged recently, allowing for arbitrary convex polyhedral cells.

Time-Dependent Data

Many volumetric datasets contain not only information about one instant in time but a whole set of consecutive time steps. Here two different ways to arrange the data are

in use. The simplest approach is to store the whole data on a static grid. Thus only the first time step has to contain the grid structure. When for example moving engine parts have to be simulated this simple approach cannot be employed since the whole geometry of the simulated domain changes over time. Here the mesh on which the data is organized has to be modified as time progresses. Since cells might degenerate more and more because they are for example stretched it is necessary to change not only the mesh shape but also its topology over time. Thus so called rezone time steps are inserted at times where cells have degenerated too strongly. Here a completely new mesh is introduced containing properly shaped cells.

Not all volume data has to be time-dependent. For example static images from CT or MRI (Magnet Resonance Imaging) scans or flow simulations which converge towards a stable solution are datasets where the temporal dimension is often ignored. Furthermore the increased computational complexity of time-dependent simulations is another cause why static models are still in use.

1.3 Contribution

There is a broad variety of approaches which can be used to directly render scalar volume data and even more when 3D vector or higher-dimensional data should be visualized in a meaningful way.

The main targets of the rendering approach which is be presented in the following chapters is flexibility and extensibility. Furthermore it should be able to cope with very large unstructured datasets which not necessarily fit into the memory of modern graphics hardware. Thus a robust and fast level of detail approach is needed. Mesh simplification techniques that produce smaller unstructured grids are prone to introduce errors when complex boundary surfaces have to be considered and do not necessarily produce better results than resampling onto structured grids for the data volume itself. Furthermore nowadays graphics hardware is used to perform volume rendering of structured grids extremely efficiently. Thus resampling portions of the dataset onto a low-resolution structured grid is the level of detail method of choice in this thesis. In order to retain the mesh boundary within those resampled portions of the dataset an approach is introduced which is able to produce exact boundary visualizations even in this case.

Some of the bricking techniques which are introduced in section 2.3 are adapted to unstructured grids, allowing for a more flexible data management. The approach to resample some bricks onto structured meshes makes it necessary to render structured and unstructured data portions as seamlessly as possible.

Nearly all of the methods developed previously are dealing with tetrahedral meshes. Modern CFD simulation packages, however, use a variety of cell types which can be converted to tetrahedra. But this conversion almost always increases the number of cells by one order of magnitude. The method proposed in this thesis can deal with different cell types without the need for a tetrahedralization. The decision to use a raycasting approach instead of a cell projection technique was due to the ever increasing size of the source data. Cell projection methods are object-order techniques, which

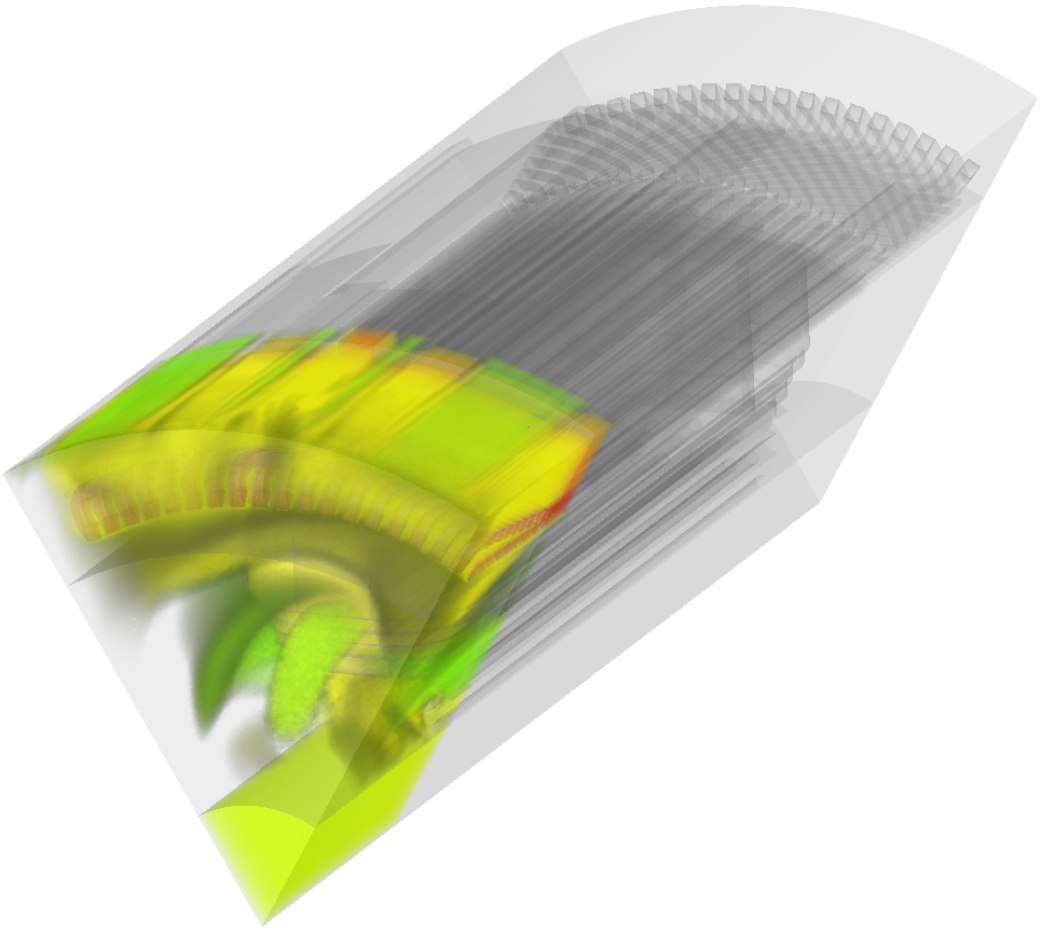


Figure 1.2: This figure shows a result visualization of the generator dataset, which has been created with the hybrid raycasting approach. Here warm and cold air (shown in yellow and green) are sucked into a central vortex (shown in light green). Further details on the analysis of this large and highly complex dataset are presented in section 6.2.

means, that the runtime of such algorithms is basically always directly proportional to at least the number of volume cells. Raycasting methods are mostly influenced by the resolution of the resulting image. Furthermore the parallel nature of raycasting-based techniques make GPU-based implementations very efficient and fast, which allows for the desired interactivity of the visualization. Figure 1.2 shows an example visualization, which has been created during an interactive analysis session with the SimVis framework in combination with the hybrid raycasting algorithm.

1.4 Thesis Organization

Chapter 2 gives a short introduction to volume visualization and the basic problems of the raycasting approach. Furthermore the SimVis system is discussed and an overview over state of the art techniques, which have been considered during the development of the hybrid raycasting method, is presented. In chapter 3 raycasting for structured and unstructured grids is introduced and related problems are discussed. Additionally an overview over the hybrid raycasting approach is presented. After this chapter 4 focuses on the data structures and memory management techniques, which are used in the hybrid raycasting algorithm. Chapter 5 presents all parts of the hybrid raycasting algorithm in more detail. Furthermore image quality and memory consumption issues are discussed. Additionally implementation considerations are presented briefly. In chapter 6 some results are discussed and comparisons between the previously available visualization method and the new hybrid raycasting approach are made. Subsequently chapter 7 summarizes the topics, which have been covered in this thesis. The thesis concludes with conclusions and topics for future future work in chapter 8.

Chapter 2

Fundamentals and State of the Art

This chapter gives short introduction to relevant fundamentals and an overview of the present (2006) state of the art of techniques related to this diploma thesis. This does not only include rendering and volume rendering methods for structured and unstructured grids, but also mesh simplification and compression approaches, that are of interest. Those topics are introduced and their relevance for this thesis is discussed. Furthermore SimVis, the visualization system into which the presented techniques are integrated, is presented.

Visualization of scalar data given on regular grids is a very thoroughly explored field, which has spawned several rendering approaches that are discussed in section 2.3. The current state of the art, which is presented in this section, mainly consists of GPU-based approaches that perform nearly all computations on graphics hardware. The next section covers techniques applicable to the more difficult case of scalar data arranged on unstructured grids, which poses severe additional challenges to the visualization system. The algorithms presented in this section also heavily use modern graphics hardware. But contrary to the structured case many approaches have to rely on the main CPU for tasks, that (still) cannot be moved to the GPU, because of the additional difficulties which are posed by unstructured meshes. Due to the fact that modern CFD simulation clusters can generate huge amounts of data, which cannot be handled interactively by current hardware, it is necessary to employ some kind of data reduction and/or compression scheme, if interactive frame rates during a visualization session should be retained. Thus the section 2.5 focuses on volume mesh simplification and resampling techniques.

2.1 The SimVis System

SimVis [23, 24, 22, 21], which has been introduced briefly in the previous chapter, is an approach for the visual analysis of multi-variate, spatially and temporally organized data. This means, that the evolution of several spatially laid out data fields over time can

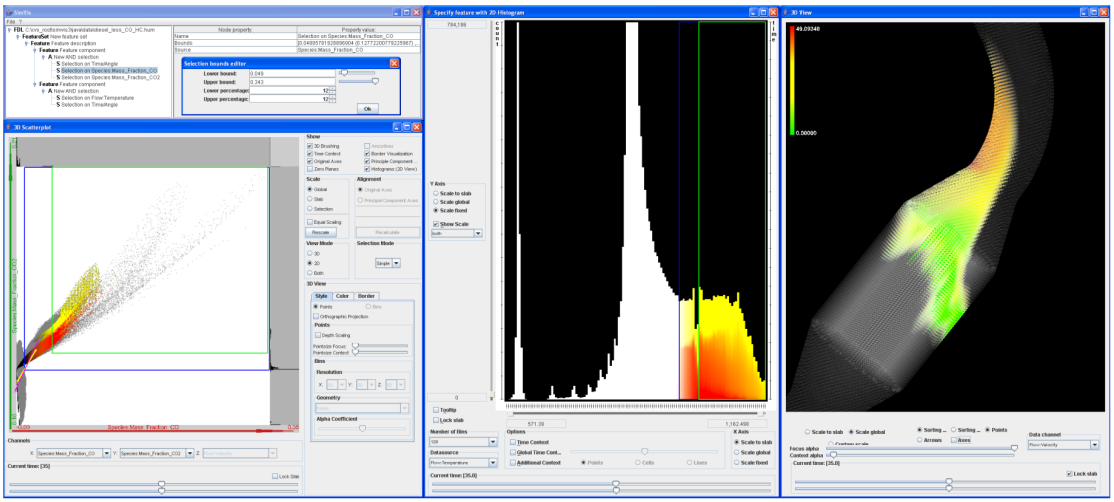


Figure 2.1: A sample SimVis scenario: simulated flow through a diesel particle filter (DPF) is visualized - the flow is shown at the time of 35 secs. after simulation started. The user has reflected his interest in flow regiions of heavy oxidation by interactively brushing data items which exhibit a lot of carbon-oxides in the scatterplot (lower left) and then refining this specification to only apply to hot regions (in the histogram, middle). The 3D view on the right shows a focus+context visualization of the DPF with the brushed data items highlighted in color (color shows velocity magnitudes).

be explored and analyzed. Presently the main field of application is the visual analysis of simulation data coming from the engineering and meteorological fields [25, 47, 26]. There are some basic methodologies, which are applied to all parts of the SimVis system and make the visual analysis of high-dimensional, time-dependent volume data possible. The next subsection introduces those methodologies. Following subsections deal with the specification mechanisms for the so called Degree of Interest (DOI) function and the view concept of the SimVis framework.

SimVis Basics

The SimVis system is a multi view approach. This means, that it can provide the user with different visualizations, showing different aspects of the source data (Figure 2.1 shows a typical SimVis session). In order to support the visual analysis process user every interaction results in an immediate response from the system. The exploration and analysis process itself is based on the specification of a DOI value for every data item, which is then used to adapt the visualization. In order to support the DOI specification process interactively several methods are employed to guide the user.

Two different types of views are provided to the user. Passive views, which are used to view a previously formulated DOI and active views, which provide the user with means to specify a DOI. Presently all active views provide interactive brushing [87] functionality, which lets the user manually select data items by marking them with the

mouse. Since data from different physically based simulation methods is most of the time smoothly distributed it is difficult for users, who want to inspect their data, to define exact boundaries between interesting and uninteresting data values. Thus the SimVis framework and all of its views support the visualization and specification of non binary Degree of Interest functions. Smooth brushing [23] is used to assign values from the interval $[0, 1]$ to all data items, with zero representing no interest and one full interest. The visualizations in the different views are adapted accordingly.

Views are bidirectionally linked to each other over a common DOI function. This means, that user interaction in one view is reflected in all other views, which are used to specify and show the same DOI function. The next important approach, which is used throughout the SimVis system is Focus+Context visualization. All views have to use the DOI to discern focus from context. Focused portions are drawn in a more prominent way whereas context parts are presented less prominently to reduce possible cluttering, but still give the user an overview over the rest of the data.

The temporal data dimension is treated specially in the SimVis system. All views in the framework support focusing on separate timesteps or specific time intervals of the source dataset. Thus the user has the ability to observe the evolution of interesting data regions over time, or even follow the accumulated information over a time span.

Feature Definition Language

In order to manage the DOI functions which are specified by the user, the SimVis system uses a concept to explicitly represent the feature specification process. This concept is called Feature Definition Language (FDL) and is organized in a tree-like hierarchical structure. Each leaf in this tree represents a DOI function, which is directly based on the data itself. A simple interval selection on a data attribute is a common example. Every internal node within the tree represents a logic combination of its child nodes. Thus the simple leave DOI functions can be combined into arbitrarily complex multi-dimensional selections. The first five levels of the FDL tree have a fixed layout. The root node has only so called *feature set* nodes as children. These nodes represent individual DOI functions which are independent of each other. A *feature set* node has only *feature description* nodes as children. They are combined by a logic "or" operation to form the parent *feature sets* DOI function. Every *feature description* node can contain several *feature components*, which are combined by a logic "and" operation. Every *feature component's* DOI is specified by an arbitrary sub tree of FDL nodes, which are managed by an active view, associated with this feature component.

In order to make the FDL tree persistent it is possible to save its complete state (including the settings of the views, which are linked to it) into an XML file. From this session file the complete visualization process can be recalled by loading this file. Furthermore it is possible to apply the visualization session information to another dataset by loading the FDL tree information, which defines what portions of data are interesting to the user on an abstract basis, and applying this abstract DOI specification to different datasets.

Active and Passive Views

In the SimVis framework active views are discriminated from passive views. An active view is associated with exactly one feature component. The subtree below this feature component can be defined freely by an active view. For example a simple histogram view can add a logic "or" node, which itself contains several simple interval selection nodes on a data attribute, which can be specified by the user. Thus such a view enables the user to select multiple intervals on a single data attribute. All user induced modifications of the DOI of one feature component are propagated through the FDL tree. In order to give direct feedback about this all active views have to monitor their encompassing feature description and feature set nodes and visualize their DOI function as well. Thus interaction, which modifies the DOI in one active view is seen in all other active views, which are descendants of the same feature set node.

Unlike active views passive views can only be used to visualize the DOI of a FDL tree node. The rendering algorithm, which is presented in this thesis is part of such a passive view. It can be linked to a FDL tree node and display its DOI values in the spatial context of the underlying dataset by applying direct volume rendering

2.2 Visualization of Volume Data

The visualization of volumetric time-dependent data is a very complex topic. Thus this section can only give a small overview over some techniques which are commonly used. The problem which has to be tackled is that a three or even four dimensional data volume containing often more than one data modality has to be projected onto a two dimensional display device. Thus occlusion and cluttering issues are most common problems of nowadays techniques.

Basically the visualization process can be divided into different stages of the so called visualization pipeline. The first one is the data acquisition. In the case of medical applications this stage is performed by some kind of imaging method like a CT, MRI. In engineering applications different kinds of measurement techniques and simulation are commonly used to create the data which has to be visualized. The data which will be in the main focus of this thesis is data from CFD (Computational Fluid Dynamics) simulation which is mostly defined on the already introduced unstructured grids and can be either time-dependent or stationary. The next step is the data enhancement. Here the data is prepared for the further visualization process by performing filtering operations or deriving additional data attributes. Which tasks have to be performed in this step is highly dependent on the type of the processed data (measured data must be filtered in most cases, while this is not necessary at all when simulation results have to be visualized). After the data enhancement stage the visualization mapping has to be performed. This is probably the most crucial step for a good visualization. Here the enhanced data is mapped into a suitable abstract visualization space. Different data attributes are represented by visualization metaphors which can be interpreted by a human user (a simple example would be mapping the temperature of a simulated engine part onto a color scale). After this mapping has been determined a projection onto a

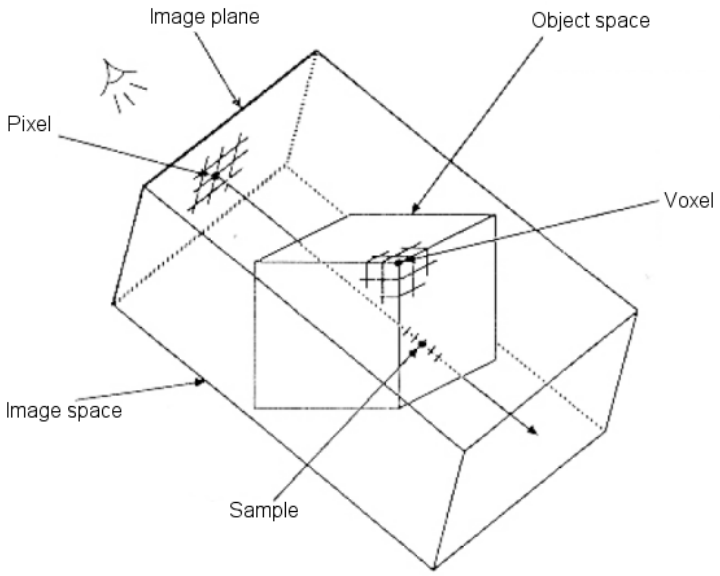


Figure 2.2: When performing volume rendering the data volume is sampled and the accumulated color values are written into the image plane [50].

two dimensional display device has to be performed. The algorithms presented in this thesis will mostly deal with this last step of the visualization pipeline. Since in nowadays visualization approaches interactivity is gaining more and more importance real- or near real-time rendering techniques will be the main focus of this work.

The following sub sections give a brief overview over common visualization techniques which are used to visualize volumetric datasets.

Direct Volume Rendering

The most straight forward technique to render volumetric datasets is direct volume rendering [49] (DVR). It is also the basis for the method proposed in this thesis. Here the source data is mapped to color and opacity values within a three-dimensional visualization space. Then this distribution is treated as a semitransparent medium which is projected onto the screen as shown in figure 2.2. The mapping function from data space into visualization space is called a transfer function. This function can be one-dimensional, if only one data modality should by visualized. When processing medical data from CT or MRI scans this is mostly enough (even though there are extensions where additional derived data modalities are used [42]). Still even the definition of a one dimensional transfer function is investigated in multiple publications by researchers like Fang et al. [31], Kindlmann et al. [41] or Castro et al. [15]. If the data which should be investigated is multi-modal, the transfer function has to be more complex to incorporate information from more than one data dimension.

There are several ways, how the projection of the opacity/color volume can be per-

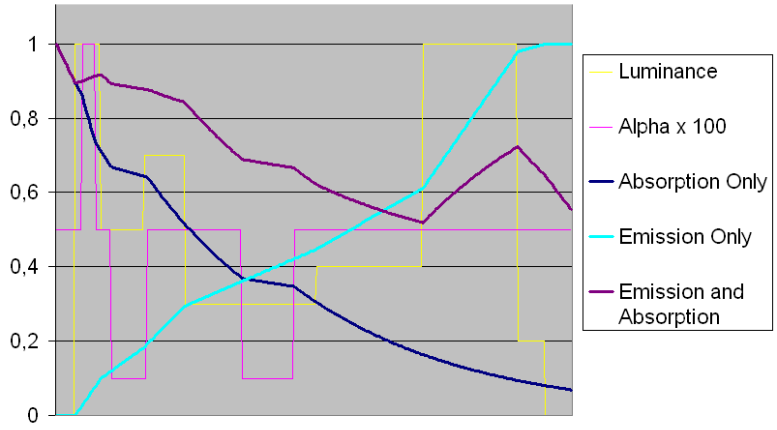


Figure 2.3: This figure shows the three basic optical models with the corresponding luminance and transparency functions.

formed which can be classified by the optical model used. Max [51] specifies basically five ways how the interaction of light with a semi transparent volume can be simulated which differ by the way how light is generated and absorbed:

- Emission only
- Absorption only
- Emission and Absorption
- Single scattering
- Multiple scattering

Figure 2.4 shows example images generated by the first three. The most basic models treat only one interaction characteristic at once. When only emission is handled the data volume represents a glowing cloud within which light is only emitted and not absorbed. This means the final light intensity I at the observers position can be expressed by the simple equation:

$$I = I_0 + \int g(t)\,dt \tag{2.1}$$

Here the function $g(t)$ denotes the glow factor of the cloud at a position t along a view ray. I_0 is the light coming from the background of the data volume. Since only a discrete volume representation is available the integral has to be evaluated numerically when applied to data specified on volume grids:

$$I = I_0 + \sum C(t) \tag{2.2}$$

Here $C(t)$ is a color sample from within the data volume along the view ray at the position t.

If only absorption should be treated the corresponding equation

$$I = I_0 \cdot \exp(-\int \tau(t)\,dt) \tag{2.3}$$

can be formulated. The factor $\exp(-\int \tau(t))$ is the remaining amount of transparency with $\tau(t)$ being the opacity of the data volume at the position t along a view ray. The discrete representation of this equation is:

$$I = I_0 \cdot \prod(1 - \alpha(t)) \tag{2.4}$$

Instead of using the transparency at a sample position the absorption $\alpha(t)$ is used here. The absorption between the position u and v along a view ray is defined as

$$\alpha = 1 - \exp(-\int_u^v \tau(t)\,dt) \tag{2.5}$$

It is obvious, that the discrete approximation of both simple models does not depend on the order in which the glow factor/transparency values are combined. This property is the most important merit of both models. The drawback of this is, that they can only convey limited depth perception.

If the basic interaction of emission and absorption is simulated, as shown in Figure 2.4, right, the analytic formulation of the optical model is the following equation:

$$I = I_0 \cdot \exp(-\int \tau(t)\,dt) + \int g(t) \cdot \exp(-\int_{u<t} \tau(u)\,du)\,dt \tag{2.6}$$

Here the glow factor's influence is multiplied by the amount of transparency $\exp(-\int_{u<t} \tau(u)\,du)$ which is left at its location within the volume. The discrete approximation to this equation which is used nowadays by most algorithms is based on this equation:

$$I = I_0 \cdot \prod(1 - \alpha(t)) + \sum C(t) \cdot \alpha(t) \cdot \prod_{u<t}(1 - \alpha(u)) \tag{2.7}$$

When evaluating this equation per sample the order in which this accumulation is performed cannot be chosen arbitrarily. Thus sorting of the samples based on their distance to the viewer is necessary if no implicit ordering can be derived from the data organization scheme.

The last two optical models include further light/volume interaction simulations which will not be discussed in great detail since they are not directly relevant to the algorithm presented in this thesis. If single light scattering is used the glow factor at a specific position within the data volume is not only weighted by the remaining transparency with respect to the view position, but also the incident light intensity which reaches the point is included. Multiple light scattering adds the simulation of inter-reflections of light within the simulated volume.

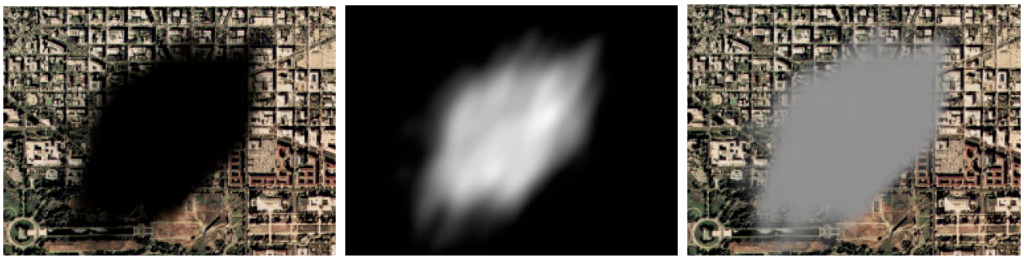


Figure 2.4: Three different optical models. From left to right: absorption only, emission only, emission and absorption. [51]

Feature-based Visualization

When direct volume rendering is used the whole data volume has to be sampled for the visualization. This is rather costly and if only parts of the data are of interest it is often more efficient to extract the important information and perform the rendering of only those features. Besides the increased efficiency feature-based visualization helps reducing the cluttering problem which is inherent to techniques trying to display the whole three-dimensional data volume at once.

Most feature-based visualization approaches are domain specific. For example methods which perform vortex core extraction can only be applied to volumes containing flow vector data. A good overview can be found in a state of the art report by Post et al. [56]. Similarly different segmentation techniques in the medical field work often only on medical data. The visualization system SimVis, into which the rendering method, which is proposed in this thesis, is integrated, incorporates a very general feature-based approach which can be applied to a broad variety of fields. It is primarily based on user interaction through which interesting portions of the data can be detected and visualized.

2.3 Direct Visualization of Scalar Data on Structured Grids

There are many fields, like meterology or medicine, where large amounts of volume data are defined on structured grids. The structure of such data, as discussed in the introduction, is very simple and intuitive and thus very suitable for fast and efficient direct visualization techniques.

Levoy [49] was the first to propose direct volume rendering of medical data. He used a simple two dimensional transfer function to map the density and its gradient values from the original data to color and opacity values (the gradient magnitude is used to emphasize tissue boundaries in the data). This new data volume (containing $RGB\alpha$ values at each voxel) is then used in the subsequent raycasting step. Here a ray is shot from each result image pixel into the volume. Along this ray one of the optical models, which were introduced in the introduction, have to be evaluated. Thus color and opacity samples have to be extracted from within the data volume. Mostly the emission and absorption model is used since samples can be taken from the data at arbitrary locations and the compositing can be performed in the correct order.

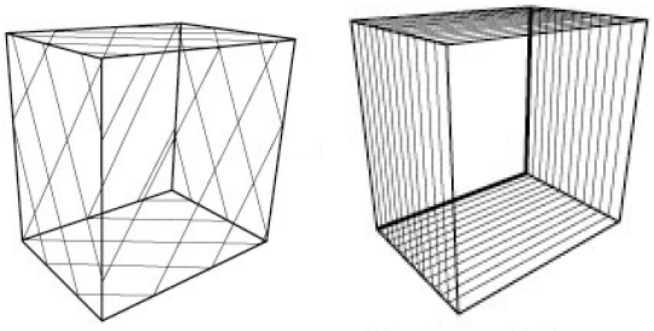


Figure 2.5: View aligned slices (left) and object aligned slices (right) [27].

Slice-Based Volume Rendering

To correctly retrieve the data samples the voxel data around them has to be interpolated. This is one of the most time consuming steps of the algorithm. Since already early workstation graphics hardware supported interpolation of volume data some research was done in order to transfer parts of the volume rendering onto this dedicated hardware. Cullip and Neumann [19] were the first who proposed such an approach. They use either object or viewplane aligned polygons, called proxy geometry, as shown in figure 2.5, which are intersected with a volume texture and thus contain slices through the data volume. The graphics hardware performs the interpolation and the subsequent blending of the data samples onto the viewport. Van Gelder et al. [78] propose to incorporate a proper lighting model based on one directional light into their GPU-based approach. Dachille et al. [20] introduce a direct volume rendering variant which distributes different tasks of the algorithm between CPU and GPU. Thus their implementation is able to profit from the flexibility of the CPU and the raw interpolation performance of the graphics hardware. Here the proxy geometry is not rendered parallel but orthogonal to the view plane. Those slices are transferred one by one to the main memory from which the CPU can perform the compositing and shading operations. The main bottleneck for this approach is the graphics bus which has to be used when the interpolation results have to be transferred from graphics memory to main memory. Westermann and Ertl [84] further demonstrate how workstation graphics hardware can be used to perform volume rendering. Additional features like clipping away user specified parts of the volume via the stencil buffer or shaded first hit raycasting are presented. Rezk et al. [58] were the first to utilize desktop graphics systems to perform volume rendering. Since the hardware did not support 3D texturing three stacks of 2D textures are used. They furthermore present techniques to reduce the artifacts introduced by the lack of trilinear interpolation.

With further advances in the programmability capabilities of graphics hardware additional refinements are introduced by Engel et al. [28] which increase image quality considerably. They suggested using a pre integration approach for the evaluation of the optical emission/absorption model which can be evaluated on the GPU. Here instead of

determining the color and opacity of a ray segment based on one sample the previous sample is included as well. Under the assumption that the scalar value varies linearily between the first and second sample along the ray it is possible to create a two dimensional lookup table. When the opacity and color of a ray segment has to be computed a lookup with the scalar value at the beginning and the end of the seqment into this table is performed returning the desired data. This approach is especially useful when very high frequency transfer functions are used, or the volume should be sampled as sparse as possible.

Raycasting-Based Volume Rendering

The slicing approach has several drawbacks. It lacks flexibility when non orthogonal projections should be used. Furthermore it is difficult to incorporate optimizations, like empty space skipping or early ray termination, which were proposed by Levoy [50] and are common in CPU-based implementations, into slice-based volume rendering algorithms. Thus as soon as the hardware was capable to process more complex operations per fragment the classical raycasting approach which was only feasible on the CPU up to this point was adapted for the GPU. Röttger et al. [63] are the first who proposed such an implementation. Krüger and Westermann [45] further refine this approach. Here empty space skipping and early ray termination is realized using multiple passes. Basically the hardware is used to cast all rays of an image at once by rendering full screen quads and querying position and ray direction information from several textures. With further technology advances one pass is sufficient to perform the raycasting. Stegmaier et al. [72] propose such a single pass raycasting algorithm mainly focusing on the flexibility and extensibility of the volume rendering framework making more complex rendering modes possible.

Bricking

One of the main drawbacks of using the GPU to perform volume rendering is the lack of memory. Especially early graphics hardware had, compared to the main memory, only a small amount of texture memory available. Furthermore advances in medical imaging technology lead to ever bigger datasets. Thus methods to reduce the necessary memory in order to retain interactive visualization are needed. Lamar et al. [46] propose to use an octree to generate a level of detail hierarchy for the data volume. The selection of the resolution within this tree is based on the distance to the viewer. Weiler et al. [82] also use a hierarchy of 3D textures making it possible to render portions of the data volume at different resolutions. They mainly focus on making the transitions between different levels of detail as smooth as possible. Boada et al. [10] use an octree-based multi resolution approach as well. Here the data volume is first sub-sampled to create an image pyramid. Each level contains 1/8th of the voxels of the previous one. Then the volume is decomposed into an octree. For each node an error value is computed. It is based on the error which will be introduced if a low resolution version of the data volume is used when displaying the node. Homogeneous or empty parts of the data volume will result in

lower error values for low resolution representations than data portions containing lots of detail. Besides this error value the user is able to specify a region of interest which the algorithm will try to display at the highest possible resolution. After the error metric and the region of interest is evaluated the octree is traversed to determine which nodes are suitable for rendering and which volume resolution should be used within each.

In order to visualize datasets which are even too large for the main memory of conventional computers Guthe et al. [37] propose a multi resolution scheme roughly based on Boada et al.'s approach. They focus mainly on the generation of interactive walkthroughs through the data volume. Thus perspective rendering is necessary and the projected size of the octree blocks has to be considered as well. In order to reduce the overall size of the data volume it is compressed using wavelet compression. The rendering itself is performed by slicing volume textures which are reconstructed on the fly from the wavelet representation. Strengert et al. [75] extend Guthe et al.'s approach to visualization clusters. The data volume is split up into equally sized portions which are distributed to the individual nodes of the cluster. The rendering is performed by each node and the resulting images are composited together.

The previously mentioned methods need the volume data to be loaded into texture memory at once. Thus compression or subsampling has to be used to reduce the amount of data. Hong et al. [40] propose to stream portions of the volume onto the graphics card during the rendering. They partition the data into equally sized cells. The cells are then rendered in a front to back order by GPU-based raycasting. Hadwiger et al. [38] also propose a bricking scheme to perform the rendering of large volume datasets. They focus on isosurface rendering and thus in many cases only parts of the whole data volume contribute to the final image. The data volume is split into equally sized cells. The ones containing parts of an isosurface are loaded into a caching texture. The rendering itself is performed by a GPU-based raycasting approach which uses an indexing texture which references the caching texture. Scharsach et al. [65] extend this technique to additionally incorporate direct volume rendering.

2.4 Direct Visualization of Scalar Data on Unstructured Grids

Besides data specified on structured regular grids which is common for medical and, to some degree, meteorological applications the field of computational fluid dynamics (CFD) mainly generates huge amounts of multimodal data on unstructured grids. Furthermore new developments of realtime simulation techniques, for deformable tissues in medical applications, make it necessary to not only render regular volume data, but also unstructured deformed portions of a volume [76].

Such unstructured volumes pose severe additional problems to the rendering system, because this data organization scheme does not provide an easy way to traverse the dataset efficiently. In regular volumes a three dimensional index vector can be deduced from a position within the volume, representing the cell containing it. The key problem of unstructured volumes is that this transformation from object into index space is

not trivially realizable. Furthermore the memory requirements are much higher in the unstructured case because the cell connectivity information has to be stored explicitly.

In the following subsections two groups of conceptually different approaches will be presented. The first group consists of techniques, performing the rendering of unstructured data in object (volume cell) order, whereas algorithms belonging to the second group are image pixel-based.

2.4.1 Object-Space Approaches

Object-space methods to render unstructured volumes try to approximate the volume rendering integral by projecting the volume cells of the dataset onto the image plane and performing compositing there. In order to guarantee the correctness of this compositing, it is necessary to perform the projection process in a certain order depending on the view direction (either front-to-back or back-to-front depending on the compositing scheme used). This means that the order in which the data is rasterized has to be recomputed every time the view direction changes, and thus this sorting is a very fundamental part of many rendering algorithms. The second important task object-space methods have to perform is the piece-wise approximation of the volume rendering integral for every volume element.

Sorting

Sorting cells in unstructured grids should thus result in a total order on the cells, such that if object A obstructs object B, then B precedes A in the back-to-front order. Finding such an order for completely arbitrary meshes containing arbitrarily shaped cells is a highly complex problem. Thus several works on this topic assume certain conditions the mesh has to fulfill.

The first constraint, which is nearly always fulfilled, is that all cells are convex. The next, more severe, requirement that has to be met for many approaches is that the unstructured grid does not contain visibility cycles. This means that there is no set of cells $\{A_1, A_2, A_3, ..., A_n\}$ where following relations exist between $A_1 < A_2, A_2 < A_3, ..., A_n < A_1$. For grids containing such circular relations no complete order can be found. Figure 2.6 shows an example of such a circle.

In theory the A-buffer introduced by Carpenter [14], would solve the problems that arise with the necessity to sort all cells of a volume. This buffer delays the blending of transparent fragments in the framebuffer, until the rendering is complete. Per screen pixel a list is generated, which stores the $RGB\alpha$ and depth information of all fragments contributing to it. After the rendering process is finished the fragments are correctly sorted and then blended together. The main problem of this approach is that the overdraw of cell projection methods is extremely high and would lead to enormous memory requirements for the A-buffer. Furthermore there is no hardware implementation for this data structure, which severely limits this approach compared to other techniques that can use dedicated hardware.

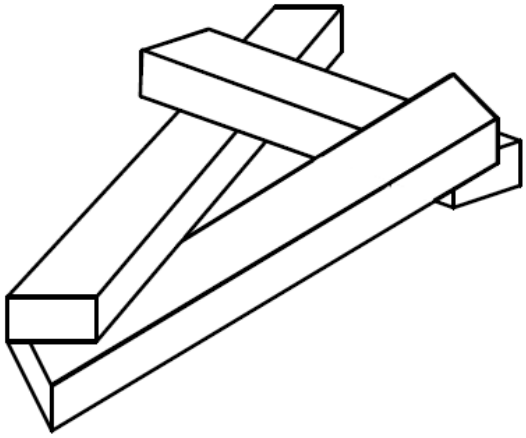


Figure 2.6: Three cells forming a circle. No total order can be specified in this case. [73]

MPVO

Most exact sorting algorithms for regular grids are based on the Meshed Polyhedra Visibility Ordering algorithm (MPVO) proposed by Williams [86]. In his original work he assumes that the grids have to be convex in addition to the required absence of circles and the convexity of all cells. His approach is based on the adjacency graph of the volume cells. The directions of the edges are set based on the view direction. Every edge represents a face that is shared by the two adjacent cells, dividing the volume in two half spaces, one containing the view point. If the half space containing cell A contains the view origin and the other half space contains the adjacent cell B the edge connecting A and B in the adjacency graph is oriented towards A. Figure 2.7 shows an example of an unstructured triangle mesh and the corresponding MPVO graph. Since this directed graph has to be acyclic because of the initial assumption, that no cell circles exist, an order can be derived by performing a topological sort on the graph. This sorting procedure repeatedly performs the following steps:

- Select a cell that has an in-degree of zero in the graph
- Add the cell to the ordering
- Remove the cell from the graph (and thus decrease the in-degree of adjacent cells)

When the algorithm terminates the adjacency graph is empty and a visibility order for the cells of the mesh has been generated. The running time of the algorithm is basically $O(n)$ where n corresponds to the number of cells. The main problem of this approach is, that the mesh, it can be applied to, has to be convex. If cells of a non convex or disconnected grid should be sorted the MPVO method does not compute a valid visibility order, because the relations between boundary cells or cells belonging to disconnected portions of the mesh obstructing each other, are not considered in this approach.

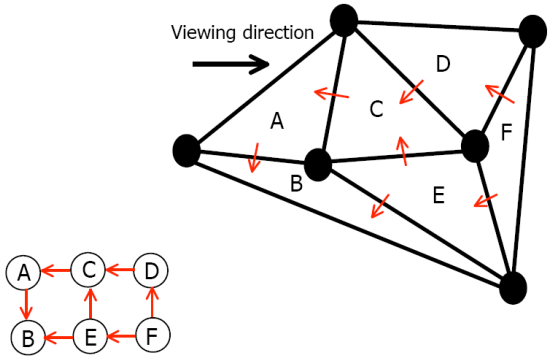


Figure 2.7: Unstructured mesh and the corresponding MPVO graph.

Coping With Concave Meshes

In order to overcome this problem Stein et al. [73] extended the Newell, Newell, and Sancha sort for polygons [55] to convex polyhedrons. The basic idea behind their approach is to perform a rough sorting of the cells based on their rearmost vertex. After this, fine tuning of the ordering has to take place, resolving errors that were introduced in the previous step. This procedure involves several tests, which have to be performed sequentially on cell pairs determining if they can be rendered in a certain order. In this process cell cycles can be detected and cell subdivision can be performed to resolve them. The time complexity of this approach is $O(n^2)$ with respect to the number of polyhedra sorted. Compared to $O(n)$ of the MPVO algorithm this running time behavior poses severe problems when large meshes should be rendered interactively.

Thus several modifications and additions have been made to the original MPVO algorithm that should roughly preserve its efficiency and make it applicable to non convex grids. Silva et al. [70] introduced the extended MPVO (XMPVO) algorithm and Comba et al. [18] presented the BSP-XMPVO approach which both were able to expand the original algorithm to non convex and disconnected meshes.

In order to perform this task, additional edges have to be introduced into the directed adjacency graph. Those edges have to represent the behind relation between the boundary cells that do not lie on the convex hull or in disconnected portions of the mesh. Silva et al. utilize a sweep approach which scans the projection of the boundary cells for special events, at which visibility order is computed and the behind relations for corresponding cells is established. The time complexity for this algorithm is $O(b^2 + n)$, where b corresponds to the number of boundary cells and n to the overall cell count. b can be expected to be bounded by $O(n^{2/3})$. Comba et al. chose another approach to extract the additional relations for the boundary cells. In their BSP-XMPVO algorithm a binary space partitioning (BSP) tree [33] is created. It contains all boundary facets of boundary cells of the mesh. By correctly traversing the tree it is possible to provide a visibility order for the boundary facets and thus for the corresponding boundary cells as well. The construction of this tree can be performed in a preprocess and thus does not

introduce additional computational costs during the sorting. During the construction of the BSP-tree it may become necessary to split boundary facets. The time complexity of this approach is highly dependent on the amount of additional facet fragments that are introduced by such splits, and thus the tree generation algorithm has to try to minimize such operations. Comba et al. provided experiments showing that most meshes in use only introduce a small amount of facet splits, basically reducing this algorithm to linear time complexity (if there is a considerable amount of fragmented boundary facets, the algorithm exhibits the time complexity $O(bp + n)$, with n being the number of cells and b the number of boundary cells. p is the number of boundary facets that are split more than once by the BSP-tree).

HAVS

Besides approaches that try to perform proper visibility sorting on all cells of the mesh Callahan et al. proposed a method that does not need a fully correct visibility order [13]. The fragments coming from cells that are not drawn correctly after each other are sorted in an image-based approach that resembles the already mentioned A-buffer algorithm, which performs the sorting of all fragments contributing to a pixel after the rendering is finished. The difference to the method Callahan et al. proposed is, that the sorting of all incoming fragments is not delayed until the end of the rendering process but instead the ordering is performed based on a limited number of fragments that are buffered in the so called k-buffer. To define the notion of a k-nearly sorted sequence (k-NSS), which is used in their work, an exactly sorted sequence (ESS) has to be defined first. Given a set $S = \{a_1, a_2, a_3, ..., a_n\}$ of real values, the vector $(b_1, b_2, b_3, ..., b_n)$ is an ESS of S if b_i is the position of a_i in a sorted list of the values contained in S. For example the vector $(2, 1, 3)$ is an ESS of the values $\{1.2, 0.1, 3.0\}$ with respect to the relation $<$. A k-NSS(S) can thus be defined as a vector $(c_1, c_2, c_3, ..., c_n)$ containing all the positions of elements from a set, where the maximal pairwise difference between an element from a corresponding ESS is equal to k (i.e., $(3, 1, 2)$ is a 1-NSS of the set mentioned in the previous example). In order to perform the correction of the sorting errors introduced by a k-NSS, at least k entries have to be kept in the a buffer. This implies that, when cells of an unstructured mesh are rasterized in a k-NSS, and the correct visibility order should be created, it is necessary to perform a so called streaming sort based on the last k fragments. This sorting technique looks for the smallest/biggest entry in the buffer and appends it to the end of the sorted sequence. The incoming entry is then added into the now empty slot in the buffer. In the case of fragments that should be sorted by their visibility, the criteria that should be used, to perform the stream sort, is their depth.

The implementation Callahan et al. suggest is GPU-based and exploits the ability of modern hardware, to render to multiple render targets at once. Furthermore the possibility of binding and reading from a texture while, at the same time, performing writing operations into it is used. This feature is undocumented and not guaranteed to remain stable in future hardware generations. Still it enables a fragment program in combination with multiple render targets to read the k-buffer data from textures,

perform the streaming sort on them, blend the selected fragment into the framebuffer and write the resulting new k-buffer back into the textures. Since the introduction of framebuffer objects [4] the perviously mentioned exploit of undefined behavior can be avoided by performing ping pong rendering between two sets of textures.

As already mentioned the input sequence to the streaming sort algorithm is not necessarily ordered completely correctly. Thus simpler and faster algorithms can be used to create the initial order. Several sorting strategies were compared and evaluated by Callahan et al. They propose to use radix sort [57] to order the cells based on their centroid. Basically this sorting algorithm has a time complexity of $O(n)$ with respect to the items to sort, but it generally only works for integer values. In order to apply this algorithm to real values they have to be transformed. When 32 bit IEEE floatingpoint variables are used, the function

```
inline unsigned int float2fint(unsigned int f)
{
  return f^((-(f >> 31)) | 0x80000000);
}
```

converts the real values into unsigned integers without destroying the order between the original numbers (a transformed value x is greater than a transformed value y iif the original value of x was greater than the original value of y).

Other Methods

All the already mentioned approaches perform some kind of sorting on the cells of an unstructured mesh to create a correct visibility order for them. Besides the classical emitter absorber model that is used in most volume rendering applications Röttger et al. [59] proposed to use only an emissive material representing the volume. This implies that no light will be absorbed and thus the rasterization order of the cells will not matter. Naturally the drawback of this method is, that the images rendered mostly lack depth and recognizing the correct depth relation between features within the volume gets very difficult. Thus this technique is best applicable to light emitting natural phenomena like fire.

When discussing techniques that rely on performing some kind of visibility sorting on elements in a three dimensional space, methods that were originally applied to polygons, but which may be extended to polyhedra, should be mentioned as well. Govindaraju et al. [36] proposed an algorithm that maps very well to the functionality of modern GPUs. They extend the classic selection sort algorithm such that not only the minimum/maximum of an unsorted rest sequence is chosen and appended to a sorted output list. Instead a whole list of ascending/descending elements is extracted by a traversal of the unsorted portions of the input data. This sequence is then added to the sorted output sequence element by element, testing for each entry, if it really fits. This can be checked by simply tracking one minimum/maximum value during the insertion steps. The permanent checking, whether a polygon is completely behind another one which has to be performed repeatedly during the run of the algorithm can be performed completely

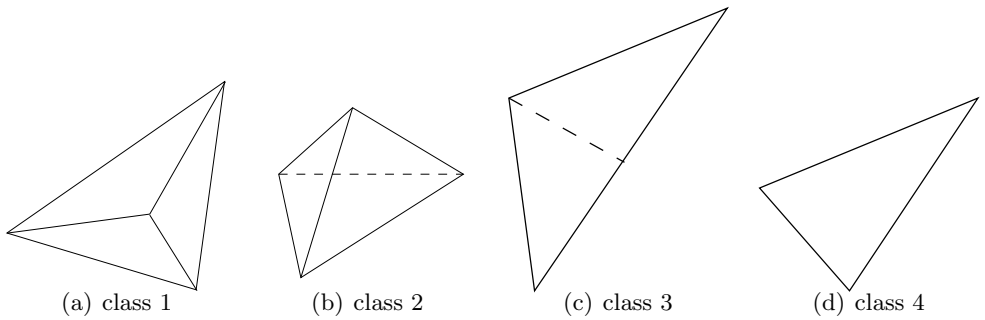


Figure 2.8: Classification of a tetrahedron based on its projected silhouette.

on nowadays graphics hardware. This is achieved by the z-buffer and the occlusion query extension which lets the programmer check whether fragments of a polygon were rasterized. The original selection sort exhibits a runtime complexity of $\Theta(n^2)$. The method proposed by Govindaraju et al. basically has the same worst case run time of $O(n^2)$, when working on lists that are sorted in the opposite direction that is desired. The advantage of their approach is, that, when the input sequence is nearly sorted, linear time complexity can be observed. Theoretically this approach can be extended to three dimensional elements, even though the number of polyhedra, which have to be sorted in larger unstructured volume datasets, is larger than the examples given in their work. Furthermore it is questionable, whether the approach remains applicable when scenes with extremely high depth complexity, like unstructured meshes of polyhedra, should be rendered.

Cell-Projection

Besides sorting approaches for the elements of an unstructured grid, the rasterization methods for the cells themselves are critical for proper volume rendering. The task all approaches have to fulfill is to generate a proper projection of a single cell. The opacity and color of this projection at a certain pixel is influenced by the thickness of the cell at the corresponding point and the color/scalar (depending on whether pre or post shading should be used) values of the entry and exit point of the faces of the cell. Because all polyhedra can be decomposed into tetrahedra (just like polygons can be decomposed into triangles) most cell projection algorithms only deal with the projection of tetrahedral cells. Besides this characteristic tetrahedra also have the property that the gradient of a scalar function given on their corner vertices is constant within their volume. This allows several simplifications which can be used to speed up the rendering process.

Projected Tetrahedra

Shirley and Tuchman [68] first proposed an approach to direct volume rendering of unstructured grids based on decomposing cells into tetrahedra and projecting those in a back-to-front order. Their method classified the cell which should be rasterized based

on its projected silhouette. The different classes are shown in Figure 2.8. Class one and two contain the most general cases whereas the other two remaining classes are degenerate cases which have to be treated separately for performance reasons. In order to utilize the triangle rasterization capabilities of modern graphics hardware Shirley and Tuchman proposed to decompose the projected tetrahedron silhouette based on this classification into triangles. The first case is split up into three triangles, the second into four, the third into two and the fourth into one. In order to create opacity and color values of the projected cell volume, correct $RGB\alpha$ values have to be assigned to the vertices of those triangles. All vertices of the original tetrahedron which are part of the silhouette are assigned zero alpha and are called *thin vertices*. The opacity for points which are generated by the triangle decomposition or are original cell vertices, but do not lie directly on the silhouette, has to be computed based on the thickness of the tetrahedron. These points are called *thick vertices* and represent the thickest part of the cell with respect to the viewing direction. The alpha and color values are then linearly interpolated over the area of the triangles by the rasterizer of the graphics hardware. This interpolation is an approximation because the opacity of a volume cell does not increase linearly with its thickness. Because the graphics hardware that was used by Shirley and Tuchman did not support texture mapping properly, the linear approximation had to be sufficient even though severe visual artifacts were introduced by this method. Stein et al. [73] presented an approach which uses a lookup texture to translate the linearly varying depth along the projected cell into the exponentially varying alpha value and thus avoid most of the problems the original PT algorithm had. Besides the thickness of the projected cell the opacity at the four original cell vertices influences the final opacity. For cells where the opacity is constant over all corners a single one-dimensional lookup texture is sufficient. When linearly varying opacity within the tetrahedron should be considered as well, a two-dimensional lookup texture has to be used. The function encoded in this texture is $1 - e^{-\tau l}$ with τ corresponding to the opacity and l representing the thickness of the cell. Max et al. [52] extended the projected tetrahedra approach to arbitrary polyhedra. They do not assume fixed decomposition classes, as it is possible with simple tetrahedas. Instead they generalize the classification algorithm to convex polyhedra which thus is able to produce polygonal regions across which the front and backfaces of a cell are the same. The further step is, as in the tetrahedral case, the rasterization of those regions with correct alpha and color values at their vertices.

GPU-Based Projected Tetrahedra

Wylie et al. [89] introduced an approach that extended the previously mentioned technique from Stein et al. to use the programmability of nowadays graphics hardware to perform the classification and proper triangle decomposition. Since a vertex program cannot add or remove a vertex and is not able to modify the topological structure of an input primitive, the maximum number of five vertices spanning up a triangle fan of four triangles has to be passed to the graphics hardware for each cell. This structure can be "morphed" into all silhouette classes of a tetrahedron by collapsing points to

one position and thus creating degenerate triangles. The graph representing it is called the *basis graph*. It is used within a vertex program to perform the mapping from the input, which consists of five dummy vertices and the corner points of the tetrahedron, which are loaded into vertex program parameters, to the output positions. Since the SIMD nature of vertex processing hardware did not support branching instructions at the time, all computations that are necessary to produce correct output positions and $RGB\alpha$ values, have to be performed for each of the five input vertices. Within the program a decision is made which result will be written into the output.

Preintegration

The previously mentioned approaches all use a two dimensional lookup texture to perform the interpolation of opacity values over the silhouette triangles. This limits the transfer function to mappings that transform the scalar data linearly into opacity values. Furthermore the color values are still interpolated linearly across the silhouette. If the mapping between the scalar data and a color value is not linear, this introduces additional errors. The approach Rottger et al. proposed [64] deals with this problem by introducing a three dimensional lookup texture which has to be precomputed based on a transfer function. The three texture coordinates, which are used to address this texture, are the scalar data value at the front face, at the back face and the thickness of the cell at the pixel. The only limitation that exist with this approach, is the resolution of the three dimensional texture which limits the frequency of the transfer function that can be used.

Extensions to Other Cell Types

Röttger and Ertl [59] propose an alternative method to PT for rendering unstructured grids, which uses a purely emissive optical model for the data volume and thus can save the cell sorting time. The necessity to decompose all cells of an unstructured grid into tetrahedra is problematic when complicated meshes have to be rendered which are composed of more complex volume elements like octahedras or hexahedras. Such decompositions mostly increase the number of volume cells to render by one order of magnitude. The algorithm presented by Röttger and Ertl is able to directly deal with arbitrary convex cells. Instead of decomposing the projected silhouette of a cell into triangles and computing the thickness values on the vertices, their approach is split up into three passes for each cell. The first two passes are used to compute the relative thickness of a volume element with respect to its maximum thickness. This means, that a value of one corresponds to the thickest parts of the cell and zero to empty space. With this trick the fixed point eight bit precision of the alpha channel in the frame buffer is sufficient to perform this computation during the rasterization. First the back faces of a cell are rendered with an alpha value representing the position within the interval between the maximum distance and the minimum distance of the cell to the view plane. After this first pass the second pass repeats this procedure with the front faces and an alpha combination function which subtracts the alpha value stored in the frame buffer from the newly generated one. The resulting value is now used in the third

pass to attenuate the color which is now added by rendering front and back faces.

2.4.2 Image-Space Approaches

If the number of volume elements, which make up a dataset, reaches the number of pixels of a result image, object-order approaches become less and less attractive because they always have to treat the complete set of cells. Furthermore the necessary visibility ordering of cells normally involves expensive sorting operations. The second group of approaches, dealing with the direct rendering of volumetric datasets, is image-based. This means, that the result images are not created by evaluating the contribution of every volume element to the volume rendering integral separately. Image-based approaches evaluate the complete volume rendering integral for the pixels of the result image separately without explicitly treating every single volume element.

In comparison to the field of structured volume rendering, image-based approaches are still rarely used when unstructured data volumes have to be visualized, because the number of cells is normally lower in this latter case. Still the amount of data specified on unstructured meshes rapidly increases with advances in simulation technologies and super computing, making image-based approaches more and more important.

Until recently dedicated graphics hardware was not able to handle the data structures which are needed to store the geometry information of unstructured data volumes. With the rapid development of graphics hardware in the last few years this limitation is starting to disappear and thus gives way to very efficient and fast implementations that can compete with state of the art object-order approaches.

Using the Adjacency Graph for Raycasting

Garrity was the first to publish an image-based approach which directly visualized volume data organized in unstructured grids [35]. In order to simplify the volume rendering the first step in his algorithm is to decompose all cells into tetrahedra. Since cells containing faces with four corners can be decomposed in multiple ways it is necessary to perform this decomposition with great care. The main problem for raycasting approaches is that the data volume has to be sampled at points along a view ray. This means that the volume element containing the sample position has to be retrieved from the dataset. In regularily structured datasets this is a trivial operation because the three-dimensional lookup position can be directly translated into a lookup index into the data array. When data is organized on unstructured volume meshes this is not possible because there is no direct mapping from world space into the data index space. Garrity solved this problem by exploiting the fact that a view ray exiting a volume element through one of its faces enters the volume element connected to this face. This means, that the process of sampling the data volume is reduced to finding the first cell intersecting the ray and tracking it through the volume elements based on connectivity information which has to be specified for every cell face. To find the volume element through which a ray enters the dataset, it is necessary to perform intersection tests between the boundary faces of the volume and the ray. To speed up this process Garrity utilized a spatial data structure into which the boundary volume elements were inserted.

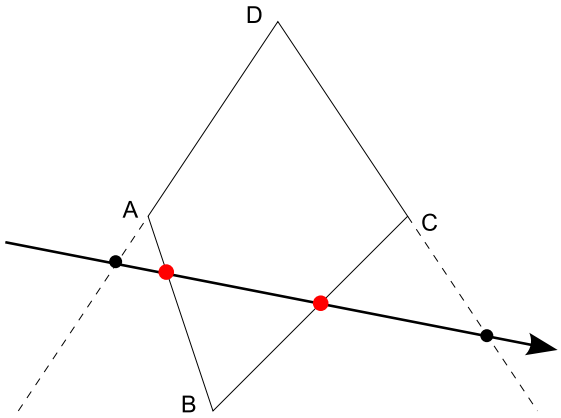


Figure 2.9: Cell-ray intersection. If the cell is convex the exit face of the ray is the first intersection behind the entry point.

Since unstructured volume meshes can be concave the rays cast through the data may exit and reenter the volume multiple times. To retrieve the face through which the ray reenters the dataset the already constructed aforementioned data structure can be used.

The determination of the face through which a ray exits a volume cell is simply accomplished by intersecting the ray with the planes of the faces. The plane intersection nearest to the entry point and, in the direction of the ray, behind it contains the correct exit face. This relation, which is illustrated in figure 2.9, is true for all convex cells.

Efficient Entry Cell Determination

The main bottleneck of Garritie's approach is the spatial data structure which is used to determine the entry cells of the viewing rays. Since screen space techniques work on a per pixel basis it is not necessary to determine the volume entry/reentry faces on a sub pixel precision level. Instead it is sufficient to know the boundary cells which are intersected by a view ray through a pixel center.

Bunyk et al. [11] exploited this by rasterizing the surface triangles of the irregular data volume and using a data structure comparable to the A-buffer [14] to store the boundary entry cell faces per pixel in a sorted list. Thus no costly search has to be performed when the entry/reentry faces have to be determined for a given pixel. The subsequent steps of Bunyk et al.'s approach are comparable to Garrity's method. The data volume is traversed by following the connectivity information which is specified per cell face.

Because image-based approaches treat each result pixel separately and the computations can be performed independently from each other they are highly suited to parallelization. Furthermore the operations which have to be performed per volume element are rather simple and do not involve complex looping or branching operations. Nowaday's graphics processing units are based on architectures capable of executing programs on many input values in parallel (MIMD). Thus the aforementioned properties of unstructured raycasting algorithms lend themselves very much to GPU-based

implementations.

GPU-Based Unstructured Grid Raycasting

The first realization of an image-based direct volume rendering algorithm on graphics hardware was proposed by Weiler et al. [80]. Their approach is primarily based on Garrity's method of traversing the data volume based on cell to cell connectivity stored for each face. To detect the cells through which the view rays enter the volumes they used Bunyk et al.'s rasterization technique. But because present graphics hardware does not support the A-buffer concept only the first entry faces can be determined by simple rasterization. Thus this first implementation was only able to deal with convex meshes directly. In order to still be able to render concave meshes they proposed a preprocessing step in which imaginary cells are inserted into the mesh to create a new convex data volume. In the context of cell sorting, which is also hampered by the existence of non-convex mesh portions, Williams [86] and later Röttger et al. [62] also proposed solutions to his problem. Weiler et al. use several screen-sized textures to store intermediate results of the raycasting algorithm:

- Current cell and face through which the has ray entered the cell
- Position of the intersection point
- Accumulated color and opacity

The connectivity information, the vertex positions and the scalar data itself is stored in several additional textures. After the convexification of the volume mesh and the initial determination of the entry face for each pixel the algorithm proposed by Weiler et al. iteratively performs the following steps several times.

- Compute exit point for current cell
- Determine scalar value at exit point
- Compute ray integral in current cell
- Blend with frame buffer
- Proceed to adjacent cell through exit point

All these steps are performed by rendering a screen-sized quad with a fragment program, reading from and writing into the intermediate textures. Thus one layer of cells can be processed by such a rendering pass. This implies that the number of passes necessary to display a complete dataset depends on the number of cells that have to be traversed in the view direction. The main drawback of Weilers et al.'s first approach was that the meshes had to be convex, if correct rendering should be guaranteed. Their convexification approach introduces additional cells which increased the dataset size and which had to be handled separately as so called virtual cells. Furthermore the data

structure storing the mesh information contained redundant information resulting in 160 byte per tetrahedral cell and severely limiting the size of the datasets which could be displayed. Thus the improvements proposed by Weiler et al. [81] mainly dealt with reducing the memory consumption and with non-convex meshes. Instead of storing vertex data for the tetrahedra separately they employed tetrahedral strips, which can be considered the three-dimensional equivalent to triangle strips, which are commonly used in computer graphics applications dealing with triangulated boundary meshes. The strip data-structure results in memory savings because only for the first tetrahedron within the strip four vertices have to be specified. All consecutive tetrahedra are defined by one new vertex. Similar to the two-dimensional case the way a tetrahedral strip is generated is crucial for the overall memory savings of the approach. Strips have to be as long as possible to maximize the efficiency of this data-structure. Since the problem of finding an optimal strip decomposition of a mesh (triangulated or tetrahedralized) is an NP-hard problem heuristic approaches have to be used. Several techniques that were developed for two dimensional triangle strips were considered by Weiler et al. and extended to tetrahedral strips. In order to perform raycasting on data represented as strips, it is necessary to store additional information for each tetrahedron, related to the connectivity of the cell faces which are not within the strip itself. The resulting data-structure reduces the original memory cost from 160 byte per tetrahedron to an average of only 15 byte per tetrahedron.

Besides improving the memory consumption Weiler et al. introduce a technique similar to depth peeling to cope with non-convex meshes. Like in the previous approach the mesh boundary has to be rendered in a first pass providing the starting positions and starting cell IDs to the further steps of the algorithm. When the ray-propagation is finished the boundary is rendered again with a second depth test applied to the rasterization of each fragment. This test is against the depth which was written in the previous boundary surface rendering pass. Thus with each subsequent pass the front most boundary surface is peeled away. In order to display a non convex mesh which contains n overlapping layers n passes have to be rendered. Espinha et al. [29] and Bernardon et al. [9] have also incorporated this technique into their approaches.

2.5 Level of Detail Methods for Unstructured Grids

Many volumetric datasets are too large for interactive visualization, even when state of the art rendering techniques techniques are used. In the section 2.3 approaches for dealing with large volume datasets on structured grids are described. In this section techniques to perform interactive visualization of large unstructured grids will be presented, which rely on creating lower resolution representations of the data. Basically two different approaches can be used to perform this data reduction which will be presented in the following sections.

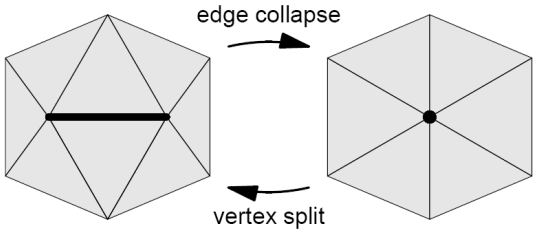


Figure 2.10: Most simplification approaches are based on the edge collapse/vertex split operation. [43]

2.5.1 Simplification

In order to reduce the available unstructured volume data, mesh simplification algorithms reduce the number of volume cells and produce a low resolution mesh which is still unstructured. The most straight-forward approach to perform this simplification is to perform atomic incremental simplification steps on the mesh until the desired level of detail is reached. All the simplification methods which rely on this approach work solely on tetrahedral meshes. Kraus and Ertl [43] propose using edge collapse operations to remove cells from a volume mesh. Here two adjacent vertices are merged which introduces degenerate volume cells which are removed (the two dimensional case of an edge collapse performed on a triangular mesh is shown in figure 2.10). Special care has to be taken when nonconvex meshes have to be simplified because hard-to-detect self intersections within the mesh may be introduced. Thus Kraus and Ertl propose to convexify the volume grid by introducing "virtual" tetrahedra, before applying their simplification method.

The preservation of isosurface topology is treated by Kraus and Ertl [44] for the case of structured meshes. Here critical points within the scalar field (minimum, maximum and saddle points) should be retained by the resampling procedure.

This isosurface topology guided downsampling is applied to unstructured meshes by Chiang et al. [16]. They propose to segment the data into topology equivalence regions which are simplified separately by edge collapse operations. Thus all isosurfaces which can be specified on the volume data retain their basic topology.

Cignoni et al. [17] propose a visualization system based on selective refinement. A refinement graph is generated whose nodes represent edge collapse/vertex split operations and whose edges represent dependencies between the mesh modification operations. The main focus of their work is the interactive selective refinement of portions of the mesh, which are presently of interest to the user. Different refinement strategies, like scalar field value based, based on a spatial user definable regions or based on the transfer function, are presented.

Yang et al. [91] also propose a visualization system which can perform mesh simplification interactively. But their main contribution is the application to meshes which are stored in a losslessly compressed format. Thus on the fly decompression, simplification

and rendering is performed in a pipelined fashion. Their approach is also based on a simplification graph which represents the interrelations between the mesh modification operations.

In order to perform view-dependent mesh simplification Röttger and Ertl [61] proposed to convert structured volume data into an unstructured grid. Thus they are able to create smooth level of detail transitions and prevent popping artifacts while navigating through the volume. Their main application is the non physically-based visualization of clouds over large terrains.

Separate Boundary and Volume Simplification

The approaches discussed above perform incremental mesh simplification starting from the original volume grid. This means, that aggressive downsampling makes it necessary to perform multiple time consuming operations on the mesh. The method proposed by Farias et al. [32] is different since most of the original mesh topology is discarded. They suggest to perform the mesh simplification in two parts. The first is to simplify the boundary mesh with an algorithm developed by Garland and Heckbert [34]. Then the vertices making up the volume are sampled and a new tetrahedral mesh is created by performing a Delauney tetrahedralization. Concavities have to be treated by the insertion of ghost vertices which convexify the data volume. This approach makes it possible to perform the downsampling faster, if stronger simplification is required. This is the opposite to the previously mentioned methods which take longer if smaller data volumes are needed.

Farias et al.'s approach may introduce problems when resampling the mesh boundary. Thus Uesu et al. [77] improved their algorithm to directly perform the resampling and retetrahedralization without the need for convexification. They propose to use conforming Delauney tetrahedralization, which was discussed by Cohen et al. [74] and Murphy et al. [54]), to perform the recreation of the simplified mesh. This algorithm takes as input not only the points which should be meshed, but also faces which should be present in the output grid. Thus the faces of the simplified boundary can be directly processed. In order to subsample the volume data points Farias et al. and Usenu et al. use a KD-tree to guarantee a proper sampling over the data volume domain.

Dynamic Level of Detail

The approach proposed by Callahan et al. [12] is completely different from the already discussed techniques. Until now every algorithm created a new simpler mesh geometry from the original data grid. Callahan et al. suggest to keep the original geometry and perform the subsampling directly on it. They select triangles from within a tetrahedral mesh which are rendered. This selection can be based on different criteria. Topology sampling creates peels from the dataset which are approximately parallel to the boundary surface. View-aligned sampling is performed by clustering the faces by their normals and displaying only the cluster which is as parallel as possible to the view plane. In order to do sampling based on the variations within the scalar field they introduce field sampling. And since bigger faces contribute more to the final image than smaller ones,

area sampling is used to consider them with higher priority.

2.5.2 Resampling

Besides the possibility to perform mesh simplification to create a simpler unstructured mesh there are approaches which perform resampling of the data into structured grids. Thus the faster and more efficient approaches which were presented in section 2.3 can be used to visualize the resampled data. Yagel et al. [90] propose to intersect the original volume mesh with view-aligned slices and render the intersection polygons. During interaction, the slices are not recomputed which enables the user to interactively modify the viewing parameters.

In order to perform also the resampling during interaction Westermann [83] proposes to use a GPU-based resampling approach. A modified version of the projected tetrahedra algorithm of Shirley and Tuchman [68] is used to rasterize slices of the data volume. The intersection computations are performed by exploiting the texturing capabilities of the graphics hardware. The resulting slices can be rendered directly or stored in a 3D texture.

This approach was improved by Weiler and Ertl [79] who use only the frontfacing cells of a tetrahedron to perform the cell - slice intersection, instead of using the Shirley and Tuchman triangulation. This reduces the number of polygons which have to be rasterized drastically. Furthermore Weiler and Ertl propose a second algorithm which performs the resampling on the CPU in an object-order fashion.

Leven et al. [48] suggest to perform the resampling in an preprocessing step and render only the structured data. Since performing the resampling into one structured data volume would either impose severe subsampling artifacts or extremely increase the amount of volume data they propose a hierarchy of grids. They use an octree with variable depth to arrange all structured grids. At each leaf node the highest resampled resolution is stored, while the internal tree nodes contain coarser versions of their children. Basically this data structure is comparable to the one LaMar et al. [46] propose. In order to perform interactive rendering Leven et al. suggest to use a three-level cache strategy, since not all of the data may fit into graphics memory or even main memory. The cache levels are the harddisk, the main memory and finally the texture memory of the GPU. The proposed rendering system tries to retain interactive frame rates by considering the time necessary to perform data transfers between the different cache levels.

The algorithms Stegmaier et al. [71] propose resample unstructured data into a hierarchy of structured grids. The hierarchy itself can be specified manually or automatically. In order to render the resampled data a conventional visualization system is used.

Splatting of Scattered Data

Besides performing the resampling into a structured grid it is possible to treat the volume data as connectionless scattered data points. Thus Meredith et al. [53] propose to do splat-based volume rendering, which was introduced by Westover [85]. They

use symmetric Gaussian filter kernels to perform the splatting of the cells. In order to support progressive refinement they utilize an octree whose internal nodes can be represented by one splat. The leaf nodes contain a fixed maximum of cells of the original mesh. The splat size within one octree node is constant and is based on the bounding box size.

2.6 Graphics Hardware Basics

In the previous sections some general information about volume rendering and visualization was presented. The need for fast and accurate rendering was stated and the difficulties when dealing with volume data on unstructured grids were mentioned. This section focuses on the basic architecture of modern graphics hardware which can be used to tackle some of the problems which arise when large amounts of volume data have to be visualized interactively.

The following subsection deals with some general information about the rendering pipeline which is available in all common graphics sub systems. After this the extended programmability of this pipeline, which was introduced in recent years, is presented. Furthermore limitations and important performance critic aspects are discussed.

Rendering Pipeline

Like the CPU the graphics processing unit (GPU), which is the core of modern graphics hardware, uses a multi stage pipeline to increase its efficiency. Furthermore several stages of this pipeline are available multiple times within one GPU which allows for parallelization of rendering tasks. The basic (simplified) structure of the rendering pipeline can be split into three parts:

- Geometry processing
- Fragment processing
- Raster operations

Figure 2.11 gives a basic overview over the rendering pipeline and the dataflow which is possible within it. An application can communicate with the graphics hardware by sending geometry which should be rendered through an API like OpenGL [7] or DirectX [1]. From there on the GPU starts its work by transforming the geometry, which is a list of triangles in most cases. In the basic case this transformation projects the geometry from the space where it is defined, which is called object space, onto a two dimensional viewport.

When the vertices of the geometry have been transformed, its primitives have to be rasterized in the fragment processing stage. The input to this pipeline step are so called fragments which can best be described as "future pixels with some additional information". For every pixel which is covered by the geometry a fragment is generated and sent into the fragment processing stage. Data attributes, which are specified per

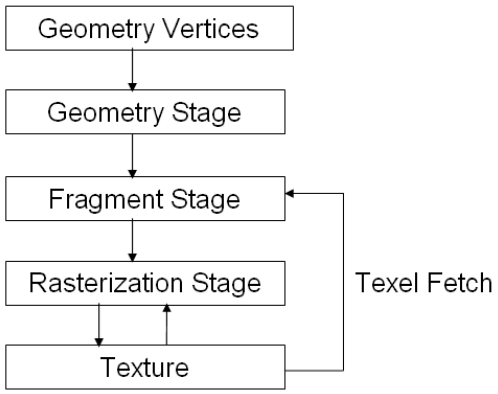


Figure 2.11: The basic structure of the rendering pipeline. Data flows in the direction of the arrows. The fragment processing stage is the only programmable stage which can access texture data efficiently.

geometry vertex, like color for example, are interpolated over the geometry's surface and passed along with each fragment. The output of this stage is normally a color, transparency and depth value for each fragment which is then passed to the final part of the rendering pipeline.

The raster operations which are finally performed on the data coming from the fragment processing stage decide if it is written into the frame buffer, which holds the rendered image. Input data can simply be discarded, overridden or combined with data already present in the frame buffer.

Programmability

The main reasons for the recent popularity of algorithms which use the GPU to perform computationally expensive tasks are the extremely high memory bandwith of modern graphics sub systems and the availability of massive parallel processing resources which become more flexible with every new hardware generation. Thus tasks involving many memory accesses and which can be parallelized efficiently mostly perform better on a GPU than a CPU. The main drawback of this highly specialized hardware can be derived directly from its advantages. Since the high performance mainly comes from a massive parallel pipelined architecture it is difficult to perform complex computations which rely on global results or large data structures.

Nowadays the geometry and the fragment processing stage are programmable, even though the degree of freedom for the programmer is still different in each pipeline stage. Programs can be more complex at the beginning and have to be simpler at the end, since the frequency of program execution is relatively low in the geometry stage and very high in the fragment stage. Geometry related programs (so called vertex programs) are

executed for every geometry vertex and can output the transformed vertex position and additional vertex attributes like color, which will be interpolated and used as input to the fragment processing stage. Programs controlling this part of the pipeline are called fragment programs. They are executed for every fragment and have to output color, opacity and depth values which are passed to the raster operations stage. Here severe limitations like limited indirect data access capabilities and limited dynamic branching possibilities pose some difficulties. Still the fragment processing stage has very high potential since it has the fastest and most direct connection to the video memory which allows for high data throughput. Furthermore the number of fragments which can be processed by clock cycle is far higher than the amount of vertices. Thus most GPU algorithms, including the implementation presented in this thesis, mainly rely on this pipeline stage to perform their computations. This can be accomplished by storing input data in a set of textures and rendering a screensized quad using a fragment program which should process the data.

Chapter 3

A System for Hybrid Raycasting

Performing direct volume rendering by casting a view ray through the data volume and accumulating color and transparency information along the way is a conceptually simple method which can be easily used to visualize structured data volumes. Contrary to the structured case unstructured grids pose severe problems, when raycasting should be used to display data specified on them. Thus the first part of this chapter focuses on efficiently solving the problems, which are introduced by unstructured data volumes. Additionally efficient solutions to solving the emission and absorption optical model for the structured and unstructured case are discussed.

The overall hybrid raycasting algorithm can be roughly divided into functional units, which differ by the frequency they are used during a visualization session. The volume decomposition for example is necessary only once whereas the raycasting has to be performed every time the user interacts with the system. Section 3.2 gives an overview over the different parts of the hybrid raycasting algorithm and their interaction with each other is discussed. After this the optical models which can be selected by the user to perform the volume visualization are presented. Since the surface of the simulated domain is often of interest to the engineer different boundary surface visualization techniques are introduced as well.

3.1 Raycasting Basics

Raycastingbased volume rendering methods are based on a simple concept. View rays are cast through each image pixel and the contribution of the data volume along this ray is calculated and written into the result pixel, as depicted in figure 2.2.

Performing raycasting on scalar volume data is an image space based approach. Its runtime can be expressed with the following equation $O(p \cdot \sqrt[3]{n})$. Here p is the number of result pixels and n the number of cells within the data volume. In most publications this dependency is neglected and image space based methods are classified as basically dependent on the number of output pixels. Still, when large data volumes have to be considered the depth or thickness of the data volume becomes a factor which cannot be ignored. $\sqrt[3]{n}$ is a good approximation for the number of cells along the viewing direction,

assuming a uniform cell distribution along all axes.

The previous chapters provided a short very general introduction to the problems which have to be tackled when dealing with volume data and the state of the art of methods which are in use. The following sections will give deeper insight into the GPU-based raycasting approach presented in this thesis. First practical approaches to the evaluation of the optical volume model will be presented. The next section deals with the point location problem discussing it for the structured and unstructured volume case.

3.1.1 Evaluating the Optical Model

The proper application and evaluation of an optical model during raycasting is crucial to the final image quality. Most interactive volume rendering approaches use a model which incorporates light emission and basic absorption which was already briefly discussed in the introduction. In order to evaluate an optical model over a discretized data volume the discrete volume rendering integral has to be considered:

$$I = I_0 \cdot \prod_{t=0}^{n}(1-\alpha(t)) + \sum_{t=0}^{n} C(t)\cdot\alpha(t)\cdot\prod_{u=0}^{t-1}(1-\alpha(u)) \tag{3.1}$$

When a discrete data volume is considered the sampling functions have to be evaluated at discrete positions within the data volume. Thus every sample represents the accumulated transparency and color of one ray segment from the last sampling position to the current one. The sample positions themselves can be chosen in two basic ways. The first is to perform the sampling at arbitrary successive positions along the viewing ray. The sampling intervals can be adapted to the underlying scalar data frequencies but are mostly constant when GPU-based raycasting is used. The second way to sample along a ray is to choose the intersections of the volume cell faces with it as the sample positions. Basically the decision which sampling approach to use has to be based on the interpolation method which is applied within the data volume.

If the scalar within one cell varies non-linearly along the view ray within one ray segment an analytic solution of the optical model within one cell becomes highly complex or even impossible. This happens, if either higher order interpolation methods are used, or if one ray segment spans several volume cells. In this case the first sampling approach is used. To produce high quality images the data volume has to be sampled finer than the smallest cells along the ray. Because of the nonlinear variation of the scalar data within the ray segment and the mostly non-linear mapping of the scalar field to color and opacity some simplifying assumptions are made by all interactive approaches. The simplest is to consider the scalar constant within one sample interval. Thus a high sampling rate has to be used if good image quality is required. Now the functions $C(t)$ and $\alpha(t)$ can be evaluated:

$$\alpha(t) = 1 - \exp\left(l(t)\cdot tf_{\alpha}\Big(s\big(r(t)\big)\Big)\right) \tag{3.2}$$

$$C(t) = tf_C\Big(s\Big(r(t)\Big)\Big) \tag{3.3}$$

Here tf_C and tf_α denote the transferfunction mapping scalar values to color and α respectively. The function $s(x)$ retrieves a scalar value from the data volume at position x. The function $r(t)$ returns the sampling position along the ray for the sample t. $l(t)$ denotes the length of the ray segment t. A more accurate approach is to consider the scalar as varying linearly between two sampling positions. Here a pure analytic evaluation of $\alpha(t)$ and $C(t)$ is often not possible since the transfer function is in most cases not specified analytically. Thus a lookup table is used which is created in a preprocess. This so-called preintegration table, which was suggested by Engel et al. [28] for structured and Röttger et al. [64] for unstructured grids, is indexed by the start and end scalar values of the ray segment. If the sampling intervals are nonuniform, an additional lookup dimension, the segment length, is necessary.

Sampling at the volume cell face intersections with the view ray is mainly used if the scalar data is really constant or varies linearly within the volume cells. Constant data values mean that no interpolation within the cells is performed. Here the already mentioned analytic solution can be used. In the case of tetrahedral meshes barycentric interpolation results in linear scalar variation and a constant gradient within a volume cell. This means that the preintegration method with a three-dimensional preintegration table can be applied.

This means that raycasting on structured grids mostly uses a large amount of samples at constant distances along the view ray. In the unstructured case the sampling is almost always performed at the cell boundaries. The unstructured grid raycaster presented in this thesis assumes no data interpolation and thus uses the analytic optical model. For the resampled structured grids the same model is used.

Until now the optical model was treated as one single equation for the whole view ray. In order to efficiently evaluate it, an incremental compositing scheme has to be used (a naive approach would have to reevaluate the product $\prod_{u=0}^{t-1}(1-\alpha(u))$ in Equation 3.1 over and over for each sample.) . There are two basic ways to compute the final color contribution of all ray segments along one view ray when the emission and simple absorption is considered. The first and most straight-forward method is to perform the computations in a back to front order. Let n be the number of samples along a ray and $i = 1$ the first sample. Furthermore the sample n corresponds to the the last one in viewing direction. All consecutive samples are farther away from the viewer than the previous one. Then the following incremental definition can be used to compute the final color I_{n-1}:

$$I_0 = C(n) \cdot \alpha(n) \tag{3.4}$$

$$I_i = C(i) \cdot \alpha(n-i) + I_{i-1} \cdot (1 - \alpha(n-i)) \tag{3.5}$$

This compositing scheme requires, that all samples along a view ray have to be considered. In many cases, especially if highly opaque transfer functions are used, the first few samples influence the final color only slightly. Thus the second compositing method performs the computations in a front to back order, making it possible to stop the accumulation when barely any change in the result is made by subsequent samples. In order to derive this front to back compositing scheme the overall equation has to be revisited. The term

$$\prod_{u=0}^{t-1}(1-\alpha(u)) \tag{3.6}$$

has to be evaluated for all the samples along the view ray with t being the index of the present sample. Since the samples are ordered in a front to back order this function can be evaluated incrementally.

$$T_0 = 1 \tag{3.7}$$

$$T_i = T_{i-1} \cdot (1-\alpha(i)) \tag{3.8}$$

Basically T_i represents the transparency which is remaining after i samples. Thus if the value of T_i is nearly zero all subsequent datasamples only contribute minimally to the final image. Thus a small $\varepsilon > 0$ can be selected which is used as a threshold value for the opacity. If $T_i < \varepsilon$ no further compositing is necessary and the computation for the view ray can be stopped. The incremental formulation of the front to back approach thus is:

$$I_0 = C(0) \cdot \alpha(0) \tag{3.9}$$

$$I_i = T_i \cdot \alpha(i) \cdot C(i) + I_{i-1} \tag{3.10}$$

The raycasting approach presented in this thesis uses this front to back compositing scheme combined with early ray termination when the remaining transparency is below a user definable value.

3.1.2 The Point Location Problem

The problem of finding a cell of the discretized data volume which contains a certain point is called the "point location problem". In the case of regularly structured data the solution is trivial. Let $\mathbf{p} \in \mathbb{R}^3$ be a point within the data volume and $\mathbf{c} \in \mathbb{R}^3$ the cell sizes for all cells in the according dimension. Then the indexing vector $\mathbf{i} \in \mathbb{N}^3$ can be computed as follows:

$$\mathbf{i} = \begin{pmatrix} \lfloor p_x/c_x \rfloor \\ \lfloor p_y/c_y \rfloor \\ \lfloor p_z/c_z \rfloor \end{pmatrix} \tag{3.11}$$

This index vector can now be used to directly access the cell which contains the point within the structured grid since its components correspond to the row, column and slice within the dataset. Thus retrieving samples from this data organization scheme is very simple and a raycasting algorithm working on this kind of grid does not have to tackle many difficulties when retrieving samples from the underlying data.

When samples from within an unstructured grid have to be retrieved, the mapping from point location to cell index is not as straight-forward as in the structured case. As already mentioned in the introduction unstructured grids are indexed by a one dimensional indexing vector $\mathbf{i} \in \mathbb{N}$ which normally is not related to the spatial location of the corresponding cell. Thus, if the indices of cells containing arbitrary points should be retrieved, additional spatial auxiliary data structures would be needed. Otherwise all cells within the dataset have to be checked wether they contain the point in question. Early raycasting techniques [35] relied on an octree to retrieve the cells through which a view ray enters the data volume. All subsequent cell indices are then retrieved without the need for a lookup in the spatial data structure. This is possible because the samples have to be retrieved in a front to back or back to front order from within the dataset when raycasting is performed. If the first cell along the ray is known the next cell has to be connected through a face to the current one. Thus it is only necessary to check through which face the view ray exits the current cell and retrieve the neighbor at that face. This means, that an additional graph has to be stored whose nodes correspond to the cells of the dataset and whose edges represent the faces of the cells. The intersection calculations necessary to determine the cell face through which a view ray exits can be formulated by the following vector equations.

$$\mathbf{n} \cdot (X - P) = 0 \tag{3.12}$$

$$X = E + s \cdot \mathbf{v} \tag{3.13}$$

Here P is a point on the cell face, E a point on the view ray, $\mathbf{n}$ the normal vector of the cell face and $\mathbf{v}$ the direction of the view ray. X is the unknown intersection point between cell face and view ray and s is an unknown scalar. Solving this for s results in the following equation.

$$s = \frac{\mathbf{n} \cdot P - \mathbf{n} \cdot E}{\mathbf{n} \cdot \mathbf{v}} \tag{3.14}$$

s has to be evaluated for all the faces whose normals point in the same direction as the view direction, which equals $\mathbf{n} \cdot \mathbf{v} > 0$. The face with the smallest $s > 0$ is the exiting face. It should be noted that this is only true for convex cells, which can be assumed for nearly all datasets which are produced by nowadays simulation packages.

3.2 The Raycasting Framework

The approach presented in this thesis is an extensible framework which allows for an easy implementation of additional visualization methods and is fully integrated into the

SimVis framework. It performs direct volume rendering of one scalar data attribute and a user-selectable DOI field in combination with a visualization representing the boundary of the data volume. In this section a very brief overview over the hybrid raycasting algorithm is given.

Figure 3.1 shows the basic pipeline, which is used to visualize volume data specified on unstructured grids. The first step is the volume subdivion. Here the data volume is divided into cubic portions, which are inserted into a spatial hierarchy. This is similar to the bricking approaches presented in section 2.3. One of the contributions of the work presented in this thesis is the adaption of a spatial subdivision technique to unstructured grids. Since the cells from unstructured grids are not axis aligned, it is necessary to clip them at the brick-boundaries. This is also performed in the volume subdivision step.

Since every brick can be treated separately by the raycaster it is possible to handle their visualization differently. The decision of how a brick should be rendered is made in the brick classification step. The entropy of the DOI function is presently used to decide, whether a high- or low-quality representation of a brick should be used. The high-quality version of a brick is its original unstructured grid, whereas a resampled low resolution version of the brick volume specified on a structured grid is used if a low-quality representation is appropriate. If a brick does not contain any data at all, which is the case if the corresponding DOI function is zero for the whole brick volume, no volume data has to be stored at all.

Now that the a rendering mode has been selected for each brick the next step in the pipeline is the direct volume rendering. For every brick the visualization of the entry- and exit-surface is combined with the volume rendering of the scalar data and the DOI volume. The the following transfer function is applied:

$$tf_{DOI}(d, s) = d \cdot tf(s) + (1 - d) \cdot l \tag{3.15}$$

Here d denotes the DOI values, s the scalar field values, l the luminance which should be used for the context portions of the data and $tf(s)$ is an arbitrary one dimensional transfer function. The α fraction of the transfer function is basically the DOI value. The hybrid raycasting method presented in this thesis has to use three different visualization algorithms corresponding to the three representations of a brick (unstructured, structured or empty). One of the contributions of this work is the algorithm used to render unstructured bricks, which is able to cope with meshes, which may contain different types of cells (previously published approaches perform a tetrahedralization before the visualization). To generate the final visualization the images resulting from the raycasting of the individual bricks have to be composited in a front to back order.

Since the SimVis system relies heavily on user interaction during a visualization session, the proposed rendering method has to properly react to user input. There are two basic interaction methods. If the viewport is changed only the last pipeline step has to be executed. When the presently visualized DOI function, or scalar data attribute changes the brick classification has to be performed again as well.

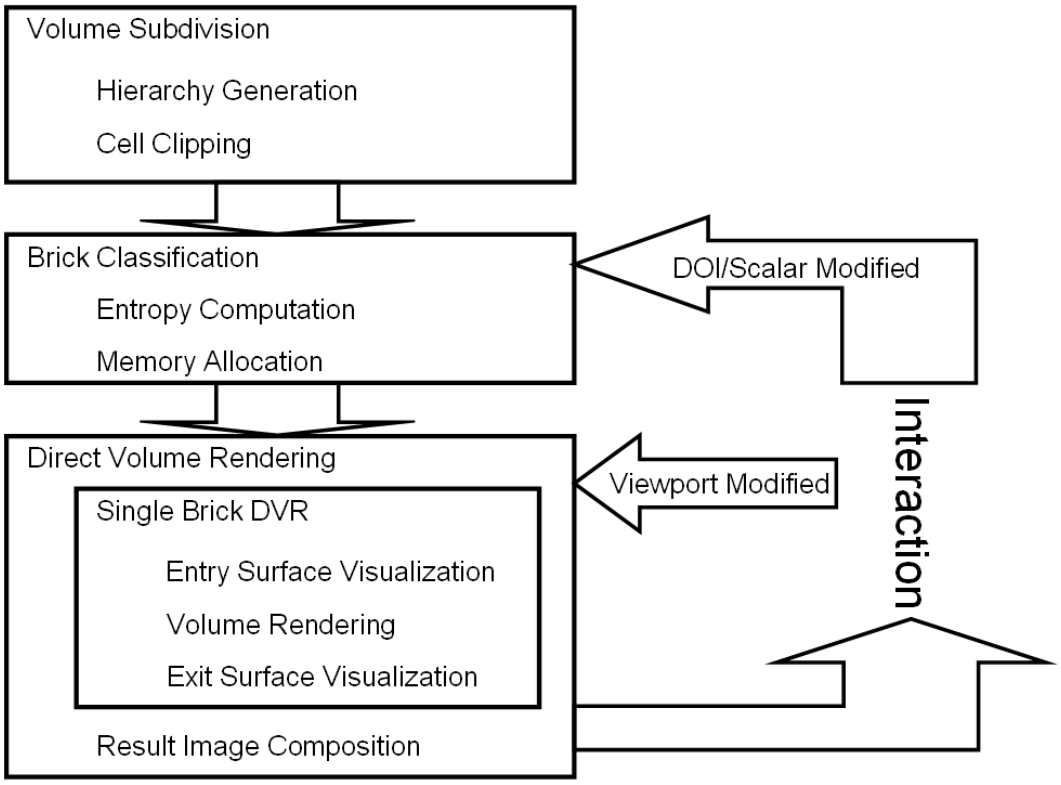


Figure 3.1: This diagram shows rendering pipeline of the the hybrid raycasting algorithm.

3.3 Rendering Modes

There are two basic ways to influence the visualizations which are produced by the hybrid raycasting algorithm. The first is the selection of an optical model, which is used to perform the volume rendering. Presently the user can choose between five different models, which are described in section 3.3.1. Additionally there are three different volume boundary visualization techniques available. All optical models can be combined with all boundary surface visualization techniques resulting in 15 different rendering modes which can be selected by the user.

The final visualization, which is produced by the optical models and the boundary surface visualization methods can be controlled by several user definable settings. Furthermore the rendering framework supplies additional parameters, which are derived automatically during the raycasting process. Table 3.1 presents a short overview over all parameters, which are used by the different optical models and boundary visualization methods. It is important to note, that all parameters can be freely interpreted by the different rendering modes. Still a loose semantic is attached to every input parameter, which should be taken into account by the rendering modes. f_α can be used to influence

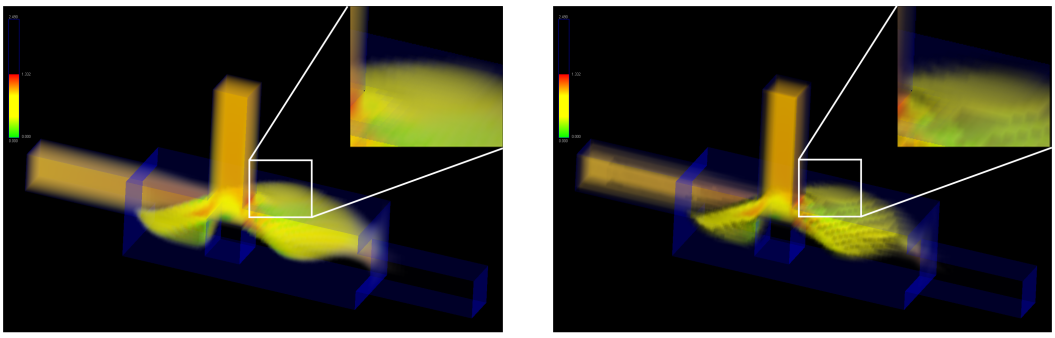
User Definable	
f_α	Alpha value for the focused data
c_α	Alpha value for the context
c_l	Luminance of the context data
b_c	Boundary color
$tf(s)$	A user selectable transfer function
tf_{min}	The lower bound, which is used when applying a transfer function
tf_{max}	The upper bound, which is used when applying a transfer function
Fixed	
t_c	Thickness of a sample in object space
t_b	Thickness of a sample clamped to the brick boundary
s_{max}	Radius of the largest cells bounding sphere
$tf_w(s)$	$\frac{tf(s)-tf_{min}}{tf_{max}-tf_{min}}$ Windowed transfer function

Table 3.1: This table shows the different parameters, which are used as input to the optical volume models and the boundary visualization. User definable parameters can be specified by the user whereas fixed parameters are supplied by the framework.

the transparency with which the focus portions of the data should be visualized whereas c_α should determine the transparency of context information. The luminance value, which should be used to represent context information is determined by c_l. b_c is the color a volume boundary visualization has to use to represent the mesh surface. Furthermore $tf(s)$ is a transfer function, which should be applied to the underlying scalar field values and which can be windowed by the parameters tf_{min} and tf_{max}. The input parameters, which cannot be specified by the user are the thickness of a sample t_c and the thickness of a sample clamped to the boundary of the current brick t_b. Additionally s_{max} represents the radius of the largest cells bounding sphere and $tf_w(s)$ is the already mentioned transfer function windowed using the tf_{min} and tf_{max} values. The following sections present all currently available rendering modes and the exact way, they use the input parameters.

3.3.1 Optical Models

The structured and unstructured raycasting methods perform the evaluation of the applied optical model by sampling the scalar field volume data and the DOI function. All input parameters which are available to the optical model are listed in table 3.1. It is important to note, that a DOI function which is zero has to result in complete transparency. This is necessary because bricks for which the DOI function equals zero over the whole volume are rendered without a raycaster. Instead only the boundary visualization is rendered, which saves large amounts of rendering time and memory resources.



(a) emission and absorption (b) shaded emission and absorption

Figure 3.2: Overview over the most commonly used optical models.

Basic Emission and Absorption

The first, most basic optical model, shown in figure 3.2(a), performs simple volume integration based on emission and absorption. It implements the two dimensional transfer function already introduced in equation 3.15. In order to produce coherent images the maximum cell diameter is used to normalize the clamped ray segment length before evaluating the α value of the sample. Thus the following formula is used by the optical model to compute an alpha value.

$$\alpha = 1 - \exp\left(-\frac{d \cdot f_\alpha \cdot t_b}{s_{max}}\right) \tag{3.16}$$

Here d is the DOI value of the current sample. f_α is used to steer the overall transparency.

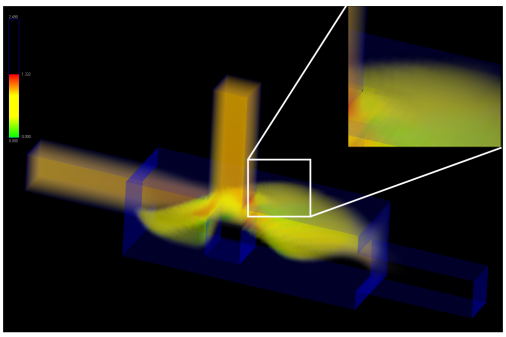
The color is evaluated based on the two dimensional transfer function, which is based on the current DOI and scalar field value of a sample. The following equation is used to compute the final color.

$$c = d \cdot tf_w(s) + (1 - d) \cdot c_l \tag{3.17}$$

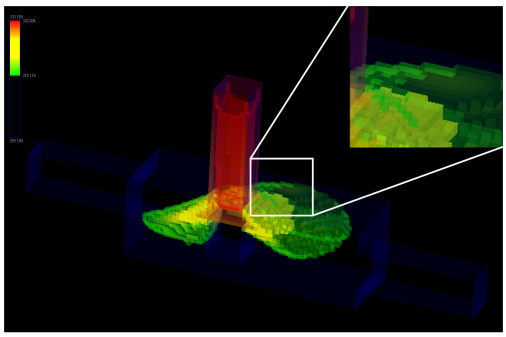
Basically the color, which is specified by the windowed transfer function is blended more and more towards a color between black and white (which is specified by the red, green and blue components equaling c_l), the lower the corresponding DOI value is. Thus the lower the DOI value the more transparent and less saturated colors are used. This optical model is most commonly used if high quality direct volume rendering results should be generated.

Emission and Absorption Approximated

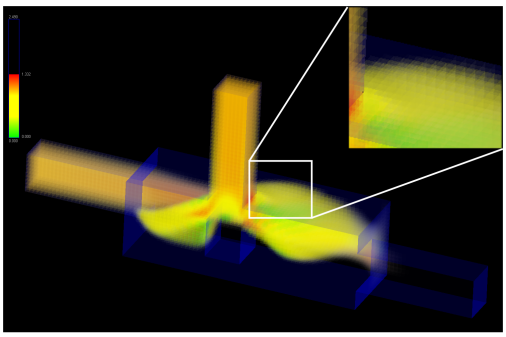
If a linear approximation to the α value is sufficient the second optical model, shown in figure 3.3(a) can be used. It uses the same equation as the previously mentioned model to compute the color values. The α value is computed by a linear approximation to the volume integral.



(a) emission and absorption approximated



(b) multi isosurfaces



(c) cell faces

Figure 3.3: This figure shows additional optical models, which are available to the user.

$$\alpha = \frac{d \cdot f_\alpha \cdot t_b}{s_{max}} \tag{3.18}$$

Here the expensive evaluation of the exponential function can be omitted which results in better performance. This model object can be applied if fast rendering speed is necessary during interaction. Its inferiority to the previously described solution is only visible if very large cells are contained within a dataset.

Shaded Emission and Absorption

The third optical model, shown in figure 3.2(b) is also based on the first one. The difference is, that a simple lighting model is applied to the volume data. The lighting is based on the spatial gradient ∇d of the DOI scalar field which is approximated in the optical model. The light direction is assumed parallel to the view ray $\mathbf{v}$ and bidirectional. Thus the color for a sample is computed by the following formula.

$$c = (d \cdot tf(s) + (1 - d) \cdot c_l) \cdot \underbrace{\left(abs\left(\mathbf{v} \cdot \frac{\nabla d}{|\nabla d|} \right) \cdot i_d + i_a \right)}_{lightcontribution} \tag{3.19}$$

The lighting model incorporates a constant ambient and a diffuse contribution i_a and i_d. Since the DOI function changes frequently and the SimVis system does not support gradient derivation, the gradient approximation is incorporated into the optical model and evaluated during the ray traversal on the GPU. In the structured case a central difference approximation is applied. Since only very limited information about the unstructured grid itself is available to the optical model and only the direction of the gradient is important for the lighting a very simple and fast method is used. Per cell face the face normal is scaled by the difference between the present DOI and the neighbors DOI and accumulated. The result is a good approximation to the DOI gradient if the cell sizes are comparable. With this optical model fine details in the DOI function can be discerned.

Multiple Isosurfaces

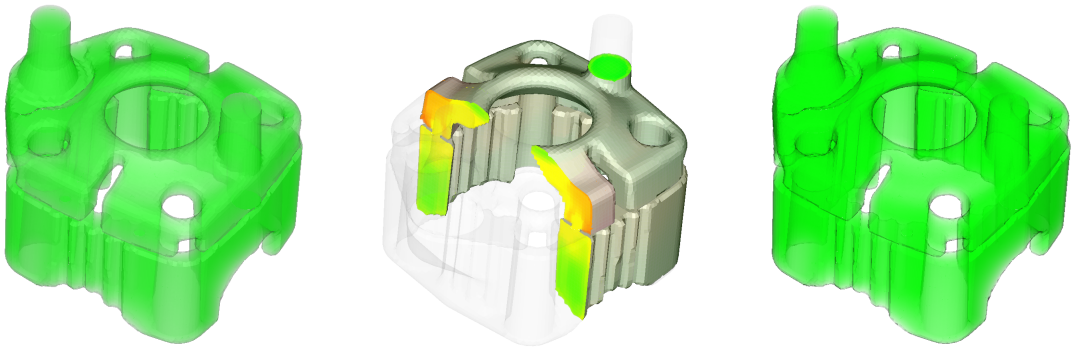
The fifth optical model, shown in figure 3.3(b) creates an isosurface visualization. But instead of one surface, three are used. This approach has already been proposed by Hauser and Mlejnek [39] in the context of the SimVis system. The user can specify an interval on the scalar field values by selecting the tf_{min} and tf_{max} parameters. The center of the interval specifies the primary isosurface, which is rendered most opaquely. The upper and lower bounds of the interval specify two additional isosurfaces and are rendered more transparently than the primary one. This helps the user to get an overview over the gradient of the field value. The closer the isosurfaces lie together the larger the gradient. Additionally the DOI function modulates the transparency of all three isosurfaces.

Cell Faces

The last optical model, shown in figure 3.3(c), which can be selected by the user is the simplest. It does not adapt the color and opacity of a sample to its thickness. Thus the resulting visualization shows the cell boundaries. This optical model is useful if the user wants to check the cell structure of the unstructured grid. Since the underlying unstructured grid data is necessary for this optical model to work properly it produces no output for resampled structured bricks.

3.3.2 Boundary Visualization

Like the different optical models which are used to evaluate color and α values for ray segments, boundary visualization methods are used to calculate color and α contributions for the mesh surface. The information which is passed to the boundary visualization methods are the DOI d of the boundary cell, the face normal $\mathbf{n}$ of the exterior face and the view direction $\mathbf{v}$. Furthermore the already mentioned visualization control parameters, which are also passed to the optical models (the focus alpha f_α value, context alpha c_α value and the context luminance c_l) are supplied to the boundary visualization methods as well. It should be noted that all bricks in all three states (unstructured,



(a) shaded boundary (b) clipped boundary (c) silhouette boundary

Figure 3.4: This figure shows all three boundary visualization methods applied to the small cooling jacket dataset.

structured or empty) use the same boundary visualization method to perform their boundary visualization in order to guarantee homogenous results.

Shaded Boundary

The most basic boundary visualization method, shown in figure 3.4(a), performs simple diffuse shading of the mesh surface using b_c as underlying color. The transparency is controlled by c_α. As in the shaded volume rendering optical model bidirectional diffuse lighting in view direction is used. This method is especially useful if the surface of the simulated domain is highly complex and the silhouette method produces too much cluttering.

Clipped Boundary

The second boundary visualization method, shown in figure 3.4(b), uses the DOI of the current cell to interpolate the transparency of the surface between f_α and c_α. The same shading as in the first object is used. Thus the boundary of cells with a DOI of 1.0 are rendered with an alpha value of f_α and cells with $d = 0.0$ are rendered with an alpha value of c_α. Thus parts of the boundary can be clipped away by the user by specifying cells whose surface should be visible.

Silhouette Boundary

The last boundary visualization method, shown in figure 3.4(c), tries to render only the silhouettes of the mesh boundary. This is accomplished by modulating the transparency of the rendered surface depending on the angle between $\mathbf{n}$ and $\mathbf{v}$. Basically it is enough to use the cosine of the angle between the two vectors to modulate the transparency of the surface. In order to define the boundary more explicitly $abs(\mathbf{v} \cdot \mathbf{n})$ is taken to the power of three and multiplied by the context transparency c_α to compute the final α

value. The unmodified b_c color is used for the boundary visualization. Contrary to the simple shaded boundary visualization method the silhouette visualization hides only a minimal amount of information. Thus it is suited if a very clear visualization should be created. The main problem of silhouette rendering is cluttering if highly complex and multi layered mesh surfaces have to be displayed.

Chapter 4

Rendering Data Structures

There are several requirements which are imposed on a rendering frontend for a visualization system. Basically the available hardware is the limiting factor for any software. Here the available main and especially graphics memory is of high interest. Furthermore the processing time on the CPU and GPU is naturally limited as well. Because the rendering frontend presented in this thesis is part of a larger visualization system those limitations are even more strict since the system itself needs resources as well. Thus an appropriate data organization scheme has to be used which can be adapted to the available resources.

A GPU-based approach, like the one presented in this thesis, basically requires all the data, which should be processed at once, to be present in the graphics memory. Since normally this memory is smaller than the main memory it is necessary to split up the dataset, as suggested by the techniques which are mentioned in section 2.3 for structured grids. The following sections will discuss limitations imposed by the data processing capabilities of the GPU and extend the bricking scheme from structured to unstructured grids. Furthermore the necessity for memory management will be discussed.

4.1 Data Organization

In order to speed up the rendering process and to save memory many visualization algorithms utilize different data structures. In the structured case mostly octrees are used while unstructured grid rendering algorithms often rely on mesh simplification graphs. The approach proposed in this thesis has to deal with structured and unstructured data and thus a spatial subdivision scheme combined with a hierarchical data structure, comparable to an octree, are used. Every brick within this data structure has to provide the rendering algorithm with all the data describing the contained data volume.

This section will discuss limitations imposed on the data organization scheme by the GPU. Furthermore the decomposition of unstructured data into bricks will be presented. After this the storage data structures will be elaborated in detail.

4.1.1 GPU Limitations

Graphics hardware can basically be used in two ways when complex computations should be performed. The difference is how strong the vertex and fragment processing capabilities are involved. Either mainly the fragment processing units or both the vertex and fragment units are used. Both parts of the graphics pipeline access data in different ways. The vertex processing units can only read data which is specified per vertex. No neighborhood information about the vertices is available within this pipeline stage. Thus algorithms relying on vertex processing capabilities often have to replicate data across vertices. Many object order methods to render unstructured grids, which are presented in section 2.4.1, are such approaches.

Raycasting techniques for structured and unstructured grids mainly use the fragment processing stages. The data access capabilities are better in this pipeline stage since data can be accessed from arbitrary locations within textures. This means that a data structure representing the volume grid and its data can be stored in the form of multiple textures. A texture is basically a one-, two- or three-dimensional image which traditionally has to have a width, height and depth which are a power of two in the number of pixels. Even though there are extensions to graphics APIs which relax this limitation [5] it is still suggested to use power of two dimensions since the data access in such textures is faster. This implies that inevitably memory is wasted. Reducing this excess memory wasting is extremely important for an efficient data structure for raycasting-based methods which have to work on a GPU.

Textures can have different formats and different internal precision which are fixed to a set of supported combinations. In order to store complex data structures within textures it is necessary to consider these limitations as well and choose formats and precisions which are suitable. Basically a texture can contain one to four scalar values per texel which are mostly interpreted as luminance, α, luminance and α, RGB or $RGB\alpha$ values within a fragment program. The precision of all the data within a texture has to be constant, which means that a RGB texture has to store all three values with the same number of bytes. There are exceptions to this limitation which will not be discussed any further since such texture formats/precisions will not be used in the proposed algorithm. The precisions which are most commonly used are either fixed or floating point. Fixed point numbers are mostly represented by one, or two bytes whereas floating point numbers are either standard IEEE single precision floats or two byte floats where 5 bits are used for the exponent, 10 bits for the mantissa and one bit for the sign.

Thus the drawbacks of storing a data volume within textures is that the texture dimensions are limited to certain sizes and that the whole texture data has to be present within the local memory of the GPU. This is not necessary for vertex data onto which no size limitations are imposed and which can be streamed directly from main memory during the rendering.

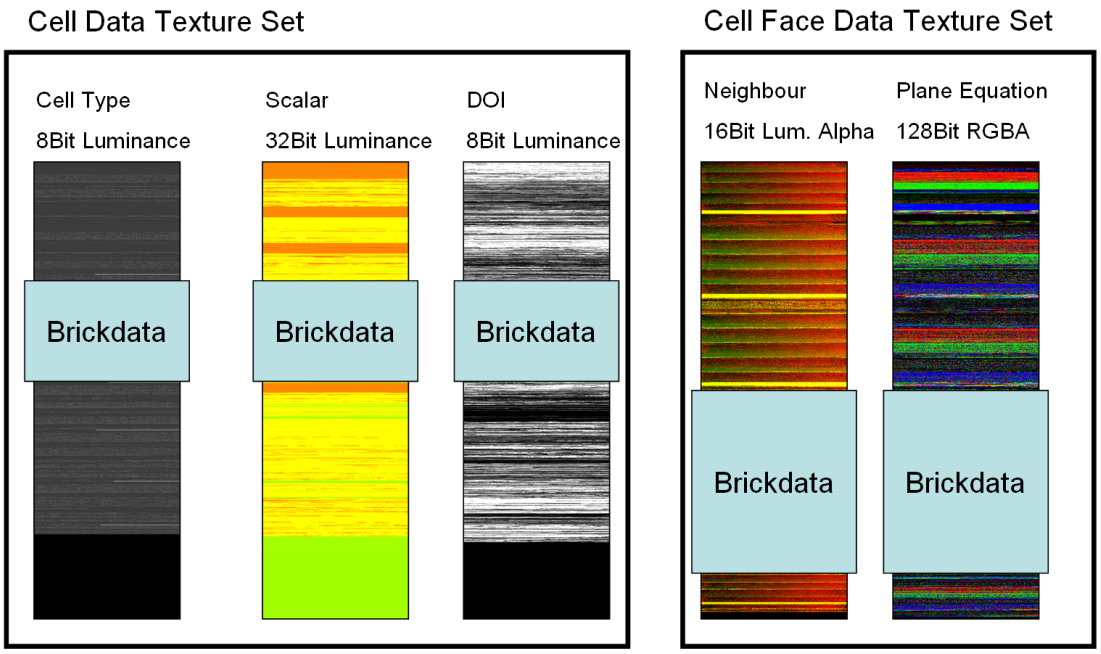


Figure 4.1: Overview over the five different types of textures, which are used to store the dataset. A cell data texture set contains only information, about a single cell, whereas cell face data texture sets store information about individual cell faces.

4.1.2 Texture Organization

The data for the raycasting algorithm has to be stored in textures in order to make it accessible for the GPU. In the following paragraphs the organization into different textures for the structured and unstructured case is presented.

The structured raycasting algorithm only requires the volume itself since the grid is implicitly specified, and thus the textures which are necessary only contain this information. The algorithm uses two three-dimensional textures. One contains the scalar field values as floating point numbers and one stores the degree of interest which has been specified in the SimVis framework and can be represented by one fixed point value. Both textures are of equal size.

In the unstructured case, data representing the grid structure has to be stored as well. Basically two different types of data must be considered: Data which is specified per cell and data which has to be present for each cell face. Per cell the cell type, the scalar field value and the degree of interest function have to be stored. The cell type and the degree of interest value can be stored as one byte fixed point numbers while floating point precision is used for the scalar field data. The data which has to be stored per cell face is the plane equation and the index of the neighboring cell. The four scalars of the plane equation are stored in floating point precision. It is important to decide how many bytes should be used to store the neighbor index. Using enough

precision to store arbitrarily large bricks would mean using at least four bytes which would translate into a maximum of about four billion cells. Datasets of this overall size are still very uncommon and because multiple bricks can be used bricks with this size are not needed. Using three bytes is not feasible since the resulting maximum of approximately 16 million cells is also still rather high and modern graphics hardware does not natively support three byte per texel textures. Thus the best choice is using two bytes for storing cell indices within bricks. This limits the maximum number of cells within a structured volume brick to approximately 64K. Figure 4.1 gives an overview over the different textures necessary to store the grid and volume information of one brick.

Besides the data precision for each property it is important to consider the dimensions of the textures which should be used and the way they should be addressed. Since a two byte addressing scheme is used it is straight forward to use two-dimensional textures. The first byte represents the x coordinate and the second the y coordinate within the texture. In order to reduce the amount of wasted texture memory it is necessary to fill up the data textures as well as possible. Thus the dimensions 256 times 4096 were chosen for all the textures. The width must be 256 because of the addressing and the height must be as big as possible to reduce memory overhead. Now every brick can reserve as many rows within those data textures as are needed. Per brick at most 255 entries within the last row are wasted. In order to address the cell data for each brick the index of its first row within a texture has to be stored. The number of rows which are needed by a brick depends on the type of data. For data which is stored per cell $\lfloor n/256 \rfloor$ rows are needed, with n being the number of cells within the brick.

When data per cell face has to be considered it is important to note that not all cells within a brick necessarily have the same number of faces. There are basically two ways how this problem can be handled. The first is to try to store the per face-data within the textures packed as tightly as possible. This means that using the two byte cell indices is not possible for this data since the space one cell needs within the texture is not constant. Thus an additional mapping texture translating the two byte index into a three byte index (two bytes will not be enough to address the data since at least four times the amount of per-cell data must be addressed) related to the data location within the cell face data textures would be necessary. This is infeasible since it would introduce additional texture lookups within the raycasting fragment program and the new index mapping texture would need additional graphics memory.

The approach which is chosen in this thesis sacrifices some texture memory within the cell face data textures in order to retain the simple cell addressing scheme. This is achieved by using the maximum number of cell faces within a brick per cell. Thus the data can be addressed by multiplying the data row of the cell index by this maximal cell face count and adding the face index.

4.1.3 Bricking for Unstructured Grids

Many volume rendering approaches which have to deal with large amounts of data decompose the volume to be visualized into smaller bricks. This makes it possible to

for example skip empty or not presently visible portions of the data. In this subsection this spatial subdivision will be introduced for structured grids. The unstructured case is more difficult since the mesh information has to be specified explicitly (in the structured case the mesh is implicitly available if the size of the cells is known). The mesh data which should be stored within a brick has to be self contained, which means that no knowledge of the overall mesh should be necessary when interpreting the data. Thus the sub-mesh generation algorithm has to create local cell indices and a local connectivity graph. Furthermore cells at the boundary of a brick have to be clipped against the brick boundary to guarantee that the individual bricks do not overlap when being rendered.

Sub-Mesh Generation

The sub-mesh generation process receives the indices of the cells which should be included as input. How these indices are selected is discussed in section 4.2. Furthermore a temporary array which can store one integer per cell is needed. The output of the algorithm is the data structure which is used by the raycasting algorithm to perform the rendering.

The first step is to create a mapping between the cell indices of the original mesh and the indices within the sub-mesh, which will be called local indices from now on. The local indices are determined by the order in which the global indices are passed to the generation algorithm. In order to create a mapping $m(i) : I_{global} \rightarrow I_{local}$ from global to local indices the temporary array is used. The input list is traversed and at the global index within the temporal array the local index, which corresponds to the present index within the input list, is inserted. Thus the temporary array contains the local index of a cell at its global index. The array is initialized with values which signal, that no local index exists and thus cells which are not part of the sub-mesh can be identified. The mapping in the opposite direction m^{-1} is already possible by using the input index list which contains a global cell index at the corresponding local index position.

The global neighborhood function $n(i, f) : I_{global} \times I_{face} \rightarrow I_{global}$ returns a cell index which is the neighbor to the input index at the input face. The local neighborhood function $\overline{n}(i, f) : I_{local} \times I_{face} \rightarrow I_{local}$ can be computed by simply applying the mapping function.

$$\overline{n}(i, f) = m(n(i, f)) \tag{4.1}$$

It has to be noted that $m()$ returns an invalid value if the neighboring cell is not part of the sub-mesh. For every brick $\overline{n}$ has to be stored in the form of a lookup table. The exact storage scheme was already discussed previously. Now that the connectivity graph has been translated for the local sub-mesh the mapping function m is not needed anymore and its lookup table, which is the temporary array which was passed to the algorithm, can be reset. It is important to note that only the parts of the array, which were used, have to be reset. Otherwise the runtime of the algorithm would not be bounded by the number of cells in the sub-mesh but instead by the number of overall cells. The function $m^{-1}()$ is sufficient when celldata has to be retrieved for the local cells which is necessary when the underlying scalar data is changed.

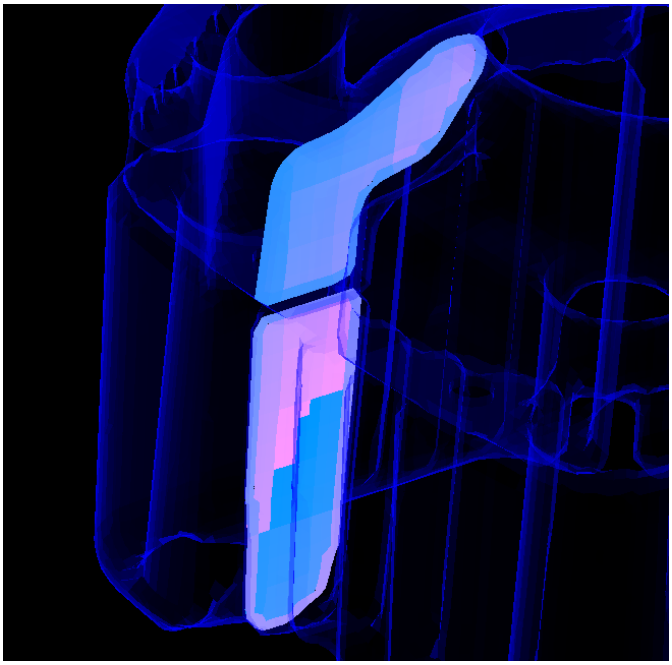


Figure 4.2: Cell intersections rendered with the GPU-based approach for one brick.

Boundary Cell Clipping

Since the data volume is divided into bricks which are bounded by six planes it is necessary to clip cells which are intersected by those planes. The raycasting algorithms require only the intersection polygon of the cells with the bounding plane to be rasterized. The two algorithms presented in the subsequent paragraphs compute this rasterization.

Basically two different methods are used to calculate the intersection of a volume cell with a plane. The first one works on the CPU and generates an explicit representation of the intersection polygon which can then be rasterized by the graphics hardware. The method works iteratively on the edges of the volume cells. First one edge which is intersected by the plane is selected and the intersection point is added to the output list of polygon vertices. To find the next polygon point one face which has not been visited (at the beginning no face is flagged as visited) and is adjacent to the present edge has to be considered. All edges of this face, except the present one, are intersected with the plane and the intersection point is added to the result vertex list. Since all cells have to be convex there has to be exactly one edge which intersects the plane. The present face is flagged as visited and the new edge replaces the previous one as active edge and a new iteration is performed. When no new face is visited anymore the result vertex list contains the vertices of the intersection polygon.

The second algorithm which is used to create an intersection polygon is GPU-based and does not generate an explicit representation of this polygon. This is an advantage since storing the polygon vertices requires additional space. The main drawback of the

GPU-based algorithm is, that the texture data for the present brick has to be loaded into video memory. In order to create the intersection a quad which covers the whole region of the cell plane intersection and whose color represents the present cell index, is rendered. Furthermore a texture coordinate is used to pass the position in object-space to the fragment stage. The size of the quad can be easily estimated by considering the boundary sphere of the cell. During rasterization a fragment program is applied which uses the color of the quad to look up the boundary face plane equations of the cell. The object-space position p which is read from the texture coordinate is tested against all face planes.

$$f_i : n_i \cdot \mathbf{x} - d_i = 0 \tag{4.2}$$

Here the n_i are the normalized plane normals and the d_i the distances to the coordinate origin in object space. Simply substituting x with p results in the signed distance of the point to the plane. To test wether p lies within the cell all distances must have the same sign. Thus all fragments which do not belong to a cell on the rendered quad can be discarded. Figure 4.2 shows the intersections of cells with the brick boundary.

4.2 Hierarchy Generation

Previous sections dealt with the bricks themselves ignoring the techniques necessary to create a proper decomposition of the data volume. This section presents a fast and robust algorithm which can be used to decompose the volume grid into bricks whose size must be limited in the number of cells contained. The decomposition process has to be as fast as possible since the hierarchy data is discarded when the visualization application is closed. Thus an efficient generation algorithm is necessary. Besides these memory and time constrains the data bricks have to store all boundary cells which are intersected by its bounding plane. Thus those cells are replicated between neighboring bricks. This has several implications on the size of the overall data structure. Basically it can be assumed that a brick which contains n cells has $O(\sqrt[3]{n})$ cells along one of its bounding face edges. Thus there are $O(\sqrt[3]{n^2})$ faces at the bounding area of the brick. This means, that with the exception of the cells at the boundary of the dataset, those cells have to be stored twice within the bricks of a decomposition. If the overall number of cells in the dataset is m then a brick decomposition without replicating cells would introduce $O(\frac{m}{n})$ bricks and a hierarchy depth of $O(\log(m))$. Since $O(\sqrt[3]{n^2})$ cells are replicated the number of bricks is:

$$O\left(\frac{m}{n - \sqrt[3]{n^2}}\right) \tag{4.3}$$

This means that the number of cells which have to be stored within the data hierarchy is.

$$O\left(\frac{n \cdot m}{n - \sqrt[3]{n^2}}\right) \tag{4.4}$$

If $n \to \infty$ then this function converges towards m minimizing the amount of replicated cells. Thus it is desirable to create as few bricks as possible when memory consumption is concerned. Thus the maximal brick size which is implied by the precision of the indexing texture is considered to be optimal and the hierarchy generation should try to match it as close as possible in order to minimize the memory overhead due to replication.

Up until now no specific spatial decomposition scheme which could be used was introduced. Many volume visualization algorithms which rely on bricking use octrees to decompose the data volume. The main benefit of this approach is, that the bricks remain cubes, when the original data volume was cubic. Furthermore there is only one type of decomposition which is applied to the internal nodes (three cuts along the xy, xz and yz planes through the center of the bounding box). Since it is desirable to minimize the number of bricks the octree is not optimal because it decomposes nodes within the tree into 8 subnodes. Performing a split into fewer subnodes means, that the brick sizes can be adapted more precisely to the desired number of cells. Thus a kd-tree is used in the approach presented in this thesis. This spatial subdivision scheme performs the split of internal nodes only along one plane through the bounding box center into two subnodes. Consecutive nodes are split alternating along the xy, xz and yz planes allowing for a better brick size distribution.

The algorithm creating the kd-tree can be split into two phases. In the first phase bricks are generated ignoring volume cells which are intersected by the splitting planes. In order to check efficiently for such intersections the cell centers and radii of their bounding spheres are computed. The tree generation works on a level by level basis, which means that with each iteration a new depth level of the kd-tree is generated by splitting too large bricks along a plane into two child bricks. If all bricks within the last level are small enough the algorithm is stopped. When the split of a brick is performed the cells which are intersected by the splitting plane are added to a list of the brick and ignored by the consecutive iterations. The number of iterations needed to perform this incomplete decomposition is bounded by the number of levels within the resulting tree. This is bounded by $O(\log(n))$ whin n being the number of cells of the data volume in a non degenerate case. If the distribution of cells is highly irregular the number of tree levels can become $O(n)$. Thus the overall runtime of this first phase is $O(n \log(n))$ in the non-degenerate and $O(n^2)$ in the degenerate case.

In the second phase the incomplete decomposition has to be completed by adding the cells, which are intersected by the splitting planes within an internal node of the kd-tree, to the bricks. This is achieved efficiently by extracting the parent nodes of each brick whose splitting plane is adjacent. Then only the cells stored in the intersection lists of those nodes have to be considered whether they are also intersected by the brick boundary. If the brick size exceeds the maximum of allowed cells in this phase it has to be decomposed again. In order to minimize the necessity to perform these additional brick splitting operations, which are more costly in this phase, the target number of cells within a brick is set lower than the absolute maximally possible cell count in the first phase. Again the cell centers and bounding sphere radii are used to check efficiently wether a cell intersects the boundary of a volume brick. The runtime of this phase is

related to the number of cells which have to be considered for each brick boundary which can be approximated by $O(\sqrt[3]{m^2})$ with m being the number of overall cells. If there are $O\left(\frac{m}{n-\sqrt[3]{n^2}}\right)$ bricks of the size n then the overall runtime of this phase is bounded by:

$$O\left(\frac{\sqrt[3]{m^5}}{n-\sqrt[3]{n^2}}\right) \tag{4.5}$$

If n is considered constant the runtime is bounded by $O(\sqrt[3]{m^5})$. This is acceptable since the hierarchy generation has to be performed only once per visualization session.

The visualization system into which the rendering frontend, which is presented in this thesis, has to be integrated can deal with time-dependent volumetric data. There are two basic types of this data, which are handled differently by the renderer. Firstly data for which the volume grid remains constant over time can be handled simply be creating the volume subdivision once and reusing the bricks for every timestep only changing the scalar field and degree of interest data. For time-dependent data where the volume grid changes over time it is necessary to create the volume decomposition for every timestep of the dataset.

4.3 Data Management

Since it is desirable to keep the memory footprint of the rendering frontend within texture and main memory as small as possible it is necessary to perform caching and swapping. It was already mentioned that resampling to a low resolution structured grid is performed to save memory. In this section it is discussed, which bricks of the unstructured grid are resampled and how the unstructured texture data is managed.

4.3.1 Texture Memory Management

Basically a brick of the datavolume is considered to be in three different states by the rendering algorithm. Either it is empty, resampled to a coarse structured grid or available as unstructured grid. If a brick is considered empty, which is the case if all the data of its subvolume is mapped to a completely transparent transmission coefficient, no texture data has to be stored for it.

If a brick is presently resampled to a structured grid two three-dimensional textures have to be allocated. Since the resolution of those textures depend on the bounding box shape and the number of cells within the brick different texture sizes are needed. Thus it is not feasible to use a large caching texture, like suggested by Hadwiger et al. [38]. The approach chosen is to simply allocate and free the 3D textures when necessary and let the graphics driver perform the memory management. Since the resampled data volumes are used as lower resolution representations the individual 3D textures are very small and thus the time necessary to allocate and deallocate them is negligible.

When the unstructured grid structure of a brick has to be stored a considerable amount of memory is necessary. Per cell one floating point value representing the scalar field and one byte for the cell type and the degree of interest function have to be stored.

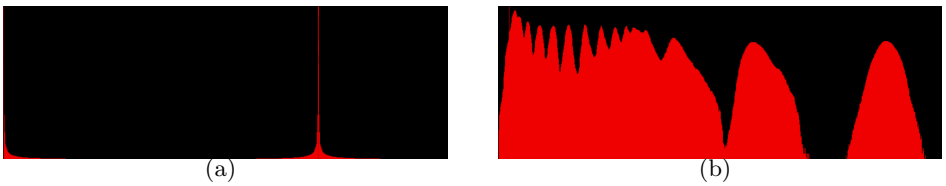
Per cell face four floating point values for the plane equation and two bytes for the neighbor index have to be used. Depending on the maximum number of faces per cell within the brick four to eight cell faces have to be stored per volume element. This means that a cell needs $4 + 1 + 1 + 4 \cdot (16 + 2) = 78$ bytes (when only tetrahedra are within the brick) to $4 + 1 + 1 + 8 \cdot (16 + 2) = 150$ bytes (when octahedra are within the brick) byte, which results in brick sizes of around 5 to 9.7 megabytes when the maximum of 64K cells are assumed per brick.

As already mentioned this data is stored within multiple textures with the dimensions 256x4096. The indexing works for the cell data by using the first byte of the index to specify the x and the second byte added to a brick specific offset as the y position within the texture. When cell face data has to be accessed the second index byte is multiplied by the maximum number of cell faces within the brick before being added to the offset of the brick. Thus the textures storing cell and cell face data can be managed on a line base assigning each brick as many lines as it needs. If a brick contains n cells and a maximum of f faces per cell it needs $\lceil \frac{n}{256} \rceil$ lines within a cell data texture and $f \cdot \lceil \frac{n}{256} \rceil$ lines within a cell face data texture. Since moving data within a texture is not as fast as copying data within main memory it is necessary to distribute the memory between the bricks of a data volume only when the underlying data changes. During normal interaction like rotating or panning the view no memory reallocation should be performed. Thus the memory assignment is only triggered when the scalar data or the degree of interest values are changed. The process is based on priorities which are assigned to every brick and will be discussed in the next subsection. The memory manager itself handles multiple cell data and cell face texture sets. The number of sets can be specified by the user to adapt the memory footprint to the available texture memory. Basically the number of cell data texture sets should be approximately one eighth of the number of cell face texture sets since they may have to store eight times the data lines. Every cell data texture set contains one cell type, one degree of interest and one scalar field value texture whereas cell face texture sets contain one plane equation and one neighbor texture. Furthermore every texture set manages a linked list of texture line allocation objects. Those objects store the number of texture lines and the line offset within the textures of the texture set.

The distribution process of the available resources is performed before the texture data itself is uploaded. Only when the assignment of all lines within the texture sets is complete the data itself is uploaded into the texture ram. The first step which is performed is the sorting of all bricks based on their priority value. Then the bricks try to allocate the data lines necessary to store their grid data based on this sequence. When a brick is not able to allocate its texture lines it is scheduled for resampling and will be rendered with the structured grid raycaster.

4.3.2 Brick Priorities

The choice whether a brick of the overall data volume should be resampled or not is based on the priorities which are assigned to each brick. Thus those priorities have to reflect the loss of information which is introduced by resampling the brickdata. Furthermore



(a) (b)

Figure 4.3: These figures show the histogram of a distribution with low (left) and high (right) entropy.

this criterion has to be evaluated very fast since the underlying degree of interest data changes often during interaction. Many simplification algorithms which were presented in chapter 2 rely on preprocessing steps which compute measures for portions of the data volume on which the decision when to simplify it can be based. In the visualization system into which the rendering frontend presented in this thesis is integrated this time consuming preprocessing cannot be used. Thus a very simple, but fast method is used.

The measure on which the priority computation for each brick is based is the entropy of the degree of interest within it. The entropy is a measure from information theory introduced by Shannon [67]. It can be used to compute the randomness of a process and thus the information which is generated by it. The definition of the entropy is based on the following formula:

$$H = -\frac{1}{\log_2(|Z|)} \cdot \sum_{j=1}^{|Z|} p_j \cdot \log_2(p_j) \tag{4.6}$$

Here Z represents the alphabet which is used by the random process and the p_j are the relative probabilities of the symbol j within a message which can be computed by dividing the number of occurrences of a symbol by the message length. Thus basically the entropy can be considered as a measure based on the normalized histogram of a message. It is 1 if the data is spread out over all histogram bins equally, which would imply a uniform distribution, and 0 if the message consists only of one repeating symbol. Figure 4.3 shows histograms of distributions with high (right) and low (left) entropy. The brick priority is computed by measuring the entropy of the degree of interest function within the brick. This can be performed very efficiently since the degree of interest is stored as one byte values per cell.

Since only the histogram of the underlying data is considered it is obvious that the spatial layout of the cells within a volume brick is not included into the measure which may result in assigning bricks with fine detail a lower priority than bricks containing only smooth degree of interest variations. To include spatial information into the priority it would be necessary to perform more complex analysis steps like performing an analysis of the data in the spatial frequency domain or directly computing an error-based on the difference between the resampled and original data. Since interactivity is highly important and the underlying degree of interest changes often the simple entropy measure is sufficient since the user is able to steer the brick priorities indirectly during

his interaction with the system.

Chapter 5

Hardware Based Hybrid Raycasting

The core of the rendering frontend which is presented in this thesis consists of two raycasting algorithms: One for structured and one for unstructured grids. Both pay special attention to the boundary of the dataset. Especially for data from the simulation field proper visualization of the mesh boundary is important and thus it is treated specially in the structured and unstructured case. It should be furthermore noted, that the structured grid raycasting algorithm is based on previous work [27, 38, 65]. The main novelty which is presented in this thesis is the unstructured raycasting technique which is able to directly cope with different types of cells. Nearly all previously published algorithms rely on tetrahedralized versions of the original mesh data whereas our approach can perform raycasting directly without the need for preprocessing. Furthermore, the combination of several state of the art techniques to perform efficient raycasting on the graphics hardware and the fusion of structured and unstructured volume visualization are notable accomplishments.

In this chapter the structured and unstructured raycasting algorithms are described. Furthermore the resampling and the handling of convex meshes is discussed. After that the different optimization techniques which are used are presented and image quality and memory consumption issues are discussed.

5.1 Structured Grids

The structured grid raycasting algorithm is comparable to the GPU-based techniques which are presented in section 2.3. It is an extension to previous work done at the VRVis Research center. The extensions are related to the handling of the degree of interest function and the special treatment of the original data volume boundary. Furthermore special care has to be taken during the raycasting process since the cells in the resampled grids are mostly not regularly sized.

In the following section a simple resampling method, which is used to create low resolution versions of the volume data of a brick, is presented. Furthermore the GPU-based

structured grid raycasting algorithm is discussed, focusing mainly on the extensions which were introduced in the scope of this thesis.

5.1.1 Resampling

Main memory and texture memory are scarce resources and thus the resampled versions of a brick are discarded when they are no longer needed. Thus the resampling has to be performed every time a brick's unstructured data does not fit into the available texture sets anymore. Furthermore the degree of interest function has to be resampled onto the structured representation every time it changes during interaction. Since the resampled versions should only approximate the original data in a lower quality (which is contrary to the resampling-based visualization approaches which were presented in section 2.5) it is feasible to use a fast, but coarse algorithm to perform it.

The method we use is an object-order approach which ignores the exact shape of the individual cells. Instead their bounding box is used to perform the resampling. Besides the two data volumes which contain the scalar field and the degree of interest values a third volume is used during resampling. The contribution of each cell, which is indirectly proportional to the distance to its center, is accumulated into this third volume. Furthermore the field values and the degree of interest values are accumulated into their respective data volumes by multiplying them by the contribution factor. When all cells have been considered, the two data volumes are normalized by dividing their values by the respective entries within the third temporary volume. So the value at a voxel of a resampled texture can be computed with the following equation:

$$V(x) = \frac{1}{\sum\limits_{x \in BB_i} \frac{1}{|x - CC_i|}} \cdot \sum_{x \in BB_i} \frac{1}{|x - CC_i|} \cdot S(i) \tag{5.1}$$

Here x is a position within the data volume, BB_i is the boundingbox of the cell i and CC_i the center and $S(i)$ a scalar value of the cell i. The term

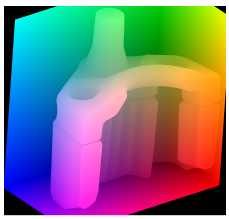
$$\frac{1}{\sum\limits_{x \in BB_i} \frac{1}{|x - CC_i|}} \tag{5.2}$$

is stored in the third temporary data volume.

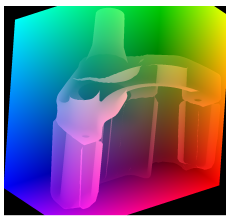
Basically the resampling process can be compared to splatting into a 3D volume using a conic filter kernel. This applies a lowpass filter on the underlying data and thus completely destroys the data volume boundary information. Thus the important visualization of the mesh surface has to be based on additional data.

5.1.2 Raycasting

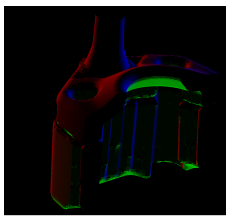
Since the resampled data is only used as a low resolution representation of the brick data the structured grid raycasting implementation has to be as fast as possible. Furthermore the image quality requirements are not high. Thus a simple and robust GPU-based approach is used. In order to perform the raycasting the starting and endpoints for



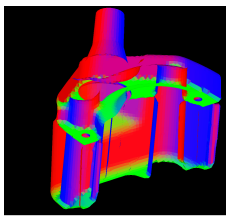
(a) start positions



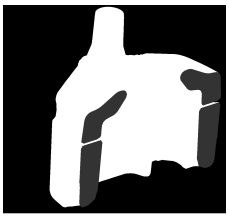
(b) end positions



(c) start position normals



(d) end position normals



(e) alpha values in the start positions texture

Figure 5.1: These figures show the different textures which are used as input to the structured grid raycasting implementation.

every view ray have to be computed. As described in Krüger and Westermann [45] the boundary of the data volume is rasterized. A fragment program is used which outputs the position of the present fragment assuming that the volume fills the unit cube. Since the underlying grid is unstructured the boundary of the original volume has to be used. There are two basic types of boundary which have to be considered. The first is the original boundary of the unstructured mesh and the second is the surface of the brick which intersects the mesh. The original mesh surface can be accessed through the rendering framework and is stored per brick. The intersection between the brick boundary and the mesh is computed by the CPU-based version of the algorithm presented in subsection 4.1.3. The GPU-based intersection algorithm cannot be used for the resampled bricks since no unstructured mesh data is available in texture memory.

Thus the raystartpositions are computed by rasterizing the frontfacing polygons of the surface. The backfacing faces are used to create the rayendpositions. Results are then stored in two floatingpoint textures which are provided as input data to the raycasting fragmentprogram. Figure 5.1(a) and figure 5.1(b) depict those start and end position textures. It should be noted that this works only if the data volume within a brick is convex. As soon as concavities exist special care has to be taken. How to properly handle this will be discussed in a subsequent section.

In order to gain additional performance the exact algorithm used to create the start and endpositions consists of the following steps:

- Render the back faces of the brick boundary into the start position texture
- Render the front faces of the original mesh boundary into the start position texture

- Render the mesh-brick boundary intersection into the start position texture
- Render the back faces of the brick boundary into the end position texture
- Render the back faces of the original mesh boundary into the end position texture

According to this list only two steps are used to create the end positions. The expensive rendering of the mesh-brick boundary intersections can be omitted. This is possible since it coincides with the back faces of the brick itself. In the start position computation phase the mesh-brick boundary has to be rasterized to correctly discard empty space within a brick. Furthermore it should be noted that the volume boundary of the original mesh has to be clipped against the brick bounding box. This can be easily done in the fragment program which outputs the fragment position.

Besides the start and end position of a ray the raycasting fragment program needs the size of the brick in object-space and the two volume textures which contain the degree of interest values and the scalar field values. The sampling of those volumes is started at the position stored in the ray start positions texture p_s. The direction of the view ray $\mathbf{v}$ is computed by subtracting the ray end position, which is stored in the ray end position texture, p_e from p_s. If $|\mathbf{v}| = 0$ then the ray passes only empty space and can be discarded. In order to properly compute the α value of a sample along the view ray it is necessary to determine the length of the ray segment. Since the cells of the resampled bricks are not regularly sized it is necessary to re-scale $\mathbf{v}$ with a nonuniform scaling-matrix before computing its length in object space. This is simply a diagonal matrix containing the object-space size of the brick. Now the two volumes can be sampled along the view ray in a front to back order. When the remaining transparency reaches a user-definable threshold before reaching p_e it is terminated early. The color and transparency computations are carried out as described in section 3.1.1, assuming constant scalar values for each ray segment.

In order to perform the boundary visualization, which will be discussed in more detail in a following section, the algorithm has to be able to discriminate between the original mesh boundary and the mesh-brick intersection. Thus this information is encoded into the alpha channel of the start and end position textures as shown in figure 5.1(e), where alpha is mapped to luminance. Furthermore the boundary visualization needs the surface normals of the mesh surface. They are rasterized during the start and end position generation into two separate textures which thus store the normals at the start and the end positions. This is possible since rendering into multiple textures at once is supported by modern graphics hardware. Figure 5.1(c) and figure 5.1(d) show both normal textures for one brick.

5.2 Unstructured Grids

The unstructured grid raycasting approach presented in this thesis is comparable to Garrity's first implementation [35]. The sampling along a ray is performed by traversing the graph which is defined by the cell neighborhood. As Bunyk et al. [11] proposed, the

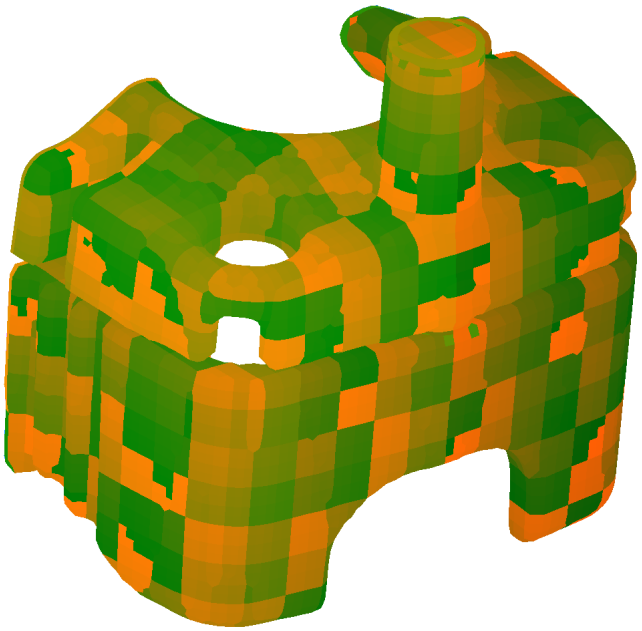


Figure 5.2: This figure shows the start cell ID texture which is used by the unstructured raycaster to determine the entry cell.

initial entry cell is determined by rasterizing the boundary of the mesh and using color to encode the cell ID. The handling of concavities will be discussed in the following section. Contrary to other methods which are discussed in section 2.4.2, the approach presented in this thesis is able to cope with different cell types. This is especially important since the evolution of simulation packages has lead to implementations which can perform their computations on complex volume meshes which are not necessarily composed of tetrahedra only. Performing the tetrahedralization of such meshes increases the number of cells by approximately one order of magnitude making it difficult to handle the resulting data interactively on standard PC hardware.

As in the structured case it is necessary to determine the start and end positions of the ray. Furthermore, the entry cell has to be determined in order to perform the traversal of the data volume. In order to do this the boundary of the data volume a brick contains has to be rasterized using a color representing the cell ID. This is easily possible since the local cell ID only consists of two bytes which are encoded in the red and green color channels. Figure 5.2 shows an example for a cell ID texture. Again, as in the structured case, the original mesh surface has to be discriminated from the mesh-brick-surface intersection. This is done by writing 255 into the blue color channel while rasterizing the intersection portion of the boundary (this is shown in figure 4.2). Contrary to the structured case here the GPU-based cell intersection algorithm, which is presented in section 4.1.3, is used to render the intersections of the brick boundary with the mesh. The cell data necessary to use this method is already available in texture

Celltype	Number of Faces	Number of Tetrahedra
Tetrahedron	4	1
Hexahedron	6	5
Prism	5	3
Pyramid	5	2
Octahedron	8	4

Table 5.1: This table shows the supported cell types and the number of tetrahedra, which would be created, by tetrahedralization.

memory since it is also needed by the raycaster itself. The rest of the boundary writes the index of the cell face which is rasterized into the blue color channel. Thus the start position of a view ray can be determined by intersecting it with the cell face which is identified by the cell index in the red and green color channel and by the face index in the blue color channel. The plane equation of this face can be retrieved from the respective texture. The plane normal, which is necessary for the boundary visualization, is thus available as well. In order to compute the start position for the rest of the boundary, where no cell face index is available, the view ray is intersected with the boundary of the brick. The view ray itself can be extracted from the matrices which are used to perform the object-space to clip-space transformation. A point at the near-plane and one at the far-plane is transformed by the inverse of those matrices. The resulting points define the view ray.

Now that the entry-point and entry-cell is known the raytraversal can be performed. Per cell the cell type is retrieved from the cell type texture and the corresponding number of faces is used to iterate through all cell faces in order to find the exit point through which the ray leaves. The cell face plane equation can be retrieved from the corresponding entry in the cell face texture. Presently only a fixed number of cell types is supported, which are listed in table 5.1, but extending the method to arbitrary convex polyhedra is possible since only the cell face plane equations and the indices of the corresponding neighboring cells are necessary to perform the ray traversal through the mesh.

Interpolation Considerations

As in the structured case the scalar data is considered constant along a ray segment within a cell. This is feasible since a large amount of simulation data (especially from the CFD field) is cell based. This means, that the data is computed for each cell and assumed to be constant over the whole cell volume. Still, nearly all approaches which are presented in section 2.4 assume vertex-based data. This means, that the scalar field values are specified per mesh vertex. Thus interpolation of the data within the cells has to be used, which results in smoother result images. Since basically all of the discussed techniques can only render tetrahedra the interpolation can be easily performed by computing the values at the entry and exit points of a ray through a cell. This can be done by barycentric interpolation from the vertices which make up the faces. The

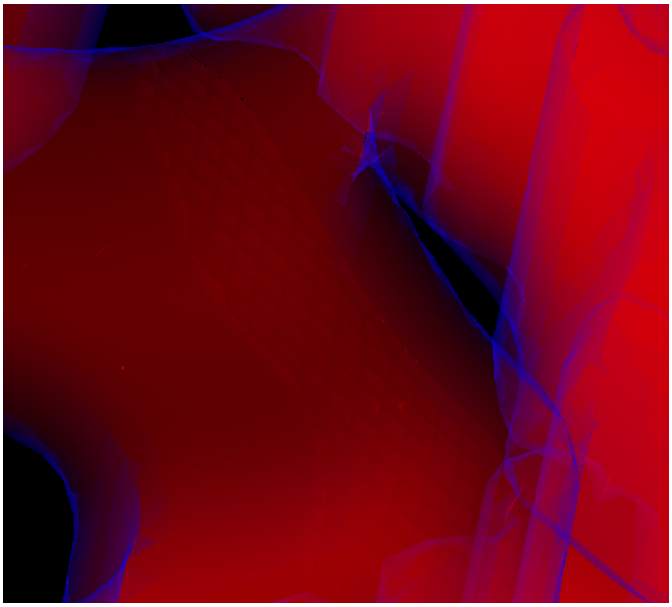


Figure 5.3: This figure shows artifacts at the brick boundary, which are introduced by not clamping the last ray segment of a view ray to the brick.

scalar field can be assumed to vary linearly within a tetrahedron since the gradient within such a cell is constant. The subsequent application of preintegration techniques, which were introduced by Röttger and Ertl [60] for unstructured and Engel et al. [28] for structured grids, to calculate the final color and opacity values is thus a very good approximation to a fully analytic solution of the optical model. As soon as more complex cell types are introduced the interpolation becomes more difficult and time consuming since the linearity of the scalar field value between entry and exit point is now only a rough approximation which becomes worse the more faces a cell has and the stronger it is distorted. Thus proper interpolation within meshes containing arbitrary cell types is left as future work.

Ray Termination

As in the structured case the accumulation of transparency and color values is performed front to back. Again the process is terminated if a user-definable threshold of the remaining transparency is reached. During ray traversal the present position along the ray is used to check whether the bounding box of the brick is reached. If this happens it is important to use only the ray segment which is within the bounding volume to compute the sample's transparency. Otherwise artifacts at the boundary between different bricks are visible, which can be seen in figure 5.3.

The boundary visualization for the last face along the ray can be performed by checking whether a face is part of the mesh surface. This information is encoded in the first three components of the face's plane equation which represent the normal vector of

the plane. All plane equations are stored in the Hessian normal form, which means that this vector has to be normalized. For all boundary faces the normal vector portion of the plane equation is scaled by 10.0. Thus such faces can be found by checking wether any of the components of the plane normal are larger than 1.0 or smaller than -1.0. Naturally the normals have to be divided by 10.0 before being used to compute the exit point of a ray within a cell. It is necessary to additionally store whether a cell face is part of the mesh surface within the plane equation since the lookup into the connectivity texture can be omitted if a boundary face is found. Furthermore additional flexibility is introduced which allows for more control over the mesh boundary visualization. For example only parts of the mesh surface are of interest in certain datasets especially from the aerodynamics field (the surface of the bounding volume is not interesting whereas the surface of a simulated plane is). Thus only scaling plane normals of interesting surfaces can remove some cluttering of the visualization, which would have been introduced by visualizing unnecessary boundary information.

5.3 Coping with Concave Data Volumes

Visualizing data which is specified on concave meshes introduces additional challenges compared to the convex case. For image order approaches this is because a ray which travels through a concave mesh may exit and reenter the data volume more than once. Thus the location of the entry cell of a ray has to be performed multiple times. Object-order methods have more difficulties sorting elements of a concave grid. Several solutions for image- and object-order approaches are discussed in section 2.4. The method which is presented in this thesis is based on Weiler et al.'s [81] suggestion to use a technique similar to depth peeling [30] to cope with concave meshes.

Normally depth peeling is used to correctly rasterize transparent concave objects without the need to perform depth sorting. This is achieved by rendering the object multiple times. After every pass the resulting depth image, which is created by the z-buffer method available in all modern graphics hardware, is stored and provided as input to the next rendering pass. Besides the normal depth test which is used to correctly determine the visibility of a fragment a second depth test is used against this depth image. If front to back depth peeling should be used this second depth test succeeds only if the present fragment is behind the depth which is stored in the additional depth image. This means, that with each successive pass one layer of the object that is rendered is peeled away and the surface which lies behind it becomes visible. After each step the output images have to be composited to form the final picture. The algorithm presented in this thesis does this by using two textures which are alternately used as rendering targets. During each pass the result texture from the previous pass is used as input to the boundary rendering fragment program which performs proper compositing between the new fragments and the previous ones by performing front to back compositing. Thus an object which consists of n overlapping surfaces has to be rendered n times with this approach.

Applying Depth Peeling to Volume Bricks

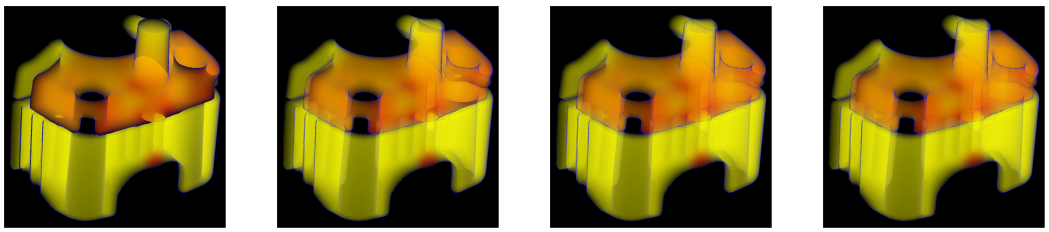
A brick can be rendered in three different ways. The method chosen depends on the data it contains and on its relevance to the user. If a brick is empty, which means that all degree of interest values within it are zero, only its boundary has to be rendered. Thus the original depth peeling method can be applied. This means, that the boundary can be rendered with arbitrary transparency without the need to sort its rendering primitives.

If a brick is presently resampled the depth peeling is used to correctly peel away the layers of the brick volume. This means that the ray start position and ray end position textures are rendered using this technique. The depth information which was generated by the previous end position rasterization is used to peel away the already rendered portion of the volume in the current pass. Thus the fragment program which is enabled during this process checks whether the new fragment is farther away from the viewer than the one created in the previous pass. It is important to note that the new start positions have to be compared with the previous end positions using the greater or equal operator since the start positions may be incident with the previous end position. This is the case if boundaries of fluid and structural simulations can be found within the dataset. The rasterization of the new end positions has to use the greater operator to perform the second depth test (if greater or equal would be used the last surface would not be peeled away). Rendering a concave volume needs half as many passes as rendering the corresponding surface since not only the volume boundary but whole layers of the volume are rendered at once. It should be noted, that only for the first layer the cell-brick-boundary intersections have to be rendered. Subsequent layers require only the rasterization of the original mesh surface. This is possible since the brick boundary is guaranteed to be convex.

In the unstructured case only the start cell and face indices have to be rasterized (the end positions are not necessary since the raycaster itself is able to determine them at the end of the traversal of a layer). Again the depth peeling method is used to render the starting information layer by layer. As in the structured case the cell-brick-boundary volume intersections have to be rendered only for the first layer.

Determining the Layer Count

Computing the number of layers which have to be rendered would be an expensive task, since it is view direction-dependent, which means that it has to be recalculated every time the user rotates the view. Furthermore a topological analysis of the mesh boundary would be necessary. In order to determine the number of layers efficiently without explicitly computing the number of necessary layers, functionality of modern graphics hardware is used. It is possible to query the number of rasterized fragments which are actually written into the frame buffer. In the OpenGL API this functionality is available through the occlusion query extension [6]. Thus it is possible to start such a query before rendering a layer and retrieve the results afterwards. If no fragment is rasterized anymore all layers have been processed and the depth peeling can be stopped. Figure 5.4 shows consecutive rendering of layers and the pixels which are updated. In this case the sixth depth peel does not add additional information to the image, which



(a) peel 1 (b) peel 2 (c) peel 3 (d) peel 6

(e) difference 1-2 (f) difference 2-3 (g) difference 3-6

Figure 5.4: Depth peeling applied to a brick of the small cooling jacket dataset. The second row of images shows the difference between the results after 1, 2, 3 and 6 peels.

terminates the rendering.

5.4 Implementation Details

The raycasting method proposed in this thesis is implemented as a rendering plugin of the SimVis system. It is implemented in C++ using Qt and OpenGl. Currently all the raycasting fragment programs are implemented in Nvidia's Cg.

The implementation of the hybrid raycasting algorithm is an extensible framework which allows for an easy implementation of additional visualization methods. Different Cg interfaces are provided which represent parts of the raycasting process and can be accessed by the user. One interface represents an optical model whereas the second important one represents the boundary visualization. Furthermore the raycasting framework has to combine the output from three different rendering approaches, which are the boundary visualization of empty bricks, the visualization of resampled structured data and the rendering of unstructured data. All three approaches have to produce comparable results. This is especially important since all three methods have to visualize some common data. For example the boundary has to be rendered consistently in all of them. Also the evaluation of the optical model for structured and unstructured bricks should result in comparable images. Thus all three fragment programs use the same interfaces to create their results.

Fusing Structured and Unstructured Bricks

In order to produce coherent visualizations it is necessary to fuse the result images from different bricks together. This is accomplished by utilizing two textures into which the rendering is performed alternately. The texture into which the previous brick has been rendered is used as input to the current bricks visualization. Every brick has to correctly blend this input with its output in front to back order. The order in which the bricks have to be rendered can be directly retrieved from the kd-tree. First the three planes along which the nodes of the tree are split have to be classified with respect to the viewing direction. The tree can then be traversed based on this classification by visiting the children first which are on the side of the viewer. This traversal produces a correct visibility order. After all the bricks have been rendered a final compositing pass is used to add the user specifiable background luminance to the result image.

5.5 Performance Characteristics

There are several characteristics by which the performance of a rendering system can be judged. The three most important ones are the amount of memory used, the time necessary to render the visualization and the image quality which can be achieved. Different optimization techniques are presented and the tradeoff between rendering times and image quality are discussed in this section.

5.5.1 Rendering Performance

The time needed to create a proper visualization is crucial when the user has to interact with the visualization system. There are different ways how the user can interact with the SimVis system which require different update operations within the rendering frontend. The most basic interaction is changing the viewing parameters of the visualization by rotating the dataset, zooming or panning. These actions must be possible at interactive frame rates to allow the user to explore the dataset. The next frequent way to interact with the SimVis system is the modification of the DOI. Since the update of the DOI itself takes some time the update of the visualization is not necessarily performed at interactive frame rates. The reassignment of a scalar field value to the visualization and other modifications of the rendering mode happen infrequently compared to the previous two interaction methods. Thus interactivity is basically only mandatory when modifying the view settings. In order to achieve this two very basic approaches are used. The first is to render the output image in a lower resolution and stretching it to cover the whole viewport. Since the raycasting approach is an image-order approach it scales very well with the size of the output. The user can adjust the resolution of the output during interaction and thus is able to adjust the visualization speed to his needs.

In order to decouple user event handling from the rendering process, it is performed in a separate thread. If updates because of DOI modifications or viewport changes are triggered and the rendering is already being performed it has to be finished as fast as possible. This is achieved by only rendering one depth peel/layer for every brick.

Dataset	Cells	In Hierarchy	Interaction	Final Image
cabin	765053	849451	185ms	909ms
cooling jacket	1537898	1631459	166ms	500ms
small cooling jacket	76816	77651	34ms	200ms
DPF	262674	282022	76ms	270ms
generator	6729806	7952899	497ms	1368ms
head	679253	858429	208ms	588ms
isabel	3125000	3586947	238ms	909ms
two-stroke	149864	165956	71ms	250ms

Table 5.2: A comparison of the number of cells in a dataset and the number of cells, which have to be stored in the brick hierarchy. The performance measures are taken during interaction and for the final image with the dataset at a viewport resolution of 512^2 using the standard emission and absorption optical volume model.

This means, that if an update event is received during the rendering process of a brick all subsequent bricks perform no depth peeling. This enables the user to still get an overview over the data during interaction. If no update events intercept the rendering process it is finished as usual.

Besides reducing the output image resolution and only rendering one layer of every brick the different rendering modes of the bricks are used to gain additional performance. The rendering time of an unstructured brick is approximately five times larger than the time necessary to render a structured brick, which itself is three times larger than the rendering time of only the mesh boundary. Furthermore the bricking approach lends itself to performing view fustrum culling. The bounding box of a brick is transformed into views pace and the bounding rectangle of the projected box is checked against the view fustrum. If the rectangle is completely outside of the view fustrum the brick is not rendered at all and.

5.5.2 Memory Requirements

One of the main limitations of current GPU-based volume rendering techniques is the availability of texture memory. Since the rendering frontend which is presented in this thesis is part of a larger visualization system this limitation is even more severe since other parts of the system require some texture memory as well.

Thus it is necessary to limit the memory which is used by the rendering frontend. This is possible by limiting the number of texture sets which are available to the bricks of the dataset. The size of a cell face texture set is $4096 \cdot 256 \cdot (4 \cdot 4 + 2 \cdot 1) = 18,874,368$ byte and a cell data texture set has the size $4096 \cdot 256 \cdot (1+1+4) = 6,291,456$ byte. Since a brick can contain a maximum of 256^2 cells it needs a maximum of 256 lines within a cell data texture set. Thus such a set can store the contents of at least 16 bricks. The number of lines a brick needs within a cell face texture set depends on the cell types within the bricks. The two most common grid types are either purely tetrahedral grids, or grids containing mostly hexahedra (see chapter 6). If only tetrahedral cells are

considered a cell face texture set can store at least four bricks. When hexahedral cells have to be stored the contents of at least two bricks can be stored. Since the brick sizes are mostly much less than 256^2 the number of bricks per texture set is mostly higher by a factor of two. Far more cell face than cell data texture sets are needed. Thus the easiest way to limit the number of overall texture memory used up for the unstructured data is to limit the number of cell face texture sets.

The amount of memory necessary to store resampled structured bricks is only five byte per voxel (one byte for the DOI and four for the scalar value). The number of voxels used for a brick is directly related to the number of cells within the brick. The resolution of the volume texture is computed by assuming a uniform cell distribution. The resolution along the longest side of the bounding volume of a brick has thus the resolution $\sqrt[3]{n}$ extended to the next power of two. In order to keep the voxel dimension as equal as possible along all axes the reciprocal of the voxel size along the largest bounding box axis is used to compute the resolution along the remaining axes, which are then again rounded up to the next power of two. Thus the maximal number of voxels is $64^3 = 262,144$ resulting in a size of $1,310,720$ byte. Again most bricks are far smaller than this, requiring mostly only half the amount of memory.

In order to determine the amount of texture memory, the rendering frontend requires, besides the data textures, the textures which are used as render targets have to be considered. Assuming a 512^2 viewport the amount of necessary memory is $512^2 \cdot 4 \cdot 2 = 1,048,576$ byte for the two compositing textures, $512^2 \cdot 4 = 524,288$ byte for the cell index texture used in the unstructured grid raycaster, $512^2 \cdot 4 \cdot = 1,048,576$ byte for the two depth textures used for the depth peeling and $512^2 \cdot 4 \cdot 4 \cdot 4 = 16,777,216$ byte for the four floating point textures which are used by the structured grid raycaster. Overall 19,398,656 byte are used up by the render target textures.

In order to allow the graphics hardware to keep all necessary data in its memory it is necessary to compute the approximate overall data which is used up by the different kinds of texture data. The number of texture sets can be limited, as well as the size of the viewport. Only the amount of texture memory necessary to store the resampled structured data is dependent on the dataset size. Thus the memory requirements of the other kinds of data have to be adapted based on the number of cells a dataset contains. Basically it must be possible to display all bricks of a dataset at once, which means that at least the resampled versions have to fit into texture memory. Thus when larger datasets have to be visualized fewer texture sets can be used for the unstructured raycasting. If all bricks contain the maximum number of cells the amount of memory necessary to store all bricks is $\frac{n}{256^2} \cdot 64^3 \cdot 5$ byte, which has to be reserved on the graphics card. Thus the number of available texture sets can be computed by solving the following equation for n_{cfd}:

$$m = \underbrace{r_x \cdot r_y \cdot 84}_{Viewportdependent} + \underbrace{\frac{n}{256^2} \cdot 64^3 \cdot 5}_{Structured} + \underbrace{n_{cfd} \cdot 18,874,368 + n_{cd} \cdot 6,291,456}_{Unstructured} \tag{5.3}$$

Here m is the amount of memory which should be used, n the number of cells in the

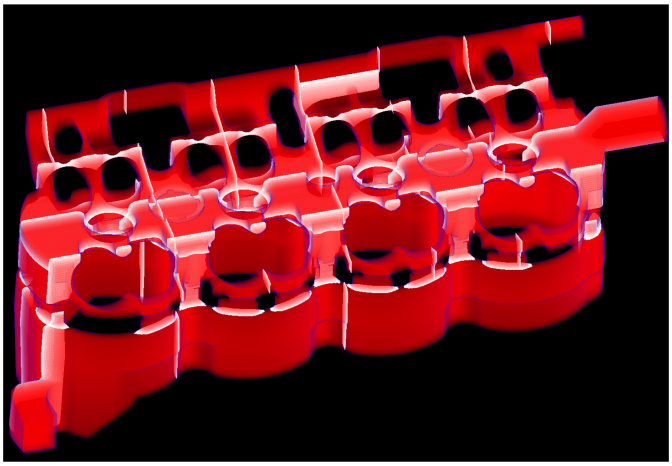


Figure 5.5: Brick decomposition of the Cooling Jacket dataset.

dataset, r_x and r_y the viewport resolution, n_{cfd} the number of cell face texture sets and n_{cd} the number of cell data texture sets. In order to solve this equation for a fixed viewport size for n_{cfd} an additional equation relating n_{cfd} to n_{cd} is necessary. As already mentioned the number of bricks a cell data textureset can store is 16. Cell face data texture sets can store $\frac{4096}{256 \cdot 6} \approx 2.6$ bricks if hexahedral cells are part of the volume grid. Thus the additional relation is

$$n_{cd} = \frac{2.6 \cdot n_{cfd}}{16} \tag{5.4}$$

Substituting this for n_{cd} and solving for n_{cfd} results in

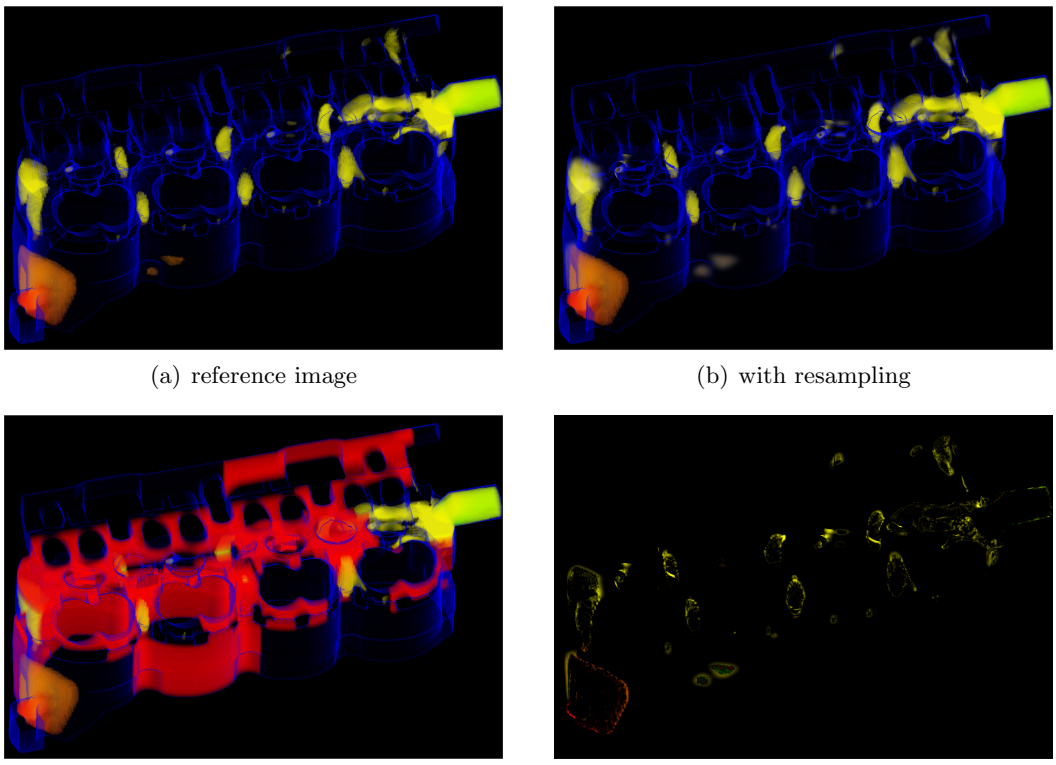
$$n_{cfd} = \frac{m - 84 \cdot r_x \cdot r_y - 20 \cdot n}{19,896,729,6} \tag{5.5}$$

If for example a dataset consisting of 7 million cells should be rendered onto a viewport with a resolution of 512 times 512 and only 400 megabyte of texture memory may be used no more than 12 cell face texture sets may be generated.

If the memory consumption of the presented algorithm is discussed it is also necessary to mention that some memory is wasted because cells intersecting the brick boundaries have to be stored multiple times. In section 4.2 an upper bound for the number of cells which are really stored within the brick hierarchy has already been presented. Table 5.2 shows a comparison between the number of cells and the number of cells which have to be stored in the data hierarchy for some datasets which have been used to test the rendering frontend.

5.5.3 Image Quality

The rendering algorithm which is presented in this thesis is able to render volume data with different quality levels. In this section the different methods to render the data



(a) reference image (b) with resampling

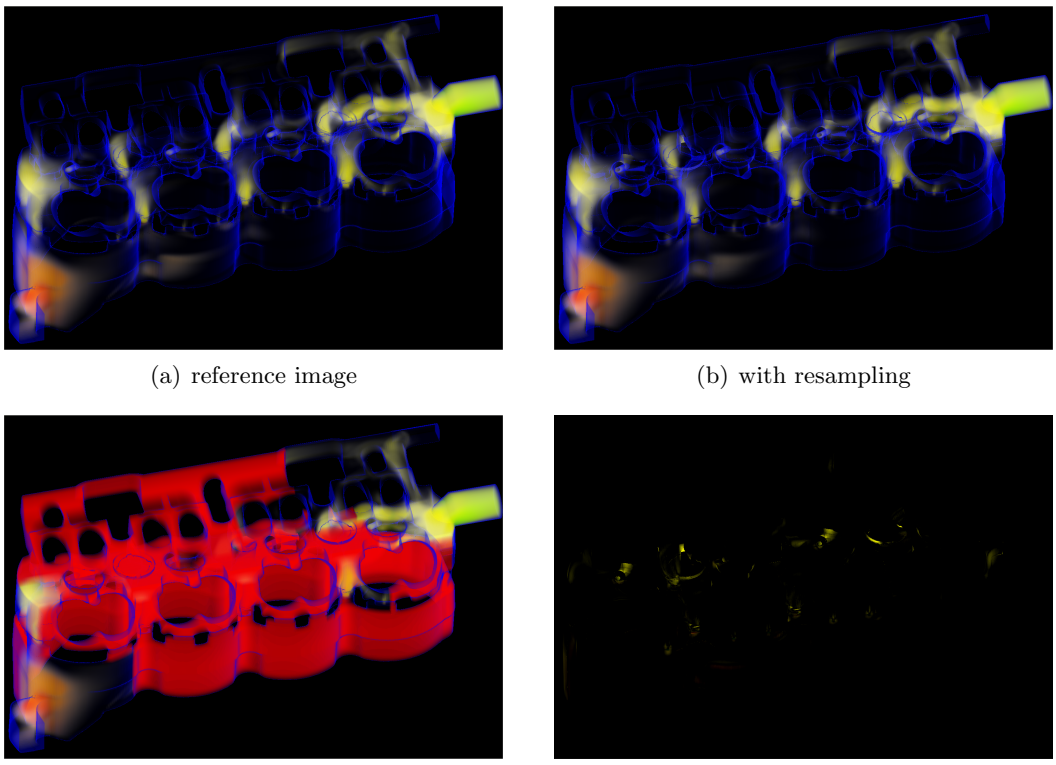
(c) resampled bricks (red) (d) difference image

Figure 5.6: The four figures show a binary selection of high turbulent kinetic energy within the cooling jacket dataset.

volume are compared. Using different texture set precisions under different selection conditions will be discussed. The results are compared to images which have been rendered with the highest possible accuracy.

The first comparison between resampled and original unstructured data can be seen in figure 5.6. Here a binary selection of high turbulent kinetic energy values within the dataset is shown. Color is mapped to pressure. The portions of the dataset, which are resampled are shown in figure 5.6(c). Figure 5.6(a) shows the reference image, which has been rendered based on the original unstructured data, without performing resampling. The brick decomposition of the dataset is shown in figure 5.5. The difference images shown in figure 5.6(d) reveals, that the original image mostly differs at the borders of the selections and at the volume boundary of the mesh.

Besides the precision issues figure 5.6 shows the lack of proper interpolation very clearly. As already discussed the data along a ray segment within a cell is considered constant. This results in the "blocky" appearance of the visualization. Because most of the datasets on which the raycasting approach was tested contain cell-centered data (data values are stored per cell and not per vertex) the lack of interpolation is not too problematic since the visualization is an exact representation of the data itself. Still



(a) reference image (b) with resampling

(c) resampled bricks (red) (d) difference image

Figure 5.7: The four figures show a smooth selection of high turbulent kinetic energy within the cooling jacket dataset.

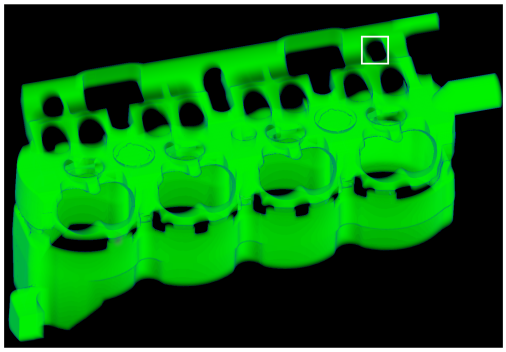
it should be noted, that using data interpolation would lead to visually more pleasing images especially if data with high spatial frequency should be displayed.

In the next comparison, which is shown in figure 5.7, the selection is smoothed out. Here figure 5.7(d) shows far less differences between the reference image and the resampled result. Basically the main differences can be found at the mesh boundary. If smooth selections have to be resampled the results are far better than in the binary case because the low pass filtering effect of the low quality resampling can cope with low frequency data better. Furthermore in the smoothed out case the lack of proper interpolation is only barely visible because of the low spatial frequency of the DOI function.

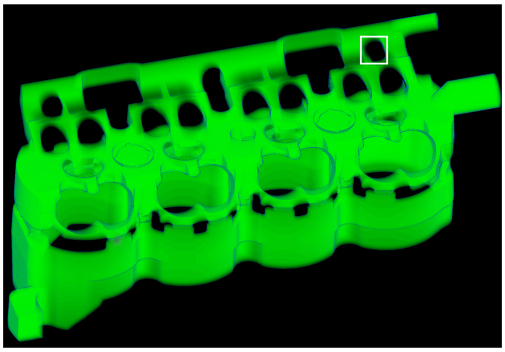
Until now always full 32 bit floating point precision has been used to generate the result images. In order to achieve higher frame rates and to save memory the rendering frontend supports using low precision 16 bit floating point numbers. This results in some precision artifacts which are the subject of the following comparison in figure 6.6. The main difference, which is responsible for the artifacts which can be observed in figure 5.8(b) and 5.8(d), is the precision of the face plane equations. If only 16 bit floating point numbers are used the ray propagation algorithm makes wrong cell ray intersection

decisions. Mostly this problem becomes apparent if the view ray is nearly parallel to faces of a cell. To illustrate this figure 5.8(e) shows the cell faces within the dataset.

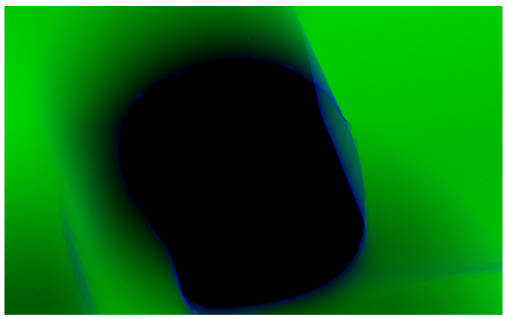
The rendering speed differs approximately by a factor of 1.5 between 16 and 32 bit floating point numbers. Thus the low precision visualization is used during the exploration and analysis of a dataset whereas result images can be produced using the high precision version.



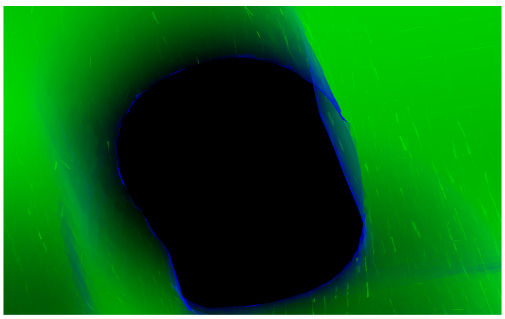
(a) overview 32 bit floats



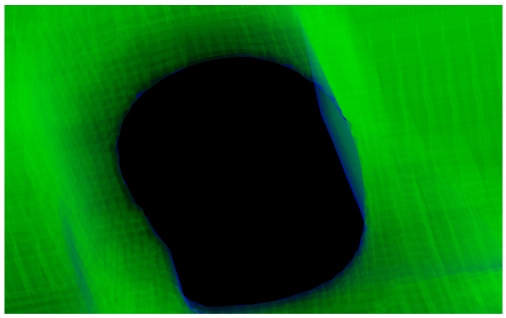
(b) overview 16 bit floats



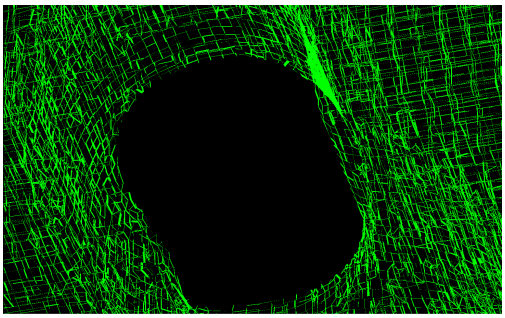
(c) detail 32 bit floats



(d) detail 16 bit floats



(e) detail cell faces



(f) differences for volume density

Figure 5.8: The first four figures show a comparison between 32 and 16 bit floating point precision. Figure 5.8(e) shows the corresponding cell structure. A contrast enhanced difference image between figure 5.8(c) and 5.8(c) is presented in figure 5.8(f). Here precision problems at the cell boundaries can be observed.

Chapter 6

Application Examples

In this chapter several application examples will be presented. They are separated into two groups. In the first the visualizations using the new rendering frontend are compared to results which can be generated by the already available visualization possibilities in the SimVis system. Here the main focus lies in emphasizing the advantages and disadvantages of the new rendering frontend and under which circumstances it can be used most efficiently. The second group of examples deals only with the new raycasting approach and its benefits when very complex and large datasets have to be visualized.

The following sections will present different datasets which have been visualized using the new hybrid raycasting approach. The point-based rendering approach, already available in the SimVis system, is introduced briefly. Furthermore its visualization results are compared to the new raycasting-based approach.

6.1 Comparison of Point-Based Rendering and Raycasting

In this section results from the point-based rendering frontend of the SimVis system, which will be introduced briefly, are compared to the new raycasting-based method.

Point-Based Renderer

The point-based renderer which is available in the SimVis system uses an object-order splatting approach to display a dataset. The centers and sizes of all cells are computed and for each cell a splat with a corresponding size is drawn. The splats are Gaussians whose width can be adapted to steer the smoothness of the result images. In order to perform proper depth compositing the splats are sorted back to front using a linear time radix sort for floating point numbers. The main field of application of this rendering frontend is providing a very fast overview over the selected data in its spatial context. Thus the quality of the resulting images is not critical.

The main drawback of this rendering approach is, that no explicit representation of the mesh boundary is displayed. Thus datasets, whose surface, and its interaction with the simulated volume, is of importance are visualized only partially. On the other hand

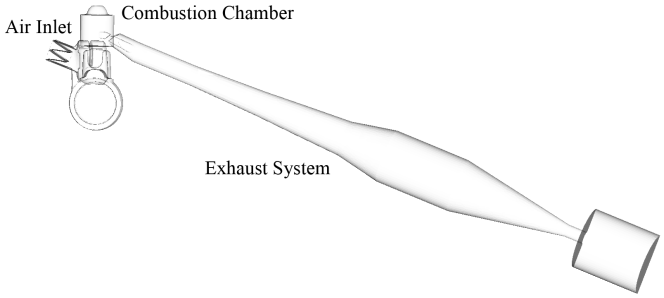


Figure 6.1: Overview over the two-stroke engine dataset. The complete engine, including the exhaust system has been modeled.

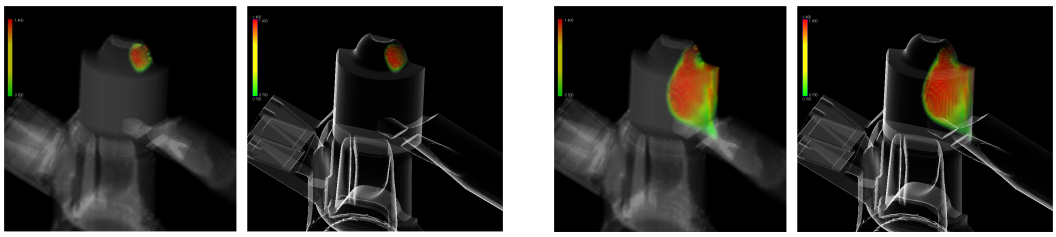
Cell Type	Number of Cells	When Converted to Tetrahedra
Tetrahedron	1156	1156
Hexahedron	129658	648290
Prism	12545	37635
Pyramid	6505	13010
Overall	149864	700091

Table 6.1: This table shows the number of cells of the two-stroke dataset. Furthermore the number of tetrahedra necessary to represent the same dataset is listed for comparison.

datasets, without important mesh boundary information, can be visualized very well with the point-based approach.

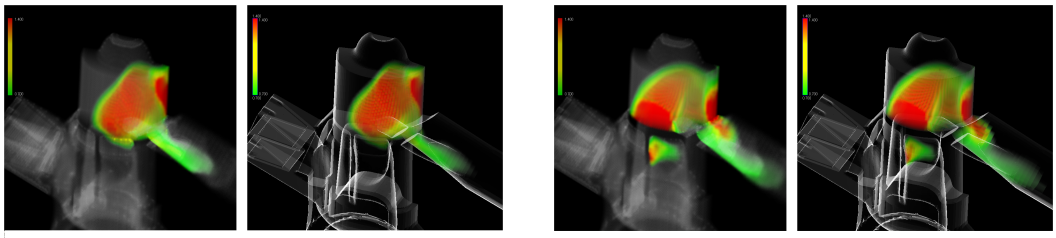
Two-Stroke Engine

The two stroke engine dataset is courtesy of the "Institut für Verbrennungskraftmaschinen und Thermodynamik" at the Technical University of Graz. In a previous work Schmidt et al. have analyzed this dataset with the SimVis approach [66] and compared the results with a VR/AR method. The dataset itself contains a complete two-stroke engine. The injection and combustion of fuel has been modeled and simulated. The volume mesh on which the dataset is based is moving, which means, that every time step a new unstructured grid is introduced. For a complete overview over the number and types of volume cells see table 6.1. Thus it is necessary to compute and store the brick decomposition of the dataset for the hybrid raycasting approach for each time step. Furthermore the selection of a new time step requires to upload not only the volume data itself but also the mesh information. Besides these drawbacks the hybrid raycasting method has the important advantage, that the boundary of the engine can be displayed. Figure 6.2 shows the fuel injection into and distribution within the combustion chamber of the engine. Equivalence ratio, which is the relation between fuel and air within a volume cell, is selected above 0.7. The best, and most combustable mixture, has an equivalence ratio between 0.7 and 1.4. Thus the colorscale is mapped to this interval.



(a) fuel injection

(b) leaking of fuel into the exhaust system



(c) further leaking

(d) connection to exhaust system is cut

Figure 6.2: The four figures show high combustion equivalence ratio within the two-stroke dataset. The leaking of fuel into the exhaust is clearly visible.

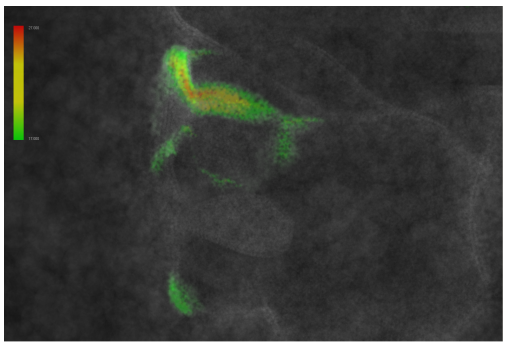
Cell Type	Number of Cells	When Converted to Tetrahedra
Tetrahedron	858429	858429
Hexahedron	0	0
Prism	0	0
Pyramid	0	0
Overall	858429	858429

Table 6.2: This table shows the number of cells of the Head dataset. Furthermore the number of tetrahedra necessary to represent the same dataset is listed for comparison.

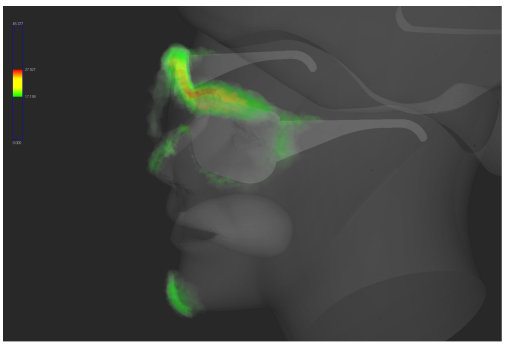
Red implies too high equivalence ratio and green too low. In figure 6.2(a) the fuel injection itself is shown. In the next image the leaking of some of the injected fuel into the exhaust system is depicted. Here the additional boundary visualization helps the user to better relate the fuel cloud to the opening of the exhaust. Figure 6.2(c) shows further leaking of fuel into the exhaust system whereas figure 6.2(d) depicts the moment where the connection between the combustion chamber and the exhaust is cut off. In all figures the standard volume density optical model is used in combination with the silhouette boundary visualization.

Head

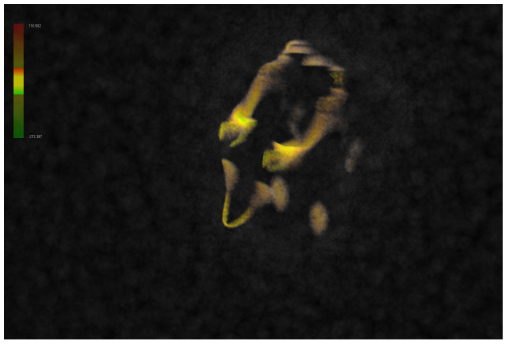
The head dataset is courtesy of Arsenal Research, Vienna, Austria and has been provided by Markus Trenker. It contains a model of a human head wearing a helmet around which the airflow has been simulated. Table 6.2 shows detailed information of the datagrid.



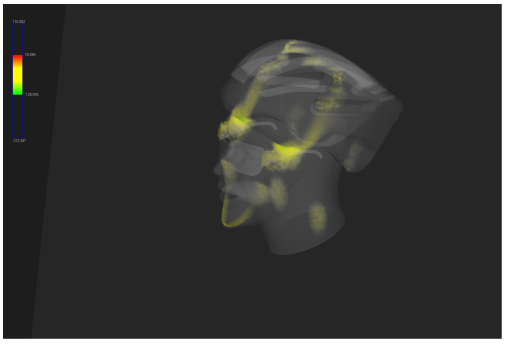
(a) high turbulent kinetic energy (Point-Based)



(b) high turbulent kinetic energy (Raycasting)



(c) high velocity (point-based)



(d) high velocity (raycasting)

Figure 6.3: The four figures show a comparison between the point-based and the hybrid raycasting visualization of two simple selections within the head dataset.

In figure 6.3(a) and 6.3(b) high turbulent kinetic energy is selected. Color is mapped to this attribute as well, which represents the amount of turbulence within a volume cell. The turbulent areas behind and around the edge of the sunglasses become clearly visible, when the hybrid raycasting approach is used. The lack of boundary visualization makes this difficult when only the point-based approach is used. Besides the turbulent areas in the dataset, which may be unpleasant to the biker, high flow velocity areas are of interest. Figures 6.3(c) and 6.3(d) show these high velocity areas within the dataset. Again the additional visualization of the mesh boundary helps to identify the exact position of the selected data portions with respect to the head, glasses and helmet.

6.2 Raycasting Results

In this section the application of the hybrid raycasting approach to a highly complex dataset is be presented. First the important characteristics of the volume mesh on which the dataset is specified is discussed. After this the results, which have been generated with the SimVis system and the new rendering frontend, are presented.

Cell Type	Number of Cells	When Converted to Tetrahedra
Tetrahedron	3728546	3728546
Hexahedron	1355000	6775000
Prism	1609776	4829328
Pyramid	36484	72968
Overall	6729806	15405842

Table 6.3: This table shows the number of cells of the generator dataset. Furthermore the number of tetrahedra necessary to represent the same dataset is listed for comparison.

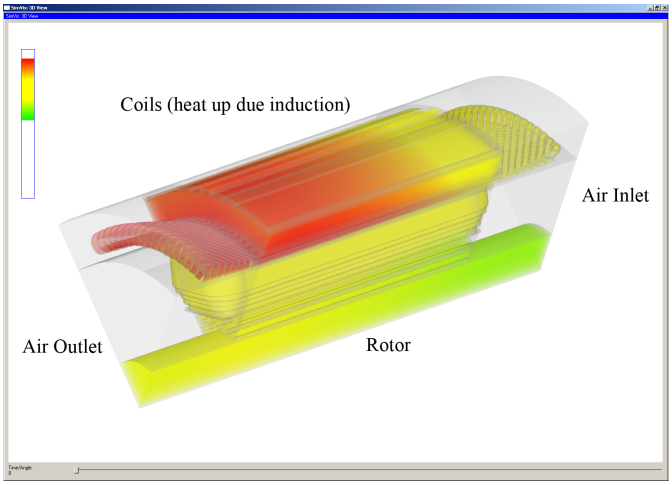


Figure 6.4: An overview over the generator dataset. Color represents temperature. Only the solid portions of the dataset are selected.

The Generator Dataset

The generator dataset is courtesy of Arsenal Research, Vienna, Austria. It contains one sixth of the geometry of a generator. Since the remaining five sixth are symmetrical this is sufficient. The data volume itself can be divided into two parts. One is the solid portion of the generator and the other the air surrounding it. Both parts are basically separate meshes which touch each other on a common surface. The problem, which has been modeled within the dataset, is the cooling of the generator. Thus heat transport and dissipation has been simulated for the solid portion of the dataset, 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 boundary between structure and flow meshes is not defined on common vertices, which results in partly overlapping grids. Besides this the mesh surface of both dataset portions has a very high depth complexity, which means, that many depth layers have to be rendered to produce the final visualization. Even when decomposed into 284 bricks every brick contains between 2 to 8 depth layers. On average 5 depth layers have to be rendered. Besides the depth complexity the sheer size of the mesh surface is notable. It

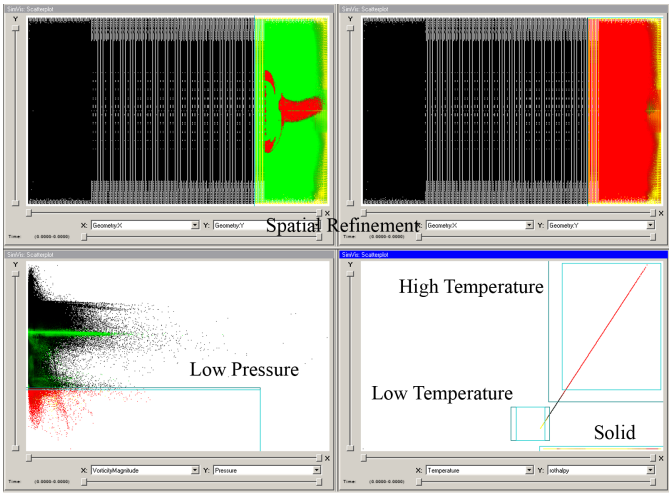


Figure 6.5: This figure shows the selections, which have been made to produce the final visualization.

consists of 861,820 triangles and 883,243 quads. Furthermore it should be noted, that only few unstructured grid visualization approaches can deal with grids containing more than 6 million cells interactively. Since nearly all state of the art methods can only handle tetrahedral volume cells the generator dataset would have to be tetrahedralized to be visualized with them, which would result in a cellcount of over 15 million as shown in table 6.3.

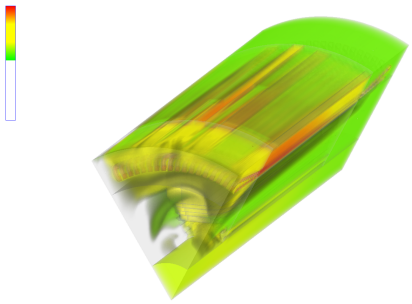
The setup within the dataset assumes, that air is being blown through the generator from one end to the other, while the central part of it, which is called the rotor, is rotating. This rotation induces electric current in the surrounding coils, which are thus heated up. In order to increase the efficiency of the generator it is necessary to provide efficient cooling, which means, that the air flow around the coils surrounding the rotor has to be optimized. Figure 6.4 gives a rough overview over the dataset geometry by showing only the solid portions of the generator with color representing the temperature. As expected the coils at the air inlet are cooler than at the air outlet.

Hybrid Raycasting Applied to the Generator Dataset

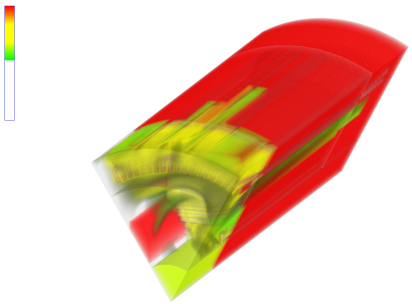
The exploration and analysis of a dataset with the SimVis system basically starts with trying to get an overview over the whole dataset. Figure 6.4 shows such a simple overview. Here only the solid portions of the dataset have been selected. The mapping of temperature to color makes it easy for the user to discriminate the air inlet from the air outlet.

The next step for the user is to focus on the data which interests her or him. In the Generator dataset very warm and very cold air is of interest in the context of the solid parts of the dataset. Thus those two portions of the air within the dataset are selected. Furthermore the whole solid structure of the generator is selected. Figure 6.6(a) gives

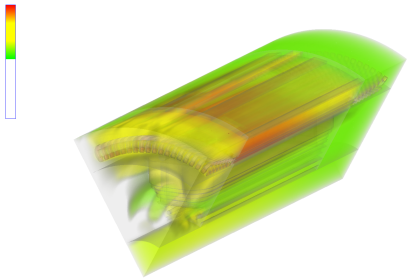
a good overview over this selections. The structure of the selected regions behind the rotor at the air outlet look, as if they are sucked into a vortex. In order to verify this, a local measure to identify vortical structures has to be used. There is a great number of different methods to identify vortices, which are discussed in a state of the art report by Post et al. [56]. In the Generator dataset a rather simple method is sufficient to identify the core of the vortex. Since the centrifugal force within a vortex causes low pressure at its center selecting dataset portions with low pressure values produces good results. Figure 6.6(c) now shows all four selections, which are low and high temperature air, all the solid portions of the dataset and the low pressure areas behind the rotor. As expected the low pressure portion of the dataset is just at the center of the structure, which has been assumed to be a vortex, after observing the low and high temperature selections. As shown in figure 6.6(f) huge parts of the dataset have been resampled to make interactive exploration and analysis possible. In order to focus on the interesting dataset portions behind the rotor at the air outlet all selections are further restricted in their spatial dimensions. Now nearly all relevant bricks of the dataset are rendered in their original unstructured representation, as shown in figure 6.6(f). In figure 6.5 all selections, which have been made in the SimVis famework on the different data dimensions, are shown.



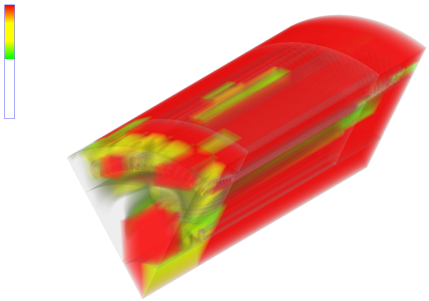
(a) high and low temperature air



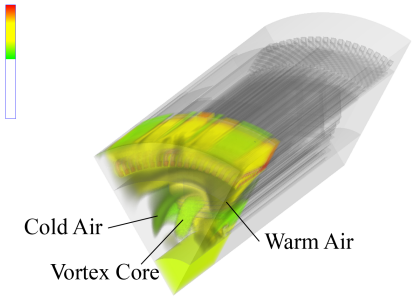
(b) resampled bricks shown in red



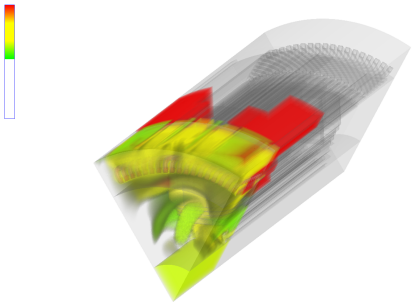
(c) high/low temperature and low pressure air



(d) resampled bricks shown in red



(e) focusing on interesting portions



(f) resampled bricks shown in red

Figure 6.6: The figures show the different steps of a visualization session with the SimVis system using the hybrid raycasting rendering frontend. First an overview is used to find interesting features. Subsequent steps are used to focus and further explore them. For each step the resampled bricks are shown in red in the right column of figures.

Chapter 7

Summary

In this chapter a summary over the contents of this thesis is presented. First a short introduction is given, following a brief summary of related work. After this the bricking approach for unstructured grids is summarized. Then the hybrid raycasting for unstructured and structured grids is revisited. The concluding two sections deal with results and present a conclusion and future work.

7.1 Introduction

Simulations in the engineering field are nowadays a very common way to test designs and concepts before building expensive prototypes. The resulting datasets are highly complex since multiple physical properties have to be computed and analyzed. The basis for the datasets are most of the time so called unstructured grids, which are designed manually by the simulation specialists to fill up the simulated spatial domain as good as possible. Most commercial simulation packages [8, 3, 2] base their computations on unstructured grids, which contain not only tetrahedra but also a variety of other cell types. Newer developments in the simulation field even introduce grids built out of arbitrary polyhedra. Current interactive visualization approaches deal with unstructured grids by tetrahedralizing them, since tetrahedra can be processed more straight forward. This increases the already large number of cells by introducing redundant data. Contrary to these approaches the method presented in this thesis is able to deal directly with unstructured grids which are used by current simulation packages.

Besides the complexity of the datasets their sheer size (even without tetrahedralization) additionally poses severe difficulties to visualization systems, which try to provide the user with an interactive visualization. Thus level of detail methods are usually used if interactivity has to be achieved. Most approaches, which directly deal with unstructured grids utilize mesh simplification techniques based on the whole dataset. The method proposed in this thesis adapts parts of the bricking concept, which is used for structured data, to unstructured grids. In order to gain rendering speedups and memory savings only portions of the dataset, which are of importance to the user, are rendered based on their original representation, while unimportant data regions are resampled to

structured grids. The specification of importance within a dataset is performed in the commercially available SimVis system, into which the presented approach is integrated.

7.2 Related Work

The rendering frontend which is presented in this thesis is integrated in the visualization system SimVis [23, 24, 22], which can be used to interactively explore and analyze volume data from different fields, like engineering or meterology. It is based on a feature-based approach which lets the user specify his interest in different portions of the data by providing different linked views of the attribute-space. This results in a so called Degree of Interest (DOI) Function which is defined over the simulated domain and assigns a value between zero and one to each volume cell. Higher values represent high and low values low user interest. The visualization of this DOI function is then performed by blending between a focus and a context representation. The rendering algorithm presented in this thesis adapts this focus+context visualization approach based on the DOI function which is provided by the SimVis system. The visualization method used, which is presented in this thesis, is based on volume rendering. There are two basic groups of methods which are used to perform volume rendering. Object-order methods project the volume elements of a dataset onto the viewing plane. Here it is necessary to perform proper visibility sorting to guarantee correct results, which can be time consuming especially if concave meshes have to be visualized. Most object-order methods are based on the Projected Tetrahedra algorithm of Shirley and Tuchman [68]. The main difficulty here is the efficient sorting, which is mostly performed by methods based on the MPVO algorithm of Williams [86]. Comba et al. [18] and Silva et al. [70] propose extensions to the MPVO which make the application to concave meshes possible. Röttger et al. [59, 60] mainly focus on the evaluation of the optical volume model based on a pre-integration approach. Callahan et al. [13] presents an algorithm which performs only an approximate visibility sort and utilizes the programmability of modern graphics hardware to perform the final sorting during the rasterization of the cells.

Contrary to object-order methods image-order approaches perform the visualization not by rasterizing the volume elements one by one, but by computing the contribution of each cell for every pixel of the result image. Thus such methods are not that highly dependent on the steadily growing number of cells of a data volume. Basically all image-order volume rendering approaches are based on the unstructured grid raycasting method proposed by Garrity [35]. Weiler et al. [80, 81] present a graphics hardware based implementation which is extended to directly coping with concave grids by applying a technique similar to depth peeling [30]. The unstructured grid raycasting method proposed in this thesis is basically based on Weiler's hardware based approach, which is extended to directly cope with more complex cell types.

In order to cope with datasets, which are too large to be visualized interactively as a whole, level of detail techniques are used. Such methods can either use stepwise simplification to reduce the amount of data, as proposed by Kraus and Ertl [43] and Cigoni et al. [17] or employ resampling to structured grids and apply level of detail

approaches used in this field, as proposed by Leven et al. [48]. When coping with structured grids mostly spatial subdivision data structures are used to decompose the data into bricks of different resolution levels. This is proposed by Lamar et al. [46] and is further refined by Weiler et al. [46], Boada et al. [10] and Guthe et al. [37]. The level of detail method proposed in this thesis is based on resampling portions of the data to structured grids to generate a low resolution representation. But contrary to Cigoni et al. high detail rendering is performed on the original unstructured data.

7.3 Hybrid Structured and Unstructured Grid Raycasting

The method proposed in this thesis copes with large volume datasets by using a three level of detail approach. The data volume is decomposed into bricks containing a fixed amount of cells. Every brick can be either displayed in its original unstructured representation, as a resampled low quality version or as context. In the context representation only the boundary of the unstructured mesh is rendered. All bricks, which are not part of the context are called focused bricks. The brick decomposition itself and the decision criteria used to determine which representation should be rendered, is discussed in section 7.4. All raycasting algorithms, which are used are fully GPU based and utilize the programmable fragment processing stage of modern graphics hardware to achieve interactivity. Thus all the input data for one selected representation of a brick has to be present in texture memory.

The hybrid raycasting algorithm can be parameterized in multiple ways to create different visualizations. First of all the optical model, which is applied during the raycasting, can be selected by the user. Presently five optical models are available, which can be used to emphasize different aspects of the data. All of the optical models take a scalar field value and the DOI function into account which results in focus+context visualizations. Furthermore the rendering method, which should be used to display the mesh boundary, can be selected. Presently three different boundary visualization techniques are available which results in a total of 15 different rendering modes.

In the following two sections the structured and unstructured grid raycasting methods are presented. To cope with concave meshes both use the technique, introduced by Weiler et al. [81], which is an extension of depth peeling [30] to volume rendering. The context visualization of the mesh boundary is performed with the original depth peeling method.

7.3.1 Structured Grid Raycasting

The structured grid raycaster is based on the GPU-based single pass approach of Krüger and Westermann [45]. The input to the algorithm are textures containing the start- and end-positions of a view ray and the face normals at these points (which are needed by the boundary visualization). All four textures are created in two passes, one for the front faces and one for the back faces, using multiple render targets. Starting at the start position a ray is traversed through the resampled data volume, using the optical model

to compute opacity and color values within the volume and the boundary visualization method to retrieve color and opacity values for the mesh boundaries.

Since the resampled representation is used as low quality version of the underlying unstructured data the raycasting itself is not supposed to create high quality images. Thus a coarse fixed sampling rate along the view ray is chosen. Front to back compositing is used in combination with early ray termination to further speed up the rendering. The resampling of the DOI and scalar field data itself is a simple object-order method which uses the bounding box of the cells to create their resampled representations.

7.3.2 Unstructured Grid Raycasting

The unstructured grid raycasting method used is comparable to the GPU-based implementation proposed by Espinha and Celes [29]. But instead of only processing tetrahedral grids the algorithm is extended to a number of supported cell types which are commonly used by different simulation packages. The data, which is used by the raycaster is stored in two sets of two-dimensional textures. The first set, the cell data texture set, contains data about the volume cells themselves, like scalar field values, DOI values and the cell type. The second texture set is the cell face data texture set, which stores data about the cell faces. This includes the neighboring cell indices and the faces plane equation parameters. In order to determine the start position and start cell index of a view ray the mesh boundary is rasterized into a texture, using the color of each face to encode its cell and face index. From there on the raycaster traverses the connectivity graph, which is implied by the neighborhood information stored in the cell face data texture set. For every cell its cell faces are intersected with the view ray. The first intersection of a plane, which faces away from the view direction, is chosen as exit point. Thus the corresponding neighbor is checked next. The ray integration assumes constant scalar and DOI values along a ray segment within a cell. Thus no interpolation is performed (interpolating in arbitrary convex polyhedra is a highly difficult task, which is left as future work for now).

7.4 Bricking for Unstructured Grids

Besides the raycasting itself the brick decomposition of the unstructured data volume is an important aspect of the presented rendering frontend. Because of the brick decomposition several optimizations are possible. First the cell neighbor indices can be stored with only two bytes per neighbor, which helps saving memory. Furthermore the number of depth peels, which have to be rendered is reduced dramatically. Additionally it allows for different rendering modes for portions of the dataset. This is especially important since empty or uninteresting portions of the dataset can be rendered with less precise, but faster methods.

Basically the brick decomposition is performed in a top down approach by creating a kd-tree whose leaves have to contain less than 64K cells, since a two byte index is used to identify them within a brick. It is important to note, that cells, which are intersected by the brick boundaries have to be stored for every brick adjacent to the boundary. Thus

the hierarch generation is performed in two steps. In the first cells, which are not cut by a brick boundary, are considered and used to create a first rough hierarchy. In the second step all remaining cells, which are stored for each subdivision plane intersecting them, are added to the corresponding bricks. If a brick exceeds the maximum number of cells it is split again in this step.

In order to perform the structured and unstructured raycasting for a brick it is necessary to rasterize the cell-bounding-plane intersections. If the unstructured grid data is available for a brick it is possible to utilize this data to perform the intersection computations. This is achieved by rendering one quad for each cells, which has to be large enough to cover the cell-boundary intersection. A fragment program is then used to discard all pixels, which lie outside the cell by checking its position relative to the cells face planes. The main advantage of this approach is, that only simple quads have to be stored and rendered, which is faster than using arbitrary polygons. If the face plane information is not available in texture memory, which is the case if a resampled brick has to be rendered, the cell-boundary intersections are computed explicitly on the CPU.

7.5 Discussion

The hybrid raycasting approach presented in this thesis introduces several state of the art techniques into the SimVis framework. Raycasting-based volume rendering is adapted to the focus+context visualization approach of the overall framework. A brick decomposition, which has up until now, only be used by pure structured grid raycasting methods, is adapted for unstructured grids. This introduces additional flexibility and allows for several optimizations which help to retain interactivity even for very large datasets, which could barely be handled by other state of the art methods. This is also possible because no tetrahedralization has to be performed before the visualization, since the hybrid raycsting approach is able to deal with unstructured grids containing different cell types. This additional flexibility has the drawback, that interpolation of data values within the cells becomes extremely difficult. The presented implementation does not support any interpolation at all, which means that data values within each cell are considered constant over the whole cell volume. This means, that result images may have a certain "blocky" quality. Still data which is truly cell centered, like results from finite volume approaches like CFD, is visualized appropriately, since the "blocky" representation is true to the underlying data.

Besides the proper volume rendering itself the visualization of the mesh boundary is of crucial importance. Many unstructured datasets from the computational simulation field have very complex surfaces, which have to be represented appropriately, to allow the user to better grasp the interrelations between the observed flow properties within the simulated data volume and its boundary. Special care has been taken to allow for exact boundary visualization even for resampled bricks.

Chapter 8

Conclusion and Future Work

This chapter concludes the thesis and gives a short overview over future work. Several possible extensions to the algorithm are suggested and discussed briefly.

8.1 Conclusion

The work presented in this thesis describes a rendering algorithm, which has been seamlessly integrated into the SimVis system. Focus+context techniques based on a Degree of Interest function are applied to raycasting-based volume rendering. In combination with the introduction of bricking into the field of unstructured volume rendering, this allows for several optimizations, like empty space skipping of completely uninteresting portions of the data volume or using a low quality resampled volume representation for barely interesting regions. Furthermore simple optimizations, like view frustum culling of individual bricks is now possible.

Several different rendering modes have been introduced. For example simple cell face visualization, which can be used to inspect the quality of the meshing of a data volume. More complex shaded volume rendering and multiple transparent isosurface visualization is also supported. The presented raycasting framework itself is highly flexible and extensible and allows for a fast integration of new rendering approaches.

Interaction and immediate feedback to user input is an extremely important aspect of the SimVis framework. Thus the rendering frontend presented in this thesis has to handle even large unstructured datasets interactively. Besides the already mentioned bricking approach, which allows for some important optimizations, interactive frame rates are made possible by displaying lower quality images and reducing the number of depth peels during interaction.

Since the SimVis system can deal with unstructured grids, which contain different cell types, already existing volume rendering approaches, which mostly can only deal with tetrahedra, have been extended to cope with other cell types as well. Thus no tetrahedralization, which would increase the number of cells and thereby also the memory requirements, is necessary. Since interpolation within arbitrary polyhedra is highly difficult the presented implementation considers the data constant within each cell. This

introduces some "blockyness" into the result images, which may disturb some users. Still the visualization of truly cell centered data coming from areas like finite volume methods (CFD) is appropriate even without interpolation. For node centered, or smoothed out cell centered, visualizations future work certainly will have to deal with interpolation.

Besides the volume rendering itself the visualization of the mesh boundary is often crucial as well. Especially data coming from simulations in the engineering field is defined on meshes with highly complex surfaces. To better grasp the interaction of the simulated flow features with the volume boundary it can be visualized with different methods. It should be noted, that even the mesh boundary within resampled bricks is still rendered explicitly allowing for homogenous results even for very large datasets.

If simple meshes, like curvilinear grids, have to be rendered the original point-based approach used in the SimVis system is still more suitable, since all cells are approximately equally sized. Furthermore the application of Gaussian splats introduces interpolation, which is not possible for the implemented raycasting approach. The new rendering frontend is especially well suited to visualize large unstructured grids with highly complex mesh surfaces. Here the proper boundary visualization and the availability of different rendering modes helps to create meaningful images.

8.2 Future Work

In this section several additional ideas are discussed, which could be used to extend the current raycasting approach. They are related to different parts of the algorithm, like the missing interpolation possibilities or the lack of real flow vector visualization.

8.2.1 Interpolation

As already mentioned the lack of interpolation introduces some "blockyness" into the result images. The interpolation within convex polyhedra is not trivial and thus has been left as future work. There are several ways this problem can be tackled. Either an on the fly decomposition into tretraheda can be performed during the traversal of a cell and the simple interpolation schemes, as used by other approaches, can be applied. This approach would not introduce additional memory requirements, since the decomposition can be performed during the raycasting. Another possibility only works, if the basis mesh is composed of cell types, which can be created by collapsing the edges of a hexahedra (tetrahedra, prisms, pyramids and octahedra are such cells). Thus each cell can be "morphed" into a unit cube and trilinear interpolation can be applied.

8.2.2 Integration of FlowVis

Since the SimVis system is mostly dealing with simulation data containing flow vector information a direct method to visualize this information would also be an interesting way to extend the existing hybrid raycasting method. There are various techniques, which are used to visualize three-dimensional flow data. Probably the most promising approach would be to apply diffusion-based techniques based on the flow vector,

generating three-dimensional structures, aligned with the flow direction. Additionally three-dimensional LIC or similar techniques could be considered. Since all those techniques need a certain neighborhood around a cell it is probably necessary to either store additional cells at the boundary of the bricks or to perform a diffusion step between bricks.

8.2.3 Fuse Structured and Unstructured Data

Since the existing hybrid raycasting approach can only fuse structured and unstructured data on a brick by brick basis a logic extension would be to support mixed bricks. Such bricks would contain data on a structured and an unstructured mesh. This is especially interesting, if measured and simulation data should be visualized at once. For example a vein from a CT data volume could be extracted and a flow simulation based on its geometry can be performed. If a fusion of structured and unstructured data within one brick is possible this result data could be visualized properly in the context of the source CT data volume. In order to do this it is probably most straight forward to do the unstructured raycasting and sample the structured datavolume while traversing the cell connectivity graph.

8.2.4 Porting to Other Graphics Hardware

The present implementation of the hybrid raycasting frontend is based on OpenGL and Cg. In theory this should guarantee portability between different graphics hardware. But since different GPU manufacturers implement different extensions and support not always the same functionality the raycasting implementation has to be adapted. Presently only Nvidia hardware from the NV30 and up is supported. It is desirable to additionally support the corresponding ATI hardware. Since it is presently better suited to execute complex shaders and is probably even faster than the presently used Nvidia GPUs.

Chapter 9

Acknowledgements

I would like to thank my supervisor Helwig Hauser, who introduced me to the highly interesting and challenging field of visualization, that he allowed me to choose the topic of this thesis so freely. Furthermore i am grateful for the help and advice provided by Markus Hadwiger and Helmut Doleisch who informally supervised the development of this work.

Furthermore I am indebted to Harald Piringer, Raphael Bürger, and Martin Brunnhuber, who are colleges at the VRVis research center and gave valuable input and helped me with the further development of the SimVis system.

Additionally I have to thank Matthias Braun, Susanne Fischer, Simon Dumke, and Stefan Tomitsch, some former students, who did their internships at the VRVis and helped greatly extending the SimVis framework.

I would like to also thank Markus Trenker and Hermann Lang from Arsenal Research for providing me with interesting datasets to push the hybrid raycasting method to its limits.

Similarly I am grateful to Oliver Schögl and Stephan Schmidt from the "Institut für Verbrennungskraftmaschinen und Thermodynamik" at the Technical University of Graz for providing interesting datasets.

Most of this work has been done at the VRVis Research Center, which is funded in part by the Kplus program of the Austrian government. Additionally I have to thank the FFG for funding parts of this work. The CFD datasets related to the automotive industry are courtesy of the "Institut für Verbrennungskraftmaschinen und Thermodynamik" at the Technical University of Graz and AVL List GmbH, Graz, Austria. The head and generator datasets are courtesy of Arsenal Research, Vienna, Austria.

Bibliography

[1] Directx. See URL: http://www.directx.com.

[2] Fire. See URL: http://www.avl.com.

[3] Fluent. See URL: http://www.fluent.com.

[4] Frame buffer object extension spec. See URL: http://www.opengl.org/registry.

[5] Non power of two extension spec. See URL: http://www.opengl.org/registry.

[6] Occlusion query extension spec. See URL: http://www.opengl.org/registry.

[7] Opengl. See URL: http://www.opengl.org.

[8] Starcd. See URL: http://www.cd-adapco.com.

[9] F. F. Bernardon, C. A. Pagot, J. L. D. Comba, and C. T. Silva. Gpu-based tiled ray casting using depth peeling. Technical Report UUSCI-2004-006, SCI Institute, University of Utah, 2004.

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

[11] P. Bunyk, A. E. Kaufman, and C. T. Silva. Simple, fast, and robust ray casting of irregular grids. In *Scientific Visualization*, pages 30–36, 1997.

[12] 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 *Proceedings IEEE Visualization 2005*, page 26, 2005.

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

[14] L. Carpenter. The a -buffer, an antialiased hidden surface method. In *Computer Graphics (SIGGRAPH '84)*, pages 103–108, 1984.

[15] S. Castro, A. Konig, H. Loffelmann, and E. Groller. Transfer function specification for the visualization of medical data. Technical report, Institute for Computergraphics and Algorithms, Vienna University of Technology, 1998.

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

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

[18] J. Comba, J. T. Klosowski, N. L. Max, J. S. B. Mitchell, C. T. Silva, and P. L. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. *Computer Graphics Forum*, 18(3):369–376, 1999.

[19] T. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture mapping hardware. Technical Report TR93-027, Department of Computer Science, University of North Carolina, Chapel Hill, 1993.

[20] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-quality volume rendering using texture mapping hardware. In *Proc. of Eurographics / SIGGRAPH Workshop on Graphics Hardware 1998*, pages 69–76, 1998.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. Real-time volume graphics. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, page 29, 2004.

[28] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. of Eurographics / SIGGRAPH Workshop on Graphics Hardware 2001*, pages 9–16, 2001.

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

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

[31] S. Fang, T. Biddlecom, and M. Tuceryan. Image-based transfer function design for data exploration in volume visualization. In *Proceedings IEEE Visualization '98*, pages 319–326, 1998.

[32] R. C. Farias, J. S. B. Mitchell, C. T. Silva, and B. Wylie. Time-critical rendering of irregular grids. In *SIBGRAPI*, pages 243–250, 2000.

[33] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80)*, pages 124–133, 1980.

[34] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics (SIGGRAPH'97)*, pages 209–215, 1997.

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

[36] N. K. Govindaraju, M. Henson, M. C. Lin, and D. Manocha. Interactive visibility ordering and transparency computations among geometric primitives in complex environments. In *SI3D*, pages 49–56, 2005.

[37] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *Proceedings IEEE Visualization 2002*, 2002.

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

[39] H. Hauser and M. Mlejnek. Interactive volume visualization of complex flow semantics. In *Proc. of the 8th Fall Workshop on Vision, Modeling and Visualization (VMV 2003)*, pages 191–198, 2003.

[40] W. Hong, F. Qiu, and A. E. Kaufman. GPU-based object-order ray-casting for large datasets. In *Volume Graphics*, 2005.

[41] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proc. IEEE Symposium on Volume Visualization '98 (VolVis '98)*, pages 79–86, 1998.

[42] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings IEEE Visualization 2001*, pages 255–262, 2001.

[43] M. Kraus and T. Ertl. Simplification of nonconvex tetrahedral meshes, January 10 2000.

[44] M. Kraus and T. Ertl. Topology-Guided downsampling. In *Proceedings of the 2001 Eurographics/IEEE TVCG Volume Graphics Workshop (VG-01))*, pages 223–234, 2001.

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

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

[47] 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 *Proceedings IEEE Visualization 2005*, pages 623–630, 2005.

[48] J. Leven, J.Corso, J. Cohen, and S. Kumar. Interactive visualization of unstructured grids using hierarchical 3D textures. In *Proc. IEEEE Symposium on Volume Visualization 2002 (VolVis 2002)*, pages 37–44, October 28–29 2002.

[49] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[50] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[51] N. L. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[52] N. L. Max, P. Williams, and C. Silva. Approximate volume rendering for curvilinear and unstructured grids by hardware-assisted polyhedron projection. *International Journal of Imaging Systems and Technology*, 11(1):53–61, 2000.

[53] J. Meredith and K. Ma. Multiresolution view-dependent splat based volume rendering of large irregular data. *2001 Symp. on Parallel and Large-Data Visualization and Graphics*, pages 93–99, 155, October 2001.

[54] M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming delaunay tetrahedralization. In *SODA*, pages 67–74, 2000.

[55] M. E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. In *ACM'72: Proceedings of the ACM annual conference*, pages 443–450, 1972.

[56] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. Feature extraction and visualization of flow fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100, 2–6 September 2002.

[57] R. Sedgewick. *Algorithms in C.* Addison-Wesley, 1998.

[58] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Proc. of Eurographics / SIGGRAPH Workshop on Graphics Hardware 2000*, pages 109–118,147, 2000.

[59] S. Röttger and T. Ertl. Cell projection of convex polyhedra. In *Proceedings of the 2003 Eurographics/IEEE TVCG Volume Graphics Workshop (VG-02)*, pages 103–108, July 7–8.

[60] S. Röttger and T. Ertl. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *Proc. IEEEE Symposium on Volume Visualization 2002 (VolVis 2002)*, pages 23–28, 2002.

[61] S. Röttger and T. Ertl. Fast volumetric display of natural gaseous phenomena. In *Computer Graphics International*, pages 74–83, 2003.

[62] S. Röttger, S. Guthe, A. Schieber, and T. Ertl. Convexification of unstructured grids. In *Proc. of the 9th Fall Workshop on Vision, Modeling and Visualization (VMV 2004)*, pages 283–292, 2004.

[63] S. Röttger, S. Guthe, D. Weiskopf, and T. Ertl. Smart hardware-accelerated volume rendering. In *Proc. of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)*, pages 231–238, 2003.

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

[65] H. Scharsach, M. Hadwiger, A. Neubauer, and K. Buhler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *Proc. of the 8th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2006)*, pages 315–322, 2006.

[66] S. Schmidt, O. Schögl, R. Kirchberger, H. Doleisch, P. Muigg, H. Hauser, M. Grabner, A. Bornik, and D. Schmalstieg. Novel visualization and interaction techniques for gaining insight into fluid dynamics in internal combustion engines. In *Proceedings of the NAFEMS Worldcongress*, page full Proceedings on CDROM, 2005.

[67] C. E. Shannon and W. Weaver. The mathematical theory of communication. *Scientific American*, July 1949.

[68] P. Shirley and A. A. Tuchman. Polygonal approximation to direct scalar volume rendering. In *Proceedings San Diego Workshop on Volume Visualization, Computer Graphics*, volume 24, pages 63–70, 1990.

[69] 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.

[70] C. T. Silva, J. S. B. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proc. IEEE Symposium on Volume Visualization '98 (VolVis '98)*, pages 87–94, 1998.

[71] S. Stegmaier, M. Schulz, and T. Ertl. Resampling of large datasets for industrial flow visualization. In *Proc. of the 8th Fall Workshop on Vision, Modeling and Visualization (VMV 2003)*, pages 375–382, 2003.

[72] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics*, pages 187–195, 2005.

[73] C. M. Stein, B. G. Becker, and N. L. Max. Sorting and hardware assisted rendering for volume visualization. In *Proc. IEEE Symposium on Volume Visualization '94 (VolVis '94)*, pages 83–89, 1994.

[74] D. C. Steiner, É. C. de Verdière, and M. Yvinec. Conforming delaunay triangulations in 3D. *Comput. Geom*, 28(2-3):217–233, 2004.

[75] 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.

[76] E. Tejada and T. Ertl. Large Steps in GPU-based Deformable Bodies Simulation. *Simulation Practice and Theory. Special Issue on Programmable Graphics Hardware*, 13(9):703–715, 2005.

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

[78] A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. In *Proc. IEEE Symposium on Volume Visualization '96 (VolVis '96)*, pages 23–ff., 1996.

[79] M. Weiler and T. Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proceedings IEEE Visualization 2001*, 2001.

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

[81] M. Weiler, P. N. Mallón, M. Kraus, and T. Ertl. Texture-encoded tetrahedral strips. In *Proc. IEEE Symposium on Volume Visualization 2004 (VolVis 2004)*, pages 71–78, 2004.

[82] M. Weiler, R. Westermann, C. D. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *Proc. IEEE Symposium on Volume Visualization 2000 (VolVis 2000)*, pages 7–13, 2000.

[83] 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.

[84] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. *Computer Graphics (SIGGRAPH '98)*, 32(4):169–179, 1998.

[85] L. Westover. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH '90*, pages 367–376, 1990.

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

[87] G. Wills. 524,288 ways to say "this is interesting". In *Proc. IEEE Symposium on Information Visualization 1996 (InfoVis '96)*, pages 54–61, 1996.

[88] M. Wolfahrt. Storytelling for presentation in volume visualization - visualization stories. Master's thesis, Vienna University of Technology, 2006.

[89] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral projection using vertex shaders. In *Proc. IEEEE Symposium on Volume Visualization 2002 (VolVis 2002)*, pages 7–12, October 28–29 2002.

[90] R. Yagel, D. M. Reed, A. Law, P. Shih, and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Proc. IEEE Symposium on Volume Visualization '96 (VolVis '96)*, pages 55–62, 1996.

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