Large Model Visualization: Techniques and Applications

Dissertation

der Fakultät für Informatik
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von
Dipl.-Inf. Dirk M. Bartz
aus Tübingen

Tübingen
2001

Tag der mündlichen Qualifikation:	9. Mai 2001
Dekan:	Professor Dr. Andreas Zell, Universität Tübingen
1. Berichterstatter:	Professor Dr. Wolfgang Straßer, Universität Tübingen
2. Berichterstatter:	Professor Dr. Amitabh Varshney, University of Maryland, College Park

Zusammenfassung

Die Größe von Datensätzen im Bereich des wissenschaftlichen Rechnens (Scientific Computing) wächst rapide. Dies wurde durch die rasante Steigerung der Verarbeitungsleistung von Computern in den letzten Jahren verursacht, die in eine erhöhte Genauigkeit und Modellgröße investiert wurde. Eine ähnliche Entwicklung ermöglichte eine beträchtliche Leistungssteigerung von modernen Schichtbildtomographen in der Medizin. Mehr als 1000 Schichtbilder mit einer jeweiligen Auflösung von 512×512 Pixel können mittlerweile in der täglichen Routine erzeugt werden. Auch Modelle aus dem Computer-aided Engineering bestehen inzwischen aus vielen Millionen von Polygonen. Leider wächst diese Datenkomplexität deutlich schneller, als die graphische Darstellungsleistung auf modernen Computersystemen. Dies wird nicht nur durch langsamere Leistungssteigerungen der Graphiksubsysteme verursacht, sondern vor allem auch durch die deutlich geringeren Zuwächse der Speicherzugriffsbandbreite bei der Übertragung der Geometrie- und Bilddaten vom Hauptspeicher auf die Graphikhardware.

Methoden der *Large Model Visualization* zielen auf eine Verkleinerung dieser wachsenden Kluft zwischen Datenkomplexität und Darstellungsleistung. Bei den meisten Methoden soll dies durch eine Verringerung der Geometrie- und Pixelkomplexität um mehrere Größenordnungen erreicht werden, die damit auch die Anforderungen an die Speicherzugriffsbandbreite entsprechend reduziert.

In dieser Dissertation werden neue Ansätze aus drei verschiedenen Bereichen diskutiert, die auf eine Verkleinerung der Verarbeitungskomplexität zielen. Die dadurch erreichten Verbesserungen ermöglichen somit eine interaktive Darstellung großer Datensätze.

Im ersten vorgestellten Ansatz, dem Volumerendering, werden Daten als diskrete Abtastwerte auf einem Gitter repräsentiert. Dadurch können die erforderlichen Speicher- und Leistungsgrößen sogar unter die der polygonalen Darstellung sinken. Leider mangelt es in der Literatur an geeigneten Bewertungen der verschiedenen Verfahren. Deshalb werden in dieser Arbeit *zum ersten mal* die wichtigsten Algorithmen des direkten und indirekten Volumerendering gegenübergestellt und bewertet. Die Bewertung wird insbesondere in Bezug auf die visuelle Qualität und den Ressourcenverbrauch als die wichtigsten Entscheidungsparameter durchgeführt.

Rekursive Baumstrukturen (Octrees, BSP-trees, k-d-Trees) gehören zu den beliebtesten Datenstrukturen der Computer Graphik um eine hierarchische Datenrepräsentation zu erzeugen. Leider ist dieser Schritt sehr aufwendig, so daß änderungen im Datensatz einen hohen rechnerischen Aufwand nach sich ziehen. In dieser Arbeit stellen wir ein neues Verfahren vor, das diese Datenstrukturen parallel auf- oder umbauen kann und so den zeitlichen Aufwand proportional zur Anzahl der verfügbaren CPUs reduziert.

Bei der computer-graphischen Darstellung große Datensätze stellt vor allem der sogenannte Speicherflaschenhals ein großes Hindernis dar. Dieser Engpaß verschärft sich durch sogenannte *reichhaltige Pixel* ("richer Pixels"), die die Anforderungen an die Speicherzugriffsbandbreiten deutlich erhöhen. Wir stellen hier neue Methoden aus der Verdeckungsrechnung vor, die neben der Geometrie auch die Pixelkomplexität deutlich verkleinern und so Bandbreitenkapazitäten freigeben.

Im zweiten Teil dieser Dissertation werden Anwendungen der hier präsentierten Ansätze vorgestellt. Speziell präsentieren wir das neue VIVENDI-System für die interaktive virtuelle Endoskopie und andere Anwendungen aus den Bereichen Maschinenbau, wissenschaftliches Rechnen und Architektur.

Abstract

The size of datasets in scientific computing is rapidly increasing. This increase is caused by a boost of processing power in the past years, which in turn was invested in an increase of the accuracy and the size of the models. A similar trend enabled a significant improvement of medical scanners; more than 1000 slices of a resolution of 512×512 can be generated by modern scanners in daily practice. Even in computer-aided engineering typical models easily contain several million polygons. Unfortunately, the data complexity is growing faster than the rendering performance of modern computer systems. This is not only due to the slower growing graphics performance of the graphics subsystems, but in particular because of the significantly slower growing memory bandwidth for the transfer of the geometry and image data from the main memory to the graphics accelerator.

Large model visualization addresses this growing divide between data complexity and rendering performance. Most methods focus on the reduction of the geometric or pixel complexity, and hence also the memory bandwidth requirements are reduced.

In this dissertation, we discuss new approaches from three different research areas. All approaches target at the reduction of the processing complexity to achieve an interactive visualization of large datasets.

As the first approach, we discuss volume rendering, which represents data as a set of discrete samples arranged on a 3D grid. The required memory and rendering costs of a volumetric approach can be significantly lower than for standard polygonal rendering. Unfortunately, the literature lacks a sufficient evaluation of the various volume rendering algorithms. Therefore, we assess the most important algorithms of direct and indirect volume rendering for the first time. In particular, we examine visual quality and resource consumption.

Recursive tree structures (octrees, BSP-trees, k-d-trees) are among the most popular data-structures in computer graphics for the construction of a hierarchical data representation. Unfortunately, this is an expensive operation which requires high computational costs. In this dissertation, we present a new approach for the parallel construction of these data-structures. Using this approach, the required computation time can be reduced by the numbers of available CPUs.

One of the major barriers for the visualization of large datasets is the

memory bottleneck. This barrier aggravates with *richer pixels* which even significantly increase the requirements for the memory access bandwidth. Therefore, we introduce new culling methods which significantly reduce the geometric and the pixel complexity, such releasing bandwidth capacity.

In the second part of this dissertation, we introduce applications of the presented approaches. Specifically, we introduce the new VIVENDI system for the interactive virtual endoscopy and other applications from mechanical engineering, scientific computing, and architecture.

Acknowledgements

The work described in this dissertation would not have been possible without the advise, support and encouragement of many people. First of all, I would like to thank my advisor Wolfgang Straßer for the tremendous degree of freedom I received to work on different topics, advise, and the support to cooperate with numerous academic and industrial partners. I also would like to thank Amitabh Varshney who agreed to be part of my graduation committee, and for support and advise. A large share of successful research is due to a pleasant and fruitful environment. I found this environment in my friends and colleagues at the WSI/GRIS, namely Edelhard Becker, Michael Doggett, Olaf Etzmuß, Thomas Grunert, Johannes Hirche, Tobias Hüttner, Rainer Jäger, Urs Kanus, Michael Keckeisen, Stefan Kimmerle, Anders Kugler, Michael Meißner, Gregor Wetekam, as well as our secretary Adelheid Ebert and our system administrators Helga Mayer and Jürgen Fechter for providing support, not only in a technical sense. To a large share, my work was also supported and shared by friends and colleagues working at the University Hospital at Tübingen. Special thanks to Andreas and Christine Bode, Dirk Freudenstein, Özlem Gürvit, Jürgen Hoffmann, Rupert Kolb, Barbara Kortmann, Martin Skalej, Marion Strayle-Batra, Mechthild Uesbeck, and Dorothea Welte.

To a large extend, the work at an University is only possible with students. Eduard Hiti, Michael Guthe, Frank Knoll, Eugen Resch, and Dirk Staneker among others worked with me on several thesis and student projects. Advising and working with them was never only an one-way street.

Furthermore, I would like to thank all industrial cooperation partners, my academic research partners, for support, advise, and helpful discussions; Jian Huang and Roger Crawfis (Ohio State), Stephan Braun (MPI for Biological Cybernetics, Tübingen), Joachim Hornegger and Stefan Schaller (Siemens Medical Systems), James Klosowski (IBM Watson), Klaus Müller (SUNY Stony Brook), Dan Olsen (EAI), Bengt-Olaf Schneider (NVIDIA), Claudio Silva (AT&T Labs), Alan Ward (HP Workstations Systems Lab), Mike Goss, and Craig Wittenbrink (HP Labs), Montserrat Boo Cepeda of the University of Santiago de Compostela, Margarita Amor Lopez of the University of Da Coruña. The various datasets used in this dissertation are courtesy of EAI, Hewlett-Packard, IBM T. J. Watson Research, Philips Research Labs, Siemens Medical Systems, and of the Clinic of Radiology of the University Hospital at Tübingen.

Work and life is only enjoyable with friends. Several accompanied me along my way. Thanks for celebrating the good times and for comfort in the bad times: Andreas Loos, Corina Sandersfeld, and Marco Zierl. In particular I would like to thank Martina Lanzendörfer for patience and love.

These acknowledgements would not be complete without thanking Elisabeth Bartz. Since my birth, I received invaluable support, advise, and love from her. Many of my dreams would not have come true without her. Thank you very much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The visual representation of complex data has always been a major motivation for computer graphics. However, there have always been computer graphics scenes which were too complex to be rendered within a reasonable time limit, or even too complex to be rendered at all with the available resources. Nevertheless, computer graphics has become one of the primary tools used for the interpretation of data from engineering or science. By the end of the eighties, the application of computer graphics methods for the visual interpretation of scientific data became a research field on its own and was called *Scientific Visualization*. By now, it is almost unthinkable to understand data, which is generated by simulations or is measured, without any graphics or visualization technology. The technology used ranges from simple plot drawings, via direct volume rendering techniques to provide semi-transparent view through a dataset, to illuminated streamlines, and line integral convolutions to visualize the flow of particles through multi-dimensional and multi-variate data.

1.1 Large Model Visualization

In recent years, large model visualization or Large Scale Data Visualization (LSDV) became one of the most important research fields in scientific computing. The reason of the emergence of LSDV lies in the fast increasing size of datasets from various sources. In the United States, research efforts are mostly driven by the Accelerated Strategic Computing Initiative (ASCI) of the US Department of Energy (DOE), focusing on nuclear weapon research, and the Large Scientific and Software Data Set Visualization program (LSSDSV) of the US National Science Foundation (NSF), motivated by simulation of natural phenomena (i.e., global and regional weather, ocean dynamics, high energy and astro-physics, etc.). Besides these initiatives, the increasing dataset size of medical scanners (i.e., multi-slice Computer Tomography, rotational biplanar X-ray) and design review tasks in product data management systems (PDM) drive the need for techniques for large datasets:

- The generated data volumes of simulations from scientific computing can easily grow into the range of tera-bytes.
- The size of scientific measured data frequently exceeds tera-bytes of storage space, not only in academic experiments, but also in commercially driven scientific tasks like in flow experiments in the aircraft and automotive industry, or the oil-and-gas exploration.
- Design review tasks in computer-aided engineering (CAE) have to deal with tessellated, polygonal models of up to 100 million polygons.
- Medical scanners routinely generate data volumes with a resolution of 512^3 voxels (see Appendix A) – some scanners like multi-slice CT even generate more than 1000 image slices in one scan; modern high field MRI scanners can go up to a slice resolution of 2048^2 voxels.

Techniques for the handling of large datasets include database management, architectural aspects of large computing systems, parallel computing, and last but not least, rendering techniques for the visualization of large datasets. Here, we focus on techniques for the advanced visualization of large datasets.

Overview

Two issues are usually the major subject of large data handling; memory efficiency and rendering performance. However, many standard visualization techniques require substantial auxiliary data like spatial data-structures or distance fields which are usually computed in a pre-process. Storing this data can exceed the memory capacities of the visualization host computer, prompting the use of different visualization algorithms. Some visualization applications, i.e. design review tasks or intra-operative navigation-based visualization, require a certain rendering performance to provide *interactive* or even *real-time frame-rates*, where an interactive frame-rate usually specifies more than five frames-per-second (fps), and a real-time frame-rate more than 20 fps. Currently available top-of-the-line computer graphics accelerators achieve a sustained performance of several million triangles per second, which is only satisfactory for the interactive rendering of medium sized models. Unfortunately, the data volume generated by applications in architecture, medicine, mechanical engineering, or scientific computing grows faster than the rapidly increasing graphics performance of modern graphics subsystems. This growing divide requires approaches which reduce the complexity by an order of magnitude.

Several methods have been proposed in the recent years to address this divide. Many of these methods have a hierarchical scene representation in common which usually provides different levels-of-detail. However, this hierarchy has to be constructed, an operation which is potentially very expensive, and imposes additional space requirements on the actual application.

Probably the best known class of methods are mesh-reduction approaches, which reduce the rendering complexity of the given geometry data depending on the required rendering performance or quality. A recent survey on mesh-reduction approaches can be found in [79]. In contrast, subdivision progressively refines a coarse polygonal base mesh until a specifiable error threshold is satisfied [244]. If only a limited transfer bandwidth is available, geometry compression methods can be applied to reduce the storage size of a model [203, 204]. Parallel processing of a given problem reduces the per-pipeline rendering complexity by increasing the number of processing pipelines with the number of CPUs. However, potential bottlenecks, required data replication, or synchronization overheads prevent many applications from achieving an optimal speed-up. While parallel rendering concentrated on large SIMD supercomputers in the past, it experienced a renaissance on modern symmetric multi-processing (SMP) systems, large non-uniform memory access (NUMA) computers, or on clusters of single PC-class or RISC-based workstations [23]. In particular the ASCI and LSS-DSV programs drive the development of methods for large NUMA- and cluster-based systems.

All the approaches so far address the lack of rendering performance by reducing the polygonal complexity of objects, or by distributing the rendering load to several processing entities. However, the overall rendering complexity of individual pixels remains the same. In contrast, visibility and occlusion culling approaches reduce that pixel complexity by removing geometry which is not visible from a specific view-point. In depth-complex scenes – where many polygons are rasterized at the same pixels of the framebuffer, due to the same location in image-space – visibility and occlusion culling enables a reduction of the polygonal complexity of up 90% [22, 25]. If interactive rendering needs to be guaranteed, a budget-oriented rendering system can be applied, which may skip rendering of parts of the models if the budget is not sufficient for the entire model [101, 22, 127].

A technique especially suitable for architectural walkthroughs is image-based rendering [53], where distant parts of the geometric model are approximated by an image (i.e., a texture), since their visual appearance is not changing much [4]. This method also reduces the per-pixel-complexity of a rendered frame. A somewhat related approach is point rendering which computes the required object-space geometry based on a sampling of the image-space [167, 176, 222]. Therefore, the complexity of the rendering is determined by the image-space complexity, not by the geometric complexity of the model. A similar approach was already proposed by Cline et al. in 1988 [42], which used attributed points instead of triangles to render volume datasets from medical scanners. In contrast to the recent methods, the point rendering complexity was determined in object-space.

Volume rendering approaches the problem from a different side. The model is no longer represented as a set of polygons, but as a discrete, volumetric set of samples. Depending on the requested modeling details, the space requirements can be substantially smaller than with a polygonal rep-

resentation [19]. Methods of virtual environments provide different interaction methods for the user of large data. In contrast to traditional rotating, translating, and zooming, the user can interact with the models in a more intuitive way.

In the following sections, we introduce the background for several approaches for an effective handling of large models. Specifically, we look into volume rendering in Section 1.2 and parallel processing 1.3. Based on this approaches, we later present our new results in the Chapters 2, 3, and 4.

1.2 Volumetric Data Representation

Traditionally, computer graphics represented a model as a set of vectors which were displayed on vector graphic displays [65]. With the introduction of raster displays, polygons became the basic rendering primitive, where the polygons of a model were rasterized into pixels, which represent the compounds of the framebuffer. The concept of an image composed of two-dimensional discrete pixels was later extended into a volume, composed of discrete voxels, arranged in a three-dimensional array. The representation of model data as a volume is particularly useful for scientific visualization, since the data is usually computed by a simulation on a multi-dimensional grid with three or more dimensions, or measured by a volumetric scanner like a CT scanner.

In the following sections, we will briefly introduce into the concepts of volume datasets (Section 1.2.1) and volume rendering. Specifically, we describe the two primary approaches to generate a visual representation of the volume dataset; indirect (Section 1.2.3) and direct volume rendering (Section 1.2.4). Both approaches use voxel-based shading, which will be reviewed in Section 1.2.2.

1.2.1 Volume Data

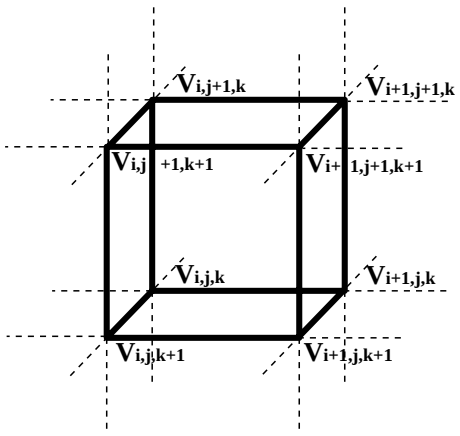


Figure 1.1: Volume cell in a rectilinear grid

A volumetric dataset is a three-dimensional array of data values, the voxels. We address these voxels with three indices i, j, k, indicating the position of a voxel within the volume dataset. If we assume an array of 512 data values in each dimension, each index runs from $i, j, k = 0..511$, forming a $512 \times 512 \times 512$ volume dataset. The three-dimensional array can also be seen as a stack of two-dimensional arrays of data values and each of these two-dimensional arrays as an image (or slice), where each of the data values represents a pixel. This alternative view is motivated by the slice oriented, traditional way physicians look at a volumetric dataset (see also Chapter 5).

Eight neighboring voxels ($V_{i,j,k}$, $V_{i+1,j,k}$, $V_{i,j+1,k}$, $V_{i+1,j+1,k}$, $V_{i,j,k+1}$, $V_{i+1,j,k+1}$, $V_{i,j+1,k+1}$, $V_{i+1,j+1,k+1}$) form a *volume cell*, or simply cell[1] (see Fig. 1.1). The *pixel distance* between the voxels within one slice or image is usually equal. However, the *slice distance* between two neighboring voxels in two neighboring slices is frequently not equal to the pixel distance. We call these datasets *anisotropic* volume datasets. If slice and pixel distance are equal, we speak of an *isotropic* volume dataset. The distances are also called *voxel spacing*. In most datasets, all voxels are aligned on a *cartesian grid*, according to voxel spacing. Therefore, we speak of a cartesian or *rectilinear grid* dataset. If the voxel spacing is constant, the grid is also called an *uniform grid*, or a *non-uniform grid* vice-versa. Some scanners (i.e., 3D ultrasound) produce volume datasets which are not aligned on a cartesian grid, but on a grid which is bent. In other words, the grid geometry has changed, but the grid topology (connectivity) has remained the same. These grids are called *curvilinear grids*. Together with cartesian grids, they are classified as *regular grids*. If also the topology is no longer of a cube-like (possibly bent) cell, the resulting grids are classified as *irregular* or *structured grids*. If the grid topology is using various cell types of different connectivity, we speak of an *unstructured grid*. In the course of this thesis, we will only encounter regular, cartesian gridded volume datasets.

1.2.2 Voxel-based Shading

The shading of an object has a significant influence on the appearance of the object, since it simulates the interaction of light and the material properties of the object. Besides material and light, the shading is determined by the normals of the object, which define its surface. In traditional computer graphics, these normals are defined at the vertices of a triangle and they are interpolated over the area of the triangle to compute the shading of the triangle.

In contrast, the normals of a volumetric representation are defined at the positions of the data values. Due to the lack of a specific surface in a volume dataset, the normals are approximated by gradient operators, which describe changes in the material properties of a volume dataset. This method

[1] An alternative, somewhat outdated terminology describes a voxel as a cubic cell around a data value which extents half way to the next neighboring voxels. The basic difference is the notion of interpolation between the data values, which is nearest-neighbor (non-continuous) for the older scheme, and trilinear (continuous, or C^0) for the current scheme.

was originally motivated by the *partial volume effect* (see Appendix A), which generates smooth density changes between two different materials (i.e., bone and tissue) in CT datasets [105], thus the density gradient represents the surface of the material interface between bone and tissue. The partial volume effect also affects other image modalities, such as MRI, or rotational angiography (see Appendix A).

The currently used standard gradient operator – which we also use later on – computes the central difference between the six direct neighbors of a voxel [105], without taking into account the value of the voxel itself. Other gradient operators include the intermediate difference operator (sometimes also called forward or backward difference operator) – which consider the voxel values at the current position and at three neighboring voxels [140] – and the Sorbel operator, which considers the full 26-neighborhood of a voxel to approximate the gradient [140].

Once the normals at the voxel positions are computed by the (normalized) gradient operators, the normals at the sample points within a volume cell are usually computed by trilinear interpolation of the normals at the eight voxels of this cell.

1.2.3 Indirect Volume Rendering

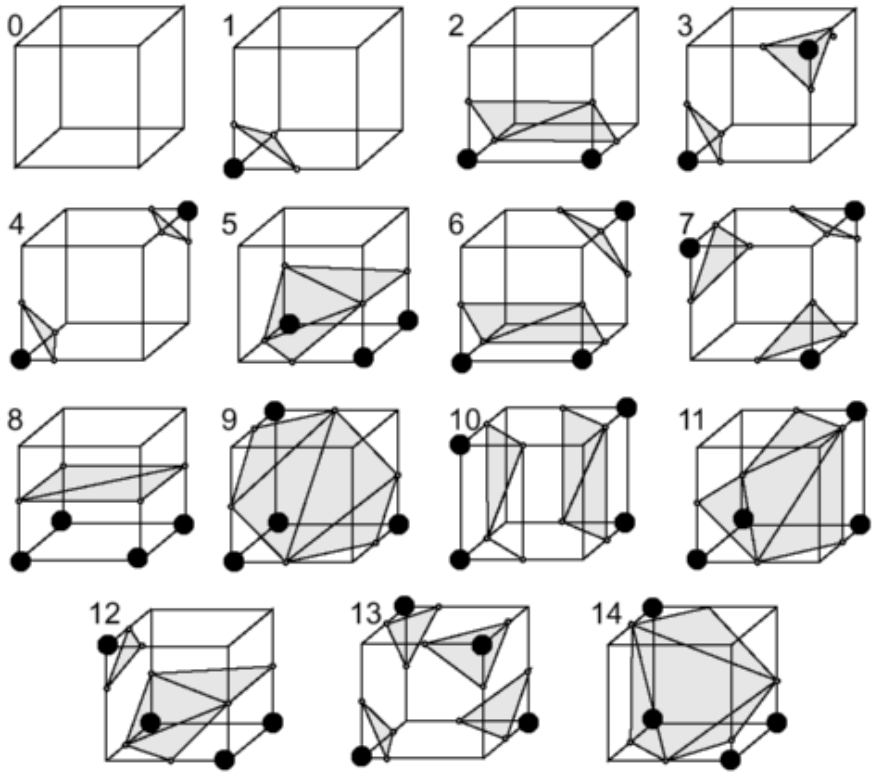


Figure 1.2: Reduced Marching Cubes case table [119]

In indirect volume rendering (IVR), a "traditional" computer graphics representation is generated, usually by extracting an isosurface or an isocontour from the dataset. The respective graphics primitives (i.e., polygons) are then rendered using standard computer graphics hardware. The most popular approach of IVR is probably the Marching Cubes algorithm [142], where each volume cell of eight neighboring voxels is classified according to the specified isovalue (which specifies the isosurface). 2^8 possible combinations whether the scalar value of a voxel of the cell is above or below (or equal to) the isovalue are stored in a table of 256 cases. In the original paper on Marching Cubes, these 256 cases were reduced to 15 cases by inverting or rotating the classification cubes (see Fig. 1.2). However, later research showed that these reduced case table generates inconsistencies, which result in holes in the isosurface [119]. Therefore, we only use a full 256 case table, similar to many other state-of-the-art implementations of the Marching Cubes algorithms [182].

Up to five triangles per cell are generated depending on the classification case. To approximate the normals at the vertices of the generated triangles, each normal is computed by a trilinear interpolation of the normals at the eight voxels of the current cell, which in turn are computed by a central difference operator (see Section 1.2.2) [105].

Several approaches addressed improvement of the basic Marching Cubes algorithm. Most of them focused on skipping of regions which do not contain the isosurface, by applying hierarchical data-structures, like quadtrees and octrees [180], k-D-trees [33], and BSP-trees [76]. Wilhelms and van Gelder introduced the Branch-ON-demand-Octree (BONO) [231], which stores the isovalue interval of the octants to skip octants whose isosurface interval does not contain the specified isovalue.

Many other approaches are known for the efficient isosurface or contour extraction of large volumetric scalar fields. Value partitioning methods [141, 192, 40] store the minimum and maximum values of the voxels of a cell as a pair in a 2D field. Livnat et al. used a k-D-tree structure [141], and Shen et al. [192] used lattice subdivision to subdivide this field. Cignoni et al. used an interval tree as search index for optimal efficiency [40]. Space and value partitioning methods are used for an efficient contour propagation, starting from a small set of seed cells [116, 8].

1.2.4 Direct Volume Rendering

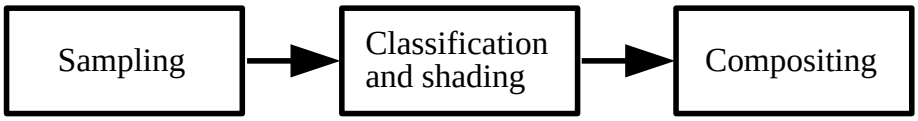


Figure 1.3: (Post-shading) Direct Volume Rendering Pipeline

In contrast to IVR, methods of direct volume rendering (DVR) are gener-

ating images without an intermediate polygonal representation. Instead, the volume dataset is projected onto an image- or view-plane. Figure 1.3 shows an overview of the direct volume rendering pipeline using a post-shading scheme. After computing a sample according to the chosen algorithms, it is *classified* by the transfer functions and shaded according to the specified lights. Finally, the lit sample is *composited* with the previous samples using the over-operator [170].

The transfer functions define the contribution of the samples by mapping their value(s) to a red, green, blue, and α (transparency) value. By evaluating the volume rendering integral (see Equation 1.1), the contribution of the volume data (through the transfer functions) lit by the possibly multiple light sources is computed for each pixel on the image- or view-plane, where I_λ is the color intensity of the pixel at $\vec{x}$, which receives a contribution of a ray of the length L – casted from $\vec{x}$ in direction $\vec{r}$ – and λ corresponds to the wavelength of a color (r, g, b). $C_\lambda(s)$ is the actual color of the sample at position s on the ray, with an extinction coefficient $\mu(s)$ at the sample, which is basically the opacity of the sample. The opacity-weight color is attenuated by the exponential opacity term, which collects the opacity of the ray through the volume (from entry point 0 to current sample s).

$$I_\lambda(\vec{x}, \vec{r}) = \int_0^L C_\lambda(s)\mu(s)e^{(-\int_0^s \mu(t)dt)}ds \tag{1.1}$$

Unfortunately, the analytical volume rendering integral cannot be computed for the general case [147], hence a discrete approximation is used, where Δs is the width of the discretized integration interval:

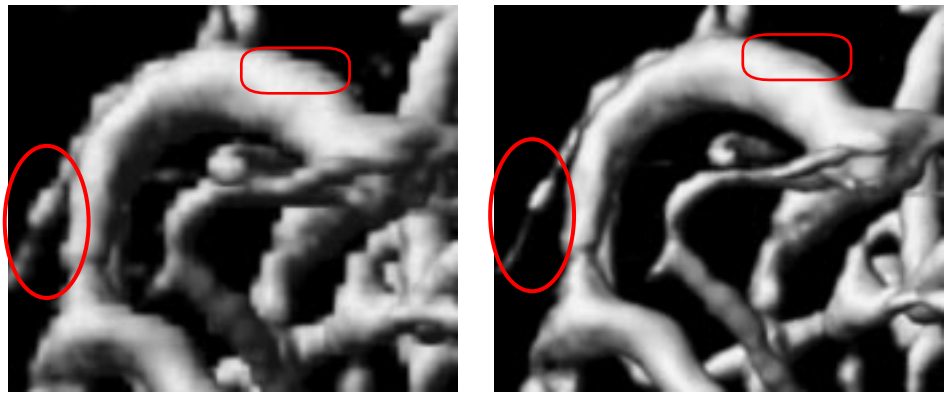
$$I_\lambda(\vec{x}, \vec{r}) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\mu(s_i) \prod_{j=0}^{i-1} e^{(-\mu(s_j)\Delta s)} \tag{1.2}$$

The theoretical basis of the volume rendering integral is the *density emitter model* introduced by Sabella [177], which assumes a simplified model of the transport theory of light [97]. This simplified models takes only absorption and emission into account; physical terms like scattering, influence of the different wavelengths (red, green, and blue), or the influence of participating media are ignored.

Generally, DVR-algorithms are classified into front- and back-projection approaches. While front-projection approaches project the contributions of a dataset onto the view-plane, back-projection approaches trace the contribution for each pixel of the view-plane through the volume. Prominent candidates for front-projection are splatting [229, 230, 161], texture-mapping [50, 35], or cell projection [233]. For back-projection, the most prominent candidate is ray-casting [210, 138].

Another important algorithmic classification is the shading scheme, which basically depends on when and how the normals at the sample points are calculated. Most approaches use the normalized gradient of the sample,

based on central differences, to compute the normal at the sample [105] (also see previous section on voxel-based shading). However, this gradient can be computed before (pre-shading), or after the sample is computed (post-shading), as described above. Pre-shading pre-computes the normal gradients – and hence the illumination and shading – for all voxel values of the volume. The resulting colors and opacities are stored as a color volume instead of the original data. During the actual rendering process, the samples are computed based on the color volume. If the lighting conditions are changing (i.e., modifications of the lights for a diffuse lighting model, or of the view-point in a Phong-like lighting model), this requires the re-calculation of this color volume. Additionally, the colors need to be pre-multiplied by the opacities in order to avoid color bleeding artifacts [236] (see Fig. 1.4). In contrast, post-shading computes the classification and normals during rendering, after computing the sample.



(a) (b)

Figure 1.4: Color bleeding artifacts in a rotational angiography dataset. (a) Non-opacity weight, pre-shaded volume rendering; color bleeding artifacts are marked with red ellipse. (b) Post-shaded volume rendering.

In the following, we will briefly describe four of the most popular direct volume rendering approaches, where each of them has their own specific advantages and disadvantages.

Ray Casting

In the image-space oriented ray casting approaches, rays are cast from the view-point through the view-plane into the volume. Along their way through that volume, samples are calculated usually at equal sampling distances between two sample points (see Fig. 1.5). A sample is computed based on trilinear interpolation within a cell of eight voxels. Thereafter, it is classified according to the transfer functions. If that sample has a contribution to the ray, the normal gradient is computed based on a trilinear interpolation of the normalized central differences at the eight voxels of the cell which contains the sample point. Finally, the sample is composited with the previous

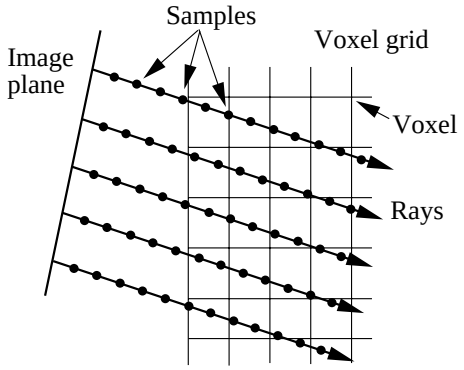


Figure 1.5: Ray casting of a volume dataset with parallel projection and uniform sampling.

samples of the ray.

Acceleration methods include *early ray termination*, where the sampling along the ray is terminated, once (almost) full opacity has been reached, or space leaping, where a distance field or other data-structures indicate empty space where no sampling is required. Ray casting can also utilize oversampling within the view-plane and along the ray to account for high-frequency data in the data volume.

Shear-Warp Factorization

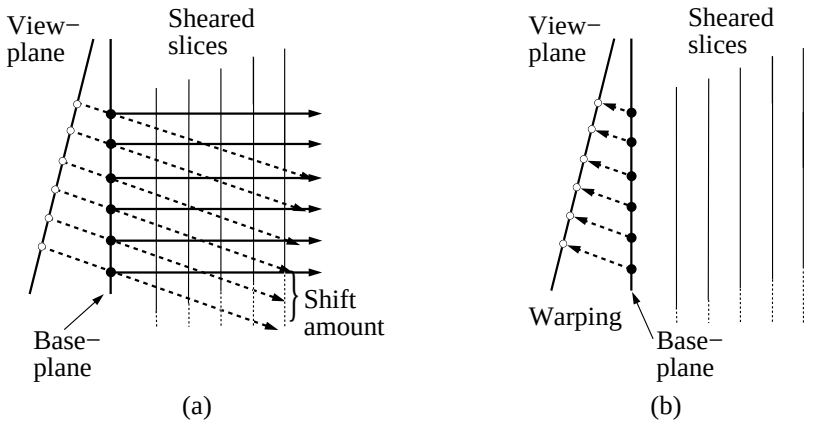


Figure 1.6: Shear-warp Factorization: (a) The volume slices are shifted (sheared) to account for the angle between view- and base-plane. (b) The base-plane image is warped to the view-plane.

A variation of the ray casting approach is the image-space oriented shear-warp factorization, which essentially factorizes the viewing transformation into a shearing and a warping matrix [133]. This enables the casting of rays from each pixel of a volume aligned base-plane, which is most parallel to the image- or view-plane, which in turn can be exploited for an optimized memory access. To take into account the angle between view- and base-

plane, the individual slices are shifted (**sheared**) accordingly (see Fig. 1.6a). The samples of a whole scanline are computed simultaneously within one slice, employing a bilinear interpolation scheme. This results in a view-dependent sampling interval, which can vary between 1.0 for axis-aligned views, to 1.73 ($= \sqrt{3}$) for corner-on views. Perspective projection also requires the scaling of the slices to address the divergence of perspectively cast rays. Finally, the computed base-plane image is **warped** onto the view-plane (see Fig. 1.6b).

The Stanford VolPack-implementation of the shear-warp factorization is highly memory optimized. A pre-computed classification and a run-length-encoding of the opacity weighted voxels are used to rapidly skip the transparent, non-contributing volume space. VolPack uses a pre-shading scheme which computes the lighting on the pre-classified voxels, before generating the sample within a slice. Note that this (pre-)shading operation occurs during rendering, not in a pre-process, which allows the modification of the light settings without re-generating the pre-classified volume dataset.

Splatting

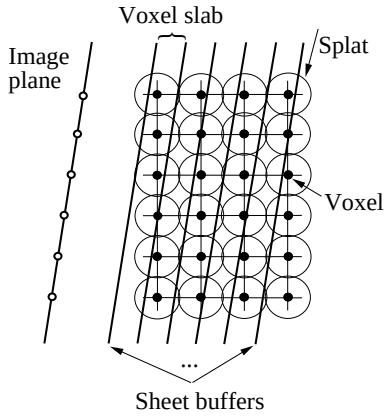


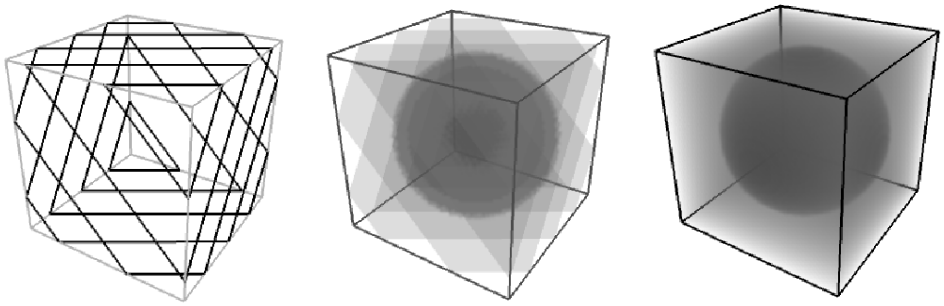
Figure 1.7: Splats arranged on view-plane aligned voxel slaps are projected into sheet buffers, which are in turn composited front to back into the final image on the view-plane.

The object-space oriented splatting approach projects each voxel onto the screen as an overlapping, orientation invariant Gaussian kernel with an amplitude scaled according to the voxel value [229, 230]. High speed is obtained by a footprint lookup table of the pre-computed, radially symmetric kernel function. For correct compositing, the volume is processed by slices oriented most parallel to the view-plane, which causes severe brightness variations, such as popping artifacts in animated views. Mueller et al. modified the splatting approach such that *slabs* of the voxel kernels were processed in an image aligned fashion [161] and projected in *sheet buffers*, which in turn are composited into the final image on the view-plane. Other modifications include the *early splat elimination* for the removal of non-

contributing splats from rasterization [162]. Similar to early ray termination, the (average) opacity of the screen area covered by a splat is tested, if it is above a specified threshold. In this case, the respective splat will not have a significant contribution (or no contribution, if the chosen threshold specifies full opacity). The test itself is performed by convolving the associated opacity buffer area (which stores the up to now accumulated opacity of each sheet buffer) with a box filter of the screen size of the splat, resulting in the opacity average of that area [162].

Instead of tri-/bilinear interpolation-based point sampling, splatting employs a sample average across the sampling distance in view direction. This introduces an additional low-pass filter operation, which reduces aliasing, but also tends to smooth signal characteristics. Finally, the recent splatting approaches [161, 162] also provide post-shaded DVR, in contrast to pre-shading provided by the original splatting approach.

Texture Mapping



(a) (b) (c)

Figure 1.8: 3D Texture Mapping-based Volume Rendering [228]: (a) 3D texture slices are generated from the volume, perpendicular to the view-plane; the texture slices are mapped onto the screen (b) and blended with the previous slice (c).

Object-space oriented texture-mapping is accumulating texture slices perpendicular to the view-plane back-to-front using the blending functionality of graphics hardware (see Fig. 1.8) [50, 35]. The sampling of the texture slices from the volume are either trilinear, if 3D texture mapping hardware is available, or bilinear, if only 2D textures are supported[2]. Shading can be achieved by the computation of a pre-shaded color volume [214], or by using multi-pass methods to visualize isosurfaces [228, 195, 51], or transparent volumes [155]. In our comparison later-on, we use the color volume approach to provide high-speed volume rendering of semi-transparent volumes. However, due to the limited precision of the alpha channel of the framebuffer (only eight bits), we do not pre-multiply the voxels with the opacity which leads to color bleeding artifacts [236] (see Fig. 1.4).

[2] On some graphics systems, quadlinear interpolation is also available, where an additional interpolation takes place between two mipmap-levels.

1.3 Parallel Processing

Parallel processing is a popular method to speed-up the computation of large problems. For this purpose, an algorithm which solves a problem needs to be *parallized*, in other words, the algorithms needs to be mapped to multiple CPUs or processing entities [24, 23]. While this mapping can be straight forward for some algorithms, it is challenging for others. The reason for the substantially varying parallelization difficulties is the necessary partitioning of a single problem into n smaller sub problems or tasks, where each of the smaller tasks is then processed by a different entity. Frequently, these individual entities need to share data or resources to solve the overall problem. Sometimes, one entity even relies on the results of another entity. All these dependencies require the synchronization of some or all entities which process the n tasks of the original problem. The associated synchronization mechanisms, such as signals, locks, mutexes (MUTual EXclusion), or semaphores are not for free; often a processing entity needs to wait for a result or a resource until another entity as completed its computation, or has released the used resource. During this time, the waiting entity does not process any task-relevant data, it is *idle*. The total of idle time and synchronization costs are called the *parallelization overhead*. Obviously, this parallelization overhead is directly related with the grade of dependencies among the sub problems; the more dependencies exist, the more synchronization effort is required, and vice versa. Unfortunately, an optimal sequential algorithm for a problem can have a poor parallel performance, due to the inherent dependencies of the parallel implementation. In these cases, a sub optimal algorithms needs to be chosen, which allows the partitioning into more independent tasks, and hence has a better parallel performance.

In the past, large massively-parallel systems, array or vector computer were dominating parallel computing. These systems – classified as SIMD systems[3] by Flynn [64] – consist of thousands of processing elements (PE) which are typically interconnected by a network of switches. This network propagates data from the host computer to the processing elements, or from one processing element to other processing elements. It is also frequently the bottleneck of SIMD-based parallel computing. SIMD systems require a very specific parallelization of an algorithm that is not always efficiently possible, which makes them difficult to use. However, SIMD computing has experienced a renaissance in modern processor design; most of the today available RISC and PC processors have a limited SIMD functionality where multiple operations can be performed at the same time.

Today, MIMD systems[4] are the predominant parallel computing approach. In contrast to the SIMD approach, a significant smaller number of standard processors are performing their individual instructions on their own data stream. There are basically four different implementations of MIMD sys-

[3] Single Instruction stream, Multiple Data steams; a large number of simple processing elements perform the same operation (single instruction) on different data (multiple data).

[4] Multiple Instruction streams, Multiple Data streams after Flynn's taxonomy.

tems; *massively parallel processing* systems (MPP) of homogeneous system components, *symmetric multiprocessing* systems (SMP), clusters of computer, and clusters of SMP systems. Currently, MPP and clustered-SMP systems are dominating the top 500 list of super-computing sites [209]. They combine high performance with a large number of processing entities.

MPP and SMP systems are tightly coupled systems, where the processing entities share more (SMP) or less (MPP) of the system components. In contrast, clustered systems (of computers or SMPs) are loosely coupled system which do not share components (except of the network) through all the system. Systems with shared memory (SMPs) are also referred to as *multiprocessors*, while systems which do not share memory are referred to as *multicomputers* [202]. In the following, we are focusing on SMP and clustered systems.

1.3.1 Symmetric Multiprocessing Systems

A *tightly coupled* SMP system shares components such as the system interconnect (bus or crossbar), system memory, or the system I/O. They are also classified with respects to their memory architecture. SMP systems are usually implementing a single hierarchy layer, where main memory access is equal for all processing entities (CPUs). This scheme is labelled as *uniform memory access* architecture (UMA). Unfortunately, only a limited number of processing entities can be supported by the used interconnects; the system bus capacity gets congested, if too many data and synchronization request are submitted, thus limiting the parallel performance. If a crossbar is used as interconnect, the hardware complexity (and hence the costs) of the crossbar are growing quadratically by the number of ports (number of possible connected components). For these reasons, many UMA systems are effectively limited to approximately 10 to 20 CPUs. To overcome these problems, additional hierarchy elements are introduced into the hierarchy. These additional hierarchy elements are altering the uniform memory access, since the main memory is now distributed through the various system components – hence these systems are also denoted as *distributed shared memory systems*. In particular, the distance of the memory modules to the processing entities now depends on their physical location within the system. The different access distances is reflected in the label for this *non-uniform memory access* architecture (NUMA).

Due to various caching techniques of modern CPUs, the memory visibility of the processing entities is not necessarily the same; the logical same data item might have a different content for one processing entity than for another entity. This problem aggravates on NUMA architectures, since more memory hierarchy layers are introduced. Some systems overcome this problems by implementing a *cache coherent* memory access protocol in hardware or in software. These systems are labelled as *cache coherent NUMA* systems (ccNUMA).

Synchronization and the exchange of data on (distributed) shared mem-

ory systems is handled via the shared memory itself, which is very effective, since no explicit messages need to be exchanged. Due to this shared memory, the data does not need to be replicated through all the processing entities, since the master copy itself can be shared. However, the different hierarchy elements introduce different memory latencies which can increase the synchronization and memory access costs. Note, that the increased memory access costs are also frequently hidden by the synchronization costs (see Section 3.3).

All parallel programming models can be used on shared memory systems, such as thread models (i.e., pthreads [34] or OpenMP [37]), or message passing libraries (i.e., MPI [67] or PVM [82]).

1.3.2 Cluster-based Systems

In contrast to tightly coupled systems are the loosely coupled systems which share less system components or non at all. Instead, all processing entities are connected via a (high-speed) network to exchange instructions and data. Loosely coupled systems composed of individual PCs are also labelled as *cluster of workstations* (CoW), *network of workstations* (NoW), or simply as *clusters*. Each individual entity is a SISD system[5] with its private memory and private I/O system. Synchronization is handled by the explicit exchange of messages via a network by a message passing library, or by other data exchange mechanisms (i.e., UNIX sockets or files).

The major attribute of thread-based systems – the implicit synchronization and exchange of information via the shared memory – is not available on cluster systems[6]. Therefore, popular threading concepts such as pthreads [34] or OpenMP [37] are not available. Applications use instead message passing libraries such as the *Message Passing Interface* (MPI) [67], or the *Parallel Virtual Machine* (PVM) [82].

1.3.3 Threading versus Message Passing

The main advantages of threading over message passing are the drastically smaller parallel programming overhead (no explicit exchange of messages, no buffer (size) checking, no buffer overflow) – hence the reduced costs for programming and testing – and the significantly faster communication through the shared memory (smaller physical distance between processing entities, no required data replication). However, the threading programming model is only available on shared memory systems – not on clusters –, while message passing is available on virtually all MIMD systems, including SMP and cluster-based systems.

[5] Single Instruction stream, Single Data stream after Flynn's taxonomy.

[6] Some distributed (non-shared) memory systems (MPP) provide an additional *virtual shared memory* layer which emulates shared memory. Access to the virtual shared memory on another processing entity is usually implemented through message passing. Therefore, these libraries only reduce the programming overhead of explicit message exchange, but they do not provide the same performance as physically shared memory systems.

SMP enables high performance and lower software implementation costs than clusters, due to the tightly coupled architecture, while the investments in computer hardware are usually significantly higher than for a cluster. The latter fact is usually the major reason why academic institutions frequently choose cluster-based systems. One of the downsides of a cluster is often ignored when estimating the costs of a parallel computer system. The maintenance costs of a cluster are usually significantly higher than the maintenance costs of a SMP system of similar size. This is mainly due to the fact that a cluster consists of many, potentially individual systems, while a SMP system is only one computer, with less periphery devices which requires maintenance.

Finally, some algorithms are difficult to map to systems which do not share the main memory (MPP and clustered systems). This problem arises if the data of a task cannot split up into smaller parts that fit into the main memory of a single entity. In these cases, a SMP system with a shared memory architecture is required.

1.4 Virtual Medicine

A major application field of large scale data visualization methods is virtual medicine. It is concerned with the simulation and visualization of biomedical processes and procedures using methods from computer graphics, namely from scientific visualization and virtual environments. Virtual medicine is almost as old as modern imaging methods. It started with two-dimensional image processing techniques to display the slice images from CT, or to enhance the relevant features in such images. Soon, methods of Computer-Aided-Geometric-Design (CAGD) had been introduced; Vannier et al. used Bezier-curves to provide planning information for complex cranial surgery [215]. Other approaches focused on the modeling of the biomechanical behavior of body parts. Waters modeled facial muscles for artificial muscle actors [223], which was later extended with Terzopoulos to a facial tissue model [224]. This model used a mechanical mass/spring model to simulate the behavior of tissue. Numerous variations and improvements were later introduced. Some of these approaches used spline models for better surface representations of 3D scanner data [104, 10], others focused on the simulation of the actual surgical procedure [121]. As a related model for the modeling of tissue, particle systems were introduced, which were originally applied to textile simulations [60] with a two-dimensional topology. In 1992, Pieper introduced Finite Element Methods (FEM) to provide more advanced modeling of human tissue [168]. However, the high computational costs and improvements in mass/spring system like models (i.e., particle systems) hindered their use. Currently, advanced mass/spring systems which enable more accurate tissue modeling [128, 59] are used next to modified FEM in this field. Another recent development is the use of haptic methods for the tactile feedback of the tissue reaction. Among other appli-

cations, it has been applied to training purposes for the palpation of tumors [55], for endoscopic simulators [131], or it has been used for navigation in volumetric datasets [15].

Other applications in virtual medicine focus on the representation and identification of tissue from scanned data. Various segmentation techniques are introduced which address the automatic or semi-automatic identification and selection of anatomical entities using diverse imaging modalities. Anatomical information systems like VOXEL-MAN [106, 107] introduced by Höhne et al. provide complex information on the topology and anatomy of the human body. Other systems use multimedia technology to present anatomical information [130, 200]. Segmentation approaches are also applied in the medical routine. However, automatic segmentation has up to now failed to accurately select specific organ systems. Therefore, semi-automatic and fully manual techniques are applied in the daily routine [83, 196].

In related areas, image processing methods are applied to enhance the display of histological information or to identify pathological structures [211]. Other approaches attempt the registration of identical organ structures in different data modalities for the fusion of these datasets. In particular, the fusion of structural anatomical data (i.e., CT, MRI, angiography, ultrasound) with functional data (functional MRI, Positron Emission Tomography (PET), Single Photon Emission Computer Tomography (SPECT)) is of interest in neuro-medicine, or neuro sciences in general.

The use of tracking systems in virtual environments has also become popular in the medical field. Often, these systems are used to provide an augmented representation, where rendered or other additional data is overlaid with the “real world”. The combined data is either provided via semi-translucent head-mounted displays [81, 179], or simply combined on a monitor or a projection screen [72]. Similar registration techniques using fiducial markers are employed in a suite for the planning of stereotactic frame neurosurgery. Serra et al. combined this planning suite with a 3D brain atlas, which is mapped to the individual patient data [184]. Other virtual environments are used for virtual drug design, where drugs are constructed virtually on a molecular base [217, 216]. Recently, methods from flow visualization were combined with immersive projection techniques to simulate and visualize the blood flow along blood vessels [66]. Of specific interest is the development of vortices and swirls along aneurysms, stents, or other features in a blood vessel.

In the daily routine of radiology departments, volumetric datasets are usually viewed in a cine mode, where image slices are browsed sequentially. However, the huge amount of data of recent scanners requires the use of three-dimensional techniques [94]. Direct volume rendering slowly emerges as an accepted three-dimensional method, since it does not require intermediate geometric representations. However, limited performance and complex parameter spaces still represent major barriers for the deployment in daily routine.

On of the most active fields in virtual medicine is the virtual-endoscopic exploration of organ systems, based on three-dimensional scanned data. A detailed discussion of this topic can be found in Chapter 5.

1.5 Outline

The remaining chapters of this dissertation are organized in two parts.

In Part I, we present new methods and techniques from three different research areas which are addressing the visualization of large models. In Chapter 2, we present the results from a new framework for the assessment and analysis of volume rendering methods. Specifically, we compare the four most popular direct volume rendering algorithms and the MarchingCubes algorithm as the most popular indirect volume rendering technique. Many acceleration methods for the efficient rendering of large data require a hierarchical organization of the data. Therefore, we present a new parallel scheme for the efficient construction of such a hierarchical organization in Chapter 3. In the last chapter of Part I, we introduce new techniques for the effective removal of not visible geometry (Chapter 4). These techniques address the full culling pipeline from effective pre-processing techniques to highly efficient geometry culling methods which enable a reduction by approximately 90% of the model geometry. Finally, we propose modifications to modern graphics subsystem which allow an even more efficient utilization of this accelerators for “richer pixel” operators.

In Part II, we put the developed techniques into the context of “real-life” applications dealing with large model data. In Chapter 5, we present VIVENDI a flexible system for the endoscopic exploration of medical data. In contrast to other virtual endoscopy system, VIVENDI allows the interactive and intuitive visualization of large medical data, without relying on application specific optimizations. In this chapter, we also critically discuss the benefits and drawbacks of virtual endoscopy in modern medicine. Applications of VIVENDI are discussed in Appendix B; specifically, we present applications to colonoscopy (Section B.1), ventriculoscopy (Sections B.2 and B.3), and angiography (Section B.4). Virtual medicine is not the only application field of the methods discussed in Part I. All introduced methods have also a large potential in fields such as computer-aided mechanical engineering (MCAD/CAE), visualization of data from scientific computing, and architectural walkthroughs. We sketch these other applications in Chapter 6. This dissertation concludes in Chapter 7 with a summary of the presented topics and achievements.

Part I

Visualization Techniques for Large Models

Chapter 2

Assessment of Volume Rendering Algorithms

A major question of using a volumetric representation is which rendering technique should be used for which objectives. However, little research is available which provide advice on this important topic. Tiede et al. compared gradient filters for ray casting and Marching Cubes in medical applications [208]. Williams and Uselton specified a framework for the specification of rendering parameter before any image comparison takes place [234]. Recently, Kwansik et al. contrasted a variety of direct volume rendering algorithm using artificial, modeled datasets to assess the rendering quality by means of root mean square error (RMS error) and other statistical metrics. Both Williams/Uselton and Kwansik et al. have focused mainly on quality. In contrast, we will discuss the quality and rendering performance of direct and indirect volume rendering (Section 2.2) [19] and the practical usage of the four most popular direct volume rendering approaches (Section 2.3) in the course of this chapter which will help researchers to choose an appropriate approach [212, 156].

2.1 Introduction

To avoid confusion of the used terminology, we briefly differentiate the various meanings of reconstruction. An overview can also be seen in Figure 2.1. A volume dataset consist of 3D data values, arranged on a 3D grid. These data values are reconstructed from projections which are the result of measurements from a 3D scanning process of an object (see also Appendix A). This reconstruction is also called *image* or *volume reconstruction*, or *reconstruction from projections*. Based on this volume datasets, two avenues can be taken; with indirect volume rendering (IVR), a polygonal isosurface is extracted and rendered by standard polygonal graphics hardware (see Section 1.2.3). This reconstruction is called *surface reconstruction*. In contrast, direct volume rendering (DVR) directly generates the image, without reconstructing an intermediate surface. In both cases, however, the material interfaces are in the focus of interest. Therefore, we call this reconstruction

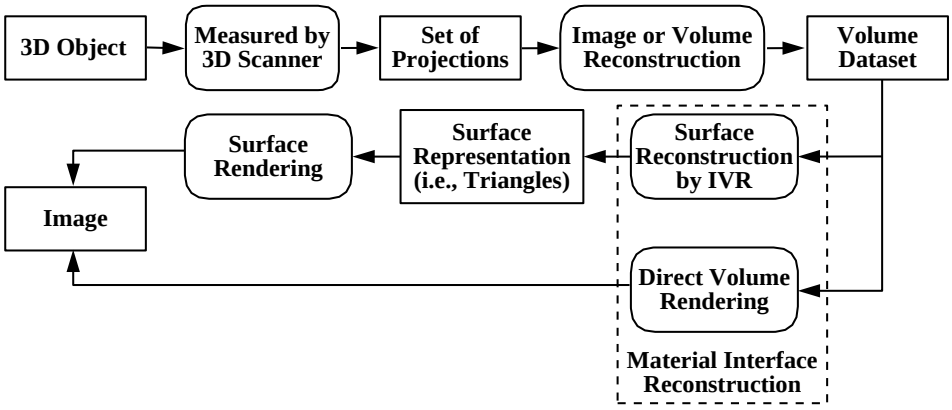


Figure 2.1: Reconstructions from object to image; IVR = Indirect Volume Rendering

the *material interface reconstruction* (see Fig. 2.1).

2.2 Voxels versus Polygons – A Practical Comparison

In our first comparison, we are comparing the benefits and drawbacks of using direct (DVR) or indirect volume rendering (IVR) methods. All experiments were performed on an SGI Octane, R10000 @ 250MHz with MXE graphics using similar viewing parameters [19].

We are using Marching Cubes (MC) as an IVR approach. Specifically, we are using the VTK implementation as being a gold standard for isosurface extraction, which visits and classifies all cells of the volume dataset [182]. Gradients are only computed for contributing cells. To avoid multiple vertices at the cell boundaries, a bucket-sort-like data-structure is used, which reduces the consumed memory significantly below the upper bound estimated later on in this section. The rendering time of the generated triangle-composed surface depends on the used graphics subsystem. Therefore, we only consider the surface reconstruction time of the MC algorithm. Depending on the rendering complexity of the extracted isosurfaces (number of polygons and opacity), the actual rendering with OpenInventor ranged from approximately 0.5 seconds to approximately five seconds for the MRI Head dataset, depending on the number of transparent isosurfaces (similar for the CT Lobster dataset).

For DVR, we use the ray casting (RC) approach with early ray termination, which stops sampling of the ray after the opacity has accumulated to 98%, which means that there is practically no further contribution to the pixel associated with this ray (oversampling is not used). Furthermore, no gradients are computed for non-contributing samples (opacity is zero). RC is a single pass algorithm where material interface reconstruction and rendering cannot be distinguished; therefore, the timings of both operations are included inherently in the measurements.

To compare the direct and indirect volume rendering approaches, we use

two different "real-life" datasets, each with eight bit voxel data (see also Table 2.1). As convention, we name voxels as *relevant voxels*, if they potentially have a contribution to the image, despite possible acceleration techniques (i.e., early ray termination, early splat termination). If the voxels, samples, or cells actually have a contribution, we name them *contributing voxels*, samples, or cells.

- CT lobster: CT scan of a lobster immersed in a cylinder of resin. We map the scalar value interval associated with the meat to opaque red, the shell scalar value interval to semi-transparent white, and the resin interval to highly transparent green (see Table 2.2).

- MRI head: MRI/3D CISS sequence (see Appendix A.4) of a human head. Head tissue and brain fluid-filled cavities of the MRI head are easy to classify, due to the good contrast of the MRI/3D CISS sequence. The color and opacity attributes for MC and RC are shown in Table 2.2, where we map the fluid-filled cavities to opaque, and the tissue to semi-transparent.

Dataset	Size/ Spacing	Relevant Voxels	Rendering Mode
CT lobster	$301 \times 324 \times 38$ $1.0, 1.0, 1.5$	2,872,082 (77.5%)	semi-transparent with opaque interior
MRI head	$258^2 \times 126$ $1.0, 1.0, 1.0$	1,861,928 (22.2%)	semi-transparent with opaque interior

Table 2.1: Benchmark datasets for direct vs. indirect volume rendering; note that the number of relevant voxels depends on the RC classification, not on the MC isovalue. This results in difference between relevant voxels and contributing cells in Table 2.4.

Material Labels	Material Interfaces	Color (RGB)	MC Opacity	RC Opacity
CT Lobster Dataset				
Resin (R)	2	0.0, 1.0, 0.0	0.13	0.02
Shell (S)	50	0.8, 0.8, 0.8	0.27	0.73
Meat (M)	130	1.0, 0.1, 0.1	1.0	1.0
MRI Head Dataset				
Tissue (T)	30	1.0, 1.0 ,1.0	0.02	0.02
Liquor (L)	70	1.0, 1.0, 1.0	1.0	1.0

Table 2.2: Overview of dataset material properties [19]: The material interfaces where used to extract the material surfaces by isovalues for the MC algorithm and as rising edges of the transfer functions of RC. However, the tissue (T) RC opacity transfer function is a peak at the threshold 30, while the rising ramp starts at 20, and the falling ramp ends at 40. We increased the MC opacity of the Resin for better visibility.

2.2.1 Visual Quality

The properties of RC and MC seem to be very obvious but as it can be seen in the next section, the impact to resource consumption can be quite severe. To enable a proper understanding of the differences, it is necessary to have a closer look at the features of both techniques with respect to volume graphics. In the following, we will discuss the two approaches and highlight their features in order to describe their strengths and weaknesses. Hereby we will try to answer the following questions:

- How good is the 3D understanding from generated images?
- How good can depth be understood from generated images?
- How accurate can smallest structures be displayed?

Display of Volumetric Information

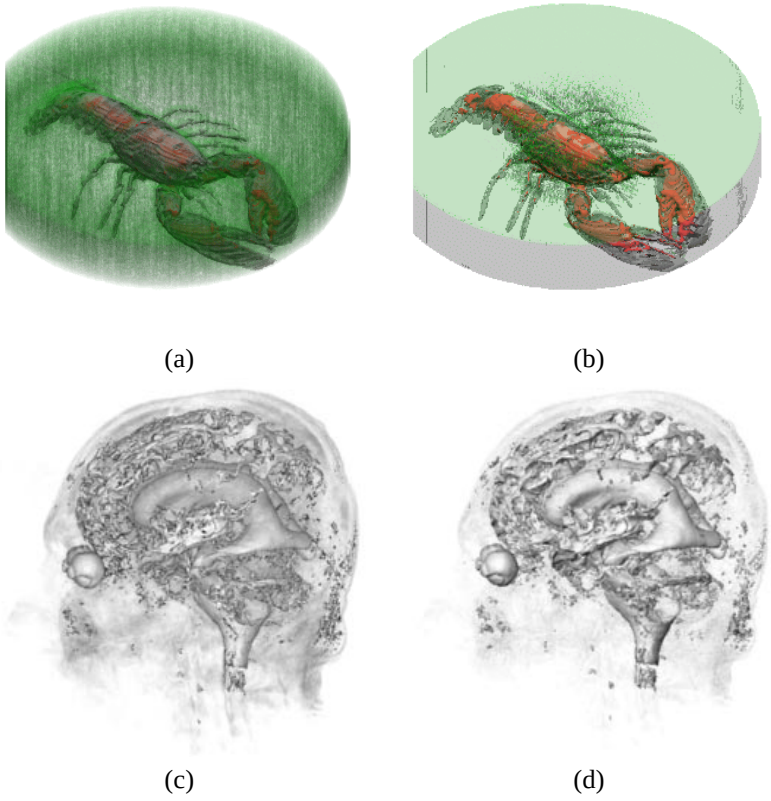


Figure 2.2: Volumetric display of CT lobster and MRI head [19]: (a) CT Lobster dataset rendered using RC, classifying resin, shell, and meat. (b) same as (a) using MC. (c) MRI head dataset rendered using RC, classifying tissue and liquor. (d) same as (c) using MC.

The MC algorithm extracts isosurfaces from volumetric data, hence real

thickness, or volumetric dimension of an object is difficult to represent[1]. This is inherent to the isosurface paradigm which is designed to extract infinite thin isosurfaces from volumetric data. As an example, images of objects with different thickness will always look alike. Therefore, the depth of highly transparent objects and the position of objects within highly transparent materials cannot be determined without rotating the object (see Figure 2.2b).

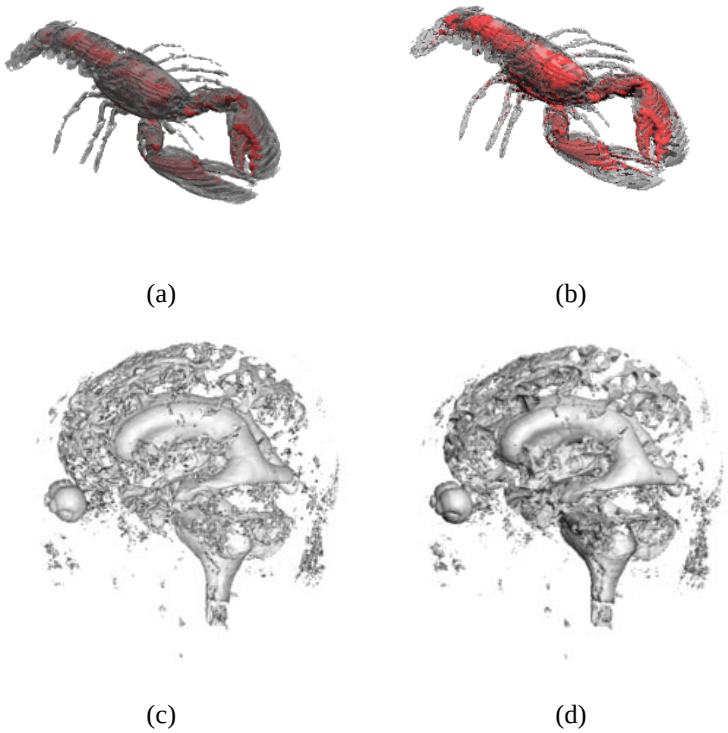
In contrast, RC samples volumetric data along rays and accumulates the obtained information according to the density emitter model which generates a noticeable depth perception by the accumulated color of the individual pixels. Cloudy, fuzzy, or gaseous objects like fog (or the resin of the CT lobster) that do not have a specific surface, can be displayed in a visual realistic way, which is not possible using MC. This effect can be seen in Figure 2.2a, where the resin is rendered as a cloudy (transparent) object but still, the position of the lobster within the resin can be conceived.

Display of Isosurfaces

When it comes to displaying real surfaces within volumetric data, the MC algorithm is capable of precisely extracting a surface which is represented by a certain isovalue, as long as objects do have surfaces. However, the surface of an object might not be specifiable by a single isosurface, since a material usually occupies an interval of isovalues. The number of required isosurfaces depends on the material interfaces of the volume dataset; i.e., the overall surface of the object resin – including the surface between lobster and resin and resin/air – can only be captured applying two isovalues. For MC in general, the number of isosurfaces which can be displayed, mainly depends on the memory limitations and on the polygonal render performance of the graphics subsystem. MC is capable of extracting smallest structures preserving their nature, as along as the structures can be specified by isovalue(s). With RC, the volumetric data is sampled along rays and the samples are interpreted using the transfer functions (post-shading), resulting in a quadruple $RGB\alpha$, and composited corresponding to the volume rendering line integral (Equation 1.2). In contrast to MC, RC is not dealing with specific isovalue(s), but rather with the classification of all generated samples using the transfer functions. The sampling rate is therefore very important to achieve sufficient image quality and to prevent aliasing artifacts. Structures can appear smaller than they actually are, if samples are computed just at the border of the structures and classified as *outside*. Their surface will not be detected before the next sample in the structure, classified as *inside*. This can be seen in Figure 2.3a where the legs and antennas of the RC rendered lobster are not as well defined than in Figure 2.3b (MC rendered).

Furthermore, very small structures which are just one voxel wide might

[1] Some approaches suggest the variation of the α-value according to thickness properties to represent volumetric differences.



(a) (b)

(c) (d)

Figure 2.3: Surface display of CT lobster and MRI head [19]: (a) CT lobster dataset rendered using RC, classifying shell and meat; (b) same as (a) using MC; (c) MRI head dataset rendered using RC, classifying liquor only; (d) same as (c) using MC.

be missed, depending on sampling and classification[2]. Therefore, classification and sampling are crucial parts of any DVR. Better results can be achieved using already segmented data – which also requires higher sampling rates, since segmentation introduces higher frequencies – or applying higher sampling rates. However, the images in this section were rendered using the original datasets without any oversampling.

2.2.2 Resource Consumption

Visualization of large datasets is in particular limited by space (memory) and time (rendering) consumption of the respective algorithms. Here we examine the specific memory and time consumption of MC and RC (see Tables 2.3 and 2.4).

[2]Techniques to adapt ray casting for accurately computing isosurfaces are available [165]. However, we only examine the "standard" approach of ray casting.

Memory Consumption

Memory consumption of DVR applications is limited to the actual representation of the volume. Some acceleration structures like octrees or BSP-trees do need significant additional amounts of memory. However, our investigated implementation of RC only uses the standard representation as a 3D array of voxel values, which limits the memory consumption to the number of voxels. If we consider eight bits per voxel, we use approximately n^3 bytes for the volume representation – where n is the number of voxels in each dimension in a cubic volume –, four times 256 bytes (eight bits per voxel) for the $RGB\alpha$ transfer functions, and $x * y * 4$ bytes for a 24+8 bits/pixel view-plane. For reasonably large volumes – which is the case for most applications –, the 3D voxel array will dominate the memory consumption.

The memory consumption of MC applications depends on the volume resolution as well. Similar to RC, we have a cubic space complexity, because potentially all cells of the volume might be contributing. In those cases, up to five triangles[3] with three vertices each are generated. Each vertex is described by six floats for position and normal vector. Therefore, we have an upper bound of 360 ($= 5 * 3 * 6 * 4$) bytes per cell[4].

Material	Time /[s]	#Visited Samples	#Contributing Samples
CT Lobster Dataset, 4,126,840 bytes used			
M	10.9	4,803,006	6,965
S	11.9	4,868,610	109,091
R	31.9	5,064,276	3,431,673
M, S	10.5	4,739,096	87,622
M, S, R	30.4	4,682,680	3,318,655
MRI Head Dataset, 8,654,344 bytes used			
T	24.1	8,453,628	1,193,236
L	11.4	6,890,357	18,480
T, L	17.5	6,890,357	841,856

Table 2.3: Resource consumption of ray casting [19]; R denotes resin, M denotes the meat of the lobster, S denotes the shell of the lobster, T denotes the tissue of the MRI Head, and L denotes the liquor of the MRI Head. Memory consumption is constant for all classifications of the same datasets.

Our complexity sketch shows that although the space complexity of RC and MC is both cubic, the actual memory consumption of MC can be significantly larger than RC (Tables 2.3 and 2.4). Most important is that the memory consumption of RC does not depend on the number of samples taken for the final image; the memory consumption – considering a fixed view-plane size – is constant. In contrast, the memory consumption of MC depends very much on the number of contributing cells. If this number is

[3] On average, about two triangles per contributing cell of the examined datasets were generated.

[4] VTK eliminates vertices which are generated more than once. Therefore, the actual number of vertices is smaller than the estimated upper bound. Table 2.4 shows only the memory used by the volume itself, the pointers of the generated triangles to the actual stored vertices, and these vertices themselves.

Material	Time /[s]	Memory /[bytes]	#Vertices	#Triangles	#Contributing Cells
CT Lobster Dataset, 3,585,300 visited cells, 3,705,912 voxels					
M	5.2	7,044,496	71,207	139,968	72,478
S	9.6	10,800,408	147,470	296,268	144,102
R	18.3	15,617,640	248,297	496,050	252,855
M, S	14.8	10,483,080	218,677	436,236	216,580
M, S, R	57.2	22,294,808	466,974	932,286	479,435
MRI Head Dataset, 8,256,125 visited cells, 8,387,064 voxels					
T	120.2	70,978,920	1,318,170	2,579,648	1,374,041
L	23.8	21,899,520	285,513	555,012	293,528
T, L	165.3	84,491,376	1,603,683	3,134,660	1,667,569

Table 2.4: Resource consumption of Marching Cubes [19]; R denotes resin, M denotes the meat of the lobster, S denotes the shell of the lobster, T denotes the tissue of the MRI Head, and L denotes the liquor of the MRI Head. Memory consumption represents the actual used memory.

high, the number of generated triangles (and their respective vertices) is high as well. Therefore, the size of the memory needed is directly related to the number of contributing cells.

Time Consumption

In the previous paragraphs, we showed that memory consumption of RC applications is dependent on the resolution of the volume dataset, but not on the number of samples taken. However, the number of samples determines material interface reconstruction and rendering time of these applications. In order to provide some operation quantification, we estimated the number of operations for both approaches (see Tab. 2.5). Additional to the mentioned operations are numerous branching (if) and loop (for) constructs which are not considered in this time complexity sketch. Generally, sampling along the rays first calculates the sample (one trilinear interpolation). Thereafter, this sample is classified (one access to the opacity function table). If the sample has a contribution to the final image (its opacity is not equal zero), its classification is completed (three accesses in the transfer function tables) and it is gradient shaded – which requires a gradient approximation using central differences (including 48 accesses to voxels which surround the sample[5]), and one trilinear interpolation of a vector. Finally, the sample is composited with the already accumulated samples.

The number of rays which are cast through the volume depends on the pixel resolution of the view-plane. In order to catch all features in the volume datasets, the sampling theorems requires that we need to cast at least four rays (2^2) through the voxels visible from the view point. Assuming a volume with a n^3 voxel dimension, this gives a quadratic complexity of the number of rays ($O(n^2)$), since we need to cast sufficient rays to cover a full

[5] Although all voxel accesses are only in a 32-voxel-neighborhood, no optimizations are used.

High-Level Operation	Mult	Add	Others
Trilinear Interp.	7	14	
Linear Vertex Interp.	7	14	
Gradient Approx.	18	12	
Transfer/Voxel Access	1	1	
Cell Access (eight voxels)	8	20	
Normalization (per vertex)	6	2	1 sqrt
Compositing	7	4	
Summary			
MC Contributing Cell	362/ 236	386/ 224	15/6 sqrt's 8 bit op's
MC Non-contrib. Cell	8	20	8 bit op's
RC Contributing Sample	169	132	
RC Non-contrib. Sample	8	15	

Table 2.5: Number of multiplications (mult) and additions (add) for each high-level operation, and for the respective cell/sample types. For the number of operations for MC Contributing Cells, we consider the worst case of five triangles per contributing cells (first value) and the measured average case of two triangles per contributing cell (second value) [19].

side of the volume, which is composed of $n \times n$ voxels. Furthermore, we sample through the volume. Again, taking the sampling theorem into account, we should sample with a step size of at least half the smallest spacing of the volume. An even higher sampling rate is required, if a segmentation of the volume introduces high frequencies. Overall, we have a view-plane resolution which depends on the volume resolution (n^2), and we have the sampling distance which depends on the volume resolution (n). Therefore, we can approximate the time complexity as cubic.

For the material interface reconstruction and rendering times of RC (see Table 2.3), the time costs do not simply depend on the number of "visited samples", which varies only by 8% (CT Lobster) up to 19% (MRI head), depending on early ray termination. More important are the samples which contribute to the final image; these are the samples with a non-zero opacity. These samples cause most of the computational costs (gradient-shading and compositing) of RC. If the opacity of large areas of the volume is low (i.e., resin), there will be a good chance that a large share of the area will contain contributing samples. Therefore, the total number of these samples will be higher, resulting not in an early but late ray termination and consequently many shading operations (see experiment R and M,S,R in Table 2.3).

The time consumption of the surface reconstruction process using the MC algorithm depends on the number of cells. As mentioned earlier, each cell is visited and classified, as inside, outside, or intersecting with the iso-surface, which requires a cell lookup and eight logical bit operations. For all the contributing cells, we need one multiplication to calculate the positions with respects to the grid spacing[6]. The cell gradients for shading

[6]One multiplication is needed for each slice, for each scanline within a slice, and for each cell of a scanline, which corresponds to approximately one multiplication for each cell.

are determined similar to RC by using eight vector central differences (no optimizations used), which includes 48 voxel accesses. Finally, up to five triangles – depending on the classification case – are generated. Each vertex of these triangles involves a linear vertex interpolation of position and normal, and one vector normalization. Note that especially the final estimate is only an upper bound which is usually never met in practice. In our measurements, on average two triangles per contributing cell were generated (see Table 2.4).

The complexity is determined by the number of cells of the volume dataset, because all cells are visited by the VTK implementation of the MC algorithm (thus the similarity between voxels and the number of visited cells). Consequently, the complexity is cubic, similar to the space complexity of RC. For each cell, we can give a constant upper bound of five triangles (and 15 vertices) for the costs of gradient, position and normal computation.

Table 2.4 shows the timing of the surface reconstruction process using the MC algorithm. The VTK implementation of MC checks if a vertex was already generated by a previous triangle of the current or a neighboring cell. This checking overhead consumes a significant share of the overall time costs. For the extraction of the isosurface of the MRI Head dataset, the overhead accounts for approximately 50% (isosurface of L) up to 68% (isosurfaces T and L) of the total extraction time. Furthermore, this overhead grows faster than the actual number of vertices. Therefore, the time spent for the generation of multiple isosurfaces is higher than the sum of each individual isosurface. The surface reconstruction times of the different isosurfaces does vary much, due to the significantly larger numbers of contributing cells, hence the larger numbers of generated triangles (see Table 2.4).

Marching Cubes Versus Ray Casting

Generally, the amount of memory need by MC – dominated by the triangles and vertices – is significantly larger than the constant amount of memory for RC – dominated by the volume itself.

For the time costs, MC is only faster for classified datasets where the number of contributing primitives (cells vs. samples) is smaller than for RC, which is only the case for the CT Lobster. Furthermore, RC is more affected by a large number of visited (but not contributing) samples than MC for visited cells, since sample (without shading) is more expensive than the cheap MC classification.

Overall, MC is very efficient for single isosurfaces with a low number of contributing cells (and triangles), while RC is faster for large volume datasets. This is in particular true for opaque structures, which can be exploited for early ray termination. However, it is important to note that rendering of MC generated polygonal isosurfaces is much faster and the surface reconstruction can be viewed as a pre-processing step. Furthermore, the polygonal isosurface is a continuous representation, and its rendering

usually is not fill-rate limited. Therefore, a larger viewport can be used at almost no additional rendering costs. This is different using RC. If a larger view-plane is used, more rays are cast through the volume. These additional rays significantly increase the material interface reconstruction and rendering time.

2.3 Comparing Direct Volume Rendering Approaches

Our next comparison focuses on four DVR approaches which are used in many applications [156]. These four algorithms, ray casting (RC), splatting (SP), shear-warp (SW), and graphics hardware-assisted 3D texture mapping (TEX) were already introduced in Section 1.2. The assessment is based on a set of "real-life" datasets from physical simulations, from medical scanners, or computed functions [212]. In contrast to the datasets of the previous comparison, all datasets provide an isotropic sampling (see also Table 2.6):

- Fuel injection: Physical simulation of diesel being injected into a combustion chamber filled with air. The presence of air in the dataset is represented by the density values; the higher the density value, the lower the presence of air. The dataset is a semi-transparent, but compact representation that requires many samples to be taken for each pixel.

- Neghip: Physical simulation of a high potential protein representing the electron probability around each atom (blue is high, green is medium, and red is low). This dataset has some dispersed elements.

- Skull: Rotational biplane X-ray scan (see Appendix A.3) of a human head (except top part of skull). Bone and teeth structures are reconstructed (from the projections) with good contrast. There is also a contrast agent enhanced segment of the left internal carotid artery.

- Blood vessel: Rotational biplane X-ray (rotational angiography, see Appendix A.3)) scan of a human head with a focus on the right hemisphere, where a contrast agent has been injected into the blood to capture the blood vessel to visualize the main feature of the dataset, an aneurysm of the internal carotid artery, between the carotid T-junction and the carotid siphon.

- Shockwave: Simulation of an unsteady interaction of a planar shockwave with a randomly-perturbed contact discontinuity, rendered with a highly translucent transfer function. All voxels potentially contribute to the display.

- Marschner-Lobb: High frequency test dataset, rendered as an iso-surface. This dataset contains very high frequencies, where 99.8% of the sinusoids are right below the Nyquist frequency.

Dataset	Size	Relevant Voxels	Rendering Mode
Fuel Injection	64^3	13,731 (5.2%)	semi-transparent with opaque interior
Neghip	64^3	121,586 (46.4%)	moderately semi-transparent
Skull	256^3	1,172,924 (7.0%)	opaque isosurface
Blood Vessel	256^3	76,929 (0.5%)	opaque isosurface
Shockwave	$64^2 \times 512$	1,206,828 (57.6%)	fully semi-transparent
Marschner-Lobb	41^3	34,387 (49.9%)	opaque isosurface

Table 2.6: Benchmark datasets for direct volume rendering comparison; all spacing is $1 : 1 : 1$ [156]

Both rendering quality and expense is likely to be dependent on viewpoint, magnification, as well as image size. To study these effects, we have rendered the datasets at four to five magnification levels each and into six image resolutions: 64^2, 128^2, 256^2, 512^2, 1024^2, and 2048^2 pixels[7]. To get statistically valid results we have also rendered a series of 24 images at random view-points over an enclosing sphere. No frame-to-frame coherence in either data access or rendering was exploited. A specific storage order of the volumes was not allowed (VolPack with its three encoded volumes is an exception). Diagonal views are especially challenging for the shear-warp algorithm in terms of quality (the ray step size is significantly below Nyquist), and for the splatting algorithm in terms of rendering time (due to the required re-alignment of the voxel slabs, see also Section 1.2). Ray casting may incur some caching delays when the volume is accessed out of stride, while the texture mapping hardware will most likely be least sensitive to the change of viewing directions.

2.3.1 Assessment of Image Quality

It is difficult to evaluate rendering quality in a quantitative manner. Often, researchers simply put images of competing algorithms side by side, appointing the human visual system (HVS) to be the judge, whereas it is well known that the HVS is less sensitive to some errors (stochastic noise) and more to others (regular patterns). In particular images with larger numerical errors (i.e., RMS), are sometimes judged as worse by a human observer than images with lower numerical errors [207]. It seems that the visual comparison is more appropriate than the numerical, since after all images are produced for a human observer and not for error functions. In that respect,

[7] The different image resolutions represent different requirements of available display media: web-based rendering, head-mounted displays, computer screens, high-definition screens, CAVE projection walls, and PowerWalls.

an error model that involves the HVS characteristics [77] would be more appropriate than a purely numerical one. But nevertheless, to perform such a comparison, we still need the volume rendered image computed by the analytical integration of the volume via Equation 1.1. As was pointed out by Max [147], analytical integration can be done when assuming that $C(s)$ and $\mu(s)$ are piecewise linear. This is, however, somewhat restrictive to our transfer functions. Hence we decided to employ visual quality assessment only.

Apart from the real-life datasets, we also chose a particularly challenging dataset for visual quality assessment; the Marschner-Lobb function [146]. This three-dimensional function is made of a combination of sinusoids and contains very high frequencies. 99.8% of these frequencies are below the Nyquist rate when sampled into a 41^3 lattice. It is extremely sensitive to filter and sampling inaccuracies and has been used for (material interface) reconstruction error evaluations.

2.3.2 Experiments

The four volume rendering approaches were devised with different objectives on rendering performance and image quality. While shear-warp and 3D texture mapping focus on frame-rates at the expense of rendering quality, image-aligned splatting and ray casting have been devised to achieve images of high quality, not to be compromised by the employed acceleration strategies. To account for this, we have divided the four renderers into two groups of two renderers each:

- High-performance volume renderers: Shear-warp (SW) and 3D texture mapping (TEX). These renderers use the pre-shading model, where SW uses pre-multiplied colors and opacities [236]. Due to the limited precision of the alpha channel of the MXE framebuffer (eight bits), the visual artifacts of opacity pre-multiplied colors are actually worse than with the original values. Therefore, we use the non-multiplied colors, taking the color bleeding artifacts.

- High-quality volume renderers: Splatting (SP) and ray casting (RC). These renderers use the post-shading model.

All presented results were generated on an SGI Octane (R10000 CPU @ 250MHz) with 250 MB main memory and MXE graphics with 4 MB of texture memory. The graphics hardware was only used by the TEX approach. Figures 2.6 and 2.7 show representative still frames of the six datasets that we rendered with the four volume rendering algorithms. Figure 2.4 relates frame times to magnification factors. The icon images next to the graphs indicate the level of magnification as well as the view-point (the icon images were rendered with RC). Figure 2.5 shows how image size affects rendering time of the screen filling shots (a), (b), and (d) of Figure 2.6, and (a), and (c) of Figure 2.7. Finally, Table 2.7 lists the average frame time for the 24 ran-

domly generated views. For these random views, we set the magnification factors such that the object just fills the screen.

2.3.3 Visual Quality

In Figures 2.6 and 2.7, we observe that the image quality achieved with TEX shows severe color-bleeding artifacts, due to the non-opacity weighted colors [236], as well as staircasing. The latter artifacts can be reduced by increasing the number of slices.

VolPack shear-warp performs much better, with a quality similar to ray casting and splatting whenever the resolution of the image matches the resolution of the respective base-plane area (Figures 2.6d, 2.7a, and c). For the other images, the rendered base-plane image was of lower resolution than the screen image and had to be magnified using bilinear interpolation in the warping step. This leads to excessive blurring, especially for the Marschner-Lobb dataset with a magnification factor of six (Figure 2.7e). A more fundamental drawback can be observed in the 45 degrees neghip view in Figure 2.6c, where – in addition to the blurring – significant aliasing in the form of staircasing is present. This is due to the insufficient ray sampling rate which is less than 1.0, and can be disturbing in animated viewing of some datasets.

The Marschner-Lobb dataset renderings for RC and SP (see Fig. 2.7e) demonstrate the differences of point sampling (RC) and sample averaging (SP). While ray casting's point sampling misses some detail of the function at the crests of the sinusoidal waves (insufficient sampling according to Nyquist), splatting averages across the waves and renders them as blobby rims. For the other datasets the averaging effect is more subtle, but still visible. For example, ray casting renders the skull and the magnified blood vessels with somewhat crisper detail than splatting does, but suffers from aliasing artifacts, if the sampling rate is not chosen appropriately. However, the quality is quite comparable, for all practical purposes.

But even though the quality is similar, there are still subtle differences in the images rendered by each algorithm. These differences are due to the different convolution kernels (bilinear for SW, trilinear for TEX and RC, Gaussian for SP) which combine transfer functions and sample values.

2.3.4 Time Consumption

Shear-warp versus 3D Texture Mapping

From Table 2.7, we observe that the frame times for both TEX and SW are always substantially faster than those of RC and SP. Both TEX and SW consistently achieve frame times in the sub-second range. With TEX, all data is always sliced and composited by the graphics hardware in brute force. SW's frame times are a function of the number of relevant voxels and opaqueness. It takes roughly three times longer to render the translucent shockwave dataset than the opaque skull, although both have about the

	Fuel Injection	Neghip	Skull	Blood Vessel	Shockwave
Ray Casting	4.96	8.15	7.78	12.31	3.02
Splatting	1.41	7.35	11.09	1.87	21.77
Shear-warp	0.09	0.24	0.27	0.09	0.91
Texture Mapping	0.06	0.04	0.7	0.7	0.14

Table 2.7: Average frame time (seconds) for 24 random views onto the five datasets [156]. The image size was 256^2, and the object was viewed in an image filling shot. Note that TEX is slower than SW for large datasets (skull, blood vessel) which exceed the 4 MB of texture memory of the used SGI Octane/MXE and requires texture swaps. (Marschner-Lobb is only used for quality assessment.)

same number of relevant voxels. This effect is due to VolPack's early ray termination which terminates sampling along the base-plane rays, once the opacity threshold is reached.

An interesting case is the blood vessel dataset, where SW is more than seven times faster than TEX (similar the skull dataset), while otherwise slower or of similar performance. This is due to the small number of relevant voxels (0.5%, see Table 2.6), in contrast to the large size of the brute-force rendered TEX volume which causes expensive texture swap operations. For similar reasons TEX is relatively insensitive to large image resolutions, since the usually fill-limited texturing operation is hidden by texture swaps. Magnifications do neither require a different 3D texture access, nor do the number of rendered pixels change. This is consistent with our TEX measurement which did not expose any magnification sensitivity. The image resolution insensitive SW only bilinearly interpolates the base-plane image to the larger viewport which also takes little time. For similar reason, SW is also insensitive to magnifications, since the base-plane size does not change.

Ray Casting versus Splatting

Splatting is an object-space oriented approach which processes the relevant voxels in object order along the voxel slabs. Therefore, if the number of relevant voxels is small (i.e., blood vessel), the number of footprints to be rasterized is small too. We can also see in Table 2.7 that the splatting of the opaque skull dataset is twice as fast as the transparent shockwave dataset, although the number of relevant voxels is almost the same. This is mainly due to the successful use of early splat elimination, since the quickly saturating opacity buffer areas of the skull occlude many of the relevant voxels, while the transparent shockwave does not allow for culling of relevant voxels.

In contrast, a high number of irrelevant voxels causes a high number of non-contributing RC samples, which dominate the rendering time, as observed with the blood vessel and fuel injection datasets. In both cases most of the rays are cast through non-relevant voxels, wasting rendering time. This is different with the skull dataset, where early ray termination skips the backward empty space. The associated costs of early ray termination

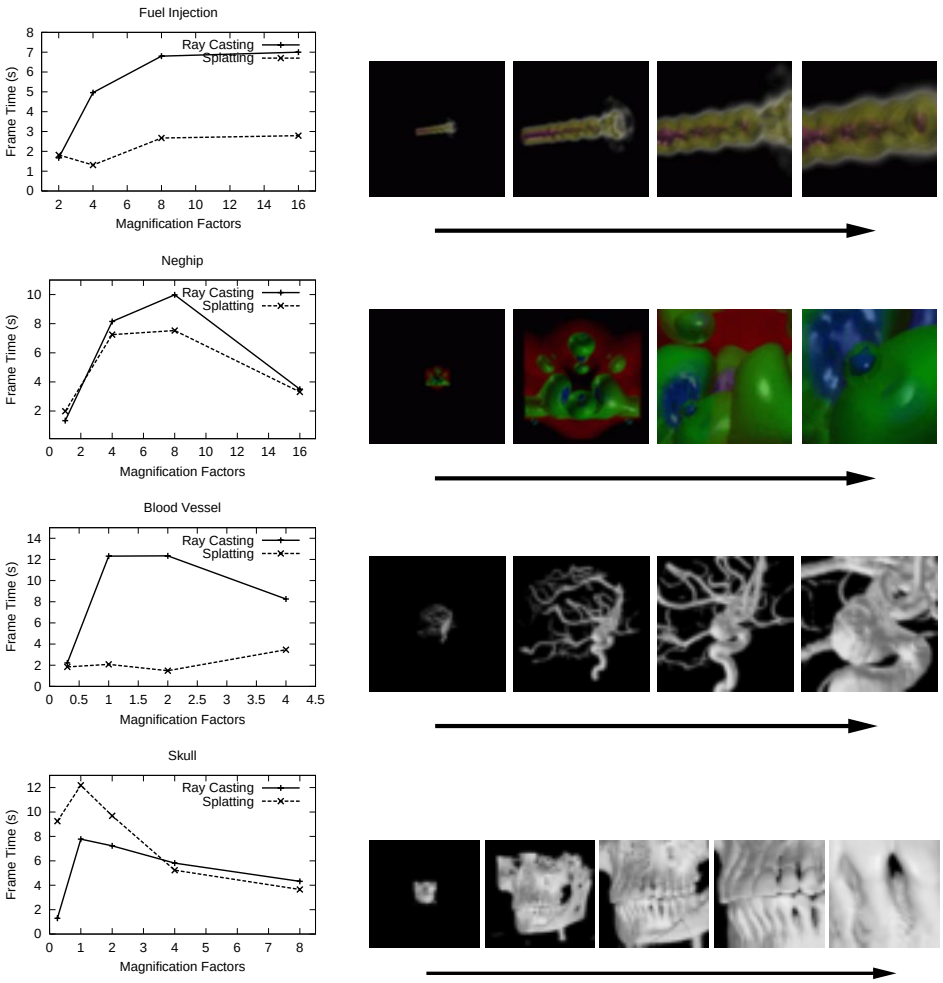


Figure 2.4: Frame time versus magnification; the timings for TEX and SW are not included, since we noticed almost no dependencies from magnification [156].

are low, since it is a simple comparison of the accumulated α-value with a specified threshold. In contrast, splatting's early splat elimination has high associated costs, thus splatting takes considerably longer to render the skull dataset. The shockwave dataset, where the low opacity of all voxels prevents both early ray termination and early splat elimination, exposes the differences in cost for trilinear interpolation versus footprint mapping. Since the rendering time for ray casting is almost seven times lower than the time for splatting (see Table 2.7), we conclude that the mapping of footprint kernels is costlier than trilinear interpolation, at least at moderate screen sizes.

The magnification plots of Figure 2.4 indicate that it is better to rasterize a small number of large SP footprints, even when they fill the entire screen, than to rasterize and transform a large number of small footprints. Good examples for this are the skull and the neghip datasets where at large magnification the number of splats that are not culled by view-frustum culling is small, but their footprints are large. In contrast, the number of rays cast

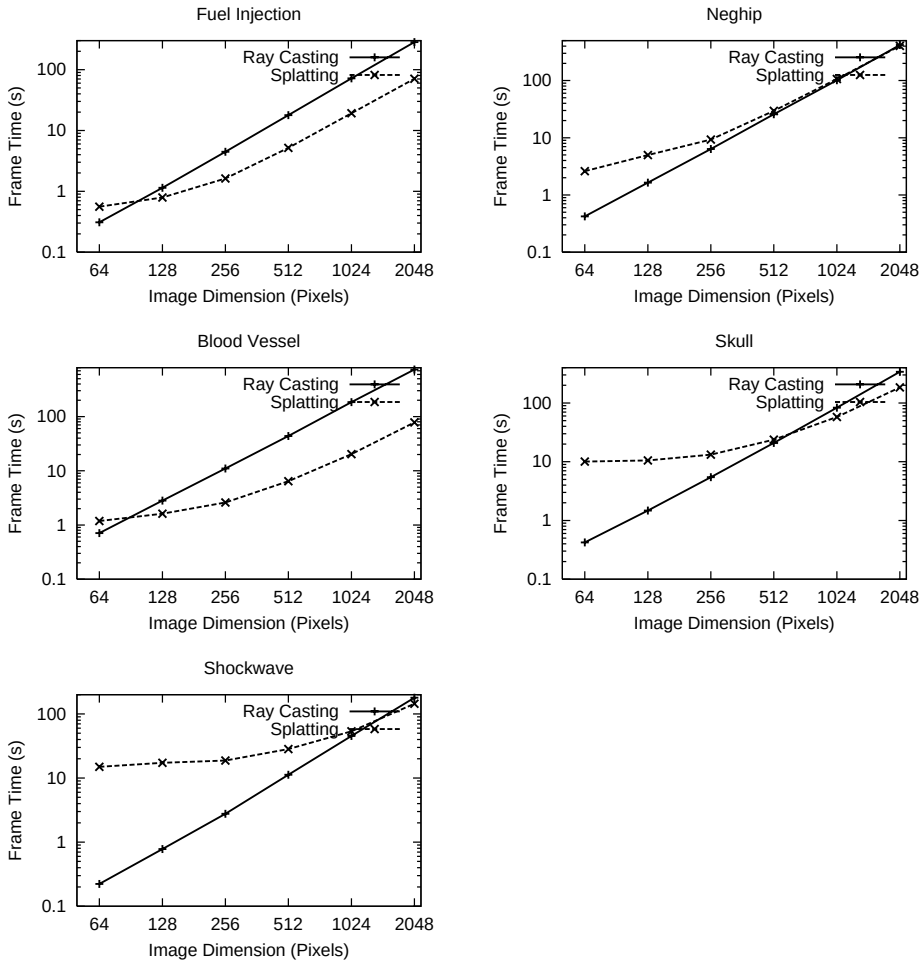


Figure 2.5: Frame time versus viewport size; note that we use a logarithmic scale. The timings for TEX and SW are not included, since we noticed almost no dependencies from the viewport size [156].

with RC is constant, since the image resolution has not changed. However, the magnified views enable more early ray termination (all opaque datasets), which reduces the required rendering time significantly.

Figure 2.5 illustrates the relationship of viewport size versus rendering time. We observe that for both sparse datasets, the fuel injection and the blood vessel datasets the differences between splatting and ray casting become more pronounced as the screen size increases, which shows the advantages of the object-space oriented splatting. However, the shockwave and the skull dataset exhibit a reversal of the rendering cost relationships at larger screen sizes, when SP becomes faster than RC. This is due to the fact that the time for transforming the voxels and for setting up the footprint rasterization (i.e., mapping the voxel center and computing the footprints screen extent) does not grow with the screen size, although the cost for footprint rasterization and opacity buffer maintenance does. Ob-

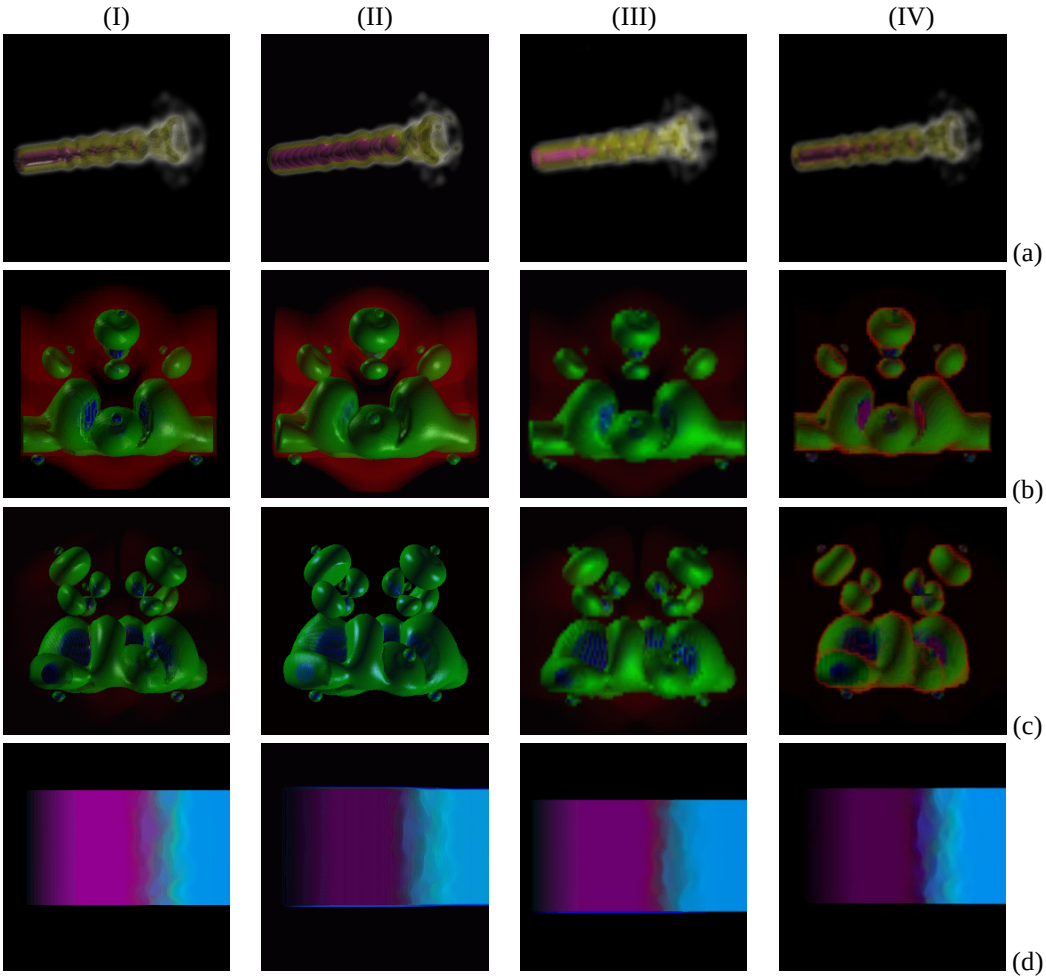


Figure 2.6: Dataset Overview I – Columns (I) to (IV) show the images from Ray Casting, Splatting, Shear-warp, and 3D Texture Mapping; Rows (a) to (d) show the images for the datasets fuel injection, neghip, neghip rotated 45^o, and shockwave. Magnification factors are 5 for (a)-(c) and 1 for (d) [156].

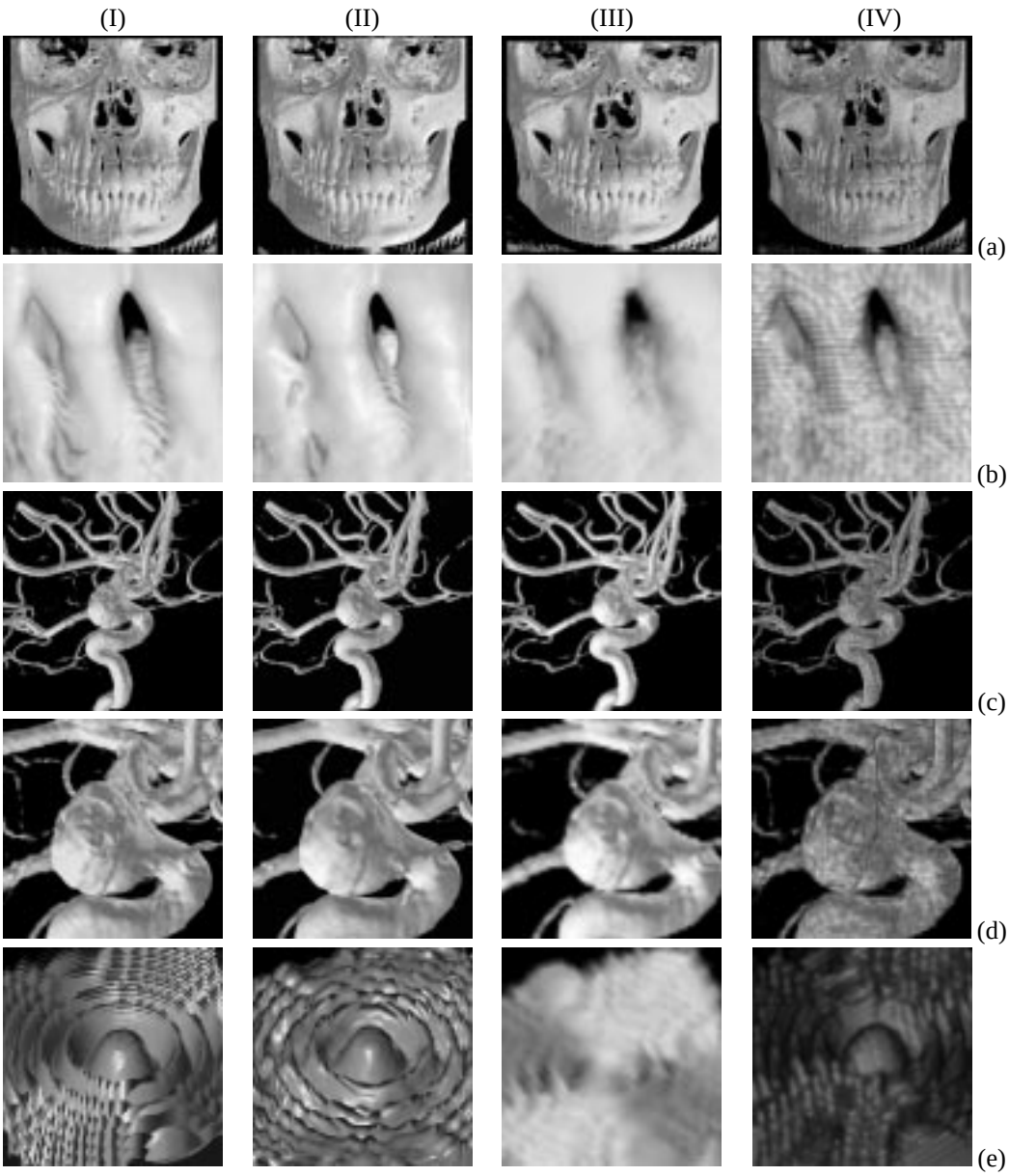


Figure 2.7: Dataset Overview II – Columns (I) to (IV) show the images from Ray Casting, Splatting, Shear-warp, and 3D Texture Mapping; Rows (a) to (e) show the images for the datasets skull (magnification 2), skull (magnification 8), blood vessel (no magnification), blood vessel (magnification 2), and Marschner-Lobb, (magnification 6) [156].

viously, the screen size-dependent costs of splatting increase not as fast as size-dependent number of trilinear interpolations, resulting in higher total RC rendering costs beyond a certain screen size threshold.

2.4 Summary

No general recommendation which approach to choose can be made. Each one has its individual properties. However, our results of the first part show that direct volume rendering (DVR) techniques can have significant resource consumption advantages compared to indirect volume rendering techniques. In particular the memory consumption of Marching Cubes can quickly exceed the available resources. Furthermore, DVR techniques can also be faster for large datasets as shown for the MRI Head dataset of the first measurement.

For the second part, two of the DVR techniques were exposed as high-performance rendering approaches on the expense of quality, while the two other techniques focus on rendering quality. Shear-warp (SW) and 3D texture mapping (TEX) provided similar rendering performance with similar rendering quality. However, both techniques exposed different weaknesses; the base-plane rendering approach of SW introduces heavy blurring, when the size of the viewport increases the size of the respective area of the base-plane, thus violating the sampling theorem. TEX in contrast suffered from a limited bit accuracy of the α-channel of the framebuffer.

Splatting (SP) and Ray Casting (RC) provided similar excellent visual quality at relatively high time costs. The Gaussian kernel of SP provides good anti-aliasing, but it also tends to blur fine detail. RC in contrast can exhibit alias problems, if the sampling rate is not chosen appropriately by obeying the Nyquist rate. The rendering time of SP basically depends on the number of relevant voxels, which in turn depends on the classification of the dataset. In contrast, the rendering performance of RC correlates with the number of samples, which depends on viewport resolution and dataset classification.

Chapter 3

Parallel Construction of Scene Hierarchies

Hierarchical data-structures play an important role in computer graphics to reduce the complexity of common problems. Specifically, multi-resolution methods are used to reduce the polygon count of large models [79], for culling of not visible geometry [92], for the reduction of the light interaction between different parts of a scene [95], for filtering [65], and so forth. Among the most popular spatial multi-resolution representations are recursive tree structures like quadtrees and octrees [180], k-D-trees [33], and BSP-trees [76]. In most cases, the construction of recursive tree structures is performed as a pre-processing step. If this step is required frequently, a parallelization becomes quickly worthwhile. Furthermore, changes in the transfer functions, color tables, or of the isovalues frequently require a fast reconstructions (or re-evaluation) of the data-structures [93, 139, 231]. In occlusion culling applications, animated objects cause a partial reconstruction of the scene representation [199].

In this chapter, we focus on the parallel construction of a hierarchical representation of volume datasets. In particular, we present a method for the parallel, asynchronous, and balanced construction of recursive tree structures [14, 29, 13]. Specifically, we apply the method on the construction of octrees as a typical candidate for a recursive tree structure. However, all techniques are also applicable to other recursive tree structures as well. Due to the tightly coupled nature of the task, we chose an implementation on shared memory systems (see Section 1.3 for a discussion of the possible approaches) which represent the current state of the art of parallel computing architectures.

In the next sections, we briefly review octrees and some of their use on computer graphics (Section 3.1). In Section 3.2, we introduce our new parallel octree construction scheme and parallel isosurface extraction, followed by a discussion of three memory allocation schemes in Section 3.3 on different SMP computer architectures. Finally, we summarize these techniques in Section 3.4.

3.1 Introduction

An octree is a hierarchical spatial data-structure to represent three-dimensional volumetric data[1] at different levels of details [180]. Starting with the superblock – representing the whole dataset – each octant is decomposed into eight child blocks. Each of these child blocks has half the size of the parent in each dimension (Fig. 3.1). This decomposition is performed until the lowest level is reached, where each block represents eight volumetric sample values (voxels). These bottom level blocks are called cells and they are identical to Marching Cubes cells, or cells in structured datasets with a rectilinear grid topology.

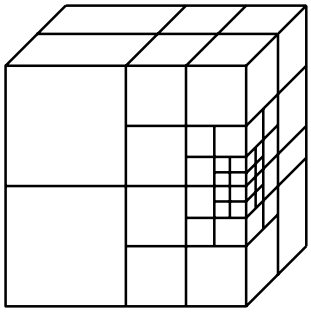


Figure 3.1: Octree of a volume dataset [29]

Due to the decomposition, the size of each octant is a power of two. Unfortunately, datasets usually do not have the exact size for this scheme (see Table 3.1). Therefore, some octants are "empty" – they do not intersect with the dataset – according to the alignment of the dataset within the octree. These "full octrees" result in wasted space allocated for empty octants. In order to save this space, we use a branch on demand octree (BONO) [231]. The BONO approach enumerates only the octants – and their children – that are not empty. Furthermore, BONO always tries to maximize the size of at least one octant by aligning the dataset to the lower-left-front corner of the octree [231].

Octrees are used in several applications to provide a multi-resolution representation. Laur and Hanrahan presented an octree-based scheme for hierarchical splatting [136]. Splats of different size and shape were used, according to the standard deviation of the color values of the different octree blocks. Grosso et al. presented a parallel implementation of this algorithm [93]. In their approach, a static parallelization of the octree construction was used, where up to eight threads were processing up to eight child blocks of the superblock. Greene et al. used an octree and an image pyramid for visibility queries in large polygonal environments [92]. Shekkar et al. used an octree representation of a volumetric dataset to generate a block-oriented polygon reduction scheme of its isosurface [191]. Levoy presented

[1] Other approaches also organize geometric data in an octree.

an approach to accelerate ray casting by using octrees [139], where coherent (non-contributing) data can be rapidly skipped. A hierarchical approach for cell-projection based volume rendering using a k-D-tree was proposed by Wilhelms et al. [232]. Wittenbrink and Kim presented an octree-based approach to accelerate permutation warping-based volume rendering where subvolumes are decomposed using the octree [235]. Swift et al. combined quadtree slices of a data volume to an octree [201]. This process starts at leaf level of the quadtree and continues up to the root, where the combination of two slices can be performed in parallel. However, details on the actual parallel implementation are sparse. Kela and Wynn proposed a parallel construction scheme for quad- and octrees, but the push-up of information during that construction process was not supported [122].

We are using a parallel implementation of the BONO approach by Wilhelms and van Gelder [231]. By storing the minimum and maximum values of the voxels at each block of the octree, the blocks which do not contain the isovalue in their minimum/maximum interval, can be rapidly skipped. After selecting all contributing voxels ("surface voxels/cells") of these blocks, the isosurface is generated.

Note that octrees only depend on a rectilinear grid topology (structured grids), not on a rectilinear grid geometry. Therefore, this approach is suited – and implemented – for cartesian and non-uniform rectilinear grids, as well as curvilinear grids [13].

3.2 Parallel Octree Construction and Isosurface Extraction

In general, recursive tree structures are constructed in two stages; a split-down of a parent into several children, and a push-up of the results of the children back to the parent, i.e. the standard deviation, or – in our case – the minimum and maximum voxel values.

The parallelization of a recursive split-down is a rather simple task. Depending on the workload and the available processors, a subtree could be assigned to a thread. Usually, the second stage causes difficulties for a balanced parallelization. Due to their recursive relationship, we need to maintain the parent/child information. On the other hand, a balanced parallelization requires a decoupling of this structure. A simple distributed top-down decomposition, as suggested for the first stage, only provides the top-down information, where every parent knows its children. For a push-up, we also need the bottom-up information – i.e., which block is the parent of the current block and needs to be updated by the current block. In our approach, we solve this problem by combining a central workload splitting job queue and our new *asynchronous push-up* [14].

In Figure 3.2, we outline the general design of our algorithm. After initially adding the superblock of the octree to the empty job queue, the algorithm starts to read the first job from the queue. If the size of this job, which

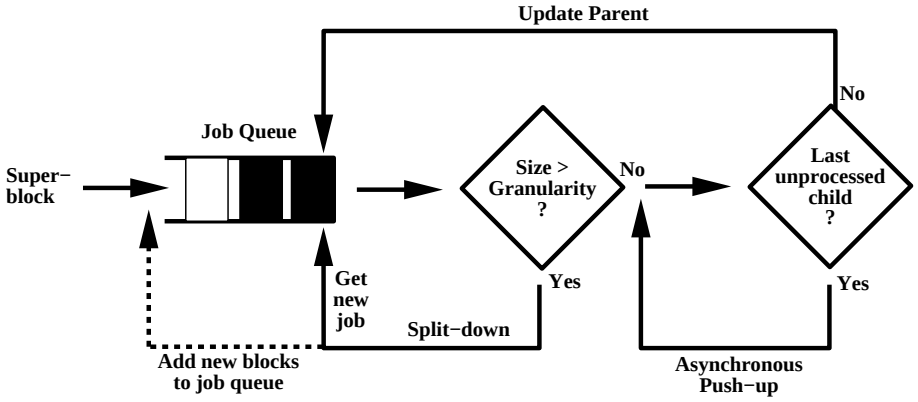


Figure 3.2: Flow of control of recursive tree construction [29]

is the block size of the octant, exceeds a certain granularity value, this block is split into its children which are added to the queue. Thereafter, a new job is read from the queue. If the block size is below the granularity value, the processing thread proceeds sequentially with the octant and the associated subtree. This differentiation of the block size granularities is necessary in order to guarantee a balance between the parallelization overhead, and the parallelization benefits.

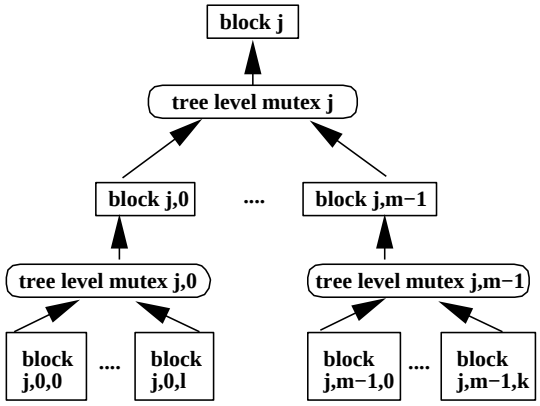


Figure 3.3: Tree level mutexes. Note that the number of child blocks of an octree block can vary significantly [29].

After processing the octant, we update its parent with the generated information (i.e., voxel value interval or standard deviation). Subsequently, we check if this octant was the last child of its parent which did not complete its computation. If this condition is matched, the processing thread continues processing the parent block and the flow of control has returned to the parent. Otherwise, the thread simply gets a new job from the queue. We call this push-up semantic an asynchronous push-up (apu). Generally, the update operation introduces a critical section to our algorithms which needs to be protected with a mutex. In our approach, we use a *tree level*

mutex which protects only one parent block and its child blocks, resulting in a minimal obstruction for other threads (Fig. 3.3). This locking mechanism is somewhat similar to *predicate locking* or *tree locking* [88]. However, the flow of update information is dictated by the control flow, which guarantees that local updates do not corrupt data in parent nodes of the octree. Hence, it is not required to lock other parts of the octree as well, which simplifies the locking mechanism significantly.

Figure 3.4 outlines the flow of control of the asynchronous push-up. Thread $t0$ is splitting the parent j into m child blocks, where thread $t1$ is processing child $j,0$, thread $t2$ is processing child j,i, and thread $t0$ is processing child $j,m-1$. After the completion of child $j,0$ and child $j,m-1$, threads $t0$ and $t1$ get a new job from the job queue. Thread $t2$ processes the last uncompleted child of parent j. Therefore, after completion of child j,i, thread $t2$ performs the asynchronous push-up and continues with parent j.

Measurements of the time spent for mutex locking show that all potential bottlenecks added by our algorithm – the mutex protected job queue access and the mutex protected asynchronous push-up – turned out to be of no significance. Up to 5% of the octree construction time was used for locking and unlocking of the job queue mutex, while no measurable time was consumed by the tree level mutexes. Comparing these numbers with the saved time due to the parallel construction (see Section 3.3 and Figures. 3.9 and 3.10), we consider this amount as insignificant. However, synchronization costs of the job queue can become significant, if the number of threads gets much higher. At no point during the computation do the threads stall because of an empty job queue. Until the final phase of the octree construction, the queue is always sufficiently filled, even with a large number of threads. However, the construction process introduces heavy memory allocation which limits the scalability of the algorithm. We will discuss this problem in detail in Section 3.3.

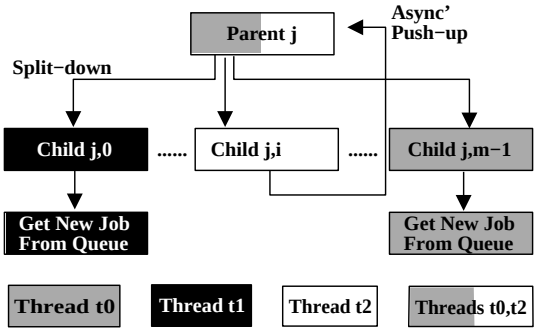


Figure 3.4: Flow of control of the asynchronous push-up (apu) [29]

Parallel Isosurface Extraction

Following Wilhelms and Van Gelder [231], we store the minimum and maximum isovalues of the voxels of an octant at all levels-of-detail. Therefore, we can rapidly decide if a subtree contains contributing, or relevant cells ("surface cells"), thus limiting the number of voxels examined by the Marching Cubes algorithm [142].

In our implementation, we recursively traverse the octree and select all contributing cells represented by the octree leaf blocks. All selected cells are assigned to active cell queues (one for each available thread) in a round-robin fashion, where at most, the number of contributing cells differs by one. After generating a balanced work distribution using this static load-balancing scheme, each thread starts its own Marching Cubes to compute the isosurface in the assigned contributing cells. To optimize cache-access, the load-balancing scheme can easily be modified to assign cells which are stored in close memory locations to one active cell queue, i.e. by assigning cells of the same volume slice to the same queue. Due to the distributed active cell queues, no additional synchronization overhead is introduced.

Note that the focus is on the construction of the octree, not on the isosurface extraction. Therefore, no time measured results on the isosurface extraction are provided in Section 3.3. However, more details can be found in [14].

3.3 Optimizing Memory Synchronization

Dataset/Size	Octree Depth	#Nodes of Full Octree	#Nodes of BONO	#Contributing cells	#triangles
A: Cavity dataset $191\times191\times191$	7	2,397K 100%	984K 41%	43K 2%	128K
B: MRI Head $258\times258\times212$	8	19,174K 100%	2,044K 11%	103K 1%	859K
C: Angiography $514\times514\times260$	9	153,392K 100%	9,917K 6%	185K 0%	1,554K

Table 3.1: Dataset overview

As pointed out earlier, the tightly coupled nature of recursive octrees requires shared memory systems for an efficient construction process. Therefore, we base our discussion on the pthread implementations on three different memory architectures of SMP systems, which provide this tightly coupled, (distributed) shared-memory; two NUMA (Non-Uniform-Memory-Access) systems (SGI Onyx2/Origin2000, SGI Origin200), and one UMA (Uniform-Memory-Access) system (SGI Challenge). All systems show different synchronization behavior, due to their architectural differences.

The Origin200 is a four processor system which is split into two subsystems with each having 512 MB of main memory and two 180 MHz MIPS

R10000 CPUs [135]. The memory and two CPUs of one subsystem are connected via a hub chip, implementing a four port crossbar. The two subsystems are connected via a "CrayLink" interconnect between the respective hub chips. The peak performance of the interconnect is 1.44 GB/s[2] Note that the performance deterioration of dataset C on the SGI Origin 200 (in contrast to the SGI Onyx2 and SGI Challenge) is due to swapping.

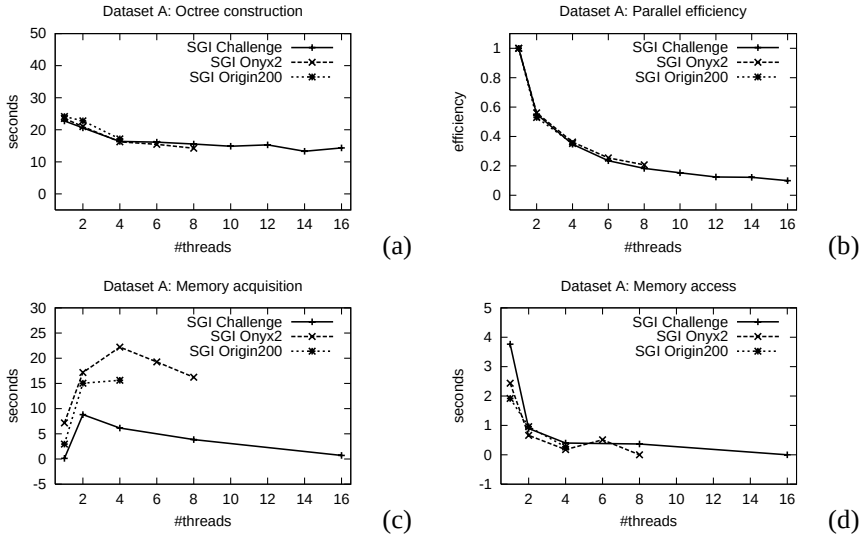


Figure 3.5: Octree construction of dataset A using **standard memory allocation**. (a) shows overall construction time, (b) shows the parallel efficiency of the octree construction, (c) shows time spent for memory allocation, and (d) shows a zoom into the time spent for memory access (the latter two are determined by profiling with 1, 2, 4, 8, and 16 CPUs only) [13].

The Onyx2 used for our measurements has ten 195 MHz R10000 CPUs. These CPUs are organized on two processor modules, with a total of five node boards [187]. The first processor module contains four node boards, while the second only contains one node board. Each node board contains up to 512 MB main memory and two CPUs. These CPUs, the memory, and the connection to other parts of the Onyx2 are interlinked via a hub chip, implementing a four port crossbar. Two node boards are connected via a six port crossbar, thus interlinking both node boards via a router to the interconnection fabric of the processor modules. This interconnection fabric interlinks both processor modules using a hypercube topology. Similar to the Origin200, the peak performance of the interconnect is 1.6 GB/s.

In contrast to the previous systems, the SGI Challenge is an UMA architecture system. 3 GB main memory is connected with sixteen 195 MHz R10000 CPUs via a system bus running at 1.2 GB/s [186]. Note that more recent SMP systems do not vary much from the discussed systems. Basically, only the CPU performance has increased, while the overall crossbar-

[2]The SGI Origin200 is in a way a reduced, two-node board version of the SGI Onyx2/Origin2000 architecture. Therefore, both systems have several common features and characteristics.

based system architecture (of the NUMA systems) is still state-of-the-art in high-performance computing.

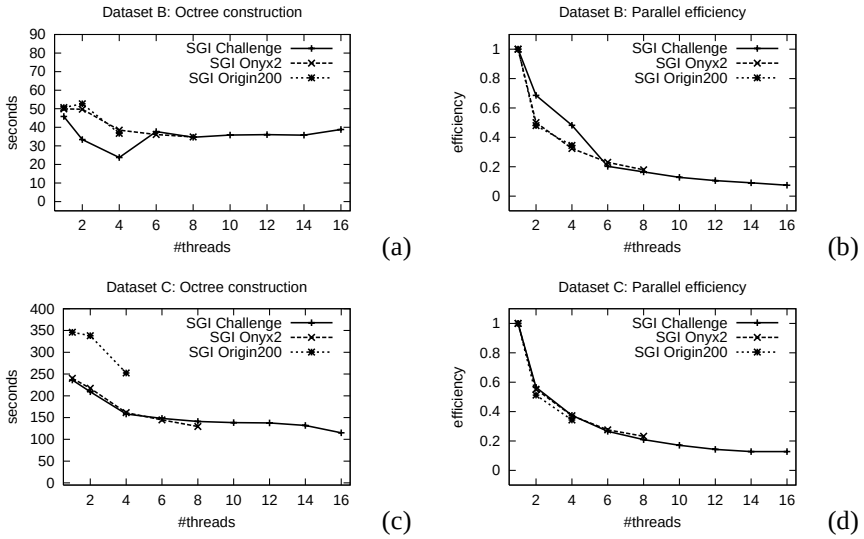


Figure 3.6: Octree construction of dataset B (a, b) and C (c, d) using **standard memory allocation**. (a, c) shows overall construction time, (b, d) shows the parallel efficiency of the octree construction [13].

Three different methods for the parallel allocation of memory are examined. These methods produce very different results on the three different architectures. For the measurements[3], we used three different volume datasets (A, B, C) of different origin (Table 3.1). Dataset A represents a velocity field generated by a computational fluid dynamics (CFD) simulation, where two sides of a fluid filled cavity are heated differently. Velocity magnitude is used as an isovalue, while the temperature is mapped as color onto the isosurface. Dataset B is a MRI scan of a human head with special focus on the cerebro-spinal-fluid (CSF) filled cavities. Dataset C is a rotational angiography dataset of a fusiform aneurysm of an arterial blood vessel in a human head. Both datasets from medical scanners (B and C) are only slightly larger (two elements) than a valid octree block size (258 and 514). Therefore, the next larger size is chosen, resulting in huge memory space savings of the BONO representation. More datasets were examined in [29, 13], also including curvi-linear grid datasets.

On all datasets, our algorithms showed the same general behavior. However, we only look in detail at dataset A, and show general behavior of datasets B and C.

[3]We measured the octree construction time with/on 1, 2, 4, 6, 8, 10, 12, 14, and 16 threads/CPUs, the detailed profiling only on 1, 2, 4, (6,) 8, and 16.

3.3.1 Standard Memory Allocation

This initial technique uses the memory allocation functions (malloc or calloc) of the C standard library (stdlib). The thread-safe versions of these functions use a global locking mechanism to guarantee mutual exclusion, usually denoted as a “big lock” [34]. Closer examination of the memory allocation using malloc/calloc shows that this mechanism introduced a significant synchronization overhead (Fig. 3.5c, Figs. 3.5, and Fig. 3.6); approximately 95% of the time spend for memory allocation is used only for synchronization. While synchronization is scaling on the SGI Challenge down to a constant overhead, memory allocation on the SGI NUMA-architecture machines (Origin200 and Onyx2) deteriorates severely. This is due to the need of synchronization of the kernel threads on each CPU of the NUMA-architectures, which is significantly more expensive than the respective synchronization on UMA-architectures. Note that memory access through-out the whole virtual memory of all three architectures scales nicely, thus exhibiting no varying memory latency between the memory levels.

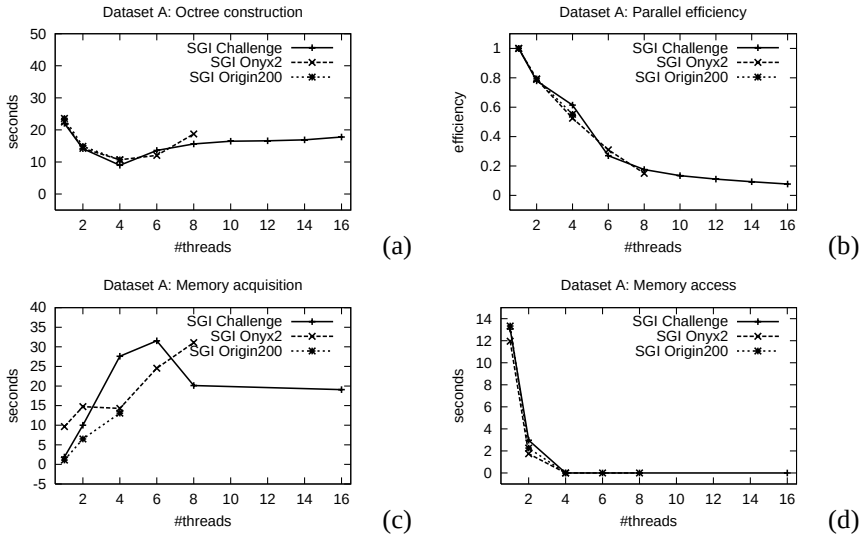


Figure 3.7: Octree construction of dataset A using **process global memory pre-allocation**. (a) shows overall construction time, (b) shows the parallel efficiency of the octree construction, (c) shows time spent for memory allocation, and (d) shows a zoom into the time spent for memory access (the latter two are determined by profiling with 1, 2, 4, 8, and 16 CPUs only) [13].

3.3.2 Process Global Pre-allocation

The previous experiment showed that the standard thread-safe memory allocation functions introduced an expensive memory-locking mechanism. However, the pthread mutexes used in the experiments suggested that the standard mutex locking mechanism is a faster, and therefore cheaper synchronization mechanism. Consequently, we introduced an alternative mem-

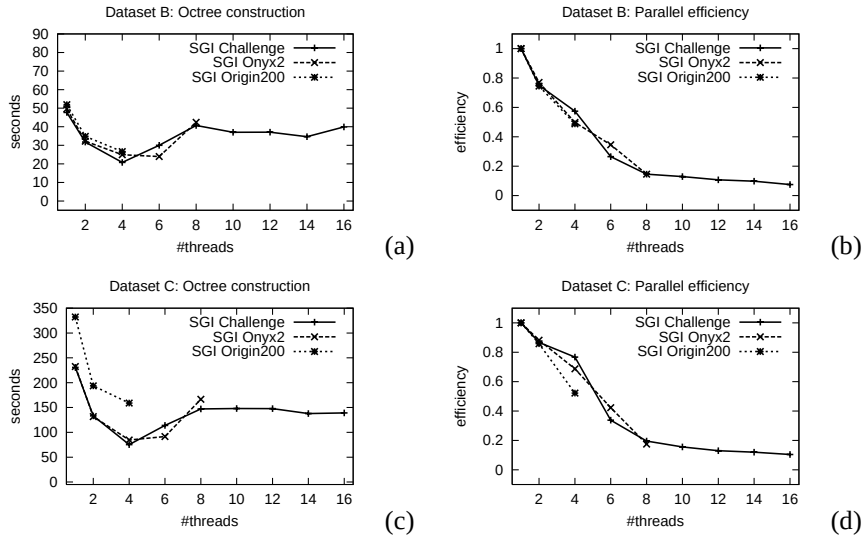


Figure 3.8: Octree construction of dataset B (a, b) and C (c, d) using **process global memory pre-allocation**. (a, c) shows overall construction time, (b, d) shows the parallel efficiency of the octree construction [13].

ory allocation method where a huge chunk of memory is allocated before entering the parallel region of our code. Later, we assign blocks of this memory to the octants using a customized data-structure, similar to an array of octants. The actual assigning action is protected by a data-structure local mutex, where only one structure is used for the whole construction process. Using this method, we obtained good scaling on the SGI Origin200 architecture (Fig. 3.7 and 3.8). Memory synchronization in particular scaled down to a fraction of the original amount. The SGI Onyx2 architecture showed a different picture. While the four CPU Origin200 only needs one additional crossbar hop to the CPUs on the other subsystem, access to all other CPUs of the Onyx2 requires up to two hops via the interconnection fabric, thus increasing the synchronization overhead similar to the standard memory allocation scheme. On the SGI Challenge, increasing memory requests of the threads increased the costs for synchronization. In contrast to memory locking using the standard library functions, global mutex locking does not scale, resulting from increasing contention of the growing number of threads. Similar to the measurements of the previous standard allocation approach, memory access does scale without any measurable latency difference between the different memory levels.

3.3.3 Thread Local Pre-allocation

From the previous experiment, we learned that mutex locking using only one global mutex can increase synchronization costs due to high contention. Consequently, we need to reduce this contention by using multiple locks. Due to the fact that all systems are shared-memory systems and that mem-

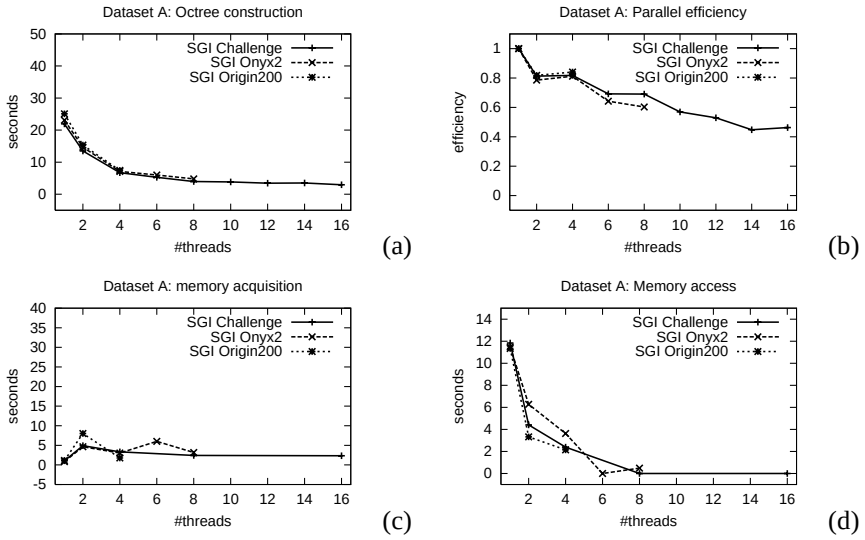


Figure 3.9: Octree construction of dataset A using **thread local memory pre-allocation**. (a) shows overall construction time, (b) shows the parallel efficiency of the octree construction, (c) shows time spent for memory allocation, and (d) shows a zoom into the time spent for memory access (the latter two are determined by profiling with 1, 2, 4, 8, and 16 CPUs only) [13].

ory access through the interconnect always scaled nicely, despite the interconnection technology (bus or crossbar), this approach uses the previous pre-allocating data-structure for each thread. Therefore, specific memory locking is not necessary.

Figures 3.9 and 3.10 shows the results of this approach. Memory allocation time (including the synchronization overhead) could be reduced to a fraction of the previous amounts on all three systems. It scales throughout all CPUs, resulting in a balanced parallelization of the complete construction process. Furthermore, the measurements of the memory access show no evidence of a varying memory latency, which is consistent with the previous measurements.

3.4 Summary

In contrast to many statements in NUMA thread programming, cross-node memory access introduced no measurable latency or bottleneck in our application. At no time, did we find evidence of access penalties, once the memory access was leaving the lower hierarchy level of node board or processor module local memory. However, this cross-node memory access latency might become more significant using 32 or more CPUs on larger systems, when synchronization costs are no longer hiding the increasing memory access latency.

Yet, synchronization turned out to be expensive on NUMA architectures. This potential bottleneck emerged during dynamic and parallel mem-

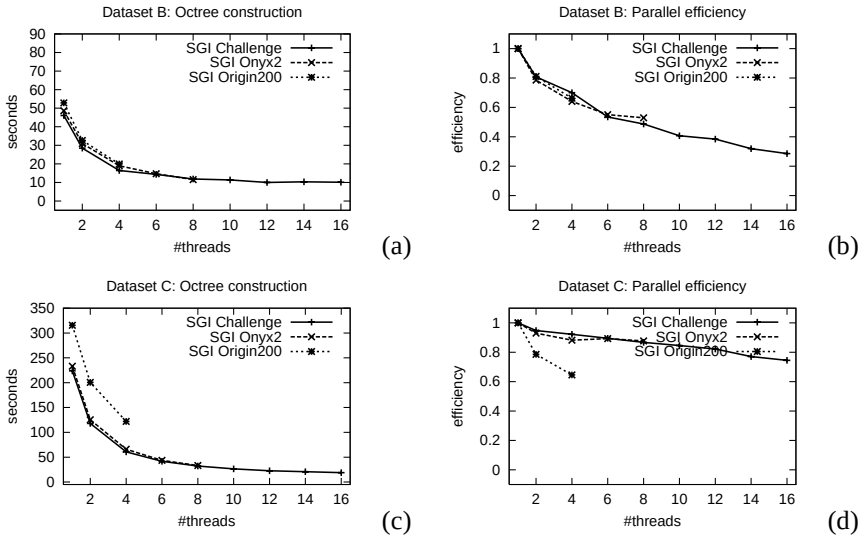


Figure 3.10: Octree construction of dataset B (a, b) and C (c, d) using **thread local memory pre-allocation**. (a, c) shows overall construction time, (b, d) shows the parallel efficiency of the octree construction [13].

ory allocation. The necessary locking mechanism represented a significant slow-down in thread-parallel applications. We discussed three different approaches which address this problem. The final approach using a thread local pre-allocation scheme solved the problem on three different architectures and ensured a scaling scheme for the construction of tree hierarchies.

Compared to the memory synchronization overhead, the mutex protected job queue access represented no significant overhead. Nevertheless, using even more threads, this access might introduce a more significant bottleneck. Therefore, future work will focus on a distributed job queue.

In this chapter, we presented a new algorithm for the parallel construction of recursive tree structures. It is used to compute hierarchical scene organizations for occlusion culling queries, as described in the next chapter. In particular the increasing size of medical volume datasets required the parallel execution to ensure a timely construction of the data-structure. Note that this algorithm is also suited for quadtrees, BSP-trees, or other recursive tree structures, although it was only discussed for octrees.

Chapter 4

Visibility and Occlusion Culling

Hidden-line-removal and visibility are among the classic topics in computer graphics [65]. A large variety of algorithms are known to solve visibility problems, including the z-buffer approach [198, 36], the painter's algorithm [65], and many more. However, the size of polygonal datasets used in scientific visualization has increased rapidly in the past decade, urging the need to reject groups of polygons before entering the rendering pipeline. The respective approaches are usually referred to as culling algorithms.

4.1 Introduction

By now, a large number of different algorithms has been published on the topic of culling. A recent survey and taxonomy can be found in the ACM SIGGRAPH 2000 course #4 [44], or in [57]. Here, we will only briefly review a small subset of culling approaches. Basically, three different approaches can be distinguished:

- Back-face culling, which removes single polygons or groups of polygons with a surface normal facing away from the view-point, thus exposing the back-face of these polygons. This is the only culling technique currently supported in core OpenGL [238]. A more recent technique has improved the performance of back-face culling [132].
- Visibility culling is frequently used as the general term for visibility and occlusion culling. Here we use visibility culling in the context of the computation of exact visibility, which determines the visibility status of a scene object.
- Occlusion culling, which – in contrast to visibility culling – computes the occlusion status of a scene object, thus returning a list of objects which are definitely occluded, leaving all other objects as potentially visible, although they might be not visible. Occlusion culling algorithms are referenced as conservative – if at least all visible scene objects are determined as potentially visible – and as non-conservative – if some visible scene objects are classified as occluded.

Most exact visibility culling algorithms are based on the aspect graph from computer vision [129, 169, 84] which compute the exact visibility of the objects of a region of view-points. However, the high complexity of this approach ($\theta(n^9)$, where n is the number of polygons in the scene) eliminates the practical use. Other approaches relax regional or exactness conditions to reduce the computational complexity [46], or compute the visibility based on specific occluders by calculating a shadow-frustum which occludes other objects [47, 114].

Occlusion culling approaches in contrast avoid the costly computation of exact visibility; instead, they compute whether objects are occluded, which is a significantly cheaper operation, since heuristics and object approximations (i.e., bounding boxes) can be used to achieve efficient culling. Occlusion culling algorithms can be classified in object-space and image-space approaches, depending on if the occlusion is computed depending on the view-points in object-space – frequently in a pre-process –, or if it is computed for each frame in image-space. Approaches for architectural walkthroughs are usually object-space approaches, which compute a potential-visible-set (PVS) based on a decomposition of the model; each room of the architectural scene is considered as a cell, and the visibility for that cell is computed based on portals, connecting this cell to other cells [117, 3, 2, 206, 205]. A variation of this approach computes the visibility through the portals in image-space [144]; the 2D screen-space bounding box of the geometry objects of every neighboring cell is intersected with the screen-space bounding boxes of the portal. If the intersection is empty, the object is not visible. Hong et al. added an OpenGL depth buffer based occlusion test to determine whether subsequent cells are occluded, if the intersection is not empty [110]. Many other object-space approaches exploit additional boundary conditions of city-walkthroughs [45, 237, 181], which is also referred as $2\frac{1}{2}$D visibility, since view-points are usually limited to street positions.

View-frustum culling is an object-space technique which is used in most culling approaches prior to the actual occlusion culling due its low computational costs. Every scene object of a hierarchical scene organization is tested for intersection with the view-frustum. Objects with no intersection can be culled rapidly, since they are not visible. View-frustum culling was first introduced by Clark [41], and later implemented as described by Garlick et al. [80]. Unlike the usual CPU-computed view-transformation [80, 101], we will later introduce a view-frustum culling technique exploiting OpenGL's selection buffer [22].

In contrast to the object-space approaches, image-space algorithms establish occlusion information for each frame by projecting the geometry or approximations of the geometry into screen-space, usually after a view-frustum culling step in object-space. Greene et al. used a pyramid of depth values to trace visual contributions of the virtually rasterized silhouette of the bounding box of scene objects, which indicate visibility [92, 89]. Zhang et al. [243, 242] used a hierarchical screen projected map of pre-selected

occluders to check if the scene elements are occluded, and finally, we proposed to use a *virtual occlusion buffer* to trace visual contributions [22]. More details on this approach will be presented later on.

While most of the presented occlusion culling algorithms are software-based, some are exploiting computer graphics hardware to accelerate the occlusion queries [243, 22]. Full hardware support was already available as a z-query in the not anymore available Denali GB graphics on the Kubota Pacific Titan workstation [92], which detected changes in the depth buffer. Two analyses and implementations of a simplified version of the hierarchical z-buffer algorithms were presented later in 1999 by Xie et al. [239] and Greene [91]. A variation of the hierarchical z-buffer algorithm was recently implemented in the "HyperZ-Technology" of ATI [159], which implements are reduced resolution z-buffer on-chip to skip texturing of occluded pixels. In 1998, we proposed an occlusion culling extension to the OpenGL rendering pipeline [21], which provides detailed quantitative and qualitative information on the visibility. Also in 1998, Hewlett-Packard released the VISUALIZE fx-series of graphics subsystems which provided the Hewlett-Packard occlusion culling flag (HP flag), indicating if geometry (i.e., a bounding box) rendered in a specific occlusion mode would generate a footprint in the depth buffer, indicating visibility [183]. This HP flag was later used in virtual endoscopy application to speed up rendering [25], and to evaluate model[1] organization hierarchies [153], as we will report on later in Section 4.6. Last year, SGI released the Visual PC [189], which introduced the OpenGL instruments extension for occlusion culling queries similar to the HP flag. Finally, we proposed an extension to the rasterizer stage of a rendering pipeline to cull small triangles and pixel groups on top of "traditional", HP flag-like occlusion culling [150, 151]. Earlier this year, the next generation of HP graphics hardware, the VISUALIZE fx5 and fx10, was released which also provide quantitative visibility information similar to the functionality proposed by Bartz et al. [21]. Furthermore, they provide multiple, asynchronous occlusion queries which take previous, not necessarily up-to-date depth buffer-based visibility information into account.

With all the different approaches (for a more detailed overview refer to the recent survey on visibility in [44] or to [57]) the question remains which algorithm is best suited for an application. Generally, object-space methods lack sufficient performance if the polygonal models become too large, because the actual visibility query is too expensive. However, this situation changes once topology information can be exploited, such as floor-plans of building walkthroughs. BSP-tree methods [164, 76] are applied with good performance in computer games, since a suitable hierarchical scene organization is inherently known due to the game design process [1]. However, if no assumption on the scene is available and the polygonal complexity of the scene is high, only image-space-based methods are able to provide sufficient performance. Nevertheless, these methods frequently require hardware support [183, 25] or introduce budget-oriented non-conservative tech-

[1] In the course of this chapter, we will use the terms *scene* and *model* in an interchangeable fashion.

niques [22, 127].

In the course of this section, we will describe the basic hierarchical occlusion culling approach used by many occlusion culling algorithms, and the specific version using the HP flag used in our applications (Section 4.2). In Section 4.3, we discuss our efficient methods to perform view-frustum culling using OpenGL. Our new software-based occlusion culling approach based on OpenGL is presented in Section 4.4, followed by a brief discussion of graphics hardware modifications necessary to provide a more detailed and efficient occlusion culling mechanism (Section 4.8). In the following sections, we will describe our techniques how to efficiently traverse the hierarchical scene organization (Section 4.5), how to generate effective hierarchical scene organizations (Section 4.6), and finally how to optimize the bounding volumes to improve the culling performance (Section 4.7).

4.2 Hierarchical Occlusion Culling

Most current occlusion culling approaches are using hierarchical techniques to reduce the complexity of an occlusion query. Usually, a model or scene is decomposed into objects and these objects are organized in a hierarchical tree representation, i.e., octrees, or *sloppy n-ary space partitioning trees* (snSP-trees) [22]. The advantage of this organization is that the number of occlusion tests is now depending on the number of tree nodes, not on the number of polygons in the model.

To clarify the terminology, we briefly introduce the terms later used to describe a hierarchical representation. Generally, a polygonal model can be decomposed into smaller parts, where this model organization can be either hierarchical or non-hierarchical. We call each part of this decomposition a *scene entity*. If information at different multi-resolution levels is required, usually a hierarchical organization is chosen, where different scene entities are combined into one parent entity which contains the whole information of the associated scene entities, or only information with less detail (a lower level-of-detail). This decomposition can be represented as a tree which is referred to as *scene tree* or *scene graph*. This tree contains two different kinds of nodes (*scene entities*); *inner scene tree node* (or *scene node* for short), and *leaf nodes*. Only the leaf nodes contains the geometry of the actual model and the bounding volume (usually a bounding box) with respect to the used decomposition method. In contrast, a scene node does not contain any geometry of the actual dataset; it only contains the spatial boundaries of the associated geometry nodes, thus the scene node is the implementation of the abstract scene entity. In Section 4.6, three different approaches to generate a scene or model hierarchy starting from given models are discussed.

Commonly, the occlusion test starts with the root node of the scene tree and descends to the leaf nodes, depending on the results of the individual tests. Clark suggested this approach as early as 1976 [41], where he proposed to use view-frustum culling as a basic culling technique (see Sec-

tion 4.3). An implementation of this technique was presented much later by Garlick et al. [80]. Hierarchical view-frustum culling was adopted by most culling approaches, while the basic difference arises in the additional specific occlusion queries used (see the beginning of this section).

```
glDepthMask(GL_FALSE);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glEnable(GL_OCCLUSION_TEST_HP); {
  render(object_bounding_volume);
  glGetBooleanv(GL_OCCLUSION_RESULT_HP, &not_occluded);
} glDisable(GL_OCCLUSION_TEST_HP);
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
if (not_occluded)
   render(object_geometry);
```

Table 4.1: Using the HP flag; modifications to depth and color buffer are disabled, the HP occlusion mode is enabled. After rendering of the bounding volume, bounding the actual geometry, the HP flag is queried if the bounding volume would be occluded. If it is not occluded, the associated object geometry is rendered after enabling depth and color modifications.

With the introduction of the HP flag [183] a fast and fully hardware-supported occlusion test became available. The HP flag-supporting graphics hardware provides an additional rendering mode (`GL_OCCUSION_TEST_HP`) which traces the depth buffer for potential changes while rendering geometry. If modifications to the framebuffer are disabled, the potential changes can be traced without actually modifying the color or depth buffer and are available after a rendering pipeline flush (see also Table 4.1). However, if the full geometry is used to test for occlusion, no saving can be gained, since most of the rendering work (transformation, lighting, and rasterization) is already done. Therefore, bounding volumes are used which approximate the shape of a scene object. In most cases, a simple axis-aligned bounding box (AABB) is used for this purpose. If this bounding box does not generate a contribution to the depth buffer, all the contained geometry will not generate a contribution either. On the other hand, if the AABB does generate a footprint in the depth buffer, the associated geometry might be visible (see Section 4.7 for a more detailed discussion).

In [183], a very simple approach was proposed, where each object of the scene was tested individually for occlusion. This approach was extended in HP's Jupiter large model rendering toolkit [101] (formerly known as DirectModel) to apply CPU-based view-frustum culling and subsequently, the HP flag-based test hierarchically to every scene node. Unfortunately, the HP flag-based occlusion test is rather expensive, since it requires a costly flush of the rendering pipeline[2]. Therefore, our HP flag based approach (see Fig. 4.1) first evaluates the scene tree using the fast OpenGL-assisted view-frustum culling (see Section 4.3). After depth-sorting of the remaining leaf

[2] According to Severson [185], the costs for one occlusion query are equivalent to the rendering of 190 triangles of the size of 25 pixels.

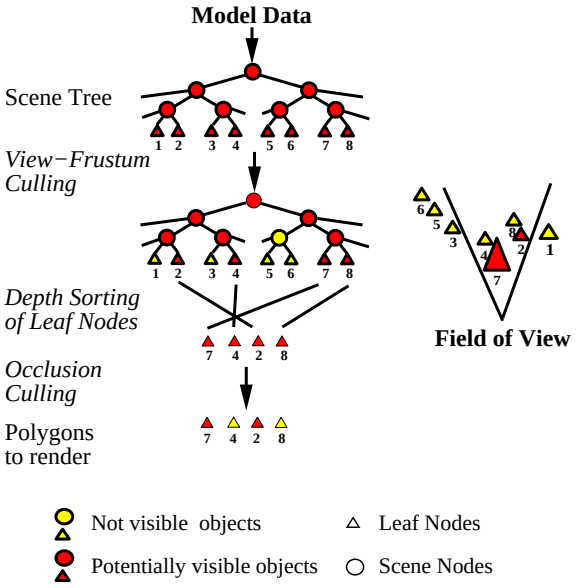


Figure 4.1: Hierarchical occlusion culling

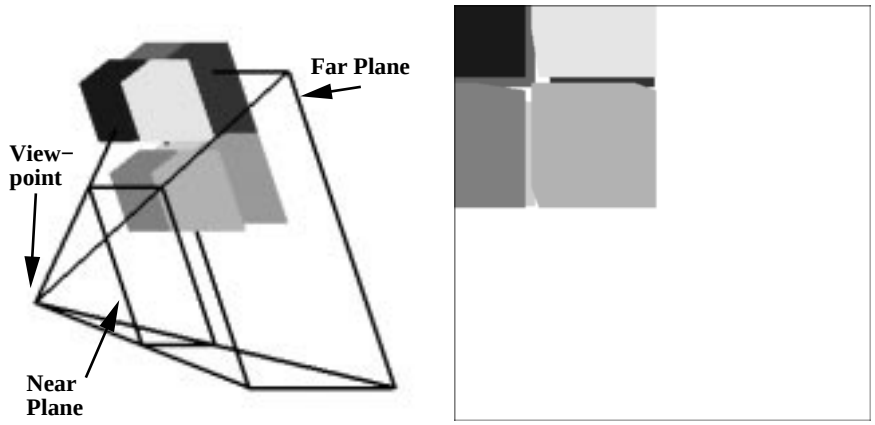
nodes of the scene tree and rendering of the n front-most objects[3] (since they are almost always visible), HP flag-based occlusion culling is applied. This approach requires a significantly lower number of occlusion culling queries than the HP/Jupiter algorithm, thus reducing the occlusion culling overhead substantially.

4.3 View-Frustum Culling

Most view-frustum culling implementations use the CPU to transform the vertices of the object or a bounding volume into normalized view-space (after (perspective) projection). These transformed vertices are then tested if they are located inside of the view-frustum, represented by the unit box [80, 173, 101].

In contrast, we use the *OpenGL selection mode* to check if the bounding volume intersects with the view-frustum (see Fig. 4.2). The selection mode is intended to identify objects rendered into a specifiable area of the screen [238] to implement the picking operation in OpenGL. In our case, the screen area of interest is the whole screen, since it is the image-space representation of the view-frustum (Fig. 4.2b). The polygonal representations of the bounding volumes (as convex hulls) are transformed, clipped against the view-frustum, and finally rendered without actually contributing to the framebuffer. Once the hit buffer of OpenGL's selection mode con-

[3] n is a very application depending parameter. For endoscopic applications, approximately 10% of the front-most nodes are virtually never occluded. For MCAD models, n is usually smaller (approximately 5%), since case elements frequently occlude the interior geometry.



(a) (b)

Figure 4.2: View-frustum culling: (a) all eight objects intersect with the view-frustum. (b) Current view of the scene using the perspective view-frustum shown in (a); all eight visible objects generate a hit in the selection buffer (box of near plane is also drawn).

tains a footprint from a bounding volume, this bounding volume intersects the view-frustum. If the entire bounding volume resides within the view-frustum – the hit buffer contains also footprints of all corners of the bounding volumes –, all child nodes of the scene tree (if a hierarchical representation is used) are also located within the view-frustum, since the bounding volume of the parent node is a convex hull of all child nodes. If the bounding volume resides only partially within the view-frustum, we recursively test the child nodes of the scene tree hierarchy. After the view-frustum culling step all leaf nodes are tagged either as *potentially visible*, if they are not culled by the view-frustum culling, or *definitely not visible* otherwise.

Occasionally, the bounding volume can completely contain the view-frustum, resulting in no footprints in the hit buffer of the selection mode, since the geometry of the bounding volume is not visible. This can be prevented by testing if the closest bounding volume plane lies between the near plane of the view-frustum and the view-point, or if the view-point lies within the bounding volume.

4.4 A Virtual Occlusion Buffer Approach

4.4.1 The Virtual Occlusion Buffer

As pointed out earlier, the basic difference between most image-space occlusion culling algorithms is in how they determine if an object (bounding volume) is occluded. Here, we present an OpenGL-assisted occlusion culling algorithm that uses a *virtual occlusion buffer* to trace footprints of an object in the depth buffer [22]. The actual implementation on an SGI O2 and

an SGI Octane/MXE uses the OpenGL stencil buffer[4] as virtual occlusion buffer (VOB), since measurements revealed the best reading performance compared to other buffers of the framebuffer. For our use as VOB, the stencil buffer is recording the ID of the object passed through the depth buffer test at that specific pixel, which indicates that this object is not occluded. Specifically, we render the bounding volume into the stencil buffer, depending on its visibility based on the OpenGL depth buffer, and trace the 2D screen-space bounding box of the rasterized object bounding volume.

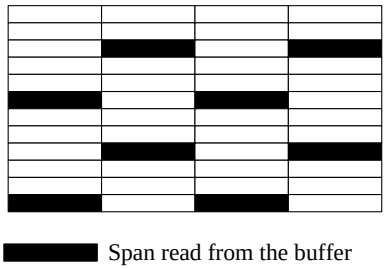


Figure 4.3: Double interleaved sampling scheme; $\frac{1}{n}$ of the 2D bounding box is read, with $n = 6$ [22].

Reading access to the framebuffer is generally expensive, since it requires high set-up costs to read from the OpenGL rendering pipeline. Consequently, reading from the VOB is the most costly single operation of this approach, which accounts for approximately 90% of the total occlusion culling costs. If a model is composed of many objects, we require many accesses to the VOB, resulting in a less efficient operation. Additionally, bounding volumes close to the view-plane often occupy a large screen-space, thus requiring read operations from large portions of the VOB. Overall, two measures limit the use of a VOB; the number of objects increasing the number of occlusion queries, and the screen-space size of the objects, increasing the size of the VOB area to be traced. While the first limiting factor can only be controlled by the geometry/object assignment, we address the second limitation by using a double interleaved, progressive sampling scheme that reads spans of pixels from the VOB (see Fig. 4.3). Basically, this scheme implements a *sampling* of the VOB, where $\frac{1}{n}$ of each 2D bounding box is read in each iteration. In other words, the algorithm needs n iterations to fully read the entire bounding box.

Depending on the graphics subsystem, the reading set-up time, or the actual reading time, is the dominating time, hence influencing the sampling scheme. On the O2, reading ten spans required almost the same time as reading all the pixels in one read operation, suggesting that the O2 graphics system is dominated by the reading time. In contrast, the required time on the Octane/MXE was significantly increased if multiple read operations were employed, revealing a set-up time dominated scheme. This

[4] Intentionally, the stencil buffer is used for multi-pass rendering, such as limiting rendering to a certain area of the screen.

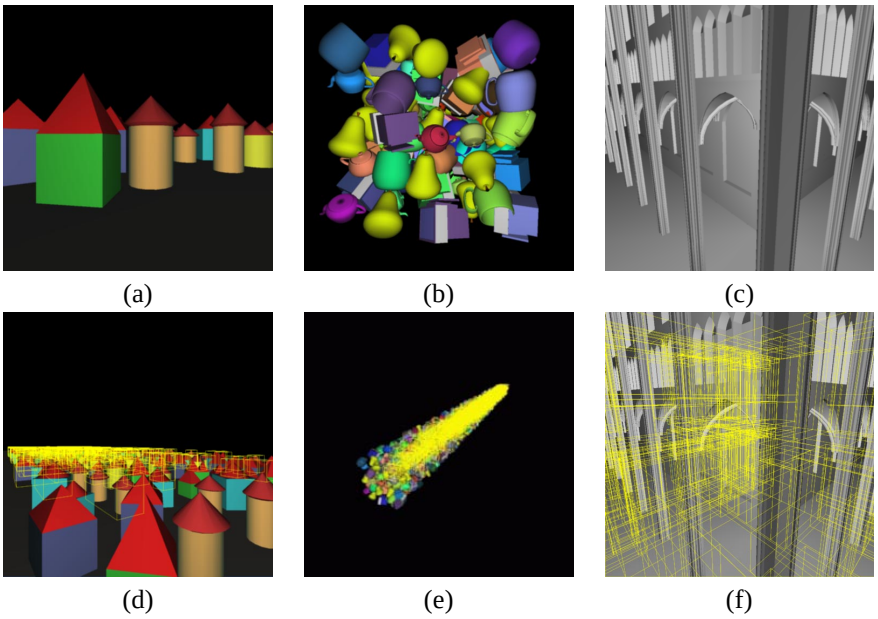


Figure 4.4: Models – (a,d) city, (b,e) garbage, (c,f) cathedral; upper row: views with occlusion culling; lower row: overviews with (yellow) bounding boxes of occluded model elements [22].

is not really surprising, since most high-performance graphics subsystems are highly interleaved, and have distributed rendering engines, which require more synchronization for reading data from the rendering pipeline than low-end graphics subsystems that are commonly not interleaved. For these reasons, we read a lower number of larger chunks from the VOB on the Octane/MXE than on the O2 to achieve a sufficient speed-up. Note that sampling introduces a non-conservative heuristic which trades off quality versus rendering performance. However, the later experiments showed no apparent artifacts (even in animations), although small image differences are present.

4.4.2 Analysis

We ran several experiments to evaluate the performance of the virtual occlusion buffer approach. For a complete listing of the results, please refer to [22].

All experiments were performed with four different scenes; an architectural model of eight gothic cathedrals – arranged on a 3D array –, a city scene, a forest scene to demonstrate quantitative culling, and – similar to [243] – the content of a virtual garbage can of rather small objects. A hierarchical model organization for each dataset was generated by manual tuning of a model hierarchy generated by SGI's OpenGL optimizer (see Section 4.6). Frame-rate and culling rate are measured over a sequence of about 100 frames on an SGI O2 workstation (256 MB, R10000 @ 175 MHz), and

on an SGI Octane/MXE (896 MB, R10000 @ 250 MHz CPU)[5].

model	#triangles	culling O2	culling MXE	frame-rate MXE/[fps]	speed-up O2/MXE
cathedrals	3,334,104	91.3%	92.5%	5.3	4.2 / 12.6
city	1,056,280	99.8%	87.7%	7.7	4.8 / 9.8
forest	452,981	84.7%	80.5%	4.7	2.6 / 6.1
QOC		89.0%	83.0%	5.0	3.8 / 6.5
garbage	5,331,146	96.0%	38.2%	0.3	7.0 / 5.0

Table 4.2: Average performance of virtual occlusion buffer culling compared to view-frustum culling only. Different culling rates are due to different sampling parameters. The forest scene reflects comparison of quantitative occlusion culling (QOC) and VOB culling to view-frustum only culling.

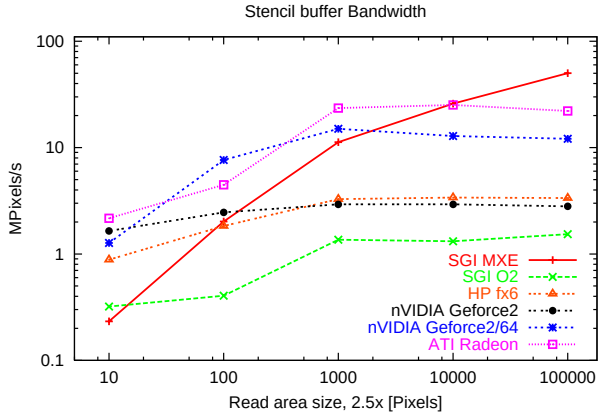


Figure 4.5: Bandwidth of stencil buffer reads on various platforms: read performance is measured by reading the constant total amount of 2.5 MPixels in a decreasing number of single read operations of a growing read area.

The polygonal complexity and speed-up numbers are listed in Table 4.2. The costs for view-frustum and occlusion culling vary with the different model organization granularities; i.e., for the cathedral dataset, view-frustum culling accounts for approximately 5% of the costs, occlusion culling (mostly dominated by reading the stencil buffer) for approximately 20%, and rendering of the as visible classified model elements for about 75% of the total frame time. The distribution and total costs basically depend on the architecture of the graphics subsystem. A highly interleaved graphics system like the InfiniteReality of SGI is likely to perform worse than the O2 for this algorithm, due to the high set-up costs of reading from the framebuffer, while single-pipeline low- and mid-end graphics are mostly limited by the total amount read from the framebuffer. This effect can also be seen in Figure 4.5, where the SGI Octane/MXE performs best for reading large areas from the stencil buffer, while it performs worse than the O2 for very

[5]Note that the datasets used for the SGI O2 were using triangle strips, while this was not possible for technical reasons on the Octane.

small areas. Modern PC graphics subsystems show a similar behavior; the high-end ATI Radeon and nVIDIA Geforce2/64 perform better reading large parts, while their performance degrades faster than the mid-range systems (nVIDIA Geforce2, HP VISUALIZE fx6) with smaller reading areas. However, it is interesting that the three years old SGI Octane/MXE performs an order of magnitude faster for the largest read area, but also an order of magnitude slower for the smallest read areas.

As a bottom-line, we can see that the PC graphics subsystems have lower set-up costs and are therefore better suited for the VOB occlusion culling approach. Nevertheless, the sampling parameters – sampling frequency and how much is read from the framebuffer per sampling – of the occlusion culling approach trade off visual quality and minimize set-up time, and need to be parametrized for each graphics system.

4.4.3 Quantitative Occlusion Culling

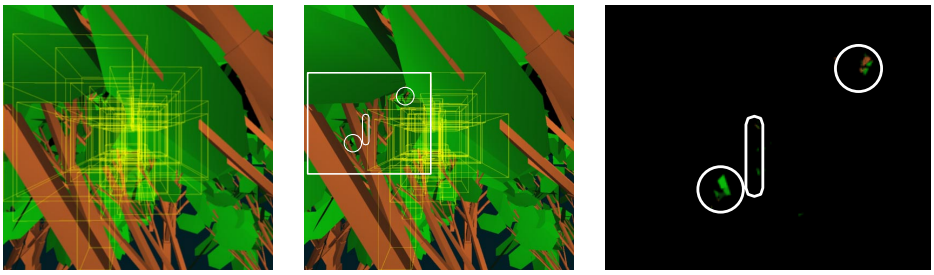
In many complex scenes some of the objects are almost occluded; they are barely noticeable, since only a few pixels contribute to the final image. Furthermore, objects which were classified as potentially visible, are actually not visible, since the occlusion test is based on the bounding volume (usually a bounding box) that is often much larger than the actual object geometry. Unfortunately, most occlusion culling approaches do not provide quantitative visibility/occlusion information; only the extensions proposed in [21] (see also Section 4.8) and the new HP VISUALIZE fx5/10 functionality provides this data. However, our VOB-based approach also provides some quantitative measures which can be combined to cull objects which are virtually occluded, either because only their bounding volume is not occluded – in contrast to the actual geometry which is occluded –, or they are contributing only a few pixels to the virtual occlusion buffer [22].

Each bounding volume of scene tree objects which generates a footprint on the virtual occlusion buffer needs to be evaluated. For a perspective projection, we consider the size of its 2D bounding box relative to the view-plane (first term in Equation 4.1) and the relative distance of the object to the view-point (eye in second term of Equation 4.1). In other words, $QOC(Obj)$ of Equation 4.1 describes the relative contribution of the 2D bounding box with respects to the distance of the object to the viewer.

$$QOC(OBJ) := \begin{cases} \frac{Size2DBB(Obj)}{SizeVP} * \frac{D(Eye)+D(Obj)}{D(Eye)} & \text{, if perspective view} \\ \frac{Size2DBB(Obj)}{SizeVP} & \text{, if parallel view} \end{cases} \tag{4.1}$$

where Size2DBB(*Obj*) returns the number of pixels of the screen projection of the bounding box, SizeVP returns the number of pixels of the view-plane, D(*Eye*) returns the distance between view-plane and view-point, and D(Obj) returns the minimal distance between the *Obj* and the view-plane. For a parallel projection, the distance term in Equation 4.1 is simply removed.

For each potentially visible object, we evaluate Equation 4.1. If the quantitative occlusion $QOC(Obj)$ of Object Obj is smaller than an user defined threshold, we consider this object as occluded. Objects which have a larger screen contribution of their 2D bounding box are consequently less likely culled than objects with a smaller contribution of the 2D bounding boxes. For perspective views, this heuristic additionally favors objects which are farther away, since they are larger in object-space than closer objects with the same projected contribution. During changes of the view-point (i.e., rotations or translations) the farther, but larger (in object-space) objects are more likely to remain visible than the closer, but smaller objects.



(a) (b) (c)

Figure 4.6: Alley of trees; bounding boxes of culled objects are marked yellow. 0.1% of the pixels rendered with quantitative occlusion culling are wrong compared to the correct image with (conservative) VOB-based rendering: (a) Quantitative occlusion culling (94% culled), (b) conservative occlusion culling (88% culled), (c) differences of images without boxes; white markers point to image differences (image is zoomed to the relevant area, according to white rectangle in (b) [22].

Once the quantitative occlusion of an object (or its bounding volume) is established, the question arise of how to deal with this object. Different strategies for almost occluded objects are possible. First, as mentioned earlier, the actual geometry is usually smaller than the associated bounding volume. A partially not occluded bounding volume does not necessarily mean that the associated geometry is not occluded. Therefore, culling of the object may not have any visual impact. Second, even if a small fraction of the actual geometry is not occluded, we will probably not see any detail[6]. Using a lower level of detail representation of the respective object geometry can be indicated, since no details will be noticeable.

Figure 4.6 shows visual results of our quantitative occlusion culling mode using the first strategy, compared to the standard occlusion culling mode of our algorithm (see also Table 4.2). A threshold of 100 (0.02% of view-plane) was used on a view-plane of 650×650 pixels. Average distance to the objects is 10m; their bounding box projection covers on average 423 pixels; the view-point is located 0.002m behind the view-plane. 94% of the geometry are culled using quantitative occlusion culling, compared to 88% with VOB-based culling with full (conservative) VOB sampling. The pixel

[6] From a sampling theory point of view, the visible fractions of the actual geometry do not allow a reasonable signal reconstruction of the projected geometry, due to severe undersampling.

difference between the quantitative (Fig. 4.6a) and the conservative Image (Fig. 4.6b), the *pixel error*, is not larger than 0.1%.

4.5 Efficient Scene Traversal

The efficiency of image-space occlusion culling approaches depends on the sequence by which occlusion is determined, since objects in the back of the scene are very unlikely to occlude other objects. Greene et al. [92, 90] used a BSP-tree front-to-back sorted scene to ensure efficiency. Zhang et al. [243] select "good occluders" from a pre-selected occluder database based on heuristics such as distance to view-point, object-space size, and number of polygons (to reduce rasterization costs).

In our VOB-based approach [22] (see also the previous section), we perform front-to-back sorting of the scene objects along their smallest depth value, which is the closest object vertex. These depth values are computed during view-frustum culling which involves the transformation of the scene objects (or their bounding boxes) into the view coordinate system. We also proposed an interleaved scheme where the occlusion test is performed also on the inner nodes of the scene hierarchy tree, once view-frustum culling has determined those as potentially visible – and sorted these nodes front-to-back by the smallest (closest) depth value of the nodes. The scene tree is traversed in a breadth-first strategy, which also enables an early occlusion culling of inner scene tree nodes, which contain several leaf nodes (and inner scene tree nodes). With this interleaved scheme, temporal coherence can be exploited. Previously as potentially visible determined nodes are most likely also potentially visible from a view-point nearby. In contrast, a previously as occluded tagged scene node might change its occlusion status; therefore, if we only test the as occluded tagged nodes, we save time spent on occlusion testing, while we guarantee that we do not cull scene objects previously established as potentially visible. If the occlusion status of a scene node changes from occluded to potentially visible, all its child nodes have to be tested as well. Once the camera motion stops, or the view-point is significantly different from the reference view-point which we used for the initially computed occlusion status of the scene nodes, we re-compute the occlusion status for the full scene tree.

Additional to time coherence, the interleaved scheme allows the assignment of an occlusion budget to balance rendering and culling costs. If this occlusion budget is used up or even overdrawn, culling is limited to cheap view-frustum culling [22]. Other strategies are also possible, such as including a second budget for rendering. If this budget is also overdrawn, the remaining scene objects can be skipped completely from testing and rendering. The resulting visual impacts can be significant, but as the remaining objects are usually behind other objects due to depth sorting, it is likely that the visual impact is limited. A similar approach was later developed by Klosowski and Silva [127], where a sorting algorithm for cells of unstruc-

tured grid datasets was used to prioritize rendering.

The early HP flag-based approaches used a brute-force testing of all scene elements with no specific object ordering [183]. We modified this approach to view-frustum culling of the complete scene hierarchy tree, sorting of the remaining leaf nodes, and subsequent front-to-back occlusion testing of the potentially visible leaf nodes [25, 153]. In contrast to depth sorting of the scene objects, a projected screen size metric was used to prioritize object rendering in the Jupiter large model rendering toolkit [101]. As it turned out, the screen size metric is well suited for the selection of an appropriate level of detail, but it is inefficient for occlusion culling; a front-to-back sorting enabled the culling of up to two-times more geometry [197].

Further measurements on the occlusion status of leaf nodes indicated additional potential for more efficient utilization of occlusion culling for scenes with a high depth complexity in a cascaded manner. The approximately 10% front-most leaf elements in virtual endoscopy scenes are not occluded in most of the cases, hence they can be rendered without the costly determination of their visibility or occlusion status. Furthermore, the 40% farthest scene elements do not significantly change the visibility information stored in the depth buffer of the graphics system, because they are almost occluded. Therefore, their occlusion status can be established without taking their own occluder potential into account, which usually is a cheaper occlusion query and can be exploited by the faster multiple occlusion queries in HP's new VISUALIZE fx5/10 graphics subsystems. However, the distribution of the objects in a scene is very application dependent and needs to be established individually. In particular data from mechanical CAD have usually a deep visibility, while case elements occlude many objects of the model.

Note that front-to-back sorting is a highly efficient scene traversal strategy for visibility and occlusion culling[7]. In contrast to a frequent misconception however, it is not necessary. Only approaches similar to the painter's algorithm [65] (i.e., hierarchical polygon tiling [90]) actually do require a front-to-back, or back-to-front sorting. In all other cases, other strategies are possible [101, 197], but usually less efficient.

4.6 Hierarchical Model Organization

In the previous sections, we pointed out the need for a hierarchical model organization, which is difficult to derive for general polygonal models. Several papers on visibility and occlusion culling touch the topic of model organization. While some approaches require the designer to provide the model organization [194, 243], others employ decomposition methods which are application specific, such as a decomposition along the skeleton of a vol-

[7] In one experiment, we examined the culling efficiency with front-to-back sorting. We calculated the full, or *perfect occlusion potential* by rendering the complete model without any culling technique. Thereafter, we traversed the model hierarchy to test for occlusion against the already rendered model. As it turned out, front-to-back sorting achieves already 99.5% of the perfect occlusion potential on the investigated MCAD models.

umetric object [110], or a floor plan of a building [3, 206, 144]. However, these schemes cannot be applied efficiently to general models. Models built in Computer-Aided-Design (CAD) systems already include appropriate model organization information in the product data management system, due to the design process which uses hierarchical notions like grouping and replication.

A more general approach is to organize a polygonal model into regular spatial decomposition schemes, such as BSP-trees [76, 164, 90] or Octrees [92]. While these decomposition schemes produce good results on polygonal models extracted by the Marching Cubes algorithm from uniform grid volume datasets – which provide a "natural" decomposition on a Marching Cubes cell base –, these schemes run into numerous problems on general models. If a polygon of the model lies across a decomposition boundary, it must be either split into several parts in order to produce a disjunct representation of the bounding entities, or handled in another special way. Splitting polygons however, can increase the number of small and narrow polygons tremendously.

Significant work on model organization has been published in the field of collision detection. Methods based on oriented bounding boxes (OBB) were explored by Gottschalk et al. [87]. A bottom-up approach for the construction of a model hierarchy is suggested in [9] in which nodes representing small parts of the geometry are "merged" into higher hierarchy nodes. A similar approach is used in [126].

In [22], the spatialization functionality of OPT's OpenGL Optimizer package [188] was used to generate model hierarchies automatically. However, our experience from these experiments showed that these model hierarchies need to be tuned manually in order to get sufficient performance and motivated the work described in this section [153, 154]. To measure the decomposition quality, we use our modified HP flag approach, as described in the last paragraph of Section 4.2. This approach interleaves the occlusion culling tests of the bounding volumes (usually bounding boxes) and the rendering of the respective geometry, if the bounding volume has been established as potentially visible.

4.6.1 Polygon-based Hierarchical Bounding Volume Optimization (p-HBVO)

The polygon-oriented Hierarchical Bounding Volume Optimization (*p-HBVO*) method decomposes recursively a set of polygons into two model or scene entities. The selection of the optimal decomposition planes, which separates the model into two parts at each decomposition step is computed by evaluating a cost function based on the barycenter of each polygon (triangle). At each decomposition level, the individual polygons are assigned to exactly one model entity of that level. Consequently, no polygons are split, hence no new polygons are generated by this method.

Starting from the root node, at each decomposition step, the polygons

are sorted along all coordinate axes, where the barycenter of each polygon serves as sorting key. Based on these three ordered lists, we evaluate the potential decomposition planes along each axis for each entry in the respective list by splitting the sorted list of polygons into a *left* and *right* part. In contrast to pre-defined decomposition planes of the median cut scheme [120], we evaluate for each possible decomposition plane – defined by the entries in the lists – a cost function which approximates the costs of rendering the polygons of one of the two model entities, generated by the respective decomposition plane. By minimizing this cost-function over all possible decomposition planes, an optimal plane is obtained generating two new scene entities; one contains all *left*−polygons, the other one contains all *right*−polygons. The decomposition process terminates when either the number of polygons, or the scene depth exceeds one of the two pre-defined parameters: *Max_Triangles_Per_Decomposition_Entity* or *Max_Decomposition_Depth*. These parameters are specified by the user and supplied at the start of the decomposition process.

This cost function is identical to one which has already been successfully applied in ray tracing environments [163], since the objective is the same; both algorithms traverse the scene graph in a similar way to determine visibility. The costs of a model entity H, with children H_{left} and H_{right}, is given by:

$$C_H(axis) = \frac{S(H_{left})}{S(H)} \cdot |H_{left}| + \frac{S(H_{right})}{S(H)} \cdot |H_{right}| \qquad (4.2)$$

where $|H|$ is the number of polygons within hierarchy H, $S(H)$ the object-space surface area of the bounding box associated with subtree H, and $axis \in \{X, Y, Z\}$.

Overall, this algorithm generates well balanced scene trees with respect to their polygon load. Furthermore, polygons of individual objects are detected and clustered together. The highest culling performance was achieved with finer decompositions, which usually requires more occlusion culling tests, resulting in higher culling costs. If these culling costs are not compensated by lower rendering costs, it results in an overall lower rendering rate (see ventricular system in Table 4.3).

4.6.2 Octree-based Regular Space Decomposition (ORSD)

While the previous approach is able to handle arbitrary sets of polygons, some data sources inherently generate regular decompositions which can be exploited at much lower cost. Polygonal models extracted as isosurfaces from volume datasets (i.e., ventricular system dataset) consist of triangles which are arranged on uniform grid (see Section 1.2.1). The Octree-based Regular Space Decomposition method (ORSD) exploits these natural decomposition borders to generate a non-polygon splitting model decomposition.

After the construction of a minimum/maximum isovalue octree (or BONO [231], see Section 3.2), ORSD selects all relevant (contributing) cells which intersect with the isosurface. It counts the number of relevant cells (relevant cell load or RCL, see also Section 5.3.2) for each octree block and selects those octree blocks which are just below the specified RCL (their parent nodes are still above the RCL) as leafs nodes. This criterion is only a rough approximation of the actual number of extracted polygons, considering that each relevant cell represents between one and five triangles. In our experience however, RCL turned out to be sufficiently accurate. Figure 4.7c visualizes one of the generated scene trees, where the drawn bounding boxes are bounding the actual geometry, not the respective octant volume.

Overall, ORSD is a simple but efficient decomposition scheme which generates an adequate polygon load balance and bounding box sizes. As shown in the results, the indirect evaluation method (RCL instead of number of polygons) does not adversely affect the occlusion culling performance. However, ORSD is limited to regular grids, or to even rectilinear grid datasets, if the scene entities need to be disjoint.

4.6.3 SGI's OpenGL Optimizer (OPT)

SGI's OpenGL Optimizer (OPT) is a C++ toolkit for CAD applications that provides scene graph functionality for handling and visualization of large polygonal scenes. The decomposition method realized in OPT is similar to the construction of an octree; each model entity is split into eight equally sized model entities. This process is repeated recursively, until a certain threshold criterion for the iterated decomposition is reached. The octree-based spatial decomposition is a simple and efficient scheme. However, the OPT model organization mechanism decomposes space not by simply bisecting edges of a cube, as in an octree, but by choosing decomposition planes so that the rendering loads of the resulting parts are similar. As a result, the amount of geometry in each model entity on each side of the cutting plane is approximately the same. Polygons which are split due to the decomposition are distributed to the respective model entities. The main parameters that can be used to control the decomposition are hints for the lowest and highest amount of triangles (tri_{min}, tri_{max}) in each model entity at the leaf-level of the scene hierarchy. However, the decomposition algorithm only tries to meet these criterion, but is not bound to it.

In general, OPT generates model hierarchies with a well-balanced polygon load. However, the bounding boxes of the model entities are less suited for occlusion culling applications, because the cost function determining the model entities is obviously not optimized with respect to the volume or the screen-space area of the bounding boxes. We observed that the right-most branch of the scene tree frequently contained large subsets (bounding box volume size) of the model, even in the lower tree levels.

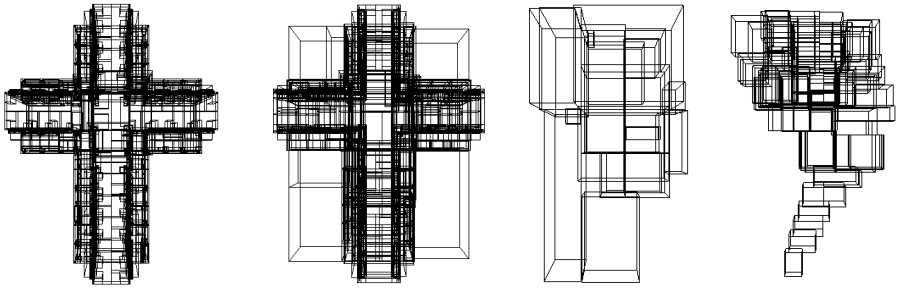
4.6.4 Evaluating the Model Organization Quality

In this section, we discuss the efficiency of the decomposition algorithms with respect to their occlusion culling-based rendering performance. All measurements are performed on an HP B180/VISUALIZE fx4 graphics workstation. The different polygonal datasets (see Table 4.3) represent typical scenarios of different application areas. Their scene trees only contain individual polygons (triangles) in order to evaluate comparable scenes. Two of the datasets are examined in more detail.

	Ventricular System	Cathedral	City
Grid Type	Uniform	Unstructured	Unstructured
Source	MRI	CAD	Modeler
#Triangles	270,882	416,763	1,408,152
#Inner / leafs nodes):			
p-HBVO	80 / 71	9 / 10	2722 / 2723
OPT	29 / 36	51 / 52	420 / 420
ORSD	6 / 28	n. a.	n. a.
Frame-rates/[fps]:			
No Culling	4.6	3.8	0.9
p-HBVO	12.3	12.4	14.0
OPT	13.6	7.8	11.8
ORSD	15.3	n. a.	n. a.
Rendering rates/[%]:			
p-HBVO	16.0	30.0	0.1
OPT	19.7	30.1	3.6
ORSD	22.4	n. a.	n. a.
Occlusion Time/[ms]:			
p-HBVO	30	4	54
OPT	16	27	33
ORSD	11	n. a.	n. a.

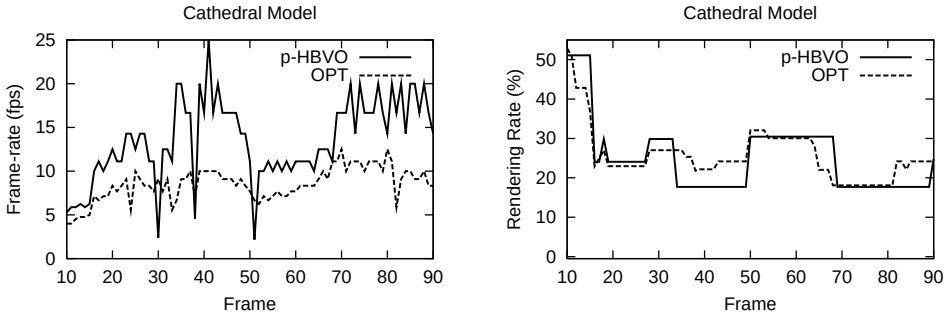
Table 4.3: Model overview – as gold standard for the speed-up due to occlusion culling, we show the frame-rate and rendering rates for the datasets with and without culling (view-frustum and occlusion). ORSD requires MarchingCubes-generated scenes, which are available only for the ventricular system. Occlusion time lists the costs required for occlusion culling (no view-frustum culling).

Scene trees of different decomposition granularities are evaluated for the respective costs of view-frustum culling, occlusion culling, and rendering (in frame-rate) of the not occluded geometry. Here, we present only culling and rendering performance on the scene trees with the best rendering performance. Culling performance is expressed as rendering rate, which gives the percentage of the geometry determined not occluded from the total model geometry. More detailed information on the evaluation can be found in [153].



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

Figure 4.7: Cathedral and ventricular system bounding box hierarchies generated by (a,d) p-HBVO, (b) OPT, (c) ORSD/OPT; the arts and pillars of the cathedral are well detected by p-HBVO (a); OPT only used a regular spatial decomposition (b). ORSD and OPT generated identical results for the ventricular system dataset (c) [153].



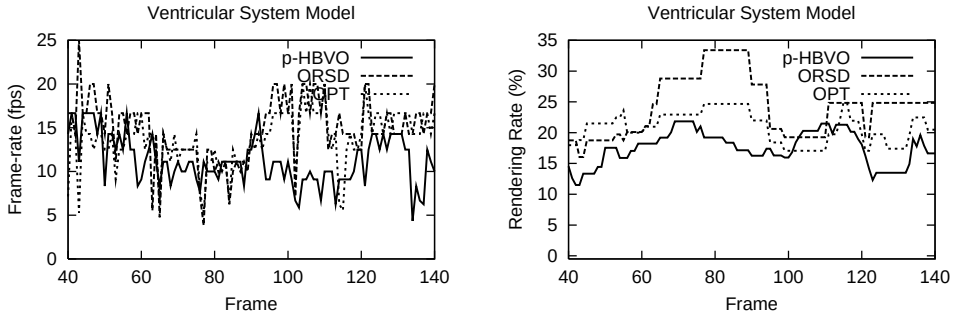
(a) (b)

Figure 4.8: Cathedral dataset, all frames of the path are shown; (a) frame-rate, (b) rendering rate, which is the percentage of remaining geometry after occlusion culling [153].

Cathedral Dataset:

This dataset represents the interior of a gothic cathedral, designed with a CAD system (see Table 4.3). Occlusion is limited to small parts of the model, because a large share of the polygons are visible from most viewpoints within the model. Figure 4.7a shows a very fine grain decomposition of the cathedral model using the p-HBVO approach which adapts very nicely to the structures of the model, such as pillars and arcs. In contrast, the decomposition generated by OPT (b) introduces very large bounding boxes, which do not adapt properly to the actual geometry.

The p-HBVO approach performed best on this dataset (see Fig. 4.8). This is due to the low culling costs, compared to OPT (see Table 4.3 and also [153]). The bounding boxes of OPT require a significantly higher time for occlusion culling compared to p-HBVO, which reduced the frame-rate severely.



(a) (b)

Figure 4.9: Ventricular system dataset, all frames of the path are shown; (a) frame-rate, (b) rendering rate, which is the percentage of remaining geometry after occlusion culling [153].

Ventricular System Dataset:

The second dataset is a polygonal model of the ventricular system of the human brain extracted from an MRI scan. We explore the dataset by moving through the lower part (Cisterna Magna) of the polygonal model. Most of the model structures through-out the walkthrough are located within the view-frustum, while the structures with the largest number of polygons (located in the upper part or lateral ventricles) were not visible due to occlusion. All polygons of this model are aligned on the uniform cell grid and are of approximately the same size. All three adapted algorithms were able to detect this "natural" decomposition boundary; only OPT generated approximately 15% additional polygons due to splitting operation between the grid points.

Figure 4.9 shows frame-rate and rendering rate of the evaluated algorithms. The most interesting detail is the low amount of time consumed by occlusion culling by ORSD, due to its coarse decomposition. The rendering rate of p-HBVO was approximately 25% better than the rendering rate of the ORSD approach. However, the finer decomposition (see Fig. 4.7d) introduced additional culling costs twice as much as for ORSD, resulting in a lower frame-rate (see Table 4.3).

City Dataset:

The final dataset is an artificial model of a city with 400 basic building models with some interior, which contains most of the polygonal complexity. Consequently, most of the polygons of this model are occluded. However, only the p-HBVO approach was able to subdivide all the interior into individual subdivision entities, hence resulting in a large number of nodes, a very low rendering rate, and high occlusion culling costs which were more than compensated by the small amount of potentially visible geometry (see Table 4.3 and Fig. 4.10). In contrast, OPT did not detect the interior objects

as well as p-HBVO; it used a coarse granular subdivision, which consequently increased the rendering rate.

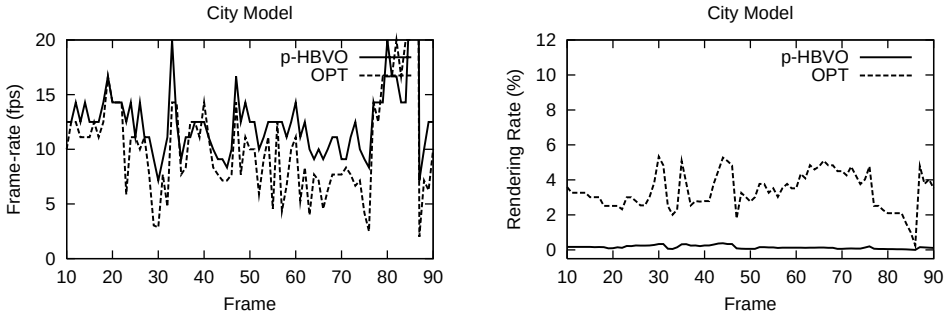


Figure 4.10: City dataset, all frames of the path are shown; (a) frame-rate, (b) rendering rate, which is the percentage of remaining geometry after occlusion culling [153].

4.6.5 Summary

Overall, the adapted model organization approaches were able to generate decompositions with faster rendering due to higher cull performance. This was achieved by reducing culling costs or by reducing the rendering rate of the dataset. On regular grid datasets, the basic ORSD approach produced a model subdivision which performed best, mostly due to the low time spent to establish occlusion or non-occlusion.

Generally, we observed that models with high occlusion do not require very fine decomposition (ventricular system dataset, p-HBVO vs. ORSD). On the other hand, a fine decomposition pays off if interior (thus completely occluded) objects are clustered in a scene entity (city dataset, p-HBVO vs. OPT). In contrast, models with low occlusion (cathedral dataset) do only benefit from finer decompositions, if the culling costs loss are compensated by the reduced rendering costs

Note that the p-HBVO approach originally built a binary scene tree. This resulted in deeper trees, hence more intermediate model entities, which in turn increased the time spent for occlusion culling significantly. Once this binary tree was re-built into a quad-tree representation, we achieved a frame-rate increase of approximately 20%.

In summary, hierarchical scene organization has a significant impact on the occlusion culling (how much is culled) and rendering performance (how many frame per second can be achieved). There is also a tradeoff between culling performance and rendering performance. A highly refined decomposition does not pay, if there is no **significant** smaller rendering rate (see ventricular dataset; p-HBVO vs. OPT); in these cases, the higher occlusion culling costs will consume all culling benefits.

4.7 Efficient Bounding Volumes

To reduce the complexity of interference tests (visibility queries in our case), a simple polygonal representation of the object to be tested is used. In the occlusion culling approaches previously introduced (and many other approaches in the various similar and related research fields, such as ray tracing [86, 163], collision detection [213], and visibility culling [92, 243, 22]), axis-aligned bounding boxes (AABBs) were used to approximate the geometric shape of the objects. However, in many cases this approximation fills a much larger volume in object-space, and a much larger screen area – once rasterized into screen-space – than the actual geometry. This results in *false positive* interference results[8], which can increase the computational load significantly.

Other bounding volume primitives are used, such as bounding spheres [226, 113], or oriented bounding boxes [175, 87, 9], where the spanning axes of the bounding box are oriented according to the shape of the object, thus generating a tighter approximation of the original shape than an AABB. While OBBs perform better for collision detection than AABBs, the benefits for image-based occlusion culling are significantly smaller. This is mainly due to the fact that the rasterized screen-area of an OBB is still of similar size as for an AABB. In 1996, Klosowski et al. [98, 126] proposed a collision detection scheme using discrete orientation polytopes (k-dops), which enabled faster collision tests than OBBs. k-dops are also used in visualization systems to compute a level-of-detail in a multi-resolution representation [7, 48]. Essentially, k-dops are an approximation of an object by computing bounding planes of an object along k orientations (k/2 directions) [120]. This generates an AABB for a 6-dop, or an AABB where the edges and corners are cut to the object surface using a 26-dop.

In this section, we explore the use of k-dops for occlusion queries. They enable a close approximation of the geometric shape of an object which in turn generate a smaller number of false positives of occlusion tests, thus reducing the rendering load.

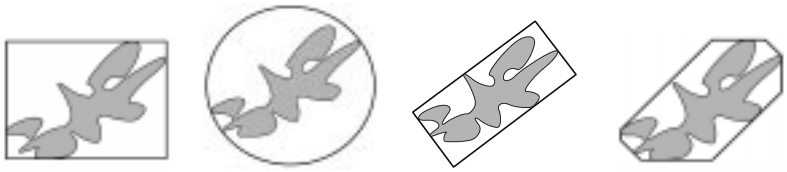
4.7.1 Discrete Orientation Polytopes (k-dops)

A discrete orientation polytope (k-dop) is composed of k facets which are determined by a set of k halfplanes, where two halfplanes at a time are parallel to each other. The halfplanes in turn are determined by k normal vectors – which determine the orientation – and a distance value. More information on k-dops can be found in [125, 158].

Four different, potentially conflicting objectives need to be considered for choosing an appropriate bounding volume primitive; (a) approximation quality, (b) computational expenses, (c) rendering complexity, and (d) interference complexity. For image-based visibility purposes, only (a)-(c) are

[8] A false positive occlusion test result determines an object as not occluded – based on its bounding volume –, while in fact it is occluded.

of interest, since interference computation is usually only meaningful in object-space.



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

Figure 4.11: Approximations of an object by four bounding volumes: (a) An axis-aligned bounding box (AABB), (b) a sphere, (c) an oriented bounding box (OBB), and (d) a k-dop (where $k = 8$) [125].

Generally, the object approximation qualities of AABBs and spheres are poor, since they disregard the actual geometric shape of the object; only minimum and maximum values are considered. This results in empty regions in particular at the edges and corners of the AABB, or the periphery of the sphere (see Fig. 4.11a and b). An OBB or the convex hull of the object in contrast provide a significantly better approximation quality (see Fig. 4.11c), unfortunately at much higher computational expenses[9]. k-dops in turn provide a sufficient approximation (see Fig. 4.11d) at reasonable costs. In a way, k-dops are an extension to AABBs; the empty regions at the corners and edges of an AABB are cut off by halfplanes of a 26-dops, thus generating the tighter approximation. Next to the six halfplanes to compute the AABB, this requires 12 halfplanes to cut off the edges, and eight halfplanes to cut off the corners.

The computational expenses of a k-dop are comparable to the expenses for an AABB. The computation of a 26-dop involves the dot-product of each object vertex with the 13 vectors (26/2 directions, or 26 halfspaces) which define the 26-dop: (1,0,0), (0,1,0), (0,0,1), (1,1,0), (1,0,1), (0,1,1), (1,-1,0), (1,0,-1), (0,1,-1), (1,1,1), (1,-1,1), (1,1,-1), (1,-1,-1). Similar to the minimum/maximum comparison for computing an AABB, this can also be simplified for each different k (see [125] for details). To construct the actual polygonal representation of the 26-dop, the 26 halfspaces are mapped into dual space, where each halfspace is represented as a vertex, and a convex hull is constructed from this vertices. The faces of the convex hull are mapped back into primal space, as vertices – including the connectivity information – which are used to construct the polygons which represent the 26-dop.

Rendering complexity is usually measured in object-space triangles (geometry), used to model the bounding volume, and image-space pixels, which cover the screen-area of the rasterized triangles. In terms of geometric rendering complexity, an AABB or an OBB use only up to six polygons, while

[9] If the model design is in a fairly stable state, the computational costs for the computation of a convex hull might be negligible. Frequent design updates, however, involve a significant share of costs for the re-computation of the bounding volume. These costs can potentially dominate the design process.

a reasonable tessellated sphere, or a convex hull use much more polygons. In contrast, a k-dop requires only up to k polygons; it therefore represents a good compromise between the geometric complexity of an AABB and a convex hull. However, the difference of geometric complexity is not relevant, since the synchronization latency introduced by one HP flag-based occlusion query hides the rendering amount for approximately 190 triangles of 25 pixels [185]. More important for an image-based occlusion culling approach is the screen-area covered by the rasterized bounding volume, which is quite large for AABBs, spheres, and also relatively large for OBBs. The least screen-space is covered by a convex hull, which is as mentioned earlier quite expensive to compute. Again, a k-dop is a good compromise which occupies only a relatively small screen area and requires only a limited number of polygons (see Fig. 4.11d).

Overall, k-dops provide a good approximation quality at a low computational and rendering complexity, which makes k-dops an excellent candidate for a bounding volume.

4.7.2 k-dops for Occlusion Culling

For our evaluation of k-dops as bounding volume for occlusion culling, we perform several experiments where 26-dops are compared with AABBs (or 6-dops). All experiments are performed on an HP J7000 workstation (4 GB, four HP PA 8500@440MHz CPUs, only one is used) using a VISUALIZE fx6 graphics subsystem.

model / #frames	#triangles / #objects	rendering rate(%)		frame-rate (fps)	
		k-dops	AABBs	k-dops	AABBs
Engine	150,248	59.7	73.7	4.5	3.8
41	149	48.6	64.5		
Screw Driver	156,424	32.9	47.3	7.8	5.5
36	83	20.9	37.0		
Racing Car	746,827	34.8	39.9	1.6	1.4
41	306	27.3	32.1		
City	1,403,096	1.6	11.2	16.8	3.3
201	420	1.6	11.3		
Angiography	1,817,731	2.8	3.0	11.3	10.7
1034	278	3.9	3.9		

Table 4.4: Model overview: culling performance is shown as the average rendering rate recorded over a sequence of frames. Rendering rate is the percentage of remaining geometry after occlusion culling. Note that the triangle (upper row) and the object percentages (lower row) are given.

Table 4.4 shows an overview of the models used for the experiments[10]. Several models from mechanical CAD (MCAD), a modeling system, and from medical scanners of various sizes are used. Every model is stored in a hierarchical tree representation.

[10]Rendering using OpenGL was not optimized for frame-rates. We used display lists of lit triangle meshes, but

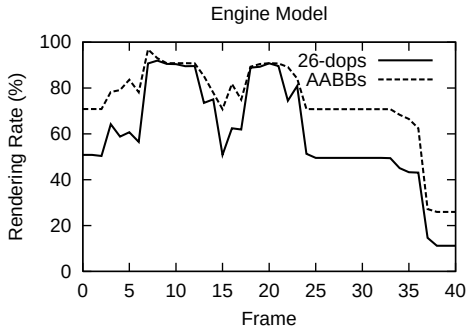


Figure 4.12: Engine dataset – the engine dataset shows a better k-dop-culling performance for frames where the camera is facing the outer hull of the engine (frames 0-6,15,16,24-40). The final frames show measurements of close-ups into the dataset

For each of these models, we perform the HP flag-based hierarchical occlusion culling approach as introduced in Section 4.2 with a standard full scene tree traversal, including the depth-sorting of the leaf nodes (see Section 4.5). The front-most n leaf nodes of the leaf node list are rarely occluded, hence we render these nodes without any occlusion test. For each model, we move through a list of arbitrary view-points, which represent typical view situations of the datasets, such as rotations, close-ups, and walkthroughs of the geometry (angiography and city models only). The results are accumulated and visualized in gnuplot diagrams.

Engine

The Engine dataset [100] is an MCAD model of a truck engine, after applying a cutting plane which removes half of the engine geometry, exposing the interior engine parts [100]. The 149 individual parts are organized in a tree hierarchy, where among other objects each of the four pistons and combustion chambers are composed into one second level node of the hierarchy (see Fig. 4.13b-d). The measurements are performed over a camera path of 41 frames of rotations and close-ups of the model.

On average, almost 40% of the geometry could be culled using k-dops, compared to only 26% using AABBs. Most of the additional k-dop-based culling takes place when the camera is facing the outer structure of the engine, while most of the engine parts are visible when facing the interior part on the clipped side (see Fig. 4.12). This behavior is caused by the fact that most AABBs intersect through the occluding geometry of the exterior hull parts (see Fig. 4.13a), while the k-dop bounding volumes approximate the geometry much tighter (see Fig. 4.13b). Some of the combustion chamber/piston parts for example are determined visible using AABBs, while they are classified as occluded using k-dops (see Fig. 4.13).

no triangle strips or vertex arrays.

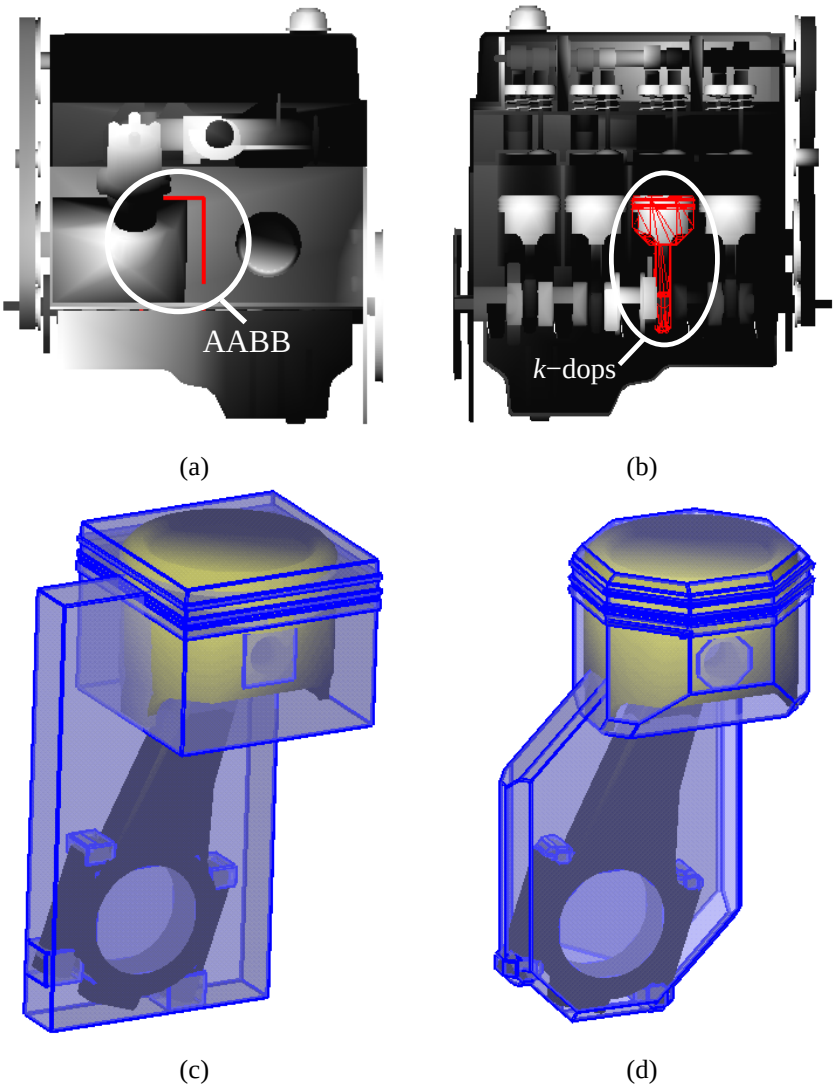


Figure 4.13: Engine model: (a) Intact outside view of Engine; the (red) AABB of a piston is visible through the occluding parts; the k-dop is not visible. (b) Clipped view of Engine; piston with k-dops (red). (c) Piston from engine with AABBs (blue), and (d) with k-dops (blue).

Screw Driver

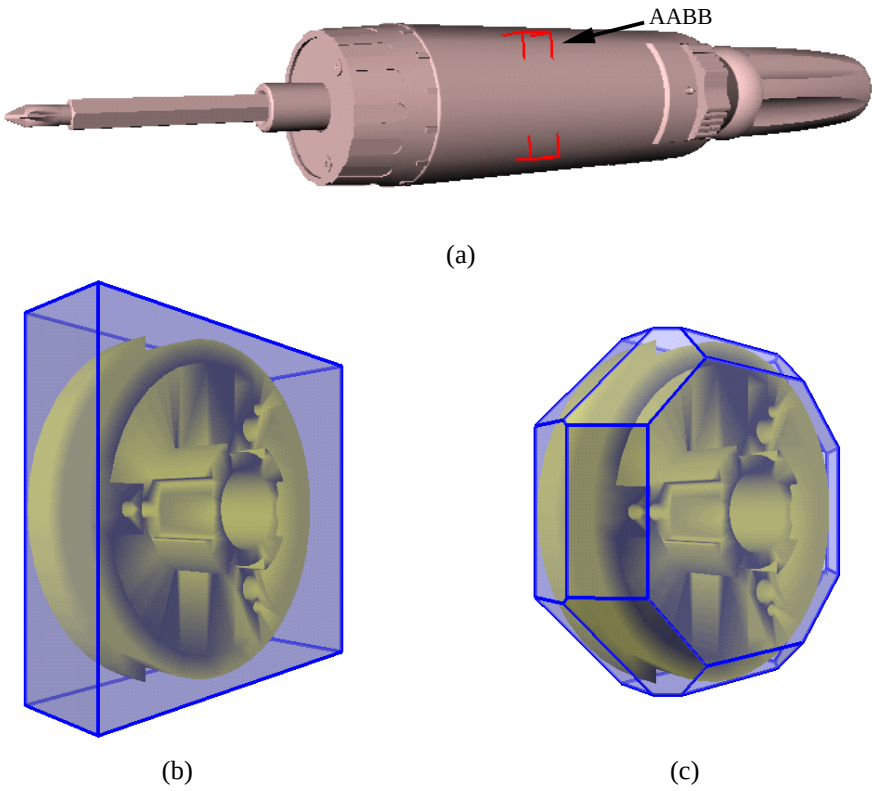
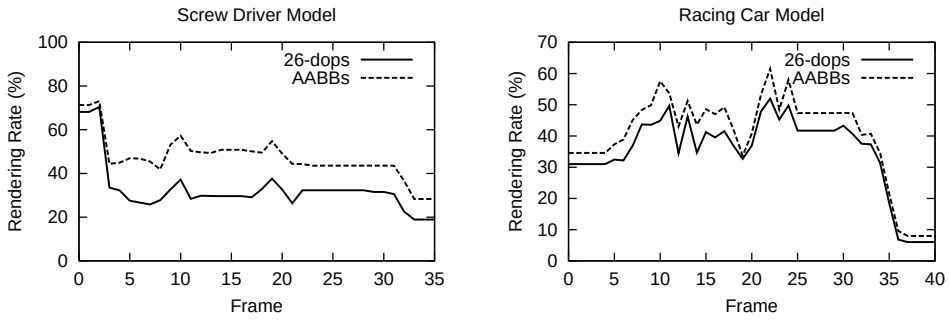


Figure 4.14: Screw driver: (a) Full dataset with (red) AABB of a motor part; the k-dop is not visible. (b) Motor part with AABB (blue), and (c) with k-dop. (blue)

The screw driver datasets [49] represents a servo screw driver model from an MCAD system [100]. Average culling performance is measured over a sequence of 36 frames of rotations and close-ups. 83 components are organized in a relatively flat tree hierarchy where its case is composed of two parts, occluding most of the interior geometry. However, the axis-aligned bounding boxes as bounding volumes frequently extend through the case parts, resulting in a not occluded visibility status. If k-dops are used instead as bounding volumes, the approximation to the actual geometry is much tighter. This can be seen in a part of the screw driver (see Fig. 4.14), which is determined occluded using k-dops, and determined not occluded using AABBs. The overall occlusion culling results can be see in Figure 4.15a, where more than twice as much geometry can be culled using k-dops (see also Table 4.4).

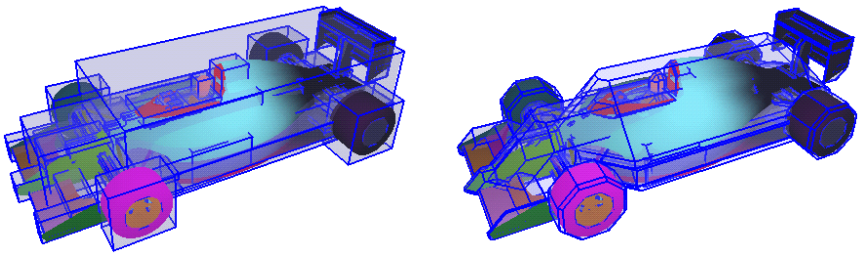
Racing Car

The racing car is a complex MCAD dataset [100] composed of 306 individual car parts organized in a flat tree hierarchy [100]. The measurements are



(a) (b)

Figure 4.15: (a) The screw driver dataset includes two case parts which cover most of the geometry. (b) The racing car dataset; the final frames show measurements of close-ups into the dataset.



(a) (b)

Figure 4.16: Racing car: (a) All car parts – in different colors – with the associated AABBs (blue), and (b) with the associated k-dops (blue).

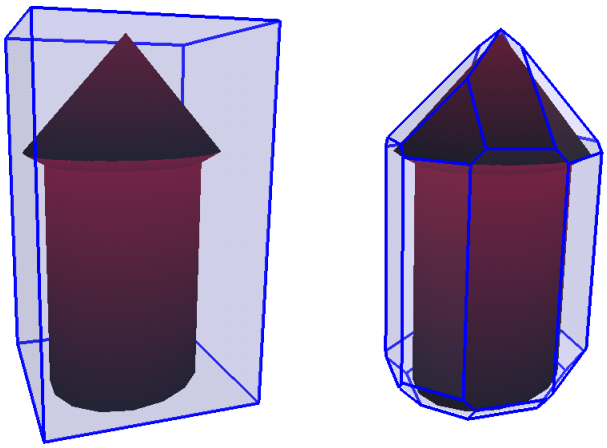
performed on a camera path of 41 frames of rotations and close-ups.

On average, an additional 5% of the geometry was culled using k-dops as opposed to AABBs (see Fig. 4.15b). The major reason for the limited improvement using k-dops is that the k-dops-based bounding volumes of many car parts still intersect through the facing of the car, although the overlapping volume is significantly smaller than with AABB-based bounding volumes (see Fig. 4.16).

City

The city model is an artificial scene generated by a modeling system (see Fig. 4.4a and d) [22]. All buildings have a very simple geometry, which can be approximated quite well by a k-dop (see Fig. 4.17). Every building contains some interior geometry, which increases the total polygonal complexity significantly. The model hierarchy is generated based on a quadtree scheme, which subdivides the model regularly. The camera follows a walk-through path of 201 frames through the city, where the AABBs expose a relative deep visibility.

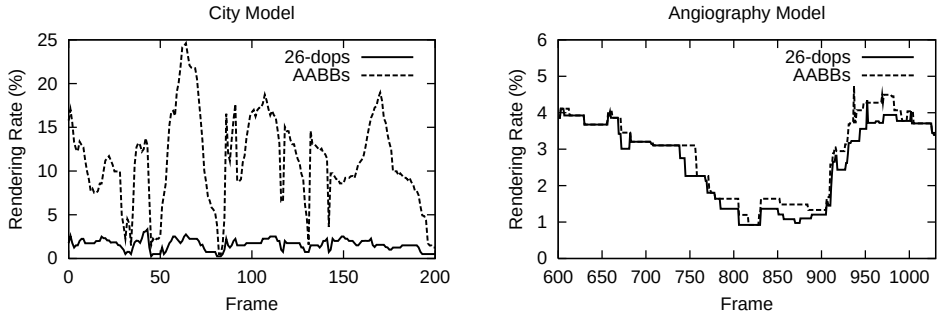
Figure 4.18a shows that the determined occlusion status of the building



(a) (b)

Figure 4.17: Building from city model: (a) with AABB (blue) and (b) with k-dop (blue).

objects change significantly while using k-dops as bounding volumes; on average, six times more geometry can be culled in comparison to an AABB-based occlusion culling approach (see Table 4.4). The main reason for this performance gain is that the geometry of many buildings is not visible from most view-points, while the AABB of those buildings are visible. The k-dops which are approximating the shape of the building better avoid these false positive results of the occlusion queries.



(a) (b)

Figure 4.18: (a) The city dataset exposes a large culling potential. However, only a limited amount can be exploited using AABBs, while the geometry determined as visible can be reduced to a 6th using k-dops. (b) The angiography datasets does not reveal an additional culling potential. AABBs and k-dops provide similar culling performance.

Angiography

The angiography model (see also Section B.4) is an extracted isosurface of an arterial blood vessel from a rotational angiography scan of a human head [31]. The isosurface of the original volume dataset is extracted us-

ing an octree-based MarchingCubes approach [13, 142], which generates a large number of small triangles. For the camera path, we perform an endoscopic walkthrough of the arterial blood vessels of about 1000 frames. The resulting culling data from a section of this camera path can be seen in Figure 4.18b.

The objects in this model are composed from octants of a specific granularity of voxels which contain the isosurface [25]. In many cases, they have a cubic shape which is already well approximated by an AABB, hence no significant improvement can be expected using k-dops. Our measurements confirm this consideration; less than 0.2% of the geometry can be additionally culled using k-dops in comparison with AABB (see Table 4.4). Over the full sequence of more than 1000 frames, we save only 11 seconds.

4.7.3 Summary

In the past paragraphs, we presented measurements on different datasets from various sources such as MCAD, medical scanners, and modeling systems. For most datasets, we achieved additional culling using k-dops in contrast to AABBs. As the major reason for the additional culling performance, we identified the tighter approximation of the k-dop-based bounding volumes. All increased culling performance also resulted in an increased rendering performance measured in frame-rate, ranging from a 5.6% increase (angiography model) to a 500% increase (city model).

Typical datasets from MCAD consist of interior model parts, i.e. engine parts, and case elements that are occluding the interior parts. However, AABBs frequently intersect the outer hull and cause a false positive occlusion result, indicating visibility. k-dops approximate the geometry of the model objects much tighter, thus reducing the number of false positive visibility hits. Nevertheless, if the occluding case elements enclose the model objects too tightly, a k-dop-based bounding volume will still generate a false positive visibility hit, as it happened with the racing car dataset.

Apparently, a k-dop-based occlusion culling algorithm does not provide (significantly) better performance than AABB-based algorithms on MarchingCubes generated polygonal models (see angiography model, Fig. 4.18b). These kinds of models are composed of small triangles which we combined into octree blocks which are well approximated by AABBs; k-dops are not a significantly tighter approximation, thus (almost) no improved performance can be gained. Furthermore, the endoscopy-mimicking camera path limits the visibility already to a small amount of visible polygons. However, other experiments with exterior camera paths (and other MarchingCubes generated models) – which are not documented here – showed that almost no additional culling performance can be gained with k-dops.

Finally, the angiography model showed that the additional polygonal complexity of the k-dops is negligible, since every occlusion culling query using the HP flag requires approximately the same time as the rendering of 190 triangles of a screen size of 25 pixels [185], which is much more than

needed for a 26-dop.

4.8 Hardware Support for Occlusion Culling

There is a long “tradition” for hardware support to solve the visibility problem in computer graphics. Most important is the z-buffer approach [198, 36] which solves the hidden-line or hidden-surface problem. A *z-query* of the Denali GB graphics on the Kubota Pacific Titan workstation was used by Greene in the hierarchical z-buffer approach [92]. Here, we outline two extensions to graphics subsystems to provide fast and detailed information on the occlusion status of objects in a computer graphics scene. While the first extension provides support for quantifying occlusion of an object, the second extension enables culling of small triangles and pixel groups on top of a standard occlusion culling approach. For more details on the hardware implementations, please refer to [21] and [152, 151].

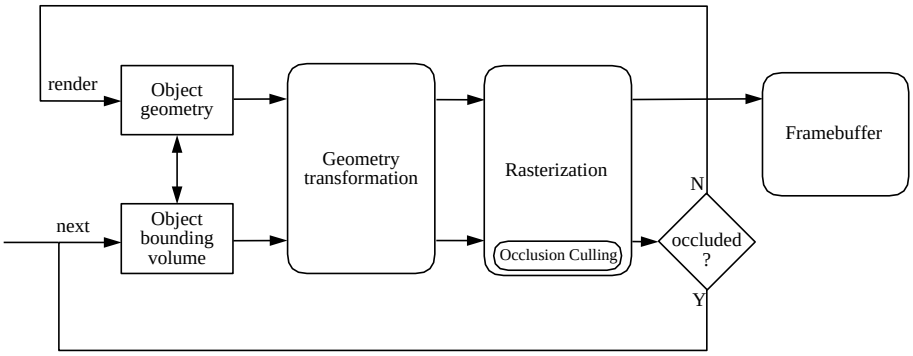


Figure 4.19: Graphics Pipeline with Occlusion Culling; The bounding volume of an object is transformed and rasterized (without actually contributing to the framebuffer). Occlusion information is generated in the rasterizer, during the depth test.

4.8.1 Hardware Support for Quantitative Occlusion Queries

The main source of performance problems of software-based occlusion culling algorithms like the hierarchical z-buffer [92] or virtual occlusion buffer [22], is the framebuffer-like design of the z-buffer. Most of the effort is spent searching for the changes due to non-occlusion[11], while this information can be easily obtained by directly catching the write enable signals of the depth buffer test, like the HP flag [99], the *Occlusion Unit* by Bartz et al. [21], or more recently, the SGI instrument extension of the Visual PC [189]. In contrast to the z-buffer, this information is data-sensitive – the actual changes are listed as results of the query – and straightforward to process. However, the provided information lacks details about where and how much of an object is visible. This information is essential to determine how to deal with

[11] In the case of the hierarchical z-buffer, additional significant effort is spent updating the z-pyramid [103].

partially occluded objects. In Section 4.4.3, we introduced quantitative occlusion culling, which depends on quantitative information on the visibility to decide if a fully detailed object is rendered, or a coarser level-of-detail of this object is chosen. To provide this quantitative information, we proposed an extension to the graphics subsystem, the *Occlusion Unit* embedded in the OpenGL pipeline (or any other rendering pipeline) [21]. A similar functionality is implemented in the recent VISUALIZE fx5/fx10 graphics subsystem of Hewlett-Packard. Figure 4.19 shows the modified graphics pipeline. The object geometry is rendered, depending on the occlusion status of the associated bounding volume.

(a) (b)

Figure 4.20: A rasterized box: (a) shows a visual representation of the pixels counted by the *Projection Hit Counter* (orange), while (b) shows the not occluded pixels behind the tree objects as counted by the *Non-Occlusion Hit Counter* (yellow).

The core of the extension are two features which provide counters to quantify the not occluded parts of the tested objects, and one extension which enables the adaptive decomposition of the occlusion sensitive areas.

- The **Projection Hit Counter (PHC)** counts the number of pixels of the projection of the scan-converted object to be rendered. Projection hits together with non-occlusion hits can provide information about how much of the projection of an object is not occluded (see Fig. 4.20a).
- The **Non-Occlusion Hit Counter (NOHC)** is used to quantify all not occluded pixels of the scan-converted object. This provides simple analysis of the non-occlusion hits; how many and on which area of the viewport (Multiple Occlusion Tiles, see below) (see Fig. 4.20b).
- With **Multiple Occlusion Tiles**, the complete viewport can be limited to smaller portions, or refined into a hierarchy of tiles. Alternatively, multiple occlusion tiles can split the area of interest into a multi-resolution non-occlusion hit representation to run a hierarchy of occlusion tests (i.e., a quadtree-like representation of occlusion in a given model (see Fig 4.21)). Multiple occlusion tiles can also be used to determine the visibility of portals in a PVS-like approach [144].

This new functionality modifies the quantitative occlusion culling Equation 4.1 which now computes the actual relative contribution to the framebuffer, as defined in Equation 4.3 (or only the first term of Equation 4.1 for the perspective view case).

$$QOC_{hw}(Obj) = \frac{NOHC(Obj)}{PHC(Obj)} \tag{4.3}$$

Alternatively, the new ratio can be multiplied with the previous Equation 4.1, if depth and viewport size are of interest. Depending on the computed value, an appropriate rendering strategy can be chosen.

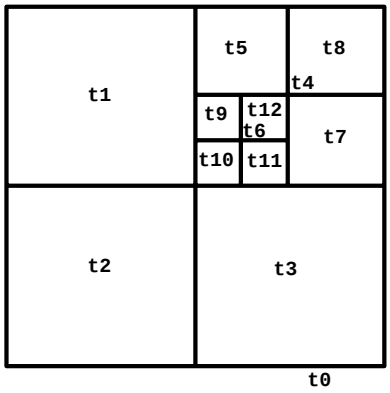


Figure 4.21: Quadtree of occlusion tiles $t_0..t_{12}$ is used [21].

The information on the occlusion status – which is returned via an OpenGL data buffer – of an object contains the number of not occluded pixels of the object (provided by the NOHC), and the number of rasterized screen pixels of the object (provided by the PHC, see Fig 4.20a). The data buffer itself is organized according to the defined occlusion tile arrangement (i.e., a quadtree as in Fig. 4.21); one hit record is generated for each available occlusion tile with a non-occlusion hit. If a hierarchical organization is used (i.e., a quadtree), the returned hit records are organized in the same hierarchy. Furthermore, the position of the not occluded pixels is given as a screen bounding box of minimum and maximum screen coordinates (clamped to the occlusion tiles), and the minimum and maximum depth values. In a detailed information mode, a list of the specific not occluded pixels can be queried. The API's access to the information is managed in a similar fashion like the OpenGL selection buffer, which provides information up to an allocated limit for picking hits in OpenGL. For details on the API specification and on the hardware implementation, we refer the interested reader to [21].

4.8.2 Visibility Driven Rasterization

In recent years, memory bandwidth and access problems have emerged as a major challenge for computer graphics hardware. *Richer pixel* operations

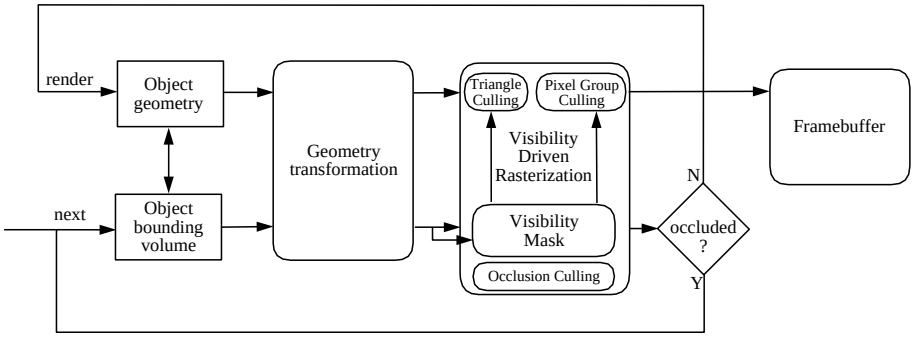
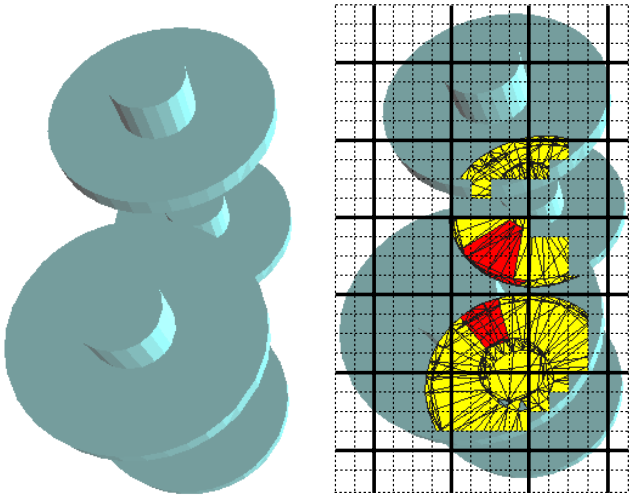


Figure 4.22: Graphics Pipeline for Visibility Driven Rasterization; The visibility mask is set during rasterization of the bounding volume. If the object is potentially visible, the information in the visibility mask is exploited for the culling of triangles (trivial reject I & II) and groups of pixels.

[124], such as multi-texturing, per-pixel shading, or advanced filtering techniques have even increased the performance requirements for this memory bottleneck. While most approaches to reduce the bandwidth requirements (such as mesh compression and reduction) only address the geometric or object-space complexity, occlusion culling also reduces the per-pixel complexity, hence also accesses to the depth buffer or other entities of the framebuffer. Most occlusion culling approaches only cull on object level (see Section 4.2), where objects are usually composed of several hundreds, or thousands of triangles. While this culling operation is quite effective, a significant culling potential still remains to be exploited on triangle and pixel level. In this section, we outline a modification to the rasterizing stage inside of a graphics subsystem which reduces the remaining pixel load on top of a standard image-space occlusion culling algorithm. For a detailed presentation of the approach, including the hardware modifications, we refer the interested reader to [150, 152].

The core of visibility driven rasterization (VDR) is a two-level *visibility mask*, which collects coverage (visibility) information during rendering of a set of polygons – i.e., an AABB, a k-dop, or any other kind of hull of the current object. This information is later exploited when the actual object geometry is rendered, possibly using richer pixel operations (see Fig. 4.22). The visibility mask itself is implemented using either a large register file or SRAM within the rasterizer, on-chip. Its actual size is a tradeoff of storage (chip real estate) and resolution (culling efficiency). A full resolution visibility mask of a viewport of 1024×1024 pixels would require an unrealistic 128K Bytes storage, while a low resolution visibility mask (i.e., 4×4 entries) misses most of the culling potential. Each of its entries is denoted as a *visibility mask tiles* (VM tile), where each VM tile represents a viewport area within the visibility mask. In a full resolution visibility mask, a VM tile would represent one viewport pixel; in a 4×4 visibility mask, a VM tile represents a 256×256 pixels viewport area (assuming a 1024×1024 pixel viewport). The second level of the visibility mask combines several



(a) (b)

Figure 4.23: Four wheel hubs of a cotton picker model in a *Visibility Mask*: (a) Two of the wheel hubs are partially occluded by the front wheel hubs. (b) The culled triangles of the partially occluded wheel hubs are colored in red, the culled pixel groups are colored in yellow. The black grid indicates the applied two-level visibility mask [151, 150].

VM tiles to a *group of VM tiles*. In our example of four wheel hubs from a cotton picker model, we show a two-level visibility mask, where 4×4 VM tiles are combined to a tile group (see Fig. 4.23).

After the visibility information of an object is collected, its triangles are only rendered if they pass two occlusion tests. The first test – trivial reject I – culls triangles which are located within a single occluded VM tile; the second test – trivial reject II – culls all triangles which are located within a single occluded VM tile group. In Figure 4.24a, all polygons except triangle B would be culled due to trivial reject II, if all four tile groups are completely occluded in the visibility mask; all polygons emerge over more than one VM tile, therefore no polygon is culled due to trivial reject I.

In addition to triangle culling, groups of pixels – based on the VM tiles of the visibility mask – are culled, if the respective VM tiles are completely occluded. In Figure 4.24b, a partially occluded triangle overlaps three tile groups; two of the tile groups are completely occluded, one is only partially occluded. All pixels of the triangle which are associated with occluded VM tiles are culled due to the pixel group culling.

In [152, 151], we performed several experiments to evaluate the appropriate resolution of the visibility mask, based on the different VM tile sizes. As it turned out, sizes of 16×16 (512 Bytes SRAM) or 32×32 (128 Bytes SRAM) are of a high practical value. Up to 40% of the remaining triangles can be culled in addition to standard view-frustum and occlusion culling. Additionally, between 40% and 55% of the remaining pixels can be skipped. This results in a significantly reduced rasterization load, which in turn reduces the required bandwidth tremendously, and potentially doubles

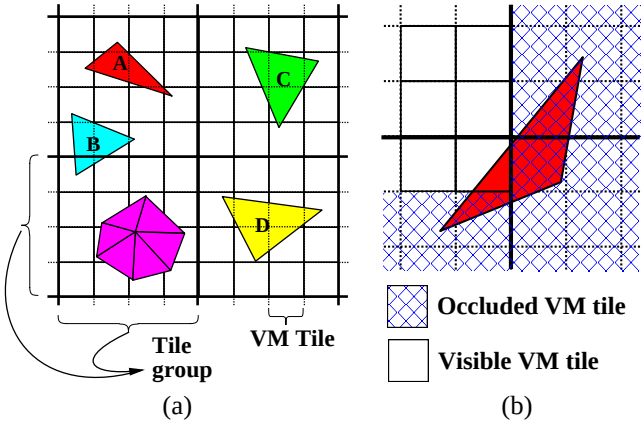


Figure 4.24: (a) Four tile groups; each VM tile represents the area of $n \times m$ pixels on the screen. The visibility information of a tile group (16 VM tiles, 4×4) is stored in one entry of the visibility mask. (b) The triangle covers several non-visible VM tiles of umpteen tile groups [151, 150].

the frame-rate.

The main advantage of the visibility mask hierarchy (measured by the second triangle test (trivial reject II)) is the broadening of the effective VM tile size range, which is important to reduce the dataset dependency of the VM tile size.

4.9 Summary

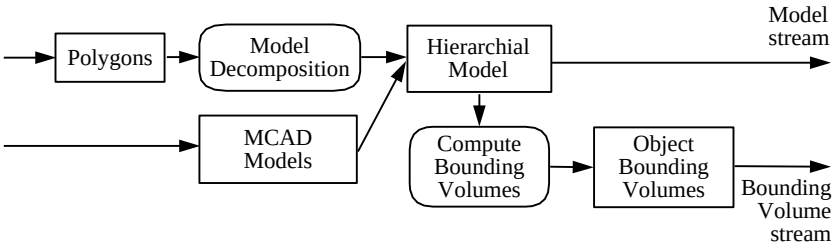


Figure 4.25: Occlusion Culling Pipeline: Preprocessing of the polygon data to generate a hierarchical model representation, and the respective bounding volumes.

In this chapter, we described multiple, hardware- and software-based techniques which are combined into a hierarchical visibility and occlusion culling pipeline. This pipeline is split into a preprocessing step (see Fig. 4.25) and an interactive culling step (see Fig. 4.26) which performs the actual culling, where hardware-based techniques replace the occlusion culling stage of the culling step.

If interactive performance is required for large polygonal models and no special assumptions are available to define a specific optimized occlusion culling approach, graphics hardware support is necessary. However,

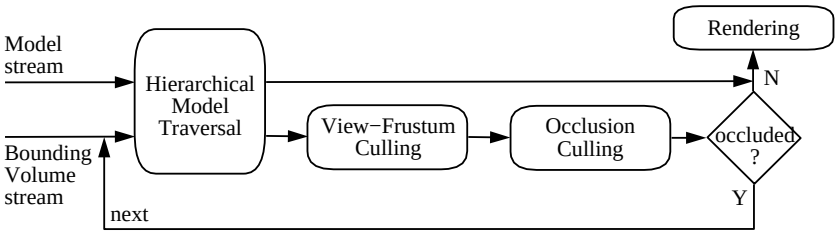


Figure 4.26: Occlusion Culling Pipeline: Interactive part of the occlusion culling pipeline. The model is hierarchically traversed; the bounding volume of each traversed node is tested for intersection with the view-frustum, and – if visible – tested for occlusion.

we also showed that a hardware-accelerated occlusion culling approach still leaves space to improve the rendering performance. Specifically, we introduced techniques for efficient scene traversal, techniques to derive an efficient hierarchical organization, and bounding volume primitives, which perform better than the standard axis-aligned bounding boxes. The powerful combination of these methods enables a highly effective occlusion culling framework for arbitrary polygonal scenes.

Part II

Applications of Large Model Visualization

Chapter 5

Applications in Virtual Medicine: Virtual Endoscopy

In this chapter, we introduce the VIVENDI system for virtual endoscopy [25] which uses several techniques presented in the previous chapters. Applications of VIVENDI are presented in Appendix B.

The data used for VIVENDI are computed by medical scanners, such as Computed Tomography (CT or CAT scanners), Magnetic Resonance Tomography (MRT or MR scanner), rotational biplane X-ray (or rotational angiography), 3D Ultrasound, Positive Emission Tomography (PET), and many more. These data acquisition techniques are described in Appendix A.

5.1 Introduction

Minimally-invasive procedures are of increasing importance in medicine because they have less deleterious effects on the patient. In particular, these procedures are used in gastroenterology, surgery, neurosurgery, radiology, and many other fields. However, several drawbacks are associated with minimally-invasive procedures. They are usually very unpleasant for patients, they are expensive (although they are still cheaper than “traditional” open surgery), and some areas of interest cannot be reached by the endoscope or catheter (due to folds and plaits). Especially in (neuro-) surgery, these procedures lack the fast access in case of serious complications, such as strong bleeding. Therefore, careful planning and realization of these procedures is essential, in order to avoid such complications. This problem aggravates, because handling and control of many of these endoscopes is very difficult, mainly due to limited flexibility of and limited field of view through the endoscope, a very limited depth perception, and the sensitive nature of the brain tissue.

In contrast, virtual endoscopy is a convenient alternative. It is based on a 3D scan of the respective body region. Examples for these scans are CT (Computed Tomography) scans of the abdominal area, MRI (Magnet Resonance Imaging) scans of the head, or rotational angiography of blood vessels at the skull base. Based on the resulting volumetric data, the organs

of interest are visualized and inspected from interior ("endo") view-points. Depending on the original endoscopic procedure, which is mimicked by virtual endoscopy, different goals can be achieved. These goals range from

- teaching: providing unusual insights into the anatomy of living patients,
- diagnosis: inspecting organs for (shape) defects, indicating unusual organ geometry,
- intervention planning: providing insight into the potentially complicated and non-standard anatomy of the patients and the individual organ location, and
- intra-operative navigation: currently, the position of a "real" endoscope is tracked by an infrared-based 3D navigation system and mapped into the image stack acquired previous to the operation. With virtual endoscopy, this position and orientation information can be exploited to provide a coupled visualization of optical and virtual endoscopy. In particular the virtual endoscopy can provide information which is not available to the optical endoscope, due to the limited flexibility and field of view.

The next sections review related work in the field of virtual endoscopy (Section 5.2), and discuss the VIVENDI system for virtual endoscopy (Section 5.3). Finally, several virtual endoscopy applications that are conducted with VIVENDI are presented (Section B.1- B.4).

5.2 Related Work

Research on virtual endoscopy is one of the most active areas in virtual medicine. In this section, we briefly introduce some of the related, published work in the field. The various developed methods of virtual endoscopy have been applied to virtual colonoscopy [218, 110, 134], bronchoscopy [219, 63, 171], ventriculoscopy [6, 27], and angioscopy [52, 32, 85, 31].

Different rendering techniques are used to provide sufficient visual quality and/or interactivity. Standard graphics hardware is used to render surface models [220, 143, 110, 25], extracted with the Marching Cubes algorithm [142]. However, the high geometric complexity of the extracted organ models exceed the interactive rendering capabilities of most of these hardware accelerators, thus requiring either high-end systems [110, 220], algorithms to reduce the rendering complexity [25], or to relinquish interactive performance [32]. In contrast, volume-rendering techniques are used, partially for better visual quality, partially for interactive speed [190, 109, 52, 241, 85]. Unfortunately, interactive speed has always compromised visual quality, general applicability, or flexibility. In [190] and [109], key-framed animations are generated offline, which frequently leads to the time-intense refinement of the key-framed animation. You et al. used a 16 processor

SGI Challenge for parallel volume-rendering of isosurfaces [241, 221]. In contrast, Gobetti et al. used the 3D texture mapping hardware abilities of high-end graphics systems for volume rendering [85]. However, the lack of shading reduced the visual quality significantly[1]. Furthermore, the size of the texture memory limits the size of datasets severely, while swapping techniques like bricking reduce the frame-rate. The Navigator software of General Electric uses isosurface ray casting with approximately one frame per second. Even if the performance of the 1996 results has significantly improved, it hardly can be viewed as interactive [52]. A mixed technique is used in [102], where a variation of template-based ray casting [240] provides visibility information later used for MarchingCubes-based polygonal rendering. Another variation was recently presented in [225], where six offline generated volume rendered movies were combined into a cube map to provide interactive rendering speed on low-end PCs similar to Chen's QuicktimeVR [38].

Besides rendering, the used navigation paradigm determines the usability of a virtual endoscopy system. Many systems [218, 109, 143, 174, 32] use a planned or automatic navigation, which generates an offline animation of a fly-through after specifying a camera path. This simple scheme reduces the interaction to a VCR-like functionality, requiring a costly refinement of the camera path (and of the animation), if the structure of interest is not well covered. A variation of the planned navigation is the "reliable navigation" [96], in which a complete "visit" of all structures of the organ is guaranteed. However, this also means that user interaction is limited and that irrelevant regions cannot be easily skipped.

A free navigation approach is followed by [52, 63, 220, 6]. Unfortunately, the complexity of the anatomical structures commonly found in the datasets is very high. Even for a specifically trained physician, it can be difficult to navigate to the target. For similar reasons, semi-automated flythroughs [110] cannot be easily integrated into free navigation frameworks. Furthermore, collision avoidance is a costly operation which is frequently not available in these systems.

In [110, 25, 31], a guided navigation paradigm [78] was adopted in order to provide full navigation flexibility, combined with user guidance and an efficient collision avoidance scheme.

5.3 VIVENDI System

For our virtual endoscopy applications interactivity, image quality, and intuitive navigation are essential. The first two goals are currently only possible with the efficient use of polygonal graphics accelerators. This situation might change, once hardware-accelerated ray casting boards are commercially available [157], enabling interactive perspective and flexible volume

[1] In 1998, Westermann and Ertl presented 3D texture mapping-based volume-rendering with isosurface shading [228]. However, this approach does not provide sufficient performance for interactive endoscopy applications.

rendering[2]. Guided-navigation provides the most intuitive navigation technique, which combines guidance and flexibility. Consequently, the VIVENDI systems uses a visibility-based polygonal rendering technique, together with a physically-based guided navigation system. In some parts, VIVENDI follows the VICON system for interactive colonoscopy [11, 110]. However, due to the different anatomical topology, it uses a different scene decomposition and visibility scheme. Therefore, the only common method is the guided-navigation system, which is already discussed in detail in work previous to this dissertation [110].

5.3.1 System Architecture

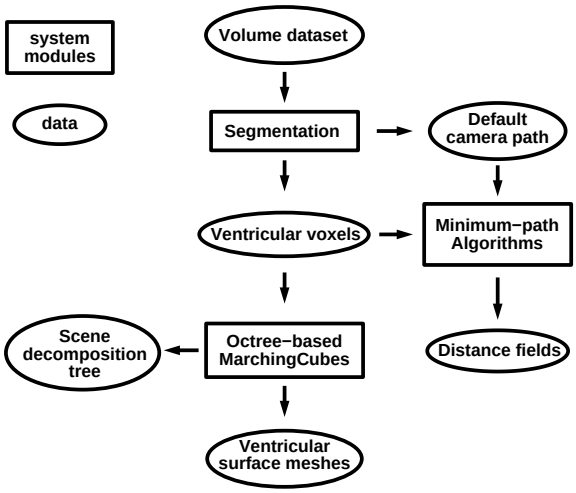


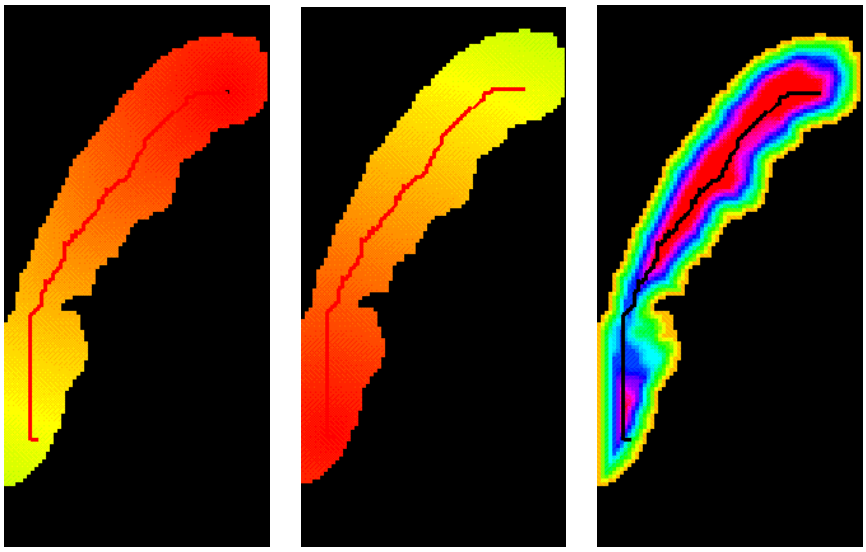
Figure 5.1: VIVENDI pre-process flow [25]

The endoscopy system itself consists of two stages: pre-process and interactive virtual endoscopy. The pre-processing stage is responsible for the generation of numerous auxiliary data, which is later used during the interactive virtual endoscopy. It is organized in three major steps, which are outlined in Figure 5.1.

In the first step, the voxels classified as part of the organ of interest are segmented. After the interactive specification of a seed voxel in the organ, a 3D region growing algorithm selects all organ voxels which are connected via other voxels satisfying the threshold based segmentation criterion. Using the seed point and an additionally specified target voxel as start and end point of the default camera path, this path is generated using Dijkstra's single source shortest path algorithm [54] (see Fig. 5.2c). As a cost function, a 3D Euclidean distance transformation is employed on the segmented organ voxels [178].

[2]The VolumePro system [166] is an already available hardware-accelerated volume rendering board. However, the lack of perspective projection and the base-plane rendering approach prevent its use for virtual endoscopy.An early case study on the use of the VIZARD II board [157] can be found in [149].

Thereafter, we extract the isosurfaces of the respective organ system, using an octree-based parallel implementation of the Marching Cubes algorithm [29] (see Chapter 3). The size of the leaf blocks of the octree depends on a specified granularity value of volume cells which intersect with the isosurface[3] to produce a roughly leaf load balanced tree. Based on this octree representation, a decomposition of the extracted isosurface is generated, where the isosurface geometry associated with an octree leaf block is considered as a scene decomposition entity.



(a) (b) (c)

Figure 5.2: Distance fields of a segment of a blood vessel; the distances are coded in colors: (a) distance to target point (red), (b) distance to start point (red), (c) distance to surface (small distance: yellow, large distance: red). The default camera path can be seen as red (a,b) and black lines (c).

Finally, three distance fields are computed, implementing a collision avoidance scheme and the simulated current in which the virtual camera for guided-navigation [110] flows towards a target point. The first two distance fields are representing the minimum path data from every organ voxel to the target voxel – implementing a current towards the target point – and from every organ voxel to the start (seed) voxel – implementing a current in the opposite direction. For the computation of both distance fields, we again use Dijkstra's single source shortest path algorithm [54], using the standard voxel distance as cost function (see Fig. 5.2a and b).

The third distance field represents the distance information from every organ voxel to the closest surface voxel of that organ[4]. This distance field is computed by the same volumetric Euclidean distance transformation as used for the default camera path computation [178] (see Fig. 5.2c).

[3] We call these cells relevant cells and the respective granularity value *relevant cell load* or RCL.

[4] A surface voxel of the organ is an organ voxel with neighboring voxels which do not belong to the organ segmentation.

For interactive colonoscopy using the VICON system, complete pre-processing time took up to ten hours. Most of the time was spent on generating the default camera path (or skeleton of the colon), the distance fields, and on the generation of the decomposition along this skeleton. However, this time can be greatly reduced by using improved data-structures and algorithms. Replacing FIFO-queue based priority queues of the pre-processing steps by Fibonacci heaps and hash tables [148], we could reduce the algorithmic time complexity[5], which in turn reduced the time consumption from several hours to a few minutes.

For the previous skeleton-based decomposition step, a back-tracking algorithm was used, in order to optimize the size of the respective decomposition cells. However, this was a computational expensive approach, which frequently lead to a processing time of a few hours. Furthermore, most organ systems do not have a tube-like shape, and hence no tube-like scene decomposition is available. Instead, we used a generalized octree-based decomposition scheme where computational costs are only a fraction of the skeleton-based method. In total, we reduced the pre-processing time down to approximately 15 minutes, depending on the size of the volume dataset.

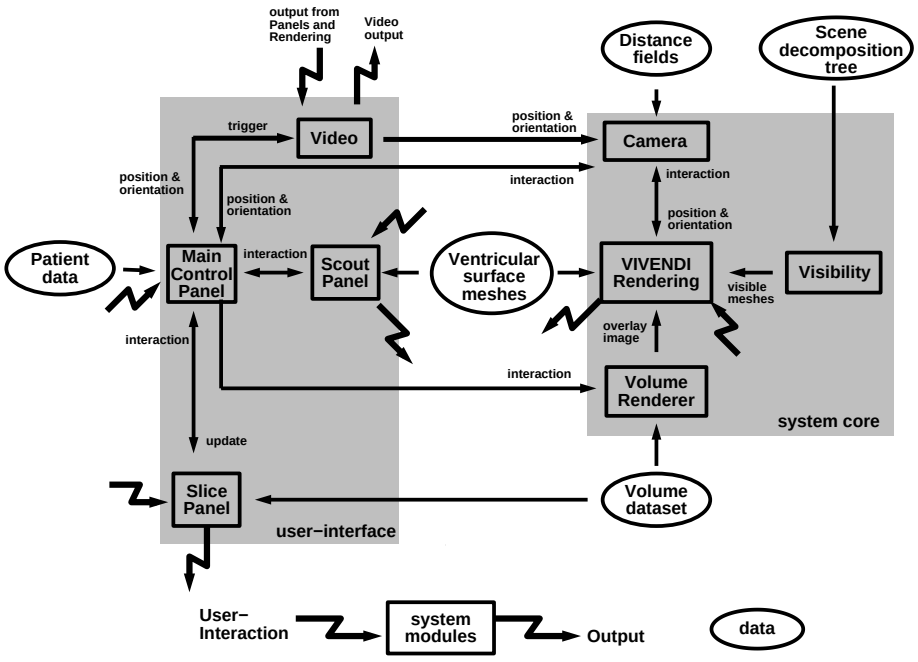


Figure 5.3: VIVENDI control flow [25]

The interactive endoscopy stage of the VIVENDI system is built from an user-interface and system core (Figure 5.3). The centerpiece of the latter is the VIVENDI rendering system (module **VIVENDI Rendering**), which is

[5]The run-time complexity of our camera path generation using Dijkstra's single source shortest path algorithms is $O(nlogn)$ with a priority queue based on a Fibonacci heap, and $O(n^2)$ with a FIFO-queue based priority queue, where n is the number of voxels selected as inside. Note that we can give an upper bound for the number of edges in our voxel grid-based graph with $3 * n$.

responsible for the OpenGL rendering of the geometry of the organ. In order to reduce the geometric complexity, a visibility culling algorithm is applied (module **Visibility**, see below for details), based on the scene decomposition generated in preprocessing step two. User-interaction (for navigation and measurement) via the rendering area of VIVENDI Rendering is bypassed to the **Main Control Panel**. Position and orientation of the virtual camera are provided by the guided-navigation system (module **Camera**, see [110] or see below for details). If the user initiates direct volume rendering (module **Volume Renderer**), the generated images are overlaid on the polygonal rendering of VIVENDI Rendering.

On the left hand side of Figure 5.3, the user-interface (see Fig. 5.4) is organized around the Main Control Panel, which provides control over camera navigation, volume rendering, video generation, and other general parameters of the system. The Main Control Panel communicates with the **Slice Panel**, which provides the three orthogonal slices at the current position of the virtual camera in the volume dataset at different resolutions. These projections represent the "traditional" imagery in radiology. Furthermore, the slices can be used for slicing through the volume data and to specify a new position and orientation of the virtual camera by clicking into organ voxel areas using the mouse. The **Scout Panel** provides a fully rotatable 3D overview of the surface of the organ system. Furthermore, it administrates the position bookmarks and controls the multiple camera paths (see below). If a bookmark is used as a jump mark, it provides the virtual camera with the new position and orientation via the Main Control Panel. Similarly, the **Video** module provides a pre-specified sequence of position and orientation data to the virtual camera, once the video generation is triggered by the Main Control Panel, to allow an automated animation of a previous walkthrough of the organ. The current rendered contents of the endoscopic view (VIVENDI Rendering), the 3D overviews of the organ system (Scout Panel), and the three orthogonal volume slices of the Slice Panel are combined into a video frame and stored to disc.

5.3.2 Visibility Culling

In Chapter 4, we presented several methods for efficient culling of geometry. While some of the approaches provide conservative culling ("Everything which might be visible will be rendered"), others provide only non-conservative culling – where possibly visible geometry is culled, if the used heuristics does not find a trace of a visual contribution of that geometry. However, medical applications have specific requirements which prohibit non-conservative culling methods. Therefore, the VIVENDI system uses a conservative visibility driven rendering approach, using hierarchical view-frustum culling, depth-oriented traversal, and occlusion culling exploiting the Hewlett-Packard occlusion culling flag ("HP flag") [25] as described in Chapter 4.

Table 5.1 shows the tradeoff between visibility test overhead and render-

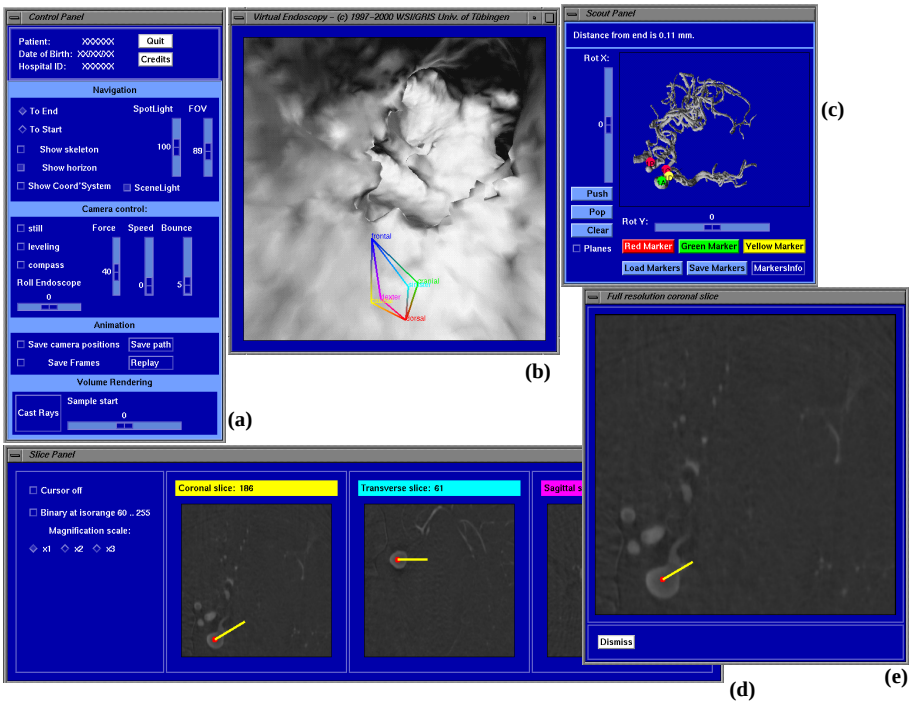


Figure 5.4: VIVENDI user-interface (snapshot from virtual angioscopy): (a) Control Panel, (b) Main Endoscopic View, (c) Scout Panel, (d) Slice Panel, and (e) Full Resolution Coronal Slice Panel [25].

ing performance for a dataset of the virtual ventricle endoscopy application; virtual colonoscopy and virtual angioscopy show similar results. The measurements are performed on an HP J-class/VISUALIZE fx6 workstation using a PA-8500 LC/440 MHz CPU, and on an HP P-class/VISUALIZE fx6 Linux PC using a PIII/750 MHz CPU. Both systems are using the HP flag and provide enough memory ($\geq$ 256 MB) for an efficient use of VIVENDI.

The first decomposition into 121 blocks achieves the best culling rate, since it allows the finest approximation of the octree leaf blocks to the geometry. However, the frame-rate of the second decomposition (67 blocks) is higher (0.6 fps), although the number of rendered polygons is 20% (2% of the total polygon count) larger. The finer scene decomposition induced more visibility tests which are quite costly. In particular the occlusion culling test requires a flush of the graphics pipeline to obtain the up-to-date occlusion information of the current query. These higher costs are not compensated by the only slightly reduced polygonal rendering load, thus the benefits of culling polygons of the model is exceeded by the occlusion culling overhead. In some areas with low occlusion, the overhead might even reduce the frame-rate (compared to view-frustum only, or straight forward rendering). Overall, a culling performance of up to 92% of the model and average frame-rates of up to 38.7 fps on the HP/J-class were achieved.

RCL	total #blocks	% blocks visible	#polygons visible	culling rate (%)	frame-rate (fps) J7000/fx6	PIII/fx6
View-frustum culling only						
1000	121	59.2	187799	40.5	12.24	10.12
2000	67	60.1	212153	32.6	11.87	9.4
4000	31	79.8	241666	23.1	11.44	8.79
View-frustum and occlusion culling						
1000	121	48.6	25417	91.9	38.03	22.58
2000	67	43.1	32048	89.8	38.7	18.67
4000	31	38.3	64709	79.4	29.15	13.46

Table 5.1: Average performance of view-frustum and occlusion culling of a ventricular system with more than 315K polygons. Other ventricular datasets and virtual colonoscopy and virtual angioscopy applications show similar results. RCL describes the relevant cell load (see Section 4.6.2) which determines the size of the hierarchy leaves.

5.3.3 Multiple Camera Settings

Segmentation and navigation depend heavily on the start point of the camera path (the seed point of the segmentation) and the camera path itself respectively. However, not all areas of interest are connected by voxels classified as organ voxels. This is due to partial volume effects, lack of resolution, or obstruction of narrow areas. Furthermore, some examinations require different camera settings in order to reach different locations within the ventricular system. Other applications (see Section B.3) even use data of different modalities, which require individual pre-processing.

For these cases, VIVENDI supports multiple camera settings, which combines the models of different areas of interest of one or more volume datasets into a joined representation. Each single model is pre-processed individually – providing its own camera path (and reconstructed isosurface if necessary) – and finally combined into the joined model with the respective number of camera settings. Alternatively, this functionality can be used to generate multiple camera paths in a single model[6]. For the actual virtual endoscopic examination, all individual models of the joined representation which are not the current one, are considered not visible (with an exception for the combined transparent version, see Section B.3), since they are outside of the current opaque model. To change to one of the other camera paths, automatic pre-defined blue model markers at the ends of the camera paths on the scout panel can be selected (Figure 5.5). In the course of changing to another camera path, the distance fields are changed too. This is necessary to reduce the look-up time of the binary tree on the compressed representation of the distance fields.

[6] In this case, the single model is not simply copied, but referenced for each additional camera path.

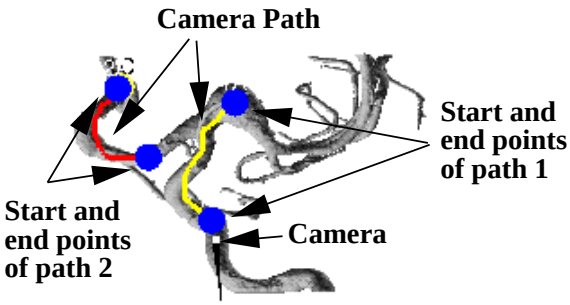


Figure 5.5: Snapshot from the Scout Panel: Two different camera paths are combined into one model. Blue (dark) markers are pre-defined model bookmarks; red, green, and yellow (different shades of grey) markers are user-defined markers [25].

5.3.4 Guided Navigation

Guided navigation is already described in detail in work previous to this dissertation. Therefore, we limit this section to a brief review of this technique. More detail can be found in [110].

Guided navigation was also introduced by Galyean [78] in 1995, where he employed pre-computed paths and a spring-based model for user-interaction. Unfortunately, the paper lacks implementation detail, specifically on the parameterization of the camera model.

For virtual colonoscopy [110], the camera mimicked the miniaturized submarine in the Academy-award-winning movie *Fantastic Voyage* (20th Century Fox, 1966), which traveled through the blood vessels to the brain. Like this submarine, the virtual endoscope travels along a default path to the target area. Two distance fields are interpreted as potential fields and are implementing a current towards a forward and backward target point (see Fig. 5.2ab). The third distance field is interpreted as a repulsion force which prevents the virtual endoscope from penetrating the surface (see Fig. 5.2c). These distance fields are evaluated at each camera position using a force function. Additional kinematic rules implement several movement models of the camera. Most important is an user-applied force, implementing an impetus of the camera (similar to the submarine of the movie) which can move the camera against the current. Details on these functions can be found in [110].

5.4 Discussion of Virtual Endoscopy

In this chapter and in Appendix B, we presented several applications of virtual endoscopy. The different objectives of the applications impose specific requirements on the virtual endoscopy system. An educational objective focuses more on the visual quality which demonstrate the general topological and geometric aspects of the specific patient anatomy. In contrast, the ac-

curacy of fine details is only of limited importance, if the details are not the subject of the examination. However, this is different for a clinical objective where the accurate rendering is one of the major factors which determine the usability, where an incorrectly represented blood vessel connection might have a fatal impact on the medical intervention. If virtual endoscopy is used for planning an intervention, it is important that relevant anatomical structures are represented appropriately, since otherwise the planned access path (i.e., in virtual ventriculoscopy) might be occluded in the "real world" anatomy. Similar, the visual representation must be highly accurate for intra-operative navigation to provide usable information to the surgeon.

There are several sources of errors which can lead to an inaccurate visual representation of anatomical structures in virtual endoscopy. Most notorious are partial volume effects and undersampling which generate "fake" connections between the various caverns that are actually not connected. Furthermore, motion artifacts can reduce the visual quality severely or distort the actual anatomical geometry. These artifacts are generated by movement of the patient during a (long) medical scanning procedure. Another example is scanning of fast moving body parts, i.e., the valves of the heart which cannot be traced by modern volumetric scanners (see Section B.4).

(a) (b) (c)

Figure 5.6: Hole of different sizes in ventricular septum (of the brain); isovalue in (a) is lower than in (b). (c) shows a slice from the original volumetric data of that area.

Even if these effects do not reduce the accuracy of the volumetric representation, the quality of the visualization depends heavily on the quality of the segmentation process. Isovalues which are not selected with sufficient particularity lead to holes in surfaces, if the isovalue is to low, or existing holes in the surface are closed and vice versa. In Figure 5.6, the hole in the ventricular septum between the two lateral ventricles varies in size, depending on the isovalue. However, the question of which isovalue is correct is not easy to answer in this case, because the volume data in that area is inconclusive (Fig. 5.6c); due to partial volume effects, it is not clear exactly what size the hole is.

Besides visual quality, interactivity is a major issue for virtual endoscopy. A rendering speed significantly below interactive rendering (10 fps) is usually not well accepted in the medical community. If virtual endoscopy is

used for inter-operative navigation, no measurable rendering lag is acceptable. The virtual endoscopy system must deliver real-time rendering performance to represent the geometry of the current view of the endoscope immediately with every movement of the endoscope. These requirements are usually not met with most virtual endoscopy systems. Even the VIVENDI system does not provide sufficient performance for all applications; virtual angioscopy of the heart provides only a few frames per second. The widely visible inner geometry of the left and right ventricles of the heart allow only a culling rate of 80% which leaves a high polygonal complexity for rendering. For all other applications however, VIVENDI meets the requirements for interactive exploration and in particular for inter-operative real-time navigation of ventricular MRI datasets.

Virtual Endoscopy versus Optical Endoscopy

The more general question of whether virtual endoscopy provides more scientific or medical insights, more patient safety or comfort, or an economic benefit is more difficult to answer. As for most scientific problems, the answer depends on the actual goal of the procedure, the qualities of alternative medical procedures, and various costs of the procedures.

In all procedures which require a histological examination of a tissue sample under a microscope, virtual endoscopy is not able to compete. The data resolution of modern 3D scanners does not reach into the resolution of a microscope, although it is already in a sub-millimeter range for rotational angiography. Furthermore, texture information, such as structure, color, and reflections is also not captured by 3D scanners. All applications which heavily depend on this information will not succeed with virtual endoscopy. Similarly, if the medical procedure includes the removal of tissue (i.e., lesions or tumors), or other objects, invasive or minimally-invasive procedures cannot be replaced by virtual endoscopy, since it does not interact with the actual body of a patient.

However, if the relevant information can be represented as geometric shape – i.e., a polyp of virtual colonoscopy –, virtual endoscopy can be used for diagnostic purposes. Furthermore, it provides insights into body parts which might not be accessible to current medical procedures. The virtual representation based on scanner data provides access to virtually all scan-able body parts. Physical limitations of optical endoscopes – i.e., the limited flexibility and navigation of the endoscope used for ventriculoscopy, the insuperable obstruction of folds in a colon for optical colonoscopy – are not shared with virtual endoscopes. Similarly, virtual endoscopes do not share the frequent unpleasantness of optical endoscopes. The patient interaction is limited to the scanning procedure, and is therefore providing much more patient comfort and acceptance. In addition, data acquisition and the actual virtual procedure are not necessarily at the same location. This geographical decoupling allows tele-medical procedures which are not possible with optical endoscopy, where data acquisition and procedure are

inseparably combined.

From an economic point of view, virtual endoscopy does produce less costs than the optical counterpart, since usually no sedation, patient preparation, or even hospitalization are required. The necessary computational expenses can be seen as additional post-processing of the volume reconstruction of the scanner. However, procedures which combine virtual and optical methods, i.e., ventriculoscopy, do not benefit from these costs; the goal of this combination was to reduce the risk of complications and to increase the success of the intervention.

Chapter 6

Other Applications

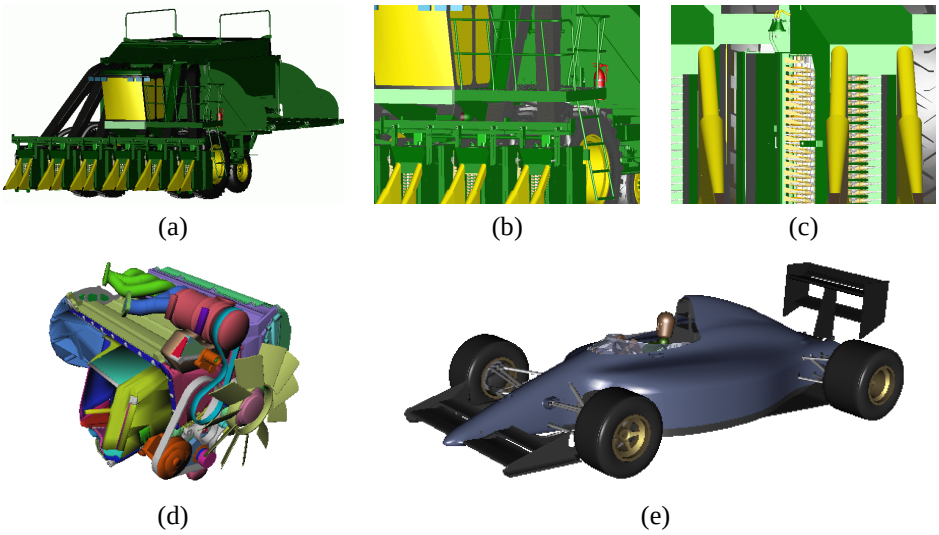
In the previous chapter, we introduced an application from virtual medicine, namely virtual endoscopy, which heavily use large model visualization techniques. However, these techniques are as well useful for numerous other application fields. Here, we briefly touch a few of these other application fields.

6.1 Mechanical Engineering

Mechanical engineering is one of the premier source of large polygonal models. Products such as cars, electronic devices, or production facilities, are designed with mechanical CAD (MCAD) software and managed by product data management systems (PDMs). Usually these models are designed with freeform surfaces, which are tessellated into polygons for rendering. The resulting multi-million polygon models introduce a tremendous polygon load to rendering systems which are assigned to design review tasks. Systems like CoCreate's OneSpace or EAI's VisMockUp employ large model visualization techniques from computer graphics to accelerate the rendering of the large models. In particular techniques like multiresolution representations and occlusion culling are used in these systems, based on HP's Jupiter rendering toolkit [101].

Examples of typical MCAD datasets can be seen in Figure 6.1. The cotton picker (a-c) is composed of more than 13.000 individual parts, organized in a hierarchical assembly list. The total model contains 10.605.158 triangles, where most of the geometric complexity is located in the six spindle compartments (1.633,137 triangles each), containing the spindle drums (694,113 triangles each) which collect the cotton flakes. These drums in particular are usually occluded by covers and chassis parts. The second dataset is an MCAD model of an engine of a BMW car (d). The model is composed of 245 individual parts, which total in almost 140.000 triangles. Finally, the third dataset is an MCAD model of a formula one racing car (e). Its assembly list contains 306 individual parts with a total of more than 740.000 triangles.

In Chapter 4, we examined visibility and occlusion culling methods which



(a) (b) (c)

(d) (e)

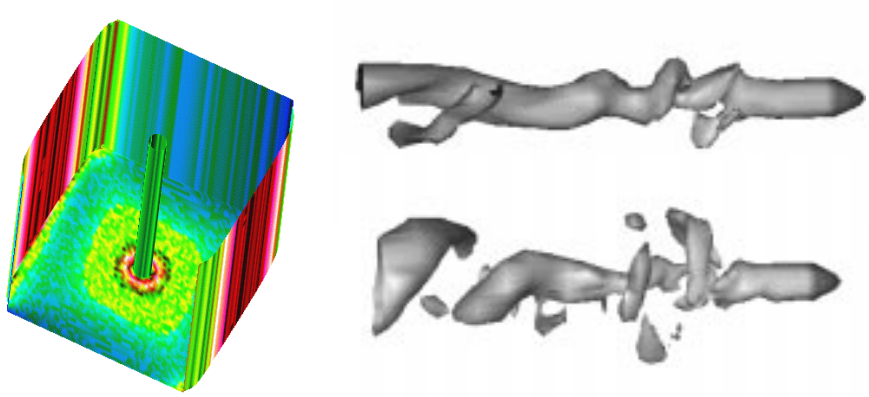
Figure 6.1: MCAD models: (a-c) Cotton picker; (b) close-up of (a), (c) further close-up of (a) showing the spindles drums; (d) engine of a BMW car; (e) racing car.

are useful in the context of rendering of MCAD models. Effective model tree organization and efficient model tree traversal are improving the frame-rate measured performance. Of particular interest for MCAD datasets is the careful design of bounding volumes used in a hierarchical, visibility driven rendering approach. In Section 4.7, we evaluated discrete orientation polytopes (k-dops) as bounding volumes of entities in a hierarchical model organization. Especially for MCAD models k-dops proved to be very efficient and reduced the quantity of rendered polygons tremendously.

6.2 Scientific Visualization

The probably major source of large scale data is scientific computing. Driven by various research programs, tera-byte datasets [123] are computed or measured, ranging from medical data, like the “visible female” dataset, to environmental simulation. Beside the management of this huge quantity of data, the visualization of the features is one of the major research fields of scientific computing. Figure 6.2 shows two examples of the visualization of datasets from computational fluid dynamics (CFD). (a) shows a fluid filled cavity, which is heated from two sides. The isosurfaces represents a surface of the same velocity magnitude; the temperature is mapped as color onto the isosurface. (b) shows a vortex breakdown of a fluid which is injected into another fluid at different time steps. The isosurfaces represent again a surface of the same velocity magnitude.

Various techniques are applied to find the “hot spots” and other interesting features in the datasets. Among the most popular are mesh-reduction, parallel processing, feature extraction [123], occlusion culling and volume rendering [165] for the fast visual representation. In Part I, we explored



(a) (b)

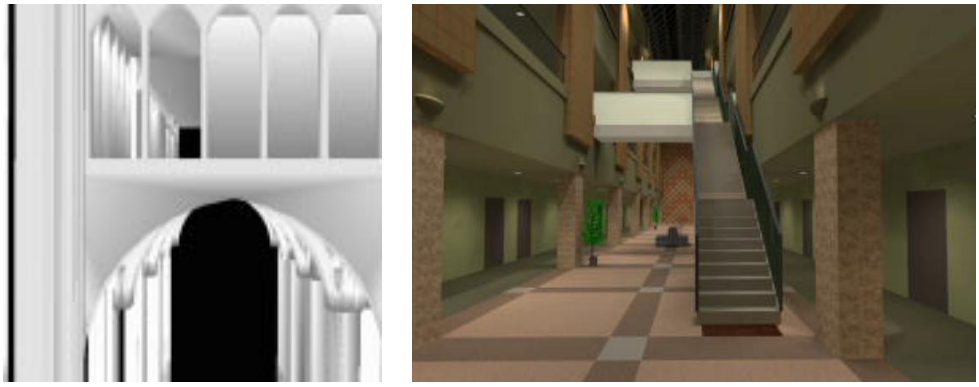
Figure 6.2: Models from scientific visualization: (a) Heated cavity with fluid; (b) vortex breakdown at two different time steps.

some of these techniques. We compared the advantages and disadvantages of direct and indirect volume rendering, and the different qualities of the four most popular direct volume rendering algorithms to decide which approach is most suited for a specific visualization problem (Chapter 2). We also introduced a technique how to parallelize the hierarchy generation of large datasets, which becomes worthwhile, if the hierarchical organization of such datasets is frequently revised (Chapter 3). And finally, the visibility and occlusion culling techniques described in Chapter 4 can be used to reduce the rendering complexity.

6.3 Architectural Walkthroughs

Computer graphics methods became very popular for the visualization of architectural models. In particular the correct illuminated rendering of a planned building is important to demonstrate the impact of the building on the environment, and the light conditions inside of the building. However, global illumination methods are of high computational expenses which make their practical use difficult. The radiosity and ray-tracing based rendering of such complex buildings can take hours, or even days, depending on the individual geometric complexity. Recently, several methods were developed which limit the energy exchange calculation to the visible patches [58, 181], thus reducing the necessary rendering time by an order of magnitude. Other visibility methods were used to accelerate the polygonal rendering of the interior of buildings to interactive frame-rates [3, 206, 22, 127]. Figure 6.3 shows examples of architectural models. (a) shows a snapshot from the interior of a gothic cathedral and (b) shows a snapshot from the interior of the atrium of the University of Aizu (Image courtesy of Karol Myszkowski, Max-Planck-Institut für Informatik, Saarbrücken).

Other architectural applications include urban planning management, where



(a) (b)

Figure 6.3: Architectural models: (a) Gothic cathedral; (b) Atrium of University of Aizu.

walkthroughs of entire current or future cities are required.

Chapter 7

Conclusions

In this dissertation, we presented several new techniques to handle the efficient visualization of large models. We put a special focus on volume rendering, parallel processing, and on the efficient culling of not visible polygonal geometry.

Assessment of Volume Rendering Techniques: We provided a framework for quality- and resource consumption-based comparison of various direct and indirect volume rendering approaches which creates a basis for deciding which approach is the most suitable for a particular application. Specifically, we compared one indirect volume rendering approach (Marching Cubes), and one direct volume rendering approach (ray casting). In the second part, we compared ray casting, splatting, shear-warping, and 3D texture mapping, since they are the four most popular volume rendering approaches.

Parallel Construction of Recursive Tree Hierarchies: With techniques like the asynchronous push-up, tree level mutexes (as a simplified version of predicate or tree locking), and the thread local memory pre-allocation, we introduced a new and efficient approach for the parallel construction of recursive tree structures, such as octrees, quadtrees, BSP-trees, or kd-trees.

Visibility and Occlusion Culling: The efficient culling of not visible geometry involves many aspects. First of all, an adequate, often hierarchical representation of the data needs to be computed. This also requires the acquisition of the respective information during the modeling stage of the data, or the retrieval of this information from a given model. During the visibility driven rendering, the representation must be traversed using appropriate heuristics which significantly influence the culling and rendering performance. Finally, the occlusion status of an object must be determined to decide if and how this object needs to be rendered. In this dissertation, we addressed all these aspects of the full culling pipeline of modeling, representation, and soft- and hardware-based visibility driven rendering.

In the second part of the dissertation, we presented several applications which heavily rely on the large scale data visualization techniques introduced in the first part. The major focus here lies on virtual endoscopy,

which visualizes anatomical structures of patient data mimicking an endoscope. We introduced VIVENDI, a virtual endoscopy system for the interactive exploration of patient organ systems. The usefulness of VIVENDI was demonstrated on several endoscopic applications (see Appendix B). In particular we looked into the endoscopic exploration of the colon (colonoscopy), of the ventricular system in the human brain (ventriculoscopy), and of the blood circulation and supply system (angioscopy).

Besides virtual endoscopy, there are many other applications for large scale data visualization techniques. Specifically, computer aided mechanical engineering, scientific visualization, and architectural walkthroughs pose challenging requirements to modern computer graphics systems, which are significantly reduced with the presented techniques.

7.1 Future Directions

Throughout the approaches discussed in this dissertation, we pointed out several possibilities for future work. Some of these topics are already explored, but are not part of this dissertation. In this section, we briefly outline some of the future research directions in the field of large model visualization, virtual medicine, and other applications of large model visualization techniques.

The comparison of volume rendering approaches still leaves enough room for the examination of advanced algorithmic techniques to improve the quality and the resource consumption. In particular the influence of oversampling and resampling of the data for the various approaches needs to be examined. Also necessary is the close examination of advanced acceleration techniques such as space-leaping, multi-resolution representation, and advanced filtering techniques.

The methods for the parallel construction of recursive tree structures also provides strategies for the on-the-fly reconstruction of data-structures. This reconstruction becomes necessary, if the location of significant parts of a model is frequently changing, i.e. in highly dynamic applications. Other future topics include the adaption of the techniques from thread-based systems to message-passing systems like large clusters of computers.

While the culling pipeline is already very efficient on a single CPU, single graphics pipeline systems, it leaves many research opportunities for the efficient deployment of multi-threaded, or multi-piped systems. However, this combination is non-trivial due to difficult synchronization and distribution issues. Other interesting research topics include the efficient combination of mesh-reduction and occlusion culling beyond the ideas presented in Chapter 4.

The field of virtual medicine and endoscopy also provides many research opportunities. A typical short term goal is the incorporation of stereographic display techniques for virtual endoscopy, since depth perception of optical endoscopy is very limited due to the “fish-eye” phenomenon.

In the long run, the combination of anatomical and physiological (functional) information for *physiological modeling* will be a major research objective. Starting from measuring the volume of anatomical structures and the acquisition of MRI-based flow data, functional simulations of organ structures of a patient will become possible. This data can be used to validate and improve physically-based models for the complex simulation of medical interventions, including a haptic aspect which provides an additional cue for training, but also for the understanding of medical data.

Part III

Appendix

Appendix A

Volume Data Acquisition Techniques

There are many sources of data for medical visualization. For our purposes, we rely on data from medical scanners, such as Computed Tomography (CT or CAT scanners), Magnetic Resonance Tomography (MRT or MR scanner), rotational biplane X-ray (or rotational angiography), 3D Ultrasound, Positive Emission Tomography (PET), and many more. These scanners produce a stack of images, where each image represents a slice from a data volume. Each entity or pixel of the images represents a sample point within the data volume and it is also called a voxel (see Section 1.2.1).

In the course of this appendix, we give an introduction into basic concepts of the scanning technology of the modalities of the volume datasets later used in chapter 5. More details on the specific technology can be found in [39, 137, 118]. Several sources of artifacts are known; most notorious are aliasing problems due to undersampling, motion artifacts if the scanned object is moving during the scanning procedure (i.e., a heart), and *partial volume effects*, which are also related to undersampling of a structure. 3D data samples (voxels) which are reconstructed from projections of a 3D scanner are averages of the local volume. If material with a high voxel value is adjacent to material with a low voxel value, this averaging can lucidly distort these voxel values, causing the partial volume effect.

A.1 X-Ray

In classical X-ray imaging, electrons are shot at the focal spot of a specific target. The resulting radiation is absorbed (and scattered) by an object (i.e., body parts) behind the focal spot and hence the attenuated intensity is recorded on detectors behind the object. Depending on the quantity of the acceleration voltage, X-rays are denoted as hard or soft radiation. The interaction of X-rays with matter is basically determined by photo-electric absorption and by scattering (Compton scattering). With hard radiation, the scattering effect is dominating the interaction, resulting in an absorption only significant in object areas with high density (high atomic number), i.e.

bones. For areas with a lower density (i.e., tissue), a sufficient absorption requires soft radiation. Today's X-ray devices feature a very high resolution (i.e., 4096×4096 pixels), but only as a 2D projection.

A variation of classical X-ray imaging is fluoroscopy. The emitted radiation is recorded with an image intensifier and displayed such that the progress of specific procedures can be observed. This permanent exposure to ionizing radiation requires a significant lower intensity than with classical X-rays. Hence, the resolution and general image quality are reduced.

A.2 Computed Tomography

The introduction of X-ray Computed Tomography (CT) in 1972 [112] provided for the first time a volumetric representation of objects, and not only a 2D projection of a volumetric object. Generally, it is seen as one of the major milestones in medical imaging [39].

As basic concept, X-rays are emitted through the object from different positions around that object (i.e., filling an angular range of 180 degree) and the intensity profile is recorded by a detector. Algorithms are applied to reconstruct an intensity attenuation layer of the object. A series of these projection with an incremental modified position generates a stack of intensity attenuation layers, which forms an intensity volume. The intensity attenuation is measured in Hounsfield units (HU), with water as reference material (0 HU). I.e., bone has a high measured intensity attenuation, while fat or air have negative HU [137]. Each of the reconstructed volumetric samples (voxels) represents the average of the local environment.

The major differences between the development stages of CT [118] are different projection reconstruction algorithms and the emitter and detector architecture (data acquisition). The first generation of CT scanners was basically the experimental setting of Hounsfield's CT scanner. It used a single pencil-like X-ray beam emitter and a single detector on the opposite side of the object. To acquire a data slice, the pencil beam is translated along the object and rotated afterwards for the next series of beams. All together, the costly mechanical movement of emitter and detector caused long scanning times, ranging from several minutes to several hours at a resolution of 80×80 pixels per scan/slice. Furthermore, the single emitter/detector architecture enabled only a poor utilization of the emitted radiation.

The next generation and first commercial generation of CT scanner used small angle fan beams and multiple detectors to scan two neighboring rotational projections at the same time. This technique reduces the number of necessary rotations, and hence the required scanning time (10 - 60 seconds, up to several minutes) needed for a sufficient reconstruction. It also provides a better utilization of the emitted radiation. Both first and second generation techniques are parallel beam devices, which use different reconstruction algorithms than the next fan beam devices.

The next improvement increased the fan beam angle and the number of

detectors to cover the whole object, thus the translating movement became unnecessary and increased the scanning speed to five seconds per slice. Similar to the previous techniques, the radiation is enabled in fixed time intervals to be measured by the detectors.

In the fourth generation, the rotating detector was replaced by a fixed circular ring detector, which reduced the technical effort of moving the larger mass of emitter and detector. Here, the radiation was permanently emitted and only the detectors were enabled at certain intervals. However, several problems of ring detectors led to further developments in favor of third generation scanners. Besides the higher costs for the detector ring, specific X-ray scattering problems reduced the image quality of these systems, while collimator technology could reduce the scattering problems with a rotating emitter/detector system of the third generation.

Currently, the state-of-the-art are spiral or helical CT, where the emitter/detector system is rotating permanently around the object, while the object is moved continuously in the perpendicular direction to acquire a full data volume. This technique enables faster scanner due to the continuous rotating movement of the emitter/detector system which saves the time for the previously needed time to accelerate and slow down these heavy parts of the scanner. Recently, multiple layers of emitters and detectors (twin or quad slices) where combined to create multi-slice CT scanners, which enable fast and isotropic scanning of large object areas.

Besides the architectural development, different volume/slice reconstruction algorithms differentiate the various systems and generations. The basic approach is the back transformation of the slice projections into the volume slices by the Radon transform [39]. In the beginning of computed tomography, algebraic reconstruction techniques (ART) were used to solve this back projection problem. However, the high computational costs of iterative solving the large matrices[1] rendered this approach as not usable for standard applications [118]. The standard method today is filtered back-projection, where each projection is composed according to the measured direction. Parallel and fan beam methods are available to perform this back-projection. However, the current fan beam methods are more complex and less efficient than state-of-the-art parallel beam reconstruction algorithms. Hence, the projections of today's fan-beam scanners are re-sorted in parallel beams before the actual reconstruction. Another modification is required to address the continuously moving object tray of modern spiral CT scanners, where a z-interpolation corrects the measured projections according to the tray movement [118]. In the future, cone beam reconstruction algorithms will probably replace the current methods [118], which are already successfully used in rotational biplane X-ray (see below).

[1] The size of a reconstruction matrix is equal to the resolution of the slice.

A.3 Rotational Biplane X-Ray

Rotational biplane X-ray is a recent scanning technology which started from digital subtraction angiography (DSA)[2]. Hence, most of the associated applications are angiography applications and this technique is frequently referred to as rotational angiography [62, 94].

To generate volumetric datasets, a series of up to 132 X-ray projections are taken from a rotation range of 200 degrees around the scanning object. In contrast to CT, rotational biplane X-ray is using a full array of up to 1024^2 detectors which enable the measurement of a full cone of rays. To account for reconstruction errors [118], a modified back-projection algorithm is used [111]. Additionally, special filter kernels are used which further reduces potential artifacts. Current rotational angiography systems provides very high resolution, isotropic datasets, good reconstruction quality, and a high data acquisition speed of up to 13 seconds for a full scan.

A.4 Magnetic Resonance Imaging

The beginning of Magnetic Resonance Imaging (MRI) dates back to the early seventies, but it was not adopted into medical use until the nineteen-eighties. It is based on the nuclear resonance of hydrogen in a magnetic field, where each of the hydrogen nuclei can be considered as a small dipole magnet which aligns itself either parallel or anti-parallel along the magnetic field.

While aligned in that field, the protons (which are identical to the hydrogen nucleus) spin arbitrarily around the axis of the field. This spinning is called the *precession* of the nuclei. If energy is applied to the magnetic field as a radio-frequency pulse (RF) at the *Larmor*-frequency, the nuclear resonance forces the protons to receive some of the energy from the RF. This pulse also forces all the protons to spin synchronously, or *in phase* and flip increasingly into the anti-parallel orientation of higher energy, until the number of parallel protons is equal to the number of anti-parallel oriented protons. The duration of the RF pulse determines the amount of precession; i.e., a *90 degree pulse* will force the protons into a 90 degree precession, where the precession vector of the protons is completely perpendicular to the magnetic field, resulting in a zero z-component (along the magnetic field). After the stimulation of the protons, they slowly release the received energy, *de-phase* and re-align with the magnetic field. This *relaxation* is described as *free induction decay (FID)* and is divided into the transverse and longitudinal relaxation. The first relaxation of the transverse magnetization – also called spin-spin relaxation – describes the de-phasing of the x/y-component of the precession. The time required for this relaxation is called T_2 and is in the order of a few milliseconds. The longitudinal, or spin-lattice relaxation describes the re-alignment of the precession with the

[2] In DSA, X-ray images are recorded before (mask) and after (filling) the injection of a contrast agent. The mask is subtracted from the filling image, thus presenting only the contrast agent enhanced object areas.

magnetic field, thus the restoring of the z-component. This relaxation time is called T_1 and is in the order of seconds. The actual measured volumetric information is the proton density σ which needs to be reconstructed at the specific voxels.

To reconstruct the spatial information of the measured signal, two additional gradient magnetic fields are applied. The first field selects the slice in z-direction of the volume, since only one layer of protons suffices the Larmor-frequency for the main and gradient magnetic fields. An additional gradient field in x-direction selects a y-slab. With a 2D Fourier reconstruction, the x/y-coordinate is encoded into that signal [137] by increasing de-phasing signal of the transverse relaxation along the x and y directions, which generate specific frequencies into the signal. Thus, the RF intensity (proton/spin intensity) is encoded into the intensity of the signal, while the position is encoded into the frequencies [39, 137].

Different protocols describe sequences of various RF pulses and the actual measurement of the signal, where the time between the initial stimulation and the measurement is called *echo time* T_E and the time between two (initial) stimulation cycles is called *repetition time* T_R. By varying T_E and T_R, different weight data can be achieved; i.e., with short echo and repetition times, the proton density is mostly dominated by the T_1 relaxation time (T_1 weighted), with long T_E and T_R, the T_2 relaxation time is dominating the signal (T_2 weighted), or with a long T_R and a short T_E, the resulting data is neither T_1 nor T_2 and can be seen as "*the pure (proton) density function*" [39]. Scanning protocols such as Turbo Spin Echo (TSE) or Constructive Interference in Steady States (3D CISS) belong to these categories. A slightly different technique is used for angiography sequences, which focus on flow information. Specifically, the Time of Flight sequence (TOF) measures the spin saturation of the protons; the repeated stimulation of the protons in the data volume leads to a saturation of the signal of stationary samples. However, the particle flow in blood vessels are continuously introducing "fresh material" into the magnetic field which gives a strong signal. Typical MRI scans take between 2 and 25 minutes. This relatively long scanning time is dominated by the relaxation of the spins, not by the time required for the measurements.

Appendix B

Virtual Endoscopy Applications of VIVENDI

B.1 Virtual Colonoscopy

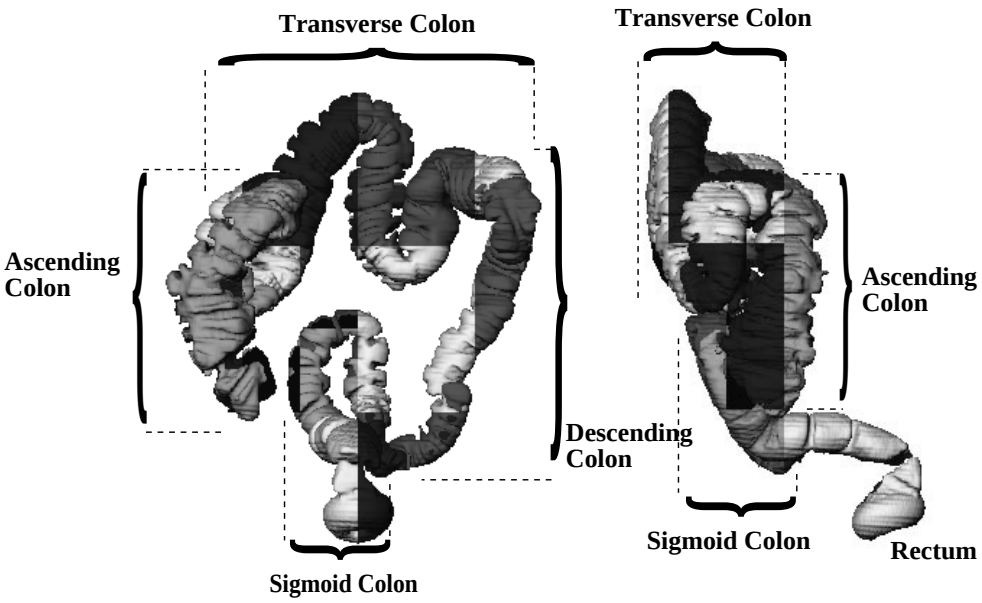


Figure B.1: Octree-based decomposition of colon dataset; the octree leaf blocks of the isosurface are represented with different colors. The left image shows a coronal view, the right image shows a sagittal view.

Originally, virtual colonoscopy used the VICON system [11, 110] with application specific algorithms for occlusion culling and volume rendering. Unfortunately, these algorithms depended on the tube-like topology of the colon which circumvented the utilization of VICON for other application areas. However, the VIVENDI system [25] does not have these limitations. As described earlier, it uses an octree-based decomposition of the isosur-

face extracted from the volume dataset. The octree structure can be seen in Figure B.1. The individual blocks of the colon of the patient dataset are rendered in different colors. In the remainder of this section, we repeat the results already reported in [110] for the sake of completeness. This time, however, we use the VIVENDI system which has a more flexible lighting model, which enables a spotlight at the camera position pointing in the view direction.

B.1.1 Motivation

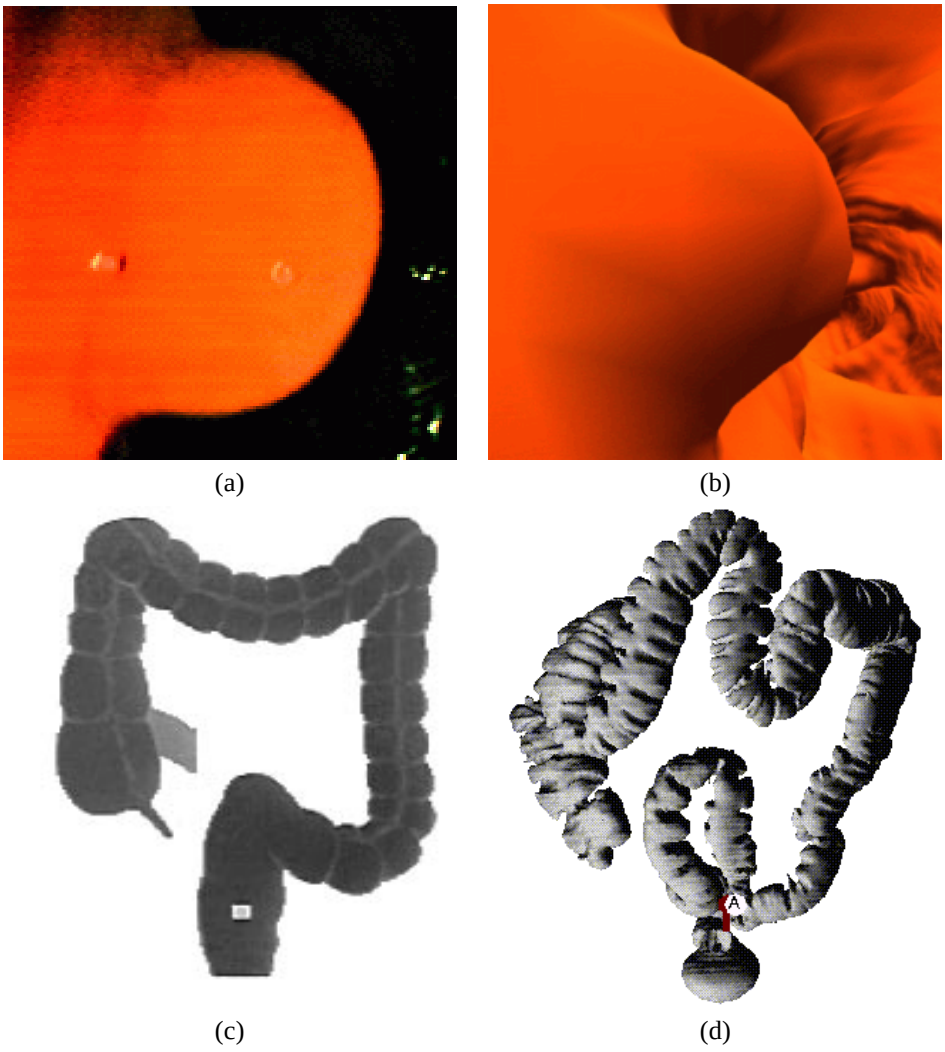


Figure B.2: An 8mm polyp in the descending colon, close to the Sigmoid colon; left (a,c): optical colonoscopy, right (b,d): virtual colonoscopy [110].

Cancer of the colon and rectum is the second leading cause of cancer deaths in the USA. Approximately 150,000 new cases of colorectal cancer are diagnosed every year [43]. Consequently, it is imperative that an effective diagnostic procedure is found to detect colonic polyps or tumors at an

early stage. Currently, optical colonoscopy and barium enema are the major procedures available for examining the entire colon to detect polyps larger than *5mm* in diameter, which are clinically considered to have a high probability of being malignant. In optical colonoscopy, a fiber optical probe is introduced into the colon through the rectum. By manipulating the tiny camera attached to the tip of the probe, the physician examines the inner surface of the colon to identify abnormalities. This invasive procedure takes about one hour and requires intravenous sedation, resulting in high costs. Barium enema in contrast requires a great deal of physical cooperation from the patient when the X-ray radiographs of the colon are taken at different views. Additionally, its sensitivity can be as low as 78% in detecting polyps in the range of *5mm* to *20mm* [160].

Both methods are either too expensive or to circumstantial for prophylactic screening examinations – resulting in a low patient acceptance –, hence virtual colonoscopy was proposed to limit optical colonoscopy to cases in which either a suspicious polyp was found – which induced a biopsy or removal of the polyp – or which were inconclusive in virtual colonoscopy [218]. The latter happens if (shape) defects of the graphical representation of the inner colon surface cannot be identified as polyps or residual stool.

After cleansing and inflating of the colon (both actions are also required for optical colonoscopy), a CT scan (or alternatively an MRI scan) is performed. The resulting image stack is pre-processed and examined using the VIVENDI system.

B.1.2 Optical and Virtual Endoscopy

We compare the results of optical and virtual endoscopy based on polyps found in both procedures. In particular we compare snapshots of two polyps (see Fig. B.2 and B.3). The first polyp (Fig. B.2) is located in the descending colon, close to the sigmoid colon. It is of a size of 8mm and hence of high clinically relevance. Figure B.2a and c show the information provided by optical colonoscopy, while b and d show the information provided by virtual colonoscopy. The shape information of the polyp is well represented by the virtual technique. However, textual information is not available, while it is very helpful in optical colonoscopy (although not obvious in Fig. B.2a or B.3a).

The overview image of virtual colonoscopy (Fig. B.2d) provides much better information than for optical colonoscopy, which is just a rough sketch of the general shape of a colon (Fig. B.2c). In particular the position information of the polyps can be misleading – optical colonoscopy estimates the position of the polyp in the sigmoid colon (see Fig. B.2c), while it is accurately reconstructed in virtual colonoscopy – locating the polyp in the descending colon.

The second polyp is of a size of 4mm and it is located in the transverse colon, not too far away from the hepatic (right) flexure. Similar to the previous polyp, the actual location is quite different from the rough estimation in

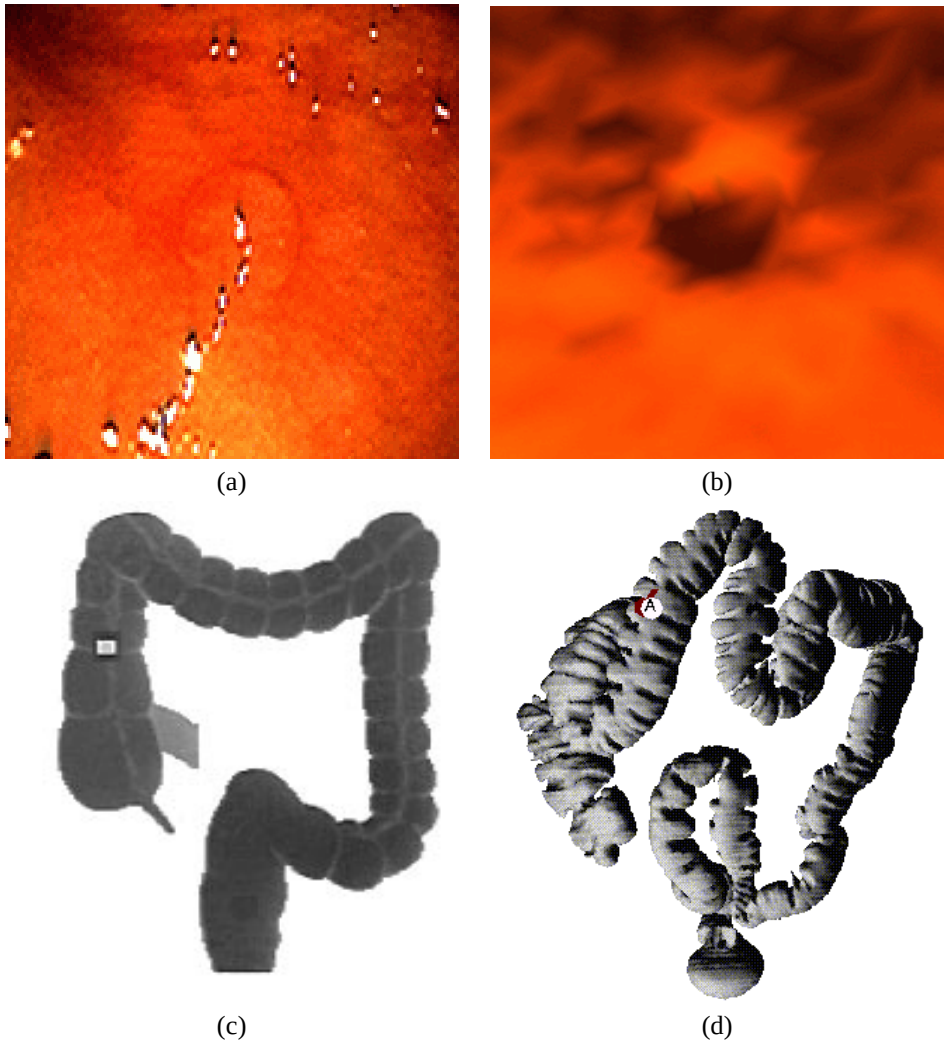


Figure B.3: An 4mm polyp in the transverse colon.; left (a,c): Optical colonoscopy, right (b,d): virtual colonoscopy [110].

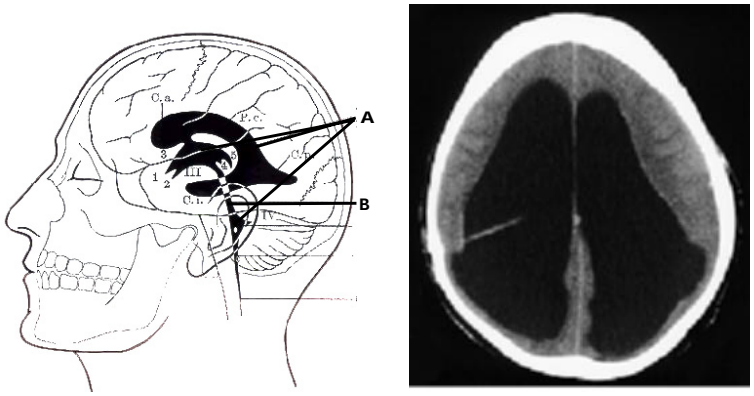
the overview image of optical colonoscopy, which locates the polyp in the ascending colon.

To summarize, virtual colonoscopy is an alternative procedure for the diagnosis of polyps in the human colon. However, it does not replace optical colonoscopy, which is still required once a found polyp has to be removed or a suspicious structure needs to be identified with additional information, such as texture, color, and histological information through a biopsy, which is generally not available by means of volume scanning methods,

Other applications of virtual colonoscopy include teaching, planning of optical colonoscopy procedure, and of intra-operative navigation.

B.2 Virtual Ventriculoscopy

The focus of (optical and virtual) ventriculoscopy is the ventricular system of the human brain, where the CSF (cerebrospinal fluid) is produced and resorbed (Figure B.4a). Specifically, the CSF is produced in the lateral (upper two) ventricles. Due to respiration and other metabolistic activity, the CSF flows through the *foramen of Monro* into the third ventricle (which is also producing CSF), and via the narrow connection of the ventricular (cerebral) aqueduct to the lower fourth ventricle. From this ventricle, the CSF is distributed to other cavities inside of the skull.



(a) (b)

Figure B.4: Ventricular system of the human head [172]: (a) A ventricles, B ventricular (cerebral) aqueduct, (b) Hydrocephalus in an image from a CT scan

B.2.1 Motivation

The drain of the third ventricles into the fourth ventricles is often blocked, due to occlusion or a stenosis of the aqueduct. This can be caused by a tumor, an accident, meningitis, or a congenitally defect. The result of such a blockage is a serious disturbance of the natural flow of the CSF, which frequently leads to a dangerous increase of pressure inside the skull and can damage the brain severely (Fig. B.4b).

The standard procedure for this hydrocephalus is the external drainage of the ventricular system into the abdominal cavity using a shunt. Unfortunately, this external drainage system is frequently the cause of complications – such as obstructions and degenerative processes – which result in the needed neurosurgical replacement of the shunt. Furthermore, the missing natural flow of CSF leads to degenerative processes of CSF producing structures and the resolving of the septum between the lateral ventricles. The treatment of the basic cause of the occlusion is usually not possible, because of the inaccessibility of the aqueduct for neurosurgical instruments. Recently, a new endoscope – small enough to pass through the foramen of Monro and with enough luminous intensity – was developed which allows

interventions inside of the ventricular system [56]. In consideration of the inaccessibility of the aqueduct – even with the new endoscope – the department of neurosurgery of the University Hospital at Tübingen is performing a *ventriculostomy*, where the natural drain via the aqueduct and the fourth ventricle is bypassed by a new drain in the floor of the third ventricle. To access the ventricles, a hole is drilled through the skull and a tube is placed through this hole, through the brain, into the posterior horn of the left or right lateral ventricle. Thereafter, the endoscope is introduced through the tube, which is used as a stable guide for the endoscope. It proceeds forward through the foramen of Monro to the floor of the third ventricle.

Because of the water-like optical property of the CSF - which fills the ventricular system, viewing of the surrounding tissue is possible. Movement of the endoscope – guided by video-control via the small field of view of the endoscope – is limited by the tube and the surrounding tissue. Micro-instruments, introduced through an additional canal inside the endoscope, can then be used to perform the actual minimally-invasive procedure, i.e., removing accessible mass lesions. In the case of a ventriculostomy, the thin membrane of the *lamina terminalis* is perforated, thus realizing a new CSF perfusion balance.

Other indications for minimally-invasive procedures include the formation of a CSF-filled cyst which also introduces pressure on blood vessels, nerves, or the ventricular aqueduct. To avoid these dangerous increases of pressure inside of the skull, the cyst is drained using the endoscope.

B.2.2 Virtual Endoscopy of the Ventricular System

The major problem of procedures as described above is the limited view and orientation through-out the intervention which increases the necessary time of the intervention and consequently, the inherent risks of serious complications. To overcome these drawbacks, we propose the use of a virtual endoscopy system to improve the planning of and orientation during this procedure [25, 27].

Based on pre-operative acquired MRI/3D CISS (Constructive Interference in Steady States) scans of the patient's head, the respective ventricular system is reconstructed and examined by the VIVENDI system. In particular the access ways to the target areas – i.e., the floor of the third ventricle – are explored to optimize the optical neuroendoscopic procedure. Besides the planning of neuroendoscopic interventions, virtual neuroendoscopy can also be applied to explore the stenosis of the ventricular aqueduct, an area which is not accessible with the endoscope. Figure B.5 shows various snapshots from virtual ventriculoscopy; the position and orientation of the virtual camera is represented in the lower row of Figure B.5. Each snapshot visualizes important anatomical structures, such as the *choroid plexus*, which is responsible for the production of CSF, and the *choroid plexus vein*, which is supplying the choroid plexus in Figure B.5a. The entry point for the endoscope into the third ventricle is shown in Figure B.5b. The pipe-like

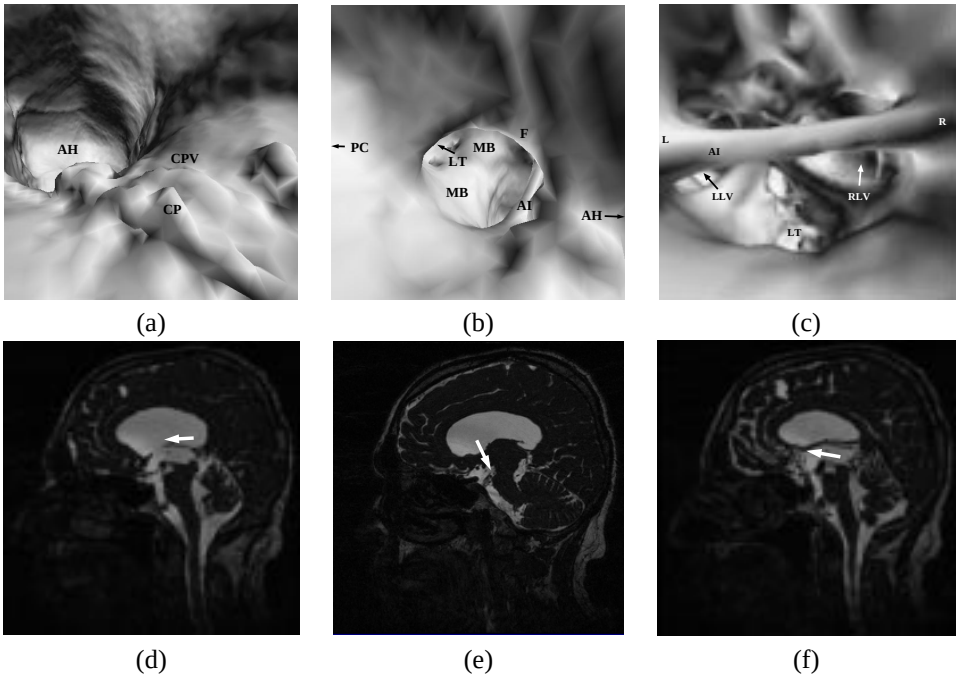


Figure B.5: Virtual Ventriculoscopy; upper row – endoscopic view, lower row – MRI/3D CISS orientation slice. (a/d) Left lateral ventricle, approach from posterior horn via pars centralis (PC) to anterior horn (AH); (b/e) foramen of Monro, approach via right lateral ventricle; (c/f) foramen of Monro, approach from third ventricle; CP = choroid plexus, CPV = choroid plexus vein, F = fornix, AI = adhesio interthalamica, MB = mamillary bodies, LT = lamina terminalis, LLV = entrance to left lateral ventricle, RLV = entrance to right lateral ventricle [27].

structure of the *adhesio interthalamica* connects the *thalamus* through the third ventricle. The upper bending of the foramen of Monro contains the *fornix*, which belongs to the *limbic system*. The limbic system is involved in the learning process which renders the fornix as a very sensitive part of the body. If it is injured by the endoscope while it enters the third ventricle, a severe learning disability can be the result. The *mamillary bodies* in the floor of the third ventricle, also belong to the limbic system. Figure B.5c shows a view from a view-point which is already not accessible for an optical endoscope. It visualizes another important structure in the floor of the third ventricle, the *lamina terminalis*, which is a thin membrane between the third ventricle and the basilar cistern (or sometimes also referred to as cistern of the lamina terminalis). This membrane is the target area for the new CSF drain of the ventricular system.

Another application for virtual endoscopy is as a 3D navigation aid to complement the current slice based navigation which tracks the tip of the endoscopic instruments and maps their registered position into the MRI dataset. The position and orientation derived from this navigation system can be loaded into VIVENDI to synchronize optical and virtual endoscopy. If a complicated anatomical situation is experienced, the area can be vir-

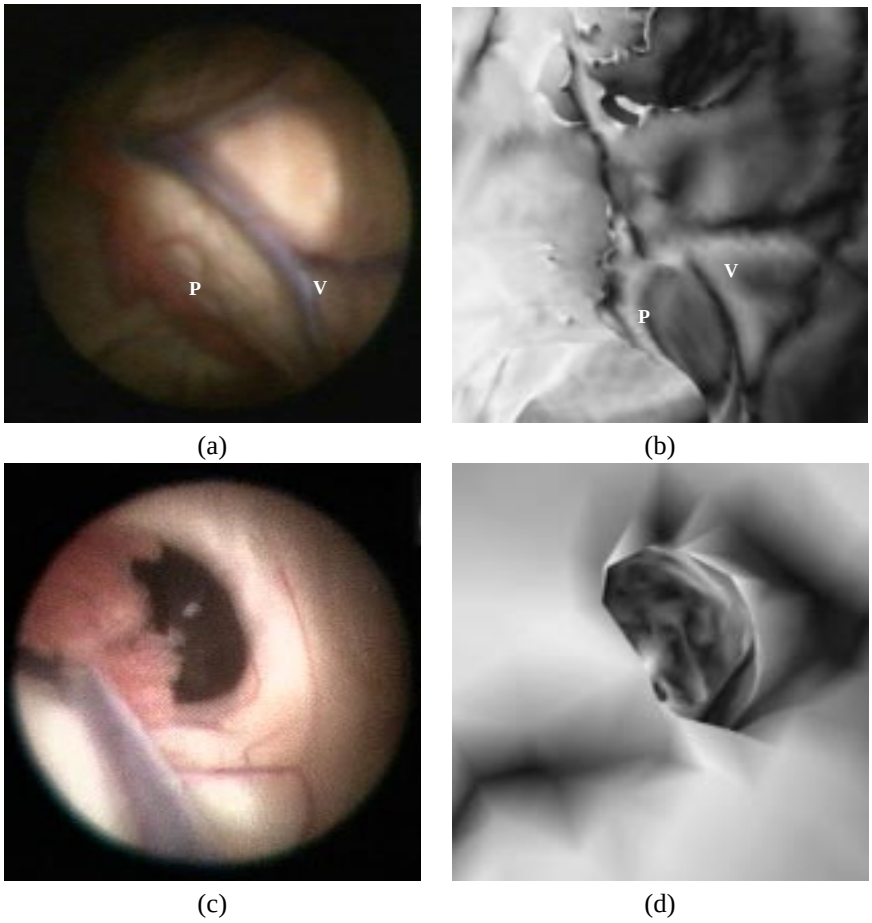


Figure B.6: Manually matched views from optical and virtual ventriculoscopy. (a, b) show the thalamostriate vein (V) and choroid plexus (P) from the right lateral ventricle. (c, d) show the right foramen of Monro, including the choroid plexus vein (Fig. B.5a) and the choroid plexus structure.

tually explored using VIVENDI to determine the appropriate action. Figure B.6 shows the (manually) matched display of optical (a, c) and virtual (b, d) endoscopy of two different datasets. Only the geometric shape information is captured by the MRI scan; all texture information, such as blood vessel color, surface color, is not available to virtual endoscopy.

B.3 Multi-modal Visualization for Neuroendoscopic Interventions

One of the most dreaded complications of minimally-invasive neurosurgery are lesion of blood vessels. Even if only a small blood vessel is injured, the resulting bleeding (“red-out”) causes a sudden loss of optical visibility through the endoscope which introduces severe difficulties for obtaining the desired results of the interventions. A more dangerous situation arises if a

major blood vessel is injured. A lesion of an artery results in a fatal mass bleeding, an usually lethal outcome of an intervention.

Unfortunately, the major basilar artery is located directly below the floor of the third ventricle without an optical visibility from the third ventricle. To avoid traumas of such blood vessels, we modified the VIVENDI-framework [25, 31] to represent multiple anatomical information of the patient data using several 3D scanning techniques [30]. For the rendering of this multiple anatomical patient data, VIVENDI provides frame-rates of more than 25 fps on an HP J7000/VISUALIZE fx6 workstation, and about 20 fps on an HP P-class/VISUALIZE fx6 PC running LINUX.

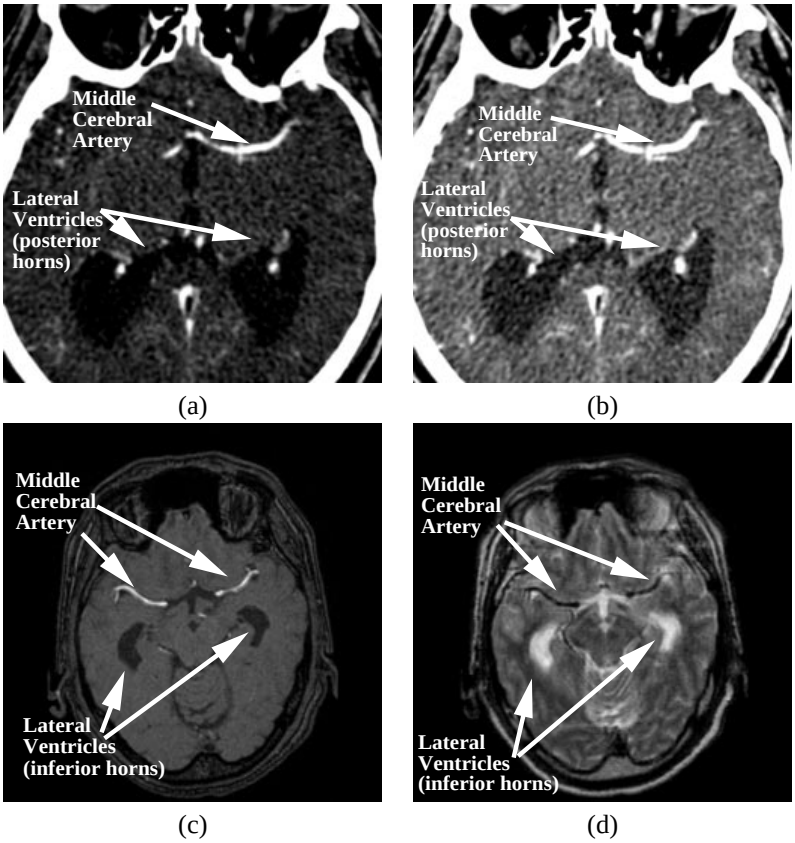


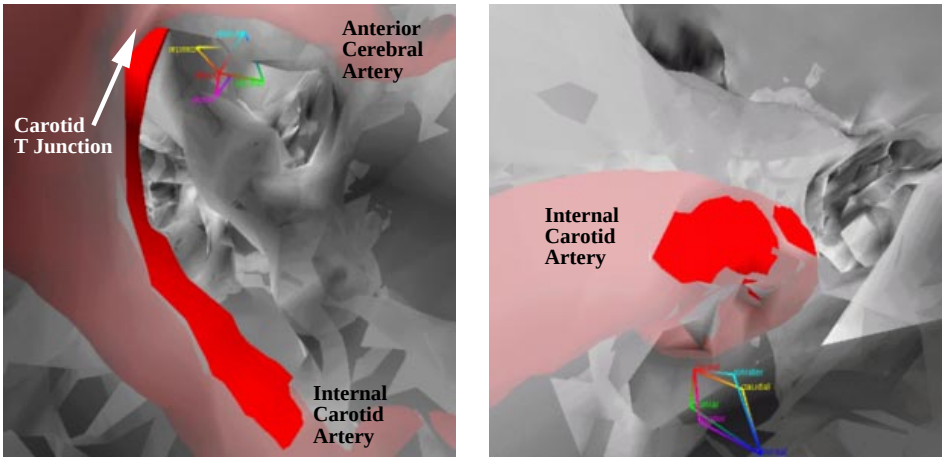
Figure B.7: Upper row: CT axial slice with the posterior horns of the lateral ventricles and middle cerebral artery with two contrast windows (a, b). Lower row: MRI axial slice with the inferior horns of the lateral ventricles and middle cerebral artery (left and right). (c) MRI angiography sequence, (d) MRI TSE sequence [30].

B.3.1 Matching Different Data Modalities

To visualize different anatomical structures, different scanning modalities and protocols are required. The associated volume datasets vary in terms of orientation, resolution, voxel dimensions, translations, and rotations. For a combined visualization of these datasets matching parameters need to

be found, which is a very difficult procedure. To minimize the necessary matching expenditure, we conducted several experiments to determine an appropriate scanning protocol, based on CT and MRI scans. For the targeted application, two anatomical structures need to be identified; the CSF-filled ventricular system and cysts, and the blood filled major arterial blood vessels in proximity to the CSF-filled target areas.

The (contrast agent-enhanced) CT scan provided a good contrast and a high resolution for the vascular system within the region of interest. However, this CT scan did not produce a sufficient contrast between the brain tissue and the CSF-filled cavities, while still preserving the complete inner surface of the cavities (Fig. B.7a and b). Furthermore, CT inherently introduces radiation, an additional drawback compared to MRI. Blood-flow in-



(a) (b)

Figure B.8: (a) Close-up to internal carotid artery. (b) View downwards on the "carotid siphon", below the carotid T junction [30].

duced MRI angiography (Time of Flight/TOF, Fig. B.7c) also reconstructs the vascular system with good quality, although the resolution is slightly lower than with a CT scan. However, it is not usable for the segmentation of CSF-filled cavities, since the ventricular system cannot be separated from the space surrounding the skull. Therefore, we perform a second MRI scan that focuses on these cavities right after the MRI angiography. We previously (see Section B.2) used an MRI 3D CISS sequence to reconstruct the ventricular system in patient datasets. Unfortunately, the different scanning orientation (sagittal/3D CISS and axial/TOF) introduced a surprisingly difficult match procedure, which qualified this sequence as impractical. Similar problems prevent a combination of MRI volumes with CT volumes, since different patient positioning and field of views pose almost insuperable matching problems. Consequently, we modified the 3D CISS sequence to an MRI TSE sequence (Turbo Spin Echo, Fig. B.7d) to reconstruct the cerebral ventricles, which provides the same orientation as the angiography sequence and unfortunately, also a smaller slice range than the 3D CISS.

However, it turned out that the resolution is sufficient for our purposes. Finally, the combination of MRI angiography and MRI TSE data delivered a satisfying matching and image quality and was henceforth used in all later experiments. Furthermore, MRI does not expose radiation, in contrast to a CT scan.

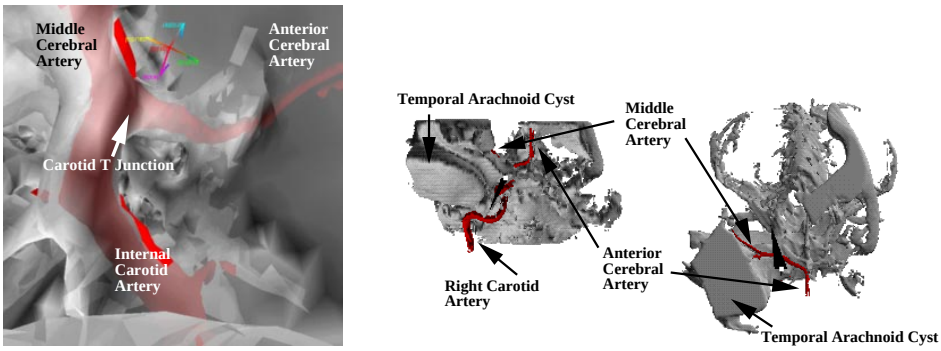
Both sequences were performed subsequently, without changing the position and orientation of the patient. It later turned out in our experiments that patient movement during both scans is negligible. Although the resolution within the axial slices is twice as large in the MRI angiography as in the MRI TSE sequence, the scans generate two well-aligned data volumes. However, the number of axial slices in the MRI data is different, and hence so is the covered scanning area. This difference requires a manual slice matching step that is performed by a neuroradiologist or neurosurgeon. The calculated axial translation generates an error which is at most the distance between two slices in the data volumes. A manifestation of this error can be found in Figures B.8 and B.9, where the red/dark, as opaque rendered artery geometry reconstructed from the MRI angiography sequence penetrates the geometry of the as transparent rendered CSF-filled cavity geometry[1]. Especially in Figure B.9a, the "original" position of the blood vessel is also visible in the geometry extracted from the MRI TSE sequence. Fortunately, the maximum error (if the matching step is correct) is always sufficiently below a critical threshold, where the "clearance area" would also cover the proposed target area of the endoscope. Figures B.10 and B.11 show a different dataset of a patient who was subject of a ventriculostomy.

Currently, the VIVENDI system is now providing the vascular topography combined with the information of the anatomical structure of the CSF-filled ventricular cavities. This information is successfully used to represent the location of the blood vessels to carefully plan the neuroendoscopic intervention. Lesions of the respective arteries can be avoided, resulting in a substantial reduction of the risk of serious complications.

B.4 Virtual Angioscopy

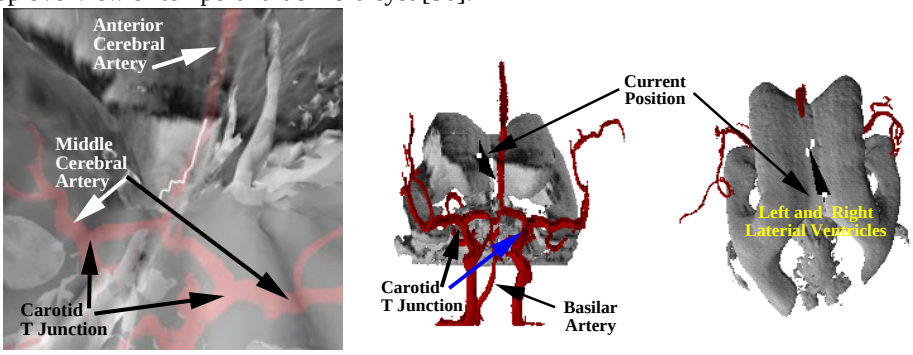
The blood circulation system is of special interest for physicians, since many injuries and diseases of this organ system can result in serious, potentially life-threatening conditions. Many of the diseases cause stenosis or aneurysms of the blood vessels. Both developments can be assessed using virtual endoscopy methods. These methods can be particularly useful in diagnostic applications where interior explorations using "real" endoscopic tools are not possible.

[1] The depth sorting of the geometry for correct transparent rendering is obtained by the view-frustum culling; all subdivision entities are sorted according to their closest depth values.



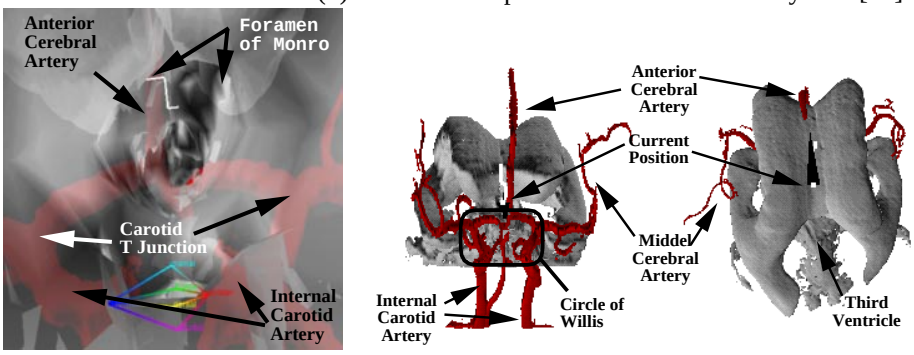
(a) (b)

Figure B.9: Temporal arachnoid cyst dataset: (a) View on to carotid T junction, where the internal carotid artery branches into the lateral middle cerebral artery and the frontal anterior cerebral artery. The red vascular geometry – extracted from MRI angiography – penetrates through the visible vascular geometry extracted from MRI TSE. (b) Frontal and top overview of temporal arachnoid cyst [30].



(a) (b)

Figure B.10: Ventriculostomy dataset: (a) Frontal view from the center of the lateral ventricles (first two ventricles); the septum between the lateral ventricles is dissolved. The white line marks the default camera path from the left lateral ventricle through the foramen of Monro into the third ventricle. (b) Frontal and top overview of ventricular system [30].



(a) (b)

Figure B.11: Ventriculostomy dataset: (a) Frontal view from the cerebral aqueduct entrance in the third ventricle. The floor of the third ventricle – the potential location for a new CSF drain – is bounded by the arterial *circle of Willis*, a potential cause for mass bleeding. (b) Frontal and top overview of ventricular system [30].

Data Acquisition

Currently, the major method for the visual representation of the angio-architecture is by 2D angiography, where a contrast agent is injected via an endovascular catheter. A subsequently taken X-ray radiograph (DSA or fluoroscopy) acquires a 2D projection of the vessels, which are of high resolution but lack spatial information.

Alternatively, other non-invasive techniques are available to visualize vessel trees in 3D. CT angiography techniques also require the injection of a contrast agent. The major drawbacks of this method are the limited resolution in Z (slice distance) and motion artifacts due to patient movements. These problems are reduced or even eliminated with the new CT multi-slice technology (i.e., Siemens SOMATOM Volume Zoom) which enable isotropic data volumes and less motion artifacts due to faster scanning, possibly triggered by a heart monitor for cardio-vascular imaging.

MRI angiography is based on the flow-induced data and hence, it does not require the injection of a contrast agent. Furthermore, it does not introduce radiation in contrast to CT-based angiography. However, the lower resolution and the relatively long scanning time of 20 minutes make MRI-based angiography sensitive to patient movement and virtually impossible to use for fast moving organs, i.e., the heart. Additionally, some MRI angiography sequences might introduce fake stenosis, due to measuring artifacts [39].

Another advanced technique is *rotational angiography* [62, 94], where up to 140 X-ray-based projections are taken from a rotation range of 200 degrees around the patients. Based on these individual projections, a 3D volume is reconstructed which represents the respective blood vessels (if a contrast agent is injected) and other anatomical structures in very high resolution. However, the relatively short scanning times of up to 13 seconds make rotational angiography still too slow for fast movements (i.e., heart beat).

Currently in clinical practice, the examinations of the vascular systems are mainly performed using Maximum-Intensity-Projections (MIP) or slicing through the 3D dataset. In contrast, using virtual endovascular methods (virtual endoscopy) enables both quantitative and qualitative analysis of the blood vessels [52].

After a segmentation and classification operation, the visual reconstruction of the blood vessels in the scanned area can be generated from the volume data. However, due to venous reflux of the contrast agent (if used), occasionally more blood vessels of the respective area are selected than the vessels of interest. This can lead to a situation where the important information is hidden behind less important information. Two techniques are applied to solve this situation. The application of virtual clips limits the segmentation of the vessel tree to the part of the vessels the physician is interested in [227, 31]. The second technique applies methods from virtual endoscopy [25, 31, 26] to generate an interactive environment for the

vascular examination from a point of view which is inside the vessels.

B.4.1 Angioscopy of Cerebral Blood Vessels

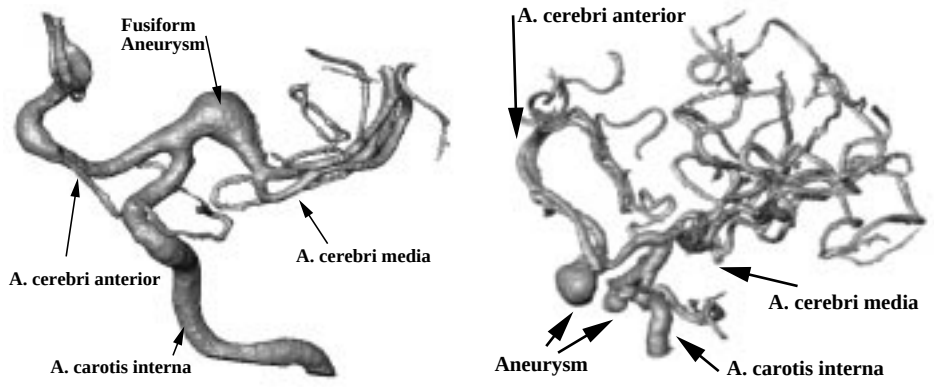


Figure B.12: Cerebral aneurysms; (a) fusiform aneurysm, (b) non-fusiform aneurysm.

A common procedure in neuroradiology is the examination of extra- and intracranial blood vessels. The major motivation behind these examinations is the diagnosis of cerebral aneurysms. Clinically, two major forms of aneurysms are distinguished; the *fusiform aneurysm* and the *non-fusiform* or *other aneurysms*. The fusiform aneurysm is an expansion of an arterial blood vessel through all of its wall layers (see Fig. B.12a). The basic criterion is that no *neck of the aneurysm* can be determined which renders the aneurysm effectively as non-treatable. In contrast, a neck or exit can be identified for *non-fusiform aneurysm* (see Fig. B.12b). From an anatomic point of view, other distinctions are possible according to the shape and location of the aneurysms. Furthermore, some aneurysms include a rupture of the inner arterial wall layers, the intima and media. The remaining adventitia layer forms a saccular deformation which is very sensitive to pressure. However, these differences cannot be easily identified which results in no clinical relevance of this distinction.

The expansion of the aneurysms can introduce pressure on other blood vessels – possibly resulting in the occlusion of that blood vessel –, or on surrounding commissures (nerve fibers). This pressure can result in severe headache, partial paralysis, or a stroke. Furthermore, strong blood flow vortices and swirls at the neck and in the aneurysms increase the risk of a highly dangerous rupture of the artery, in particular if the blood pressure is increasing due to physical exercises. This rupture in turn results in the serious destruction or necrosis of the surround brain tissue or in a lethal mass bleeding.

The usual procedures to treat aneurysms include neurosurgical and neuroradiological interventions. The major neurosurgical procedure is the exclusion of the aneurysm from the blood flow by positioning a clip on the

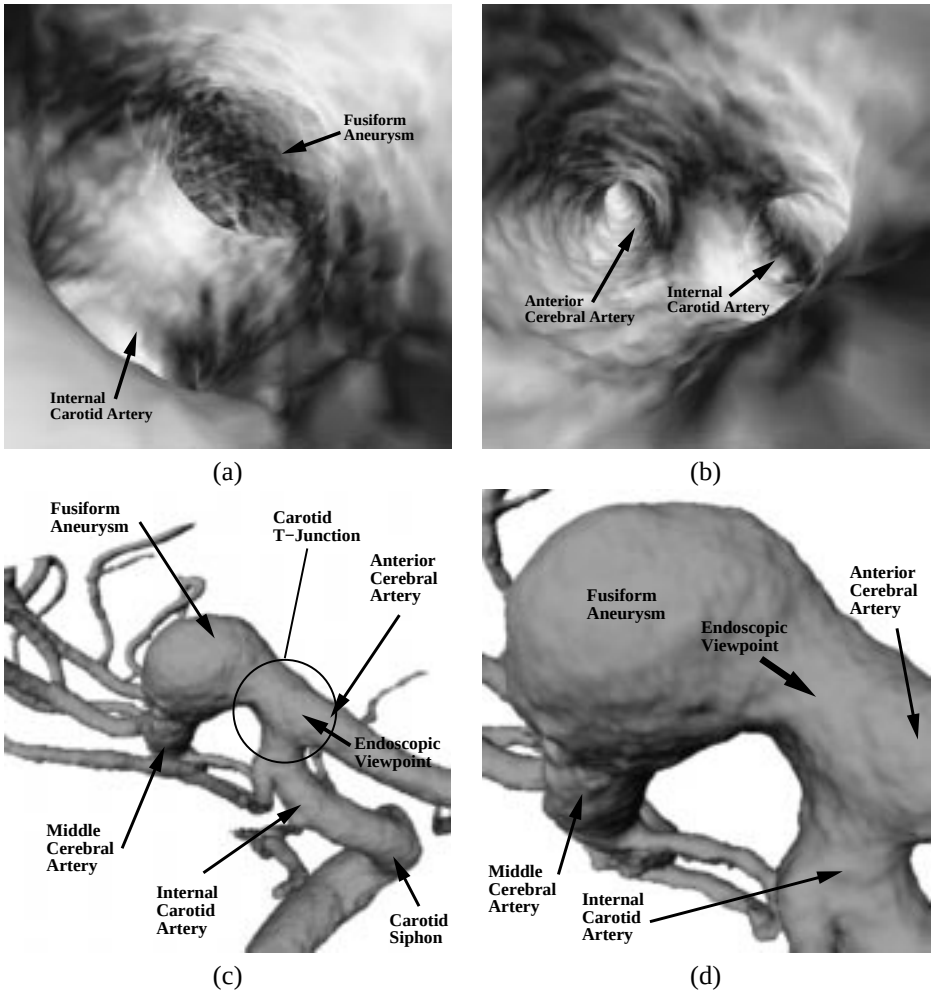


Figure B.13: Patient One – (a) Fusiform aneurysm of the middle cerebral artery. (b) Carotid T-junction as seen from the middle cerebral artery. (c) and (d) show magnifications of views on the aneurysm from outside of the blood vessels. The respective view-points of (a), (b) are annotated in (c) and (d).

neck of the aneurysm. In neuroradiology, tiny platinum spirals (“coils”) are introduced into the dome of the aneurysm using a micro-catheter, which is usually inserted via one of the femoral arteries of the legs. Together with the clotting of the thrombocytes, the coils close the aneurysm from the blood stream. All these interventions require the identification of the neck and exit of the aneurysms. However, these tasks can frequently not be achieved with standard 2D angiographies, MIPs, or slicing through the volume dataset. 3D geometry reconstructions using direct or indirect volume rendering techniques [19] have been recently introduced into the clinical practice of research hospitals [94] to provide a better understanding of the angio-architecture of aneurysms in complex blood vessel trees. In particular endovascular inspections of the blood vessel can provide valuable information on the position, orientation, and connection of the aneurysm [145].

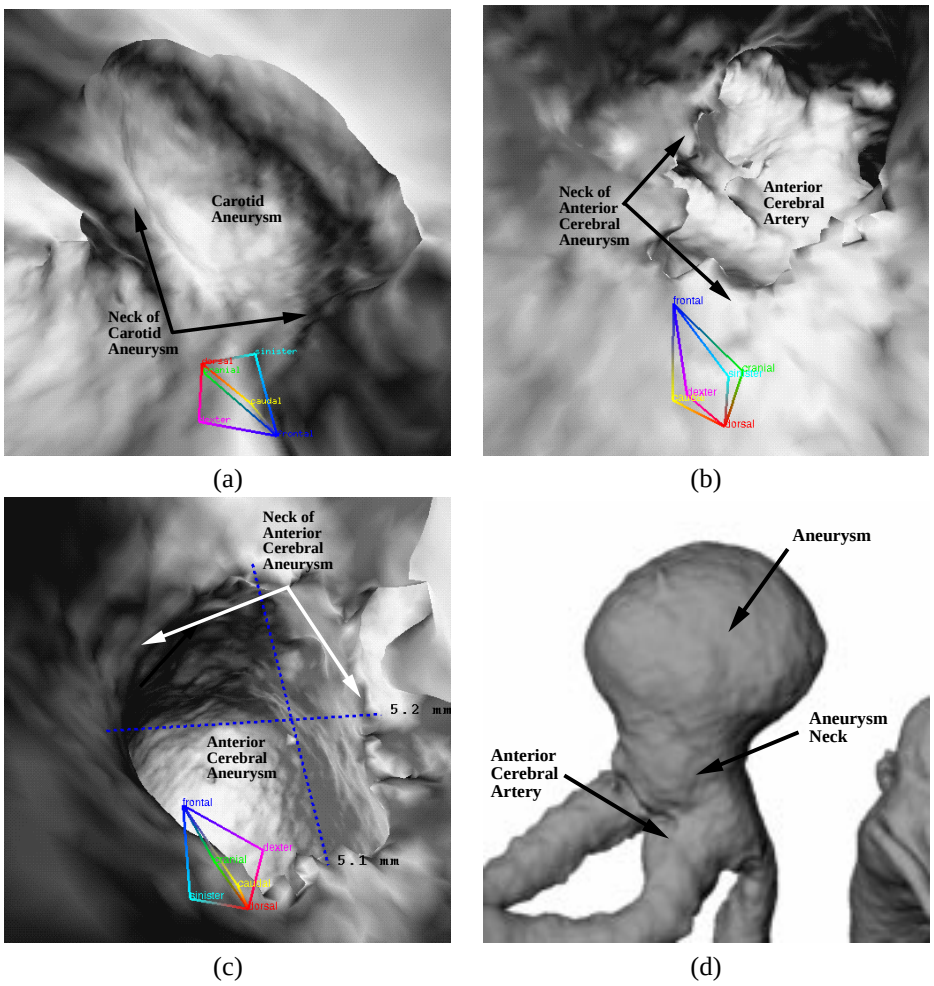


Figure B.14: Patient Two – (a) Exit of carotid aneurysm. (b) View from anterior cerebral aneurysm into anterior cerebral artery. (c) View from anterior cerebral artery into aneurysm. Measurements show the size of the aneurysm neck. (d) Anterior cerebral artery from outside.

We applied the VIVENDI framework for the endovascular identification of the neck, exit, and dome of cerebral aneurysms [31]. The geometry representing the inner surface of the blood vessels is reconstructed using data from rotational angiography, which provides an isotropic volume dataset at a very high resolution with a voxel spacing in the sub-millimeter range.

The first patient dataset shows a fusiform aneurysm of the middle cerebral artery (see Fig. B.12a). It is located close to the carotid T-junction, where the internal carotid artery branches into the anterior and middle cerebral arteries. Figure B.13a shows the exit (entrance) of the fusiform aneurysm as seen from the anterior cerebral artery at the T-junction. Below is the entrance of the internal carotid artery. An endovascular view from the middle cerebral artery to the T-junction can be seen in Figure B.13b. To the left is the entrance to the frontal anterior cerebral artery, to the right is the entrance to the internal carotid artery. In Figure B.13c and d, magnifications of the

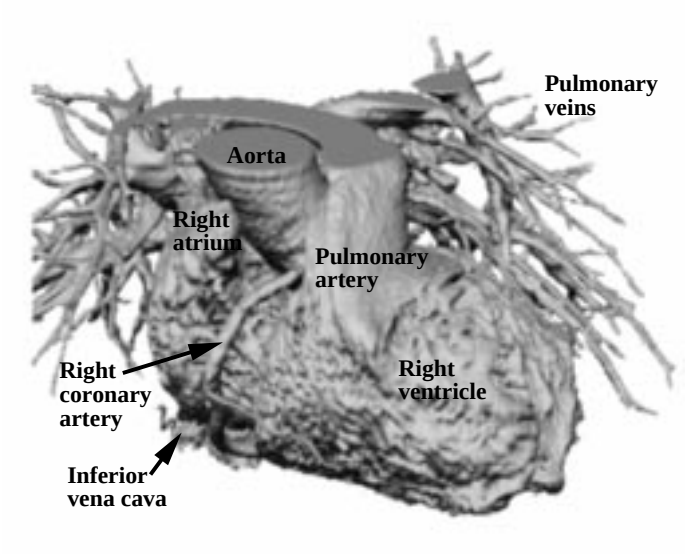
aneurysm from view-points outside of the blood vessels can be seen. Unfortunately, the fusiform nature of the aneurysm does not permit an effective treatment.

The second patient has several non-fusiform aneurysms; an aneurysm of the internal carotid artery (see Fig. B.12b right) and an aneurysm of the anterior cerebral artery (see Fig. B.12b left and Fig. B.14d). Figure B.14a shows the exit (entrance) of the carotid aneurysm. The neck is easily identified from the endovascular view. Similar, Figure B.14c shows the neck of the anterior cerebral aneurysm. Measurements suggest that the approximate radius of the aneurysm's head is 5mm. Figure B.14b shows the respective view from inside of the aneurysm to the anterior cerebral artery. Finally, Figure B.14d shows the aneurysm from an outside view. The visualizations of the anterior cerebral aneurysm show that all three branches of the anterior cerebral artery are closely located to the aneurysm. However, clipping allows the exclusion of the aneurysm without occluding one of these branches. This result was confirmed by the neurosurgical examination and treatment which applied a clip to the anterior aneurysm. The smaller carotid aneurysm was coiled during a successful interventional neuroradiologic procedure.

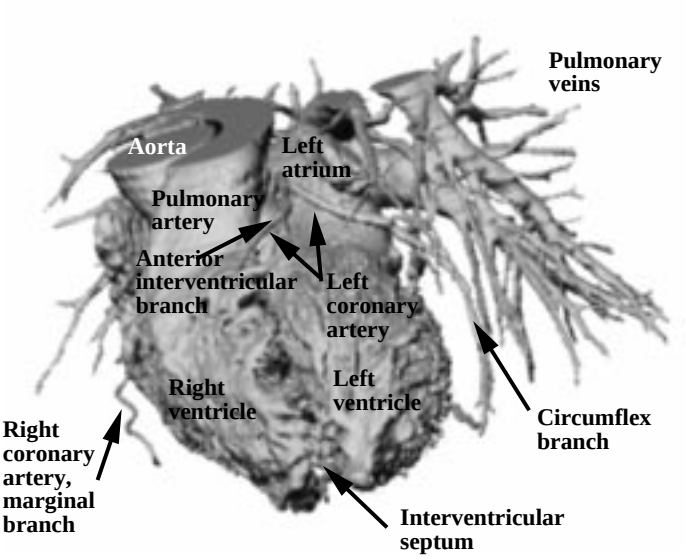
B.4.2 Angioscopy of Coronary Blood Vessels

The heart of the human body is responsible for circulating the blood through the blood vessels of the human body. It is organized into four chambers, the atria and ventricles of the heart (see Fig. B.15). One atrium and one ventricles belong to the *right heart*, or *left heart* respectively. The heart is connected with the blood vessel system through veins (leading towards the heart) and arteries (leading away from the heart). The venous blood from the various body parts arrives in the *right atrium* through the *great veins*, the *superior* and *inferior vena cava*. It enters through the right *atrioventricular valve*, the *tricuspid valve*, into the *right ventricle*, where it is pumped via the *pulmonary valve* into the *pulmonary artery* to the lungs. After the refreshing of the blood with oxygen in the lungs, it arrives through the *pulmonary veins* in the left atrium. It passed via the *left atrioventricular valve*, the *mitral valve*, into the *left ventricle* and is pumped via the *aortic valve* into the *aorta*, which distributes the blood to all body parts. Directly after the aortic valve are the entrances to the *left* and *right coronary arteries*, which supply the heart muscle with blood. The left coronary artery bifurcates into the *anterior interventricular branch* and the *circumflex branch*, which supplies most of the left heart, and it is the main source of supply of the interventricular septum – which separates the left and right ventricles –, that includes most of the conducting system of the heart. The right coronary artery extends over the right heart. It supplies this part of the heart, the interatrial septum – which separates the left and right atria –, and additionally the interventricular septum, including the sinuatrial and atrioventricular nodes, which are the major parts of the conduction system of the heart [108].

Cardiac diseases are among the number one causes of life-threatening



(a)



(b)

Figure B.15: Contrast media filled cavities of the heart: (a) Front/top view, (b) left/top view [17].

diseases in Europe and North-America. Usually, the under-supply of the heart muscle with oxygen through the blood leads to severe arrhythmia. This arrhythmia can cause an electro-mechanic decoupling of the conduction of the heart, resulting in a dangerous reduction of the pumping performance, in a possible collapse of the blood circulation, and finally the death of the patient. Even if the arrhythmia is not leading to a fatal decoupling of the conduction, the missing supply of oxygen leads to the necrosis of that part of the heart muscle, if necessary medical procedure are not applied in time. The actual cause of the under-supply of the heart muscle is usually a

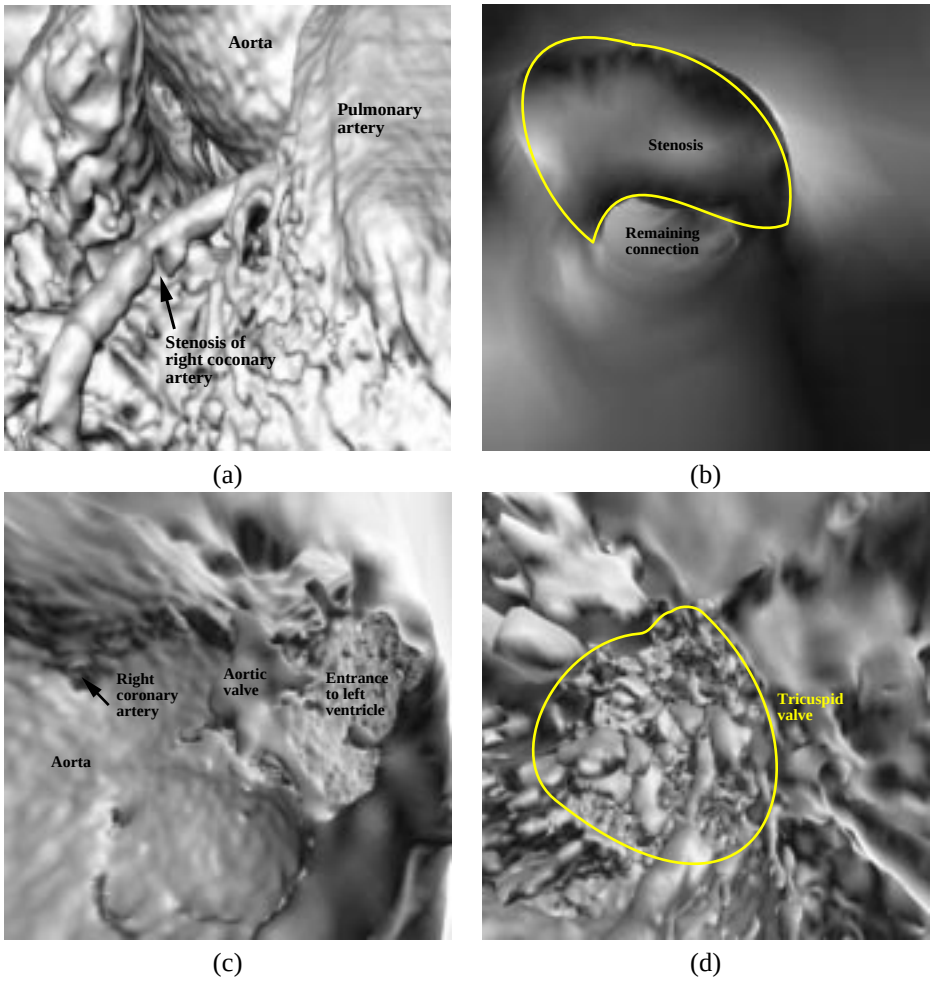


Figure B.16: Virtual endoscopy of the heart: (a) Outside reconstruction of stenosis of right coronary artery; (b) endoscopic view on stenosis; (c) view from aorta onto (remainder of) aortic valve; (d) view from right ventricle onto (remainder of) tricuspid valve [17].

stenosis or an occlusion of a coronary artery. Other reasons for arrhythmias are direct injuries of the heart muscle by force, failure of the valves of the heart, or infectious diseases.

For the diagnosis of these malfunctions, several clinical and laboratory tests are applied. For imaging, echo cardiography, coronary angiography, or nuclear medicine methods are used. However, only 3D scanning methods are able to generate volumetric data of the heart. Unfortunately, most modalities are too slow to avoid motion artifacts of the fast moving heart. More or less only ECG-triggered (Electro-Cardio-Gramme) spiral CT provides scanning which captures volumetric data of the heart that is widely motion artifact free. With the introduction of multi-slice CT, enough spatial resolution is also available for 3D coronary angiography.

Based on an anisotropic dataset from multi-slice CT coronary angiography, we explore the use of virtual endoscopy for virtual angioscopy of the

heart [17]. The dataset consist of 150 slices at a slice distance of 1.25mm. Each of the slices has a resolution of 512×512 pixels at a pixel distance of 0.59 mm. Outside views on the reconstructed, contrast media filled cavities of the heart can be seen in Figure B.15. The reconstructed geometry of the heart is very complex. From more than two million polygons, more than 80% of the geometry was culled by our visibility driven rendering, thus obtaining a frame-rate with an average of 4.2 fps on a typical path through the right heart. The low culling and frame-rate are a result of the deep visibility within the chambers of the heart which reduces possible culling benefits.

The coronary valves, which open and close at a high speed, cannot be reconstructed completely, due to motion artifacts in the scanned dataset (see Fig. B.16c and d). In particular the right atrium of the heart is subject to artifacts which are due to the injection of the contrast agent and they can be seen in the respective slices of the CT dataset. However, other important features of the anatomy of the heart of the patient are visible, such as the aorta, the pulmonary artery, the great veins, the pulmonary veins, and the coronary arteries. Of special interest are the latter one's, since most of the cardiac emergencies result from a stenosis of these arteries. Figure B.16a and b shows such a stenosis of the right coronary artery, which supplies the right heart and important parts of the conduction system of the heart. Measurements of the diameter of the blood vessel at of the stenosis provide a quantitative evaluation of the stenosis. Additionally, measurements of the volume of the ventricles provide information of the performance of the heart.

Bibliography

[1] M. Abrash. Inside Quake. *Dr. Dobb's Sourcebook*, Jan/Feb:41–45, 1996.

[2] J. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Department of Computer Science, University of North Carolina, Chapel Hill, 1990.

[3] J. Airey, J. Rohlf, and F. Brooks. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 41–50, 1990.

[4] D. Aliaga, D. Manocha, J. Cohen, S. Kumar, T. Funkhouse, M. Lin, A. Wilson, and D. Luebke. Interactive Walkthroughs of Large Geometric Datasets. In *ACM SIGGRAPH Course 18*, 2000.

[5] H. Arabnia, D. Bartz, A. Jacobsen, M. Meißner, M. Misra, H. Shen, and G. Thiruvathukal (eds). *Parallel and Distributed Processing Techniques and Applications (PDPTA)*, volume III of *ISBN 1-892512-6-8*. CSREA Press, 1998.

[6] D. Auer and L. Auer. Virtual Endoscopy - A New Tool for Teaching and Training in Neuroimaging. *International Journal of Neuroradiology*, 4:3–14, 1998.

[7] L. Avila and W. Schroeder. Interactive Visualization of Aircraft and Power Generation Engines. In *Proc. of IEEE Visualization*, pages 483–486, 1997.

[8] C. Bajaj, V. Pascucci, and D. Schikore. Fast Isocontouring for Improved Interactivity. In *Proc. of Symposium on Volume Visualization*, pages 39–46, 1996.

[9] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D. In *Proc. of Eurographics*, pages 387–396, 1996.

[10] D. Bartz. Modellbasierte Veränderungen von B-Spline-Oberflächen zur Simulation von Verformungen der Gesichtshaut. Studienarbeit, Dept. of Computer Science, University of Erlangen-Nürnberg, 1995.

[11] D. Bartz. Prototyping a Virtual Colonoscopy System. Master's thesis, Dept. of Computer Science, University of Erlangen-Nürnberg, 1996.

[12] D. Bartz, editor. *Visualization in Scientific Computing*. ISBN 3-211-83209-2. Springer Verlag-Wien, 1998.

[13] D. Bartz. Optimizing Memory Synchronization for the Parallel Construction of Recursive Tree Hierarchies. In *Proc. of Eurographics Workshop on Parallel Graphics and Visualization*, pages 53–60, 2000.

[14] D. Bartz, R. Grosso, T. Ertl, and W. Straßer. Parallel Construction and Isosurface Extraction of Recursive Tree Structures. In *Proc. of WSCG*, volume III, pages 479–486, 1998.

[15] D. Bartz and Ö. Gürvit. Haptic Navigation in Volumetric Datasets. In *Proc. of PHANToM User Research Symposium*, pages 43–47, 2000.

[16] D. Bartz, Ö. Gürvit, D. Freudenstein, H. Schiffbauer, and J. Hoffmann. Integration of Navigation, Optical and Virtual Endoscopy in Neurosurgery and Oral and Maxillofacial Surgery. In *3rd Caesarium on Computer Aided Surgery*, 2001.

[17] D. Bartz, Ö. Gürvit, M. Lanzendörfer, A. Kopp, A. Küttner, and W. Straßer. Virtual Endoscopy for Cardio Vascular Exploration. In *Proc. of Computer Assisted Radiology and Surgery*, pages 960–964, 2001.

[18] D. Bartz, J. Klosowski, and D. Staneker. Tighter Bounding Volumes for Better Occlusion Performance. In *Visual Proc. of ACM SIGGRAPH*, page 213, 2001.

[19] D. Bartz and M. Meißner. Voxels versus Polygons: A Comparative Approach for Volume Graphics. In *Proc. of Volume Graphics*, pages 33–48, 1999.

[20] D. Bartz and M. Meißner. Voxels versus Polygons: A Comparative Approach for Volume Graphics. In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, ISBN 1-85233-192-5, pages 171–184. Springer Verlag-London, 2000.

[21] D. Bartz, M. Meißner, and T. Hüttner. Extending Graphics Hardware for Occlusion Queries in OpenGL. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 97–104,158, 1998.

[22] D. Bartz, M. Meißner, and T. Hüttner. OpenGL-assisted Occlusion Culling of Large Polygonal Models. *Computers & Graphics*, 23(5):667–679, 1999.

[23] D. Bartz, C. Silva, and B. Schneider. Rendering and Visualization in Parallel Environments. In *ACM SIGGRAPH Course 13*, 2000.

[24] D. Bartz, C. Silva, and B. Schneider. Rendering and Visualization in Parallel Environments. In *IEEE Visualization, tutorial 7*, 2000.

[25] D. Bartz and M. Skalej. VIVENDI - A Virtual Ventricle Endoscopy System for Virtual Medicine. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 155–166,324, 1999.

[26] D. Bartz, M. Skalej, D. Welte, and W. Straßer. 3D Interactive Virtual Angiography. In *Proc. of Computer Assisted Radiology and Surgery*, pages 44–48, 1999.

[27] D. Bartz, M. Skalej, D. Welte, W. Straßer, and F. Duffner. A Virtual Endoscopy System for the Planning of Endoscopic Interventions in the Ventricle System of the Human Brain. In *Proc. of BiOS: Biomedical Diagnostics, Guidance and Surgical Assist Systems*, volume 3514, pages 91–100, 1999.

[28] D. Bartz, M. Skalej, D. Welte, W. Straßer, D. Freudenstein, and F. Duffner. VIVENDI - Ein Planungssystem für minimal-invasive Eingriffe in der Neurochirurgie. In *Proc. of Workshop Bildverarbeitung in der Medizin*, Informatik Aktuell, pages 197–202, 1999.

[29] D. Bartz and W. Straßer. Asynchronous Parallel Construction of Recursive Tree Structures. In *Parallel Computation (Proc. of ACPC)*, LNCS 1557, pages 427–436, 1999.

[30] D. Bartz, W. Straßer, Ö. Gürvit, , D. Freudenstein, and M. Skalej. Interactive and Multi-modal Visualization for Neuroendoscopic Interventions. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 157–164, 2001.

[31] D. Bartz, W. Straßer, M. Skalej, and D. Welte. Interactive Exploration of Extra- and Intracranial Blood Vessels. In *Proc. of IEEE Visualization*, pages 389–392,547, 1999.

[32] J. Beier, T. Diebold, H. Vehse, G. Biamino, E. Fleck, and R. Felix. Virtual Endoscopy in the Assessment of Implanted Aortic Stents. In *Proc. of Computer Assisted Radiology*, pages 183–188, 1997.

[33] J. Bentley. Multidimensional Binary Search Trees Used for Associative Search. *Communications of the ACM*, 18(9):509–516, 1975.

[34] D. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, Reading, MA, 1997.

[35] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Proc. of Symposium on Volume Visualization*, pages 91–98, 1994.

[36] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.

[37] R. Chandra, R. Menon, L. Dagum, David Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP*. Morgan-Kaufmann Publishers, San Francisco, CA, 2000.

[38] S. Chen. Quicktime VR – An Image-based Approach to Virtual Environment Navigation. In *Proc. of ACM SIGGRAPH*, pages 29–38, 1995.

[39] Z. Cho, J. Jones, and M. Singh. *Foundations of Medical Imaging*. John Wiley, New York, NY, 1993.

[40] P. Cignoni, P. Parino, E. Montani, E. Puppo, and R. Scopigno. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics*, 2(12):158–170, 1997.

[41] J. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[42] H. Cline, W. Lorensen, S. Ludke, C. Crawford, and B. Teeter. Two Algorithms for the Three-Dimensional Construction of Tomograms. *Medical Physics*, 15(3)(3):320–327, 1988.

[43] L. Cohen, P. Basuk, and J. Waye. *Practical Flexible Sigmoidoscopy*. Igaku-Shoin, New York, NY, 1995.

[44] D. Cohen-Or, Y. Chrysanthou, and C. Silva. A Survey of Visibility for Walkthrough Applications. In *ACM SIGGRAPH Course 4: Visibility: Problems, Techniques, and Applications*, 2000.

[45] D. Cohen-Or and E. Zadicario. Visibility Streaming for Network-based Walkthoughs. In *Proc. of Graphics Interface*, pages 1–7, 1998.

[46] S. Coorg and S. Teller. Temporally Coherent Conservative Visibility. In *Proc. of ACM Symposium on Computational Geometry*, pages 78–87, 1996.

[47] S. Coorg and S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[48] International Business Machines Corporation. User's Guide, IBM 3D Interaction Accelerator^{TM}. Version 1 release 2.0. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, 1995.

[49] Structural Dynamics Research Corporation. Screw driver dataset. CD-ROM, 1997.

[50] T. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, University of North Carolina at Chapel Hill, 1993.

[51] 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*, pages 69–76, 1008.

[52] C. Davis, M. Ladds, B. Romanowski, S. Wildermuth, J. Knoplioch, and J. Debatin. Human Aorta: Preliminary Results with Virtual Endoscopy Based on Three-dimensional MR Imaging Data Sets. *Radiology*, 199:37–40, 1996.

[53] P. Debevec, C. Bregler, M. Cohen, R. Szeliski, L. McMillan, and F. Sillion. Image-Based Modeling, Rendering, and Lighting. In *ACM SIGGRAPH Course 35*, 2000.

[54] E. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–270, 1959.

[55] M. Dinsmore, N. Langrana, and G. Burdea. Virtual Reality Training Simulation for Palpation of Subsurface Tumors. In *Proc. of IEEE Symposium on Virtual Reality and Applications (VRAIS)*, 1997.

[56] F. Duffner, W. Dauber, M. Skalej, and E. Grote. A New Endoscopic Tool for the CRW Stereotactic System. In *Stereotactic and Functional Neurosurgery*, volume 67(3-4), pages 213–217, 1994.

[57] F. Durand. *3D Visibility: Analytical Study and Applications*. PhD thesis, Université Joseph Fourier, 1999.

[58] F. Durand, G. Drettakis, and C. Puech. The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool. In *Proc. of ACM SIGGRAPH*, pages 89–100, 1997.

[59] B. Eberhardt, O. Etzmuß, and M. Hauth. Implicit-Explicit Schemes for Fast Animation with Particle Systems. In *Proc. of Eurographics Workshop on Computer Animation and Simulation*, 2000.

[60] B. Eberhardt, A. Weber, and W. Straßer. A Fast, Flexible Particle-System Model for Cloth Draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, 1996.

[61] U. Ernemann, M. Skalej, D. Bartz, D. Freudenstein, and K. Voigt. Virtual Endoscopy of Cerebral Vessels. In *Proc. of Neuroendoscopy*, 2000.

[62] R. Fahrig. *Computed Rotational Angiography*. PhD thesis, University of Western Ontario, 1999.

[63] G. Ferretti, D. Vining, J. Knoplioch, and M. Coulomb. Tracheobronchial Tree: Three-Dimensional Spiral CT with Bronchoscopic Perspective. *Journal of Computer Assisted Tomography*, 20(5):777–781, 1996.

[64] M. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C-21:948–960, 1972.

[65] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, Reading, MA, 2nd edition, 1996.

[66] A. Forsberg, D.Laidlaw, A. van Dam, R. Kirby, and J. Elion. Immersive Virtual Reality for Visualizing Flow Through an Artery. In *Proc. of IEEE Visualization*, pages 457–460, 2000.

[67] MPI Forum. MPI-2: Extensions to the Message-Passing Interface. Technical Report MPI 7/18/97, Message-Passing Interface Forum, 1997.

[68] D. Freudenstein, D. Bartz, R. Boldt, M. Skalej, and F. Duffner. A New System for Virtual Neuroendoscopy. In *Proc. of Neuroendoscopy*, 2000.

[69] D. Freudenstein, D. Bartz, Ö. Gürvit, M. Skalej, W. Straßer, and F. Duffner. Virtual Endoscopy for the Ventricular and Vascular System of the Human Brain. In *Proc. of Medicine meets Millenium. Brain Diseases: Advances in Neuroradiology*, 2000.

[70] D. Freudenstein, D. Bartz, M. Skalej, J. Rachinger, and F. Duffner. Virtual Neuroendoscopy. Clinical Usefulness and Problems. In *Computer Assisted Surgery and Rapid Prototyping in Medicine (CAS)*, 1999.

[71] D. Freudenstein, D. Bartz, M. Skalej, J. Rachinger, and F. Duffner. Ein neues System für die virtuelle Neuroendoskopie. *Klinische Neuroradiologie*, 3:182, 2000.

[72] D. Freudenstein, F. Duffner, D. Bartz, M. Skalej, and E. Grote. Virtual Neuroendoscopy. *Journal of Neurosurgery*, 92:536, 2000.

[73] D. Freudenstein, F. Duffner, D. Bartz, M. Skalej, and E. Grote. Virtual Neuroendoscopy. In *Proc. of AANS 68th Annual Meeting*, 2000.

[74] D. Freudenstein, F. Duffner, M. Skalej, R. Mosterz, F. Hostenstein, and D. Bartz. Virtuelle Neuroendoskopie (Abstract). In *Endoskopie Heute - Forum bildgebender Verfahren*, page 52, 1999.

[75] D. Freudenstein, R. Mostertz, F. Duffner, D. Bartz, and M. Skalej. VIVENDI - Virtual Neuroendoscopy. *Zentralblatt für Neurochirurgie*, Suppl. 75, 2000.

[76] H. Fuchs, Z. Kedem, and B. Naylor. On Visible Surface Generation by a Priori Tree Structures. In *Proc. of ACM SIGGRAPH*, pages 124–133, 1980.

[77] A. Gaddipati, R. Machiraju, and R. Yagel. Steering Image Generation with Wavelet Based Perceptual Metric. In *Proc. of Eurographics*, pages 241–251, 1997.

[78] T. Galyean. Guided Navigation of Virtual Environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 103–104, 1995.

[79] M. Garland. Multiresolution Modeling: Survey and Future Opportunities. In *Eurographics STAR report 2*, 1999.

[80] B. Garlick, D. Baum, and J. Winget. Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations. In *ACM SIGGRAPH course notes: Parallel Algorithms and Architectures for 3D Image Generation*, 1990.

[81] W. Garrett, H. Fuchs, M. Whitton, and A. State. Real-Time Incremental Visualization of Dynamic Ultrasound Volumes Using Parallel BSP Trees. In *Proc. of IEEE Visualization*, pages 235–240, 1996.

[82] A. Geist, A. Beguelin, J. Dongarra, W. Jian, R. Machek, and V. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, 1994.

[83] D. Gering, A. Nabavi, R. Kikinis, W. Grimson, N. Hata, P. Everett, F. Jolesz, and W. Wells. An Integrated Visualization System for Surgical Planning and Guidance using Image Fusion and Interventional Imaging. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 1999.

[84] Z. Gigus and J. Malik. Computing the Aspect Graph for Line Drawings of Polyhedral Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, 1990.

[85] E. Gobbetti, P. Pili, A. Zorcolo, and M. Tuveri. Interactive Virtual Angioscopy. In *Proc. of IEEE Visualization*, pages 435–438, 1998.

[86] J. Goldsmith and J. Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7:14–20, 1987.

[87] S. Gottschalk, M. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proc. of ACM SIGGRAPH*, pages 171–180, 1996.

[88] H. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan-Kaufmann Publishers, San Francisco, CA, 1993.

[89] N. Greene. *Hierarchical Rendering of Complex Environments*. PhD thesis, Computer and Information Science, University of California, Santa Cruz, 1995.

[90] N. Greene. Hierarchical Polygon Tiling with Coverage Masks. In *Proc. of ACM SIGGRAPH*, pages 65–74, 1996.

[91] N. Greene. Occlusion Culling with Optimized Hierarchical Buffering. In *Visual Proc. of ACM SIGGRAPH*, page 261, 1999.

[92] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In *Proc. of ACM SIGGRAPH*, pages 231–238, 1993.

[93] R. Grosso, T. Ertl, and R. Klier. A Load-Balancing Scheme for Parallelizing Hierarchical Splatting on a MPP System with Non-uniform Memory Access Architecture. In *Proc. of High Performance Computing for Computer Graphics and Visualization*, pages 125–134, 1995.

[94] Ö. Gürvit, M. Skalej, R. Riekmann, U. Ernemann, and K. Voigt. Rotational Angiography and 3D Reconstruction in Neuroradiology. *electro medica*, 68(1):31–37, 2000.

[95] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Proc. of ACM SIGGRAPH*, pages 197–206, 1993.

[96] T. He and L. Hong. Reliable Navigation for Virtual Endoscopy. In *Proc. of IEEE Medical Imaging*, 1999.

[97] H. Hege, T. Höllerer, and D. Stalling. Volume Rendering – Mathematical Foundations and Algorithmic Aspects. Technical Report TR 93-7, Konrad-Zuse Zentrum für Informationstechnik Berlin, 1993.

[98] M. Held, J. Klosowski, and J. Mitchell. Real-time Collision Detection for Motion Simulation Within Complex Environment. In *Visual Proc. of ACM SIGGRAPH*, page 151, 1996.

[99] Hewlett-Packard. Occlusion test, preliminary. Hewlett Packard Company, Palo Alto, available from http://oss.sgi.com/projects/ogl-sample/registry/HP/occlusion_test.txt, 1997.

[100] Hewlett-Packard. Jupiter 1.0. CD-ROM, 1998.

[101] Hewlett-Packard. Jupiter 1.0 Specification. Technical report, Hewlett Packard Company, Palo Alto, 1998.

[102] R. Hietala and J. Oikarinen. A Visibility Determination Algorithm for Interactive Virtual Endoscopy. In *Proc. of IEEE Visualization*, pages 29–36, 2000.

[103] E. Hiti. Hierarchischer Z-Buffer – Implementierung und Analyse. Master's thesis, Dept. of Computer Science (WSI), University of Tübingen, 1998.

[104] M. Hoch. Modellbasierte Animation von Gesichtsausdrücken. Master's thesis, Dept. of Computer Science, University of Erlangen-Nürnberg, 1992.

[105] H. Höhne and R. Bernstein. Shading 3D-Images from CT using Gray-level Gradients. In *IEEE Transactions on Medical Imaging*, volume MI-5, pages 45–47, 1986.

[106] K. Höhne, M. Bomans, M. Riemer, R. Schubert, and U. Tiede. A 3D Anatomical Atlas Based on a Volume Model. *IEEE Computer Graphics and Applications*, 12:72–78, 1992.

[107] K. Höhne, B. Pflesser, A. Pommert, K. Priesmeyer, M. Riemer, T. Schiemann, R. Schubert, U. Tiede, H. Frederking, S. Gehrmann, S. Noster, and U. Schumacher. *VOXEL-MAN 3D Navigator: Inner Organs. Regional, Systemic and Radiological Anatomy*. Springer-Verlag Electronic Media, Heidelberg, Germany, CD-ROM edition, 2000.

[108] W. Hollinshead and C. Rosse. *Textbook of Anatomy*. J.B. Lippincott Company, Philadelphia, PA, 4th edition, 1985.

[109] L. Hong, A. Kaufman, Y. Wei, A. Viswambharan, M. Wax, and Z. Liang. 3D Virtual Colonoscopy. In *Proc. of IEEE Symposium on Biomedical Visualization*, pages 26–32, 1995.

[110] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive Navigation in the Human Colon. In *Proc. of ACM SIGGRAPH*, pages 27–34, 1997.

[111] J. Hornegger. Rotational Angiography. Siemens Medical Systems, Personal Communications, 2000.

[112] G. Hounsfield. A Method of and Apperatus for Examination of a Body by Radiation such as X-ray or Gamma Radiation. British Patent No. 1283915, 1972.

[113] P. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[114] T Hudson, D. Manocha, J. Cohen, M. Lin, Kenneth E. Hoff, and H. Zhang. Accelerated Occlusion Culling Using Shadow Frusta. In *Proc. of ACM Symposium on Computational Geometry*, pages 2–10, 1997.

[115] T. Hüttner, M. Meißner, and D. Bartz. OpenGL-assisted Visibility Queries of Large Polygonal Models. Technical Report WSI-98-6, ISSN 0946-3852, Dept. of Computer Science (WSI), University of Tübingen, 1998.

[116] T. Itoh and K. Koyamada. Automatic Isosurface Propagation Using an Extrema Graph and Sorted Boundary Cell Lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.

[117] C. Jones. A New Approach to the ”Hidden Line” Problem. *The Computer Journal*, 14(3):232–237, 1971.

[118] W. Kalender. *Computer Tomography*. PUBLICIS MCD Verlag, München, Germany, 2000.

[119] A. Kaufman, R. Avila, L. Sobierajski, and R. Yagel. Volume Visualization Algorithms and Applications. In *IEEE Visualization, tutorial 1*, 1996.

[120] T. Kay and J. Kajiya. Ray Tracing Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 269–278, 1986.

[121] E. Keeve, S. Girod, P. Pfeifle, and B. Girod. Anatomy-based Facial Tissue Modelling Using the Finite Element Method. In *Proc. of IEEE Visualization*, pages 21–28, 1996.

[122] A. Kela and M. Wynn. Parallel Computation of Exact Ouadtree and Octree Approximations on Distributed Memory Multiprocessors. In *Proc. of 4th Conference on Hypercubes, Concurrent Computers, and Applications*, pages 1193–1196, 1989.

[123] D. Kenwright, D. Banks, S. Bryson, R. Haines, R. Liere, and S. Uselton. Panel: Automation or Interaction: What’s Best for Big Data. In *Proc. of IEEE Visualization*, pages 491–495, 1999.

[124] D. Kirk. Unsolved Problems and Opportunities for High-quality, High-performance 3D Graphics on a PC Platform. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 11–13, 1998.

[125] J. Klosowski. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments*. PhD thesis, State University of New York, Stony Brook, 1998.

[126] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[127] J. Klosowski and C. Silva. Rendering on a Budget. In *Proc. of IEEE Visualization*, pages 155–122, 1999.

[128] R. Koch, M. Gross, F. Carls, D. Büren, G. Fankhauser, and Y. Parish. Simulating Facial Surgery Using Finite Element Methods. In *Proc. of ACM SIGGRAPH*, pages 421–428, 1996.

[129] J. Koenderink and A. van Doorn. The Singularities of the Visual Mapping. *BioCyber*, 24(1):51–59, 1976.

[130] H. Kretschmann and W. Weinrich. *Neurofunctional Systems*. Thieme Interactive, Stuttgart, Germany, CD-ROM edition, 1999.

[131] U. Kühnapfel, H. Cakmak, and H. Maaß. Endoscopic Surgery Training Using Virtual Reality and Deformable Tissue Simulation. *Computers & Graphics*, 24(5):671–682, 2000.

[132] S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical Backface Computation. *Computers & Graphics*, 23(5):681–692, 1999.

[133] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proc. of ACM SIGGRAPH*, pages 451–458, 1994.

[134] A. Laghi, P. Pavone, V. Panebianco, I. Carbone, and L. Francone. Volume-rendered Virtual Colonoscopy: Preliminary Clinical Experience. In *Proc. of Computer Assisted Radiology and Surgery*, pages 171–175, 1999.

[135] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *24th Annual International Symposium on Computer Architecture (ISCA)*, pages 241–251, 1997.

[136] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. In *Proc. of ACM SIGGRAPH*, pages 285–288, 1991.

[137] T. Lehmann, W. Oberschelp, E. Pelikan, and R. Repges. *Bildverarbeitung für die Medizin: Grundlagen, Modelle, Methoden, Anwendungen*. Springer Verlag, Heidelberg, Germany, 1997.

[138] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[139] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[140] B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction into Volume Rendering*. Prentice Hall, Upper Saddle River, NJ, 1998.

[141] Y. Livnat, H. Shen, and C. Johnson. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):73–84, 1996.

[142] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proc. of ACM SIGGRAPH*, pages 163–169, 1987.

[143] W. Lorensen, F. Jolesz, and R. Kikinis. The Exploration of Cross-Sectional Data with a Virtual Endoscope. In R. Satava and K. Morgan, editors, *Interactive Technology and New Medical Paradigms for Health Care*, pages 221–230. IOS Press, 1995.

[144] D. Luebke and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 105–106, 1995.

[145] B. Marro, D. Galanaud, C. A. Valery, A. Zouaoui, A. Biondi, A. Casasco, M. Sahel, and C. Marsault. Intracranial Aneurysm: Inner View and Neck Identification with CT Angiography Virtual Endoscopy. *Journal of Computer Assisted Tomography*, 21(4):587–589, 1997.

[146] S. Marschner and R. Lobb. An Evaluation of Reconstruction Filters for Volume Rendering. In *Proc. of IEEE Visualization*, pages 100–107, 1994.

[147] N. Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[148] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 1999.

[149] M. Meißner and D. Bartz. Translucent and Opaque Direct Volume Rendering for Virtual Endoscopy Applications. In *Proc. of Volume Graphics*, pages 375–384, 2001.

[150] M. Meißner, D. Bartz, and R. Günther. Visibility Driven Rasterization. In *Visual Proc. of ACM SIGGRAPH*, page 275, 2000.

[151] M. Meißner, D. Bartz, R. Günther, and W. Straßer. Visibility Driven Rasterization. Technical Report WSI-2000-16, Dept. of Computer Science (WSI), University of Tübingen, 2000.

[152] M. Meißner, D. Bartz, R. Günther, and W. Straßer. Visibility Driven Rasterization. *Computer Graphics Forum*, 20(4):283–294, 2001.

[153] M. Meißner, D. Bartz, T. Hüttner, G. Müller, and J. Einighammer. Generation of Subdivision Hierarchies for Efficient Occlusion Culling of Large Polygonal Models. Technical Report WSI-99-13, ISSN 0946-3852, Dept. of Computer Science (WSI), University of Tübingen, 1999.

[154] M. Meißner, D. Bartz, T. Hüttner, G. Müller, and J. Einighammer. Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models. In *Vision, Modeling, and Visualization*, pages 225–232, 2001.

[155] M. Meißner, U. Hoffman, and W. Straßer. Enabling Classification and Shading for 3D Texture Mapping Based Volume Rendering. In *Proc. of IEEE Visualization*, pages 207–214, 1999.

[156] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A Practical Evaluation of Four Popular Volume Rendering Algorithms. In *Proc. of Symposium on Volume Visualization and Graphics*, pages 81–90, 2000.

[157] M. Meißner, U. Kanus, and W. Straßer. VIZARD II, A PCI-Card for Real-Time Volume Rendering. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 61–68, 1998.

[158] T. Möller and E. Haines. *Real-Time Rendering*. A. K. Peters, Natick, MA, 1999.

[159] S. Morein. ATI Radeon HyperZ Technology. In *Hot 3D Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 1–11, 2000.

[160] C. Morosi, G. Ballardini, and P. Pisani. Diagnostic Accuracy of the Double-Contrast Enema for Colonic Polyps in Patients with or without Diverticular Disease. *Gastrointestinal Radiology*, 16:346–347, 1991.

[161] K. Mueller and R. Crawfis. Elminating Popping Artifacts in Sheet Buffer-Based Splatting. In *Proc. of IEEE Visualization*, pages 239–246, 1998.

[162] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. High-quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):116–134, 1999.

[163] G. Müller and D. Fellner. Hybrid Scene Structuring with Application to Ray Tracing. In *Proc. of International Conference on Visual Computing (ICVC)*, pages 19–26, 1999.

[164] B. Naylor. Partitioning Tree Image Representation and Generation from 3D Geometric Models. In *Proc. of Graphics Interface*, pages 201–212, 1992.

[165] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan. Interactive Ray Tracing for Isosurface Rendering. In *Proc. of IEEE Visualization*, pages 233–238, 1998.

[166] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro Real-Time Ray-Casting System. In *Proc. of ACM SIGGRAPH*, pages 251–260, 1999.

[167] H. Pfister, M. Zwicker, J. Baar, and M. Gross. Surfels: Surface Elements as Rendering Primitives. In *Proc. of ACM SIGGRAPH*, pages 335–342, 2000.

[168] S. Pieper. *CAPS: Computer Aided Plastic Surgery*. PhD thesis, Massachussets Institute of Technology, 1992.

[169] W. Plantinga and C. Dyer. Visibility, Occlusion and the Aspect Graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[170] T. Porter and T. Duff. Compositing Digital Images. In *Proc. of ACM SIGGRAPH*, pages 253–259, 1984.

[171] J. Rodenwaldt, L. Kopka, R. Roedel, A. Margas, and E. Grabbe. 3D Virtual Endoscopy of the Upper Airways: Optimization of the Scan Parameters in a Cadaver Phantom and Clinical Assessment. *Journal of Computer Assisted Tomography*, 21(3):405–411, 1997.

[172] J. Rohen. *Topographische Anatomie*. Schattauer Verlag, Stuttgart, Germany, 8th edition, 1987.

[173] J. Rohlf and J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In *Proc. of ACM SIGGRAPH*, pages 381–394, 1994.

[174] G. Rubin, C. Beaulieu, V. Argiro, H. Ringl, A. Norbash, J. Feller, M. Dake, R. Jeffrey, and S. Napel. Perspective Volume Rendering of CT and MR Images: Application for Endoscopic Imaging. In *Radiology*, volume 199, pages 321–330, 1996.

[175] S. Rubin and T. Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 11–116, 1980.

[176] S. Rusinkiewicz and M. Levoy. Qsplats: A Multiresolution Point Rendering System for Large Meshes. In *Proc. of ACM SIGGRAPH*, pages 343–352, 2000.

[177] P. Sabella. A Rendering Algorithm for Visualizing 3D Scalar Fields. In *Proc. of ACM SIGGRAPH*, pages 51–58, 1988.

[178] T. Saito and J. Toriwaki. New Algorithms for Euclidian Distance Transformation of an N-Dimensional Digitized Picture with Applications. *Pattern Recognition*, 27(11):1551–1565, 1994.

[179] G. Sakas and S. Walter. Extracting Surfaces from Fuzzy 3D Ultrasonic Data. In *Proc. of ACM SIGGRAPH*, pages 465–474, 1995.

[180] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1994.

[181] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative Volumetric Visibility with Occluder Fusion. In *Proc. of ACM SIGGRAPH*, pages 229–238, 2000.

[182] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1998.

[183] N. Scott, D. Olsen, and E. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *The Hewlett-Packard Journal*, (May):28–34, 1998.

[184] L. Serra, W. Nowinski, T. Poston, N. Hern, L. Meng, C. Guan, and P. Pillay. The Brain Bench: Virtual Tools for Stereotactic Frame Surgery. *Medical Image Analysis*, 1(4):317–329, 1997.

[185] K. Severson. VISUALIZE fx Graphics Accelerator Hardware. Technical report, Hewlett Packard Company, Palo Alto, available from http://www.hp.com/workstations/support/documentation/whitepapers.html, 1999.

[186] SGI. Power Challenge. Technical report, Silicon Graphics Inc., Mountain View, 1994.

[187] SGI. Onyx2 Reality and Onyx2 InfiniteReality. Technical report, Silicon Graphics Inc., Mountain View, 1997.

[188] SGI. *OpenGL Optimizer Manual*. Silicon Graphics Inc., Mountain View, 1997.

[189] SGI. Silicon Graphics 320, Visual Workstation. OpenGL Programming Guide, available from http://support.sgi.com/nt/index.html, 1999.

[190] R. Shadidi, V. Argiro, S. Napel, L. Gray, H. McAdams, G. Rubin, C. Beaulieu, R. Jeffrey, and A. Johnson. Assessment of Several Virtual Endoscopy Techniques Using Computed Tomography and Perspective Volume Rendering. In *Proc. of Visualization in Biomedical Computing*, LNCS 1131, pages 521–528, 1996.

[191] R. Shekkar, W. Fayyad, R. Yagel, and J. Frederick. Octree-Based Decimation of Marching Cubes Surface. In *Proc. of IEEE Visualization*, pages 287–294, 1996.

[192] H. Shen, C. Hansen, Y. Livnat, and C. Johnson. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proc. of IEEE Visualization*, pages 287–294, 1996.

[193] M. Skalej, D. Bartz, and F. Duffner. VIVENDI – Virtuelle Ventrikelendoskopie (Poster). In *Jahrestagung der deutschen Ges. für Neuroradiologie, Bad Homburg*, 1998.

[194] J. Snyder and J. Lengyel. Visibility Sorting and Compositing without Splitting for Image Layer Decompositions. In *Proc. of ACM SIGGRAPH*, pages 219–231, 1998.

[195] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. TIVoR: An Interactive Visualization and Navigation Tool for Medical Volume Data. In *Proc. of WSCG*, pages 361–372, 1998.

[196] S. Sonntag, G. Glombitza, F. Floemer, M. Knopp, W. Lamade, and H. Meinzer. Registration of Spatially Consecutive 3D MR Data Sets for an Enhanced Structural Analysis of Intrahepatic Vessel Trees. In

Proc. of Computer Assisted Radiology and Surgery, pages 129–133, 1999.

[197] D. Staneker. Ein hybrider Ansatz zur effizienten Verdeckungsrechnung. Master's thesis, Dept. of Computer Science (WSI), University of Tübingen, 2001.

[198] W. Straßer. *Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten*. PhD thesis, Technische Universität Berlin, 1974.

[199] O. Sudarsky and C. Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. In *Proc. of Eurographics*, pages 249–258, 1996.

[200] J. Sundsten. *The Digital Anatomist: Interactive Brain Atlas*. University of Washington, Seattle, WA, CD-ROM edition, 1999.

[201] L. Swift, T. Johnson, and E. Livadas. Parallel Creation of Linear Octrees from Quadtree Slices. *Parallel Processing Letters*, 4(4):447–453, 1994.

[202] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Upper Saddle River, NJ, 1992.

[203] G. Taubin. 3D Geometry Compression and Progressive Transmission. In *Eurographics STAR report 3*, 1999.

[204] G. Taubin, M. Deering, C. Gotsman, S. Gumhold, and J. Rossignac. 3D Geometry Compression. In *ACM SIGGRAPH Course 38*, 2000.

[205] S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Department of Computer Science, University of California, Berkeley, 1992.

[206] S. Teller and C. Sequin. Visibility Pre-processing for Interactive Walkthroughs. In *Proc. of ACM SIGGRAPH*, pages 61–69, 1991.

[207] P. Teo and D. Heeger. Perceptual Image Distortion. In *Proc. of Conference on Image Processing*, pages 982–986, 1994.

[208] U. Tiede, K. Höhne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of Medical 3D-rendering Algorithms. *IEEE Computer Graphics and Applications*, 19(2):41–53, 1990.

[209] TOP500. Top500 supercomputer sites. http://www.top500.org, 2000.

[210] H. Tuy and L. Tuy. Direct 2D Display of 3D Objects. *IEEE Computer Graphics and Applications*, 4(10):29–33, 1984.

[211] M. Uesbeck. Automatische Quantifizierung von Gewebeschnitten für die Diagnose von Entzündungsreaktionen in Organen. http://www.gris.uni-tuebingen.de/people/staff/uesbeck/beschreibung_ikfz.html, 1999.

[212] WSI/GRIS University of Tübingen. Volume Datasets. http://www.volvis.org, 2000.

[213] G. van den Bergen. Efficient Collision Detection of Complex Deformable Models Using AABB Trees. *Journal of Graphics Tools*, 3(5):1–13, 1997.

[214] A. van Gelder and K. Kim. Direct Volume Rendering with Shading via Three-dimensional Textures. In *Proc. of Symposium on Volume Visualization*, pages 23–30, 1996.

[215] M. Vannier, J. Marsh, and O. Warren. Three Dimensional Computer Graphics for Craniofacial Surgical Planning and Evaluation. In *Proc. of ACM SIGGRAPH*, pages 263–273, 1983.

[216] A. Varshney, F. Brooks, D. Richardson, W. Wright, and D. Manocha. Defining, Computing, and Visualizing Molecular Interfaces. In *Proc. of IEEE Visualization*, pages 36–43, 1995.

[217] A. Varshney, F. Brooks, and W. Wright. Linearly Scalable Computation of Smooth Molecular Surfaces. *IEEE Computer Graphics and Applications*, 14(5):19–25, 1994.

[218] D. Vining, R. Shifrin, E. Grishaw, K. Liu, and R. Choplin. Virtual Colonoscopy (abstract). In *Radiology*, volume 193(P), page 446, 1994.

[219] D. Vining, R. Shifrin, E. Haponik, K. Liu, and R. Choplin. Virtual Bronchoscopy (abstract). In *Radiology*, volume 193(P), page 261, 1994.

[220] D. Vining, D. Stelts, D. Ahn, P. Hemler, Y. Ge, G. Hunt, C. Siege, D. McCorquodale, M. Sarojak, and G. Ferretti. FreeFlight: A Virtual Endoscopy System. In *First Joint Conference, Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery*, LNCS 1205, pages 413–416, 1997.

[221] M. Wan, Q. Tang, A. Kaufman, Z. Liang, and M. Wax. Volume Rendering Based Interactive Navigation within the Human Colon. In *Proc. of IEEE Visualization*, pages 397–400, 1999.

[222] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The Radomized Z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In *Proc. of ACM SIGGRAPH*, 2001.

[223] K. Waters. A Muscle Model for Animating 3D-Facial Expression. In *Proc. of ACM SIGGRAPH*, pages 17–24, 1987.

[224] K. Waters and D. Terzopoulos. A Physical Model of Facial Tissue and Muscle Articulation. In *Proc. of Visualization in Biomedical Computing*, pages 77–82, 1990.

[225] R. Wegenkittl, A. Vilanova, B. Hegedüs, D. Wagner, M. Freund, and E. Gröller. Mastering Interactive Virtual Bronchioscopy on a Low-End PC. In *Proc. of IEEE Visualization*, pages 461–465, 2000.

[226] H. Weghorst, G. Hooper, and D. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

[227] D. Welte and U. Klose. Segmentation and Selective Imaging of Arteries and Veins from Contrast-Enhanced MRA Data. In *Proc. of European Congress of Radiology (ECR)*, 1999.

[228] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Proc. of ACM SIGGRAPH*, pages 169–177, 1998.

[229] L. Westover. Footprint Evaluation for Volume Rendering. In *Proc. of ACM SIGGRAPH*, pages 367–376, 1990.

[230] L. Westover. *SPLATTING: A Parallel Feed-Forward Volume Rendering Algorithm*. PhD thesis, University of North Carolina at Chapel Hill, 1991.

[231] J. Wilhelms and A. van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[232] J. Wilhelms, A. van Gelder, P. Tarantino, and J. Gibbs. Hierarchical and Parallelizable Direct Volume Rendering for Irregular and Multiple Grids. In *Proc. of IEEE Visualization*, pages 57–64, 1996.

[233] J. Wilhelms and A. van Geldern. A Coherent Projection Approach for Direct Volume Rendering. In *Proc. of ACM SIGGRAPH*, pages 275–284, 1991.

[234] P. Williams and S. Uselton. Metrics and Generation Specifications for Comparing Volume-rendered Images. *Journal of Visualization and Computer Animation*, 10:159–178, 1999.

[235] C. Wittenbrink and K. Kim. Data Dependent Optimizations for Permutation Volume Rendering. In *Proc. of SPIE Visual Data Exploration and Analysis V*, pages 284–294, 1998.

[236] C. Wittenbrink, T. Malzbender, and M. Goss. Opacity-weighted Color Interpolation for Volume Sampling. In *Proc. of Symposium on Volume Visualization*, pages 135–142, 1998.

[237] P. Wonka and D. Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. In *Proc. of Eurographics*, pages 51–60, 1999.

[238] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. Addison Wesley, Reading, MA, 2nd edition, 1997.

[239] F. Xie and M. Shantz. Adaptive Hierarchical Visibility in a Tiled Architecture. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 75–84, 1999.

[240] R. Yagel and A. Kaufman. Template-based Volume Viewing. In *Proc. of Eurographics*, pages 153–167, 1992.

[241] S. You, L. Hong, M. Wan, K. Junyapreasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang. Interactive Volume Rendering for Virtual Colonoscopy. In *Proc. of IEEE Visualization*, pages 343–346, 1997.

[242] H. Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, Department of Computer Science, University of North Carolina, Chapel Hill, 1998.

[243] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In *Proc. of ACM SIGGRAPH*, pages 77–88, 1997.

[244] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for Modeling and Animation. In *ACM SIGGRAPH Course 23*, 2000.

Lebens- und Bildungsgang

22. Juli 1967	geboren in Simmern, Hunsrück
1977 - 1983	Integrierte Gesamtschule Kastellaun, Hunsrück
1983 - 1986	Herzog-Johann-Gymnasium, Simmern, Hunsrück Abschluß: Abitur
1986 - 1988	Ausbildung zum Datenverarbeitungskaufmann bei Mannesmann-Kienzle, Villingen-Schwenningen
1988 - 1990	Zivildienst im Rettungsdienst, Deutsches Rotes Kreuz, Kreisverband Rhein-Hunsrück
1990 - 1996	Studium der Informatik an der Friedrich-Alexander-Universität Erlangen-Nürnberg
1996	Diplomarbeit an der State University of New York at Stony Brook, NY
1997	Wissenschaftliche Hilfskraft am Lehrstuhl für Graphische Datenverarbeitung, Friedrich-Alexander-Universität Erlangen-Nürnberg
seit 1997	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Graphisch-Interaktive Systeme am Wilhelm-Schickard-Institut für Informatik der Eberhard-Karls-Universität Tübingen (Prof. Straßer)