Technische Universität Wien
Vienna University of Technology

DISSERTATION

Real-Time Mono- and Multi-Volume Rendering of Large Medical Datasets on Standard PC Hardware

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

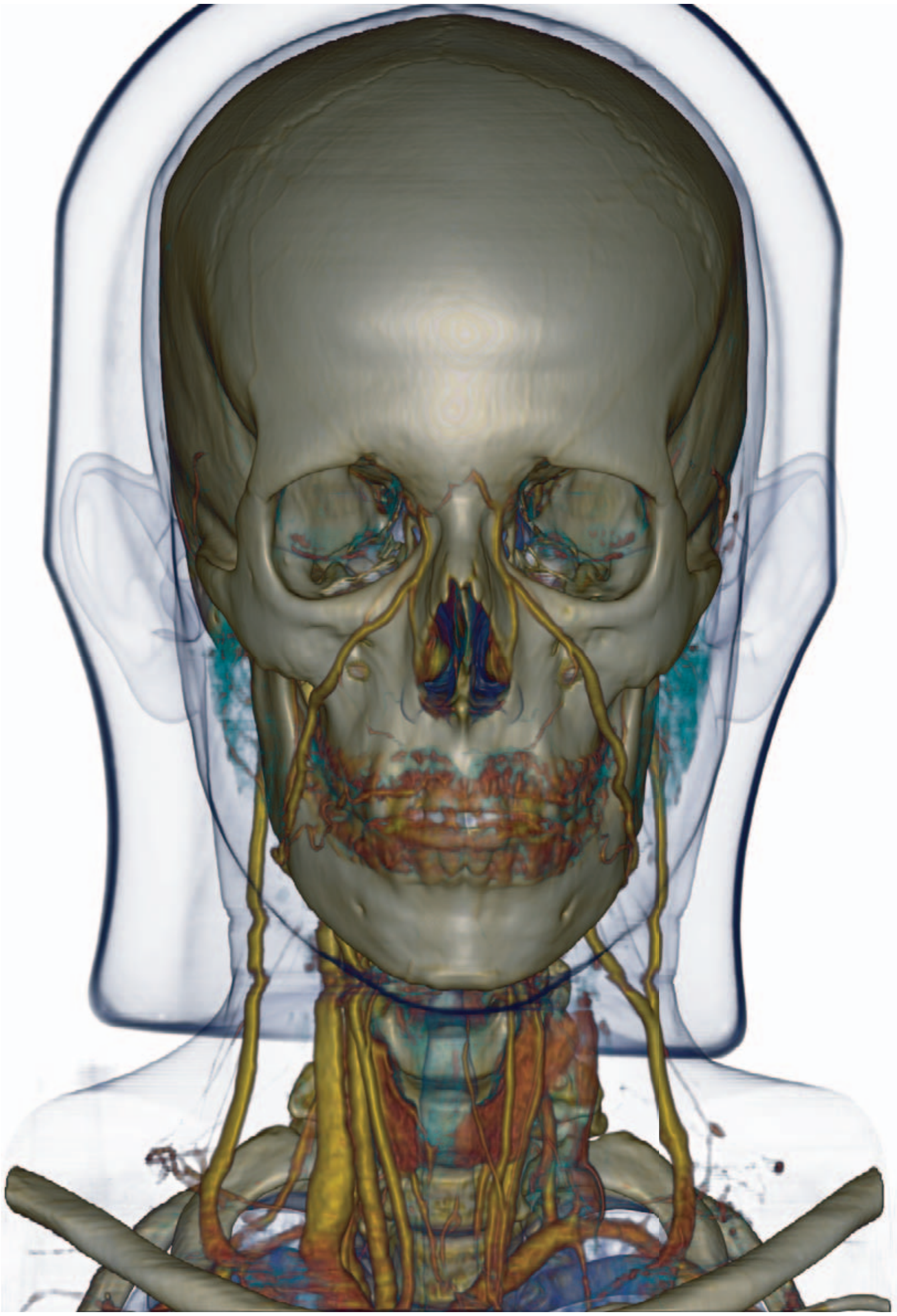
eingereicht an der Technischen Universität Wien, Fakultät für Informatik,
durch

Dipl. Inform. Sören Grimm
Matrikelnummer: 0427227
Gaullachergasse 33/35
1160 Vienna, Austria
geboren am 16.02.1973 in Stuttgart, Deutschland

Wien, im Februar 2005

Der Gegenstand der wissenschaftlichen Forschung soll die Entwicklung und Adaption von Methoden der Computergraphik und insbesonders der 3D Volumenvisualisierung sein. Ziel des Projektes ist, ein Softwaresystem für die tägliche medizinische Praxis in radiologischen Fachbereichen zu entwickeln, welches Schwerpunkte der computerunterstützten Diagnose (Computer Aided Diagnosis) – Diagnose, Analyse und Planung – umfassen soll.

Zielsetzung des Forschungsprojektes ADAPT.



Abstract

Direct Volume Visualization is an efficient technique to explore complex structures within volumetric data. Its main advantage, compared to standard 3D surface rendering, is the ability to perform semitransparent rendering in order to provide more information about spatial relationships of different structures. Semitransparent rendering requires to process a huge amount of data. The size of volumetric data is rapidly increasing, on the one hand due to the boost of processing power in the past years, and on the other hand due to improved capabilities of newer acquisition devices. This large data presents a challenge to current rendering architectures and techniques. The enormous data sizes introduce a growing demand for interactive 3D visualization. Conventional slicing methods already reach their limit of usability due to the enormous amount of slices. 3D visualization is more and more explored as an attractive alternative additional method for examinations of large medical data to support the necessary 2D examination.

Within this dissertation a set of approaches to handle and render large volumetric data is developed, enabling significant performance improvements due to a much better utilization of the CPUs processing power and available memory bandwidth. At first, highly efficient approaches for addressing and processing of a cache efficient memory layout for volumetric data are presented. These approaches serve as a base for a full-blown high-quality raycasting system, capable of handling large data up to 3GB, a limitation imposed by the virtual address space of current consumer operating systems. The core acceleration techniques of this system are a refined caching scheme for gradient estimation in conjunction with a hybrid skipping and removal of transparent regions to reduce the amount of data to be processed. This system is extended so that efficient processing of multiple large data sets is possible. An acceleration technique for direct volume rendering of scenes, composed of multiple volumetric objects, is developed; it is based on the distinction between regions of intersection, which need costly multi-volume processing, and regions containing only one volumetric object, which can be efficiently processed. Furthermore, V-Objects, a concept of modeling scenes consisting of multiple volumetric objects, are presented. It is demonstrated that the concept of V-Objects in combination with direct volume rendering, is a promising technique for visualizing medical data and can provide advanced means to explore and investigate data.

In the second part of the dissertation, an alternative to grid-based volume graphics is presented: Vots, a point-based representation of volumetric data. It is a novel primitive for volumetric data modeling, processing, and rendering. A new paradigm is presented by moving the data representation

from a discrete representation to an implicit one.

Kurzfassung

Direkte Volumenvisualisierung ist eine effiziente Methode um komplexe Strukturen volumetrischer Datensätze zu untersuchen. Der Hauptvorteil, verglichen zur normalen Oberflächenvisualisierung, ist die Möglichkeit halbtransparente Visualisierungen zu generieren, dadurch erhält man mehr Informationen über die räumlichen Zusammenhänge verschiedener Strukturen. Um solch halb transparenten Visualisierungen zu erzeugen, muss eine enorme Datenmenge abgearbeitet werden. Durch immer leistungsfähigere Prozessoren und durch verbesserte Aufnahmegeräte werden diese Datenmengen immer größer. Diese enormen Datenmengen stellen eine große Herausforderung für derzeitige 3D Rendering Architekturen und Algorithmen dar. Die immer größeren Datenmengen erhöhen die Nachfrage an 3D Visualisierung. Die herkömmliche 2D Visualisierung erreicht, durch die enorme Anzahl von Schichtbildern, bereits die Grenze der Benutzbarkeit. Die 3D Visualisierung wird immer mehr als eine alternative unterstützende Methode von Schichtbild-Untersuchungen großer medizinischer Datensätze eingesetzt.

In dieser Dissertation werden effiziente Verfahren zur Handhabung und dem Rendern großer volumetrischer Daten vorgestellt, welche zu einer signifikanten Geschwindigkeitssteigerung, durch bessere Ausnützung von Prozessor- und Speicher-bandbreite, führen. Zuerst werden Verfahren zum Adressieren und Verarbeiten eines Cache-effizienten Speicher-Layouts für große volumetrische Daten vorgestellt. Diese Verfahren dienen als Basis für ein komplettes hochqualitatives Raycasting-System, welches in der Lage ist große Daten bis zu 3 GB, eine Limitierung des virtuellen Adressbereichs heutiger Betriebssysteme, zu handhaben. Die Hauptbeschleunigungs-Komponenten dieses System sind eine raffinierte Caching Methode für die Gradientenberechnung in Verbindung mit einer hybriden Technik zum Überspringen und Entfernen transparenter Regionen, wodurch die Menge der zu verarbeitenden Daten signifikant reduziert wird. Dieses System wird dann so erweitert, dass effizientes Verarbeiten mehrerer volumetrischer Datensätze möglich ist. Ein Beschleunigungsverfahren zum Rendern von Szenen, die aus mehreren volumetrischen Datensätzen bestehen, wird vorgestellt. Die Grundidee des Verfahrens basiert auf der Unterscheidung zwischen Regionen, in denen sich mehrere Objekte überschneiden, die eine teure Verarbeitung erfordern, und Regionen in denen sich nur ein Objekt befindet und somit eine effizienteres Verarbeiten erlauben. Weiterhin werden V-Objects, ein Konzept zur Modellierung von Szenen, welche aus mehreren volumetrischen Objekten bestehen, vorgestellt. Es wird gezeigt, dass das Konzept der V-Objects zusammen mit direkter Volumenvisualisierung eine viel versprechende Methode zur Visualisierung medizinischer Daten ist und das es eine weiterte Möglichkeit zum Untersuchen und

Erkunden der Daten bietet.

Im zweiten Teil der Dissertation wird eine Alternative zur gitter-basierten Volumengraphik vorgestellt: Vots, eine punkt-basierte Representation volumetrischer Daten. Es ist ein neues Primitive zur Modellierung, Verarbeitung und Rendern von volumetrischen Daten. Ein neues Paradigma wird präsentiert durch die Umwandlung der Daten von einer diskreten zu einer impliziten Darstellung.

Acknowledgements

The work described in this dissertation would not have been possible without the support and encouragement of many people. First of all, I would like to thank my advisor **Prof. Eduard Gröller** for the tremendous degree of freedom I received to work on different topics and the support to cooperate with Tiani MedGraph. His patient supervision, fruitful discussions, and his immense knowledge provided a very important foundation for this work. I also would like to thank **Miloš Šrámek** who agreed to be part of my graduation committee, and for his support and advise.

I am highly indebted to my main collaborator and former student **Stefan Bruckner**. His immense eagerness to work and his great competence significantly helped to make this thesis and the corresponding publications happen.

Furthermore I want to thank all members of the Visualization Group. Especially I want to thank the members of the ADAPT team **Armin Kanitsar**, **Ivan Viola**, **Rainer Wegenkittl**, and **Michi Knapp** for the excellent team-work.

Last but not least, I would like to express my gratitude for all kinds of non measurable support and experiences in life to my family, **Horst Grimm**, **Ines Tibi**, **Hanna Moser**, and to **Adriana Paluszny** for her patience and moral support through all these years.

This work has been funded by the AD**A**PT project FFF-804544. AD**A**PT is supported by *Tiani Medgraph*, Vienna (http://www.tiani.com), and the *Forschungsförderungsfonds für die gewerbliche Wirtschaft*, Austria. See http://www.cg.tuwien.ac.at/research/vis/adapt for further information on this project. The used data sets, if not stated otherwise, are courtesy of Univ.-Klinik Innsbruck, AKH Vienna, Univ.-Klinikum Freiburg, Tiani MedGraph AG, and NLM.

Contents

List of Figures

Chapter 1

Introduction

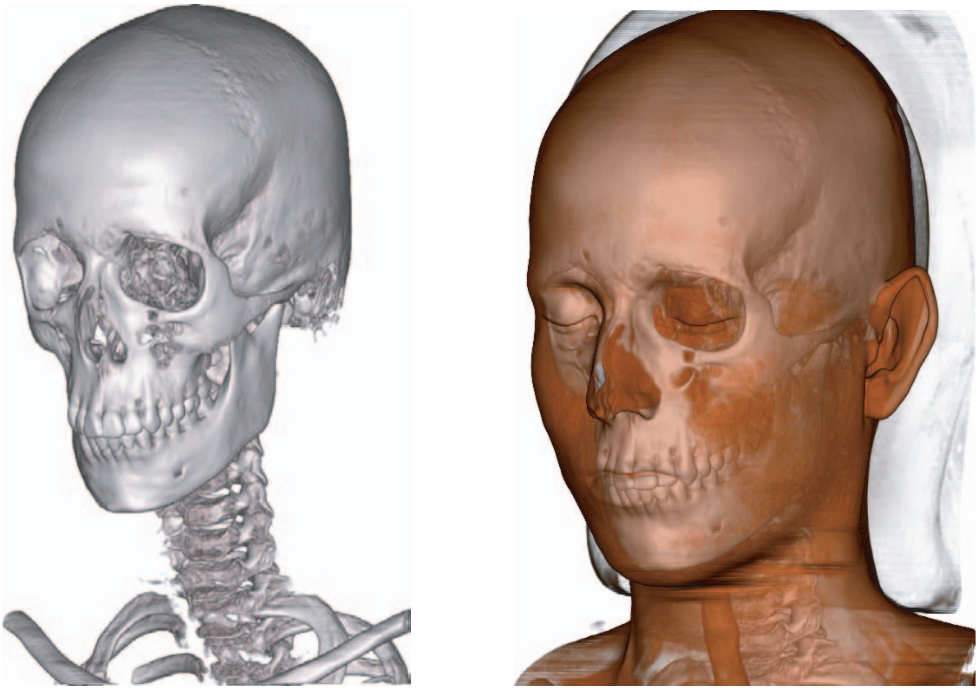


Figure 1.1: Volume Visualization.

Just as 2D raster graphics superseded vector graphics, volume graphics has the potential to supersede surface graphics for 3D geometric scene representation, manipulation, and rendering.

Arie E. Kaufman [24]

1.1 Volume Visualization

Volume visualization is a method used to extract meaningful information from volumetric data sets. A typical medical imaging example is shown in Figure 1.1. A *volumetric data set* is the 3D conceptual counterpart of a 2D *image*. It consists of *volume elements*, called voxels, which together represent an entire volume, analogous to *picture elements*, or pixels, which together represent a 2D *image*, see Figure 1.2.

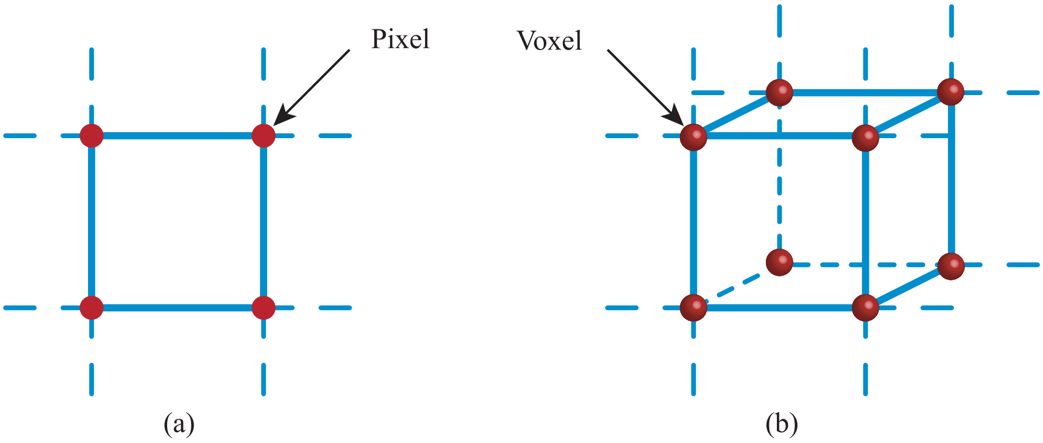


Figure 1.2: (a) Image cell composed of four pixels. (b) Volume cell composed of eight voxels.

Each *voxel* is a quantum unit of a *volumetric data set* and has one or more numeric values associated with it. The sources of volume data are sampled objects, measured phenomena, or the result of simulations. Volume data occurs in many areas, typical examples are medical imaging (computed tomography, magnetic resonance imaging, and ultra sound), biology, geoscience, and meteorology.

1.1.1 Grid Types

The voxels of a volumetric data set are usually arranged on a grid structure to allow efficient spatial addressing of the data. The different grid-structures can be classified into three main grid-types: rectilinear grids, curvilinear grids, and unstructured grids. Rectilinear grids consist of convex cells with implicit neighborhood connectivity, see Figure 1.3a. Curvilinear grids can be obtained by applying a non-linear transformation on a rectilinear grid, see Figure 1.3b. Unstructured grids do not have a regular topology; therefore, the implicit neighborhood connectivity is lost, see Figure 1.3c. Each of these different grid topologies requires different rendering algorithms for visualization. Within the scope of this dissertation, the focus is on rectilinear grids.

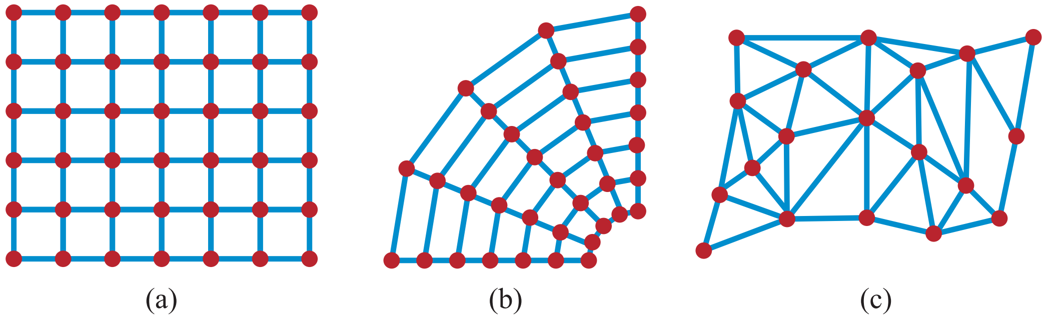


Figure 1.3: Different grid types: (a) Rectilinear grid. (b) Curvilinear grid. (c) Unstructured grid.

1.1.2 Classification and Segmentation

Classification is used to define different features within the data. The classification information is represented by a transfer function: for each possible voxel value, possibly combined with other input parameters, an opacity, color and other properties are specified as output. Opacity allows the user to explore the inside of an object by hiding and showing certain features within the data, see Figure 1.4a. Additional different material properties can be assigned which influence shading and lighting. In practice it is rather difficult and time-demanding to specify the appropriate transfer function. Usually, histograms are used to simplify the transfer function set up, because they provide a better understanding of the distribution of voxel values within the data set. Furthermore, in order to highlight feature boundaries, opacity can be modulated according to the gradient magnitude [35].

Classification is a very powerful tool to explore the data, however, due to the scanning process, different types of material are often mapped to the same voxel value. In this case, traditional classification fails and the data must be segmented. Each voxel is assigned a certain label. Depending on the acquisition method and the scanned object this can either be done manually, semiautomatically, or fully automatically. This label information is then later used during the visualization process in order to identify the different objects. In Figure 1.4b, for example, the brain and the major vessels have been segmented.

Segmentation is an effective technique to simplify the transfer function specification. However, it does not solve the problem of simultaneously visualizing multiple occluding objects. One way to enable such a visualization is achieved by the use of clipping, as shown in Figure 1.4c. In this visualization the most common and straightforward clipping technique based on axis aligned cutting planes is used. There are more advanced clipping techniques which take into account properties, such as the distance between the eye and the object, or segmentation information [75, 3, 72].

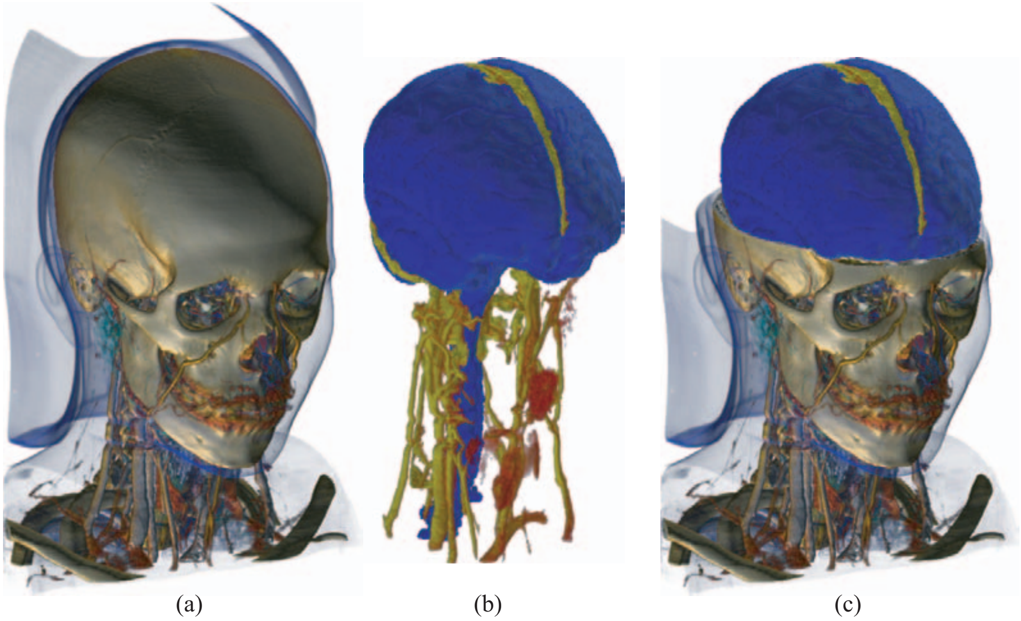


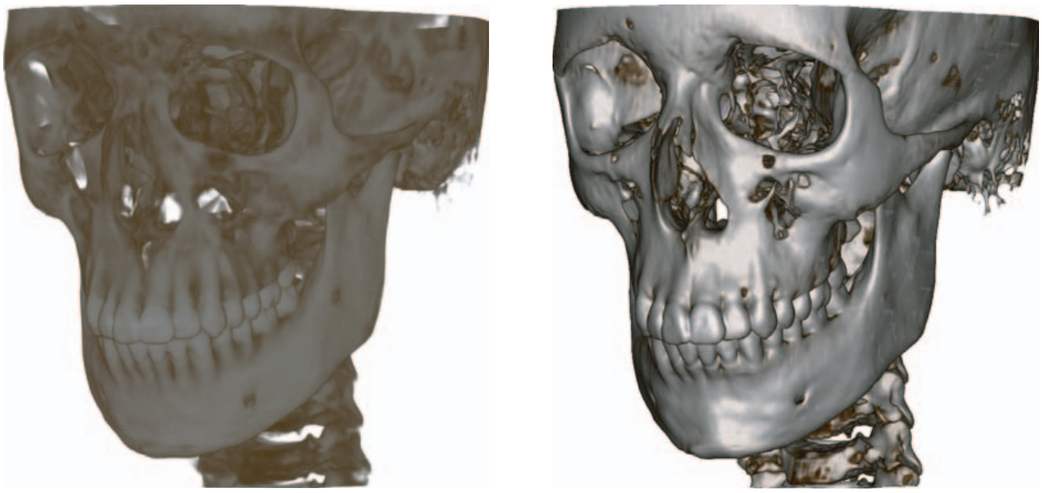
Figure 1.4: CT scan of a human head (512x512x333, 16 bit). (a) Classification. (b) Segmentation. (c) Clipping planes.

1.1.3 Illumination and Shading

Illumination and shading within volume rendering refers to the same illumination models and shading techniques used in polygon rendering. The goal is to enhance the appearance of rendered objects, especially to emphasize their shape and structure, by simulating the effects of light interacting with the object. A comparison between shaded and non shaded volume rendering is shown in Figure 1.5. Two different illumination models are distinguished: global illumination and local illumination.

The contribution from the light that goes directly from the light source and is reflected from the objects is described by a *local illumination model.* In this model the shading of any object is independent from the shading of all other objects.

A *global illumination model* adds to the local model the light that is reflected from other objects to the current object. It is more comprehensive, more physically correct, and produces images that appear more realistic. This model is also computationally more expensive.



(a) (b)

Figure 1.5: Volume rendering: (a) Without shading. (b) With shading. CT scan of a human head (512x512x333, 16 bit).

For volume rendering usually a local illumination model is used, due to its lower computational costs. Most of the time the Phong illumination model is applied [60]. Although it lacks physical validity, it is commonly used, as it can be efficiently computed. The Phong model addresses point and directional light sources only and models three types of reflected light, namely:

Ambient Reflection models the reflection of light which arrives at the surface from all directions, after having bounced around the scene in multiple reflections from all the surfaces of the scene. Basically, it represents the natural light of a scene. In the Phong model, ambient light is assumed to have a constant intensity throughout the scene. Each surface, depending on its physical properties, has a coefficient of ambient reflection that measures what fraction of the light is reflected from the surface. For an individual surface the intensity of ambient light reflected is:

$$I_{amb} = I_a \cdot \kappa_{amb}$$

where I_a is the constant intensity of the ambient light and κ_{amb} is the coefficient of the ambient reflection of the surface.

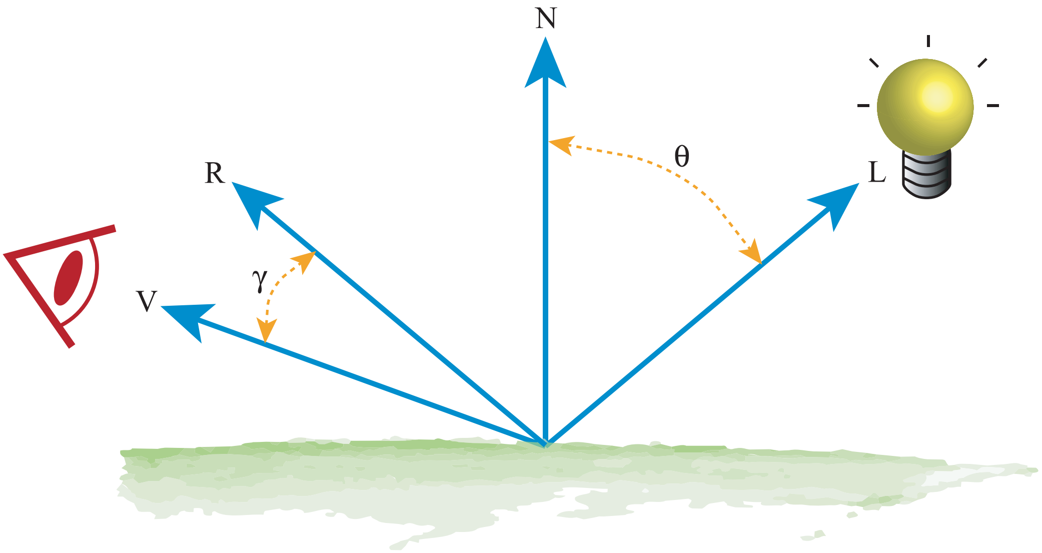


Figure 1.6: Parameters of Phong's illumination model.

Diffuse reflection models the reflection from non shiny surfaces from which light is equally scattered in all directions. Thus, the intensity at a point on a surface, as perceived by the viewer, does not depend on the position of the viewer. For an individual surface the intensity of diffusely reflected light is:

$$I_{diff} = \begin{cases} I_i \cdot \kappa_{diff} \cdot \cos(\theta), & |\theta| < 90^\circ \\ 0 & otherwise \end{cases}$$

where I_i is the intensity of the incident light, κ_{diff} is the coefficient of diffuse reflection for the material, and θ is the angle between the surface normal and the light vector L, see Figure 1.6.

Specular Reflection models the reflection of light from mirror-like surfaces. Specular reflection is caused by the mirror-like property of a surface. A perfect mirror will reflect light arriving at the surface at an angle of incidence θ to the normal at a reflected angle of θ to the normal, in the same plane as the normal and the incident light. This means that only a viewer on the reflected ray will actually see the reflected light. In practice, no surface is a perfect mirror and there will be a certain amount of light scattered around the reflected direction. The reflected light is, therefore, seen as a highlight over an area of the surface. For an individual surface the intensity of specularly reflected light is:

$$I_{spec} = \begin{cases} I_i \cdot \kappa_{spec} \cdot \cos^m(\gamma), & |\gamma| < 90^\circ \\ 0 & otherwise \end{cases}$$

where I_i is the intensity of the incident light, κ_{spec} is the coefficient of specular reflection for the material, γ is the angle between the reflection vector R and the viewing vector V, and m controls the extension of the highlight, see Figure 1.6.

The complete basic reflection model is given as:

$$I = I_{amb} + I_{diff} + I_{spec}$$

1.1.4 Reconstruction

Many volume rendering algorithms resample the volumetric data by using rays, planes, or random sample points. As these resample points most likely do not match the existing voxel locations, the volume has to be reconstructed at these new positions. Thus, new voxel values have to be interpolated from the existing ones. There are numerous different interpolation methods, which differ in quality and computational costs. The most popular methods for volume rendering are:

Nearest neighbor interpolation is the simplest and fastest method; it also is the most inaccurate one. Given a resample position, the closest of all neighboring voxel values is assigned which results more in a selection than filtering. This approach leads to severe artifacts, such as staircase effects.

Trilinear interpolation is obtained by the application of a linear interpolation in each dimension, it assumes a linear relation between neighboring voxels. In 1D there a two, in 2D there are four, and in 3D there are eight interpolation points. It is superior to the nearest neighbor interpolation. When using large magnification factors, three dimensional crosses (diamonds) appear due to the nature of the trilinear kernel.

Cubic interpolation provides a much better quality, however, the computational costs are considerably high. In contrast to linear interpolation, cubic interpolation takes four interpolation points into account, along a single dimension. Thus, two more multiplications are necessary, compared to linear interpolation. This leads in 3D to a considerable amount of instructions. Cubic filtering allows a more extended flexibility with respect to reconstruction. The influence on the final result of each interpolation point is controlled by parameters. Thus, smoothing or sharpening can be achieved.

Higher order interpolation methods, such as cubic convolution, are superior to trilinear interpolation regarding the achievable image quality. However, trilinear interpolation provides sufficient quality for most applications and, therefore, provides a good tradeoff between quality and computational costs.

1.1.5 Gradient Estimation

In order to achieve a realistic display of 3D volumetric objects, shading and illumination are essential, see Section 1.1.3. Shading requires a gradient, i.e., a surface normal vector, to compute the diffuse and specular components. Volumetric data is usually obtained by sampling continuous objects, after the sampling the surface normals are not available anymore. Therefore, the normal of the surface is estimated by investigating the close neighborhood of a given voxel. The quality of the generated image is strongly influenced by the estimation method used for the normal vector computation. The most popular gradient estimation methods for volume rendering are:

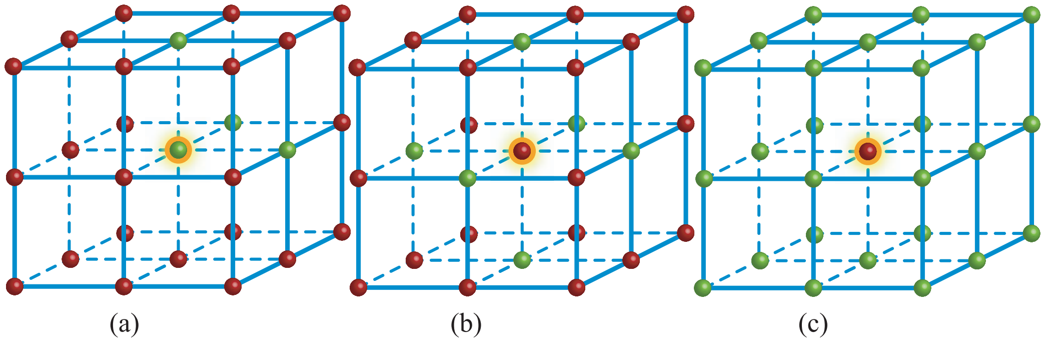


Figure 1.7: Input patterns, green spheres, of different gradient estimation schemes: (a) Intermediate difference gradient. (b) Central difference gradient. (c) Neumann gradient.

Intermediate difference gradient takes as input four neighboring voxels, see Figure 1.7a. For a given voxel V at the position (x, y, z) the gradient ∇ is:

$$\begin{aligned}
\nabla_x &= V_{x+1,y,z} - V_{x,y,z} \\
\nabla_y &= V_{x,y+1,z} - V_{x,y,z} \\
\nabla_z &= V_{x,y,z+1} - V_{x,y,z}
\end{aligned}$$

Central difference gradient takes as input six neighboring voxels, see Figure 1.7b. For a given voxel V at the position (x, y, z) the gradient ∇ is:

$$\begin{aligned}\nabla_x &= V_{x+1,y,z} - V_{x-1,y,z}\\ \nabla_y &= V_{x,y+1,z} - V_{x,y-1,z}\\ \nabla_z &= V_{x,y,z+1} - V_{x,y,z-1}\end{aligned}$$

Neumann gradient takes as input 26 neighboring voxels, see Figure 1.7c. In general, this gradient estimation approach is a theoretical framework based on linear regression; it is the generalization of many previous approaches. In order to achieve highest gradient quality, Neumann et al. [54] recommend, for a given voxel V at the position (x, y, z), to use the following 3x3x3 filter masks for gradient ∇:

$$\nabla_x := \tfrac{1}{52}\left(\begin{pmatrix} -2 & 0 & 2\\ -3 & 0 & 3\\ -2 & 0 & 2\end{pmatrix}\begin{pmatrix} -3 & 0 & 3\\ -6 & 0 & 6\\ -3 & 0 & 3\end{pmatrix}\begin{pmatrix} -2 & 0 & 2\\ -3 & 0 & 3\\ -2 & 0 & 2\end{pmatrix}\right)$$

$$\nabla_y := \tfrac{1}{52}\left(\begin{pmatrix} 2 & 3 & 2\\ 0 & 0 & 0\\ -2 & -3 & -2\end{pmatrix}\begin{pmatrix} 3 & 6 & 3\\ 0 & 0 & 0\\ -3 & -6 & -3\end{pmatrix}\begin{pmatrix} 2 & 3 & 2\\ 0 & 0 & 0\\ -2 & -3 & -2\end{pmatrix}\right)$$

$$\nabla_z := \tfrac{1}{52}\left(\begin{pmatrix} -2 & -3 & -2\\ -3 & -6 & -3\\ -2 & -3 & -2\end{pmatrix}\begin{pmatrix} 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix} 2 & 3 & 2\\ 3 & 6 & 3\\ 2 & 3 & 2\end{pmatrix}\right)$$

This method of gradient estimation also allows to filter the original voxels values with low additional computational costs, as intermediate results of the gradient estimation can be reused. The filter mask for the voxel value is:

$$V := \frac{1}{52}\left(\begin{pmatrix} 2 & 3 & 2\\ 3 & 6 & 3\\ 2 & 3 & 2\end{pmatrix}\begin{pmatrix} 3 & 6 & 3\\ 6 & 0 & 6\\ 3 & 6 & 3\end{pmatrix}\begin{pmatrix} 2 & 3 & 2\\ 3 & 6 & 3\\ 2 & 3 & 2\end{pmatrix}\right)$$

In Figure 1.8 the different gradient estimation methods are compared.

The advantage of the intermediate difference operator is that it detects high frequency detail, which can be lost when using the central difference operator. This also leads to less appealing images when rendering data sets with a lot of noise.

The central difference operator is a good low-pass filter, nevertheless, very narrow structures are sometimes missed due to the central difference. Similar

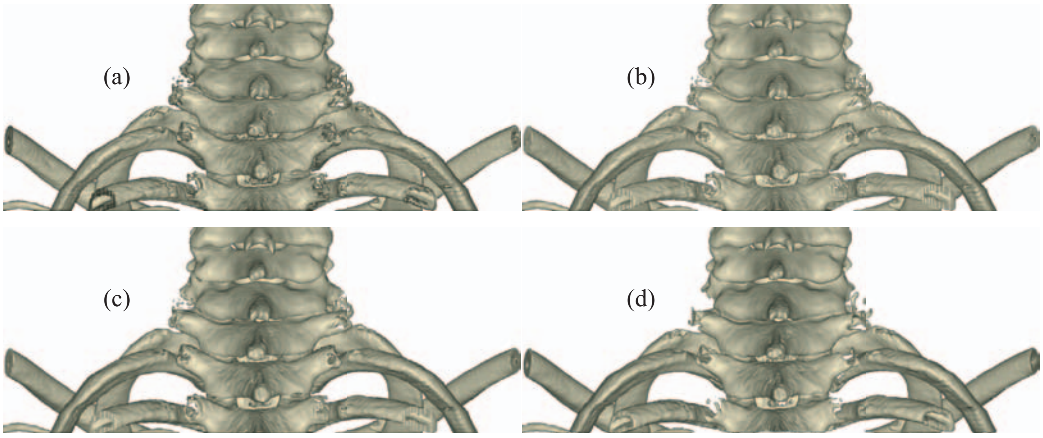


Figure 1.8: Different gradient estimation methods: (a) Intermediate difference gradient estimation. (b) Central difference gradient estimation. (c) Neumann gradient estimation. (d) Neumann gradient estimation and voxel filtering.

to the intermediate difference gradient, the central difference gradient is not isotropic; this may become troublesome when using the gradient magnitude as further input parameter to assign opacities.

The Neumann gradient estimation method, as well as other higher order gradient estimation approaches, produce nearly isotropic gradients and less fringing artifacts than other methods. Although this gradient estimation method has considerably high computational costs, it is used later on in the presented high-quality rendering system due to its superior quality. This quality can be supported by additionally using the filtered density value, however, this causes fading of small details. While this smoothness effect is beneficial to the visual appearance of the image, this approach typically cannot be used in real medical applications.

More detailed information about different gradient estimation methods can be found in [84, 47, 70].

1.1.6 Compositing

Compositing refers to the process where all contributions to a pixel are combined into one final pixel value. It can be expressed as an approximation of the well-known Low-Albedo volume rendering integral [2, 29, 41]:

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

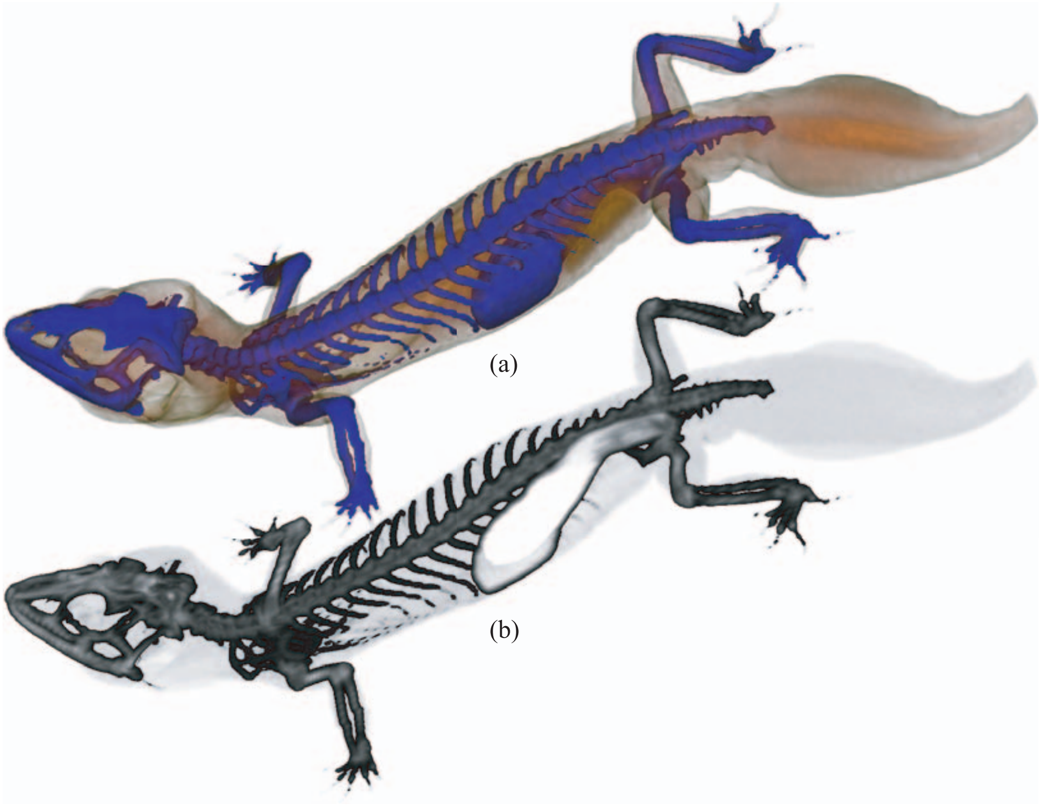


Figure 1.9: (a) Standard compositing. (b) Maximum Intensity Projection. CT scan of a gecko (512x512x88, 16 bit, courtesy of University of Veterinary Medicine Vienna).

L is the length of ray r. Considering the volume to be composed of particles with certain densities μ, then these particles receive light from all surrounding light sources and reflect this light towards the eye point according to their specular and diffuse material properties. Additionally, the particles may also emit light by themselves. Thus, in Equation 1.1, C_λ is the light of wavelength λ reflected and/or emitted at the location s in the direction r. In order to take into account the higher reflectance of particles with larger densities, the reflected color is weighted by the particle density. The light scattered at s is then attenuated by the density of the particles between s and the eye according to the exponential attenuation function.

In practice, it is impossible to evaluate this integral analytically. Therefore, the integral is approximated by:

$$\mathrm{I}_\lambda(x,r) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1}(1-\alpha(s_j)) \tag{1.2}$$

Here, $\alpha(s_i)$ represent the opacity samples along a ray and $C_\lambda(s_i)$ are the local color values derived from the illumination model. C and α are referred to as transfer functions. These functions assign color and opacity to each intensity value in the volume.

There are also other popular compositing operators, such as Maximum Intensity Projection (MIP) and X-Ray Projection. For MIP the final pixel value is determined by the maximum along a ray:

$$I(x, r) = max(\mu(s)), s \in [0, L] \tag{1.3}$$

L is the length of a ray and μ are the densities. For X-Ray projection the final pixel value is determined by the sum of intensity values along a ray:

$$I(x, r) = \sum_{i=0}^{L/\Delta s} \mu(s_i) \tag{1.4}$$

L is the length of a ray and μ is the density.

In practice, Equation 1.2 and Equation 1.3 are the most often used ones, examples are shown in Figure 1.9.

1.1.7 Indirect Volume Rendering

Using Indirect Volume Rendering (IVR), the volume data is converted into an intermediate *traditional* computer graphics representation. Usually an iso-surface or an iso-contour is extracted and the resulting graphics primitives (polygons) are then rendered using standard computer graphics hardware. The most popular approach of IVR is probably the Marching Cubes algorithm [38], where each volume cell of eight neighboring voxels is classified according to a specified iso value. Each voxel has two states; it is either inside or outside of an iso surface. Thus, there are $2^8 = 256$ ways a surface can intersect the cube. In the original paper on Marching Cubes, these 256 cases were reduced to 15 cases by inverting or rotating the classification cubes, see Figure 1.10. However, later research showed that this reduced case table generates inconsistencies, which result in holes in the iso surface. Therefore, the full 256 case table is usually used.

1.1.8 Direct Volume Rendering

In contrast to IVR, methods of Direct Volume Rendering (DVR) generate images without the necessity of creating an intermediate polygonal representation. Instead, the volume data set is projected onto an image plane.

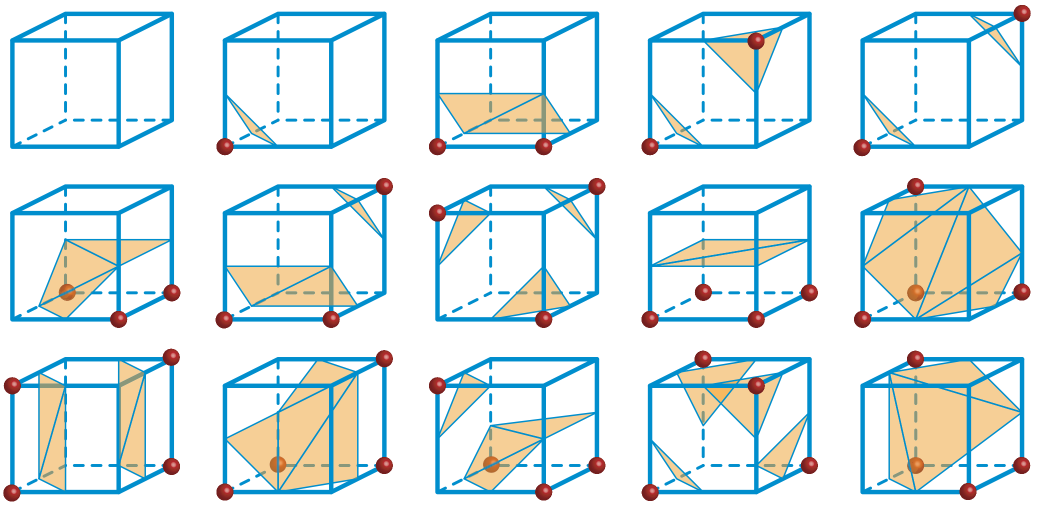


Figure 1.10: Marching Cubes.

Raycasting

In the image-space oriented ray casting approaches, rays are cast from the view-point through the view-plane into the volume, see Figure 1.11. The volume is equidistantly sampled along the ray and the integral of Equation 1.2 is computed by repeated application of the *over* operator [61] in front-to-back order. That is, at each resample location, the current color and alpha values for a ray are computed in the following way:

$$\begin{aligned} c_{out} &= c_{in} + c(x)\alpha(x)(1-\alpha_{in}) \\ \alpha_{out} &= \alpha_{in} + \alpha(x)(1-\alpha_{in}) \end{aligned} \tag{1.5}$$

c_{in} and α_{in} are the color and opacity the ray has accumulated so far. x is the reconstructed function value, and $c(x)$ and $\alpha(x)$ are the classified and shaded color and opacity for this value. Basically, at every resample position a sample is interpolated out of the corresponding surrounding eight voxels, see Figure 1.2b. This sample is then classified according to a transfer function. If the sample has a contribution to the ray, a gradient is also computed from the surrounding voxels, in order to apply shading. Finally, the sample is composited with the previous samples of the ray. Most acceleration methods avoid unnecessary computations, such as early ray termination, where sampling along the ray is terminated once full opacity has been reached, or space leaping where fully transparent areas of the volume are left out based on a given distance field. Raycasting has seen the largest body of publications over the years; this is mainly because raycasting provides the highest quality and its computation scheme allows to exploit parallel computing as well as memory efficient data-structures [33, 57, 73, 26].

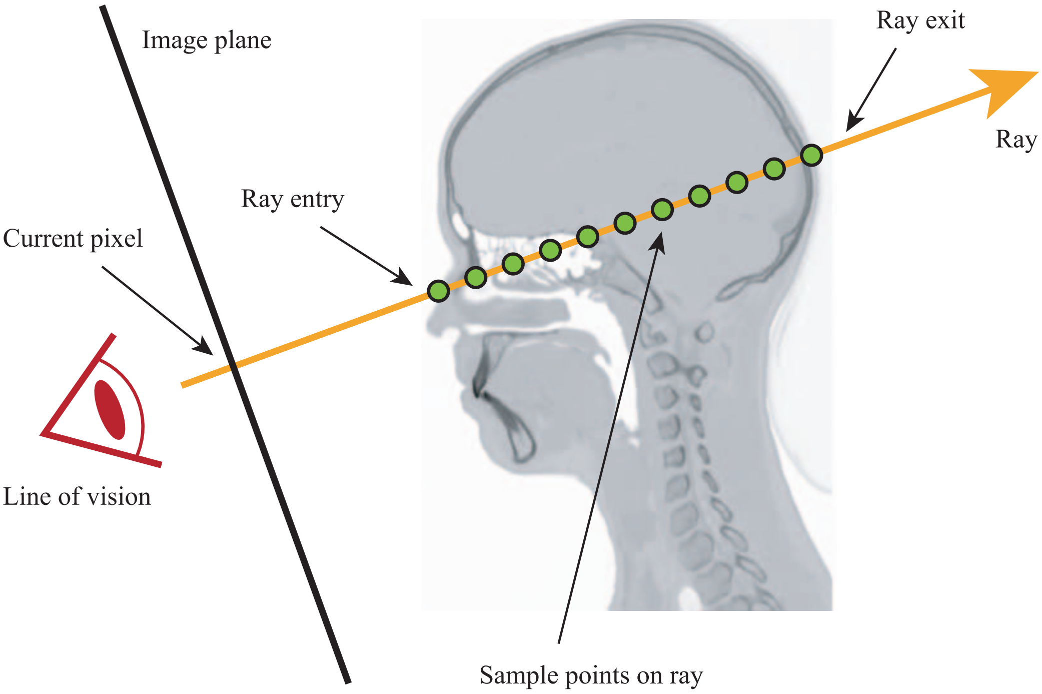


Figure 1.11: Raycasting.

Splatting

Splatting, first proposed by Westover [78, 79], is as opposed to raycasting an object-order approach. Each voxel is projected onto the image plane as an overlapping, orientation invariant Gaussian kernel with an amplitude scaled according to the voxel value, see Figure 1.12a. The major advantage of splatting is that voxels that, due to fully transparency, do not contribute to the final image can be immediately left out of processing. This enormously reduces the amount of data to be processed. In order to obtain correct compositing, the volume has to be processed in the correct visible order. The most common method is to use a sheet buffer [50], see Figure 1.12b. The voxel kernels are processed within slabs that are aligned parallel to the image plane. All voxel kernels that overlap a slab are summed into a sheet buffer which is then composited with the previous sheet. This method significantly reduces the severe popping artifacts that can occur during splatting. The splatting method that provides the best quality is currently Elliptical

Weighted Average (EWA) splatting [86]. Their footprint function combines an elliptical Gaussian reconstruction kernel with a Gaussian low-pass filter, thereby the image quality is significantly improved.

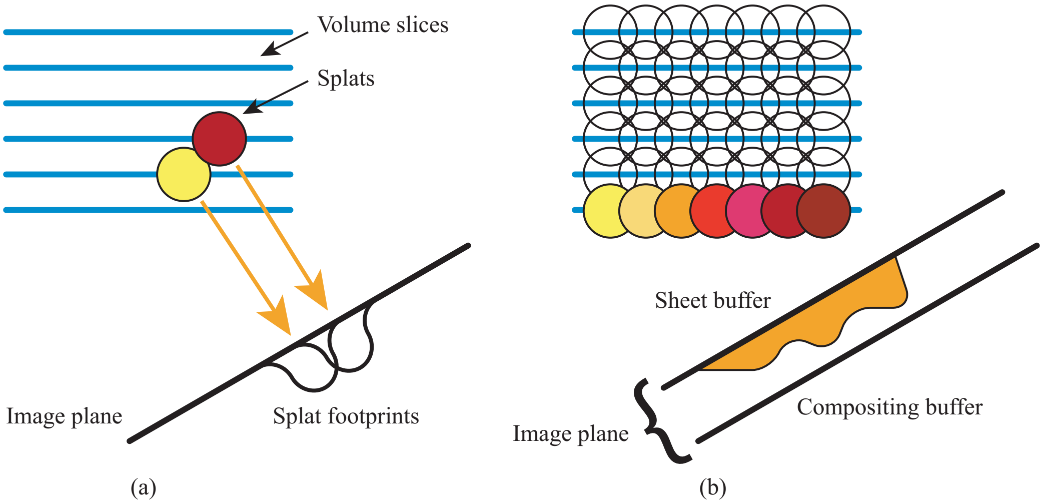


Figure 1.12: Splatting: (a) Splat footprints. (b) Sheet buffer compositing of splats.

Shear warp

The shear-warp was first proposed in a orthographic projection version by Cameron and Undrill [6] and then extended and improved by Lacroute and Levoy [32]. It is still considered to be one of the fastest software volume renderers. The algorithm is based on a shear-warp factorization of the viewing transformation. The volume data is transformed by a shear into the so called *sheared object space*, see Figure 1.13. In the *sheared object space* all viewing rays are parallel to the major axis which is most perpendicular to the viewing direction. Here, the sheared volume slices are composited together in front-to-back order using the *over* operator, see Section 1.1.8. This step projects the volume into a 2D intermediate image in *sheared object space*. Finally, the intermediate image is transformed to an image space by using a warping operation.

This slice-wise processing scheme allows memory efficient processing as it allows straightforward prefetching of data from main memory into the cache. To accelerate the processing, screen space RLE-encoding can be exploited; this encoding is always updated whenever a pixel does not alter its value any further, for instance, if full opacity has been reached or the corresponding ray leaves the volume. Furthermore, due to its inherent parallelism, the performance of the algorithm is significantly accelerated by mapping onto a SIMD architecture [43, 6].

The encoding scheme is axis aligned, therefore, it requires the construction of a separate encoded volume for each axis. Furthermore, the resampling takes place within the volume slices using bilinear interpolation, which leads to a significant lower image quality compared to raycasting.

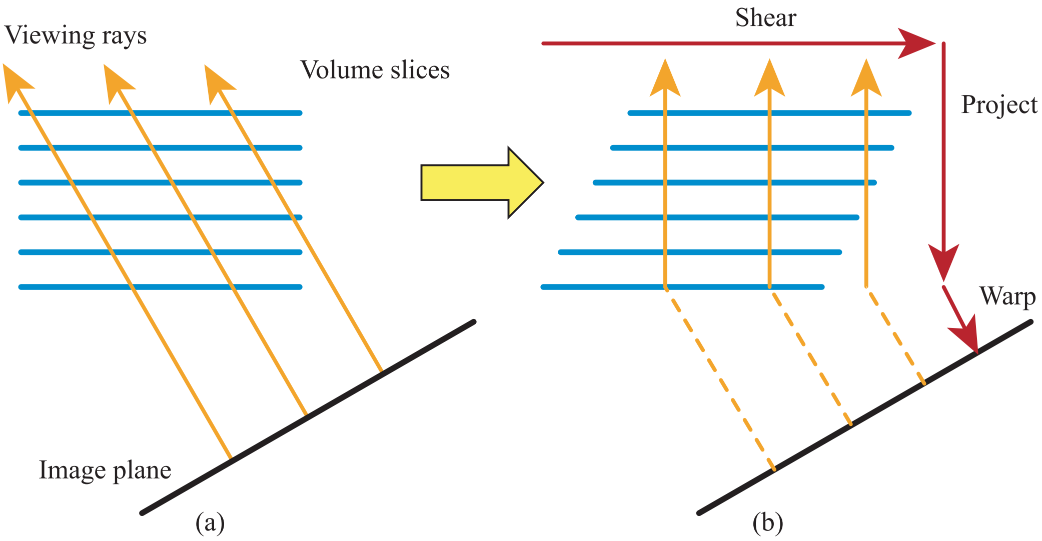


Figure 1.13: Shear Warp: (a) Object space. (b) Sheared object space.

Graphics Hardware

Opposed to the pure software-based approaches, hardware-accelerated techniques try to offload the computational expensive processing of the volume data onto the graphics hardware. The data is, therefore, packed into a texture. Two types of textures are distinguished:

2D Textures: This approach stores stacks of slices for each major viewing axis as two-dimensional textures in memory [4, 64]. Similar to the shear-warp method described in Section 1.1.8, the texture stack that is most perpendicular to the viewing direction is chosen, see Figure 1.14. For rendering each texture is mapped onto an appropriate geometry. These geometries are rendered in back to front order using alpha blending. Although this approach provides high performance it is of limited practicability, as the three texture stacks consume a considerably high

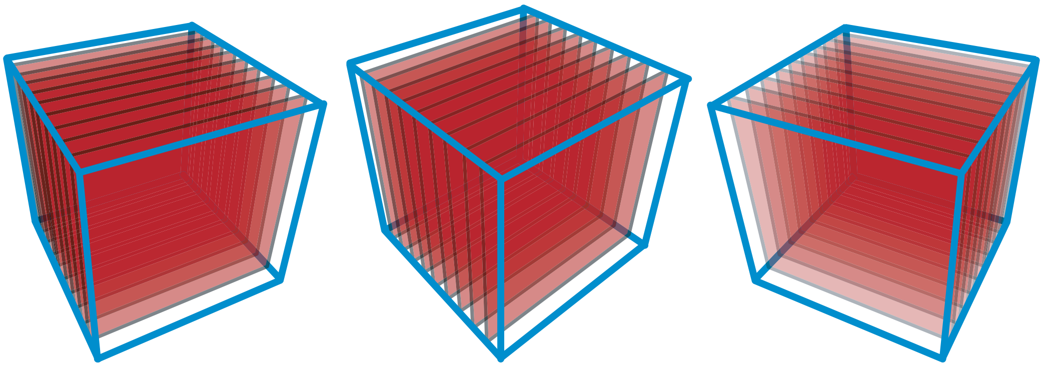


Figure 1.14: 2D texture-based Volume Rendering.

amount of graphics card memory. It is more practicable to use 3D textures, now supported by almost every graphics card.

3D Textures: This approach stores the volume data in memory as one three dimensional texture [11, 9, 77, 44]. For rendering, viewplane aligned polygons are drawn from back to front as illustrated in Figure 1.15. For each polygon the graphics hardware trilinear interpolates the correct colors and opacities from the 3D texture. These polygons are blended together using alpha blending. The entire set of viewing rays is treated simultaneously, resulting in a tremendous speedup due to the high rasterization performance of the graphics accelerator.

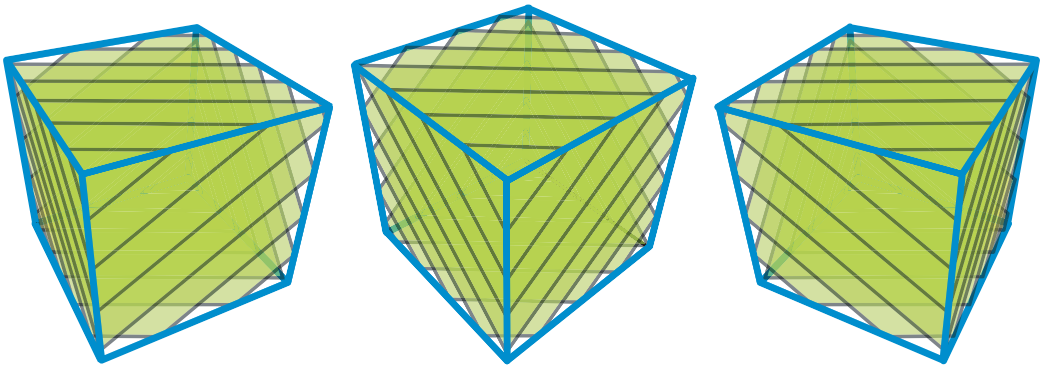


Figure 1.15: 3D texture-based Volume Rendering.

The main problem of these approaches is the limited amount of memory provided by graphics cards. In recent years the amount of available memory considerably increased due to strong demands of the gaming industry. Nevertheless, the amount of data produced by newer acquisition devices even increased with a higher pace. A considerable amount of research is devoted to overcome this issue by using compression and advanced data structures which allow to upload only the data needed for processing [18]. The increasing programmability of the graphics hardware motivated several acceleration techniques for GPU-based volume rendering [65, 30]. However, performance and implementation remains specific to hardware.

Special Purpose Hardware

Several special-purpose volume rendering architectures have been proposed, due to the high computational demands of direct volume rendering. Recent research has focused on accelerators for raycasting of rectilinear volume data, such as VOGUE [25], VIRIM [17], VIZARD II [46], and EM-Cube [56]. A comprehensive comparison of these architectures can be found in [63]. The EM-Cube architecture evolved into a commercially available volume rendering hardware, VolumePro board [58], which is capable of rendering a 512-cubed data set at approx. 30 frames/second.

Comparison of Direct Volume Rendering Approaches

A comparison of popular direct volume rendering algorithms can be found in [45]. Although research has evolved since this study was performed the basic conclusions are still valid. Shear-warp, as well as 3D Texture mapping, provide the highest performance, however, the performance benefit is accompanied by a loss of image quality. Raycasting and splatting produce comparable image quality. Regarding performance, splatting is superior if sparse data is processed, on the other hand, raycasting performs better for semitransparent, dense data.

1.2 Outline of Dissertation

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

Part I describes the handling and processing of large rectilinear volume data, specifically in the context of direct volume rendering. There are two major strategies to accelerate volume rendering. The first one is to reduce the computational costs at one resampling location. The second strategy is to efficiently avoid processing of data regions that do not affect the rendering result.

Most volume rendering systems based on CPU volume raycasting still suffer from inefficient CPU utilization and high memory usage. In Chapter 2 these issues are targeted and a new technique for efficient data addressing is presented. Furthermore, a new processing scheme for volume raycasting, that exploits thread-level parallelism is introduced. By using these techniques the computational cost at one resample position is significantly reduced.

Most CPU-based volume raycasting approaches achieve high performance by advanced memory layouts, space subdivision, and excessive precomputing. Such approaches typically need an enormous amount of memory. They are limited to sizes which do not satisfy the medical data sizes used in daily clinical routine. In Chapter 3 a solution is presented based on image-order raycasting with object-order processing; it is able to perform high-quality rendering of very large medical data in real-time on commodity computers. For large medical data, such as computed tomographic (CT) angiography run-offs (512x512x1202, 16 bit), rendering times of up to 2.5 fps on a commodity notebook are achieved. This is realized by introducing a memory efficient acceleration technique for on-the-fly gradient estimation, and a memory efficient hybrid removal and skipping technique of transparent regions. Quantized binary histograms, granular resolution octrees, and a cell invisibility cache are employed. These acceleration structures require just a small extra storage of approximately 10%.

In Chapter 4 an approach to efficiently visualize multiple intersecting volumetric objects is presented, using the previously presented techniques to accelerate mono-volume rendering. The concept of V-Objects is introduced. V-Objects represent abstract properties of an object connected to a volumetric data source. A method to perform direct volume rendering of a scene comprised of an arbitrary number of possibly intersecting V-Objects is presented. The main idea is to distinguish between regions of intersection; separating the ones that need costly multi-volume processing, from regions containing only one V-Object, which can be processed using a highly efficient brick-wise volume traversal scheme. By using this method, a significant performance gain for multi-volume rendering is achieved. Furthermore, possible medical applications, such as surgical planning, diagnosis, and education, are presented.

Part II deals with alternatives to grid-based volume graphics. In Chapter 5 Vots, a point-based representation of volumetric data, are introduced. An individual Vot is specified by the coefficients of a Taylor series expansion, i.e., the function value and higher order derivatives at a specific point. A Vot does not only represent a single sample point, it represents the underlying function within a region. With the Vot representation a more intuitive and high-level description of the volume data is achieved. This allows direct analytical examination and manipulation of volumetric data sets. Vots enable the representation of the underlying scalar function with specified precision. User-centric importance sampling is also possible, i.e., unimportant volume parts are still present but represented with just very few Vots. As a proof of concept Maximum Intensity Projection, based on Vots, is shown.

This dissertation concludes with Chapter 6, which summarizes the presented topics and achievements.

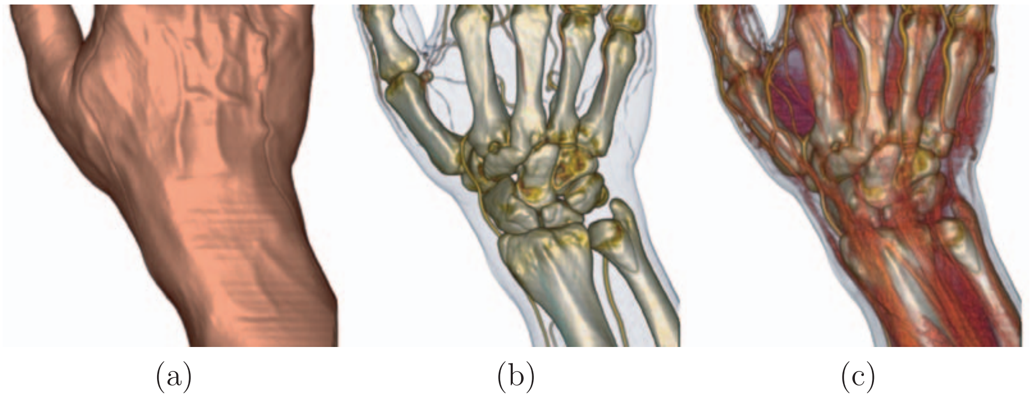
> *... in 10 years, all rendering will be volume rendering.*
>
> ---
>
> Jim Kajiya at SIGGRAPH '91 [10]

Part I

Efficient Raycasting of Rectilinear Volume Data

Chapter 2

Low-level Acceleration Techniques



(a) (b) (c)

Figure 2.1: CT scan of human hand (244x124x257, 16 bit).

This chapter is based on the following publications:

Grimm S., Bruckner S., Kanitsar A., Gröller E., **A Refined Data Addressing and Processing Scheme to Accelerate Volume Raycasting**, *Computers & Graphics*, 28(5), pp. 719-729, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **Hyper-Threaded Cache Coherent Raycasting**, Technical Report TR-186-2-03-05, *Vienna University of Technology*, 2003.

2.1 Introduction

Three main volume rendering approaches can be distinguished. Two of them are hardware based; the first one utilizes commodity graphics-cards [77, 30, 65, 18, 19]; the second one utilizes special purpose hardware, e.g. VolumePro [58] and Vizard [46]; the third is CPU based ([9, 85, 77, 43, 48]).

Purely hardware based solutions provide real-time performance and high quality; however, they are limited in their functionalities: basic visualization systems are supported by hardware volume rendering solutions; consequently, they are the mostly applied approach in the practice. Advanced visualization systems provide preprocessing features such as filtering, segmentation, and morphological operations. If such operations are not supported by hardware, they have to be performed on the CPU and the result must be transferred back. This transfer is very time consuming; thus, no interactive feed-back is possible.

In contrast to that, in a purely CPU based solution this transfer is unnecessary. Therefore, CPU based solutions are still commonly used in such systems. Another advantage of such a solution is the high flexibility it provides. Many high-level optimization techniques are developed to achieve high performance CPU solutions. Most of these techniques make one common assumption: only parts of the data need to be visualized. This assumption is still valid, but the data delivered by new higher-resolution acquisition devices increases rapidly. This introduces a new main issue: an enormous amount of data must be handled. Previous volume raycasting approaches, such as Knittel [26] or Mora [48], achieved impressive performance by using a spread memory layout. The main drawback of these approaches is the enormous memory usage. In both systems, the usage is approximately four-times the data size, plus storage for gradients if shading is used. This memory consumption is quite a limitation, considering that the maximum virtual address space is 3 GB on commodity computer systems. This issue is analyzed in order to present a new approach that uses significantly less memory. In contrast to other methods in the presented approach, all computations are done on-the-fly. To accelerate this on-the-fly computation, refined data addressing techniques for a bricked volume layout are presented. Furthermore, a data processing scheme is presented, which exploits common and new hardware technologies such as thread-level parallelism. Such a technology enables more efficient CPU utilization, and consequently provides a significant speedup.

The presentation of ideas is as follows: Section 2.2 surveys related work; Section 2.3 provides a brief introduction to caches, describes the used volume memory layout and the new data addressing schemes; Section 2.4 presents a highly efficient multi-threaded raycasting approach. Finally, in Section 2.5

the approach is discussed and in Section 2.6 the conclusions are presented.

2.2 Related Work

A prominent volume rendering approach which achieves high performance by using cache coherency is the Shear-Warp Factorization algorithm [32]. Cache coherency is achieved by resampling slice-wise and keeping the data in memory for each major viewing axis. The main drawbacks are the low quality and the threefold memory usage. In contrast to this Knittel [26] achieved very high cache coherency by introducing a spread memory layout for fast access. He virtually locked all needed address lookup tables and color lookup tables into the cache. This leads to a rather high cache coherency and, therefore, high CPU utilization; however, memory usage is increased by a factor of four. This memory storage requirement is too high, considering that the maximum virtual address space of today's mainstream workstations is three Gigabyte. Consequently, the maximum data-set size is limited by three Gigabyte divided by four = 768 MB. This seems to be an adequate size. However, in most of the visualization systems the simultaneous examination of multiple data sets is required. Furthermore, additional volumes or data-structures have to be kept in memory to support various operations such as segmentation and filtering. Mora [48] also based his approach on this spread-memory layout; he used an octree to obtain even more performance for first-hit volume raycasting. The enormous memory usage of both systems is a considerable limitation on state-of-the-art commodity computer systems. Moreover, these approaches are more limited by memory bandwidth than by CPU performance. This is not favorable, since the evolution of computer systems shows that CPU performance tends to increase faster than memory bandwidth does.

Law and Yagel [33] proposed a parallel raycasting algorithm for a massively parallel processing system: they proved that appropriate data subdivision and distribution to the available caches lead to high cache coherency. The scheme can be adapted to commodity single- and multi-processors. The use of this scheme leads to high cache coherency of all caches; however, high CPU utilization is not inherent. Thread-level parallelism and advanced data addressing schemes turn out to be a solution to this utilization issue. Throughout the research process the basic data processing scheme was extended, in order to significantly increase CPU utilization, accelerating the raycasting process.

2.3 Efficient Memory Layout

There are two main requirements to achieve high CPU utilization: first, execution units have to operate at full capacity; second a high cache hit rate is desirable, which implies that no cache trashing occurs. The first condition is fulfilled by thread-level parallelism, see Section 2.4. For the second condition working sets are defined so that they follow two known principles of locality, *temporal locality* - an item referenced now will be referenced again in the near future, and *spatial locality* - an item referenced now also causes its neighbors to be referenced.

The cache hierarchy of a x86-based system is shown in Figure 2.2. Going up the cache hierarchy towards the CPU, caches get smaller and faster. In general, if the CPU issues a load of a piece of data the request is propagated down the cache hierarchy until the requested data is found. It is very time consuming if the data is only found in a slow cache. This is a consequence of the propagation itself as well as of the back propagation of data through all the caches. Since the focus is on speed, frequent access to the slower caches has to be avoided. Accessing the slower caches, like hard disk and main memory, only once would be optimal. This is straightforwardly achieved for the hard disk level, as it is assumed that there is enough main memory to load all the data at once. To achieve this for the main memory is much more sophisticated.

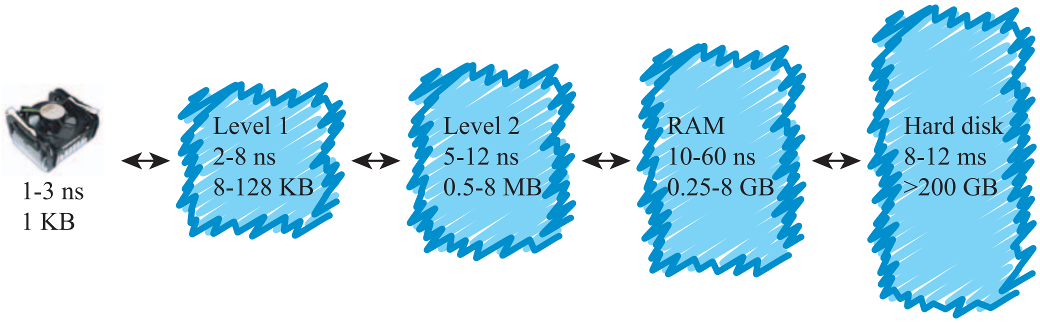


Figure 2.2: Cache hierarchy.

The work-flow of a standard volume raycasting algorithm on a linearly stored volume, commonly xyz-storage order, is as follows: for every pixel of the image plane a ray is shot through the volume and the volume data is resampled along this ray. At every resample position resampling, gradient computation, shading, and compositing is done. From a performance point of view this work flow is very inefficient:

- The closer the neighboring rays are to each other, the higher the probability is that they partially process the same data. Given the fact that rays are shot one after the other, the same data *has to be read several times from main memory*, because the cache is not large enough to hold the entire processed data of a single ray.

- Different viewing directions cause different numbers of cache-line requests to load the necessary data from main memory which leads to a *varying frame-rate*.

These are the two main reasons, which lead to a bad CPU utilization.

The main focus is on supporting semitransparent as well as first-hit views with high real-time performance, without increasing the memory usage dramatically. It is a known fact that bricking of data is an effective way to achieve high cache coherency, without increasing the memory usage. The concept of bricking supposes the decomposition of data into small fixed-sized data bricks. The brick data in this case is stored linearly in common xyz-order; the bricks themselves are linearly stored in common xyz-order. However, accessing data in a bricked volume layout is very costly. In contrast to the proposed two-level subdivision hierarchy in [57], a one-level subdivision of the volume data is chosen. This is due to the fact that every additional level introduces costs for addressing the data. For this one-level subdivision layout a very efficient addressing scheme is developed, see Section 2.3.1.

The presented raycasting system is able to support different brick-sizes, as long as each brick-dimension is a power of two. If the brick-size is set to the actual volume-dimensions a common raycaster is obtained that operates on a simple linear volume layout. This allows to make a meaningful comparison between a raycaster that operates on a simple linear volume layout and a raycaster which operates on a bricked volume layout with optimal brick size. To underline the effect of bricking, different brick sizes are benchmarked. Figure 2.3 shows the actual speedup achieved by brick-wise raycasting. For testing purposes a semitransparent transfer-function is specified, such that the impact of all high level optimizations, such as early ray termination, brick skipping, and zero-opacity skipping is overridden. In other words, the final image was the result of brute-force raycasting of the whole data. Data sizes up to 512 MB are tested. The size of the data set had no influence on the actual optimal performance gains. The exemplary tested data shown in Figure 2.3 was a 256x256x256 typical medical data set, with 16 bit per voxel. Furthermore, a worst-case comparison with respect to the viewing direction is done. In case of small bricks the worst case is similar to the best case. In contrast to that, using large bricks shows enormous performance decreases depending on the viewing direction. This is the well known fact of view-dependent performance of a raycaster operating on a simple linear volume layout. The constant performance behavior of small bricks is one of the main advantages of a bricked volume layout. There is nearly no view dependent performance variation anymore.

Going from left to right in the chart shown in Figure 2.3, first there is a speedup of about 2.0 with a brick-size of 1KB. Increasing the brick-size up to 64KB also increases the speedup. This is due to more efficient usage of the cache. The chart shows an optimum at a brick size of 64KB (32x32x32) with a speedup of about 2.8. This number shows the optimal tradeoff between the needed cache space for ray-structures, sample data, and the color lookup table. Further increase leads to performance decreases due to exceeding the cache capacity and bricking overhead. This performance drop-off is reduced, once the brick subdivisions almost correspond to no subdivision. In other words, with a brick size of the same size as the volume itself, the ray-structures have no influence on the performance. In this case there is only one brick and, therefore, only one list of rays must be processed. This is exactly the same rendering context of a common raycaster for a simple linear volume layout.

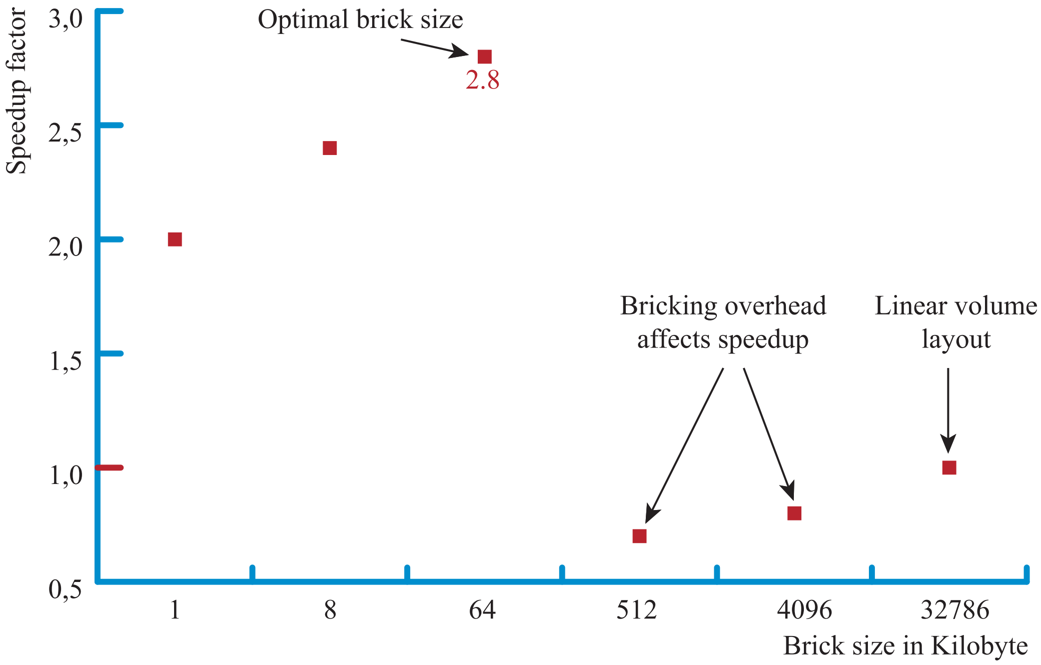


Figure 2.3: Brick-based raycasting speedup compared to common raycasting on a linear volume. Test system specification: Pentium 4 Xeon 512KB Level-2 cache.

2.3.1 Addressing

The evolution of CPU design shows that the length of CPU pipelines grows progressively. This is very efficient as long as conditional branches do not initiate pipeline flushes. Once a long instruction pipeline is flushed there is a significant delay until it is refilled. Most of the present systems use branch prediction. The CPU normally assumes that if-branches will always be executed. It starts processing the if-branch before actually checking the outcome of the if-clause. If the if-clause returns false, the else-branch has to be executed. This means that the CPU flushes the pipeline and refills it with the else-branch. This is a very time consuming procedure.

Using a bricked volume layout one will encounter this problem. The addressing of data in a bricked volume layout is more costly than in a linear volume layout. To address one data element, one has to address the brick itself and the element within the brick. In contrast to this addressing scheme, a linear volume can be seen as one large brick. To address one sample it is enough to compute just one offset. In algorithms like volume raycasting, which need to address a certain neighborhood of data in each processing step, the computation of two offsets instead of one has a considerable performance impact. To avoid this performance penalty, an if-else statement can be constructed. The if-clause consists of testing, if the needed data elements can be addressed within one brick. If the outcome is true, the data elements can be addressed as fast as in a linear volume. If the outcome is false, the costly address calculations have to be done. On the one hand, this simplifies address calculation, but on the other hand the involved if-else statement incurs pipeline flushes. In the following this issue is addressed.

For raycasting, two major neighborhood access patterns are distinguished. The first one is for resampling; the second one is for gradient computation. The latter will be solved generally for a 26-connected neighborhood access pattern. For the resampling computation the eight surrounding samples are needed. The necessary address computations in a linear volume layout are:

$$\begin{aligned}
&\text{SampleOffset}_{i,j,k} && \rightarrow \text{i+j}\cdot\text{D}_x\text{+k}\cdot\text{D}_x\cdot\text{D}_y \\
&\text{SampleOffset}_{i+1,j,k} && \rightarrow \text{SampleOffset}_{i,j,k}+1 \\
&\text{SampleOffset}_{i,j+1,k} && \rightarrow \text{SampleOffset}_{i,j,k}+\text{D}_x \\
&\text{SampleOffset}_{i+1,j+1,k} && \rightarrow \text{SampleOffset}_{i,j,k}+1+\text{D}_x \\
&\text{SampleOffset}_{i,j,k+1} && \rightarrow \text{SampleOffset}_{i,j,k}+\text{D}_x\cdot\text{D}_y \\
&\text{SampleOffset}_{i+1,j,k+1} && \rightarrow \text{SampleOffset}_{i,j,k}+1+\text{D}_x\cdot\text{D}_y \\
&\text{SampleOffset}_{i,j+1,k+1} && \rightarrow \text{SampleOffset}_{i,j,k}+\text{D}_x+\text{D}_x\cdot\text{D}_y \\
&\text{SampleOffset}_{i+1,j+1,k+1} && \rightarrow \text{SampleOffset}_{i,j,k}+1+\text{D}_x+\text{D}_x\cdot\text{D}_y
\end{aligned}$$

Thereby $\text{D}_{\{x,y,z\}}$ define the volume dimensions and i, j, k the integer parts of the current resample position in 3D. This addressing scheme is very efficient.

Once the lower left sample is determined the other needed samples can be accessed just by adding an offset. In contrast to the linear volume addressing, the brick volume addressing is given by:

```
if((i' < $BD_x$-1) and (j' < $BD_y$-1) and (k' < $BD_z$-1 ))
{
  $\text{SampleOffset}_{i,j,k} \rightarrow i'+j'\cdot BD_x+k'\cdot BD_x\cdot BD_y$
  $\text{SampleOffset}_{i+1,j,k} \rightarrow \text{SampleOffset}_{i,j,k}+1$
  $\text{SampleOffset}_{i,j+1,k} \rightarrow \text{SampleOffset}_{i,j,k}+BD_x$
  $\text{SampleOffset}_{i+1,j+1,k} \rightarrow \text{SampleOffset}_{i,j,k}+1+BD_x$
  $\text{SampleOffset}_{i,j,k+1} \rightarrow \text{SampleOffset}_{i,j,k}+BD_x\cdot BD_y$
  $\text{SampleOffset}_{i+1,j,k+1} \rightarrow \text{SampleOffset}_{i,j,k}+1+BD_x\cdot BD_y$
  $\text{SampleOffset}_{i,j+1,k+1} \rightarrow \text{SampleOffset}_{i,j,k}+BD_x+BD_x\cdot BD_y$
  $\text{SampleOffset}_{i+1,j+1,k+1} \rightarrow \text{SampleOffset}_{i,j,k}+1+BD_x+BD_x\cdot BD_y$
}
else
{
  $\text{SampleOffset}_{i,j,k} \rightarrow i'+j'\cdot BD_x+k'\cdot BD_x\cdot BD_y$
  $\text{SampleOffset}_{i+1,j,k} \rightarrow$ ComputeOffset(i+1,j,k)
  $\text{SampleOffset}_{i,j+1,k} \rightarrow$ ComputeOffset(i,j+1,k)
  $\text{SampleOffset}_{i+1,j+1,k} \rightarrow$ ComputeOffset(i+1,j+1,k)
  $\text{SampleOffset}_{i,j,k+1} \rightarrow$ ComputeOffset(i,j,k+1)
  $\text{SampleOffset}_{i+1,j,k+1} \rightarrow$ ComputeOffset(i+1,j,k+1)
  $\text{SampleOffset}_{i,j+1,k+1} \rightarrow$ ComputeOffset(i,j+1,k+1)
  $\text{SampleOffset}_{i+1,j+1,k+1} \rightarrow$ ComputeOffset(i+1,j+1,k+1)
}
ComputeOffset(i,j,k) $\rightarrow \text{BlkOffset}_{i,j,k}\cdot(BD_x\cdot BD_y\cdot BD_z)$ +
                       $\text{OffsetWithinBlk}_{i,j,k}$
$\text{BlkOffset}_{i,j,k} \rightarrow (i''+j''\cdot BVD_x+k''\cdot(BVD_x\cdot BVD_y))$
$\text{OffsetWithinBlk}_{i,j,k} \rightarrow (i'+j'\cdot BD_x+k'\cdot(BD_x\cdot BD_y)$
```

Thereby $BD_{\{x,y,z\}}$ define the brick dimensions, $D_{\{x,y,z\}}$ define the volume dimensions. $BVD_{\{x,y,z\}}$ denote the brick volume dimensions defined by $BVD_{\{x,y,z\}} = (D_{\{x,y,z\}}/BD_{\{x,y,z\}})$, i, j, and k are the integer parts of the current resample 3D-position, i', j', k' are defined by i' = (i mod BD_x), j' = (j mod BD_y), and k' = (k mod BD_z), and i", j", k" are defined by i" = (i div BD_x), j" = (j div BD_y), and k" = (k div BD_z).

To avoid the costly if-else statement and the expensive address computations, a lookup table to address all the needed samples is created.

The straight-forward approach would be to create a lookup table for each possible sample position in a brick. Since the optimal brick size is 32^3, this would mean that 32^3 different lookup tables are needed to address the

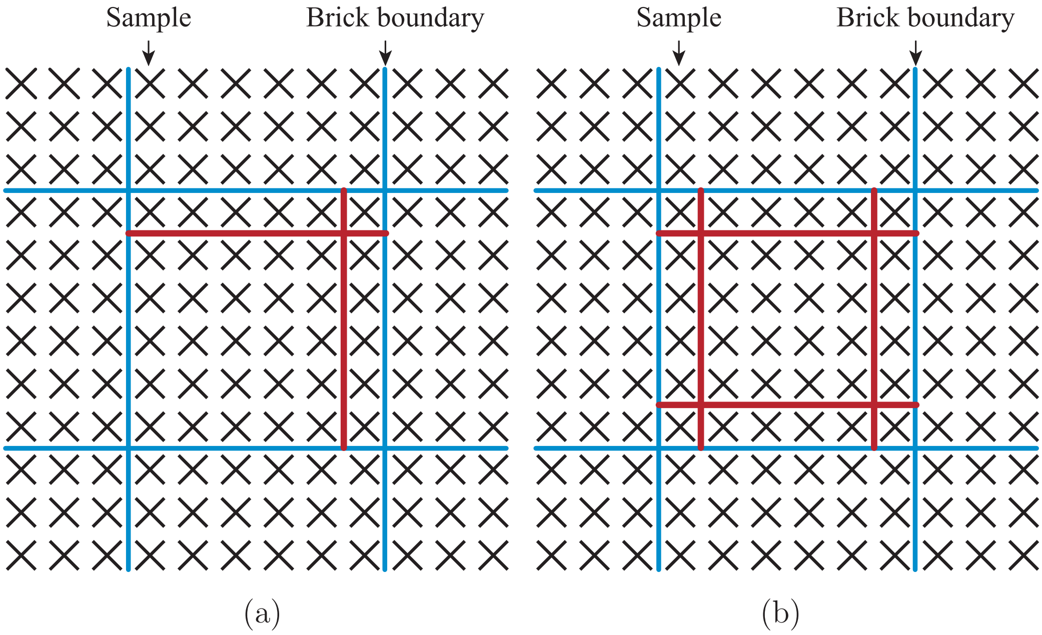
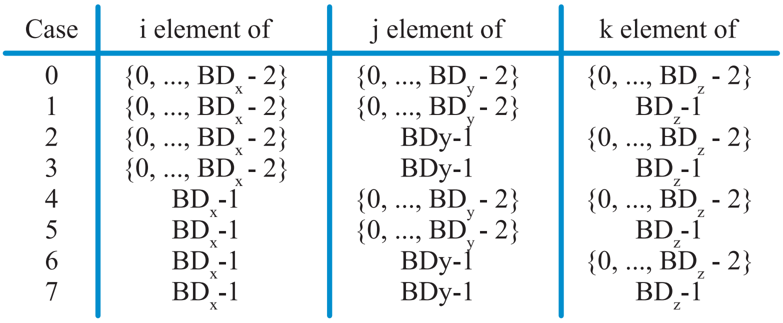


Figure 2.4: Sample position (i, j, k) is defined by the integer parts of the resample position. The sample positions (i, j, k) of a brick are subdivided into subsets. The membership depends on the location of the adjacent samples. They are either in the same brick or in one of the neighboring bricks. (a) Resampling: Four areas, because only samples to the right and to the top are accessed. (b) Gradient computation: Nine subsets, because samples in every direction are accessed.

neighboring samples. In the resampling case, seven neighbors need to be addressed; the corresponding size of the lookup tables is $32^3 \cdot 7 \cdot 4$ Bytes $= 896$ KB (4 Bytes per offset). In the gradient computation case more neighbors need to be addressed: 26 neighbors need to be addressed, which leads to a table size of $32^3 \cdot 26 \cdot 4$ Bytes $=$ 3,25 MByte (4 Bytes per offset) in total. Such a huge size for a lookup table is not preferable, due to the limited size of the cache. However, the addressing of such a lookup table is straightforward, because the indexes in the lookup table are the corresponding offsets of the current sample position, assuming the given offset is relative to the brick memory address. A more efficient approach is developed, first for the resampling access pattern and then for a general 26-connected neighborhood, commonly needed for gradient estimation.

2.3.2 Efficient Addressing: Resampling

The possible resample locations are distinguished by the locations of the needed neighboring samples. The first sample location (i, j, k) is defined by the integer parts of the current resample position. The access pattern of adjacent samples during resampling is defined by accessing samples to the right, top, and back. Moreover, the samples of a brick are subdivided into subsets. For the largest subset the seven adjacent samples of a sample (i, j, k) lie within the same brick. The other subsets are defined by samples (i, j, k) on the border of the current brick. The adjacent samples lie partially or completely within neighboring bricks. These other subsets are defined by the needed neighbor bricks to access all seven adjacent samples. The 2D case is illustrated in Figure 2.4a. Only samples to the right and to the top are needed; thus, there are only four cases. Basically, if the sample (i, j) lies on one or two of the brick faces (top-, and right-face), neighboring bricks must be accessed. This can be straightforwardly mapped to the 3D case, by also taking into account the back-face. The eight occurring cases in 3D are shown in Figure 2.5.



Case	i element of	j element of	k element of
0	$\{0, ..., BD_x - 2\}$	$\{0, ..., BD_y - 2\}$	$\{0, ..., BD_z - 2\}$
1	$\{0, ..., BD_x - 2\}$	$\{0, ..., BD_y - 2\}$	BD_z-1
2	$\{0, ..., BD_x - 2\}$	BDy-1	$\{0, ..., BD_z - 2\}$
3	$\{0, ..., BD_x - 2\}$	BDy-1	BD_z-1
4	BD_x-1	$\{0, ..., BD_y - 2\}$	$\{0, ..., BD_z - 2\}$
5	BD_x-1	$\{0, ..., BD_y - 2\}$	BD_z-1
6	BD_x-1	BDy-1	$\{0, ..., BD_z - 2\}$
7	BD_x-1	BDy-1	BD_z-1

Figure 2.5: The eight neighbor brick constellations.

As mentioned before, the bricks themselves are stored in xyz-order, therefore, the necessary offsets for the eight neighboring samples can be precomputed and stored in a lookup table. Furthermore, the lookup table is the same for each brick. The lookup table contains $8 \cdot 7 = 56$ offsets. There are eight cases, and for each sample (i, j, k) the offsets to its seven adjacent samples are needed. The seven neighbors are accessed relative to the sample (i, j, k). Since each offset consists of four bytes the table size is 224 bytes. This is an improvement of a factor of 4096 compared to the straight-forward solution.

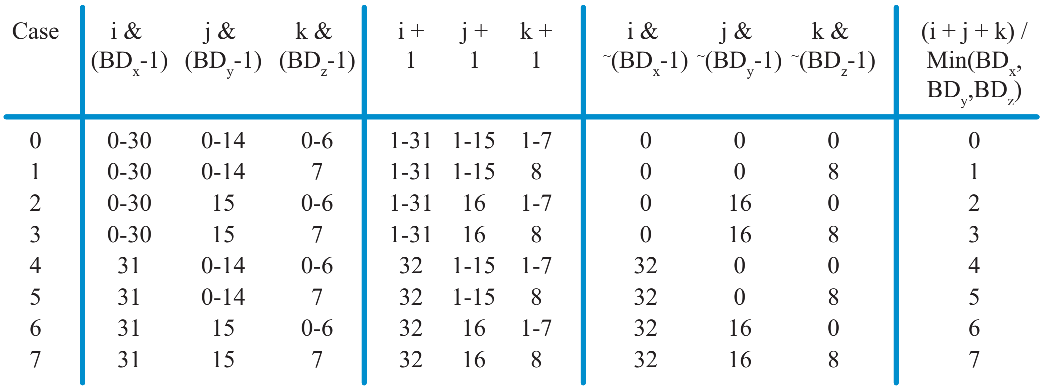
By compressing the lookup tables in this way the index computations for the lookup table access become more difficult. It can be achieved efficiently if the brick dimensions are a power of two, and a power of two apart. The second constraint can be removed by introducing a simple shift operation to virtually keep the constraint. To exemplify the algorithm it is assumed that the brick dimensions are 32x16x8. The input of the lookup table addressing function is the sample position (i, j, k). As first step the brick offset part from i, j, and k is extracted by anding the corresponding BD$_{\{x,y,z\}}$-1. The result can be seen in Figure 2.6 second column. The next step is to increase all by one. By this operation the current maximal possible value BD$_{\{x,y,z\}}$-1 is moved to BD$_{\{x,y,z\}}$. All the other possible values stay within the range [1,BD$_{\{x,y,z\}}$-1]. Then a conjunction of the resulting value and the complement of BD$_{\{x,y,z\}}$-1 is performed. After this operation the input values are mapped to $\{0$, BD$_{\{x,y,z\}}\}$, as shown in Figure 2.6, in column four. The last and final step is to sum up the three values and divide the result by the minimum of the three brick-dimensions BD$_{\{x,y,z\}}$, which maps the result into the range [0,7]. This last division can be substituted by a shift operation. The final algorithm for a 32x16x8 brick is:

```
SampleOffset$_{i,j,k}$             → ComputeOffset(i,j,k)
Index                              → ((((i&0x1F)+1)&0xE0)+
                                   →  (((j&0x0F)+1)&0xF0)+
                                   →  (((k&0x07)+1)&0xF8))>>3
SampleOffset$_{i+1,j,k}$           → SampleOffset$_{i,j,k}$+Lut[Index][0]
SampleOffset$_{i,j+1,k}$           → SampleOffset$_{i,j,k}$+Lut[Index][1]
SampleOffset$_{i+1,j+1,k}$         → SampleOffset$_{i,j,k}$+Lut[Index][2]
SampleOffset$_{i,j,k+1}$           → SampleOffset$_{i,j,k}$+Lut[Index][3]
SampleOffset$_{i+1,j,k+1}$         → SampleOffset$_{i,j,k}$+Lut[Index][4]
SampleOffset$_{i,j+1,k+1}$         → SampleOffset$_{i,j,k}$+Lut[Index][5]
SampleOffset$_{i+1,j+1,k+1}$       → SampleOffset$_{i,j,k}$+Lut[Index][6]
```

The *ComputeOffset* step can be simplified to only the offset calculation within one brick; this is possible as the processing is done brick-wise; therefore, the brick-offset remains constant, while processing one brick.

2.3.3 Results

Compared to the if-else solution which has the costly computation of two offsets in the else branch, a speed up of about 30% is achieved. The benefit varies, depending on the brick dimensions. For a 32x32x32 brick size the else-branch has to be executed in 10% of the cases and for a 16x16x16 brick size in 18% of the cases. With larger brick-sizes the percentage of the else-branch executions is smaller and, therefore, also the benefit decreases. The focus is, nevertheless, on small brick sizes, for which the overhead is significantly reduced. The other important benefit is that it does not matter anymore where in the brick adjacent samples are accessed; addressing is always performed with constant computational time.



Case	i & (BD_x-1)	j & (BD_y-1)	k & (BD_z-1)	i + 1	j + 1	k + 1	i & ~(BD_x-1)	j & ~(BD_y-1)	k & ~(BD_z-1)	(i + j + k) / Min(BD_x, BD_y,BD_z)
0	0-30	0-14	0-6	1-31	1-15	1-7	0	0	0	0
1	0-30	0-14	7	1-31	1-15	8	0	0	8	1
2	0-30	15	0-6	1-31	16	1-7	0	16	0	2
3	0-30	15	7	1-31	16	8	0	16	8	3
4	31	0-14	0-6	32	1-15	1-7	32	0	0	4
5	31	0-14	7	32	1-15	8	32	0	8	5
6	31	15	0-6	32	16	1-7	32	16	0	6
7	31	15	7	32	16	8	32	16	8	7

Figure 2.6: Lookup table addressing for resampling. Thereby $BD_{\{x,y,z\}} = \{32,16,8\}$.

2.3.4 Efficient Addressing: Gradient Estimation

A similar addressing approach, as presented for resampling, can also be done for the gradient estimation. A general solution for a 26-connected neighborhood is presented. Analogous to the resample case, 27 cases are distinguished. The 2D case is illustrated in Figure 2.4b. Depending on the position of sample (i, j, k) a brick is subdivided into 27 subsets. In contrast to the resample situation, additional sample positions on the bottom-, left-, and front faces must be handled.

The first step is to extract the brick offset, by anding $BD_{\{x,y,z\}}$-1 as shown in Table 2.7, second column. Then, one is subtracted, and a conjunction with $BD_{\{x,y,z\}}$+$BD_{\{x,y,z\}}$-1 is used to separate the case if one or more components are zero. In other words zero is mapped to (2·$BD_{\{x,y,z\}}$-1) (Table 2.7, third column). All the other values stay within the range {0,...,$BD_{\{x,y,z\}}$-2}. The other case which has to be separated is the case if one or more of the components are $BD_{\{x,y,z\}}$-1. This can be done by adding one, after the previous minus one operation is undone by a disjunction with 1, without loosing the separation of the zero case. The result can be seen in Table 2.7, fourth column. Now all the cases are mapped to {0,1,2} to obtain a ternary-system; this is achieved by dividing the components by the corresponding brick-dimensions. These divisions can be substituted by fast shift operations. The last and final step is then the calculation of 9·i+3·j+k to get unique values in the range of [0,26]. The final lookup table index computation for a 32x16x8 brick is:

Case	i &	j &	k &	i - 1 &	j - 1 &	k - 1 &	i | 1	i | 1	i | 1	i / BD_x	j / BD_y	k / BD_z	9 i + 3 j + k
	(BD_x - 1)	(BD_y - 1)	(BD_z - 1)	(BD_x + BD_x - 1)	(BD_y + BD_y - 1)	(BD_z + BD_z - 1)	+1	+1	+1				
0	1 - 30	1 - 14	1 - 6	0 - 29	0 - 13	0 - 5	2 - 30	2 - 14	2 - 6	0	0	0	0
1	1 - 30	1 - 14	7	0 - 29	0 - 13	6	2 - 30	2 - 14	8	0	0	1	1
2	1 - 30	1 - 14	0	0 - 29	0 - 13	15	2 - 30	2 - 14	16	0	0	2	2
3	1 - 30	15	1 - 6	0 - 29	14	0 - 5	2 - 30	16	2 - 6	0	1	0	3
4	1 - 30	15	7	0 - 29	14	6	2 - 30	16	8	0	1	1	4
5	1 - 30	15	0	0 - 29	14	15	2 - 30	16	16	0	1	2	5
6	1 - 30	0	1 - 6	0 - 29	31	0 - 5	2 - 30	32	2 - 6	0	2	0	6
7	1 - 30	0	7	0 - 29	31	6	2 - 30	32	8	0	2	1	7
8	1 - 30	0	0	0 - 29	31	15	2 - 30	32	16	0	2	2	8
9	31	1 - 14	1 - 6	30	0 - 13	0 - 5	32	2 - 14	2 - 6	1	0	0	9
10	31	1 - 14	7	30	0 - 13	6	32	2 - 14	8	1	0	1	10
11	31	1 - 14	0	30	0 - 13	15	32	2 - 14	16	1	0	2	11
12	31	15	1 - 6	30	14	0 - 5	32	16	2 - 6	1	1	0	12
13	31	15	7	30	14	6	32	16	8	1	1	1	13
14	31	15	0	30	14	15	32	16	16	1	1	2	14
15	31	0	1 - 6	30	31	0 - 5	32	32	2 - 6	1	2	0	15
16	31	0	7	30	31	6	32	32	8	1	2	1	16
17	31	0	0	30	31	15	32	32	16	1	2	2	17
18	0	1 - 14	1 - 6	63	0 - 13	0 - 5	64	2 - 14	2 - 6	2	0	0	18
19	0	1 - 14	7	63	0 - 13	6	64	2 - 14	8	2	0	1	19
20	0	1 - 14	0	63	0 - 13	15	64	2 - 14	16	2	0	2	20
21	0	15	1 - 6	63	14	0 - 5	64	16	2 - 6	2	1	0	21
22	0	15	7	63	14	6	64	16	8	2	1	1	22
23	0	15	0	63	14	15	64	16	16	2	1	2	23
24	0	0	1 - 6	63	31	0 - 5	64	32	2 - 6	2	2	0	24
25	0	0	7	63	31	6	64	32	8	2	2	1	25
26	0	0	0	63	31	15	64	32	16	2	2	2	26

Figure 2.7: Lookup table addressing for a 26-connected neighborhood. Thereby $BD_{\{x,y,z\}} = \{32,16,8\}$.

```
i’     → ((i & 0x1F) - 1) & 0x3F
j’     → ((j & 0x0F) - 1) & 0x1F
k’     → ((k & 0x07) - 1) & 0x0F
i”     → ((i’ | 0x01) + 1) >> 5
j”     → ((j’ | 0x01) + 1) >> 4
k”     → ((k’ | 0x01) + 1) >> 3
Index → (i”·9+j”·3+k”)
```

2.3.5 Results

Compared to the if-else solution which has the costly computation of two offsets in the else branch, a speed up of about 40% is achieved. The index computation is more costly compared to the resample lookup table indexing computation. However, the percentage where the else-branch has to be executed nearly doubled. Consequently, the more costly index computation is compensated by the higher percentage of costly cases. The size of the lookup table has not been discussed so far. It is 27 cases $\cdot$ 26 offsets $\cdot$ 4 byte per offset = 2808 Bytes. This can be reduced by a factor of two due to symmetry reasons. Therefore, a very small lookup table of 1404 Bytes is obtained. This is an improvement of approximately a factor of 2427 compared to the straight-forward solution. Thus, the resample lookup table and the 26-connected neighborhood lookup table fit into 2KB.

It is assumed that bricks are stored linearly; this has simplified the explanation of the addressing scheme. However, storing the bricks at arbitrary locations in memory is also possible. It requires creating a specific lookup table for each brick. The base structure of the address lookup tables and their indexing remain the same; only the stored offsets change according to the memory locations of the adjacent neighboring bricks. This possibility enables the exploitation of different brick arrangements, such as arrangements based on space filling curves, to improve the spatial locality. Storing an address lookup table for each brick requires only a small additional storage of $(1/65536) \cdot (2808 + 224) \cdot 100 \approx 4.63\%$ per brick. The brick size in this case is $32^3 \cdot 2$ byte = 65536 byte, the resample lookup table size is 224 byte, and the 26-neighbor address lookup table size is 2808 byte. The symmetry of the 26-neighbor address lookup tables can not be exploited, due to the arbitrary brick arrangement requirement.

Another possible option to simplify the addressing would be to inflate each brick by an additional border of samples. However, such a solution considerably increases the overall memory usage. For instance, for a brick size of 32x32x32 the total memory usage of the volume data by is increased approximately 20%. This is an inefficient usage of memory resources and the storage redundancy reduces the effective memory bandwidth. In contrast to that, our approach has a memory usage increase of just 4.63 % per brick even if an arbitrary brick arrangement is permitted. Hardly any additional memory is required for a linear brick arrangement, as all bricks share one global address lookup table. In conclusion, a very efficient approach to access neighboring samples within a brick based volume layout by using a small lookup table has been presented. Furthermore, a refined index computation has been developed to access the lookup tables in a very efficient manner.

2.4 Multi-threading

So far, CPU designers have tried to improve the CPU performance mainly by increasing the clock-rate. Today's main processors perform at 3 GHz resulting in 0.33 ns per clock-cycle. Achieving higher rates becomes more and more difficult due to physical laws and manufacturing costs. Other strategies to increase CPU performance were explored. The Pentium CPU was the first to allow the parallel execution of several instructions per clock-cycle. However, this feature was insufficient, because normally there are not enough sequential instructions that can be performed in parallel. To overcome this issue an out-of-order execution unit was introduced. This unit reorders the instruction stream such that the CPU can execute more instructions in parallel. This concept is called instruction level parallelism. At first sight this is a very efficient solution, but studies have shown that in a typical application at most 2.5 instructions can be found to be executed simultaneously [40, 27].

Thus, there are still unused execution resources on the CPU. In order to use them, hyper-threading technology for commodity CPUs has been introduced to exploit thread-level parallelism. With this technology, the CPU designers go one step further. Additionally to the instruction level parallelism, thread level parallelism (TLP) is introduced to identify even more instructions for parallel execution. In the past, the out-of-order execution unit could choose from an instruction buffer of only one thread. Now, this buffer contains instructions of two threads which obviously increases the likelihood of finding data-independent instructions. This technology makes a single *physical* processor appear as two *logical* processors. It just duplicates the architectural state, while the physical execution resources and caches are shared, see Figure 2.8. In other words, the CPU is capable of holding two thread contexts at the same time. The two threads are executed simultaneously on the same execution units, using the same caches. If one thread stalls due to a cache miss, the other one uses the idle execution resources.

More information to hyper-threading technology can be found in [22, 23, 40].

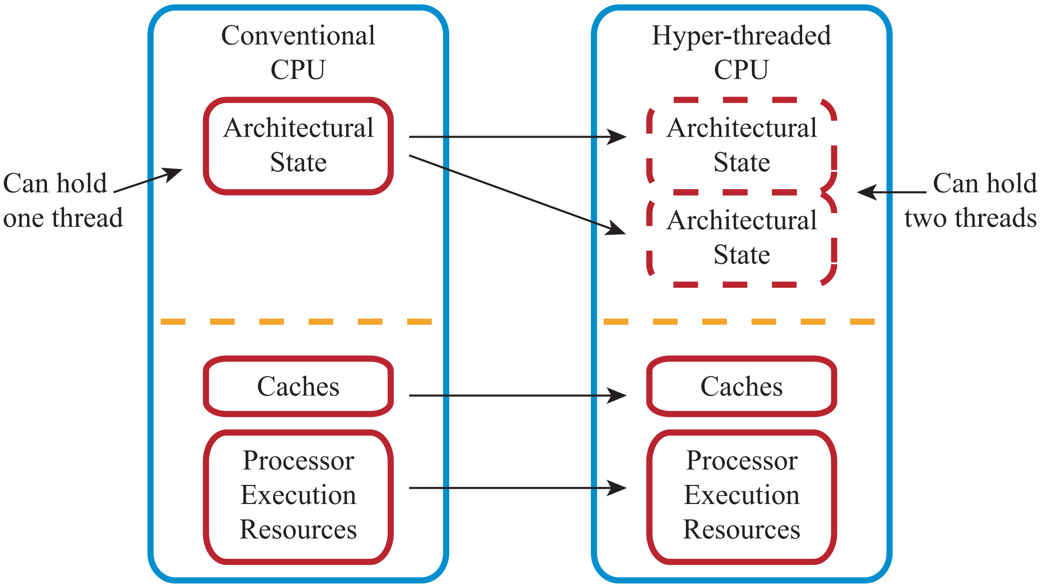


Figure 2.8: Hyper-threading technology duplicates the architectural state of the physical processor, providing two *logical* processors.

In Section 2.3 a bricked memory volume layout with a highly optimized addressing scheme has been presented. This layout is now the base for a highly efficient volume raycasting system. The main idea to perform raycasting on a bricked volume layout is to have data working-sets which can be shared between two hyper-threads. This is very important since hyper-threads share caches.

2.4.1 Processing Scheme

To get optimal cache coherence, high CPU utilization, and the ability to do efficient threading, the system is based on Law's and Yagel's [33] raycasting method. They proposed a parallel raycasting algorithm for a massively parallel processing system (Cray T3D Supercomputer). The data was subdivided into small units and then evenly distributed among the processors, so that optimal cache coherence was achieved. This distribution scheme can still be used on current multi-processor systems; however, it can not be used straight-forward for a hyper-threaded system. Therefore, their distribution scheme is extended to support logical CPUs within one physical CPU. The

```
Initialization:
1.) Create ordered list of bricks.

Preprocessing
2.) Assign rays to bricks that are hit first.

Raycasting:
for (all bricks b)
    while (brick b contains rays to process)
        for (all active rays r of brick b)
            1.) Resampling at position of ray r.
            2.) Gradient computation at position of ray r.
            3.) Shading at position of ray r.
            4.) Compositing at position of ray r.
            5.) Advance ray r.
            6.) if (ray enters subsequent brick)
                    (i) Remove ray r from current brick.
                    (ii) Assign ray r to subsequent brick.
```

Figure 2.9: Brick-wise raycasting algorithm.

main difference is that logical CPUs within one physical CPU need to operate on the same data to be efficient.

The work-flow of the algorithm, as shown in Figure 2.9, is as follows: It is based on the previous described bricked volume layout and uses the optimal brick size of 32x32x32, see Section 2.3. Each brick contains data-structures for high-level optimizations and a reference to a list of rays to process. The bricks themselves are stored in xyz-order. Initially a list of bricks is created. It is sorted by the traversal order of the rays. Therefore, each brick has to be processed only once. Law and Yagel [33] showed that this has to be done only once for the eight view quadrants, see Figure 2.10. Each brick has initially an empty list of rays. In the preprocessing phase all viewing rays are assigned to those bricks through which they first enter the volume. During volume raycasting, each of the bricks is processed until all the rays enter subsequent bricks. If a ray enters a subsequent brick, it is removed from the current brick

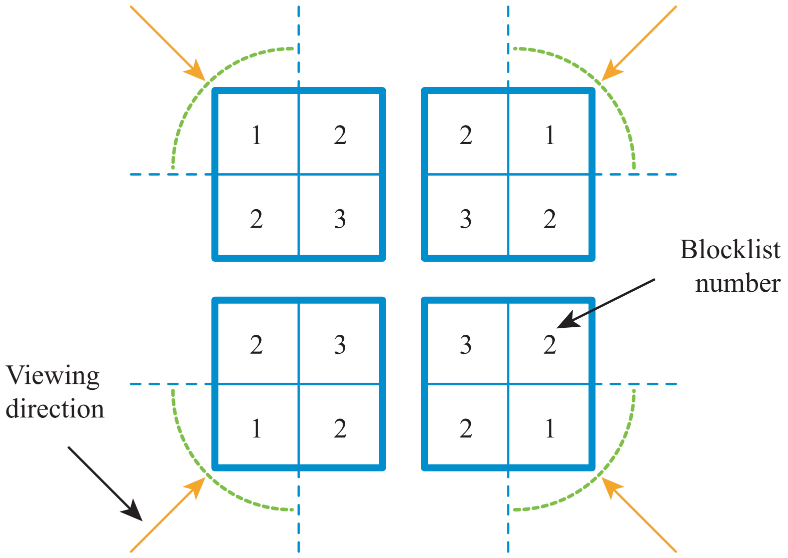


Figure 2.10: Brick-wise processing order.

and assigned to the subsequent one. Subsequent bricks, which now contain the rays, are processed in the same manner. By this mechanism the rays are completely carried through the volume when all bricks are processed. These bricks basically define the data working-sets mentioned in Section 2.3.

The work-flow of the volume raycasting system exploiting thread-level parallelism on a system with two physical CPUs supporting hyper-threading technology, illustrated in Figure 2.11, is as follows:
In the beginning seven treads, T1, ...,T7, are started. T1 is responsible for all the preprocessing. In particular it has to assign the rays to those bricks through which the rays enter the volume first. Then T1 chooses the lists of bricks which can be processed simultaneously, with respect to the eight distinguished viewing directions. Each list is evenly subdivided by T1 and sent to T2 and T3. After a list is sent, T1 sleeps until its slaves are finished. Then, it sends the next list to process, and so on. T2 sends one brick after the other to T4 and T5. T3 sends one brick after the other to T6 and T7. After a brick is sent, T2 and T3 sleep until their slaves are finished. Then they send the next brick to process, and so on. T4, T5, T6, and T7 perform the actual raycasting. T4 and T5 simultaneously process one brick, and T6 and T7 simultaneously process one brick. By this mechanism all bricks are processed in the correct order. During this process the most critical part is when rays are assigned to subsequent blocks. As mentioned before, each block holds a list of rays to process. It can happen that several threads simultaneously

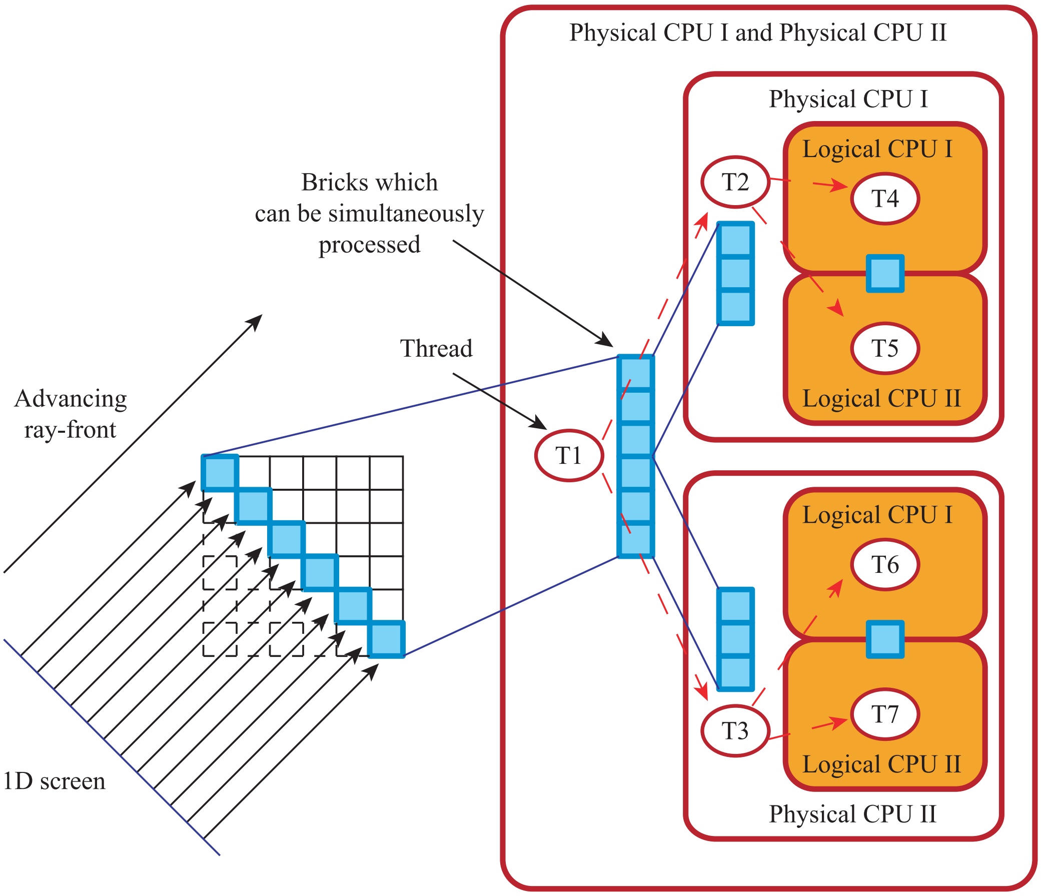


Figure 2.11: Volume raycasting system exploiting thread-level parallelism speedup.

want to assign rays to the same block. This problem is illustrated in Figure 2.12a. The straight-forward solution would be to ensure that only one thread at a time can assign rays to a block. But this decreases the performance drastically. There are two common synchronization mechanism to choose from, one is a mutex the other is to define a critical section. In general, a critical section is used to synchronize threads, because it is considerably faster. Using hyper-threading, however, introduces the same synchronization issue as when using multi-processing. Once two threads can be executed simultaneously a critical section is internally implemented as a slower mutex. This leads to an enormous performance decrease, due to the fact that the operating system has to check the mutex periodically to wake up waiting threads if the mutex is free. Performance can be increased by setting the spin loop count, in other words by changing the period of time the operating system is checking the mutex. Since this is still not optimal a solution is developed without any synchronization; it is illustrated in Figure 2.12b. Each thread has its own ray-list in a block. Thus, if a thread assigns a ray to a block, it is ensured that it is the only one which is writing into that list. There are four threads and only two threads are processing a single block, therefore, each thread has to read from two lists at a time. With this approach the rays of all lists are processed. Processing also includes removing rays from the lists. However, this is not critical. It is ensured by the processing order of the blocks that a ray can never be assigned to a block which is currently being processed. The last open question is the load-balancing. This is done by interleaving the rays during initialization.

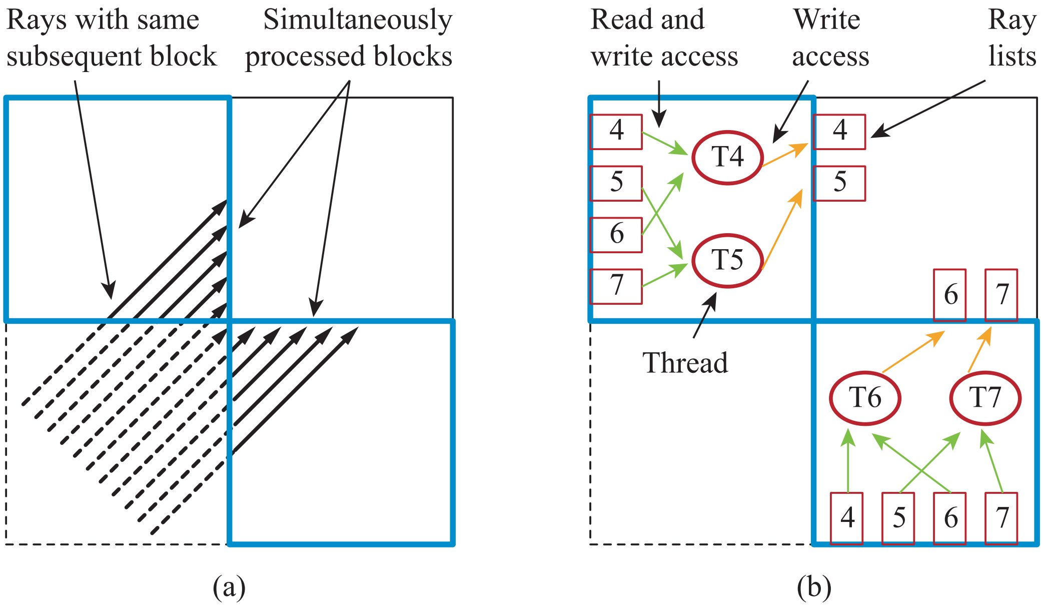


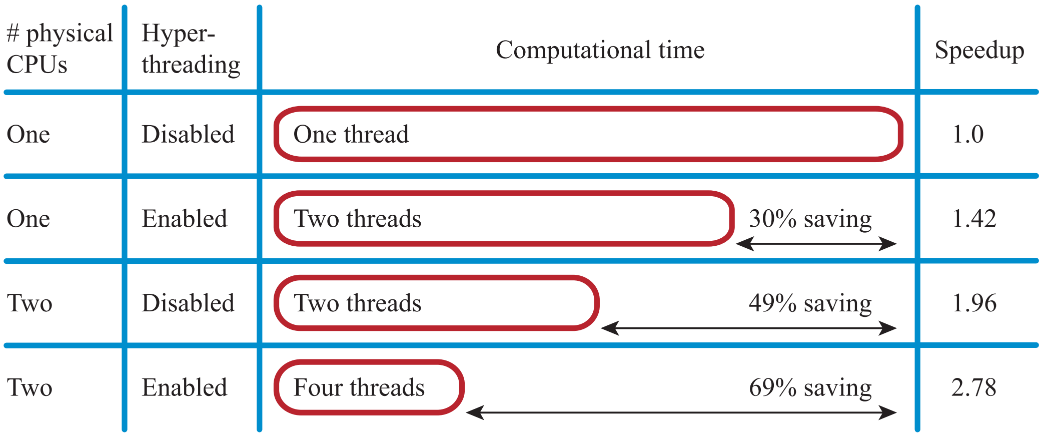
Figure 2.12: (a) Several threads assign rays to the same subsequent block. (b) Solution without synchronization.

Implementation issues

In order to keep complete control over the thread execution and synchronization flow, no special parallel programming library was used. The threads are created once during start-up, according to the number of physical and logical CPUs and synchronized by light-weight events. Expensive thread creation is avoided and thread context switches are kept low. For optimal CPU utilization a CPU-specific compiler was employed. Full optimization was enabled, performing interprocedural optimizations and inlining across multiple source files.

2.4.2 Results

Test system specification: Dual Pentium Xeon 2.4 GHz, 512 KB level-2 cache, 8 KB level-1 data cache, 1GB Rambus memory, and a GeForce IV graphics-card. The graphics card capabilities are only used to display the final image. The system is able to force threads on specific physical and logical CPUs. The system can also be forced to run only on one physical CPU using both logical CPUs. Benchmarks were performed using several different data sets and transfer functions. Figure 2.1 shows the results of an exemplary test run using a CT scan of a human hand (244x124x257, 16 bit). Different transfer-functions were specified in order to achieve high work loads. The images (512x512) from left to right show renderings with progressively more translucent transfer function settings. Measured render timings are: (a) 0.46 sec, (b) 2.0 sec, (c) 4.6 sec. Non-translucent transfer functions lead to frame rates of up to 2.5 fps for this particular data set. All test runs consistently showed the same speedup factors.



# physical CPUs	Hyper-threading	Computational time	Speedup
One	Disabled	One thread	1.0
One	Enabled	Two threads 30% saving	1.42
Two	Disabled	Two threads 49% saving	1.96
Two	Enabled	Four threads 69% saving	2.78

Figure 2.13: Thread-level parallelism speedup.

The achieved thread-level parallelism speedup is shown in Table 2.13. Testing thread-level parallelism on only one CPU showed an average speedup of 30%. While changing the viewing direction, the speedup varies from 25% to 35%, due to different transfer patterns between the level-1 and the level-2 cache. Whether hyper-threading is enabled or disabled, adding a second CPU approximately reduces the computational time by 50%. This shows that the thread-level parallelism scheme scales very well on multi-processor machines. Moreover, the hyper-threading benefit of approximately 30% is maintained if the second hyper-threaded CPU is enabled.

Figure 2.14 shows the thread-level parallelism speedup according to different brick sizes. The speedup significantly decreases with larger brick sizes. Once the level-2 cache size is exceeded, the two threads have to request data from main memory. Therefore, the CPU execution units are less utilized. Very small brick sizes suffer from a different problem. The data fits almost into the level-1 cache. Consequently, one thread can utilize the execution units more efficiently, and the second thread is idle during this time. The overall disadvantage is the inefficient usage of the level-2 cache. The optimal speedup $1/((100 - 30)/100) \approx 1.42$ is achieved with 64KB (32x32x32). This is also the optimal brick size for the bricked volume memory layout, see Section 2.3.

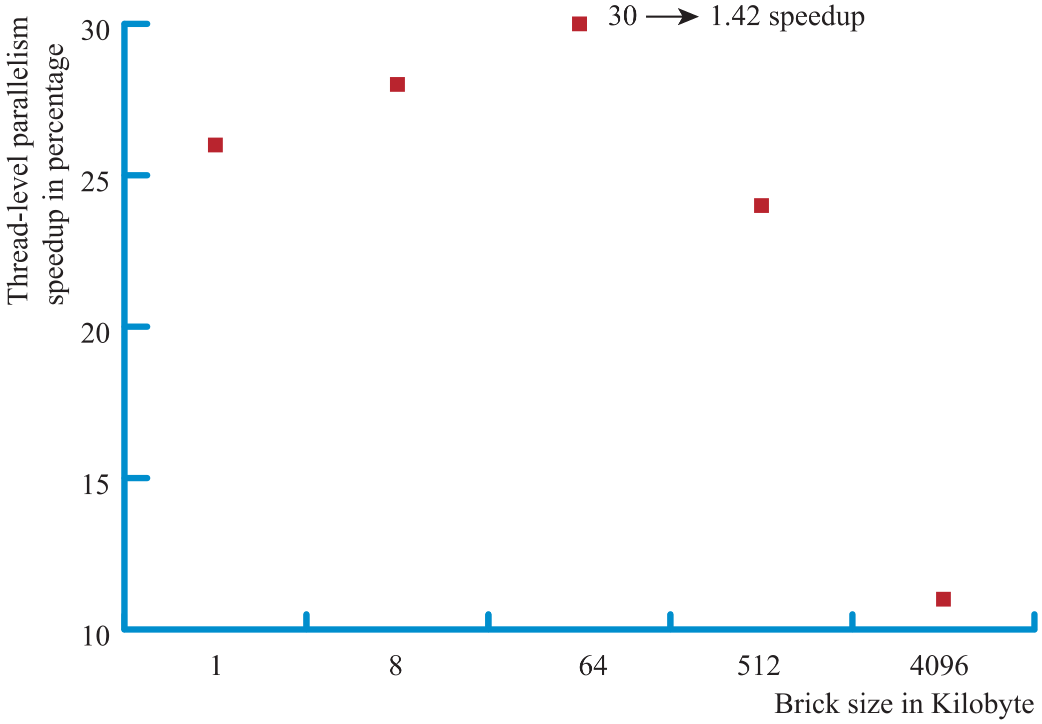


Figure 2.14: Thread-level parallelism speedup for different brick sizes.

2.5 Discussion

Efficient usage of hardware resources for basic graphics algorithms is very important. It is a known fact, that thread-level parallelism can increase performance by a factor of 30% for effectively parallelized algorithms. However, effectiveness is only achieved if threads operate on coherent data. Tests have shown that large brick-sizes lead to very low thread-level parallelism performance benefits. In this case there is a low cache hit rate, and, therefore, expensive main memory requests result in execution stalls. For instance, by just splitting the image plane in half and assigning each half to a hyperthread, result in a performance decrease instead of an increase; two threads are constantly requesting data from different memory locations. This leads to enormous cache trashing, since the two threads share caches.

The bricking speedup is about a factor of 2.8. However, it is important to note that this speedup factor characterizes the improvement in traversal, resampling and gradient computation. These are the components of the system which are directly affected by the accelerated memory access. Other parts, such as compositing and shading do not benefit from the presented optimizations. In the system, however, these parts only play a minor role in overall performance. It uses front-to-back compositing and Phong shading with two light sources.

Experiments showed that with the optimal brick-size of 32x32x32 a speedup factor of 2.8 is achieved. Enabling thread-level parallelism results in an additional speedup of 1.42. The combined speedup is 2.8*1.42 $\approx$ 4.0. High-level optimizations, such as empty space skipping or early ray termination, did not influence this speedup factor. The efficient addressing scheme considerably reduces the cost of addressing in a bricked volume layout. Its influence on the overall performance gain depends on the filter support size used for resampling and gradient estimation, as well as on the complexity of the remaining calculation, such as shading and compositing.

2.6 Conclusion

A very efficient raycasting system utilizing thread-level parallelism has been presented. A bricked volume layout has been utilized in order to design a highly efficient threading scheme that maximizes the benefits of thread-level parallelism. The high cache coherency inherently present in a bricked volume layout combined with the two refined addressing schemes significantly reduced the costs of resampling and gradient computation.

For the efficient usage of thread-level parallelism a multi-threading scheme has been introduced, such that two threads running on one physical CPU simultaneously process one data brick. Processing the same data brick simultaneously with both hyper-threads is essential for exploiting this technology. The results have proven that inefficient CPU utilization can be significantly reduced by taking advantage of hyper-threading technology. The realization of the system showed that using this new technology is not straightforward. Systems have to be adapted in order to take advantage of this architecture. Most of today's used multi-threaded systems have to be redesigned. By just starting more threads one can encounter a significant performance decrease instead of an increase, due to the fact that hyper-treads share caches.

A significant speedup has been achieved by using the new addressing method in a bricked volume layout. The new addressing scheme can be used for any volume processing algorithm, which has to address adjacent samples. The results showed that conditional branches have quite some performance impact, due to the growing length of the CPU pipeline. Advanced low-level optimizations lead to efficient CPU utilization, as well as to a significant speedup factor of 4.0.

Chapter 3

High-level Acceleration Techniques

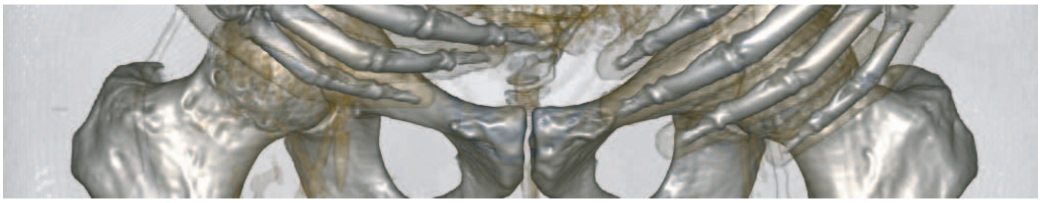


Figure 3.1: Close-up of the visible male.

This chapter is based on the following publications:

Grimm S., Bruckner S., Kanitsar A., Gröller E., **Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data**, *Proceedings of IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, pp. 1-8, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **A Refined Data Addressing and Processing Scheme to Accelerate Volume Raycasting**, *Computers & Graphics*, 28(5), pp. 719-729, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data**, Technical Report TR-186-2-03-11, *Vienna University of Technology*, 2003.

3.1 Introduction

Direct Volume Rendering is known as a powerful technique to visualize complex structures within three-dimensional data. Its main advantage, compared to standard 3D surface rendering, is the ability to perform translucent rendering in order to provide more information about spatial relationships of different structures. In general, 3D visualization helps to understand patient's pathological conditions, improves surgical planning, and is a big aid in medical education. A typical data size of today's clinical routine is up to 512x512x512. However, some examinations, such as peripheral CT angiography run-offs, require even larger scans. For example, Rubin et al. [67] reported a mean of 908 transverse reconstructions. Furthermore, due to improved capabilities of newer acquisition devices, it is possible to scan with even higher resolution. The high resolution is often used for difficult cases resulting in larger data sets. This large data presents a challenge to current rendering architectures and techniques. The increasing demand of interactive 3D visualization is basically driven by the size itself. Conventional slicing methods have already reached their limit of usability due to the enormous amount of slices. 3D visualization is more and more explored as an attractive alternative additional method for examinations of large medical data to support the obliged 2D examination. Figure 3.1 shows an example of a 3D visualization.

Within the research area of accelerating volume rendering, two main research streams can be identified. One stream is focused on exploiting special purpose hardware such as Volume Pro (Pfister et al. [58]), Vizard (Meissner et al. [46]) or graphics cards (GPU) (Cabral et al. [4], Westermann et al. [77], Guthe et al. [18] and many others). This approach usually provides high performance when data fits into internal memory. However, this issue becomes the most critical bottleneck once the data size exceeds the onboard internal memory capacity. Expensive transfers of data from main to internal memory have to be performed, which lead to an enormous performance penalty. Furthermore, the accelerated pace of the GPU's development cycle produces heterogenous multi-user hardware environments. This makes the adoption of such special purpose hardware solutions even more difficult. The other research stream is based on CPU technologies. In general, they provide better performance for large data due to the inherent larger memory capacity. Many proposed approaches for CPU based volume raycasting achieve high performance by utilizing super-computers or clusters; e.g., Parker et al. [57] presented a volume rendering approach on an SGI Reality Monster and were capable to render the Visible Woman (approx. 1 GB) with up to 20 fps utilizing 128 processors. However, it is a large scale solution which does not apply to the needs and capacities of an ordinary medical environment.

The purpose of this work is to present a solution which resolves the issues presented before: an interactive real-time volume rendering approach for large medical data, capable of performing in a heterogeneous hardware environment, by using commodity computers such as notebooks, and providing high performance and high quality images. This is achieved by introducing an efficient method for on-the-fly gradient estimation and an efficient hybrid removal and skipping technique of transparent regions. The presentation of the new approaches is subdivided as follows: Section 3.2 surveys related work. Section 3.3 presents a brief overview of the raycasting processing workflow. In Section 3.4 acceleration techniques such as a refined caching scheme for gradient estimation and a hybrid skipping and removal of transparent regions method to reduce the amount of data to be processed is introduced. In Section 3.5 the conclusion is presented.

3.2 Related Work

The most popular CPU-based direct volume rendering algorithms are shear-warp, splatting, and raycasting. Shear-warp is considered to be the fastest software algorithm (Lacroute et al. [31]); however, the inherent bi-linear interpolation provides quality which is, in general, insufficient for medical purposes. Splatting was first proposed by Westover et al. [78]; it was later improved in terms of quality and speed by Mueller et al. [51, 52], and Hung et al. [21]. This technique provides high quality images. However, it still lacks the speed provided by the general volume raycasting technique.

Volume raycasting is still widely used when high quality rendering of large data is desired. Several acceleration techniques for volume raycasting have been proposed over the last decade. Knittel et al. [26] and Mora et al. [48] proposed volume raycasting approaches for commodity computers. They achieve impressive frame-rates by using a spread memory layout and precomputed gradients; however, their method requires an enormous amount of additional memory. The spread memory layout itself increases the memory usage by a factor of four. This becomes a rather limitation factor if large data needs to be handled, or if the the rendering system is part of a larger visualization systems and memory resources need to be shared.

In contrast, the presented approach does not rely on extensive precomputing or a spread memory layout; it is based on a bricked volume layout. In order to achieve high performance advanced acceleration structures and techniques are necessary. In the following Sections several memory efficient acceleration approaches are presented.

3.3 Volume Raycasting Work-flow

The following paragraph presents a brief overview of the work-flow of the volume raycasting approach which is based on the memory layout and low-level acceleration techniques presented in Chapter 2. The volume data is decomposed in bricks and the processing is performed brick-wise. The volume raycasting process is subdivided into preprocessing, prerendering, rendering, and postrendering. The preprocessing step is done only once during start-up and the remaining steps are performed every time the image needs to be rerendered.

Preprocessing: During loading, the data is decomposed into small bricks of size 32^3. The data within the bricks and the bricks themselves are stored in common xyz-order. For each brick information about the contained density values is stored, e.g., min-max values, quantized binary histograms, etc.

Prerendering: In this phase transparent regions are removed and the rays-volume intersections are computed. There are eight different brick lists which are defined by the eight possible viewing-octants in 3D. Depending on the viewing direction the appropriate list is selected to process the volume brick-wise and in correct visibility order.

Rendering: According to the brick list, all rays traverse the bricks in visibility order, until all bricks are processed or all rays are terminated due to complete opacity accumulation. During traversing regular resampling, gradient computation, classification, shading and composition are performed.

Postrendering: At this point the final image is displayed, written to a file, or sent over the network to a client.

3.4 Acceleration Structures

There are two major strategies to accelerate volume raycasting. The first one is to reduce the computational cost at one resampling location. This is achieved by using an acceleration technique for gradient estimation. The second strategy is to efficiently remove and skip transparent regions, which is achieved by using quantized binary histograms, granular resolution octrees, and a cell invisibility cache.

3.4.1 Efficient Gradient Caching

The most common method to accelerate gradient estimation is to read precomputed gradients from memory. However, this acceleration technique has several drawbacks. In order to gain high performance the gradients must be stored in memory, resulting in an inefficient usage of resources. Furthermore such a solution is limited by memory bandwidth instead of preferably CPU throughput. The evolution of computer systems has shown that CPU performance increases faster than memory bandwidth. Going one step further if the data exceeds the main memory capacity, out-of-core rendering has to be performed and the gap between CPU throughput and memory bandwidth becomes even larger. Experience has shown that not every gradient estimation scheme performs equally well on all kinds of data. Therefore, the ability to switch between different gradient estimation schemes is an important feature and basically not efficiently given if precomputing is used. Additionally, often only gradient direction is stored and the gradient magnitude is omitted, otherwise the storage requirements can become considerably high. Finally, precomputing the gradients is quite time consuming. Considering a now-a-days medical visualization system, the doctor's main interest is to carry out the examination as fast as possible. The total time from scanning the patient to the actual examination is a highly critical factor.

To avoid these issues, the presented approach performs on-the-fly gradient estimation. In order to obtain highly accurate images, a dense object and image sample distance is inevitable, which implies high computational costs. A typical resampling resolution illustrated in 2D is shown in Figure 3.2a. In this case there are eight resample locations within a cell. Each gradient at the corners of one cell has to be computed eight times. Furthermore, each corner is shared between four cells in 2D. The total amount of redundant gradient computations at one corner is eight resampling positions multiplied by four cells which gives a total of 32 computations. In 3D the computational costs are even considerably higher. These very costly redundant gradient computations can be avoided by refined caching. However, not every gradient estimation scheme is suitable for caching. There are several studies on gradient filters for volume rendering with focus on accuracy, importance in terms of image quality and efficiency. Especially, Moeller et al. [47] give a thorough comparison of commonly used normal estimation schemes. They differentiate between four types of gradient estimation schemes:

1. *Continuous Derivative* uses a derivative filter which is preconvolved with the interpolation filter. The gradient at a resample location is then computed by convolving the volume with this combined filter.

2. *Analytic Derivative* uses a special gradient filter derived from the interpolation filter for gradient estimation.

3. *Interpolation First* computes the derivative at the resample position by resampling the data on a new grid, such that the used derivative operator can be applied directly. This is very beneficial if orthographic rendering is performed.

4. *Derivative First* computes the gradients at the grid-points and then interpolates these at the desired resample position.

For scheme one and two no caching mechanism is available. Only schemes three and four can be considered for efficient gradient caching. Due to their numerical equivalence only a comparison with respect to efficiency is necessary. Moeller et al. [47] proposed the *Interpolation First* method as the most efficient one. Considering volume rendering and no caching this is quite obvious. However, applying the *Interpolation First* scheme requires resampling of the original grid to a much larger grid if the object sample distance is significantly smaller than one. Already an object sample distance of 0.25 increases the grid size by a factor of four. This enormous amount of data makes caching inefficient and difficult. Especially if the object sample distance should be kept dynamical or if jittering techniques are applied to improve the image accuracy. Due to these reasons the *Derivative First* gradient estimation scheme is more efficient from a performance point of view, since it is more suitable for caching. In this case, the amount of data to cache is always determined. This makes interactive changes of the object sample distance possible.

An efficient per brick gradient caching approach is introduced; the caching scheme requires two data structures: the cache itself and a second structure to store the corresponding valid bits. The used processing entity is not the whole volume; in fact the volume is decomposed in bricks and each brick defines a processing entity. The size of the cache matches the number of gradients needed for one brick. The most straightforward way to use this cache would be to precompute all gradients which correspond to the current brick and use those during brick processing. This would be very inefficient, since more gradients than necessary would be precomputed if only parts of a brick are visible. In contrast to that additionally valid bits are used, which encode if a gradient is already computed and stored in the cache. During brick processing every time a gradient needs to be computed, it is checked if the gradient is already stored in the gradient cache. If not, the gradient is computed and stored in the cache and the corresponding valid bit is set to true. This mechanism ensures that gradients are computed only once at each sample position during brick processing. The cache remains only valid during the processing of one brick. Once the next brick is processed the cache is reset. This has the effect that the gradients which are also needed in adjacent bricks are processed more than once. The resulting performance penalty is low, since the number of those gradients is small compared to the number of all gradients.

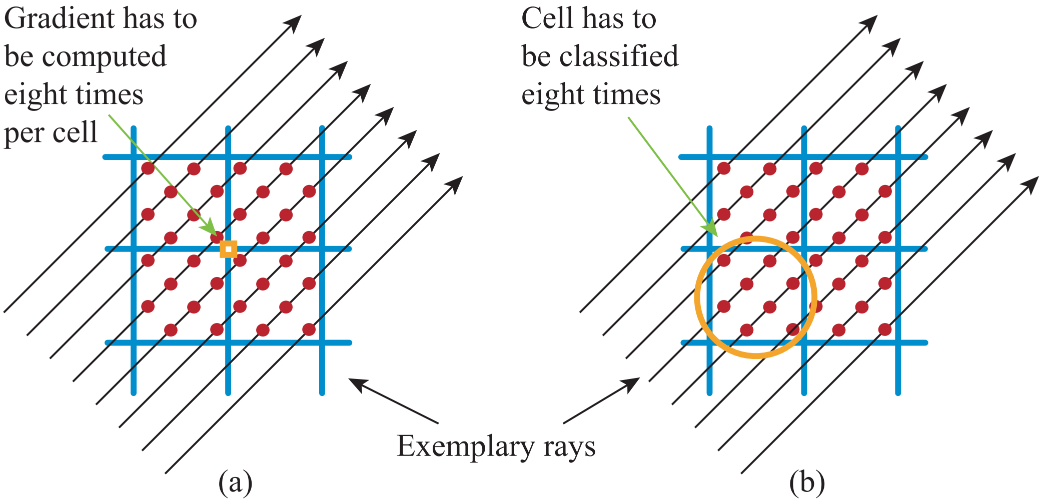

Gradient has to be computed eight times per cell
Cell has to be classified eight times
Exemplary rays
(a)
(b)

Figure 3.2: Typical resampling resolution of a cell in 2D. (a) In the shown case each gradient at the cell corners has to be computed 8 times while processing one cell. (b) In the shown case a cell has to be classified 8 times.

3.4.2 Results

The memory consumption of the gradient cache is very low. The cache size is not related to the volume dimensions. It is related to the brick dimensions. The brick dimension is 32x32x32, the size of the gradient cache is (*dimension of brick*+1)^3 multiplied by *dimension of gradient* multiplied by *size of gradient component*, which is $(33)^3 \cdot 3 \cdot 4 \approx 421{,}14$ KByte. Additionally for each cache entry a valid bit is stored, which adds up to 33^3 Bit $\approx$ 4.39 KByte. This is altogether less than 512 KB. For performance reasons the data shall remain in the level 2 cache. This is not an issue as current commodity CPUs have a level 2 cache size of 1 MB.

Figure 3.3 shows the effect of per brick gradient caching compared to per cell gradient caching and no gradient caching at all. Per cell gradient

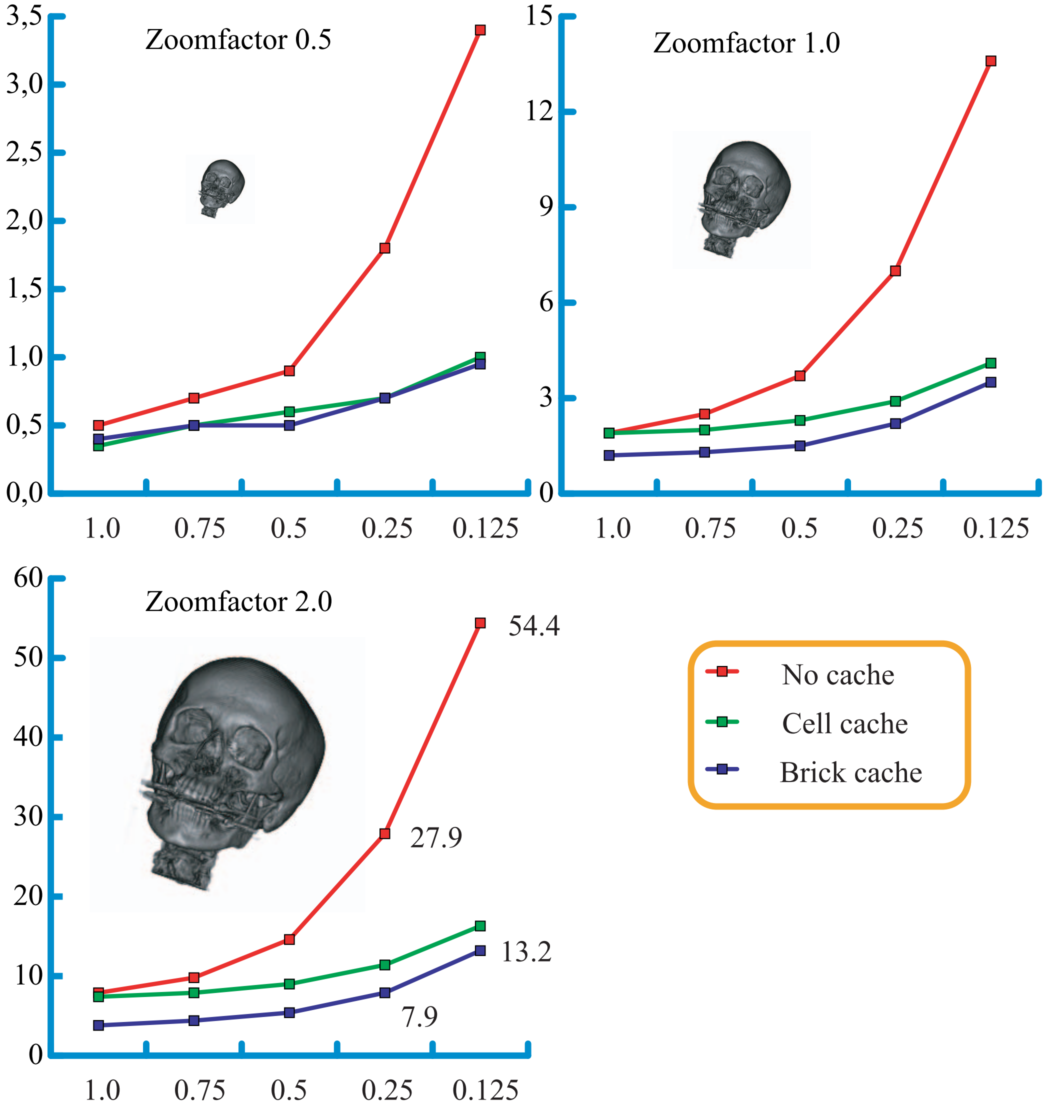


Figure 3.3: Comparison between no gradient caching, per cell gradient caching, and per brick gradient caching. Timings are given in seconds. The tested object sample distances are 0.125, 0.25, 0.5, 0.75, and 1.0. Data: UNC head, 256x256x224, 12 bit. Intensity range [0,1136] is mapped to 0.0 opacity and range [1136,4095] to a linear opacity ramp between 0.0 and 1.0. System specification: CPU - Intel®Pentium®M 1.6 GHz, Cache - 1 MB Level2, RAM - 1 GB, GPU - GForce4 4200 Go (32MB).

caching means that gradients are cached while a ray resamples a cell. For gradient estimation the gradient filter proposed by Neumann et al. [54] is used, see also Chapter 1. This filter produces slightly better quality than the Sobel filter, supports inherent volume filtering and has approximately the

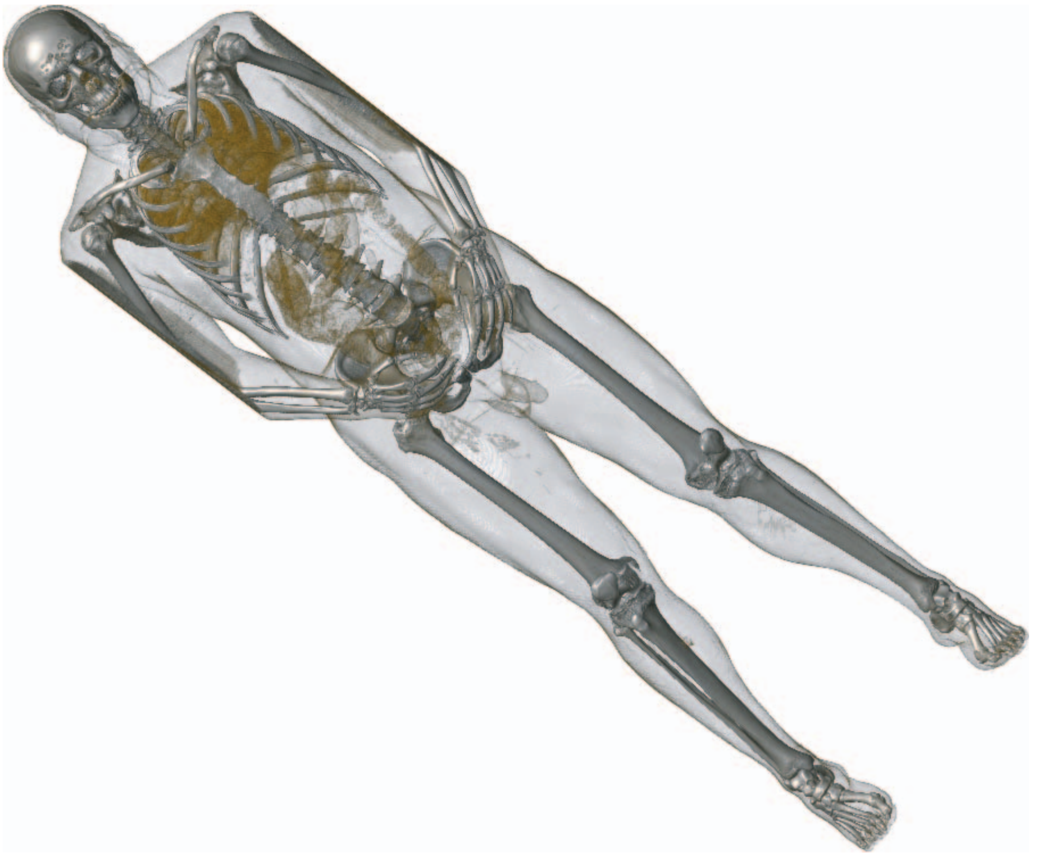


Figure 3.4: Data: Visible Male (587x341x1878). System: (Notebook) Intel®Pentium®M 1.6 GHz. Image size: 1024x768. Object sample distance: 0.25. Render timings: no gradient caching → 21.1 sec, with full gradient caching → 9.6 sec.

same computational cost. Due to the on-the-fly computations, the filtering can be enabled and disabled interactively. The on-the-fly filtering has low computational cost and can be used to increase the quality, when a smaller number of rays are shot to increase the frame-rate during interaction.

For testing an adequate opacity transfer function is chosen to enforce translucent rendering. The charts in Figure 3.3 show different timings for object sample distances from 1.0 to 0.125 for three different zooming factors 0.5, 1.0, and 2.0. In case of zooming factor 1.0 there is one ray per cell, already here per brick gradient caching performs better than per cell gradient caching. This is due to the shared gradients between cells. For a zooming out factor of 0.5 both gradient caching schemes perform equally well. The rays are so far apart that nearly no gradients can be shared. On the other hand for zooming in (factor 2.0), per brick caching performs much better than per cell caching. This is due to the increased number of rays per cell. As more rays process the same cell, the more beneficial the per brick caching becomes. Per brick gradient caching compared to no caching shows already with a zoom factor of 2.0 and an object sample distance of 0.5 an impressive speedup of approximately 3.0. The speedup favorably scales with the zoom factor. Figure 3.4 shows an example rendering of the Visible Male with a high proportion of transparency. The caching scheme compared to no caching shows a speedup factor of ≈ 2.2.

3.4.3 Empty Space Skipping

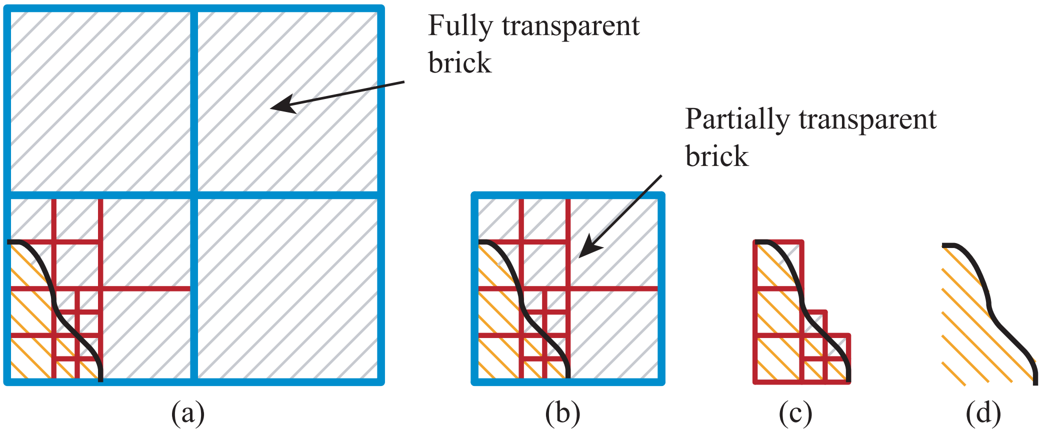


Figure 3.5: General work-flow of the hybrid transparent region removal and skipping technique: Blue are brick boundaries, red are octree boundaries, grey are transparent regions and yellow is visible volume data. (a) → (b): removal of transparent bricks, (b) → (c) removal based on octree projection and (c) → (d) removal using the cell invisibility cache.

For medical imaging, interactive classification of data is mandatory. In general, during examination large parts of the data are often classified as transparent to allow a more precise view of the region of interest. For acceleration purposes it is quite beneficial to exploit this transparency information and start the actual resampling of the data right where the visible data begins. The work-flow of the hybrid transparent region removal and skipping technique is shown in Figure 3.5. At first transparent regions are removed on a brick basis (Figure 3.5a → Figure 3.5b). Then to support an even more refined removal of smaller transparent regions octree projection is performed (Figure 3.5b → Figure 3.5c). Due to efficiency reasons the octree subdivision does not fully go down to individual cells. The granular resolution of the octree leads to conservative rays-volume intersections. To overcome the resulting performance penalty a Cell Invisibility Cache (CIC) is introduced to skip the remaining transparent cells (Figure 3.5c → Figure 3.5d). In the following the hybrid transparent region removal and skipping technique is described in more detail.

Quantized Binary Histograms

At first, an efficient encoding for finding transparent bricks is described. The most common methods are minimum-maximum encodings and summed area tables. A summed area table encodes the opacity transfer function by

$$\begin{aligned} &S(0) = \alpha(0) \\ &S(k) = S(k-1) + \alpha(k) \end{aligned}$$

Hereby $k \in H = [0..4095]$, which is the possible range of Houndsfield units and α represents the opacity. i_{min} and i_{max} denote the minimum and maximum density value within a brick. The integral of the discrete function α over the interval $[i_{min}, i_{max}]$ can be approximated in constant time by performing two table lookups:

$$\sum_{k=i_{min}}^{i_{max}} \alpha(k) = S(i_{max}) - S(i_{min})$$

If $S(i_{max}) - S(i_{min}) = 0$ then the brick is transparent and can be skipped. At this point pre- and postclassification is distinguished. For postclassification the min-max encoding is the most accurate, since due to interpolation of data all values between the minimum and the maximum may occur. However, if preclassification is performed the min-max encoding may be too granular when applied on large regions. Figure 3.6 shows an example where the min-max encoding is too conservative. The min-max encoding would report both bricks as being visible. The main issue is that the min-max encoding accuracy relies heavily on the underlying data. If the region is large it is quite likely that its values differ considerably. The min-max encoding becomes too granular to effectively encode the area. A more refined structure, i.e., a quantized binary histogram is used. In general, a binary histogram is encoded as:

$$\sigma_x(B) = \begin{cases} 1, & x \in B \\ 0, & otherwise \end{cases}$$

Hereby B is the set of all density values a brick contains, with $B \subseteq H = [0..4095]$. $\sigma_x(B) = 1$ means that the density value x is given at least at one grid position in the underlying brick. This encoding is effective, however, it is quite inefficient in terms of memory usage and efficient evaluation. Additionally to the binary codomain quantization also the domain itself is quantized. This *quantized binary histogram* is stored for each brick.

It is determined by

$$\sigma_i(B) = \begin{cases} 1, & \exists x : x \in B, x \in [128 \cdot i..128 \cdot (i+1)[\\ 0, & otherwise \end{cases}$$

Where $(0 \leq i \leq 31)$. Within quantized binary histograms the existence of data within a specific interval is encoded. The intervals are concatenated, disjunct, have same length, and cover the range of Houndsfield units. In the preprocessing phase every brick is parsed and encoded. The same encoding can be performed for the transfer-function with respect to opacity:

$$\lambda_i = \begin{cases} 1, & \exists x : opacity(x) \neq 0, x \in [128 \cdot i..128 \cdot (i+1)[\\ 0, & otherwise \end{cases}$$

Hereby $x \in H$ and $i \in [0, 31]$. Every time the transfer-function changes, the transfer-function is re-encoded in this way.

With this information one can quickly determine the transparent bricks. A brick is transparent if

$$\forall i \in [0..31] : \lambda_i \wedge \sigma_i = 0$$

This conjunction test can be done very efficiently on an x86 based CPU. Note that this is a conservative estimate of a brick's visibility. It is possible that due to the chosen encoding a brick is considered as visible although all contained values are classified as transparent. However, looking at Figure 3.6, it can be seen that the quantized binary histogram would report the bricks correctly if preclassification is performed. This is due to the fact that the quantized binary histogram is more sensitive for largely varying data values. This property can be efficiently exploited if the binary histogram encodes a segmentation information volume. In such a volume, segmented objects are encoded by labels. These labels can differ largely and interpolation is not applicable. Only preclassification can be performed in this case

Granular Resolution Octrees

With the method described in Section 3.4.3 entirely transparent bricks are determined and unnecessary processing is avoided. Also avoiding processing of transparent regions within a brick is desired. Therefore, each brick contains a granular resolution octree to enable the determination of transparent regions within a single brick. A min-max octree is one of the best known

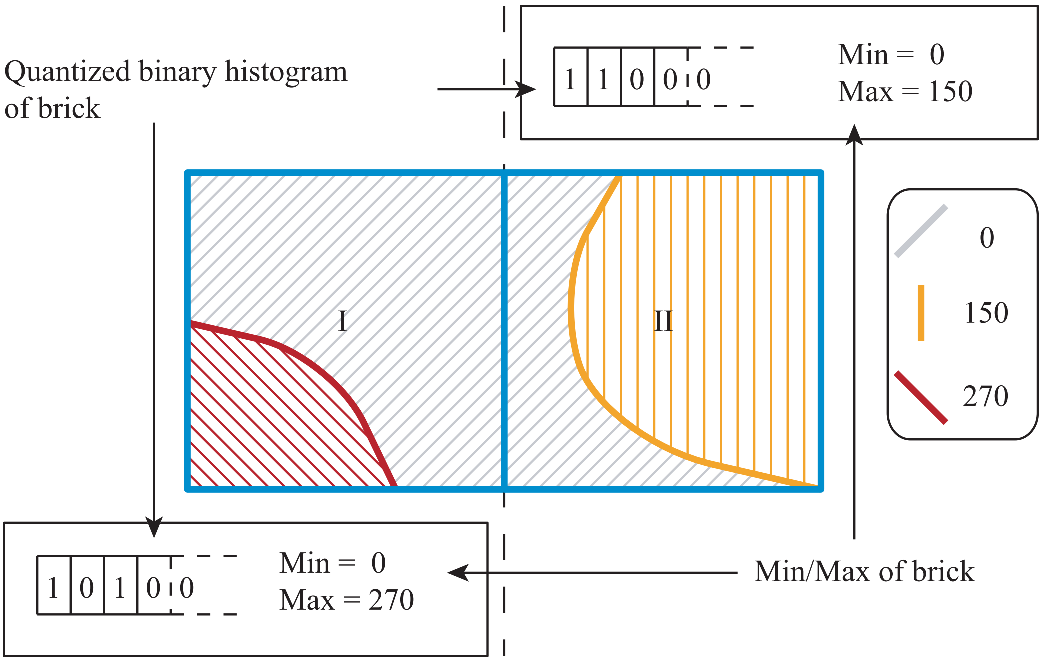


Figure 3.6: Min-Max encoding granularity issue if preclassification is performed: If the range of non-transparent values is set from 130 to 150, the min-max encoding would report both bricks as being opaque. The quantized binary histograms would report brick I being transparent and brick II being opaque.

space subdivision structures to support refined skipping of small transparent areas (Lacroute [31], Wilhelms et al. [80], and Mora et al.[48]). Each brick (32x32x32) contains a 3-level min-max octree, shown in Figure 3.7a. For each octree level the minimum and maximum value is stored as a pair of numbers. For level 0 there are eight pairs, level 1 needs eight by eight = 64 pairs, and level 2 needs eight by eight by eight = 512 pairs. When classification changes the octree is recursively evaluated by a summed area table for all bricks. The classification information is efficiently stored by hierarchical compression [26]. Nodes of level 2 are either opaque or transparent. All other nodes have an additional inhomogeneous state. The information whether a node of level 2 is transparent or opaque is stored in one bit. The state of a level 1 node is determined by testing of one byte, which contains all the bits of its children. For level 0 such a hierarchical compression requires to test eight bytes for a node and 64 bytes for the brick. Due to efficiency reasons the state information of level 0 is explicitly stored. There are three possible states; thus, two bits are needed for each level 0 node. Thus, additionally two bytes per brick are required. Due to this encoding, the octree can be very efficiently traversed.

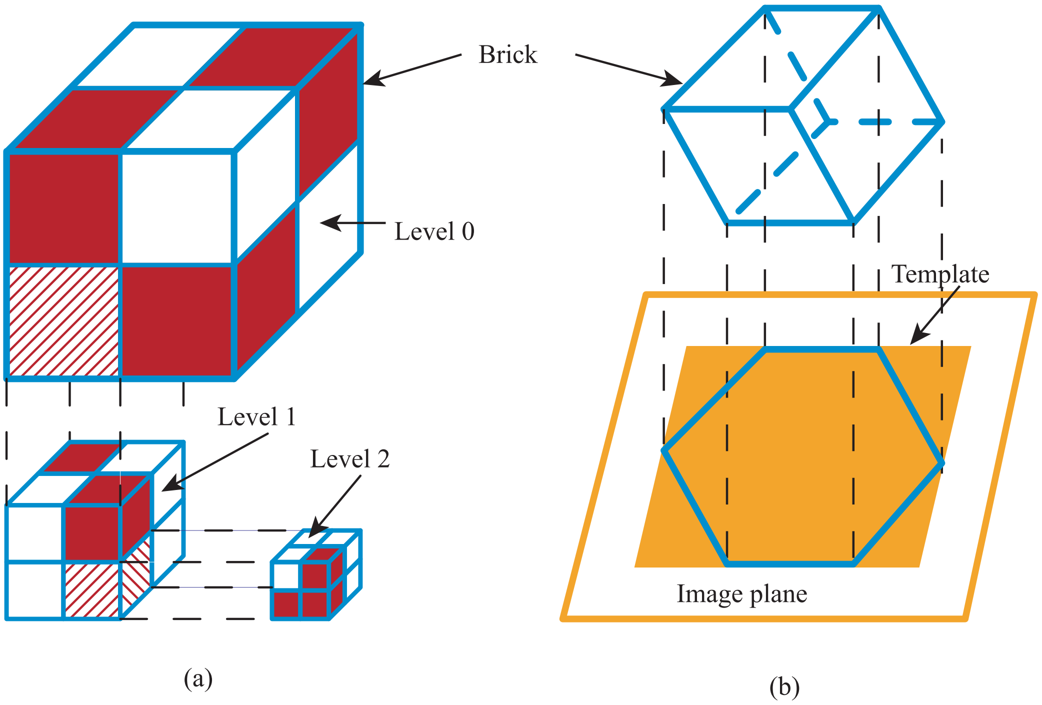


Figure 3.7: (a) Octree classification scheme for an individual brick. Transparent bricks are white, opaque bricks are red and inhomogeneous bricks, partially transparent and partially opaque, are striped. (b) Brick projection template.

Removing of Transparent Regions

There are two structures, a quantized binary histogram and a granular octree, to find the rays-volume intersections up to the resolution of the granular octree (Figure 3.5c). The bricked geometry of the volume and the octrees within the bricks are converted to a polygonal structure and rendered into a z-buffer [68]. Basically the brick list is traversed and it is determined which brick has visible data and needs to be evaluated for rays-volume intersection [18]. The evaluation is performed by quantized binary histograms in case of preclassification or min-max encodings in case of postclassification. The octree of those bricks is evaluated and the sub-bricks which contain visible data are rendered. This rays-volume intersection computation by rendering requires the *granular* resolution of the octree. With more than three octree levels the number of polygons would exceed the rendering performance of commodity graphics hardware.

Utilizing OpenGL for rendering provides high performance and high accuracy; however, if the approach is used as an integrated module it requires off-screen rendering. This is available in OpenGL by PBuffers. Unfortunately, this feature is not available on every graphics card. Furthermore the rendering requires a huge amount of graphics cards memory. Considering a 1024x1024 image, the needed buffer is already 8 MB. Most of the more advanced medical visualization systems support high-resolution dual-displays. This feature normally utilizes all the available graphics card memory. There is no space left for graphics hardware accelerated off-screen rendering. Due to this reason, the rendering is also developed in software. This can be done very efficiently, if the simple polygonal structure of the bricked octree layout is exploited. Since every brick is of the same structure, one can use template based projection of the octree. Similar work has be done by Srinivasan et al. [69]. The main idea is to project just one brick per viewing direction for each octree level as shown in Figure 3.7b. This projection is used as a template for all other bricks of the same level. Any other brick of the same level has the same projection footprint and is obtained by translation. The projected footprint consists of z-values, since the z-buffer footprint of the octree is of interest. All possible entry bricks are rendered in a front-to-back order by using the projected z-value template. The resulting z-buffer footprint of the octree is then used to determine the rays-volume intersections. This is as fast as the OpenGL implementation, since the costly projection itself has to be done only for one brick per viewing direction. Furthermore no costly OpenGL glReadPixels() instruction is involved and the resulting z-buffer directly contains the z-components of the ray starting-positions.

Cell Invisibility Cache

As the granular resolution octree does not go down to cell level, a cell invisibility cache is used to skip the remaining transparent cells ((Figure 3.5c → Figure 3.5d)). The volume-rays intersections estimation by template-based projection of the octree subbricks brings us as close as 4x4x4 samples to the visible data. This is inefficient from a performance point of view. Especially if first-hit-raycasting is performed every non skipped sample has a large impact on the resulting frame-rate. A resolution of 4x4x4 results in a large number of non skipped samples. This is depicted by the red samples shown in Figure 3.8. All these samples have to be classified in order to determine which cell can be skipped. Depending on the object-sample distance and the zoom factor these cells have to be classified several times. This is shown for a typical resampling resolution in Figure 3.2b. In this case each cell has to be classified eight times. Considering the same example in 3D, the number of

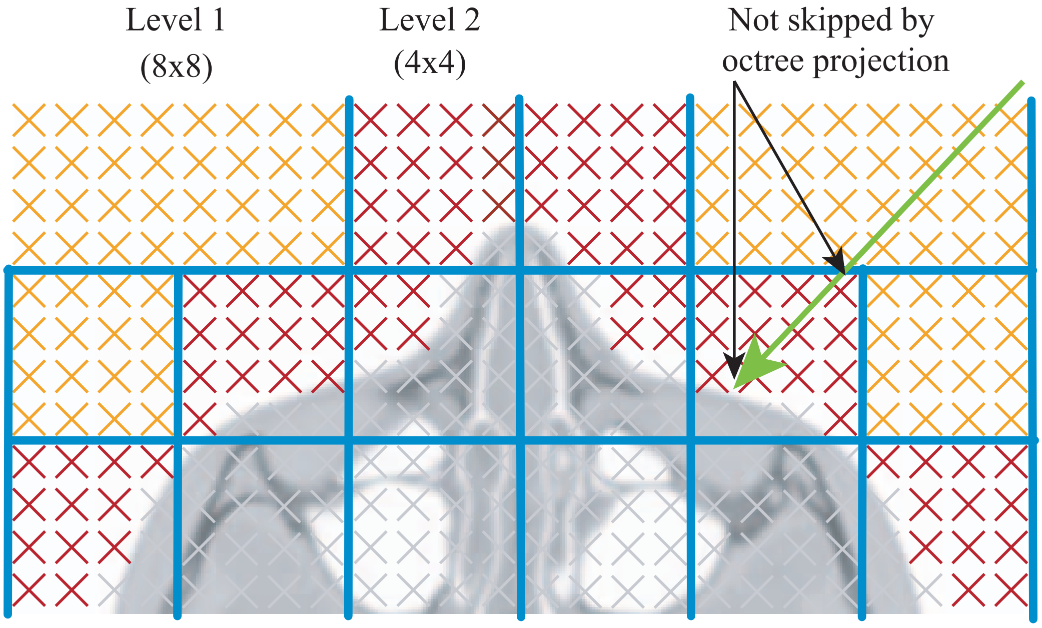


Figure 3.8: Zoom in granular octree of one brick. Yellow crosses: skipped samples, white crosses: opaque samples, and red crosses are samples that can not be skipped due to the granular resolution of the octree.

redundant cell classifications would be considerably larger. Due to this reason a refined cell invisibility caching is introduced. The volume raycasting pipeline is extended in such a way that classification of these invisible cells has to be done only once. The extended pipeline is shown in Figure 3.9. A Cell Invisibility Cache (CIC) is attached at the beginning of the traditional volume raycasting pipeline. This CIC is initialized in such a way that it reports every cell as visible. In other words every cell has to be classified. Now, if a ray is send down the pipeline, every time a cell is classified as invisible (all its samples have zero opacity contribution) this information is cached in the CIC. A cell can either be invisible or visible, this information can be encoded in just one bit. Once a cell is classified as invisible, the costly classification of a whole cell is exchanged by a binary test. This leads to an enormous performance increase. On the one hand, this is due to the reduced memory access and on the other hand due to the inherent classification and conjunction information of 8 samples. The information stored in the CIC remains valid as long no transfer-function change is performed. The CIC is stored per brick and, therefore, allows interactive changes of the transfer function. If the transfer function changes only the CICs of the bricks which are affected need to be reset. During the examination of the data, e.g., by changing the viewing direction, the CIC fills up and the performance increases progressively. The same mechanism is also very beneficial for general empty space skipping within the data.

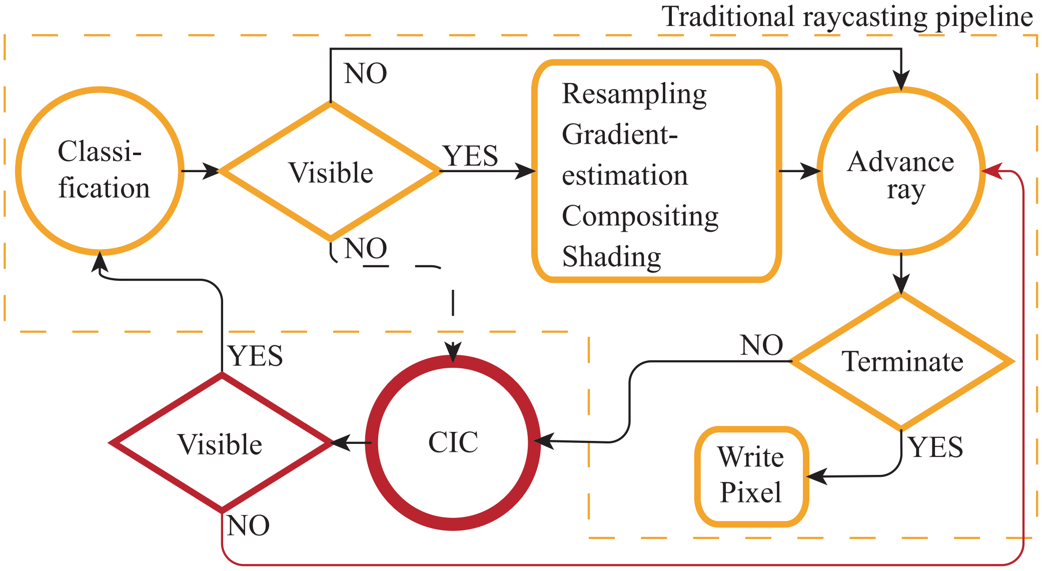


Figure 3.9: Cell Invisibility Cache (CIC) - Acceleration by caching invisibility information of cells. The acceleration path is emphasized in red.

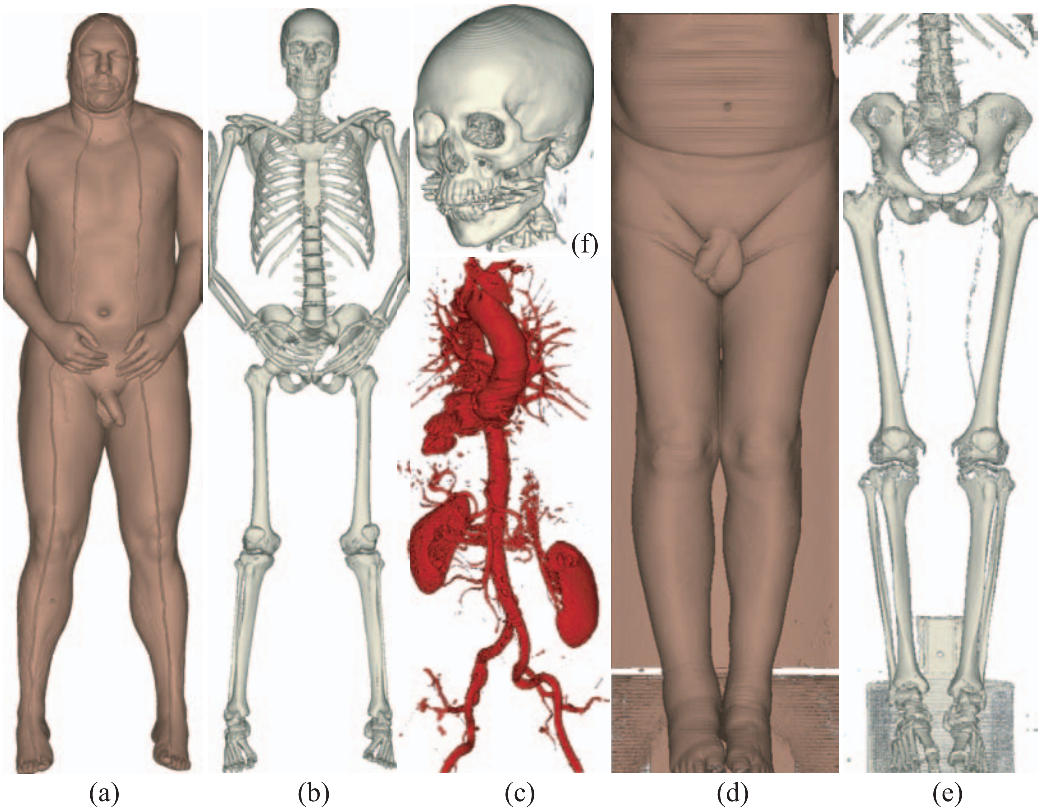
3.4.4 Results

The additional memory usage of all three acceleration structures, i.e., quantized binary histogram, granular resolution octree, and the cell invisibility cache, is rather low. Considering the size of the volume as 100%, they increase the size by approximately 10%. Bricks of size 32x32x32 are used storing 2 bytes for each sample, which is a total of 65536 bytes. Additionally for each brick is stored: Quantized binary histogram → 4 byte, Min-max information → (512+64+8+1) * 4 = 2340 byte, Octree classification information → (64 + 2) = 66 byte, and Cell Invisibility Cache → 32^3 bit /8 = 4096 byte. In total the storage increase is $((4 + 2340 + 66 + 4096)/65536) \cdot 100 \approx 9.9\%$.

Figure 3.10 shows the effect of the hybrid removal and skipping technique of transparent regions and shows the corresponding rendering output. For benchmarking a commodity notebook is used equipped with an Intel®Pentium®M 1.6 GHz CPU, 1 MB Level2 cache, 1 GB RAM, and a GForce4 4200 Go (32MB). The graphics card capabilities are only used to display the final image. Different data sets are tested: A rather small data set, the UNC head is used to be able to compare the speed of the presented approach to the approach of Mora et al. [48]. This approach is slightly faster than the UltraVis system [26]. They are both based on a spread memory layout and use precomputed gradients. This leads to an inefficient memory usage and so they are restricted to rather small data. Mora's total render time is approximately a factor of two faster than the presented approach. However, Mora's approach uses precomputed gradients, does preshading, and its template based interpolation scheme limits the zooming to a zooming-factor of four. In contrast to that some performance is sacrificed for increased flexibility, high quality, and a significantly lower memory usage. This enables us to render large data, used in clinical routine, on commodity hardware. Three different large typical medical data sets are tested. The results show that the acceleration techniques typically achieve render-times of about 2 fps even for these large data sets. Figure 3.10, fourth column, shows the total render time achieved by brick based transparent region removal. In the fifth column additionally the granular octree projection is applied. And finally in the sixth column the Cell Invisibilty Cache is enabled to see the overall total render time achieved by the combined effect of all three acceleration structures.

3.5 Discussion and Conclusion

A volume raycasting approach which provides high-quality images in real-time for large data on standard commodity computers without advanced graphics hardware has been presented. For large medical data such as computed tomographic (CT) angiography run-offs (512x512x1202) rendering times up to 2.5 fps on a commodity notebook have been achieved. This shows that real-time rendering of such large data on commodity notebooks is within reach. The method can be straightforwardly adapted to other modalities such as MR. Furthermore, the approach can utilize symmetric multi-processing systems as processing is performed brick-wise. It scales well and achieves a speedup factor of approximately 2.0 on a dual CPU machine. This is very beneficial if a large amount of data has to be processed. Costly precomputing is avoided and although each part of the volume raycasting pipeline is computed on-the-fly, performance in the same range as approaches which heavily rely on the memory bandwidth such as Mora et al. [48] has been achieved. The refined caching scheme for gradient estimation in conjunction with hybrid skipping and removal of transparent regions enables us to achieve high quality while maintaining high performance. The efficient memory consumption of the acceleration structures (quantized binary histogram + granular



Name	Dimension	Size	BR	BR + OR	BR+OR+CIC
(a) Visible Male	587x341x1878	0.70 GB	0.61 sec	0.46 sec	0.40 sec / 2.5 fps
(b) Visible Male	587x341x1878	0.70 GB	0.68 sec	0.53 sec	0.45 sec / 2.2 fps
(c) Run-off	512x512x1112	0.54 GB	1.16 sec	0.93 sec	0.61 sec / 1.6 fps
(d) CTA run-off	512x512x1202	0.59 GB	0.86 sec	0.70 sec	0.64 sec / 1.6 fps
(e) CTA run-off	512x512x1202	0.59 GB	0.69 sec	0.46 sec	0.37 sec / 2.7 fps
(f) UNC head	256x256x256	0.03 GB	0.71 sec	0.26 sec	0.18 sec / 5.6 fps

Figure 3.10: Performance results for different data sizes, which are used in daily clinical routine. Image size: 512x512, Sample rate: 0.5, and Hardware: CPU - Intel®Pentium®M 1.6 GHz, Cache - 1 MB Level2, RAM - 1 GB, GPU - GForce4 4200 Go (32MB). BR: brick based transparent region removal. OR: octree projection based transparent region removal. CIC: cell based transparent region skipping.

resolution octree + Cell Invisibility Cache) and the bricked volume layout allow to handle very large data. All acceleration structures require only an extra storage of approximately 10%. Data sizes up to 3 GB are possible, which is a limitation imposed by the virtual address space of current consumer operating systems.

Chapter 4

Rendering of Multiple Volumes

Figure 4.1: Multi-volume rendering of several V-Objects.

This chapter is based on the following publications:

Grimm S., Bruckner S., Kanitsar A., Gröller E., **Flexible Direct Multi-Volume Rendering in Dynamic Scenes**, *Proceedings of Vision, Modeling, and Visualization*, pp. 379-386, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data**, *Proceedings of IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics*, pp. 1-8, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **A Refined Data Addressing and Processing Scheme to Accelerate Volume Raycasting**, *Computers & Graphics*, 28(5), pp. 719-729, 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **V-Objects: Flexible Direct Multi-Volume Rendering in Interactive Scenes**, Technical Report TR-186-2-04-06, *Vienna University of Technology*, 2003.

4.1 Introduction

Direct volume rendering is an important and flexible technique for visualizing 3D data. It allows the generation of high quality images without a need of an intermediate interpretation. Traditionally, medical volume visualization systems feature only simple scenes consisting of a single volumetric data set. It has been proposed to extend these scenes to a more complex description [53]. In this work a flexible data structure called V-Objects is introduced for representing scenes containing multiple volumetric objects. An efficient approach to render a scene composed of V-Objects is presented. Furthermore, by presenting practical examples for possible applications it is demonstrated that medical visualization systems can take advantage of V-Objects. The main contributions are an efficient technique to render V-Objects and to show that common medical volume visualization systems can greatly extend their flexibility by supporting concurrent display of multiple volumetric objects.

The presentation is subdivided as follows: Section 4.2 surveys related work. In Section 4.3 the new data-structure V-Objects and in Section 4.4 an approach to efficiently render a scene composed of multiple V-Objects is presented. In Section 4.5 the results are presented. In Section 4.6 possible medical applications are presented. Finally, in Section 4.7 ideas for future work are given and the work is concluded.

4.2 Related Work

The past decade has seen significant progress in volume visualization, driven by applications such as medical imaging. A number of different volume rendering algorithms have been developed, improved, and extended [35, 32, 51]. Today, it is possible to perform interactive high-quality volume rendering on commodity hardware [65, 14, 13]. Hybrid algorithms have been designed, which allow concurrent display of intersecting volumetric and polygonal objects [36, 28]. Direct rendering of scenes consisting of multiple volumetric objects, however, has received less attention. With the increasing performance of modern hardware, this topic will become more important in the future. This work is inspired by Leu and Chen, who introduced a two-level hierarchy for complex scenes of non-intersecting volumes [34]. While their approach allows multi-volume rendering, the individual volumes cannot intersect. However, the display of intersecting semitransparent objects can be a powerful visualization technique. In the work of Nadeau [53] intersecting volumes are possible, however, the whole scene description has to be resampled every time a volume's transformation changes. The idea behind the presented method, is to allow multiple intersecting volumetric objects to be rendered directly, without requiring costly resampling. Thus, the approach is well-suited for animations.

4.3 V-Objects

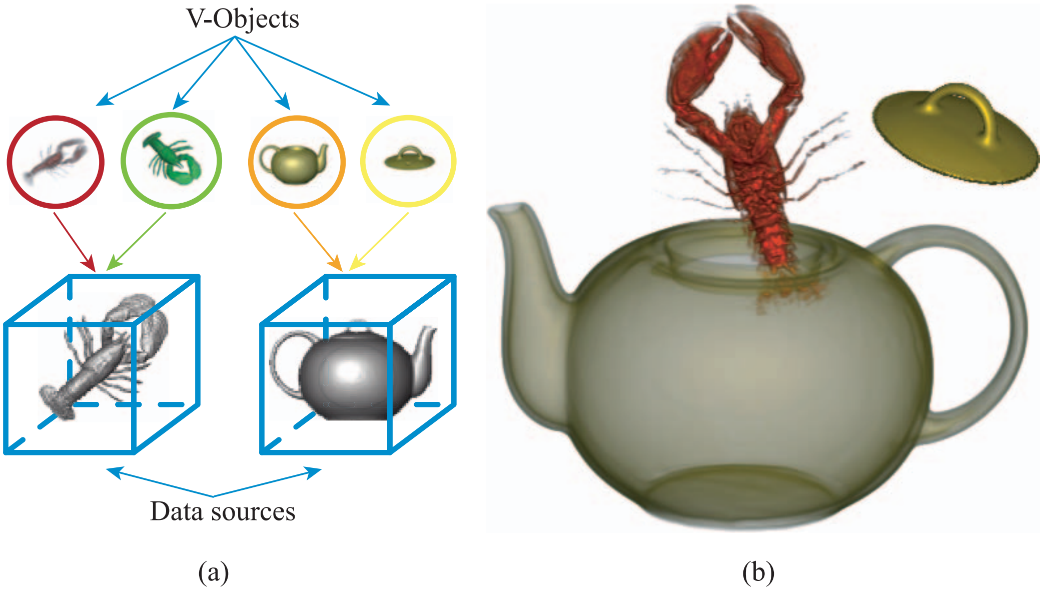


Figure 4.2: Different representations of the same data source using V-Objects: (a) V-Object definition. (b) Rendering of multiple V-Objects.

A V-Object is an element of a scene description which is connected to one volumetric data source. The V-Object comprises the following visual properties:

- *Illumination:* This includes the selected Illumination model and its parameters. For example, for the Phong-Blinn Illumination model the ambient, diffuse, specular, and emissive material coefficients are specified.

- *Transfer Functions:* For each defined region in the attached volumetric data source a mapping function between scalar values and colors as well as opacities is stored.
- *Region of Interest:* An arbitrary number of planes define a convex region of interest.
- *Transformation:* An affine transformation defining position, orientation, and scaling of the V-Object.

The separation of visual properties and volumetric data sources allows an arbitrary number of varying representations of the same data source within one scene, as illustrated in Figure 4.2(a). This is achieved by assigning several V-Objects to the same volumetric data source. To take full advantage of the V-Objects direct volume rendering is applied to simultaneously visualize a scene consisting of several V-Objects, see Figure 4.2b and Figure 4.1. The properties of a V-Object influence the positions in a scene at which it is defined. For example, regions of the volume which are classified as transparent due to the transfer function specification are not part of a V-Object. In Chapter 2 it has been shown how mono-volume raycasting can be greatly accelerated by a bricked volume layout and an appropriate brick-wise processing scheme. This performance benefit is exploited in Section 4.4 to accelerate multi-volume raycasting.

4.4 Rendering of V-Objects

Before describing the approach to accelerate multi-volume rendering, it is briefly discussed how compositing of multiple volumes is performed [5]. The basic idea of volume raycasting is to cast for each pixel of the image plane a ray through the volume. For a single object the final color and opacity of the image pixel is determined by the over-operator [61] in front-to-back order. That is, at each resample location, the current color and alpha values for a ray are computed in the following way:

$$\begin{aligned} c_{out} &= c_{in} + c(x)\alpha(x)(1-\alpha_{in}) \\ \alpha_{out} &= \alpha_{in} + \alpha(x)(1-\alpha_{in}) \end{aligned} \tag{4.1}$$

c_{in} and α_{in} are the color and opacity the ray has accumulated so far. x is the reconstructed function value and $c(x)$ and $\alpha(x)$ are the classified and shaded color and opacity for this value.

For the simultaneous processing of multiple volumes it must be decided how they should be combined. Volumes are considered as clouds of particles and all volume are simultaneously taken into account at the corresponding sample position. The simultaneously compositing of multiple volumes is achieved by sequentially applying Equation 4.1 for each individual volume. Hereby, an approximation for compositing multiple volumes at the same location is obtained.

As described in Chapter 2, the key to high performance for mono-volume rendering is to optimize the memory access pattern. This can be achieved by using a bricked volume layout. However, to find such a suitable memory layout for multi-volume rendering is very difficult, due to the unpredictable memory access pattern. Every time multiple volumes have to be simultaneously processed, several data entities from memory regions far apart have to be accessed. This leads to enormous cache trashing. To keep this cache trashing penalty as low as possible, multi-volume rendering is separated from mono-volume rendering within a scene composed of multiple volumes. While the processing of the intersection between multiple volumes requires costly computations, the non-intersecting regions could be efficiently computed by using a mono-volume rendering technique. In the following this issue is addressed and a solution is presented to efficiently decompose the scene in mono- and multi-volume rendering regions.

It is distinguished between the concepts of a mono- and a multi-volume renderer. Both types of renderers are initially supplied with a list of rays. Each entry in the ray data structure contains, among other data, a start and an end depth. A mono-volume renderer processes one V-Object efficiently by using the cache coherent volume traversal, based on a bricked memory layout as presented in Chapter 2. A multi-volume renderer performs compositing in multiple V-Objects. As the V-Objects can have different volumetric data sources and transformations, efficient brick-wise traversal is, in general, not possible. Thus, the multi-volume renderer performs classical ray traversal. As resample positions along a ray are likely to lie within totally different memory locations for different V-Objects, lacking cache coherence results in considerable performance penalties. In the approach, a mono-volume renderer is set up for every V-Object in the scene, while there is only one global multi-volume renderer.

The main idea is to first identify different segments along a ray's path in the initialization phase. There are two basic types of segments:

- *Mono-object segment:* A mono-object segment is a continuous interval along a ray lying within the same V-Object.

- *Multi-object segment:* A multi-object segment is a continuous interval along a ray lying within one or more V-Objects. Thus, each mono-object segment is also a multi-object segment.

The goal is to maximize the combined length of mono-object segments, as mono-object segments can be processed more efficiently. In general, the intersections between a ray and all V-Objects need to be known. A conservative approach is applied by using the octree projection explained in the previous Chapter 3. Theoretically using bounding box projections would also be possible, however, they do not provide sufficient resolution. An example of these projections of multiple V-Objects and the resulting multi-volume rendering is shown in Figure 4.3. For each V-Object, the entry and exit points of all rays can be obtained by projecting its octree and setting the depth test to either less or greater. By a depth sort of these entry and exit points mono- and multi-object segments can be determined.

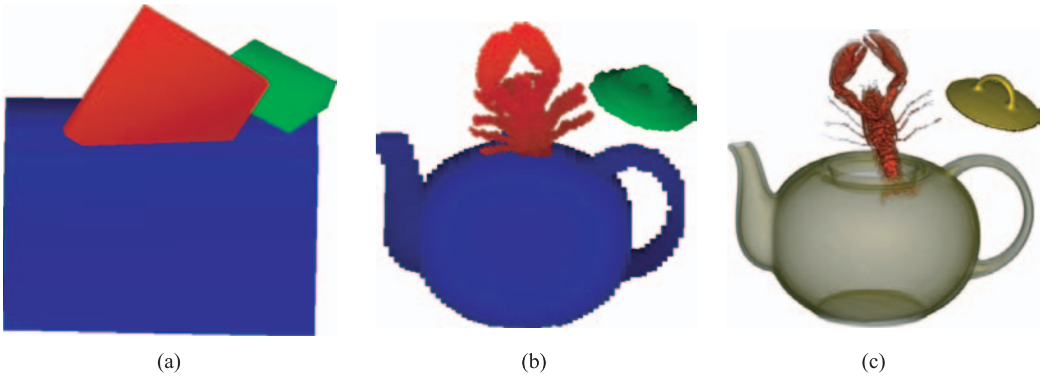


Figure 4.3: (a) Bounding box projection of multiple V-Objects. (b) Octree projection of multiple V-Objects. (c) Resulting multi-volume rendering.

All mono-object segments are assigned to the mono-volume renderer which is responsible for this V-Object. These segments can then be processed efficiently, as brick-wise traversal is possible. All other segments are added to one global multi-volume renderer. This renderer traverses every ray segment and performs multi-volume compositing in every step for all V-Objects which are defined at one resample location. This distribution of ray segments between mono- and multi-volume rendering is done in the initialization phase. After all ray segments have been assigned to their corresponding renderer, the actual raycasting process is started. All renderers operate independently. After they have finished, a final compositing step is required which combines the values accumulated in each segment of a ray.

The distribution of ray segments between mono- and multi-volume renderers is illustrated in Figure 4.4. Ray A consists only of one mono-object segment (A1). Ray B consists of the mono-object segments B1 and B3 which pass through V-Object II, and the multi-object segment B2 which passes through the intersection of the two V-Objects. Ray C consists of the mono-object segments C1 and C3 which pass through V-Object II, and the multi-object segment C2 which passes through the intersection of the two V-Objects. Thus, A1 is the only ray segment added to the mono-volume renderer for V-Object I. The segments B1, B3, C1, and C3 are added to the mono-volume renderer for V-Object II. The multi-volume segments B2 and C2 are added to the global multi-volume renderer. Through this kind of distribution only a fraction of the ray segments have to undergo costly multi-volume processing.

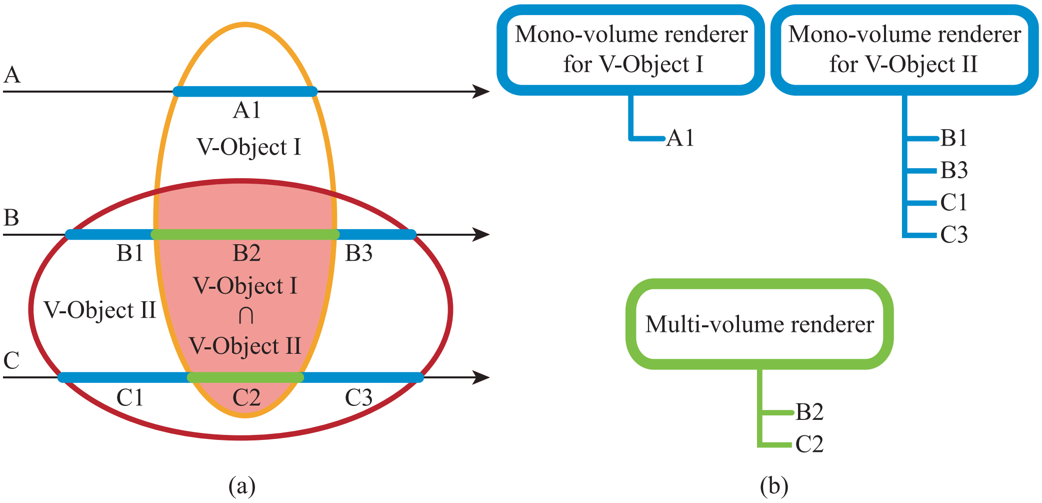


Figure 4.4: (a) Segmentation of three rays, A, B, and C, in a scene consisting of two V-Objects. (b) Distribution of ray segments among mono- and multi-volume renderers.

4.5 Results

The performance gains achieved through the multi-volume rendering approach depend on the size of intersections between the individual V-Objects. Typically, intersections are small, which allows us to exploit the high performance of mono-volume processing to accelerate the rendering. Moreover, if there is no intersection between any V-Object, the algorithm evaluates to a pure mono-volume renderer.

The approach is compared to a standard multi-volume raycaster where rays are processed sequentially, performing resampling and compositing in each of the V-Objects defined at every point along a ray. Figure 4.5 shows the speed-ups achieved for different degrees of intersection. As can be seen from the figure, the performance gains due to the algorithm depend on the size of the intersection between the V-Objects. For typical scenes, where the size of intersections is normally not excessively large, speed-ups of about 2.5 are achieved.

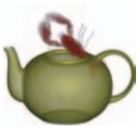
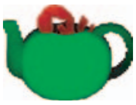
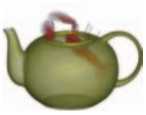
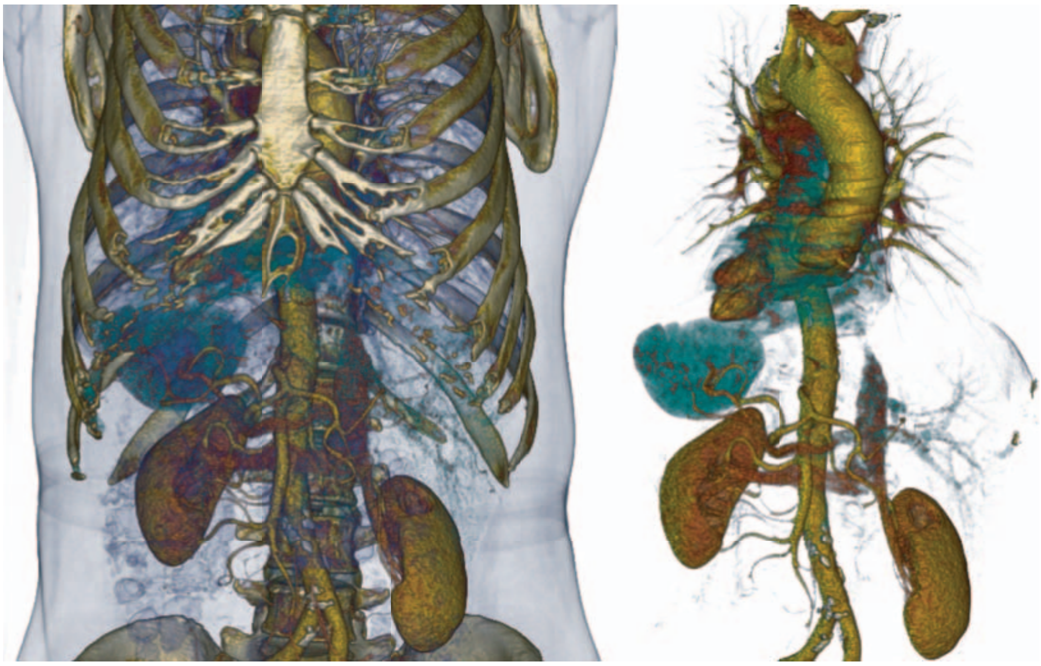
Speedup	Overlap	Octree Projection	Direct Multi-Volume Rendering
2.78	0.0 %		
2.43	21.7 %		
1.93	43.5 %		

Figure 4.5: Obtained speed-ups for different degrees of intersection. The first column shows the achieved speed-up of the approach compared to brute-force multi-volume rendering, and the second column shows the ratio between the combined lengths of all ray segments and the multi-object ray segments. The third column shows the octree projections of both V-Objects. The images (512 × 512) in the fourth column show the rendering results. Test system specifications: Intel Pentium 4, 2.4 GHz, 1 GB RAM.

4.6 Application of V-Objects

In general, in a volume rendering system there exist several means to enhance visual perception. Such means are for example transfer function, segmentation, and clipping. Since their introduction to volume visualization by Levoy [35], piecewise linear transfer functions mapping scalar values to colors and opacities are featured in virtually every volume visualization system. Many researchers have proposed to extend transfer functions to higher dimensions to increase their usability, as their specification is a non trivial task. Some approaches for semi automatic transfer function definition have been presented. However, a fully automatic transfer function selection remains a widely unsolved problem. Still, much research needs to be done to produce results automatically as shown in Figure 4.6a.



(a) (b)

Figure 4.6: (a) Transfer function example. (b) Segmentation example: The main vascular structures and the kidneys have been segmented.

One way to simplify the transfer function specification is segmentation. Many methods exists to identify certain structures within in the data. Most of these approaches produce a labeling of the data. Different transfer functions can be assigned to the identify regions or objects. Figure 4.6b shows an example of segmentation based on region growing. The main vascular structures and the kidneys have been segmented.

One problem in the visualization of volumetric data is occlusion. While transparency can be useful to simultaneously display different structures of interest, it can lead to cluttering when used extensively. It is common to cut away opaque objects to reveal occluded features. Cutting can be performed using axis aligned or arbitrary oriented planes, convex regions or arbitrarily shaped objects. However, complex cutting shapes are often difficult to interpret by the user. Figure 4.7 shows the use of cutting planes to reveal the inner parts of a torso.

In the following it is shown how such basic tools can be greatly enhanced by the use of V-Objects. Though many systems allow modification of transfer

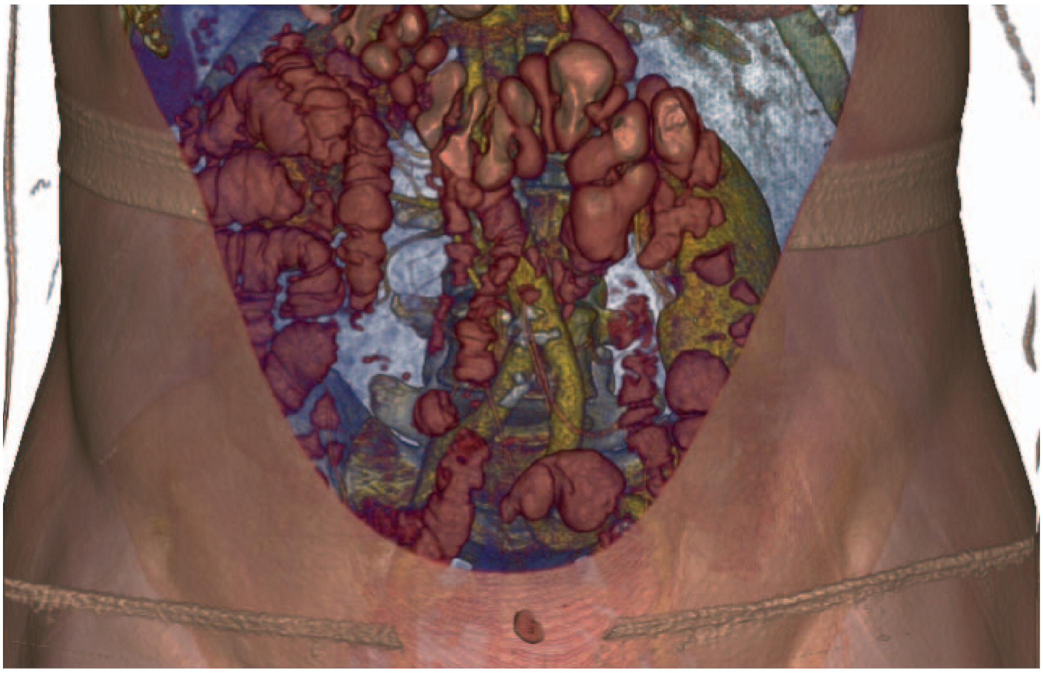


Figure 4.7: Clipping example.

functions and illumination properties can be specified on a per object basis certain limitations apply: objects typically can not intersect, are unique and static. These limitations are overcome in a natural way by assigning V-Objects to components of a data set. For example, with V-Objects it is possible to move an object while keeping a virtual copy in place. These two objects can have a totally different appearance. This capability can be used in applications such as surgical planning, education, illustration, and for investigation or exploration of data. In the following several examples for the use of V-Objects are given in combination with the previous described basic tools. Examples are shown for moving objects, simultaneously changing the appearance of objects, time varying, and multi-modal data. As some of these concepts are very hard to illustrate in still images, several animation sequences have been produced. The application features an interactive tool for the generation of animation sequences using key-framing. V-Objects can be interactively positioned in the scene and their properties can be modified. Between the key-frames, V-Object states are interpolated. This enables specification of animation paths, transfer function fades, light movement, control of clipping planes, change of data sources, enabling and disabling of objects, etc. For each of these properties, different interpolation schemes can be applied. In the following application examples are presented.

4.6.1 Advanced Browsing Techniques

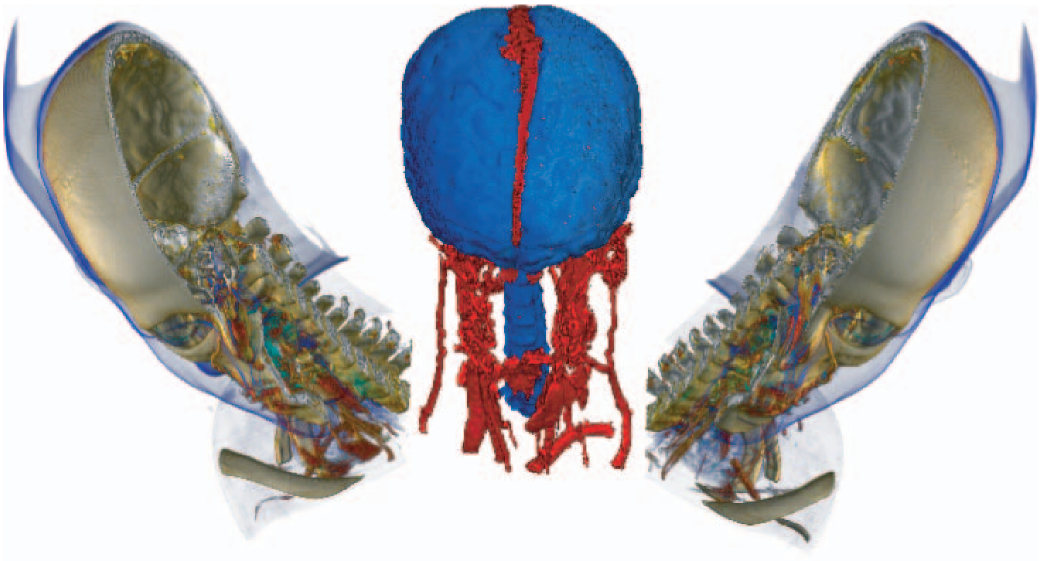


Figure 4.8: Virtual dissection of a human skull.

Traditional techniques for inspecting volumetric data like cutting involve removing portions of the data. This has the disadvantage of potentially hiding important contextual information. McGuffin et al. [42] have presented novel tools for browsing volume data which employ transformations and deformations of semantic layers contained in the data set. In the future, multi-volume rendering will be an important tool to improve the visual quality of such interaction techniques. Figure 4.8 demonstrates that V-Objects can be used to realize this kind of visualization. A virtual dissection of a human skull can be seen. Stills of a whole animation can be seen in Figure 4.9. Although only affine transformations are supported at present, the concept can be extended to more complex deformations in the future. On the other hand V-Objects can be used to indicate the positions of displaced structures. Multiple V-Objects can be assigned to the same structure in the data, only one of these V-Objects is deformed, the other objects remain in place to indicate the original position in space. Transparency is especially useful to indicate the position while not hiding surrounding important information. In Figure 4.10, an example of such an application of V-Objects is shown. The images represents stills of an animation. In the animation it

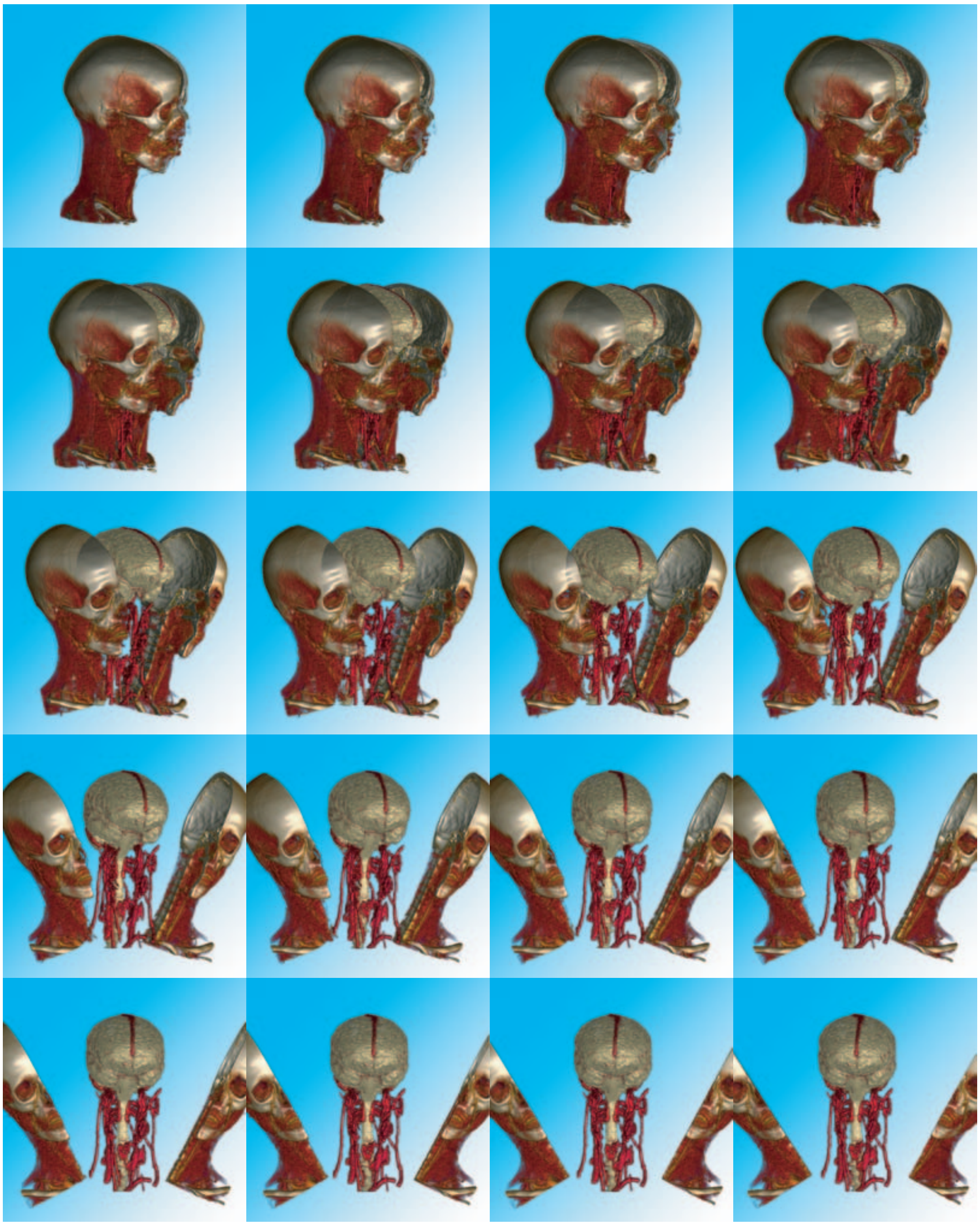


Figure 4.9: Stills of an animation to illustrate advanced browsing of volumetric data based on V-Objects. Virtual dissection of a human skull, uncovering the nervous and vascular system.

can be seen that a kidney is relocated to the left to reveal a tumor. Also other interesting features, such as transfer function fading, moving, and ob-

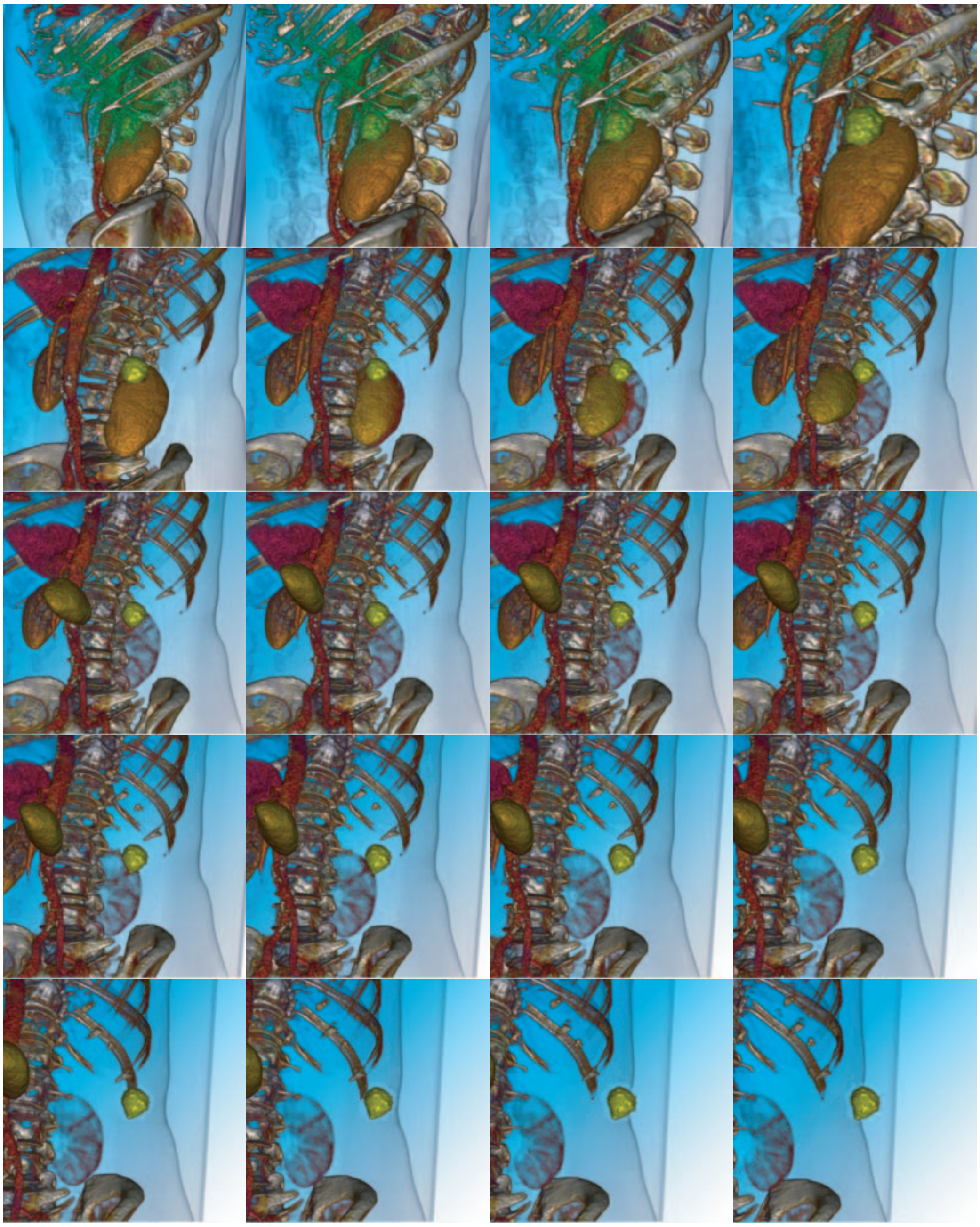


Figure 4.10: Stills of an animation to illustrate advanced browsing of volumetric data based on V-Objects. Enhancing anatomical features by spatial displacement (kidney and tumor).

ject specific cutting planes are shown. There are other applications, such as preoperative planning. For example, in maxillofacial surgery an important task is to fit implants, see Figure 4.11. Multi-volume rendering allows to simulate this process using actual patient data.

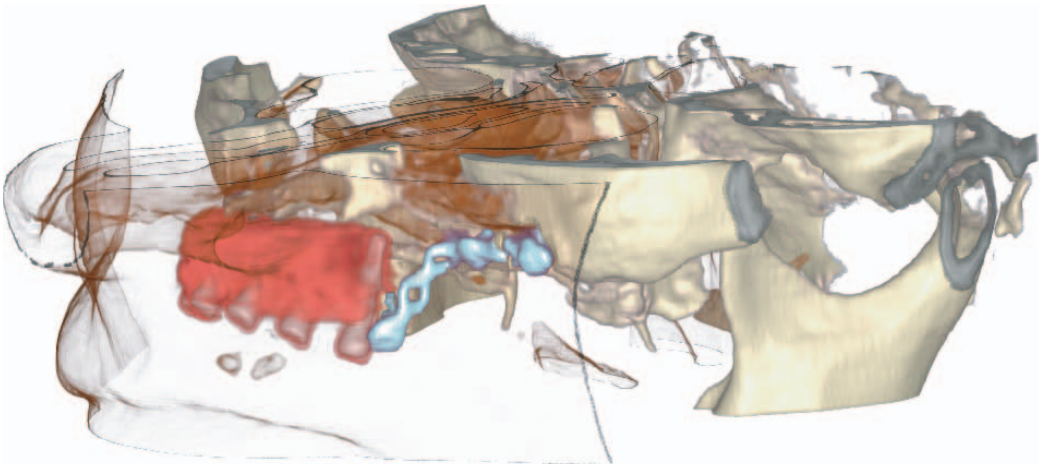


Figure 4.11: Maxillofacial surgery.

4.6.2 Time Varying Data

Visualization of time varying volume data is a very complex task, due to their dynamic nature. Animations often do not allow an in depth analysis of certain data characteristics. Static images on the other hand, often suffer from cluttering when many time-steps are visualized. It has been proposed to apply more advanced projection and mapping techniques to aid the understanding of such data. For example, Woodring at al. [82] apply hyperslicing to a 4D dataset. The flexibility of V-Objects allows to use a variety of different mappings using the combination of transfer functions and spatial arrangement of objects. An example of a time varying data set is shown in Figure 4.12. It is a electrocardiogram triggered CT scan of a beating heart. Time is mapped to color and opacity. This type of visualization allows simultaneously the three dimensional examination of several time steps. The fanning in time allows to convey similarities and differences in the progress of time. Furthermore, a topological relationship between different time steps is visualized. In general, V-objects support the explorations of useful layouts and mappings.

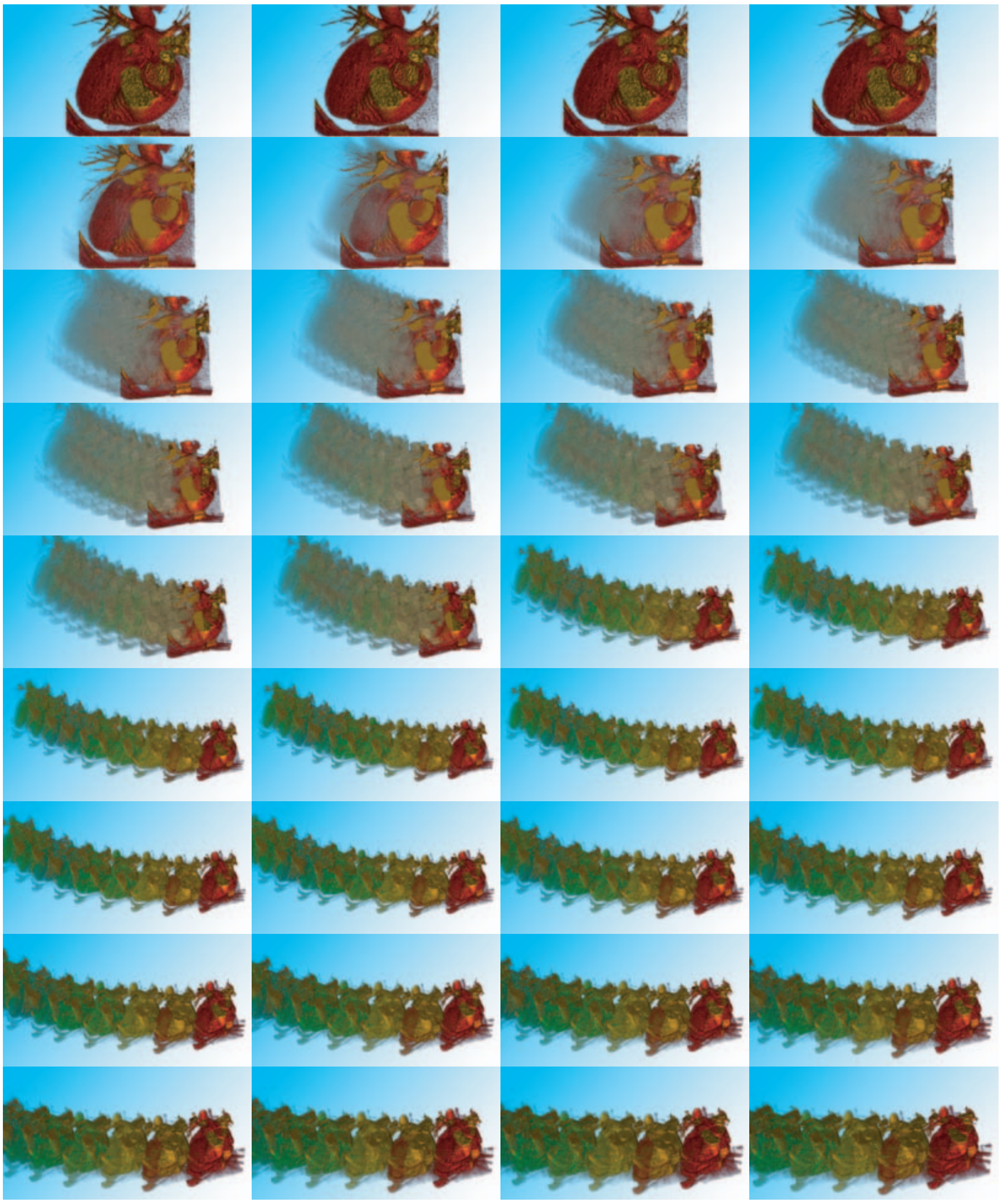


Figure 4.12: Stills of an animation to illustrate 4D visualization based on V-Objects. Fanning in time allows to convey similarities and differences in the progress of time.

4.6.3 Multi Modal Imaging

Multi-modal imaging is a method for combining disparate sets of 3D imaging data that contain complementary information on overlapping length scales.

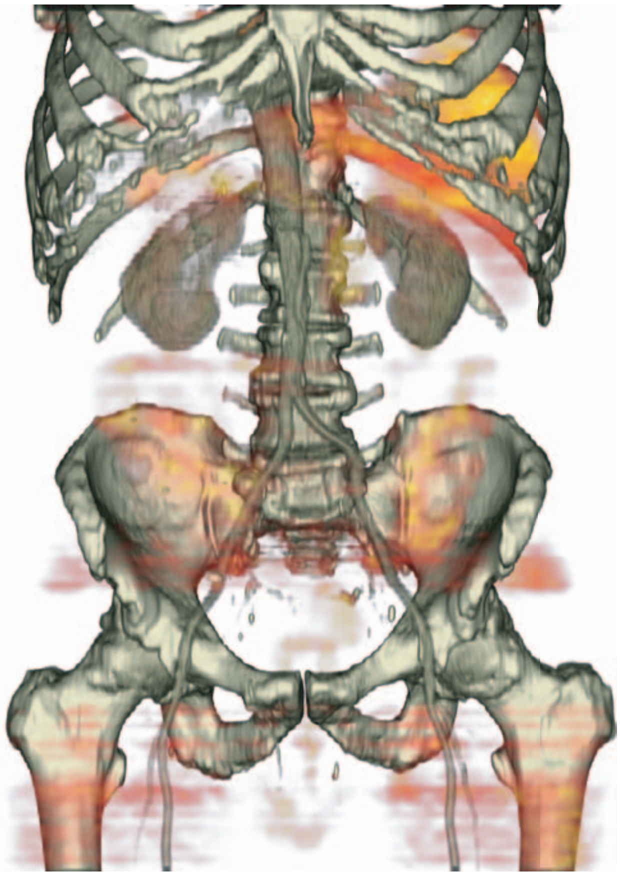


Figure 4.13: Fusion of CT and PET scan based on V-Objects.

In medicine, for instance, morphological modalities are combined with functional modalities to increase the information content of the resulting image. The concept of V-Objects inherently includes the ability to perform multimodal visualization of registered data sets. In Figure 4.13 an example is shown of a combination of computed tomography (CT) and positron emission tomography (PET). While the CT method supplies high precision, it is difficult to distinguish between tumors and healthy tissue. PET, on the other hand, is a functionally oriented method that allows to identify tumors, but provides lower resolutions.

4.7 Conclusion

V-Objects, a concept of modeling scenes consisting of multiple volumetric objects has been presented. It has been demonstrated that this concept, in combination with direct volume rendering, is a promising technique for visualizing medical data. Three examples for its application have been shown: browsing, time varying data, and multi-modal imaging. It has been shown that V-Objects can provide advanced means to explore and investigate data. Multi-volume visualization has great potential to improve medical applications.

Part II

Alternative Representation of Volume Data

Chapter 5

A Point-based Primitive for Volume Data

This chapter is based on the following publications:

Grimm S., Bruckner S., Kanitsar A., Gröller E., **VOTs: VOlume doTS as a Point-Based Representation of Volumetric Data**, *Computer Graphics Forum*, 23(3), pp. 661-668. 2004.

Grimm S., Bruckner S., Kanitsar A., Gröller E., **VOTs: VOlume doTS as a Point-Based Representation of Volumetric Data**, Technical Report TR-186-2-04-08, *Institute of Computer Graphics and Algorithms*, Vienna University of Technology, 2004.

5.1 Introduction

In general, volume graphics is the subfield of computer graphics that employs a volume buffer to represent a scene and is concerned with synthesizing, manipulating, and rendering such scenes. In this context it is referred to as *grid-based volume graphics*, when the volume data has a grid-based representation, and it is referred to as *point-based volume graphics*, when the volume data has a point-based representation. Figure 5.1 shows the taxonomy and dataflow of point-based volume graphics and grid-based volume graphics. The use of point-based volume graphics in various stages of volume processing and visualization is shown with red lines, the correspondences for grid-based volume graphics are shown with black lines. The major source of data for both volume graphics paradigms are analytical or discrete data. Grid-based volume graphics is very popular, as it allows efficient spatial addressing. The different grids, typically used, can be categorized into three

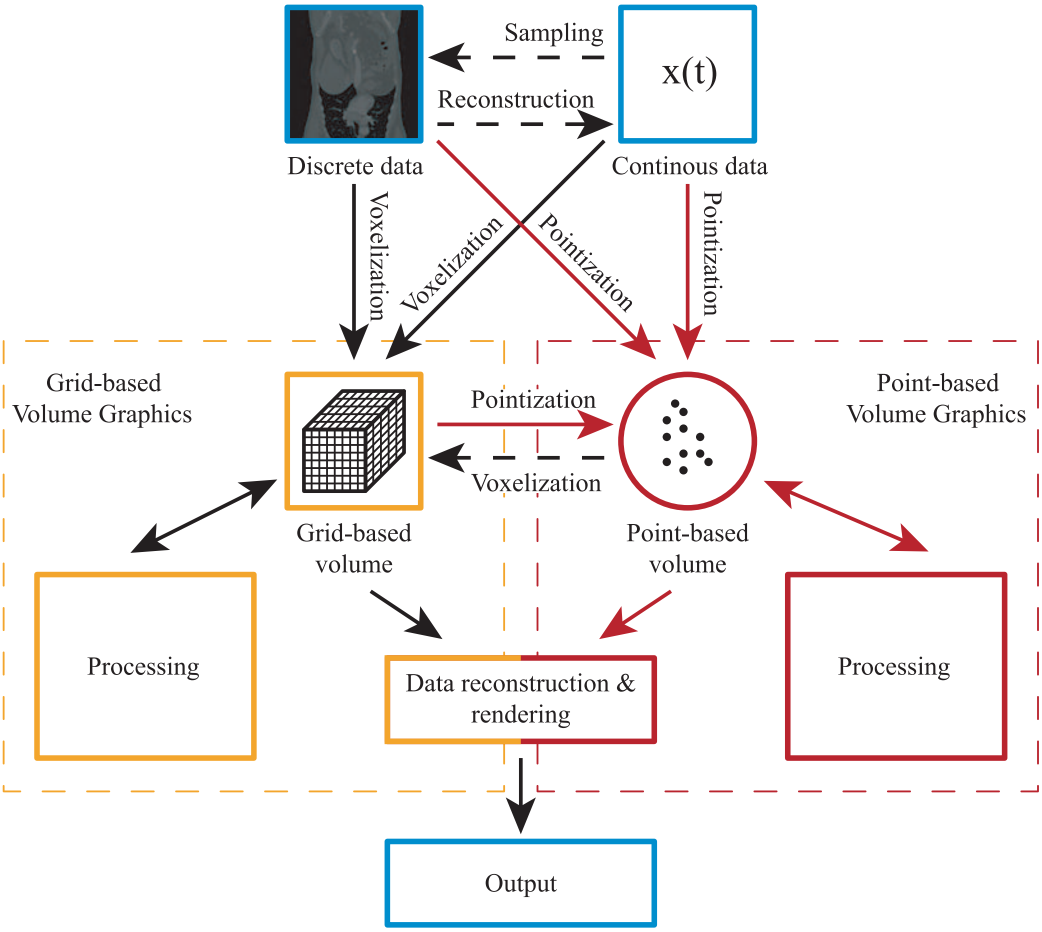


Figure 5.1: Grid-based volume graphics versus point-based volume graphics.

main types: rectilinear grids, curvilinear grids, and unstructured grids, see Chapter 1.

Research has shown that using grid structures has many advantages, such as the implicit efficient addressing of data elements and their regularity which is exploited by most algorithms. However, grids also have several limitations:

- The regular sampling pattern and limited resolution of such a grid representation results in limited position precision for sample points. This makes them inefficient when precisely representing high-frequency details.

- Data is only given at discrete locations, reconstruction for positions in between is necessary by using: nearest neighbor, linear, cubic, or sinc filters. Reconstruction inaccuracies are associated with it.

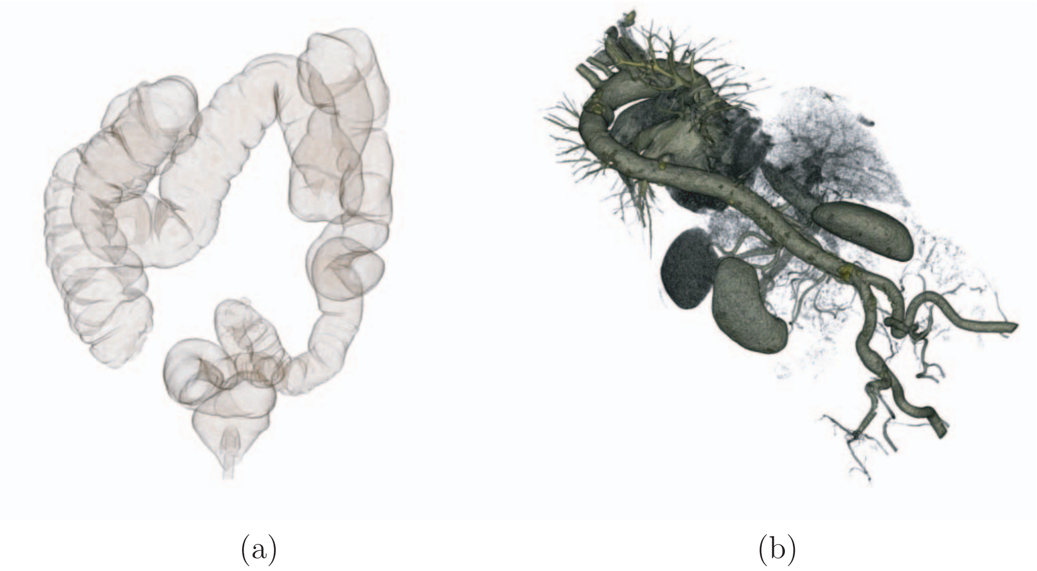
- Grid points regularly cover the underlying 3D domain, irrespective of the complexity of the volumetric function. It is a regular and not adaptive sampling.
- Mixing different grid types within a scene is considerably difficult as each of the grid structures needs a different kind of processing.
- The size of volumetric data sets is constantly increasing due to more advanced acquisition devices. The resulting data output of such devices can be enormous. In many cases, however, only a small portion or region of this data is actually of interest to the user.

A lot of research is devoted to efficiently handle grids and to overcome their limitations with sophisticated algorithms. In the previous chapters an efficient direct volume rendering approach for regular grids was presented [14, 13, 12]. Efficient algorithms for curvilinear grids have also been investigated [74, 16].

Medical applications, for example, suffer from several of the above mentioned limitations. Especially, the large amount of data which needs to be processed represents a huge challenge. Medium sized data, used in today's clinical routine, can be handled by current visualization systems. However, if it comes to time varying data or large scans, the systems are beyond their capabilities. The data sizes exceed the available physical memory and processing power. For example, Rubin et al. [67] reported a mean of 908 transverse slices for CT angiography. The grid resolution in this case is 512x512x908. Due to improved capabilities of newer acquisition devices it is possible to scan with higher resolution. The higher resolution is often used for sophisticated cases which also results in larger data. This large data presents a challenge to current volumetric data processing methods, such as rendering and segmentation. The applied data handling techniques are the outcome of many years of international research. There are several approaches which deal with large or time-varying data. Many of these approaches are based on compression or down-sampling techniques. The main drawback is that accuracy is traded off for performance. While this may be appropriate for some applications, it is generally not the case.

Volumetric data often only contains small regions of interest relevant to the application context. If the data is given on a regular grid non relevant volumetric regions are explicitly represented. This explicit representation demands costly storage resources and introduces additional complexity in processing algorithms. Furthermore, volumetric data sets where not all parts have to be represented with the same accuracy or resolution, i.e., *sparse* data sets, are often used.

Medical applications such as angiography or colonoscopy are examples that use sparse volumetric data sets. Figure 5.2a and Figure 5.2b show typical angiography and colonoscopy scans. The aorta is roughly 20 percent of the data set, and the colon even less because just the colon wall is of interest. Due to its rigid shape a regular grid structure is very inefficient for such sparse volumetric data sets. When examining Computed Tomography (CT) angiography run-offs, only the aorta is of main interest although the rest of the data contains much more information. The other parts of the data are less interesting, they are of lower importance. These other parts should not be completely ignored or removed, because they are needed as 3D context information. It would be sufficient to represent these other unimportant parts with lower accuracy and the important part (aorta) with higher accuracy. While this is a good property, which could be exploited to reduce the amount of data to handle, it is difficult to achieve using grid-based volumetric data representations. For a grid-based data representation, it is hard to change resolution in some parts of the volume; it would require to remove some grid positions. This, however, destroys the benefits of the grid and introduces additional problems.


(a)

(b)

Figure 5.2: (a) Colon - regular grid size 512x512x400 (b) Aorta, including and kidney and other features - regular grid size 512x512x1200.

Therefore, a paradigm shift from grids, as a spatial-oriented representation, to points, as an object-oriented representation, promises considerable advantages. Removing the grid-structure and introducing a point-based representation of volumetric data has the potential to simplify, and often even completely avoid many problems incurred by a grid-based representation.

In this chapter, the prior issues are addressed and Volume dots (Vots) [15] a new point-based primitive for volumetric data modeling, processing, and rendering is proposed. Vots are a functional representation of sample-based volumetric data. A Vot comprises the coefficients of a Taylor series expansion, which describes the underlying function in a given region. This approach converts a discrete representation into an implicit representation, and allows to exploit the advantages of analytically processing the data. Vots are a more intuitive and high-level description of the data. They enable the application of focus and context strategies to visualize data, and allow the representation of regions with different levels of detail. Vots also allow to leverage resources where they are needed, because they can be placed at any position.

The Vot representation opens new ways of data processing. But, it is not intended to replace the conventional representations. Vots are not well suited for very complex data sets with a high level of variation among the samples, where all samples are of equal importance. Vots work well for volumetric data in which only parts of the volume are of great importance.

This chapter is structured as follows: Section 5.2 surveys related work. Section 5.3 presents the general Vots data structure. Section 5.4 describes the Vot generation. Section 5.5 shows Maximum Intensity Projection based on Vots as an example application. In Section 5.6 results are presented and discussed. Finally, in Section 5.7 a conclusion is given and ideas for future work are presented.

5.2 Related Work

Recently more and more research is focused on point-based primitives for representation, modeling, processing, and rendering. The main reason for this is the increasing amount of data due to more advanced acquisition devices. Common representations reach their limits in the sense of performance and usability, therefore, new ways of data representations have to be exploited. The presented work, in this chapter, is also focused on such a point-based representation and is mainly inspired by the following work:

Levoy et al. [37] first proposed points as a rendering primitive in the mid-eighties. Following this idea, Pfister et al. [59] discussed Surfels as a method to efficiently render complex geometric objects at interactive frame rates. Unlike classical surface discretizations, i.e., triangles or quadrilateral meshes, surfels are point primitives without explicit connectivity. Welsh et al. [76] use wavelets to convert regular sampled point data to an irregular point hierarchy without reducing the precision of the data. Alexa et al. [1] propose a smooth manifold surface derived from a set of points close to the original surface. It is based on local maps from differential geometry, approximated by the method of moving least squares. Carr et al. [7] propose to use Radial Basis Functions (RBF) to reconstruct surfaces from point-cloud data. Surfaces are defined implicitly as the zero set of a RBF. Hopf et al. [20] propose a hierarchial splatting algorithm to visualize very large scattered point data at interactive frame-rates. Expensive resampling of the data is hereby avoided. Qu et al. [62] propose a rendering primitive called O-Buffers. It is a flexible structure that stores the positions of arbitrarily distributed samples relative to a regular grid. Rössl et al. [66] present a new approach to reconstruct non-discrete models from gridded volume samples. As a model, they use quadratic, trivariate super splines on a uniform tetrahedral partition. Csebfalvi et al. [8] propose a volume-rendering technique based on Monte Carlo integration. A point cloud of random samples is generated using a normalized continuous reconstruction of the volume as a probability density function. This point cloud is then projected onto the image plane. Lu et al. [39] present a framework for an interactive direct volume illustration system that simulates traditional stipple drawing. Xie at al. [83] address the problem of surface reconstruction from highly noisy point clouds. They fit at each sample point a quadric field which are then blended together to produce a pseudo signed distance field. Turk et al. [71] introduce new techniques for modeling with interpolating implicit surfaces. A 3D implicit function is created using a variational scattered data interpolation approach. The result surface is described by an iso-surface of this function. Ohtake et al. [55] investigate a shape representation, the multi-level partition of unity implicit surface, that allows to construct surface models from sets of points. There are also approaches which propose hybrid solutions. For example Wilson et al. [81] propose to mix conventional hardware assisted texture-based volume rendering with point-based rendering.

These ideas are extended and integrated, presenting a new primitive for volumetric data modeling, processing, and rendering. Vots are an alternative point-based representation of volumetric data.

5.3 Vot Data-Structure

Volumetric data sets can be seen as a volumetric scalar function $f : \mathcal{U} \subseteq \Re^3 \rightarrow \Re$, where $\mathcal{U}$ is the domain of f. With Vots a piece-wise representation $\tilde{f}_i : V_i \subseteq \mathcal{U} \rightarrow \Re$ of the volumetric scalar function f is presented. V_i with $\bigcup_{i=1..N} V_i \subseteq \mathcal{U}$ partitions the underlying space $\mathcal{U}$. Unimportant areas of $\mathcal{U}$, e.g., background areas are omitted. The scalar function over set V_i is represented by an individual Vot $\mathcal{V}_i$. Analogous to a Taylor series expansion all the relevant information for local function reconstruction is concentrated at a specific point P_i within a Vot $\mathcal{V}_i$. Relevant information includes the function value and higher order derivatives, such as gradient and Hessian matrix, at this point P_i. Vots are, thus, a set of points $\{P_1, P_2, ..., P_N\}$ in $\Re^3$. Each point $P_i = (P_i{}^x, P_i{}^y, P_i{}^z)$ constitutes a Taylor expansion point and locally represents the scalar volume function via the Taylor series expansion.

In general, the Taylor series expansion is defined as:

$$f(P + \Delta P) = \sum_{|\alpha| \leq N} \frac{1}{\alpha!} \partial^\alpha f(P) \Delta P^\alpha + R$$

$$R = \sum_{|\alpha| = N+1} \frac{1}{\alpha!} \partial^\alpha f(P + \theta \Delta P) \Delta P^\alpha, \; \theta \in [0, 1]$$

with:

$$\begin{array}{lcl} \alpha & \in & \{x, y, z\}^* \\ \partial^\alpha & = & \frac{\partial^{\alpha_x}}{\partial x^{\alpha_x}} \frac{\partial^{\alpha_y}}{\partial y^{\alpha_y}} \frac{\partial^{\alpha_z}}{\partial z^{\alpha_z}} \\ \alpha! & = & \alpha_x! \alpha_y! \alpha_z! \\ P^\alpha & = & \underbrace{P^x \cdots P^x}_{\alpha_x} \underbrace{P^y \cdots P^y}_{\alpha_y} \underbrace{P^z \cdots P^z}_{\alpha_z} \end{array}$$

Hereby α_μ denotes the number of occurrences of character μ in string α. For Vots, only terms of the Taylor series expansion up to a specific degree N are taken into account. The Taylor series expansion is approximated by:

$$f(P + \Delta P) \approx \tilde{f}(P + \Delta P) = \sum_{|\alpha| \leq N} \frac{1}{\alpha!} \partial^\alpha \tilde{f}(P) \Delta P^\alpha \tag{5.1}$$

Due to increasing storage demands and computational complexity with higher degrees, a degree of two or three is a good choice from a practical point of view. The derivatives up to degree three $\partial^\alpha \tilde{f}(P)$, $1 \leq |\alpha| \leq 3$ can be combined as follows:

$$\begin{aligned}
\nabla\tilde{f}(P) &= \begin{pmatrix} \tilde{f}_x \\ \tilde{f}_y \\ \tilde{f}_z \end{pmatrix} \\
H_{\tilde{f}}(P) &= \begin{pmatrix} \tilde{f}_{xx} & \tilde{f}_{xy} & \tilde{f}_{xz} \\ \tilde{f}_{yx} & \tilde{f}_{yy} & \tilde{f}_{yz} \\ \tilde{f}_{zx} & \tilde{f}_{zy} & \tilde{f}_{zz} \end{pmatrix} \\
T_{\tilde{f}}(P) &= \begin{matrix} \begin{pmatrix} \tilde{f}_{xxx} & \tilde{f}_{xyx} & \tilde{f}_{xzx} \\ \tilde{f}_{yxx} & \tilde{f}_{yyx} & \tilde{f}_{yzx} \\ \tilde{f}_{zxx} & \tilde{f}_{zyx} & \tilde{f}_{zzx} \end{pmatrix}, \\ \begin{pmatrix} \tilde{f}_{xxy} & \tilde{f}_{xyy} & \tilde{f}_{xzy} \\ \tilde{f}_{yxy} & \tilde{f}_{yyy} & \tilde{f}_{yzy} \\ \tilde{f}_{zxy} & \tilde{f}_{zyy} & \tilde{f}_{zzy} \end{pmatrix}, \\ \begin{pmatrix} \tilde{f}_{xxz} & \tilde{f}_{xyz} & \tilde{f}_{xzz} \\ \tilde{f}_{yxz} & \tilde{f}_{yyz} & \tilde{f}_{yzz} \\ \tilde{f}_{zxz} & \tilde{f}_{zyz} & \tilde{f}_{zzz} \end{pmatrix} \end{matrix}
\end{aligned}$$

Hereby ∇ denotes the gradient, H denotes the Hessian matrix, and T the tensor of third partial derivatives. The Hessian matrix, as well as the tensor of the third derivatives are symmetric. This property can be exploited for storage optimization.

Since each Vot $\mathcal{V}_i$ represents a certain neighborhood of the volumetric scalar function f, a validity area V_i is defined. This area V_i can be of arbitrary shape. From a practical point of view convex shapes such as spheres, ellipsoids, boxes, k-dops, etc. are advantageous. Each of these validity areas V_i is defined in such a way that for a given error ϵ (which might be zero) it holds:

$$\forall\,(P_i + \Delta P_i) \subseteq V_i : |f(P_i + \Delta P_i) - \tilde{f}(P_i + \Delta P_i)| < \epsilon. \tag{5.2}$$

A basic Vot $\mathcal{V}_i$ consists of:

- Position P_i.
- $(\partial^\alpha \tilde{f}(P_i))_{|\alpha| \leq N}$
- Validity area V_i for a given ϵ.

Vot properties can be extended to include attributes such as time-step, importance, etc. Vots represent the underlying volume data with data centric importance. Important data areas (i.e., large function variation) are represented with many small Vots (small validity areas V_i). Homogenous areas are represented with just a few large Vots. In addition the user may locally vary the approximation error ϵ, allowing a user-centric importance sampling. Focus areas are represented with $\epsilon = 0$. The context areas might have large errors, which could, for example, increase with distance to the focus area. The Vots data structure concentrates the information where data needs it and the user wants it.

5.4 Vot Generation

Before presenting a general method for Vot construction it is first shown, for illustration purposes, how to directly obtain a Vot representation for a 2x2x2 cell of a rectilinear grid. As reference reconstruction trilinear interpolation within the cell is assumed.

5.4.1 Vot Generation of a Cell

A cell is given as eight pairs: $(P_{ijk}, f_{P_{ijk}})_{i,j,k \in \{0,1\}}$. Hereby P_{ijk} denotes a position at one of the corners of the cell, and $f_{P_{ijk}}$ the corresponding function value. Furthermore a trilinear reconstruction filter is assumed. The expansion point needed for the Taylor expansion is defined as the center of the cell by:

$$P = \frac{1}{8} \sum P_{ijk}$$

The terms $\tilde{f}(P)$, $\nabla \tilde{f}(P)$, $H_{\tilde{f}}(P)$, and $T_{\tilde{f}}(P)$ for the Taylor series expansion up to degree three can directly be specified by:

$$\begin{array}{rcl}
\tilde{f}(P) & = & \frac{1}{8} \sum f_{P_{ijk}} \\
\nabla \tilde{f}(P) & = & \frac{1}{4} \begin{pmatrix} \sum_{j,k} f_{P_{1jk}} - \sum_{j,k} f_{P_{0jk}} \\ \sum_{i,k} f_{P_{i1k}} - \sum_{i,k} f_{P_{i0k}} \\ \sum_{i,j} f_{P_{ij1}} - \sum_{i,j} f_{P_{ij0}} \end{pmatrix} \\
H_{\tilde{f}}(P) & = & \frac{1}{2} \begin{pmatrix} 0 & \tilde{f}_{xy} & \tilde{f}_{xz} \\ \tilde{f}_{xy} & 0 & \tilde{f}_{yz} \\ \tilde{f}_{xz} & \tilde{f}_{yz} & 0 \end{pmatrix} \\
T_{\tilde{f}}(P) & = & \tilde{f}_{xyz}
\end{array} \tag{5.3}$$

where

$$\begin{aligned}\tilde{f}_{xy} = & \textstyle\sum_{ijk\in\{000,001,110,111\}} f_{P_{ijk}} - \\ & \textstyle\sum_{ijk\in\{010,011,100,101\}} f_{P_{ijk}} \\ \tilde{f}_{xz} = & \textstyle\sum_{ijk\in\{000,010,101,111\}} f_{P_{ijk}} - \\ & \textstyle\sum_{ijk\in\{001,011,100,110\}} f_{P_{ijk}} \\ \tilde{f}_{yz} = & \textstyle\sum_{ijk\in\{000,011,100,111\}} f_{P_{ijk}} - \\ & \textstyle\sum_{ijk\in\{001,010,101,110\}} f_{P_{ijk}}\end{aligned}$$

and

$$\begin{aligned}\tilde{f}_{xyz} = & \textstyle\sum_{ijk\in\{001,010,100,111\}} f_{P_{ijk}} - \\ & \textstyle\sum_{ijk\in\{000,011,101,110\}} f_{P_{ijk}}\end{aligned}$$

Every data value within the cell can be reconstructed by evaluating Equation 5.1. The Vots representation of a cell requires altogether eight values. $\tilde{f}(P) \leadsto 1$, $\nabla\tilde{f}(P) \leadsto 3$, $H_{\tilde{f}}(P) \leadsto 3$, and $T_{\tilde{f}}(P) \leadsto 1$. All the remaining values are either redundant due the symmetry of the Hessian matrix or are zero. In terms of storage requirement a Vot representation is equal to a cell representation. However, in a regular grid, grid points are reused for (eight) neighboring cells. A straightforward conversion of a regular grid into a Vot structure would therefore increase the storage requirements by a factor of eight. A closer look reveals that only every other cell (in each of the three spatial directions) must be represented by a Vot. The function in cells which do not contain a Vot can be exactly reconstructed from the neighboring Vots. Such an approach would not change the storage requirements. With the Vots representation a more intuitive specification of the underlying function is obtained.

5.4.2 General Vot Generation

One of the major reasons for the introduction of Vots is to be able to leverage resources where they are needed. Homogeneous or non important regions should be represented just by few resources. On the other hand, inhomogeneous and important regions should be represented by an adequate amount of resources. The amount is defined by the desired accuracy. Vots provide this feature, as they can be placed at any arbitrary position.

In the following an approach is presented to generate a Vot for a given set of m scattered data points $Q_j \in \Re^3$ with data values f_{Q_j}. It is assumed that a Vot uses the Taylor series expansion up to degree $N = 3$, as shown in Equation 5.1. This can be extended to arbitrary degrees straightforwardly. To generate the Vots, an approach is employed which is similar to the linear regression approach for normal vector estimation used in [54]. To be able to apply this approach the mean square error of the fitting process is defined as:

$$E(\cdots) = \sum_{j=1}^{m} (\tilde{f}(Q_j) - f_{Q_j})^2 \tag{5.4}$$

Hereby f_{Q_j} denotes the function values given at the m scattered points, $\tilde{f}(Q_j)$ denotes the function value reconstructed by the approximated Taylor series $\tilde{f}$ at Q_j, and E is the sum of the squared differences between the original values and the reconstructed values. As Taylor series expansion point the center of gravity is chosen:

$$P = \frac{1}{m} \sum Q_j$$

The unknown variables of E are:

$$\tilde{f}, \tilde{f}_i, \tilde{f}_{ij}, \tilde{f}_{ijk}$$

where $i, j, k \in \{x, y, z\}$ and $\tilde{f}_i, \tilde{f}_{ij}, \tilde{f}_{ijk}$ denote the partial derivatives $\partial^i \tilde{f}$, $\partial^{ij} \tilde{f}$, $\partial^{ijk} \tilde{f}$. The number of unknowns can be reduced due to symmetry of the Hessian matrix H and the the tensor T of the third derivatives to 20 unknowns. Furthermore it is defined:

$$\Delta Q_j \; = Q_j - P = ({Q_j}^x - P^x, {Q_j}^y - P^y, {Q_j}^z - P^z)$$

The minimum of the error function E is determined by taking the partial derivatives with respect to the unknowns and setting these partial derivatives to zero. To achieve this, $\tilde{f}(Q_j)$ is substituted according to Equation (5.1) with:

$$\sum_{|\alpha| \leq 3} \frac{1}{\alpha!} \partial^\alpha \tilde{f}(P) \Delta {Q_j}^\alpha$$

The derivatives are given as:

$$\begin{aligned}
\frac{\partial E}{\partial \tilde{f}} &= 2\sum_{j=1}^{m}\left(\sum_{|\alpha|\leq 3}\tfrac{1}{\alpha!}\partial^{\alpha}\tilde{f}(P)\Delta Q_j{}^{\alpha} - f_{Q_j}\right)\\
\frac{\partial E}{\partial \tilde{f}_x} &= 2\sum_{j=1}^{m}\left(\sum_{|\alpha|\leq 3}\tfrac{1}{\alpha!}\partial^{\alpha}\tilde{f}(P)\Delta Q_j{}^{\alpha} - f_{Q_j}\right)\Delta Q_j{}^{x}\\
&\vdots\\
\frac{\partial E}{\partial \tilde{f}_{xy}} &= 2\sum_{j=1}^{m}\left(\sum_{|\alpha|\leq 3}\tfrac{1}{\alpha!}\partial^{\alpha}\tilde{f}(P)\Delta Q_j{}^{\alpha} - f_{Q_j}\right)\Delta Q_j{}^{x}\Delta Q_j{}^{y}\\
&\vdots\\
\frac{\partial E}{\partial \tilde{f}_{xxy}} &= 2\sum_{j=1}^{m}\left(\sum_{|\alpha|\leq 3}\tfrac{1}{\alpha!}\partial^{\alpha}\tilde{f}(P)\Delta Q_j{}^{\alpha} - f_{Q_j}\right)(\Delta Q_j{}^{x})^2\Delta Q_j{}^{y}\\
&\vdots
\end{aligned} \tag{5.5}$$

Setting the derivatives to zero, leads to the following system of linear equations:

$$M\begin{pmatrix}\tilde{f}\\ \tilde{f}_x\\ \vdots\\ \tilde{f}_{xy}\\ \vdots\\ \tilde{f}_{xxy}\\ \vdots\end{pmatrix} = \begin{pmatrix}\sum_{j=1}^{m} f_{Q_j}\\ \sum_{j=1}^{m} f_{Q_j}\Delta Q_j{}^{x}\\ \vdots\\ \sum_{j=1}^{m} f_{Q_j}\Delta Q_j{}^{x}\Delta Q_j{}^{y}\\ \vdots\\ \sum_{j=1}^{m} f_{Q_j}(\Delta Q_j{}^{x})^2\Delta Q_j{}^{y}\\ \vdots\end{pmatrix} \tag{5.6}$$

where M is the 20×20 matrix resulting from the following sum of vector direct products:

$$M = \sum_{j=1}^{m} \begin{pmatrix} 1 \\ \Delta Q_j^{x} \\ \Delta Q_j^{y} \\ \Delta Q_j^{z} \\ (\Delta Q_j^{x} \Delta Q_j^{x})/2 \\ \Delta Q_j^{x} \Delta Q_j^{y} \\ \Delta Q_j^{x} \Delta Q_j^{z} \\ (\Delta Q_j^{y} \Delta Q_j^{y})/2 \\ \Delta Q_j^{y} \Delta Q_j^{z} \\ (\Delta Q_j^{z} \Delta Q_j^{z})/2 \\ (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{x})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{z})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{y} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{z} \Delta Q_j^{x} \Delta Q_j^{z})/6 \\ \Delta Q_j^{x} \Delta Q_j^{y} \Delta Q_j^{z} \\ (\Delta Q_j^{y} \Delta Q_j^{y} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{y} \Delta Q_j^{y} \Delta Q_j^{z})/6 \\ 3 \cdot (\Delta Q_j^{y} \Delta Q_j^{z} \Delta Q_j^{z})/6 \\ (\Delta Q_j^{z} \Delta Q_j^{z} \Delta Q_j^{z})/6 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ \Delta Q_j^{x} \\ \Delta Q_j^{y} \\ \Delta Q_j^{z} \\ (\Delta Q_j^{x} \Delta Q_j^{x})/2 \\ \Delta Q_j^{x} \Delta Q_j^{y} \\ \Delta Q_j^{x} \Delta Q_j^{z} \\ (\Delta Q_j^{y} \Delta Q_j^{y})/2 \\ \Delta Q_j^{y} \Delta Q_j^{z} \\ (\Delta Q_j^{z} \Delta Q_j^{z})/2 \\ (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{x})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{x} \Delta Q_j^{z})/6 \\ 3 \cdot (\Delta Q_j^{x} \Delta Q_j^{y} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{z} \Delta Q_j^{x} \Delta Q_j^{z})/6 \\ \Delta Q_j^{x} \Delta Q_j^{y} \Delta Q_j^{z} \\ (\Delta Q_j^{y} \Delta Q_j^{y} \Delta Q_j^{y})/6 \\ 3 \cdot (\Delta Q_j^{y} \Delta Q_j^{y} \Delta Q_j^{z})/6 \\ 3 \cdot (\Delta Q_j^{y} \Delta Q_j^{z} \Delta Q_j^{z})/6 \\ (\Delta Q_j^{z} \Delta Q_j^{z} \Delta Q_j^{z})/6 \end{pmatrix}^{T} \tag{5.7}$$

The inversion of matrix M produces the solution for the unknown variables. The error is calculated by $\epsilon = max_j|\tilde{f}(Q_j) - f_{Q_j}|$. The mechanism described allows the construction of a Vot for a given input set of points $Q_j, (j = 1, ..., m)$.

5.4.3 Vot-Space

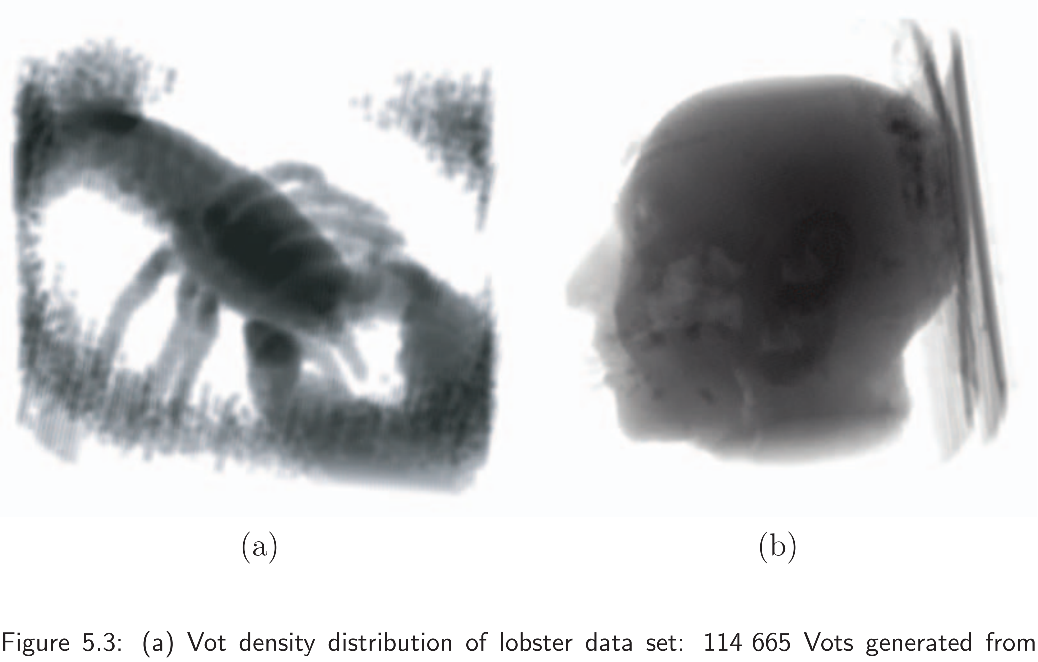
Vots allow an importance-based representation of volume data. To achieve this they abandon the implicit connectivity information of regular grids. The most basic question the data structure has to answer is: Given an arbitrary point P find the corresponding Vot $\mathcal{V}_i$ $(P \in V_i)$ so that the function value at position P can be determined. Depending on the shape of the validity area V_i efficient indexing structures from computational geometry, such as range trees, interval trees, octrees and bounding volume hierarchies can be used to accelerate this search. Application dependent indexing structures might also help to quickly address, for example, all the Vots whose gradient is within a certain magnitude or direction range.

A Vot-Space $(\mathcal{V}_j, \mathcal{I})$ comprises a set of Vots $\mathcal{V}_j$ and a set of indexing structures $\mathcal{I}$. $\mathcal{I}$ contains at least I^*, which is an unsorted list of all Vots. It is used to address each Vot. For some application this simple indexing structure is sufficient, see Section 5.5. It is application specific on which kind of indexing structure a Vot-Space depends on.

5.5 MIP as an Application of Vots

To give a proof of concept of the new data structure, an application of Vots is shown. Maximum Intensity Projection (MIP) of a Vot-Space is presented. Maximum Intensity Projection [49] is a technique that displays the maximum scalar value seen through each image pixel. By depicting the maximum data value, high intensity structures contained in the data are captured. A straight-forward method for calculating Maximum Intensity Projection is to perform ray casting and search for the maximum sample value along each ray. This visualization method is used to illustrate the advantages of Vots. Instead of using sampling, the maximum of a viewing ray within the validity area of a Vot is analytically determined.

5.5.1 From a Grid-based to a Vot-based Representation



(a) (b)

Figure 5.3: (a) Vot density distribution of lobster data set: 114 665 Vots generated from 445 568 input samples. (b) Vot density distribution of UNC head data set: 570 690 Vots generated from 1 746 360 input samples.

As input a rectilinear grid is assumed. It is given as a set of pairs $G = \{(P_j, f_{P_j}),\ j = \{1, ..., N\}\}$ where $P_j = ({P_j}^x, {P_j}^y, {P_j}^z)$ defines a position within the grid and f_{P_j} the corresponding function value $f(P_j)$. The task is to find a small number of Vots $\mathcal{V}_j$ which covers completely the underlying volumetric data. Reconstruction should be bound by an error ϵ, as given in Equation 5.2.

A growing approach is used. As growing criterion the function $\mathcal{F}(T)$ is defined as follows:

$$\mathcal{F}(T) = max_j|\tilde{f}(Q_j) - f_{Q_j}|, Q_j \in T \tag{5.8}$$

where $T \subseteq G$. $\mathcal{F}$ implements Equation 5.6 for the point set T, it solves the linear equation system and returns the maximal error. Furthermore, for simplicity box-shaped validity areas of the Vots are assumed. As possible start positions of Vots the positions given by G are choosen. The validity area of each Vot is initially cell-sized, covering eight adjacent grid positions. At this point, the growing process of each Vot is iteratively started by increasing one of its validity area dimensions either in the positive or negative direction. In every step the error computed by $\mathcal{F}$ is tested. The current input set T is defined by all grid positions which lie within the increased validity area. According to the outcome of $\mathcal{F}$ and the given ϵ-bound either the increased validity area is kept as new area or the old validity area is kept. A Vot grows until error $\mathcal{F}(T)$ is larger than ϵ. According to Equation 5.8 the error is just tested at positions Q_j.

The result of this process is a set of the largest possible Vots for every grid position with ϵ accuracy. The next step is to find a minimal subset of the set of Vots which completely cover the underlying volumetric data. In order to achieve this a cover weight $\mathcal{W}_j$ is assigned to each Vot $\mathcal{V}_j$. The weights initially correspond in size to the validity areas V_j. The Vots are sorted according to their weights $\mathcal{W}_j$ in decreasing order. The Vot with the largest $\mathcal{W}_j$ is taken and is included in the minimal subset of Vots. The number of grid points is determined this Vot would cover in the grid. Only those grid positions are counted, which are not covered by other Vots in the current small subset. The weights $\mathcal{W}_j$ of all the remaining Vots are adapted, according to the number of grid positions they could cover, which are not already covered by other Vots out of the current small subset. Then the Vots are resorted and the process starts all over. Once the complete grid is covered the process stops and a small subset of Vots is found which defines the Vot-Space with a given indexing structure I^*.

Figure 5.3 shows the Vot distribution of typical data sets. Dark areas correspond to high Vot density and bright areas to low Vot density.

5.5.2 Maximum Intensity Projection

For Maximum Intensity Projection the following question arises: Given a Vot $\mathcal{V}$ and a viewing direction, how to compute the maximum along this ray. The answer is provided by the Taylor series expansion of one Vot, as given in Equation 5.1. For simplicity reasons $N = 2$ is chosen in the explanation. From this follows that the Taylor series is given as:

$$\tilde{f}(P + \Delta P) = \tilde{f}(P) + \nabla \tilde{f}(P)\Delta P + \tfrac{1}{2}\Delta P \; H_{\tilde{f}}(P) \; \Delta P \tag{5.9}$$

A ray r is given as:

$$r(t) = S + tD$$

To determine the extreme the ray equation is substituted into Equation 5.9 and the first derivative is taken. Furthermore $\Delta P_S := S - P$ is defined and it is:

$$\partial^1 \tilde{f}(S + tD) = a_0 + a_1 t$$

with

$$\begin{aligned} a_0 = \; & \tilde{f}_x D^x + \tilde{f}_y D^y + \tilde{f}_z D^z + \\ & \tilde{f}_{xx} D^x \Delta {P_S}^x + \tilde{f}_{yy} D^y \Delta {P_S}^y + \tilde{f}_{zz} D^z \Delta {P_S}^z + \\ & \tilde{f}_{xy}(D^x \Delta {P_S}^y + D^y \Delta {P_S}^x) + \tilde{f}_{xz}(D^x \Delta {P_S}^z + D^z \Delta {P_S}^x) + \\ & \tilde{f}_{yz}(D^y \Delta {P_S}^z + D^z \Delta {P_S}^y) \\ a_1 = \; & \tilde{f}_{xx}(D^x)^2 + \tilde{f}_{yy}(D^y)^2 + \tilde{f}_{zz}(D^z)^2 + \\ & 2\tilde{f}_{xy} D^x D^y + 2\tilde{f}_{xz} D^x D^z + 2\tilde{f}_{yz} D^y D^z \end{aligned}$$

The position of the extreme is then determined by setting the first derivative to zero. The resulting equation is solved with respect to t:

$$t = \frac{-a_0}{a_1}$$

To determine wether the extreme is a maximum or minimum, $S + tD$ is substituted into Equation 5.9, and the second derivative is taken:

$$\begin{aligned} \partial^2 \tilde{f}(S + tD) = \; & \tilde{f}_{xx}(D^x)^2 + \tilde{f}_{yy}(D^y)^2 + \tilde{f}_{zz}(D^z)^2 + \\ & 2\tilde{f}_{xy} D^x D^y + 2\tilde{f}_{xz} D^x D^z + 2\tilde{f}_{yz} D^y D^z \end{aligned}$$

The sign of the second derivative determines if the extreme is a maximum or a minimum. Knowing these two derivatives it is straightforward to determine the maximum along a ray within the validity area of a Vot. There are two cases:

1. A maximum within the validity area is found.

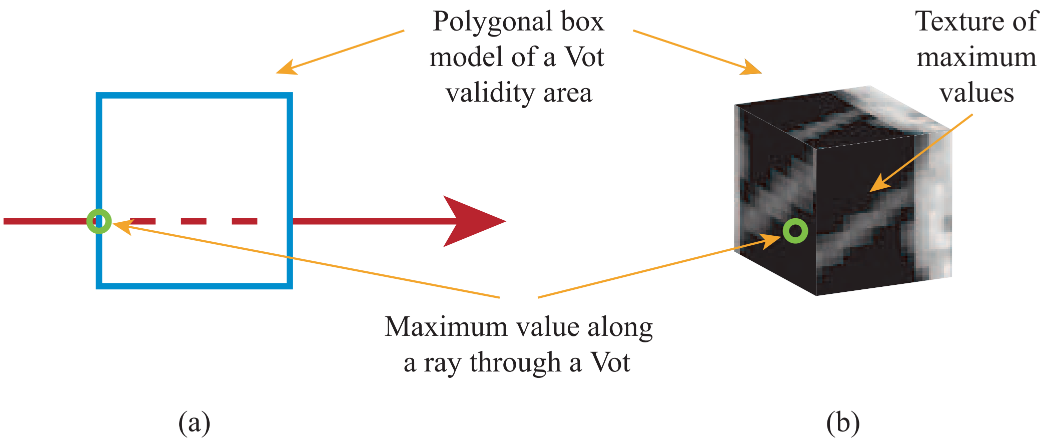

Polygonal box model of a Vot validity area
Texture of maximum values
Maximum value along a ray through a Vot
(a)
(b)

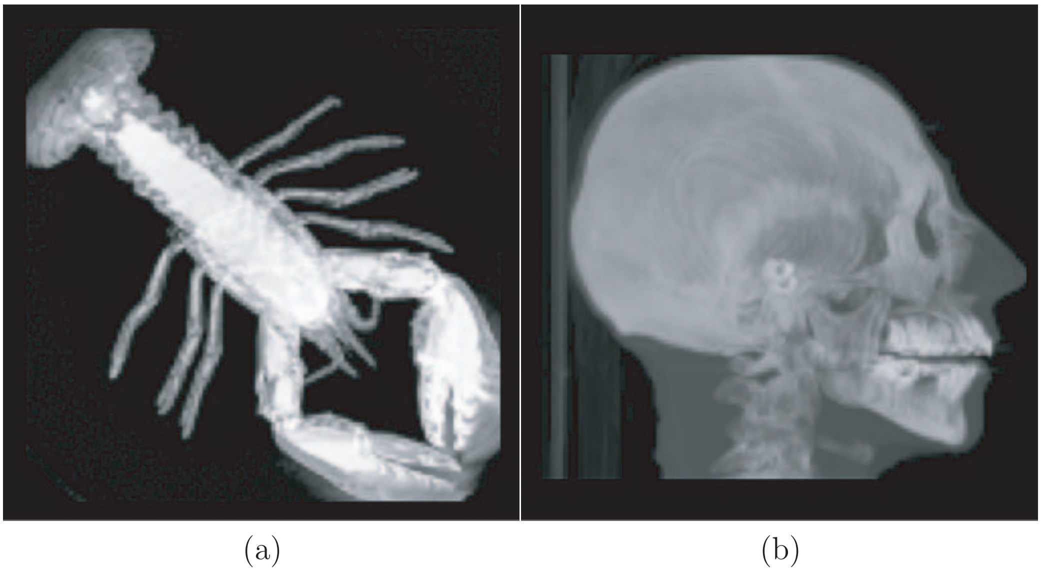
Figure 5.4: Maximum Intensity Projection of one Vot: (a) Illustrated in 2D. (b) Illustrated in 3D.

2. No maximum within the validity is found, the maximum along the ray occurs at one of the intersection points with the validity area box.

With this method, the maximum scalar value of an arbitrary ray passing through a Vot can be determined analytically. The MIP algorithm works as follows: The unsorted list of Vots is traversed. For each Vot, its validity region is represented by a polygonal model. For every ray that intersects the Vot, the intersection point S is calculated and the maximum along the ray is computed and stored in an image. There is one image for each visible face of the polygonal model. In this case, the validity area is always box-shaped, therefore at most three faces are visible. Texture mapping is used to transform the images according to the validity area geometry. The images are textured onto the corresponding faces of the model, as illustrated in Figure 5.4. The graphics hardware's capability is used to perform maximum blending of different Vots. With a prototype application two example data sets are rendered, see Figure 5.5.

5.6 Discussion and Results

This work is at an initial stage and still has many issues to address. The brute-force algorithm, of Section 5.5.1, for converting a rectilinear data set into a Vot-Space representation has high computational complexity. Even for small data sets, for example of size 128^3, it needs several hours of computation time on a commodity PC. This is mainly due to the applied exhaustive growing strategy. The larger a validity area V_i becomes, the higher is the



(a) (b)

Figure 5.5: (a) Maximum Intensity Projection of Lobster: 114 665 Vots. (b) Maximum Intensity Projection of UNC head: 570 690 Vots.

number of fitting operations that have to be performed. Currently Vots are grown from every grid-position and later on a huge number of redundant Vots are discarded. It is obvious that for example centers of homogeneous regions are better choices to place Vots than inhomogeneous regions. Applying a more sophisticated seeding strategy will considerably reduce the Vots generation effort.

Another issue which needs to be addressed is the size of Vots and the number of Vots needed. In Figure 5.3 quite a high number of Vots is used. Such high numbers are only of limited practicability. A Vot with a Taylor series expansion of up to degree three defines a rather rigid underlying volume function with limited shape possibility. Medical data often contains high frequencies and noise. To adhere to the defined error bound the Vots are, thus, rather limited in size. To obtain bigger Vots or a smaller number of Vots one has to allow a higher error ϵ. However, a higher error ϵ leads to discontinuities. Figure 5.6 shows the resulting number of Vots for different error bounds ϵ. Discontinuity at the borders can be handled through blending of adjacent Vots or of Vots with overlapping validity areas V_i. Overlapping is not an issue in the presented MIP application, as the maximum along a ray remains the same even if it is determined multiple times due to the overlapping. In the current implementation the error function of Equation 5.8 is evaluated only at sample positions Q_j. To ensure the error bound

Error ε:	0.4096	4.096	40.96	409.6
(a) # Vots:	570 690	538 919	333 650	35 079
(b) # Vots:	114 665	114 665	112 604	60 353

Figure 5.6: Number of Vots for different error bounds ϵ. The error ϵ defines the maximum allowed deviation with respect to the original samples. Reconstructing data samples in between the original samples can lead to larger errors. The data range interval is [0,4095]. (a) UNC head (126x126x110): 1 746 360 samples. (b) Lobster (118x118x32): 445 568 samples.

also in between a finer sampling of this error function is recommended. The overall fitting process remains the same.

Finally, a rendering approach of a Vot-Space is needed, which performs at least in the same range as conventional volume rendering approaches. In general, it should be possible to apply image- and object-order direct volume rendering. For an image order approach a ray is send through the Vot-Space, *jumping* from Vot to Vot in the correct visible order while resampling each individual Vot along the ray. The *jumping* corresponds to empty space skipping. For object order rendering one could apply a technique similar to splatting. However, an appropriate interpolation kernel has to be developed as a Vot contains the condensed information of a region instead of one sample position.

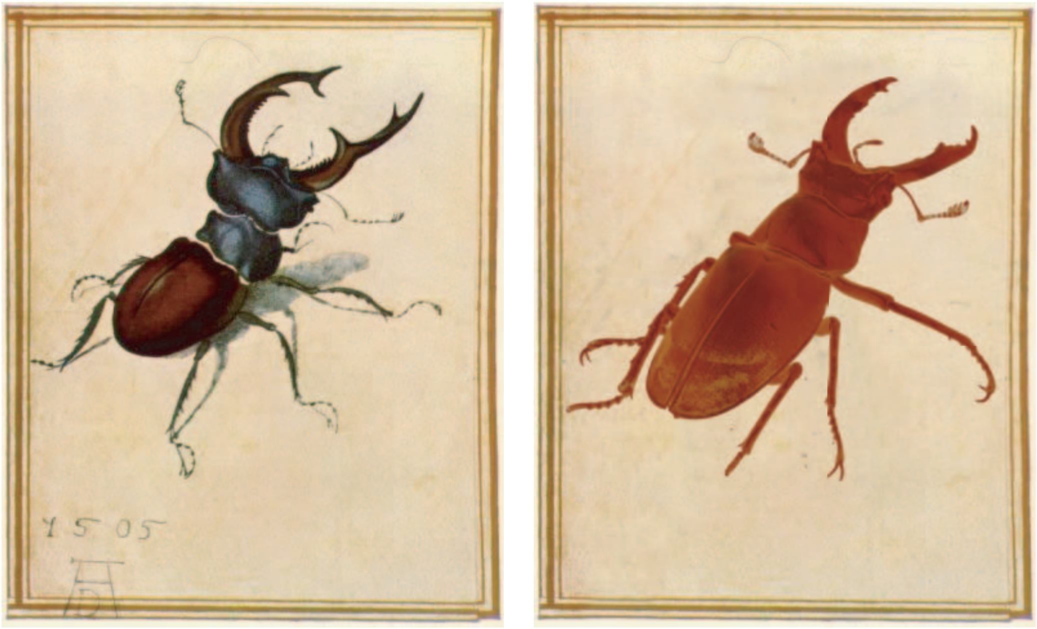
5.7 Conclusion and Future Work

A novel primitive for volumetric data modeling, processing, and rendering has been proposed. As the data representation is moved from a discrete to an implicit representation, a new paradigm is presented. The new function oriented paradigm is a more intuitive and constructive representation of the data. The volumetric data is divided into regions to achieve a more expressive representation. Some Vots represent larger regions than others, but all Vots represent the data in the same way. The size of a Vot can be adjusted by modifying the allowed error bound ϵ. This allows user-centric importance sampling. Unimportant regions are represented by just a few Vots, while important regions are represented with many Vots. One Vot contains all the information about the volumetric data within a region; thus, no explicit connectivity between Vots is necessary for reconstruction. Furthermore, the Vots representation allows to process the data analytically as shown with Maximum Intensity Projection. In general, Vots open a wide range of new data examination approaches. For future work it would be interesting to explore the new possibilities and approaches that Vots allow. One challenge is to construct Vots from other types of data structures, such as point clouds, unstructured grids, curvilinear grids, etc. Different strategies must be developed to obtain an accurate conversion. Vots provide a starting point for new types of volume processing, new ways of rendering, even exploring new types of visualizations based on the condensed information that a Vot contains. It would also be interesting to map Vots onto graphics hardware and analyze their capabilities from that point of view.

A related interesting area is to investigate a point cloud representation of volume data. Moving Least Square (MLS) could be used to dynamically fit a Vot to an arbitrary resampling position. In this case the fitting process of Section 5.4.2 will be slightly modified as each point is weighted by the distance to the resampling position. Similar operations as done for MLS fitting of surfaces can also be done for MLS fitting of volumetric functions, such as point cloud thinning, importance-based sorting of the point cloud, etc.

Chapter 6

Conclusion



(a)

(b)

Figure 6.1: Comparison between hand drawing and volume rendering: (a) Dürer, Albrecht (1471-1528): Der Hirschkäfer (1505). (b) Direct volume rendering: CT scan of a stag beetle (2005) (1005x1005x748, 16 bit, courtesy of J. Kastner, Wels College of Engineering).

> *Don't fear failure so much that you refuse to try new things. The saddest summary of a life contains three descriptions: could have, might have, and should have.*
>
> Louis E. Boone

In this dissertation several new techniques have been presented to handle the efficient visualization of large data on commodity PC hardware. Special focus was put on volume rendering, memory management, parallel processing, and on efficiently skipping parts of the volume that do not contribute to the visualization results.

It has been shown that efficient memory management is fundamental to achieve high performance and to be able to handle large amounts of data. Furthermore, it has been demonstrated that well-known parallelization processing schemes for large parallel systems can be adapted and extended to exploit evolving technologies, such as simultaneous multi-threading. A bricked volume layout has been utilized in order to design a highly efficient threading scheme that maximizes the benefits of thread-level parallelism. The high cache coherency inherently present in a bricked volume layout combined with the two presented refined addressing schemes significantly reduced the costs for resampling and gradient computation. The new addressing scheme can be used for any volume processing algorithm that has to address adjacent samples.

A full-blown high-quality raycasting system for large data has been discussed, using the presented memory layout for volumes and the highly efficient data addressing and processing scheme. The core acceleration techniques are a refined caching scheme for gradient estimation in conjunction with a hybrid skipping and removal of transparent regions to reduce the amount of data to be processed. Due to the efficient memory consumption of the acceleration data structures (quantized binary histogram + granular resolution octree + Cell Invisibility Cache) and the bricked volume layout the system is able to handle very large data. All acceleration structures require only an extra storage of approximately 10%. Data sizes up to 3 GB can be processed, which is a limitation imposed by the virtual address space of current consumer operating systems. The memory efficient data structures enable the system to achieve a several frames per second performance even for large data sets. A key point of this dissertation has been the demonstration that standard computers without advanced graphics hardware are able to achieve the performance necessary for real-world medical applications. Is has been demonstrated that for large medical data, such as computed tomographic (CT) angiography run-offs (512x512x1202, 16 bit), rendering times of up to 2.5 fps can be achieved on a commodity notebook. Real-time rendering of large data on commodity notebooks is within reach. In future work, out-of-core techniques and compression methods will be investigated to permit the processing of even larger data sets. The system is not only capable of rendering large data, it also supports all the corresponding standard features necessary for a general purpose medical imaging system, such as cutting planes, segmented objects, separate transfer functions, and any combination thereof, see Figure 6.2. It is integrated into a commercially available medical workstation and its usage has been proved in practice.

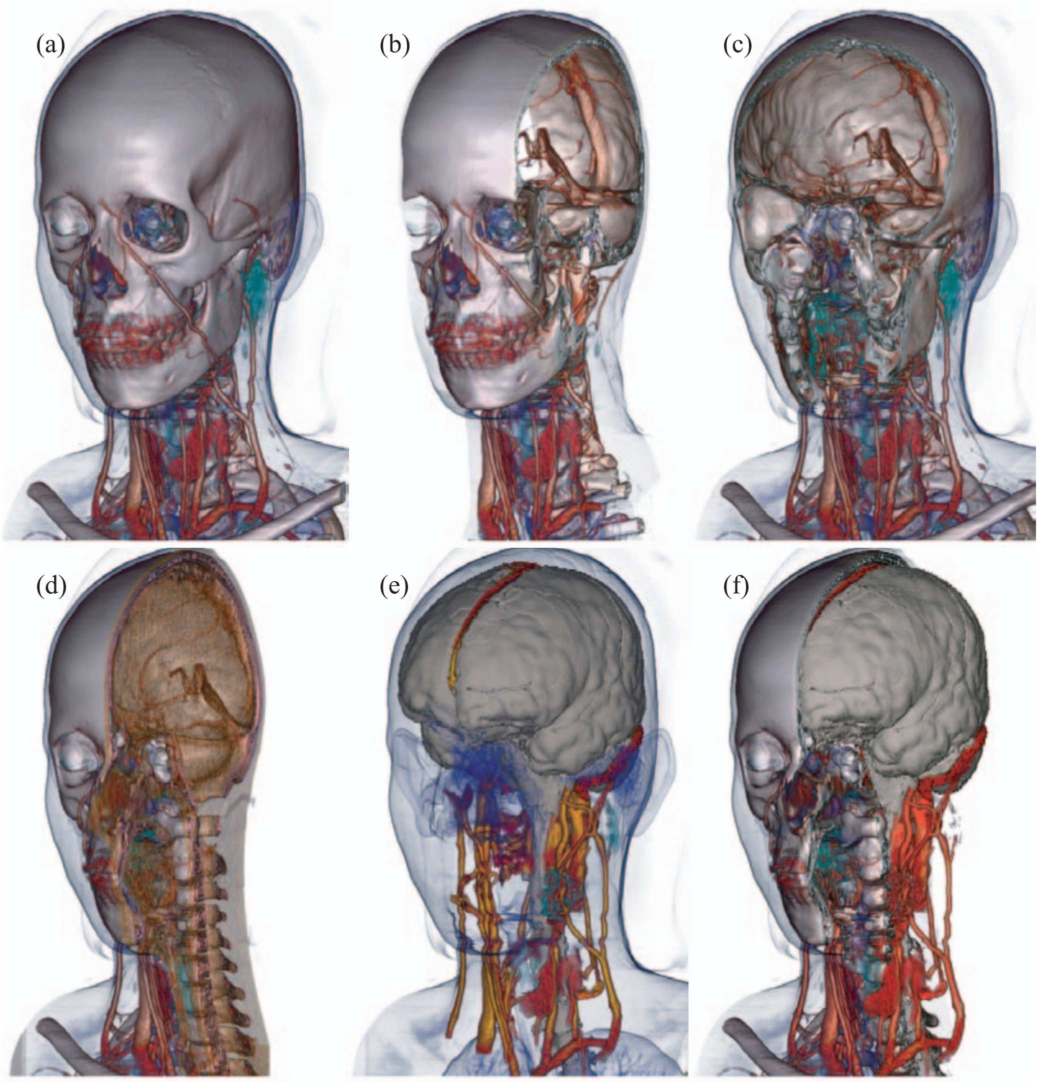


Figure 6.2: Integrated Features of the high quality raycasting system, CT scan of a human head (512x512x333, 16 bit): (a) High quality rendering. (b) Axis aligned cutting planes. (c) Viewing aligned cutting planes. (d) Transfer functions on cutting planes. (e) Segmented objects with separate transfer functions. (f) Segmented objects with separate cutting planes.

Due to the low memory footprint of the acceleration structures, the system is also capable of efficiently handling multiple large data sets. An acceleration technique for direct volume rendering of scenes composed of multiple volumetric objects has been presented. The idea is to distinguish between regions of intersection, which need costly multi-volume processing, and regions containing only one volumetric object, which can be processed using the presented highly efficient mono-volume rendering system. Furthermore, V-Objects, a concept of modeling scenes consisting of multiple volumetric objects, have been presented. V-Objects in combination with direct volume rendering, are a promising technique for visualizing medical data. Different examples for its application,i.e., browsing, time varying data, and multimodal data have been presented. Each of example illustrated that the concept of V-Objects can provide advanced means to explore and investigate data.

In the second part of the dissertation, an alternative to grid-based volume graphics: Volume Dots (Vots), a point-based representation of volumetric data, has been investigated. It is a novel primitive for volumetric data modeling, processing, and rendering. The data representation has been moved from a discrete to an implicit representation. The novel, function oriented approach is a more intuitive and constructive representation of the data. The volumetric data is divided into regions to achieve a more expressive representation. Some Vots represent larger regions than others, but all Vots represent the data in the same way. The size of a Vot can be adjusted by modifying the allowed error bound ϵ. This allows user-centric importance sampling. Unimportant regions are represented by just a few Vots, while important regions are represented by a larger number of Vots. One Vot contains all the information about the volumetric data within a region; thus, no explicit connectivity between Vots is necessary for reconstruction. Furthermore, the Vots representation allows to process the data analytically as shown with Maximum Intensity Projection. In general, Vots open a wide range of new data examination approaches. In the future, it would be interesting to further explore the possibilities and approaches that Vots introduce. One challenge is to construct Vots from other types of data structures, such as point clouds, unstructured grids, and curvilinear grids. Different strategies must be developed to obtain an accurate conversion.

6.1 Visualization Results

Finally, several rendering examples of the high-quality raycasting system are shown, Figure 6.1, and Figure 6.3 to Figure 6.10.

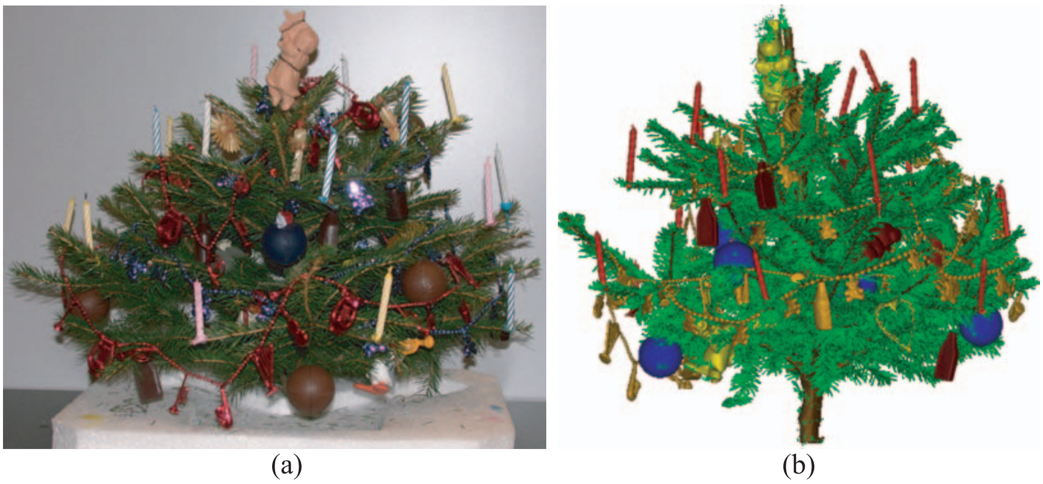


Figure 6.3: Comparison between real data and direct volume rendering: (a) Photograph of xmas tree. (b) CT scan of xmas tree (512x512x999, 16 bit, courtesy of Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria).

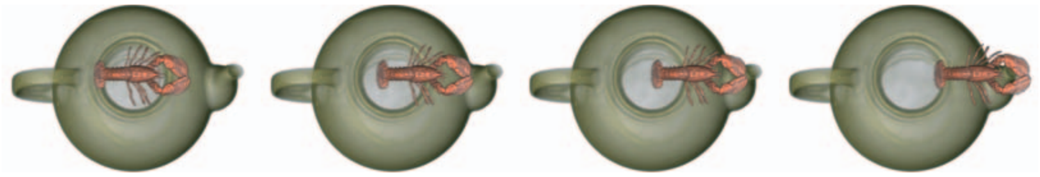


Figure 6.4: Fleeing lobster: CT scan of a teapot with lobster inside (256x256x178, 16 bit, courtesy of Terarecon Inc, MERL, Brigham and Women's Hospital).

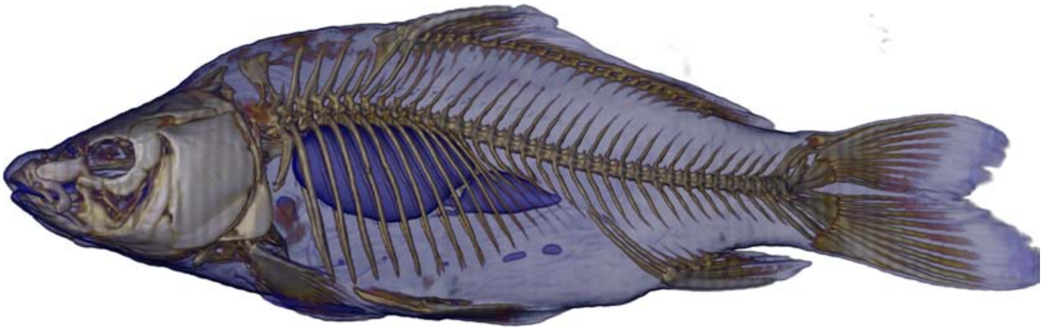


Figure 6.5: CT scan of a carp (256x256x512, 16 bit, courtesy of Michael Scheuring, Computer Graphics Group, University of Erlangen, Germany).

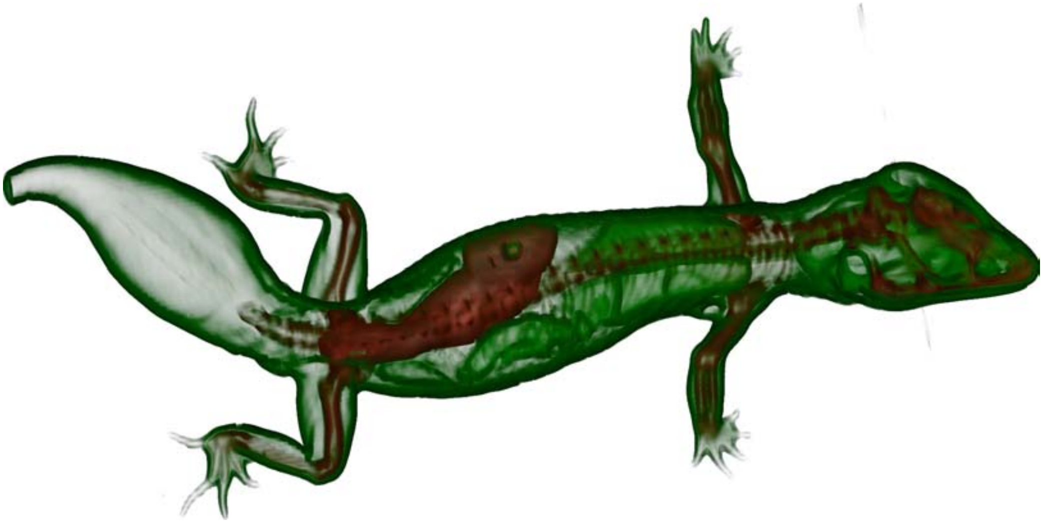


Figure 6.6: CT scan of a gecko (512x512x88, 16 bit, courtesy of University of Veterinary Medicine Vienna).

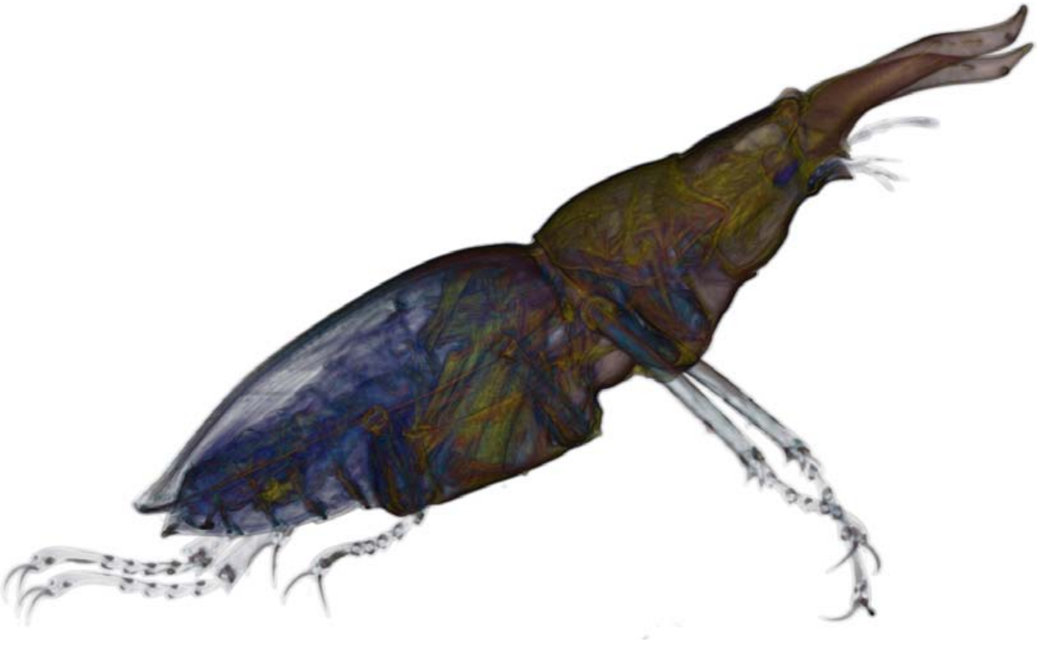


Figure 6.7: CT scan of a stag beetle (1005x1005x748, 16 bit, courtesy of J. Kastner, Wels College of Engineering).

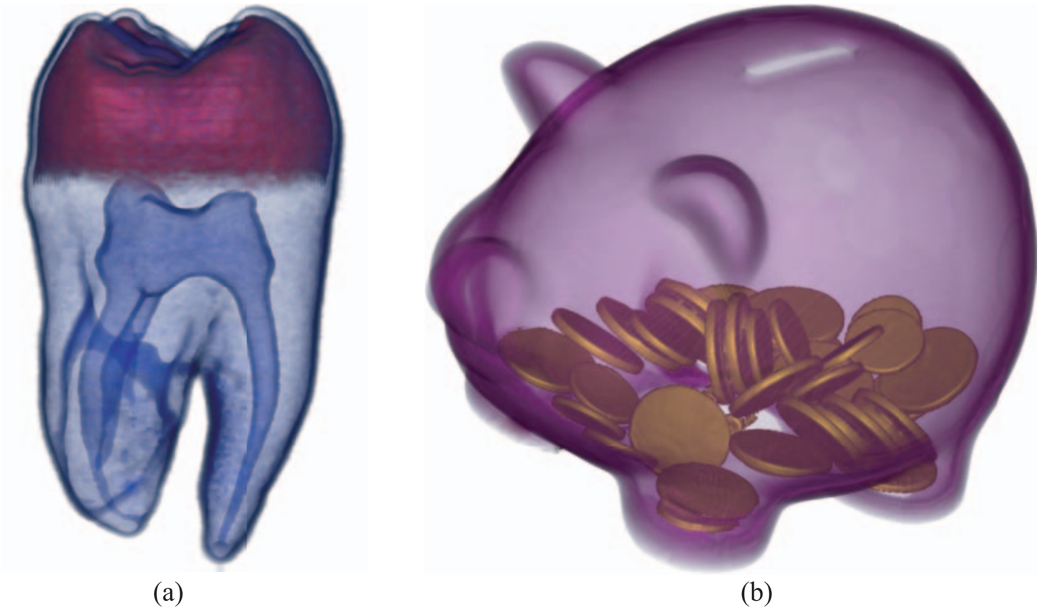


Figure 6.8: CT scans: (a) Tooth (256x256x161, 16 bit, courtesy of GE Aircraft Engines, Evendale, Ohio, USA). (b) Piggy bank (512x512x134, 16 bit, courtesy of Michael Bauer, Computer Graphics Group, University of Erlangen, Germany).

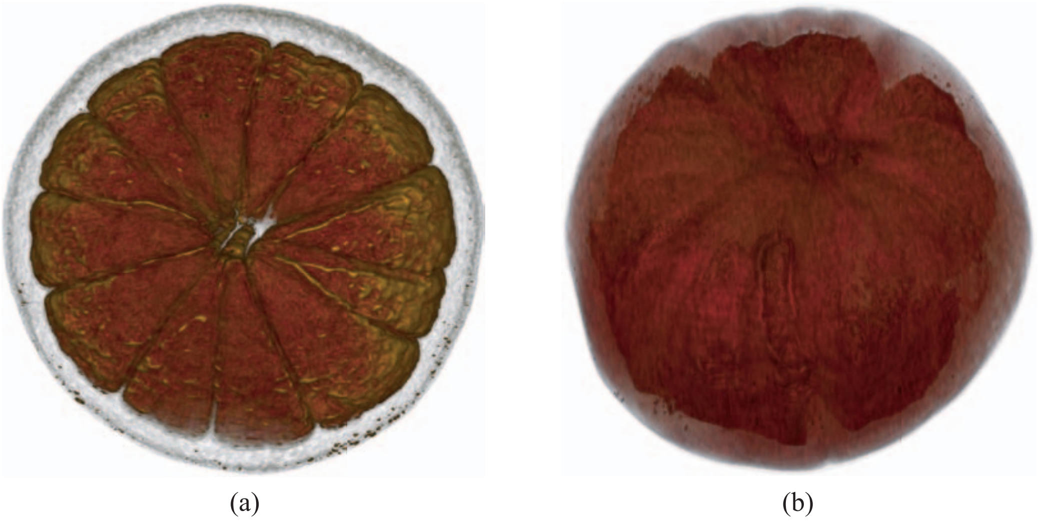


Figure 6.9: MRI scans: (a) Orange (256x256x64, 8 bit). (b) Tomato (256x256x64,8 bit). Courtesy of Bill Johnston and Wing Nip Information and Computing Sciences Division, Lawrence Berkeley Laboratory, USA).

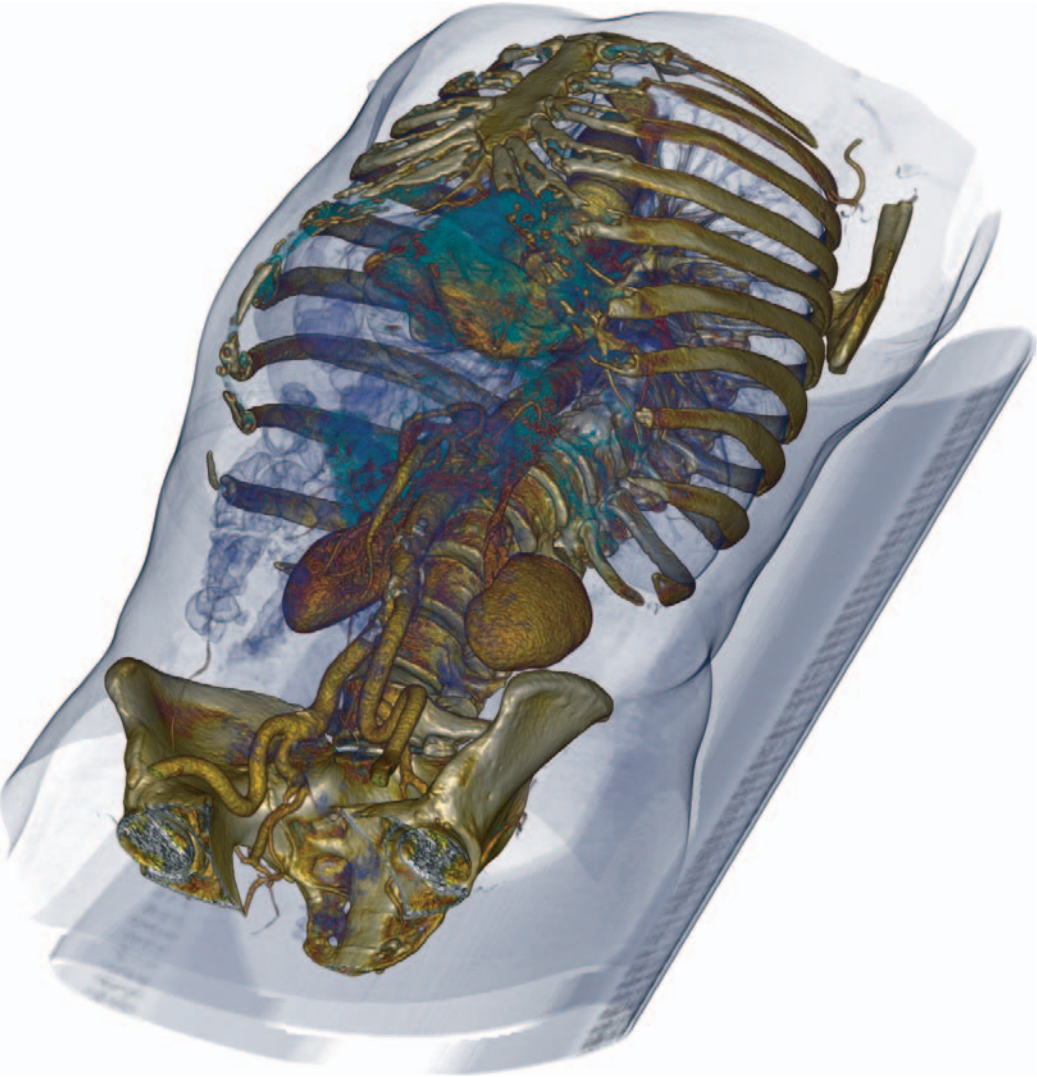


Figure 6.10: CT scan of a human torso (512x512x1112, 16 bit, courtesy of Tiani MedGraph AG).

> *Every day you may make progress. Every step may be fruitful. Yet there will stretch out before you an ever-lengthening, ever-ascending, ever-improving path. You know you will never get to the end of the journey. But this, so far from discouraging, only adds to the joy and glory of the climb.*
>
> ---
>
> Sir Winston Churchill (1874 - 1965)

Bibliography

[1] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. Point set surfaces. In *IEEE Visualization* (2001), pp. 21–28.

[2] Blinn, J. F. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics 16*, 3 (1982), 21–29.

[3] Bruckner, S., Grimm, S., Kanitsar, A., and Gröller, E. Illustrative context-preserving volume rendering. In *Eurographics / IEEE VGTC Symposium on Visualization* (2005). to appear.

[4] Cabral, B., Cam, N., and Foran, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization* (1994), pp. 91–98.

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

[6] Cameron, G. G., and Undrill, P. E. Rendering volumetric medical image data on a SIMD architecture computer. In *Eurographics Workshop on Rendering* (1992), pp. 135–145.

[7] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction and representation of 3D objects with radial basis functions. *Computer Graphics 28*, Annual Conference Series (2001), 67–76.

[8] Csebfalvi, B., and Szirmay-Kalos, L. Monte carlo volume rendering. In *IEEE Visualization* (2003), pp. 449–456.

[9] Dachille, F., Kreeger, K., Chen, B., Bitter, I., and Kaufman, A. E. High quality volume rendering using texture mapping hardware. In *SIGGRAPH/Eurographics workshop on graphics hardware* (1998), pp. 69–76.

[10] Elvins, T. T. A survey of algorithms for volume visualization. *Computer Graphics 26*, 3 (1992), 194–201.

[11] Gelder, A. V., and Kim, K. Direct volume rendering with shading via three-dimensional textures. In *Symposium on Volume Visualization* (1996), pp. 23–30.

[12] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, E. Flexible direct multi-volume rendering in dynamic scenes. In *Workshop on Vision, Modeling, and Visualization* (2004), pp. 379–386.

[13] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, E. Memory efficient acceleration structures and techniques for CPU-based volume raycasting of large data. In *IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics* (2004), pp. 1–8.

[14] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, E. A refined data addressing and processing scheme to accelerate volume raycasting. *Computers and Graphics 28*, 5 (2004), 719–729.

[15] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, E. Vots:volume dots as a point-based representation of volumetric data. *Computer Graphics Forum 23*, 5 (2004), 661–668.

[16] Grimm, S., Meissner, M., Kanitsar, A., and Gröller, E. Parallel peeling of curvilinear grids. Tech. Rep. TR-186-2-04-07, Vienna University of Technology, 2004.

[17] Günther, T., Poliwoda, C., Reinhart, C., Hesser, J., Männer, R., Meinzer, H.-P., and Baur, H.-J. VIRIM: A massively parallel processor for real-time volume visualization in medicine. *Computers & Graphics 19*, 5 (1995), 705–710.

[18] Guthe, S., Wand, M., Gonser, J., and Strasser, W. Interactive rendering of large volume data sets. In *IEEE Visualization* (2002), pp. 53–60.

[19] Hadwiger, M., Berger, C., and Hauser, H. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *IEEE Visualization* (2003), pp. 301–308.

[20] Hopf, M., and Ertl, T. Hierarchical splatting of scattered data. In *IEEE Visualization* (2003), pp. 433–440.

[21] Huang, J., Müller, K., Shareef, N., and Crawfis, R. Fast splats: optimized splatting on rectilinear grids. In *IEEE Visualization* (2000), pp. 219–226.

[22] Intel Cooperation. *IA-32 Intel Architecture Software Developer's Manual: Volume I, II, III.* Intel, Order Numbers: 245470-010, 245472-010, and 245472-010, 2003.

[23] Intel Cooperation. *Intel Pentium 4 and Intel Xeon Prozessor Optimization, Reference Manual.* Intel, Order Number 248966-007, 2003.

[24] Kaufman, A. E., Cohen, D., and Yagel, R. Volume graphics. *IEEE Computer 26*, 7 (1993), 51–64.

[25] Knittel, G. A scalable architecture for volume rendering. *Computers and Graphics 19*, 5 (1995), 653–665.

[26] Knittel, G. The Ultravis system. In *IEEE Symposium on Volume visualization* (2000), pp. 71–79.

[27] Koufaty, D., and Marr, D. T. Hyper-threading technology in the netburst microarchitecture. *IEEE Micro 23*, 2 (2003), 56–65.

[28] Kreeger, K. A., and Kaufman, A. E. Mixing translucent polygons with volumes. In *IEEE Visualization* (1999), pp. 191–198.

[29] Krueger, W. The application of transport theory to visualization of 3D scalar data fields. In *IEEE Visualization* (1990), pp. 273–280.

[30] Krüger, J., and Westermann, R. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization* (2003), pp. 287–292.

[31] Lacroute, P. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation.* PhD thesis, Stanford University, Computer Systems Laboratory, 1995.

[32] Lacroute, P., and Levoy, M. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics 28*, Annual Conference Series (1994), 451–458.

[33] Law, A., and Yagel, R. Multi-frame trashless ray casting with advancing ray-front. In *Graphics Interfaces* (1996), pp. 70–77.

[34] Leu, A., and Chen, M. Modelling and rendering graphics scenes composed of multiple volumetric datasets. *Computer Graphics Forum 18*, 2 (1999), 159–171.

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

[36] Levoy, M. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications 10*, 2 (1990), 33–40.

[37] Levoy, M., and Whitted, T. The use of points as display primitives. Tech. Rep. 85–022, The University of North Carolina at Chapel Hill, Department of Computer Science, 1985.

[38] Lorensen, W., and Cline, H. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH* (1987), pp. 163–169.

[39] Lu, A., Morris, C. J., Ebert, D., Rheingans, P., and Hansen, C. Non-photorealistic volume rendering using stippling techniques. In *IEEE Visualization* (2002), pp. 211–218.

[40] Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A., and Upton, M. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal 6*, 1 (2002), 4–15.

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

[42] McGuffin, M., Tancau, L., and Balakrischnan, R. Using deformations for browsing volumetric data. In *IEEE Visualization* (2003), pp. 401–408.

[43] Meissner, M., Grimm, S., Strasser, W., Packer, J., and Latimer, D. Parallel volume rendering on a single-chip SIMD architecture. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2001), pp. 107–113.

[44] Meissner, M., Hoffmann, U., and Strasser, W. Enabling classification and shading for 3D texture mapping based volume rendering using OpenGL and extensions. In *IEEE Visualization* (1999), pp. 207–214.

[45] Meissner, M., Huang, J., Bartz, D., Mueller, K., and Crawfis, R. A practical evaluation of popular volume rendering algorithms. In *Symposium on Volume Visualization* (2000), pp. 81–90.

[46] Meissner, M., Kanus, U., Wetekam, G., Hirche, J., Ehlert, A., Strasser, W., Doggett, M., Forthmann, P., and Proksa, R. Vizard II, a reconfigurable interactive volume rendering system. In *Eurographics Workshop on Graphics Hardware* (2002), pp. 137–146.

[47] Möller, T., Machiraju, R., Müller, K., and Yagel, R. A comparision of normal estimation schemes. In *IEEE Visualization* (1997), pp. 19–26.

[48] Mora, B., Jessel, J., and Caubet, R. A new object order raycasting algorithm. In *IEEE Visualization* (2002), pp. 107–113.

[49] Mroz, L., Gröller, E., and König, A. Real-time maximum intensity projection. In *Data Visualization* (1999), Eurographics, pp. 135–144.

[50] Müller, K., and Crawfis, R. Eliminating popping artifacts in sheet buffer-based splatting. In *IEEE Visualization* (1998), pp. 293–246.

[51] Müller, K., Möller, T., and Crawfis, R. Splatting without the blur. In *IEEE Visualization* (1999), pp. 363–370.

[52] Müller, K., Shareef, N., Huang, J., and Crawfis, R. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 116–134.

[53] Nadeau, D. R. Volume scene graphs. In *IEEE Symposium on Volume Visualization* (2000), pp. 49–56.

[54] Neumann, L., Csebfalvi, B., König, A., and Gröller, E. Gradient estimation in volume data using 4D linear regression. *Computer Graphics Forum 19*, 3 (2000), 351–358.

[55] Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. Multi-level partition of unity implicits. In *Transactions on Graphics* (2003), ACM, Ed., pp. 463–470.

[56] Osborne, R., Pfister, H., Lauer, H., McKenzie, N., Gibson, S., Hiatt, W., and Ohkami, T. EM-Cube: An architecture for low-cost real-time volume rendering. In *Workshop on Graphics Hardware* (1997), pp. 131–138.

[57] Parker, S., Shirley, P., Livnat, Y., Hansen, C., and Sloan, P. Interactive ray tracing for isosurface rendering. In *IEEE Visualization* (1998), pp. 233–238.

[58] Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., and Seiler, L. The VolumePro real-time ray-casting system. In *SIGGRAPH* (1999), pp. 251–260.

[59] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. Surfels: Surface elements as rendering primitives. In *SIGGRAPH* (1987), pp. 335–342.

[60] Phong, B.-T. Illumination for computer generated pictures. *Communications of the ACM 18*, 6 (1975), 311–317.

[61] Porter, T., and Duff, T. Compositing digital images. *Computer Graphics 18*, 3 (1984), 253–259.

[62] Qu, H., Kaufman, A. E., Shao, R., and Kumar, A. A framework for sample-based rendering with O-buffers. In *IEEE Visualization* (2003), pp. 441–448.

[63] Ray, H., Pfister, H., Silver, D., and Cook, T. A. Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 210–223.

[64] Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., and Ertl, T. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Workshop on Graphics Hardware* (2000), pp. 109–118.

[65] Roettger, S., Guthe, S., Weiskopf, D., Ertl, T., and Strasser, W. Smart hardware-accelerated volume rendering. In *Symposium on Data Visualisation* (2003), pp. 231–238.

[66] Rössl, C., Zeilfelder, F., Nürnberger, G., and Seidel, H.-P. Visualization of volume data with quadratic super splines. In *IEEE Visualization* (2003), pp. 393–400.

[67] Rubin, G. D., Schmidt, A. J., Logan, L. J., and Sofilos, M. C. Multi detector row CT angiography of lower extremity arterial inflow and runoff: Initial experience. *Radiology 221* (2001), 146–158.

[68] Sobierajski, L., and Avila, R. A hardware acceleration method for volumetric ray tracing. In *IEEE Visualization* (1995), pp. 27–34.

[69] Srinivasan, R., Fang, S., and Huang, S. Volume rendering by template-based octree projection. In *Eurographics Workshop on Visualization in Scientific Computing* (1997), pp. 155–163.

[70] Tiede, U., Hoehne, K. H., Bomans, M., Pommert, A., Riemer, M., and Wiebecke, G. Investigation of medical 3D-rendering algorithms. *Computer Graphics and Applications 10*, 2 (1990), 41–53.

[71] Turk, G., and O'brien, J. F. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (2002), 855–873.

[72] Viola, I., Kanitsar, A., and Gröller, M. E. Importance-driven volume rendering. In *IEEE Visualization* (2004), pp. 139–145.

[73] Šrámek, M., and Kaufman, A. E. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Transactions on Visualization and Computer Graphics 6*, 3 (2000), 236–252.

[74] Weiler, M., Kraus, M., Merz, M., and Ertl, T. Hardware-based ray casting for tetrahedral meshes. In *IEEE Visualization* (2003), pp. 333–340.

[75] Weiskopf, D., Engel, K., and Ertl, T. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003), 298–312.

[76] Welsh, T., and Mueller, K. A frequency-sensitive point hierarchy for images and volumes. In *IEEE Visualization* (2003), pp. 425–432.

[77] Westermann, R., and Ertl, T. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH* (1998), pp. 169–177.

[78] Westover, L. Footprint evaluation for volume rendering. *Computer Graphics 24*, 4 (1990), 367–376.

[79] Westover, L. *SPLATTING: A Parallel Feed-Forward Volume Rendering Algorithm.* PhD thesis, University of North Carolina at Chapel Hill, 1991.

[80] Wilhelms, J., and Gelder, A. V. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (1992), 201–227.

[81] Wilson, B., Ma, K.-L., and Cormick, P. S. M. A hardware-assited hybrid rendering technique for interactive volume visualization. In *IEEE Symposium on Volume Visualization and Graphics* (2002), pp. 123–130.

[82] Woodring, J., Wang, C., and Shen, H. High dimensional direct rendering of time-varying volumetric data. In *IEEE Visualization* (2003), pp. 417–414.

[83] Xie, H., Wang, J., Hua, J., Qin, H., and Kaufman, A. E. Piecewiese C^1 continous surface reconstruction of noisy point clouds via local implicit quadric regression. In *IEEE Visualization* (2003), pp. 91–98.

[84] Yagel, R., Cohen, D., and Kaufman, A. Normal estimation in 3D space. In *Visual Computer* (1992), pp. 278–291.

[85] Zuiderveld, K. J., Koning, A. H. J., and Viergever, M. A. Acceleration of ray-casting using 3D distance transforms. In *Visualization in Biomedical Computing* (1992), pp. 324–335.

[86] Zwicker, M., Pfister, H., Baar, J. V., and Gross, M. EWA splatting. *IEEE Transaction on Visualization and Computer Graphics, 8*, 3 (2002), 223–238.

Appendix A

Curriculum Vitae

Dipl. Inform. Sören Grimm,
born on the 16th of February 1973, in Stuttgart, Germany.

Education

Oct, 2002-	**Ph.D. student in Informatics**, Vienna University of Technology, Austria.
2000	**Diploma in Informatics (Diplom Informatiker)**, Karl-Eberhard University of Tübingen, Germany. Diploma thesis was published in Proceedings of PVG 2001 [43].
1996	**Pre-Diploma in Informatics**, Karl-Eberhard University of Tübingen, Germany.
1993-2000	Graduate studies in Informatics, Karl-Eberhard University of Tübingen, Germany.
1993	**Diploma from secondary school (Abitur)**, Ferdinand-Von-Steinbeis Gymnasium, Tuttlingen, Germany.
1991-1993	Secondary school, Ferdinand-Von-Steinbeis Gymnasium, Tuttlingen, Germany.
1988-1990	Secondary school, Otto-Hahn Gymnasium, Tuttlingen, Germany.
1987-1987	Secondary school, Remstal Gymnasium, Weinstadt, Germany.
1984-1986	Secondary school, Lise-Meitner Gymnasium, Böblingen, Germany.
1984	**Diploma from elementary school**, Österfeld Schule, Vaihingen, Germany.
1980-1984	Elementary school, Österfeld Schule, Vaihingen, Germany.

Professional Experience

2002- **Research Associate** at Vienna University of Technology, Austria.
2001-2002 **Senior Software Engineer and Researcher** at Viatronix Inc., USA.
1999-2000 **Contractor** at Rother & Partner, Germany.
1997-1999 **Contractor** at Suedwest Landesbank, Germany.

Patents

Co-inventor of pending US patent in the field of medical image visualization.

Publications

Published several papers in journals, conferences, symposia, and workshops. For a detailed list, see bibliography.

Professional Activities

Conference, Symposia, Workshops Referee

- ACM SIGGRAPH Conference.
- IEEE Visualization Conference.
- Winter School of Computer Graphics Conference.
- Eurographics/IEEE VGTC Symposium on Visualization.
- Brazilian Symposium on Computer Graphics and Image Processing.
- Workshop on Volume Graphics.
- Workshop on Vision, Modeling, and Visualization.