Technische Universität Wien

DISSERTATION

Interactive Volume-Rendering Techniques for Medical Data Visualization

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

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
E 186
Institut für Computergraphik und Algorithmen

eingereicht an der Technischen Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik

von

M.Sc. CSÉBFALVI Balázs
Matrikelnummer: 0027214
Nobilegasse 26/2/27, A-1150 Wien
geboren am 16. November 1972 in Budapest

Wien, im Mai 2001

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

Kurzfassung

Direktes Volumenrendering ist eine flexible, aber Rechenzeit-intensive Methode für die Visualisierung von 3D Volumsdaten. Wegen der enorm großen Anzahl an Voxeln ("volume elements"), die verarbeitet werden müssen, ist es so gut wie nicht möglich einen hochauflösenden Datenblock mit Hilfe von "brute force" Algorithmen wie klassischem Ray Casting oder Splatting auf einer Singleprozessormaschine interaktiv zu rendern.

Eine Alternative ist, Hardwarebeschleunigung zu verwenden. Bildwiederholfrequenzen in Echtzeit können erreicht werden, wenn man parallele Versionen der Standardalgorithmen auf großen Multiprozessorsystemen ausführt. Diese Lösung ist nicht billig und wird daher auch nicht oft verwendet.

Eine weitere Alternative ist, die Beschleunigung allein mit speziellen Softwaretechniken zu erreichen, um auch auf low-end PCs eine schnelle Volumsdarstellung erreichen zu können. Die Methoden, die diesem Ansatz folgen, verwenden normalerweise einen Vorverarbeitungsschritt um die Volumsdarstellung schneller zu machen. Zum Beispiel können Kohärenzen in einen Datensatz durch Vorverarbeitung ausgenutzt werden.

Software- und Hardware-Beschleunigungsmethoden repräsentieren parallele Richtungen in der Volumenrendering-Forschung, die stark miteinander interagieren. In Hardware implementierte Algorithmen werden oft auch als reine Software-Optimierungen verwendet und üblicherweise werden die schnellsten Softwarebeschleunigungstechniken in Hardware realisiert.

Wenn man die oberen Aspekte bedenkt, folgt diese Arbiet der reinen Software-Richtung. Die neuesten schnellen Volumenrendering Techniken wie der klassische Shear-Warp-Algorithmus oder auf "distance transformation" basierende Methoden beschleunigen die Darstellung, können aber nicht in interaktiven Anwendungen verwendet werden.

Das primäre Ziel dieser Arbeit ist die Anwendungs-orientierte Optimierung existierender Volumenrenderingmethoden, um interaktive Bildwiederholfrequenzen auch auf low-end Rechnern zu ermöglichen. Neue Techniken für traditionelles "alpha-blending rendering", Oberflächenschattierte Darstellung, "maximum intensity projection" (MIP) und schnelle Voransicht mit der Möglichkeit, Parameter interaktiv zu verändern, werden vorgestellt. Es wird gezeigt, wie man die ALU einer Singleprozessorarchitektur anstatt einer Parallelprozessoranordnung verwenden kann, um Voxel parallel zu verarbeiten. Die vorgeschlagene Methode führt zu einem allgemeinen Werkzeug, das sowohl "alpha-blending rendering" als auch "maximum intensity projection" unterstützt.

Weiters wird untersucht, wie man die zu verarbeitenden Daten, abhängig von der verwendeten Renderingmethode, reduzieren kann. Zum Beispiel, verschiedene Vorverarbeitungsstartegien für interactive Iso-Flächendarstellung und schnelle Voransicht, basierend auf einem vereinfachten Visualisierungsmodell, werden vorgeschlagen.

Da die in dieser Arbeit präsentierten Methoden kein Supersampling unterstützen, können Treppenstufenartefakte in den generierten Bildern entstehen. Um diesen Nachteil zu kompensieren, wird ein neues Gradientenschätzungsverfahren, welches eine glatte Gradientenfunktion liefert, vorgestellt.

Abstract

Direct volume rendering is a flexible but computationally expensive method for visualizing 3D sampled data. Because of the enormous number of voxels (volume elements) to be processed, it is hardly possible to interactively render a high-resolution volume using brute force algorithms like the classical ray casting or splatting on a recent single-processor machine.

One alternative is to apply hardware acceleration. Real-time frame rates can be achieved by running the parallel versions of the standard algorithms on large multi-processor systems. This solution is not cheap, therefore it is not widely used.

Another alternative is to develop pure software-only acceleration techniques to support fast volume rendering even on a low-end PC. The methods following this approach usually preprocess the volume in order to make the rendering procedure faster. For example, the coherence inside a data set can be exploited in such a preprocessing.

The software and hardware acceleration methods represent parallel directions in volume-rendering research strongly interacting with each other. Ideas used in hardware devices are often adapted to pure software optimization and usually the fastest software-acceleration techniques are implemented in hardware, like in the case of VolumePro board.

Taking the upper aspects into account, this thesis follows the software-only direction. The recent fast volume-rendering techniques like the classical shear-warp algorithm or methods based on distance transformation speed up the rendering process but they cannot be used in interactive applications.

The primary goal of this thesis is the application-oriented optimization of existing volume-rendering methods providing interactive frame-rates even on low-end machines. New techniques are presented for traditional alpha-blending rendering, surface-shaded display, maximum intensity projection (MIP), and fast previewing with fully interactive parameter control. It is shown how to exploit the ALU of a single-processor architecture for parallel processing of voxels instead of using a parallel-processor array. The presented idea leads to a general tool supporting alpha-blending rendering as well as maximum intensity projection.

It is also discussed how to reduce the data to be processed depending on the applied rendering method. For example, different preprocessing strategies are proposed for interactive iso-surface rendering and fast previewing based on a simplified visualization model.

Since the presented methods do not support supersampling, staircase artifacts can appear in the generated images. In order to compensate this drawback a new gradient estimation scheme is also presented which provides a smooth gradient function.

Acknowledgements

This dissertation is dedicated to everyone who gave me moral, technical, and material support for my Ph.D. studies:

First I would like to express my gratitude to my supervisor Prof. Eduard Gröller who helped me with useful pieces of advise and constructive reviews of this thesis and all the related publications. Special thanks go to Prof. Werner Purgathofer who encouraged me to do my Ph.D. at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology.

Special thanks go also to Prof. László Szirmay-Kalos who was my supervisor during my Ph.D. studies at the Department of Control Engineering and Information Technology of the Technical University of Budapest. He was the one who put me up to the ropes of computer graphics and gave me significant technical support in writing my early publications.

Thanks to László Neumann, Lukas Mroz, Helwig Hauser, Andreas König, and Gábor Márton who were co-authors of my publications for their useful ideas and comments.

Thanks to all the people working at the Institute of Computer Graphics and Algorithms who helped me in my work especially the LaTex expert Jan Prikryl.

I would like to express my gratitude also to my parents who gave me moral support and permanently urged me to write this thesis.

Last but not least very special thanks go to my bride Erika Szalai for being patient while I was working on my papers and this dissertation.

This work has been funded by the V^{is}M^{ed} project (`http://www.vismed.at`). V^{is}M^{ed} is supported by *Tiani Medgraph*, Vienna (`http://www.tiani.com`), and by the *Forschungsförderungsfond für die gewerbliche Wirtschaft* (`http://www.telecom.at/fff/`).

Related publications

This thesis is based on the following publications:

1. B. Csébfalvi and E. Gröller: Interactive Volume Rendering based on a “Bubble Model”, accepted paper for the conference *Graphics Interface*, Ottawa, Canada, 2001.

2. L. Neumann, B. Csébfalvi, A. König, and E. Gröller: Gradient Estimation in Volume Data using 4D Linear Regression, *Computer Graphics Forum (Proceedings EUROGRAPHICS 2000)*, pages 351–358, Interlaken, Switzerland, 2000.

3. B. Csébfalvi, A. König, and E. Gröller: Fast Surface Rendering of Volumetric Data, Proceedings of *Winter School of Computer Graphics*, pages 9–16, Plzen, Czech Republic, 2000.

4. B. Csébfalvi: Fast Volume Rotation using Binary Shear-Warp Factorization, Proceedings of *Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 145–154, Vienna, Austria, 1999.

5. B. Csébfalvi, A. König, and E. Gröller: Fast Maximum Intensity Projection using Binary Shear-Warp Factorization, Proceedings of *Winter School of Computer Graphics*, pages 47–54, Plzen, Czech Republic, 1999.

6. B. Csébfalvi and L. Szirmay-Kalos: Interactive Volume Rotation, *Journal Machine Graphics & Vision*, Vol.7, No.4, pages 793-806, 1998.

7. B. Csébfalvi: An Incremental Algorithm for Fast Rotation of Volumetric Data, Proceedings of *14th Spring Conference on Computer Graphics*, pages 168–174, Budmerice, Slovakia, 1998.

Contents

1 Introduction 8

2 Classification of volume-rendering methods 9
- 2.1 Indirect volume rendering . . . 10
  - 2.1.1 “Marching cubes” - a surface-reconstruction technique . . . 11
  - 2.1.2 Frequency domain volume rendering . . . 15
- 2.2 Direct volume rendering . . . 18
  - 2.2.1 Different visualization models . . . 19
  - 2.2.2 Image-order methods . . . 20
  - 2.2.3 Object-order methods . . . 23

3 Acceleration techniques 25
- 3.1 Fast image-order techniques . . . 26
  - 3.1.1 Hierarchical data structures . . . 26
  - 3.1.2 Early ray termination . . . 26
  - 3.1.3 Distance transformation . . . 27
- 3.2 Fast object-order techniques . . . 28
  - 3.2.1 Hierarchical splatting . . . 28
  - 3.2.2 Extraction of surface points . . . 28
- 3.3 Hybrid acceleration methods . . . 30
  - 3.3.1 Shear-warp factorization . . . 30
  - 3.3.2 Incremental volume rotation . . . 31

4 Interactive volume rendering 32
- 4.1 Fast volume rendering using binary shear transformation . . . 33
  - 4.1.1 Introduction . . . 33
  - 4.1.2 Definition of the segmentation mask . . . 33
  - 4.1.3 Ray casting . . . 34
  - 4.1.4 Binary shear transformation . . . 37
  - 4.1.5 Resampling . . . 39
  - 4.1.6 Shear-warp projection . . . 40
  - 4.1.7 Rotation of large data sets . . . 40
  - 4.1.8 Adaptive thresholding . . . 41
  - 4.1.9 Implementation . . . 42
  - 4.1.10 Summary . . . 43
- 4.2 Fast maximum intensity projection . . . 45
  - 4.2.1 Introduction . . . 45
  - 4.2.2 Encoding of the density intervals . . . 46
  - 4.2.3 Maximum intensity projection (MIP) . . . 47

4.2.4 Local maximum intensity projection (LMIP) . . . 48
4.2.5 Shear-warp projection . . . 49
4.2.6 Extensions . . . 49
4.2.7 Implementation . . . 49
4.2.8 Summary . . . 50
4.3 Interactive iso-surface rendering . . . 51
4.3.1 Introduction . . . 51
4.3.2 Extraction of the potentially visible voxels . . . 51
4.3.3 Shear-warp projection . . . 54
4.3.4 Decomposition of the viewing directions . . . 55
4.3.5 Interactive cutting operations . . . 55
4.3.6 Implementation . . . 57
4.3.7 Summary . . . 59
4.4 Normal estimation based on 4D linear regression . . . 60
4.4.1 Introduction . . . 60
4.4.2 Linear regression . . . 61
4.4.3 Interpolation . . . 63
4.4.4 The weighting function . . . 64
4.4.5 Implementation . . . 65
4.4.6 Summary . . . 68
4.5 Interactive volume rendering based on a “bubble model” . . . 69
4.5.1 Introduction . . . 69
4.5.2 The “bubble model” . . . 70
4.5.3 Interactive rendering . . . 72
4.5.4 Implementation . . . 74
4.5.5 Summary . . . 74

5 Conclusion 77

Bibliography 79

Chapter 1

Introduction

In the last two decades volume visualization became a separate discipline in computer graphics. In the early eighties, as the scanning technologies used in medical imaging like computed tomography (CT) or magnetic resonance imaging (MRI) became more developed and provided higher resolution images, a natural demand arose to process the 2D slices and visualize the entire data set in 3D. Previously the data was analyzed by generating only cross-sectional 2D images of arbitrary directions mapping the data values onto gray levels of the given display device.

Although the sequence of slices carries spatial information the traditional computer graphics techniques are not directly appropriate to visualize the data in 3D. These methods rely on conventional modeling defining the virtual scene by geometrical primitives. In contrast, a volume data set represents the scene by data values available at regular or irregular grid points. The first volume-rendering techniques were developed in order to display medical data but later this research direction leaded to a general visualization tool. Volumes can be obtained by discretizing geometrical models or using any other 3D scanning technology, like measuring geographical data to render different underground layers.

Recently, there are several parallel directions in volume rendering research. Generally, the main problem is how to process the enormous amount of data (a modern CT scanner can provide a slice resolution 512×512 with 16 bits/voxel precision). One approach is to render the data interactively using a specialized multi-processor hardware support. Since these devices are not cheap they are not widely used in practice. Another alternative is to preprocess the volume in order to make the visualization procedure faster. Although there are several software-only acceleration methods volume rendering on low-end machines is still far from interactivity.

This dissertation follows the latter approach, and presents new fast volume visualization techniques which do not require any specialized hardware in order to achieve interactive frame rates. Therefore, they can be widely used in medical imaging systems. Although the main orientation is the medical application area most of the proposed algorithms are considered to be acceleration tools for general volume rendering.

In Chapter 2 the currently existing volume visualization methods are classified into different categories. Chapter 3 gives an overview of recent software-only acceleration techniques analyzing their advantages and disadvantages. Chapter 4 contains the main contribution of this thesis presenting new interactive volume-rendering algorithms based on application-oriented optimization. Finally, in Chapter 5 the new results are summarized.

Chapter 2

Classification of volume-rendering methods

There are two fundamentally different approaches for displaying volumetric data. One alternative is indirect volume rendering, where in a preprocessing step the volume is converted to an intermediate representation which can be handled by the graphics engine. In contrast, the direct methods process the volume without generating any intermediate representation assigning optical properties directly to the volume elements (voxels).

Early indirect methods aim at the visualization of iso-surfaces defined by a certain density threshold. The primary goal is to create a triangular mesh which fits to the iso-regions inside the volume. This can be done using the traditional image-processing techniques, where first of all an edge detection is performed on the slices and afterwards the contours are connected. Having the contours determined the corresponding contour points in the neighboring slices are connected by triangles. This approach requires the setting of many heuristic parameters thus it is not flexible enough to use them in practical applications. A more robust approach is the "marching cubes" iso-surface reconstruction [38], which marches through all the cubic cells and generates an elementary triangular mesh whenever a cell is found which is intersected by an iso-surface. Since the volumetric data defined in the discrete space is converted to a continuous geometrical model, the conventional computer graphics techniques, like ray tracing or z-buffering can be used to render the iso-surfaces.

Recently, a completely new indirect volume-rendering approach has been proposed, where the intermediate representation is a 3D Fourier transform of the volume rather than a geometrical model [39][57][37]. This technique aims at fast density integral calculation along the viewing rays. Since the final image is considered to be an X-ray simulation, this technique is useful in medical imaging applications. The main idea is to calculate the 3D Fourier transform of the volume in a preprocessing step. This transformation is rather expensive computationally but it has to be executed only once independently on the viewing direction. The final image is calculated performing a relatively cheap 2D inverse Fourier transformation on a slice in the frequency domain. This slice is perpendicular to the current viewing direction and passes through the origin of the coordinate system. According to the *Fourier projection-slice theorem* the pixels of the generated image represent the density integrals along the corresponding viewing rays.

Another alternative of volume visualization is to render the data set directly without using any intermediate representation. The optical attributes like a color, an opacity, or an emission are assigned directly to the voxels. The pixel colors depend on the optical properties of the voxels intersected by the corresponding viewing rays. The direct volume-rendering techniques can be classified further into two categories. The object-order methods process the volume voxel-by-voxel projecting them onto the image plane, while the image-order methods produce the image

pixel-by-pixel casting rays through each pixel and resampling the volume along the viewing rays. The direct techniques represent a very flexible and robust way of volume visualization. The internal structures of the volume can be rendered controlled by a transfer function which assigns different opacity and color values to the voxels according to the original data value. Although there is no need to generate an intermediate representation direct volume rendering is rather time-consuming because of the enormous number of voxels to be processed.

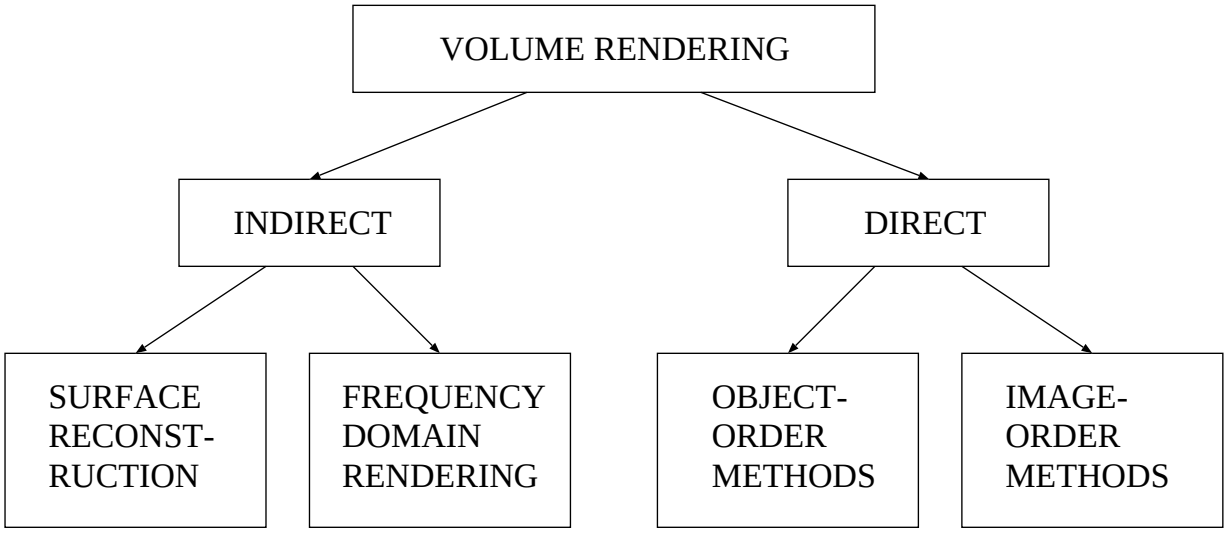


Figure 2.1: Classification of different volume-rendering methods.

Figure 2.1 depicts the classification of recent volume-rendering techniques. The following sections discuss these algorithms in detail presenting one representative method from each family and evaluating their advantages and disadvantages.

2.1 Indirect volume rendering

In this section two indirect methods are presented. The first one is the classical *marching cubes* surface-reconstruction algorithm which converts the discrete volumetric data to a continuous geometrical model. This technique is used for iso-surface rendering, which can be performed in real time using the conventional graphics hardware. The other presented indirect method is the *frequency-domain volume rendering* aiming at fast display of simulated X-ray images from arbitrary viewing directions. Here the intermediate representation is the 3D Fourier transform of the volume which is considered to be a continuous 3D density function sampled at regular grid points.

2.1.1 "Marching cubes" - a surface-reconstruction technique

In the eighties the volume-rendering research was mainly oriented to the development of indirect methods. At that time no rendering technique was available which could visualize the volumetric data directly without performing any preprocessing. The existing computer graphics methods, like ray tracing or z-buffering [54] had been developed for geometrical models rather than for volume data sets. Therefore, the idea of converting the volume defined in a discrete space into a geometrical representation seemed to be quite obvious. The early surface-reconstruction methods were based on the traditional image-processing techniques [1][58][60], like edge detection and contour connection. Because of the heuristic parameters to be set these methods were not flexible enough for practical applications. The most important milestone in this research direction was the *marching cubes algorithm* [38] which had been proposed by Lorensen. This method does not rely on image processing performed on the slices and requires only one parameter which is a density threshold defining the iso-surface.

The algorithm is called "marching cubes" since it marches through all the cubic cells and generates a local triangular mesh inside those cells which are intersected by an iso-surface. The main steps of the algorithm are the following:

1. Set a threshold value defining an iso-surface.
2. Classify all the corner voxels comparing the densities with the iso-surface constant.
3. March through all the intersected cells.
4. Calculate an index to a look-up table according to the classification of the eight corner vertices.
5. Using the index look up the list of edges from a precalculated table.
6. Calculate intersection points along the edges using linear interpolation.
7. Calculate unit normals at each cube vertex using central differences. Interpolate the normals to each triangle vertex.

After having an iso-surface defined by a density threshold (Step 1.) all the voxels are investigated whether they are below or above the surface, comparing the densities with the surface constant. This binary classification (Step 2.) assigns the value of one to the voxels of densities higher than the threshold and the value of zero to the other voxels. The algorithm marches through all the intersected cells (Step 3.), where there are at least two corner voxels classified differently. For such cells an index to a look-up table is calculated according to the classification of the corner voxels (Step 4.) (Figure 2.2).

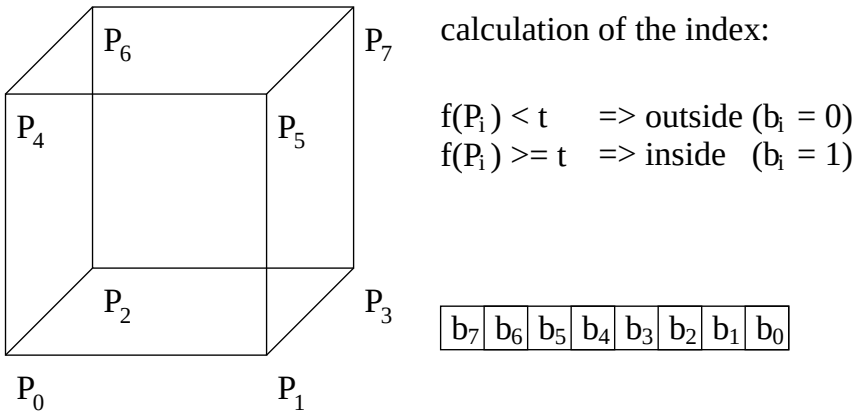


Figure 2.2: Calculation of the index to the look-up table.

The index contains eight bits associated with the eight corner voxels of the cubic cell and their values depend on the classification of the corresponding voxels. This index addresses a look-up table containing all the 256 cases of elementary triangular meshes (Step 5.). Because of symmetry reasons, there are just 14 topologically distinct cases among these patterns thus in practice the look-up table contains 14 entries instead of 256. Figure 2.3 shows the triangulation of the 14 patterns.

After having the list of intersected edges read from the look-up table (Step 6.) the intersection points along these edges are calculated using linear interpolation. The position of vertex V_{ij} along the edge connecting corner points P_i and P_j is computed as follows:

$$V_{ij} = ((t - f(P_i)) \cdot P_j + (f(P_j) - t) \cdot P_i))/(f(P_j) - f(P_i)), \tag{2.1}$$

assuming that $f(P_i) < t$ and $f(P_j) > t$, where f is the spatial density function and t is the threshold defining the iso-surface.

Since the algorithm generates an oriented surface with a normal vector at each vertex position, the last step is the calculation of the surface normals (Step 7.). First of all, the normals $n(x_i, y_j, z_k)$ at the cube vertices are determined using *central differences*:

$$n(x_i, y_j, z_k) \approx \frac{1}{2} \cdot \begin{bmatrix} (f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)) \\ (f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ (f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \end{bmatrix}. \tag{2.2}$$

At an intersection point V_{ij} along the edge connecting grid points P_i and P_j the surface normal N_{ij} is calculated using linear interpolation between the corresponding normals denoted by N_i and N_j respectively:

$$N_{ij} = ((t - f(P_i)) \cdot N_j + (f(P_j) - t) \cdot N_i))/(f(P_j) - f(P_i)). \tag{2.3}$$

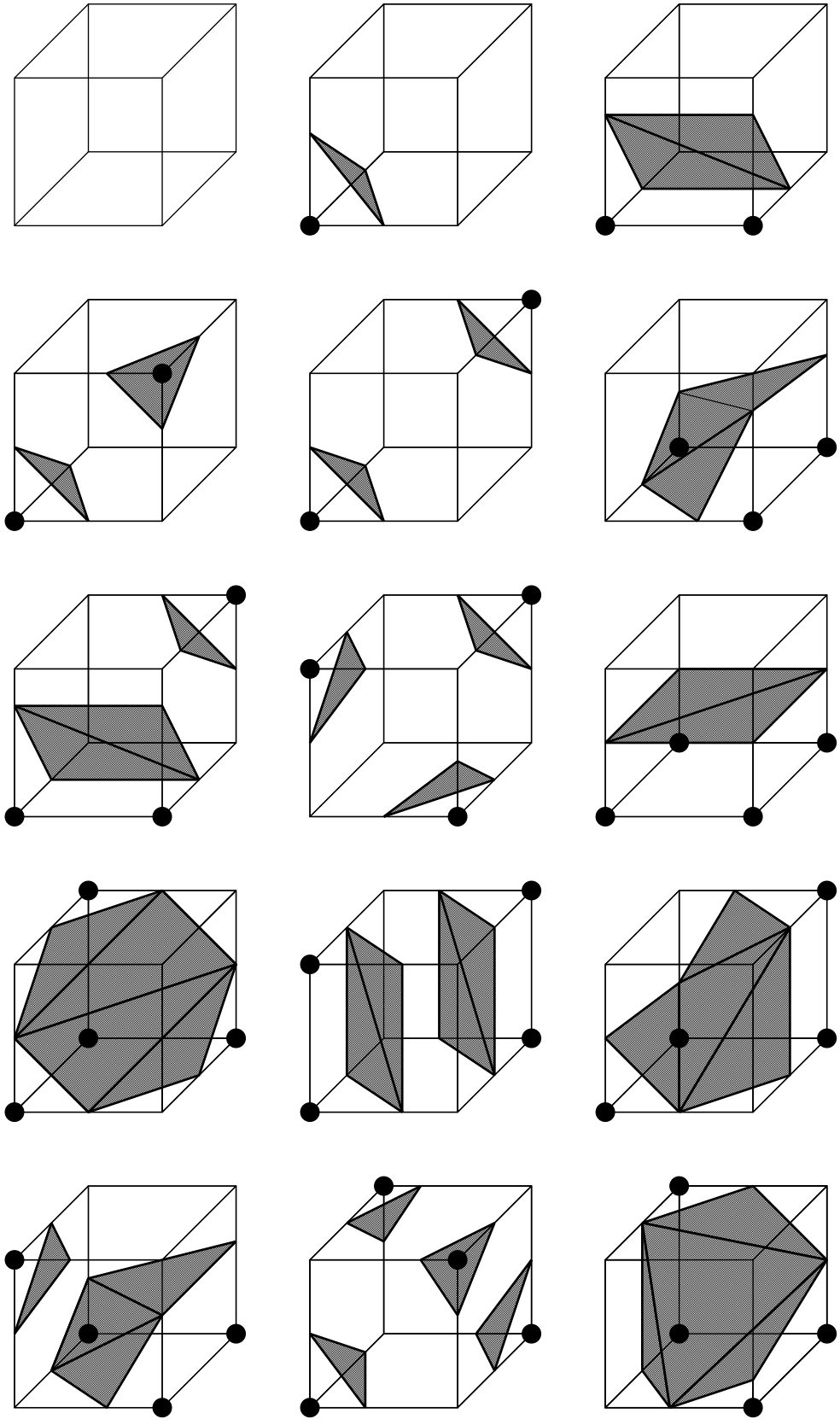


Figure 2.3: The triangulated cells.

The continuous geometrical model generated by the marching cubes algorithm can be rendered using the traditional computer graphics techniques. The conventional graphics acceleration devices which are based on the hardware implementation of the z-buffering hidden surface removal can render such a model in real time using Phong shading and Gouraud interpolation [54]. Figure 2.4 shows the rendering of the surface reconstructed from the CT scan of a human head. The density constant was set to the threshold of the bone in order to visualize the surface of the skull.

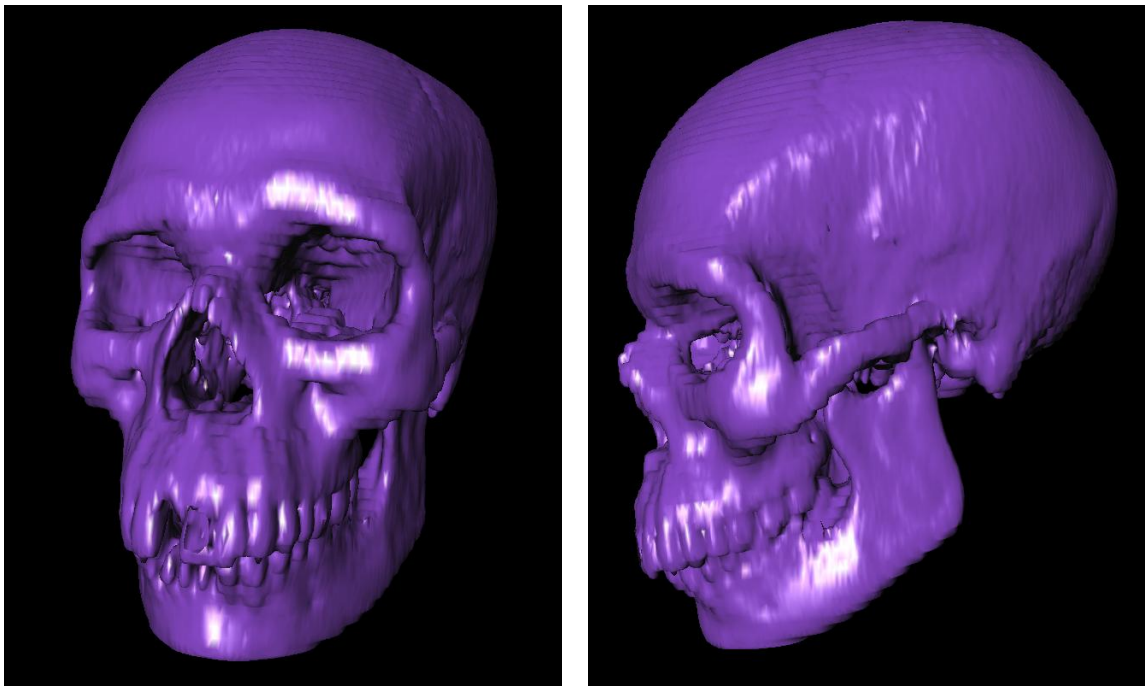


Figure 2.4: The surface of a human skull reconstructed from CT data using the marching cubes algorithm [5].

The main disadvantage of the marching cubes algorithm is the computationally expensive preprocessing. Especially having high resolution data sets the number of the generated triangles can be enormous. Since the interactivity is strongly influenced by the complexity of the model usually some postprocessing is performed on the initial mesh in order to simplify it [51]. Furthermore the triangular mesh is not uniform because the vertices are located on the edges of cubic cells, therefore some mesh refinement is also required.

However, this method is capable only for iso-surface rendering thus the internal structures of the volume cannot be visualized. After the preprocessing, the original data values are not available anymore thus cutting planes are not supported. Cutting operations are rather important in medical imaging applications, where the physician can define an arbitrary cross section of the 3D model and render the slice displaying the original gray-scale data values. Furthermore, the modeling of semi-transparent tissues - which is the most important feature of direct volume rendering - is not supported either.

2.1.2 Frequency domain volume rendering

In this section another indirect volume rendering approach is presented which has been proposed for interactive rendering of simulated X-ray images [39][57][37]. Here the intermediate representation is a 3D Fourier transform of the volume which is considered to be a continuous spatial density function sampled at regular grid points. Although the calculation of the 3D Fourier transform is very time-demanding it has to be performed only once in a preprocessing step. Afterwards, according to the *Fourier projection-slice theorem*, an arbitrary projection of the volume can be generated by a relatively cheap 2D inverse Fourier transformation of a single slice. This inverse transformation can be rapidly executed even on low-end hardware. The final image contains in each pixel the density integral along the corresponding viewing ray, thus it can be considered to be an X-ray simulation.

This approach for rendering volumetric data sets is based on the inverse problem of tomographic reconstruction [39]. Generally, the aim is to achieve a fast computation of the density integral of a given 2D function $f(x, y)$ along a line $l(t, \mu)$. The Fourier projection-slice theorem is given as:

$$P_\theta = F(\omega cos\theta, \omega sin\theta), \tag{2.4}$$

where $P_\theta(\omega)$ is the Fourier transform of

$$p_\theta = \int_{-\infty}^{\infty} f(x(t, \mu), y(t, \mu))dt, \tag{2.5}$$

and (ω, θ) defines the frequency plane. Thus a slice of the Fourier transform of a given function $f(x, y)$ at an angle θ represents the one-dimensional Fourier transform of its projection $p_\theta(\mu)$. Figure 2.5 illustrates this relationship.

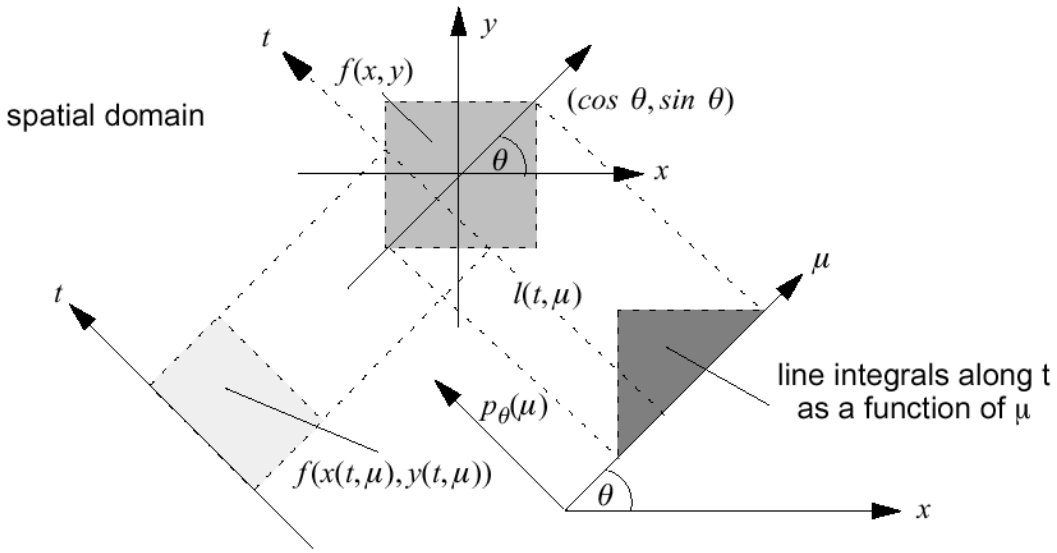


Figure 2.5: Illustration of the Fourier projection-slice theorem in 2D.

In 3D, the Fourier projection-slice theorem can be used for calculating the intensity integrals of a function $f(x, y, z)$ at a given point (μ, ν) onto an oriented projection plane along a

perpendicular ray t. We get the intensity $i(\mu, \nu)$ in the image plane as:

$$i(\mu, \nu) = \int_{-\infty}^{\infty} f(x(t, \mu, \nu), y(t, \mu, \nu), z(t, \mu, \nu))dt. \tag{2.6}$$

The computed intensity distribution and its parameters in spatial domain are illustrated in Figure 2.6.

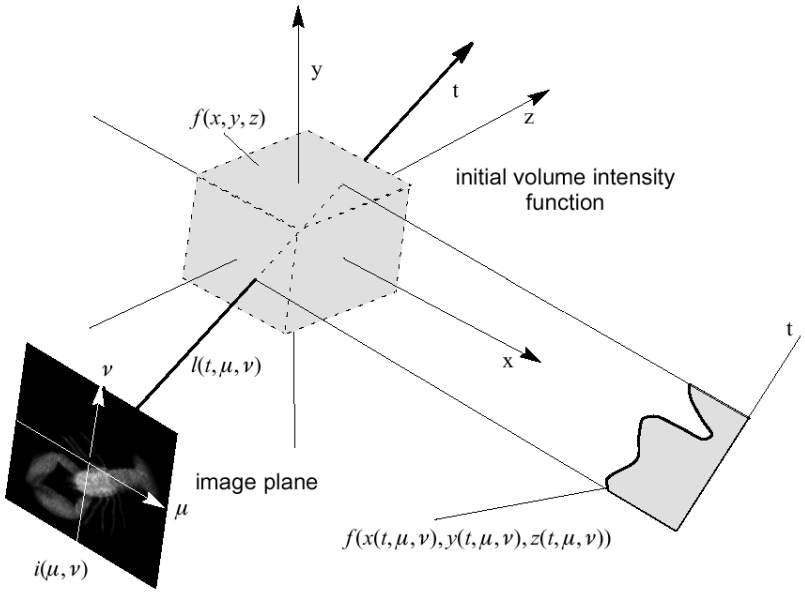


Figure 2.6: Illustration of the Fourier projection-slice theorem in 3D.

The 3D Fourier transform of the spatial density function $f(x, y, z)$ is given by:

$$F(\omega_1, \omega_2, \omega_3) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y, z)e^{-i2\pi(x\omega_1+y\omega_2+z\omega_3)}dxdydz \tag{2.7}$$

According to the Fourier projection-slice theorem [17] $F(\omega_1, \omega_2, \omega_3)$ has to be computed along the slicing plane through the origin defined by two vectors u and v in order to obtain $I(u, v)$.

$$F(\omega_1(u, v), \omega_2(u, v), \omega_3(u, v)) = I(u, v) =$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y, z)e^{-i2\pi(x\omega_1(u,v)+y\omega_2(u,v)+z\omega_3(u,v))}dxdydz \tag{2.8}$$

The inverse FFT of the resulting 2D function $I(u, v)$ leads to the intensity integral (eq. 2.6), of a bundle of parallel rays perpendicular to the plane (u, v). This image is considered to be an X-ray image of the volume. Figure 2.7 shows two example images rendered using this volume rendering technique.

The 2D inverse FFT can be calculated very fast, therefore having the 3D Fourier transform of the entire volume it can be rendered from an arbitrary viewing direction achieving interactive frame rates. Since the final image is just an X-ray simulation this technique is mainly used in

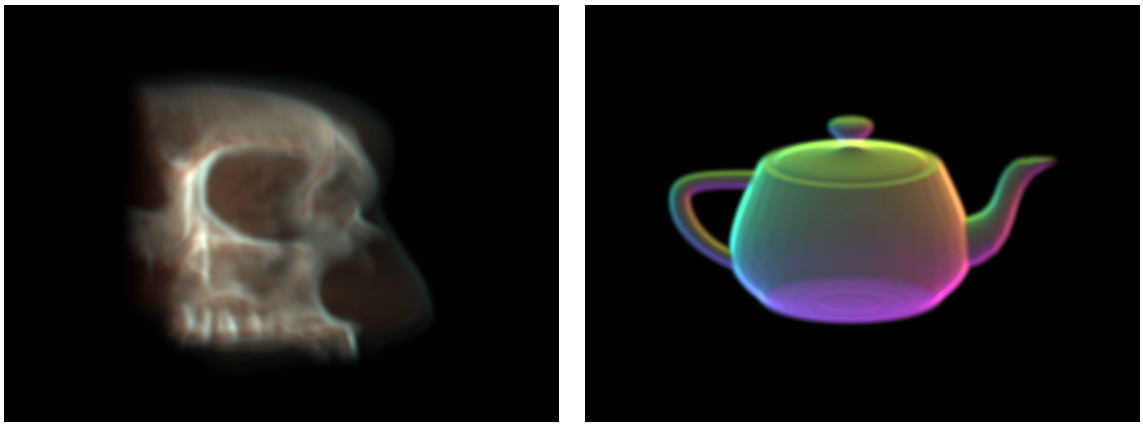


Figure 2.7: Example images rendered using frequency domain volume rendering [57].

medical imaging applications and not considered to be a widely usable general volume rendering method. The main drawback is the lack of depth information, since for the calculation of the density integrals a distance-dependent weighting function or an opacity manipulation cannot be used. In the next section it will be shown that direct volume-rendering methods are much more flexible in this sense supporting the modeling of several optical phenomena like emission, reflection, and attenuation.

2.2 Direct volume rendering

Direct volume-rendering methods do not require any intermediate representation in order to visualize the data set, since it is processed directly in the rendering pipeline. The volume $V : Z^3 \rightarrow R$ is considered to be a spatial density function $f : R^3 \rightarrow R$ sampled at regular grid points, where the density samples are defined as:

$$V_{i,j,k} = f([x_i, y_j, z_k]). \tag{2.9}$$

In order to generate an image of the volume a viewing ray is cast from the virtual camera location through each pixel (picture element) of the image plane (Figure 2.8). The rays can be defined by their parametric equation:

$$\mathbf{r}(t) = \mathbf{x} + \omega \cdot t, \tag{2.10}$$

where location $\mathbf{x}$ is the origin, vector ω is the unit direction and t is the ray parameter. Each viewing ray piercing the volume can be characterized by a density profile p which depends on the density values along the ray segment which is intersected by the volume:

$$p(t) = f(\mathbf{r}(t)) = f(\mathbf{x} + \omega \cdot t). \tag{2.11}$$

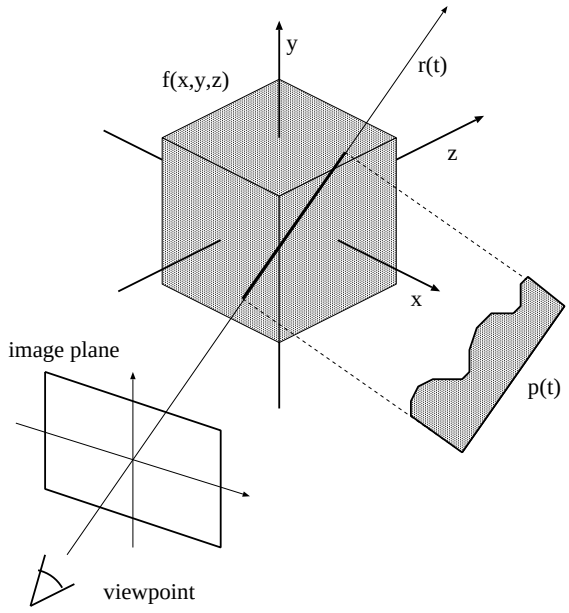


Figure 2.8: Illustration of the density profile of a viewing ray.

Since the volume data is given as a 3D array of samples, the original continuous density function is not available. In order to get the value $p(t)$ of the ray profile at an arbitrary value of parameter t a 3D interpolation method has to be used.

The color of a pixel depends on the density profile of the corresponding ray and the applied visualization model which determines how to map this profile as a one-dimensional function onto a single pixel value. The following subsection overviews the most important recently used volume-rendering models.

2.2.1 Different visualization models

The simplest visualization models directly map the density profile onto pixel intensities. For instance, one possibility is to calculate each pixel value $I(\mathbf{x}, \omega)$ as the *density integral* along the corresponding viewing ray defined by origin $\mathbf{x}$ and direction ω:

$$I(\mathbf{x}, \omega) = \int_t f(\mathbf{x} + \omega \cdot t)dt. \tag{2.12}$$

This model is equivalent with the Fourier volume rendering resulting in simulated X-ray images. Similar visual effect can be achieved approximating the density integrals by the maximum density value along the viewing rays:

$$I(\mathbf{x}, \omega) = max_t\ f(\mathbf{x} + \omega \cdot t). \tag{2.13}$$

This model is well known in medical imaging as *maximum intensity projection* (MIP) and it is mainly used for visualization of blood-vessel structures. Assuming that a trilinear filter is applied for function reconstruction the exact maximum densities can be analytically calculated [49]. In practice the density profile is approximated by a a piecewise constant function taking a finite number of evenly located samples, and the maximum density sample is assigned to the given pixel.

The main drawback of maximum intensity projection is the loss of depth information. For example, in a medical application it might be confusing that a higher density blood vessel can "hide" other blood vessels which are closer to the view-point. In order to avoid this problem Sato proposed a technique called *local maximum intensity projection* (LMIP)[50]. Instead of the global maximum along the corresponding viewing ray the first local maximum which is above a predefined threshold is assigned to each pixel (Figure 2.9).

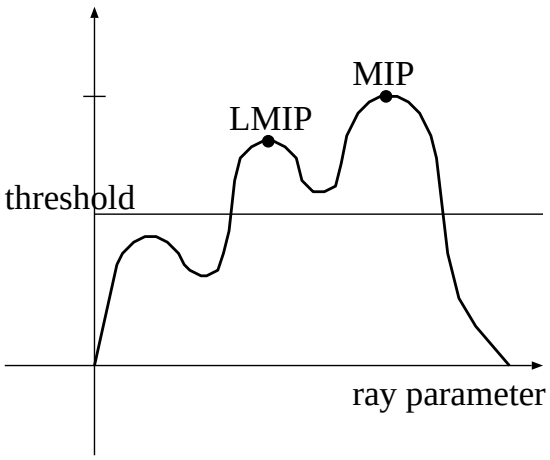


Figure 2.9: Local maximum intensity projection (LMIP).

In order to model physical phenomena like scattering or attenuation optical properties are assigned to the volume samples as functions of their density values. Each pixel intensity $I(\mathbf{x}, \omega)$ is composed from the assigned properties of the samples along the associated viewing ray according to the well known *light transport equation*[34][36][15][40]:

$$I(\mathbf{x}, \omega) = \int_s e^{-\int_0^s \sigma(t)dt} I(s)ds, \tag{2.14}$$

where $\mathbf{x}$ is the origin of the ray, ω is the unit direction of the ray, $\sigma(t)$ is the differential attenuation at $\mathbf{x} + \omega \cdot t$, and $I(s)$ is the differential intensity scattered at $\mathbf{x} + \omega \cdot s$ in the direction

$-\omega$. In practice, the integral of Equation 2.14 is approximated by a summation of equally spaced samples:

$$I = \sum_{0 \le i \le n} (\prod_{0 \le j \le i} e^{-\int_j^{j+1} \sigma(t)d(t)}) I(i). \tag{2.15}$$

Here the sample spacing is assumed to be unity. The sum in Equation 2.15 is equivalent to the integral in Equation 2.14 in the limit as the sample spacing goes to zero [41][46].

Introducing $\alpha(j)$ as the accumulated opacity of ray segment $[j, j+1]$:

$$\alpha(j) = 1 - e^{-\int_j^{j+1} \sigma(t)d(t)}, \tag{2.16}$$

Equation 2.15 can be written as follows:

$$I = \sum_{0 \le i \le n} [\prod_{0 \le j \le i} (1 - \alpha(j))] I(i). \tag{2.17}$$

Equation 2.17 can be evaluated recursively running through each ith sample in back-to-front order:

$$I_{out} = \alpha(i) \cdot I(i) + (1 - \alpha(i)) \cdot I_{in}, \tag{2.18}$$

where I_{in} is the intensity before the evaluation of the ith sample, $I(i)$ is the scattering of the ith sample, and I_{out} is the intensity after having the contribution of the ith sample added to the weighted sum. The initial value of I_{in} is the ambient light. In fact, Equation 2.18 is the Porter-Duff *over* operator used for compositing digital images [45]. In the following subsections two different strategies are presented for approximating the light transport equation using the *over* operator.

2.2.2 Image-order methods

Image-order volume-rendering techniques represent a *backward-mapping* scheme. The image is generated pixel-by-pixel casting rays from the view-point through each pixel of the image plane rasampling the volume along the viewing rays.

Early methods cast parallel or perspective rays from the pixels of the image plane and determine only the first intersection points with a surface contained in the volume [59]. One of these methods is *binary ray casting* aiming at the visualization of surfaces contained in binary volumetric data. Along the viewing rays the volume is resampled at evenly located sample points and the samples take the value of the nearest voxel. When the first sample with a value of one is found the corresponding pixel color is determined by shading the intersected surface point.

Another alternative is *discrete ray casting* [65], where the continuous rays are approximated by discrete 3D lines generated by a Bresenham-like algorithm [54] or a 3D line scan-conversion (voxelization). According to the topology of the scan-converted lines three types of discrete paths can be distinguished: 6-connected, 18-connected, and 26-connected, which are based upon the three adjacency relationships between consecutive voxels along the path. 6-connected paths contain almost twice as many voxels as 26-connected paths, so an image created using 26-connected paths would require less computation. Nevertheless, a 26-connected path may miss an intersection that would be detected using a 6-connected path.

The closest intersection points are stored for each pixel and afterwards an image-space *depth-gradient shading* [6][55] can be performed. Better results can be achieved applying object-space shading techniques like *normal-based contextual shading* [25][3]. Normal computation methods based on surface approximation try to fit a linear [2] or a biquadratic [61][62]

function to the set of points that belong to the same iso-surface. These techniques take a larger voxel neighborhood into account to estimate the surface inclination.

Direct volume rendering of gray-level volumes is not restricted to surface-shaded display like in the case of binary data sets. Here a composite projection of the volume can be performed by evaluating the *light transport equation* (Equation 2.14) along the viewing rays. Composition requires two important parameters, the color and an opacity at each sample location. Levoy [34] proposed an image-order algorithm which assigns these parameters to each grid location in a preprocessing step. The opacity and color values at an arbitrary sample point are calculated by first-order interpolation.

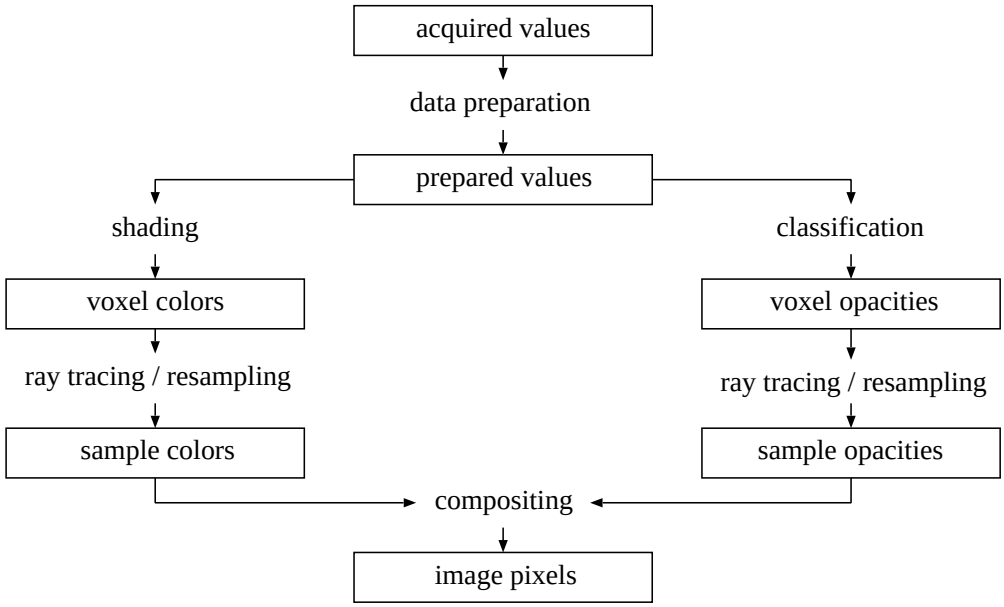


Figure 2.10: The direct volume-rendering pipeline.

The direct volume-rendering pipeline is shown in Figure 2.10. The first step is data preparation which may include, for example, the interpolation of additional slices to obtain an isotropic volume. The prepared array is the input of the shading process where colors are assigned to the voxels depending on their densities. The assigned colors are shaded according to the Phong model [54]. The shading model requires a normal vector at each voxel location. In gray-level volumes the normals can be obtained as the estimated gradients calculated from the *central differences*:

$$\nabla f(x_i, y_j, z_k) \approx \frac{1}{2} \cdot \begin{bmatrix} (f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)) \\ (f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ (f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \end{bmatrix}, \tag{2.19}$$

where $f(x_i, y_j, z_k)$ is the discrete 3D density function. The output of the shading process is an array of voxel colors. In a separate step, classification is performed yielding an additional array of voxel opacities. After having the color and opacity values assigned to each voxel, rays are cast from the view-point through each pixel of the image plane resampling the volume at a finite number of evenly located sample points. The color and opacity values at an arbitrary sample point are calculated by trilinearly interpolating the colors and opacities at the eight closest voxels (Figure 2.11). The final step is the composition of the ray samples using the Porter-Duff *over* operator for approximate evaluation of the *light transport equation* (Equation 2.14).

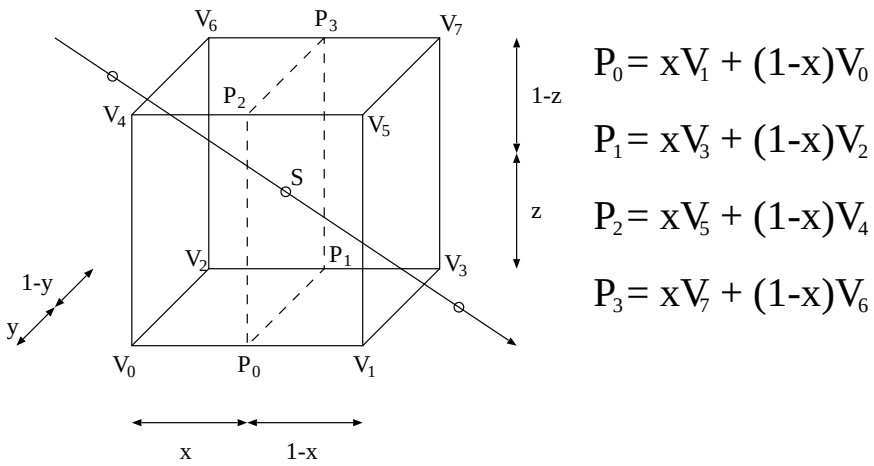


Figure 2.11: Resampling using trilinear interpolation.

Composite ray casting is a flexible approach for visualizing several semi-transparent surfaces contained in the data and produces high quality images (Figure 2.12). However, the *alpha-blending* evaluation of viewing rays is computationally expensive, especially when super sampling is performed trilinearly interpolating each single sample. Therefore, several techniques have been proposed for accelerating ray casting usually exploiting data, image, or frame-to-frame coherence. Chapter 3 gives an overview of these acceleration methods.

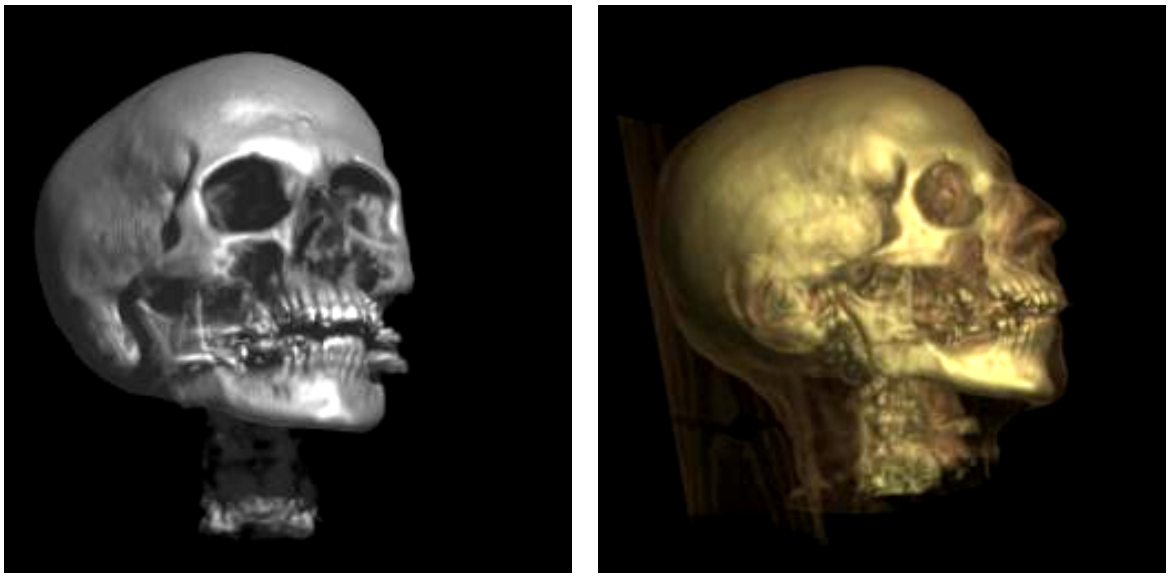


Figure 2.12: A CT scan of a human head rendered using composite ray casting [31].

2.2.3 Object-order methods

Object-order volume-rendering techniques apply a *forward-mapping* scheme. Here the volume is projected voxel-by-voxel onto the image plane. The first object-order methods, similarly to the early image-order methods, aimed at the rendering of binary volumes. Processing each sample, only the voxels with value of one are projected onto the screen. The samples are projected in *back-to-front* order to ensure correct visibility. If two voxels are projected onto the same pixel, the first processed voxel must be farther away from the image plane. This can be accomplished by traversing the data plane-by-plane, and row-by-row inside each plane. For arbitrary orientations of the data in relation to the image plane, some axes may be traversed in an increasing order, while others may be considered in a decreasing order. The ordered traversal can be implemented with three nested loops indexing x-, y-, and z-directions respectively. Such an implementation supports axis-parallel clipping planes. In this case, the traversal can be limited to a smaller rectangular region by simply modifying the bounds of the traversal.

The depth image of the volume can be easily generated. Whenever a voxel is projected onto a pixel, the pixel value is overwritten by the distance of the given voxel from the image plane. Similarly to the early image-based methods the distance image can be passed to a simple 2D discrete shader.

More sophisticated object-order methods like *splatting* proposed by Westover [63] are not restricted to the rendering of binary volumes. According to his approach a gray-scale volume is treated as a 3D discrete density function. Similarly to the ray-casting method a convolution kernel defines how to reconstruct a continuous function from the density samples. In contrast, instead of considering how multiple samples contribute to a sample point, it is considered how a sample can contribute to many other points in space. For each data sample $s = (x_s, y_s, z_s)$, a function C defines its contribution to every point (x, y, z) in the space:

$$C(x, y, z) = h(x - x_s, y - y_s, z - z_s) \cdot f(s), \tag{2.20}$$

where $f(s)$ is the density of sample s. The contribution of a sample s to an image plane pixel (x, y) can be computed by an integration:

$$C(x, y) = f(s) \int_{-\infty}^{\infty} h(x - x_s, y - y_s, u)du, \tag{2.21}$$

where the u coordinate axis is parallel to the viewing ray. Since this integral is independent of the sample density, and depends only on its (x, y) projected location, a *footprint function* F can be defined as follows:

$$F(x, y) = \int_{-\infty}^{\infty} h(x - x_s, y - y_s, u)du, \tag{2.22}$$

where (x, y) is the displacement of an image sample from the center of the sample's image plane projection. The footprint kernel F is a weighting function which defines the contribution of a sample to the affected pixels. A footprint table can be generated by evaluating the integral in Equation 2.22 on a grid with a resolution much higher than the image plane resolution. All the table values lying outside of the footprint table extent have zero weight and therefore need not be considered when generating an image. A footprint table for data sample s is centered on the projected image plane location of s, and sampled in order to determine the weight of the contribution of s to each pixel on the image plane.

Computing a footprint table can be difficult due to the integration required. Although discrete integration methods can be applied to approximate the continuous integral, generating a footprint table is still a costly operation. However, in case of orthographic projection, the footprint table of each sample is the same except for an image plane offset. Therefore, only one

footprint table needs to be calculated per view. Since this would require too much computation time anyway, only one generic footprint table is built for the kernel. For each view, a view-transformed footprint table is created from the generic footprint table. The generic footprint table can be precomputed, therefore it does not matter how long the computation takes. Generation of a view-transformed footprint table from a generic footprint table is described in detail in [63].

There are three modifiable parameters of the splatting algorithm which can strongly affect the image quality. First, the size of the footprint table can be varied. Small footprint tables produce blocky images, while large footprint tables may smooth out details and require more space. Second, different sampling methods can be used when generating the view-transformed footprint table from the generic footprint table. Using a nearest-neighbor approach is fast, but may produce aliasing artifacts. On the other hand, using bilinear interpolation produces smoother images at the expense of longer rendering times. The third parameter which can be modified is the reconstruction filter itself. The choice of, for example, a cone function, Gaussian function, *Sinc* function or bilinear function affects the final image.

Alpha-blending composition is also supported by the splatting algorithm. The voxel contributions of slices mostly perpendicular to the viewing direction are evaluated on associated sheets parallel to the image plane. After having each sheet generated image composition is performed applying the Porter-Duff *over* operator.

Using the splatting algorithm approximately the same image quality can be achieved as applying a composite ray casting. The advantages of splatting over ray casting are the following. First, the cache coherency can be exploited since the voxels are sequentially traversed in the same order as they are stored in memory. In contrast, ray casting requires random access to the voxels. Furthermore, the splatting approach supports incremental rendering in back-to-front or front-to-back order. Using splatting smooth surfaces can be rendered without staircase artifacts, unlike in the case of ray casting. The main drawback of splatting is that the generated images are blurred because of the spherically symmetric reconstruction kernel. In contrast, using ray casting with trilinear reconstruction sharp object boundaries are obtained.

Chapter 3

Acceleration techniques

This chapter overviews the state of the art of fast direct volume-rendering algorithms. Since this thesis concentrates on software-only optimization techniques the recent hardware acceleration tools are not discussed. The fast image-order and object-order methods are presented in Section 3.1 and Section 3.2 respectively. It will be shown that the advantageous properties of these two different approaches are complementary. Therefore, hybrid methods which combine image-order and object-order techniques have been proposed by several authors (see Section 3.3). One of them is a two-pass ray-casting and back-projection algorithm which exploits the frame-to-frame coherency. Another one is the classical shear-warp algorithm, which is based on run-length encoding of volume and image scan-lines exploiting the volume and image coherency respectively. Our goal is to improve the well known general acceleration methods using application oriented optimization. In Chapter 4 it will be shown that applying appropriate algorithms interactive direct volume rendering is possible even on low-end machines if the visualization procedure is restricted to a certain purpose, like fast display of shaded iso-surfaces or fast maximum intensity projection.

3.1 Fast image-order techniques

3.1.1 Hierarchical data structures

Early methods [15][33][53] use hierarchical data structures like octrees, K-d trees, or pyramids to efficiently encode the empty regions in a volume. Such data structures are widely used in computer graphics for accelerating traditional algorithms, like ray tracing. The idea is to quickly find the first intersection point for an arbitrary ray without evaluating the intersections with all the objects. Ray tracing in continuous analytically defined scenes requires a hierarchical structure with arbitrarily fine resolution. In contrast, in volume rendering the discrete representation of the scene can be exploited. For example, a pointerless complete octree can be represented by a *pyramid* [64][35].

Without loss of generality, let us assume that the resolution of the volume is $N \times N \times N$, where $N = 2^M + 1$ for some integer M. A pyramid is defined as a set of $M+1$ volumes. These volumes are indexed by a level number $m = 0, ..., M$, and the volume at level m is denoted by V_m. Volume V_0 measures $N-1$ cells on a side, volume V_1 measures $(N-1)/2$ cells on a side and so on up to the volume V_M which is a single cell.

Levoy [35] applied a binary pyramid in order to quickly traverse the empty regions in a volume. A cell of V_0 represents the rectangular regions between eight neighboring voxels in the original data. The value of a cell in V_0 is zero if all of its eight corner voxels have opacity equal to zero, otherwise its value is one. At higher levels of the pyramid zero is assigned to a cell if all the corner cells at one level lower have value of zero.

For each ray, first the point where the ray enters a single cell at the top level is calculated. Afterwards the pyramid is traversed in the following manner. Whenever a ray enters a cell its value is checked. If it contains zero the ray advances to the next cell at the same level. If the parent of the next cell differs from the parent of the previous one then the parent cell is investigated and the ray is traced further at one level higher. If the parent cell is empty then it can be skipped, and the iteration continues until a non-empty cell is found. In this case, moving down in the hierarchical structure, the first elementary cell is determined which has at least one opaque corner voxel. In such an elementary cell samples are taken at evenly spaced locations along the ray and compositing is performed. Using such a hierarchical ray traversal larger empty regions can be easily skipped. Since the non-empty cells in the binary pyramid represent the regions, where opaque voxels are present the algorithm is called *presence-acceleration*.

Denskin and Hanrahan [15] improved this algorithm using pyramids not only for skipping the empty ray segments but for approximate evaluation of homogeneous regions. Therefore, their technique is called *homogeneity acceleration*. Instead of using a binary pyramid they construct a so called *range* pyramid which contains the maximum and minimum values of subvolumes at one level lower. If the maximum and minimum values of a cell are nearly the same then it is considered homogeneous and an approximate evaluation is performed.

3.1.2 Early ray termination

Associating an accumulated opacity to each pixel of the image plane ray casting can be performed evaluating the rays in *front-to-back* order. In case of *back-to-front* composition, all the samples along the ray have to be taken into account. This computation is usually redundant since several samples can be occluded by a ray segment which is closer to the viewer and has accumulated opacity of one. Therefore, these samples do not contribute to the image. In contrast, using front-to-back composition, the rays can terminate when the accumulated opacity exceeds a predefined threshold [35]. This technique is well known as *early ray termination* or

α-acceleration.

This acceleration method introduces a systematic bias in the image because of the predefined threshold. In order to avoid this, a technique called *Russian Roulette* can be used for unbiased estimation of pixel values [15].

Russian Roulette was developed for Monte-Carlo particle-tracing methods [54] to avoid simulating particles with low weights, but still without biasing the results. Similarly, it can be used to cut down on the average penetration of rays into the volume. Here a ray is assumed to be a particle. Taking a sample defined by the distance t from the origin of the ray, the particle tracing ends before reaching the distance $t + 1$ with probability $p(t)$ which is equal to the opacity $\alpha(t)$. Therefore the probability of the ray termination is the accumulated opacity. One possibility is to assign the value of $I(t)$ to the corresponding pixel when the ray terminates at distance t. This would lead to a high variance of pixel values. Instead, the rays are evaluated using the Porter-Duff *over* operator and they terminate with a probability equal to the accumulated opacity. Using this *Russian Roulette* approach instead of a predefined threshold for ray termination an unbiased estimation of pixel values is ensured.

3.1.3 Distance transformation

The main drawback of acceleration techniques based on hierarchical data structures is the additional computational cost required for the traversal of cells located at different levels of the hierarchy. Furthermore, the rectangular cells only roughly approximate the empty regions.

Cohen proposed a technique called *proximity clouds* for fast 3D grid traversal. Here the geometric information required for empty space skipping is available with the same indices used for the original volume data [7]. The data structure is a simple 3D distance map generated from binary volumes. The input binary volume encodes the transparent and opaque cells similarly to Levoy's approach [35]. In the distance volume each cell contains the distance to the nearest non-empty cell. The distance between two cells is represented by the Euclidean distance of their center points.

Ray casting is performed applying two different cell traversal strategies. The algorithm switches between these two strategies depending on the distance information stored in the current cell. Using fixed-point arithmetic and integer division it is easy to find the cell which contains the current ray sample. If the current sample is in the vicinity of an object a simple incremental ray traversal is performed. If this is not the case, the distance value d stored in the current cell is used for fast skipping of empty regions. The new sample is determined by adding the unit direction of the given ray multiplied by $d - 1$ to the current sample location. The distance from the nearest object has been calculated from the center of the current cell, therefore using a stepping distance $d - 1$, skipping beyond the free-zone can be avoided.

Distance maps can be generated based on several distance metrics, like City-Block, Euclidean, or Chessboard distance. Approximate distance maps are usually calculated applying the efficient *Chamfering* method. The basic idea is to use a mask of local distances and propagate these distances over the volume. The generation of a more accurate distance map requires a larger mask, therefore the preprocessing time is longer. On the other hand, less samples have to be taken in the ray-casting process when more exact distance information is available. Therefore, the applied distance metric is a compromise between the preprocessing and rendering times.

3.2 Fast object-order techniques

3.2.1 Hierarchical splatting

Object-order volume rendering typically loops through the data, calculating the contribution of each volume sample to pixels on the image plane. This is a costly operation for high resolution (e.g. 512^3) data sets. One possibility is to apply *progressive refinement*. For the purpose of interaction, first a lower quality image is rendered. This initial image is progressively refined when a fixed viewing direction has been selected.

For binary data sets, bits can be packed into bytes such that each byte represents a $2 \times 2 \times 2$ portion of the data [59]. The volume is processed bit by bit to generate the full resolution image but lower resolution images can be produced processing the volume byte-by-byte. A byte is considered to represent an element of an object if it contains more than four non-zero bits, otherwise it represents the background. Using this technique, an image with one-half the linear resolution is produced in approximately one-eight the time.

A more general method for decreasing data resolution is to build a pyramid data structure, which for on original data set of N^3 samples, consists of a sequence of $log(N)$ volumes. The first volume is the original data set, while the second volume of one-eighth the resolution is created by averaging each $2 \times 2 \times 2$ group of samples of the original data set. The higher levels of the volume pyramid are created from the lower levels in a similar way until $log(N)$ volumes have been created. An efficient implementation of the splatting algorithm, called *hierarchical splatting* [33] uses such a pyramid data structure. According to the desired image quality, this algorithm scans the appropriate level of the pyramid in a back-to-front order. Each element is splatted onto the image plane using the appropriate sized splat. The splats themselves are approximated by polygons which can be rendered by conventional graphics hardware.

3.2.2 Extraction of surface points

Although there are several optimization techniques based on empty space leaping or approximate evaluation of homogeneous regions, because of the computationally expensive alpha-blending compositing the rendering is still time-demanding.

One alternative to alpha-blending volume visualization is the extraction of relevant voxels and the optimization of the rendering process for sparse data sets. Following these approach interactive applications can be developed which support flexible manipulation of the extracted voxels. In medical imaging systems, for example, the cutting operations are rather important, where a shaded iso-surface and an arbitrary cross-sectional slice can be rendered at the same time.

Sobierajski[52] proposed a fast display method for direct rendering of boundary surfaces. From the volume data only those boundary voxels are extracted which are visible from a certain set of viewing directions. In many cases the six orthographic views are sufficient to obtain an approximate set of all the potentially visible boundary voxels. Better approximation can be achieved increasing the number of directions.

Taking only the six orthographic views into account the visibility calculations can be performed efficiently using a 2D boundary tracing algorithm on the slices perpendicular to each coordinate axis. The output of the surface-extraction algorithm is a list of boundary voxels in which duplicate elements are removed. The generated list stores for each voxel all the attributes which are necessary for the rendering process, like the coordinates or the normal vector.

The set of surface points is passed to the rendering engine. Since adjacent voxels are mapped onto not necessarily adjacent pixels, holes can appear in the produced image. In order to avoid

this problem one voxel is projected onto several pixels depending upon the viewing direction. Since the rendering speed is directly related to the length of the voxel list, for a specific viewing direction the number of voxels to be projected can be reduced by *voxel culling*. This is similar to back-face culling in polygon rendering [54]. If the dot product of the surface normal and the viewing direction is positive the given voxel belongs to a back face, therefore it is not rendered.

Since the presented algorithm follows a direct volume rendering approach, cutting planes can be easily implemented. The projected boundary surface points are shaded according to the lighting conditions and the voxels intersected by the cutting plane are rendered by projecting their original density values onto the image plane.

A fast previewing algorithm proposed by Saito[47] is also based on the extraction of boundary voxels. Similarly to the previous method a set of surface points is stored in a list and it is passed to the rendering engine. In contrast, in order to increase the rendering speed, only a subset of the boundary voxels are extracted according to a uniform distribution. The extracted surface samples are converted to geometrical primitives like cross-lines perpendicular to the surface normal and projected onto the image plane.

3.3 Hybrid acceleration methods

3.3.1 Shear-warp factorization

Since the advantages of image-order and object-order techniques are complementary several authors proposed hybrid methods which combine these two different approaches. One of them is the classical shear-warp algorithm published by Lacroute and Levoy [31]. This algorithm is based on the factorization of the viewing transformation M_{view} into a 3D shear transformation M_{shear} and a 2D warp transformation M_{warp} (Figure 3.1):

$$M_{view} = M_{warp} \cdot M_{shear}$$

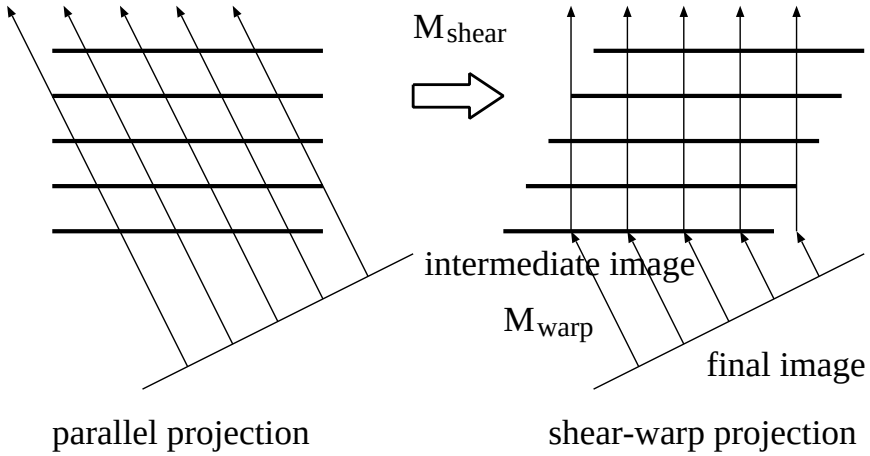


Figure 3.1: Shear-warp factorization of the viewing transformation.

An intermediate image is generated in the sheared space processing the volume in object order supporting efficient caching. Since the intermediate image is parallel to the volume slices it is easy to implement a fast parallel projection. Compositing is performed on the sheared slices in front-to-back order using the Porter-Duff *over* operator. The final image is produced from the intermediate image transforming it by a relatively cheap 2D warp operation.

In order to improve efficiency, the algorithm preprocesses the volume exploiting that the voxel scanlines in the sheared volume are aligned with pixel scanlines in the intermediate image. It means that the volume and image data can both be traversed in scanline order. Taking advantages of coherence in the volume, the scanlines are run-length encoded in order to skip runs of transparent voxels. Therefore, the encoded scanlines consist of transparent and non-transparent runs. The coherence in the image can also be exploited. For each pixel a pointer to the first non-opaque pixel in the same scanline is stored. An image pixel is defined to be opaque if its accumulated opacity exceeds a user-specified threshold. The offsets associated with the image pixels are used to skip runs of opaque pixels without examining each single pixel. The pixel array and the offsets represent a run-length encoding of the intermediate image which is computed on-the-fly during the rendering. Using simultaneously the run-length encoded voxel and image scanlines the transparent and occluded voxels can be precisely skipped.

3.3.2 Incremental volume rotation

As it was previously discussed, the classical shear-warp algorithm exploits both volume and image coherency. In this subsection a direct surface rendering technique proposed by Gudmundsson and Randén [23] is presented. This algorithm is based on the frame-to-frame coherency. The idea is to perform a brute force first hit ray casting detecting the first surface points intersected by the viewing rays. Of course, the generation of this first frame is rather slow but the further frames are calculated incrementally from the previous ones providing interactive frame rates.

The surface points are added to a list storing all relevant attributes, like the coordinates, the estimated normal, or the assigned color which are necessary for the rendering. Whenever the viewing direction is changed, each surface point in the list is rotated and projected back onto the image plane taking into account the possible occlusions. Those surface points which are hidden by another one are removed from the list. If the viewing direction is just slightly changed the number of pixels which are not covered by any projected surface points is relatively small. In order to fill these "holes" in the image brute force ray casting is performed for the corresponding rays and new surface points are detected and added to the list. Therefore, after having a frame generated the number of surface points stored in the list is at most the number of pixels. Thus the computation cost is proportional to the image size rather than to the volume size.

The main steps of the algorithm which are executed for each projection in the rotation sequence are the following:

1. Rotate all the surface points in the list using a geometric transformation.
2. Project all the transformed points onto the image plane.
3. Remove those points from the list that have become hidden.
4. Detect new surface points by selective ray casting and insert them to the list.
5. Calculate the pixel values by resampling and shading the projected surface points.

Note that, the back-projected surface samples are not uniformly distributed. Therefore, resampling in image space is necessary in Step 5. For example, to each pixel the nearest projected sample can be assigned. In order to reduce the number of rays which need to be cast in Step 4. one surface point can be projected onto several pixels similarly to the splatting algorithm. This improvement is easy to implement in case of orthographic mapping using projection templates.

Yagel and Shi [67] improved this incremental rotation technique using a hierarchical data structure for fast space-leaping in the ray casting step.

Chapter 4

Interactive volume rendering

In this chapter the main contribution of this dissertation is presented. New interactive volume-visualization algorithms are proposed for traditional alpha-blending rendering, surface shaded display, maximum intensity projection (MIP), and fast previewing.

In Section 4.1 a technique called *binary shear transformation* is described. It is considered to be a general acceleration tool, which can be applied for volume rendering methods based on different visualization models. Section 4.1 and Section 4.2 present its adaptations for fast alpha-blending ray casting and maximum intensity projection respectively. These algorithms exploit the data coherence and redundancy of ray casting, since some segments of the viewing rays usually do not contribute to the final image. Using the presented data structure transformed by a binary shear operation, these empty segments can be easily skipped significantly accelerating the rendering process.

Section 4.3 describes another approach which is rather surface oriented. Here an iso-surface is rendered efficiently after having the volume preprocessed. In order to reduce the number of voxels to be projected onto the image plane, only the potentially visible voxels are extracted from the volume. The presented technique also supports interactive cut operations by arbitrary cutting planes which is important in medical applications.

Since the fast surface-rendering technique presented in Section 4.3 does not support super-sampling staircase artifacts can appear in the generated images. In order to compensate this drawback, in Section 4.4 a new gradient estimation scheme based on 4D linear regression is described which provides a smooth gradient function.

Section 4.5 presents a fully interactive direct volume rendering technique. It is based on a simplified visualization model called *bubble model*. The iso-surfaces are rendered as thin semi-transparent membranes similarly to blown soap bubbles. In the rendering procedure the data reduction is exploited, therefore high frame-rates can be achieved without hardware acceleration.

4.1 Fast volume rendering using binary shear transformation

This section presents a fast volume rendering algorithm based on an incremental rotation technique [9][10][14]. In order to precisely skip the empty regions along the rays to be evaluated a binary volume is generated indicating the locations of the transparent cells. This mask is rotated by an incremental binary shear transformation, executing bitwise boolean operations on integers storing the bits of the binary volume. The ray casting is accelerated using the transformed mask and an appropriate lookup-table technique for finding the first non-transparent cell along each ray.

4.1.1 Introduction

Our primary goal is to develop a fast volume rendering algorithm which can be used in interactive applications. The main idea is to render the first hit iso-surface with high frame rates during the rotation. After having an appropriate viewing direction selected, the final image is rendered according to a predefined transfer function. In order to accelerate also the alpha-blending evaluation of viewing rays the same data structure is used as for the fast rotation.

Early methods [15][33][53] use hierarchical data structures like pyramids, octrees or K-d trees to quickly traverse the transparent regions decreasing the number of samples to be evaluated. Hierarchical data structures are used in homogeneity acceleration techniques as well [15][21], which apply a simplified approximate evaluation for the homogeneous regions. Although these methods speed up the ray-casting process the rendering times are far from interactivity. This is due to the significant overhead required for handling the hierarchical data structure itself. For instance, in case of an octree structure, for each ray the entry and exit intersection points with rectangular cells have to be calculated.

Recent algorithms like distance transformation based methods [7][68] or Lacroute's shear-warp algorithm [31] concentrate on a more precise skipping of empty ray segments instead of approximated evaluation of homogeneous regions. The main advantage of these techniques over hierarchical methods is the applied encoding scheme, since the information about the empty cells is available with the same indices as used for the volume data. However, these methods cannot be applied in interactive applications either. The 3D distance map does not provide the exact distance of the nearest surface point hit by a given ray. Therefore, several samples have to be calculated until the surface is reached. Lacroute's algorithm evaluates the entire run-length encoded volume representation even if only the first hit iso-surface is required to be displayed.

In contrast, our method can render the first hit iso-surface interactively using a special data structure and it significantly speeds up the alpha-blending evaluation as well. First a special case is described, where the viewing direction is axis parallel and the segmentation mask is stored in a very simple data structure supporting the ray casting acceleration. Afterwards it is shown how to extend the algorithm to arbitrary viewing directions transforming the segmentation mask by an efficient binary shear operation.

4.1.2 Definition of the segmentation mask

The input data of a direct volume rendering pipeline is a spatial density function $f : R^3 \rightarrow R$ sampled at regular grid points, yielding a volume $V : Z^3 \rightarrow R$ of size $X \times Y \times Z$, where

$$V_{i,j,k} = f(x_i, y_j, z_k). \tag{4.1}$$

In the classification process, according to the density values different attributes like opacity or color are assigned to each voxel. The opacity function maps the volume V onto a classified volume C of the same size, where $C.\alpha : Z^3 \rightarrow [0,1]$ and $C.color : Z^3 \rightarrow [0,1]^3$ define the opacity and color fields respectively. In order to handle the empty cells efficiently many acceleration algorithms create a binary volume assigning zero to the transparent and one to the non-transparent cells. A cell is considered transparent if all of its eight corner voxels have zero opacities. In our method the definition of a cell depends on the principal direction of the viewpoint. Without loss of generality, it is assumed that the principal component of the viewing direction is its z-coordinate. In this case, the proposed algorithm resamples the volume only in planes $z = z_k$, where *k = 0,1,2, ... Z-1*. The density samples on these planes are computed from the densities of the four closest voxels using bilinear interpolation. The opacity of the sample is non-zero if at least one of the four corner voxels is opaque, thus a binary volume $B : Z^3 \rightarrow \{0,1\}$ of size *(X-1)* $\times$ *(Y-1)* $\times$ *Z* is defined in the following way:

$$B_{i,j,k} = \begin{cases} 1 \text{ if } C.\alpha_{i,j,k} > 0 \text{ or} \\ \quad C.\alpha_{i+1,j,k} > 0 \text{ or} \\ \quad C.\alpha_{i,j+1,k} > 0 \text{ or} \\ \quad C.\alpha_{i+1,j+1,k} > 0 \\ 0 \text{ otherwise.} \end{cases} \tag{4.2}$$

4.1.3 Ray casting

The binary volume B can be stored in an integer array, where an integer represents a segment of a bit row parallel to the z-axis (in the further discussion, this array is referred to as *mask*). In the special case, when the viewing direction is parallel to the z-axis the volume can be rendered very efficiently by parallel ray casting, since the problem of finding the first non-transparent cell hit by a ray can be reduced to the problem of finding the first non-zero bit inside an integer. Assume that, one integer stores 32 bit long segments of bit rows parallel to the z-axis. In order to determine the position of the first non-zero bit inside a segment (representing the first non-empty cell hit by the corresponding viewing ray) the following binary search algorithm can be used:

```
int Depth(int segment) {
  int e, pos = 0, threshold;
  for(e = 16; e > 0; e /= 2) {
    threshold = pow(2, pos + e);
    if(segment >= threshold) pos += e;
  }
  return 31 - pos;
}
```

Function *pow* resulting in the discrete power $2^{\mathrm{pos+e}}$ is just a symbolic subroutine call, since these constants can be stored in a lookup table. Of course, the optimal solution would be the direct addressing of a lookup table with the segment value which stores the position of the first non-zero bit for all possible combinations. It would require the allocation of an array of size 2^{32} bytes. Instead of this, the first non-zero byte can be found by binary search, and the offset position inside this byte is read from a lookup table. Figure 4.1 shows how to use such a lookup table. Note that, the lookup table contains only 256 entries. Assuming that the most significant bit is the nearest one to the viewer the following routine provides the position of the first non-zero bit, where the size of an integer is supposed to be 32 bits:

```
int Depth(int segment) {
  if(segment < 0x00010000) {
    if(segment < 0x00000100) return 24 + lut[segment];
    else return 16 + lut[segment >> 8];
  } else {
    if(segment < 0x01000000) return 8 + lut[segment >> 16];
    else return lut[segment >> 24];
  }
}
```

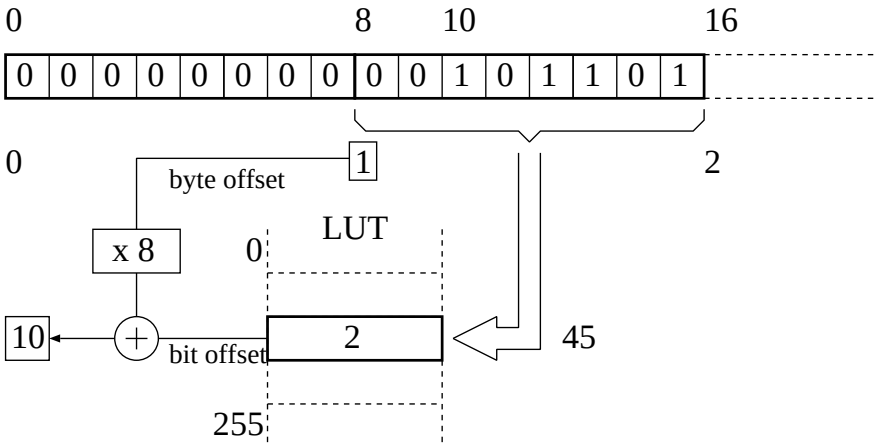
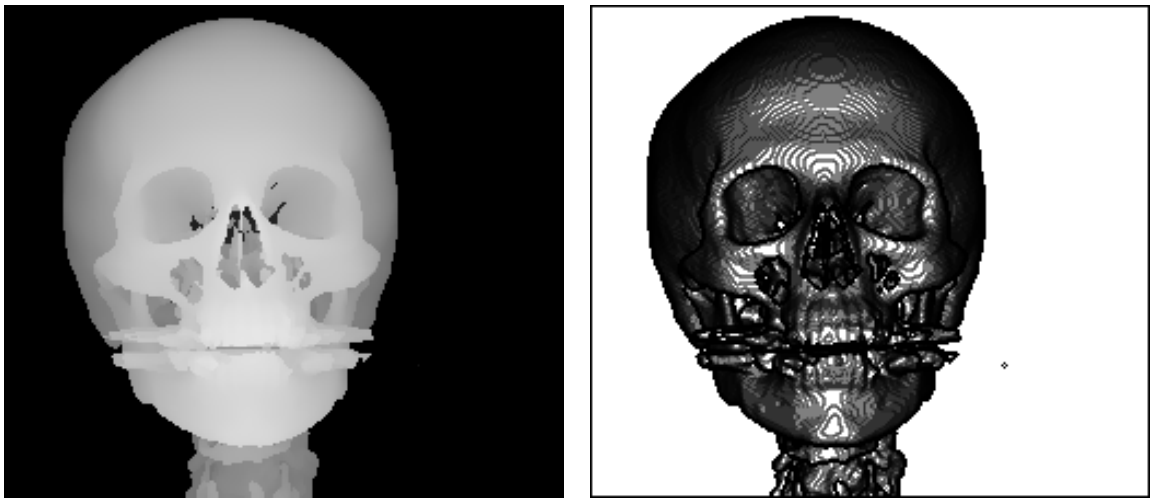


Figure 4.1: An example for a LUT entry.

Usually one integer is not enough for storing a complete row of the binary volume B, thus the segments of the rows stored in integers have to be checked sequentially and the routine *Depth* is called only for the first non-zero integer. Therefore, the complete "ray-tracing" procedure can be implemented as follows:

```
int Trace(int x, int y) {
  int z = 0, segment;
  z_max = (Z + 31) / 32;
  for(z = 0; z < z_max; z++)
    if(segment = mask[z][y][x])
      return z * 32 + Depth(segment);
  return Z - 1;
}
```

Having the first intersection point determined for each ray the x, y projection of the volume can be rendered directly by depth cueing. The greatest advantage of depth-cueing rendering is, that it is not necessary to store the original volume in the main memory. Therefore, it can be applied in fast previewer applications. Furthermore, in order to display the shaded boundary iso-surface a very simple 2D image-based shading can also be used without performing a shading



(a) (b)

Figure 4.2: Distance image (a) and depth-gradient shading (b) of a human skull.

procedure for the entire volume. Figure 4.2 shows the distance image (a) and depth-gradient shading (b) of a human skull.

Depth-gradient shading [6][55] approximates the gradient from the z-buffer calculating the differences between the depths of the given and the neighboring pixels. This approach produces sharp contours where in the neighboring pixels different objects are visible or where there is a drastic jump between the depth values. In order to avoid this artifact *context sensitive normal estimation* [66] can be used which takes also these object and slope discontinuities into account.

In order to achieve higher image quality the original volume is shaded in a preprocessing step. The shaded colors are computed for each voxel according to the view independent Lambertian shading model:

$$C.shadedcolor_{i,j,k} = C.color_{i,j,k} \cdot \frac{\nabla f(x_i, y_j, z_k)}{|\nabla f(x_i, y_j, z_k)|} \cdot L, \tag{4.3}$$

where L is the direction of the light source, $C.color_{i,j,k}$ is the own color of the voxel at the position (x_i, y_j, z_k) and $\nabla f(x_i, y_j, z_k)$ is the estimated gradient vector, which is the surface normal in the given voxel location:

$$\nabla f(x_i, y_j, z_k) \approx \frac{1}{2} \cdot \begin{bmatrix} (f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)) \\ (f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ (f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \end{bmatrix}. \tag{4.4}$$

The previously presented lookup-table technique supports also the alpha-blending evaluation of viewing rays. Using the binary rows representing the non-transparent cells along a ray, the empty regions can be precisely skipped during the ray casting process. The following algorithm does not evaluate the fully transparent cells, since the routine *Depth* jumps into the position of the next cell which has at least one opaque corner voxel.

```
rgb AlphaBlending(int x, int y) {
  int z, i, segment;
  rgb color = BLACK;
  voxel v;
  double trans = 1.0; // accumulated transparency
  int i_max = (Z + 31) / 32;
  for(i = 0; i < i_max; i++) {
    segment = mask[i][y][x]);
    while(segment) {
      z = Depth(segment); v = volume[32 * i + z][y][x];
      trans *= 1.0 - v.opacity; if(trans  < threshold) return color;
      color += v.color * v.opacity * trans;
      segment &= ~(0x80000000 >> z); // deletes the processed bit
    }
  }
  return color;
}
```

The accumulated transparency is stored in variable $trans$ and when it falls under a predefined *threshold* the front to back evaluation terminates. This acceleration technique is well known in volume rendering literature as *early ray termination* [35]. In the variable $segment$ each bit is erased after the corresponding cell has been processed. Therefore, routine *Depth* can be called again in order to efficiently find the z-position of the next non-zero bit. After having this z-position determined which is a bit offset inside the ith segment the z-coordinate of the corresponding voxel is calculated as $32 * i + z$.

4.1.4 Binary shear transformation

The lookup table technique described in the previous section works only for a special case when the viewing direction is parallel to the z-axis. In this section it is discussed how to extend it to viewing directions, where the principal component is the z-coordinate using a binary shear transformation of the mask volume. This operation effectively shifts the bits of the binary mask volume perpendicularly to the principal component of the viewing direction. In an interactive volume rendering application the volume is required to be rotated continuously by small difference angles in order to perceive the topology of the surfaces much better than in a static image. Furthermore, generating an animation sequence by rotating the volume a drastic change in viewing directions is not usual. If the difference angle is small enough then there is no slice in the binary volume which has to be shifted by more than one voxel. In this case, one shear operation can be performed very efficiently since just bitwise operations need to be executed on neighboring integers. That is the reason why our method shears the binary volume B incrementally, applying a technique similar to the method proposed by Cohen-Or and Fleishman [8]. They used their so called *incremental alignment algorithm* in order to reduce the communication overhead in a large multi-processor architecture supporting shearing of volumes. Since some bits can be shifted out of the integer array storing the binary volume B, it has to be extended by $Z/2$ rows filled with zero values along the x-axis and along the y-axis as well in both directions as it is illustrated in Figure 4.3.

This extended array is defined as: *int mask*[*depth*][*height*][*width*], where *depth* $= (Z + 31)\ div\ 32$, *height* $= Y + Z$ and *width* $= X + Z$. The following routine performs one shear step in the left direction processing 32 voxels in each step of the internal loop:

```
void ShearLeft() {
  int i, j, k;
  for(k = 0; k < depth/2; k++) {
    for(j = 0; j < height; j++) {
      for(i = 0; i < width-1; i++)
        mask[k][j][i] = mask[k][j][i] & ~shift_x[k] |
                        mask[k][j][i+1] & shift_x[k];
      mask[k][j][width-1] &= ~shift_x[k];
    }
  for(k = depth/2; k < depth; k++) {
    for(j = 0; j < height; j++) {
      for(i = width-1; i > 0; i--)
        mask[k][j][i] = mask[k][j][i] & ~shift_x[k] |
                        mask[k][j][i-1] & shift_x[k];
      mask[k][j][0] &= ~shift_x[k];
    }
}
```

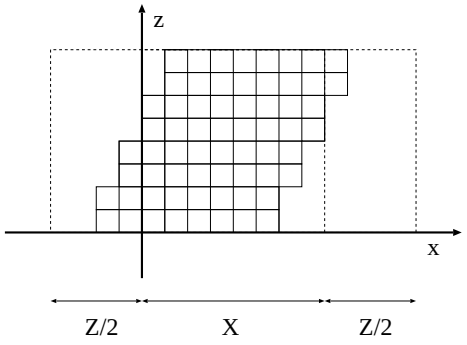


Figure 4.3: Extension of the binary mask.

For the sake of clarity, this routine is not optimized but it can be improved introducing local pointer variables in order to avoid unnecessary array addressing. On the other hand only that part of the extended mask needs to be sheared which contains the non-zero bits representing the non-transparent cells. The integer array *shift_x* is defined as: *int shift_x*[*depth*] and it stores a binary vector of size Z indicating those z positions where the corresponding slices have to be shifted in the given shearing phase. There is also such an array denoted by *shift_y* for the y direction.

In order to determine the offsets of the slices in different shearing phases, another two arrays are introduced for storing the real translations along the x-axis and the y-axis and they are defined as: *double trans_x*[*Z*] and *double trans_y*[*Z*] respectively. These translation arrays contain the x and y coordinates of those points, where the $z = z_k$ planes intersect the 3D line connecting the point p_0(*trans_x*[0],*trans_y*[0],0) with point p_1(*trans_x*[*Z-1*],*trans_y*[*Z-1*],*Z-1*). Initially, this line aligns to the z-axis, thus the translation arrays contain zeros.

Before executing a binary shear operation the *shift* vectors are evaluated in advance according to the rotation direction. For example, when a clockwise rotation around the y-axis is needed, the point p_0 is translated by one along the x-axis into negative direction and p_1 is translated as well, but into positive direction. After this, the intersection points of the line connecting the new p_0 and p_1 and the planes $z = z_k$ are computed anew and the coordinates are stored in the

translation arrays. The new binary shift vectors are determined according to these translation values. For example, the new *shift_x* array can be computed using the following routine:

```
void ComputeNewShiftX()
  {
    double x0 = trans_x[0] -= 1.0, x1 = trans_x[Z-1] += 1.0;
    int bit = 0x80000000, l = Z - 1;
    for(int z = 0; z < Z; z++) {
      double x = (x0 * (l - z) + x1 * z) / l;
      if(floor(x) != floor(trans_x[z]))
        shift_x[z/32] |= bit >> (z % 32);
      else
        shift_x[z/32] |= ~(bit >> (z % 32);
      trans_x[z] = x;
    }
  }
```

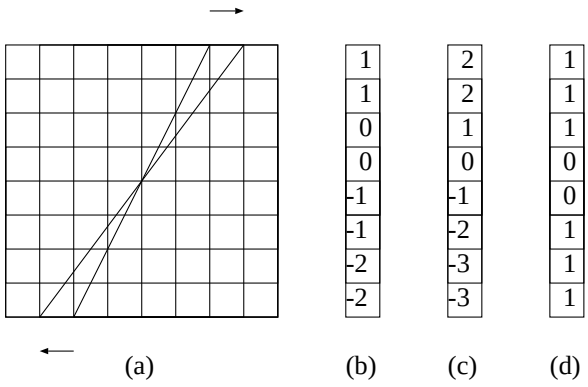


Figure 4.4: (a) An incremental shear step. (b) Floors of the old values of vector trans_x. (c) Floors of the new values of vector trans_x. (d) The binary vector shift_x.

If the floors of the previous and the new translation values are not the same then the corresponding bit is set to one in the *shift_x* array, indicating that the associated slice needs to be shifted in the next binary shear operation (Figure 4.4). Since the difference between the old and new translations is the greatest in the plane $z = z_0$ (or $z = z_{Z-1}$) the difference cannot be greater than one in the intermediate z points, thus there is no slice which needs to be shifted with more than one bit.

4.1.5 Resampling

Using the transformed binary volume an intermediate image of size *width* $\times$ *height* is generated casting the rays from the grid points. Due to the shear transformation the *Depth* routine can be applied in the general case as well, since the segments of the rows perpendicular to the temporary image plane are stored in integers. The position of the first non-zero bit in a row determines the index i of the $z = z_i$ plane, where the first non-transparent cell is located along the corresponding ray. The accurate location of the sample point inside this cell is computed taking into account the exact translation values at the given depth z. By subtracting the values of *trans_x*[i] and *trans_y*[i] respectively from the x and y coordinates of the given cell, the sample

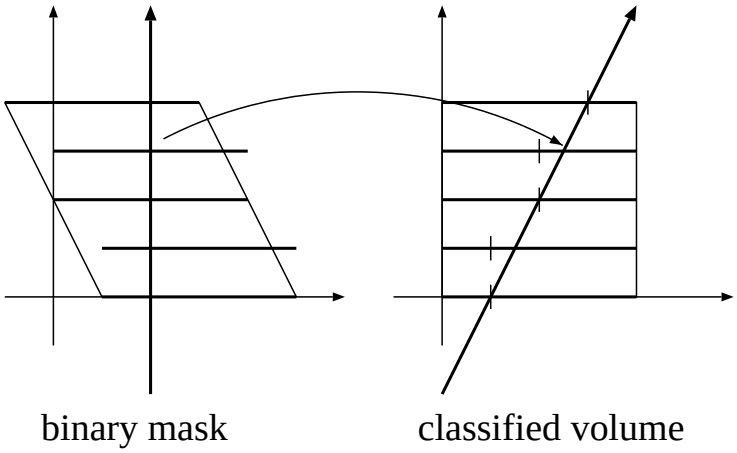


Figure 4.5: Mapping the sample location from the sheared space into the original object space.

location is easily mapped from the sheared space to the original object space, where the color and opacity values are stored. This mapping is illustrated in Figure 4.5.

In order to calculate the density in the mapped sample point location bilinear interpolation is used for the four corner voxels of the cell. As the opacity of the sample is not necessarily one so the ray has to be traced further and evaluated according to the transfer function. If the volume contains internal empty regions (like a human skull) it makes sense to use the binary volume for evaluating the rest of the samples. First the integer representing a z-row of bits is copied into a temporary variable, and whenever a non-zero bit has been processed it is set to zero, thus the routine *Depth* can be used again for finding the z-position of the next non-transparent cell.

4.1.6 Shear-warp projection

According to the shear-warp approach proposed by Lacroute [31] the final image is produced from the intermediate image by a relatively cheap 2D warp operation (Figure 4.6). The resolution of the intermediate image depends on the volume resolution. In order to generate higher resolution images the intermediate image can be resampled as a texture map using bilinear interpolation. Since the 2D warp operation is defined by a transformation matrix the scaling factors can be easily built into it. Therefore, the image zooming does not require additional effort. The final image is generated pixel-by-pixel mapping each pixel location onto the intermediate image and the corresponding color is bilinearly interpolated from the four closest pixel values.

The extension to arbitrary viewing directions is obvious since a binary mask can be generated for each principal direction. The next two subsections describe two further improvements, which could be useful in a practical implementation.

4.1.7 Rotation of large data sets

Due to the incremental shear transformation the effective speed of the rotation could be low, especially processing larger volumes (256^3). Since most of the time is used for rendering, a possible solution to this problem is to render the volume after a couple of incremental shears. Increasing the size of the data set the ratio of the rendering and shearing times is getting lower, thus this strategy is not the best one. Another alternative is the introduction of super cells, which are square regions in the slices perpendicular to the principal direction. In the binary volume, one is assigned to the corresponding super cell if at least one voxel in it is opaque.

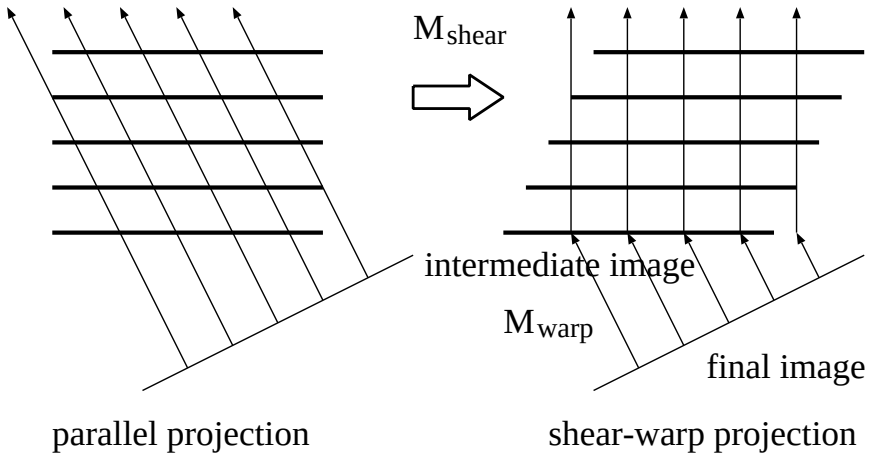


Figure 4.6: Shear-warp factorization of the viewing transformation.

Increasing the size of the super cell the shear transformation becomes faster but the rendering process slows down since the routine *Depth* does not necessarily return the exact depth but only a lower bound, so the rays have to be traced further until a non-transparent sample is found. In Section 4.1.9 the performance is analyzed investigating the optimal cell size for data sets of different sizes.

4.1.8 Adaptive thresholding

The primary limitation of the presented technique is that the volume has to be classified in advance in order to generate the binary volume and afterwards the opacity function cannot be modified in a flexible way. Supposing that, the user wants to operate with a fixed number of transfer functions an appropriate density encoding scheme can be used to allow rapid switching between them. Each transfer function has a lower density threshold, where below this threshold zero opacity is assigned to the given sample. Let us assume that we want to use only three transfer functions. The lower density thresholds divide the density domain into intervals I_0, I_1, I_2, I_3 sorted in ascending order by their borders with increasing index. To each interval the two bit binary format of the corresponding index is assigned as a unique code. The code of a cell is defined as the code of the interval which contains the highest corner voxel density. The cell codes are stored in an integer array which is similar to the *mask* array but it contains two bits for each cell instead of one. This array can be sheared as well, but the bits of a code should always be moved at the same time in order to avoid the cutting of the codes. In the rendering phase the routine *Depth* must use the appropriate lookup table depending on the bit pattern to be searched for. Whenever the user changes the transfer function the variable *lut* has to be set to the address of the corresponding lookup table. This encoding scheme allows rapid access to the first non-transparent cell along the viewing ray independently from the selected transfer function. Let us take an example from the medical imaging practice, where only four materials (air, fat, soft tissue, and bone) can be separated according to the *Hounsfield densities*[16][44]. In this case it makes sense to divide the density domain according to the lower density threshold of fat, soft tissue, and bone respectively. For example, having selected a transfer function which assigns zero opacities to the samples below the lower threshold of the soft tissue, only the codes “10” and “11” will be searched for in the binary volume, precisely skipping the transparent cells. In this case, the corresponding lookup table contains the bit offset of the first “10” or “11” pattern inside the given byte. The presented density encoding scheme does not affect significantly the

performance and allows fast switching between the predefined transfer functions.

4.1.9 Implementation

The presented fast rotation technique has been implemented in C++ and it has been tested on an SGI Indy workstation. Table 4.1 summarizes the running time measurements for a CT scan of a human head and Table 4.2 contains the test results for a higher resolution volume of the same data. The applied transfer function assigns high opacities to the voxels representing the bone thus rays terminate right after reaching the boundary of the skull (Figure 4.7).

cell size	shearing time	rendering time	frame rate
1	0.019 sec	0.114 sec	6.87 Hz
2	0.005 sec	0.107 sec	8.64 Hz
3	0.002 sec	0.118 sec	8.11 Hz
4	0.001 sec	0.156 sec	6.26 Hz

Table 4.1: Test results for the CT head of size $128 \times 128 \times 113$.

cell size	shearing time	rendering time	frame rate
1	0.160 sec	0.590 sec	1.21 Hz
2	0.040 sec	0.492 sec	1.81 Hz
3	0.017 sec	0.535 sec	1.78 Hz
4	0.009 sec	0.709 sec	1.37 Hz

Table 4.2: Test results for the CT head of size $256 \times 256 \times 225$.

Note that, the optimal super cell size is not necessarily the one which the highest frame rate belongs to, since with larger super cell size the effective rotation speed is higher. In order to rotate the volume continuously the cell size must be set small and higher rotation speed can be achieved by setting larger cell size producing approximately the same frame rates.

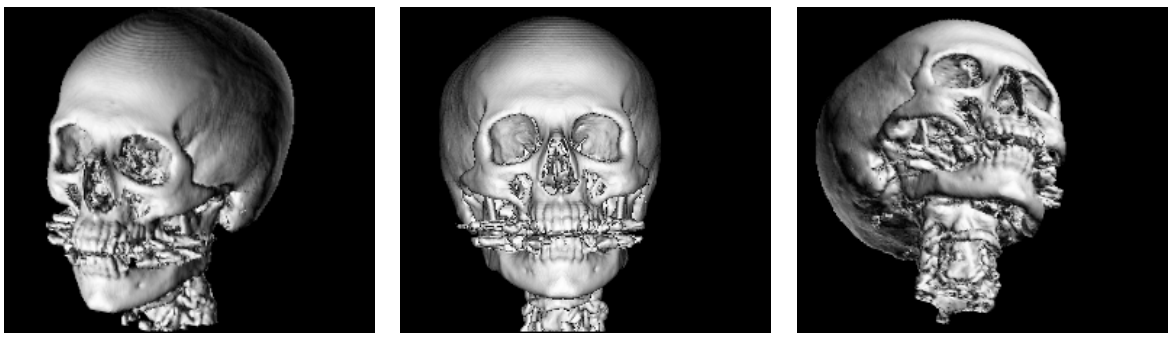


Figure 4.7: Interactive rotation using fast iso-surface rendering.

Using transfer functions which assign low opacity values to different tissues the rendering time increases drastically, since after skipping the empty regions the alpha-blending evaluation

of the semi-transparent segments is computationally very expensive. Although setting larger super cell size higher rotation speed can be achieved in the fast previewing phase the high quality rendering slows down since the binary volume contains less precise information about the transparent cells. Table 4.3 and Table 4.4 show the average rendering times for the low and high resolution data sets respectively using three different transfer functions illustrated in Figure 4.8.

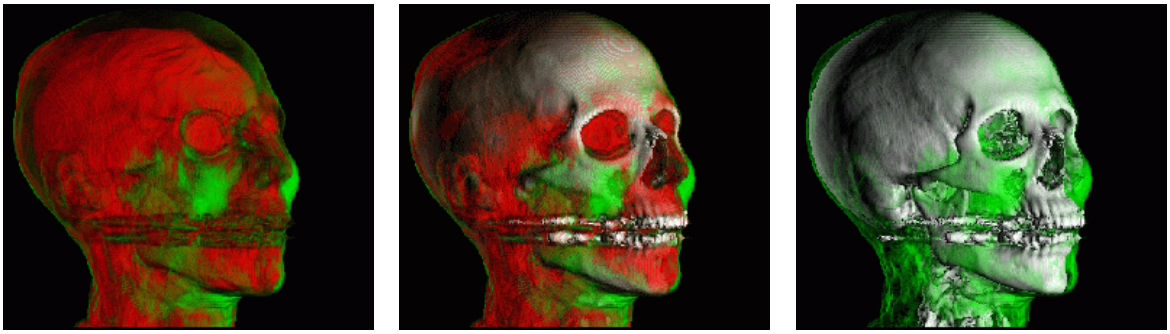


Figure 4.8: Alpha-blending rendering using different transfer functions.

cell size	transfer function A	transfer function B	transfer function C
1	0.36 sec	0.19 sec	0.18 sec
2	0.39 sec	0.21 sec	0.21 sec
3	0.43 sec	0.25 sec	0.24 sec
4	0.51 sec	0.32 sec	0.31 sec

Table 4.3: Rendering times for the volume of size $128 \times 128 \times 113$.

cell size	transfer function A	transfer function B	transfer function C
1	1.97 sec	1.02 sec	0.99 sec
2	1.93 sec	1.01 sec	0.96 sec
3	2.08 sec	1.09 sec	1.08 sec
4	2.44 sec	1.45 sec	1.46 sec

Table 4.4: Rendering times for the volume of size $256 \times 256 \times 225$.

4.1.10 Summary

In this section a fast volume rotation technique has been presented which provides interactive frame rates without using any specialized hardware support. Fast rotation can be achieved using binary or near binary opacity functions, when rays terminate right after reaching an opaque surface. In this sense the proposed technique is a surface oriented algorithm but unlike other interactive iso-surface methods it significantly speeds up the classical transfer function based ray casting. Because of the precise skipping of empty regions it is approximately as efficient as

the classical shear-warp algorithm based on run-length encoding [31]. In a practical implementation it can be applied as a fast previewer rendering the iso-surface defined by the lower density threshold of the selected transfer function, where the viewing direction can be set interactively, and the final image is rendered using the alpha-blending evaluation. The effective rotation speed can be increased by setting a larger super cell size and applying the proposed adaptive thresholding extension the user can switch rapidly between predefined transfer functions.

4.2 Fast maximum intensity projection

In this section it is shown that the previously presented technique based on binary shear transformation is a universal acceleration tool for different visualization models. It supports not only the classical model based on transfer functions but maximum intensity projection as well [12].

4.2.1 Introduction

Maximum Intensity Projection (MIP) is a widely accepted rendering technique which can be used especially for the visualization of location, shape, and topology of blood vessels. Although the shaded surfaces of different tissues cannot be rendered applying this method, practically it is much more important in diagnosis then the classical transfer function based rendering. Therefore, a MIP rendering module is part of almost every commercial medical-imaging system. On the other hand it can be considered as a simulated X-Ray rendering method since it approximates the density integrals along the projection rays. Therefore, it is used for general CT or MRI medical data sets as well. Although maximum intensity projection is a very useful technique for analysis of medical data, not many efficient algorithms exist for finding the maximum intensity along a mathematical ray.

The classical brute force implementation of MIP resamples the density volume at a finite number of evenly located sample points along the rays and selects the maximum density sample, assuming a piecewise constant approximation of the density function. The accuracy depends on the sampling density and the applied resampling filter as well. Instead of the computationally expensive trilinear interpolation a nearest neighbor resampling can be used, which ensures faster rendering for a slight reduction of image quality.

In order to achieve higher accuracy, for each cell intersected by the given ray the exact entry and exit points need to be determined, and the local maximum value has to be found in the ray segment bounded by these points [49]. Considering the volume to be a continuous 3D function, where the density of the intermediate points is the trilinear interpolation of the densities of the eight closest voxels, the density function along the internal ray segment is a polynomial of third degree. The local maximum location can be calculated analytically or using heuristic approximations [49]. Exploiting that the interpolated density cannot be greater than the maximum density of the eight closest voxels this calculation has to be performed only for the promising cells, where the maximum corner density of the cell is greater than the current ray density. In spite of this, algorithms investigating the local maximum locations inside a cell are rather time demanding, therefore they cannot be applied in interactive applications.

In the last two decades several volume rendering techniques have been published aiming at the accelerated evaluation of the well known light transport equation [15][31][35][44][21][7][68]. Most of these methods use hierarchical data structures to speed up the ray traversal exploiting the coherence of the data set [15][33][53]. The min-max versions of these data structures like octrees, K-d trees, or pyramids can be used for accelerated maximum intensity projection as well. Among the direct volume rendering acceleration techniques, the greatest time savings have been made with Lacroute's shear-warp factorization algorithm, based on run-length encoding of transparent regions [31]. Since this method was proposed for the fast alpha-blending evaluation, it cannot be used for MIP without modification. The new method presented in this section also uses the shear-warp projection approach, but the preprocessing is performed in a completely different manner optimized for maximum intensity projection.

4.2.2 Encoding of the density intervals

The main problem in maximum intensity projection is that several volume samples have to be investigated along each viewing ray and only one sample per ray contributes to the final image. In order to avoid the unnecessary resampling our method [12] decomposes the density domain into a finite number of intervals. At first, for each ray the interval is determined which contains the maximum density, then the volume is resampled at those voxel locations where the densities reside in the given interval.

For the sake of clarity, the volume is considered to be a piecewise constant 3D density function and later it is discussed how to extend the method to use a more exact bilinear interpolation for resampling. To each voxel a binary code is assigned representing an interval which contains the density of the given voxel. The density domain is divided into n intervals I_0, I_1, ... I_{n-1} sorted in ascending order by their borders with increasing index. To each interval the binary format of the corresponding index is assigned as a unique code. In a preprocessing step an additional volume is created which stores the density interval codes for each voxel.

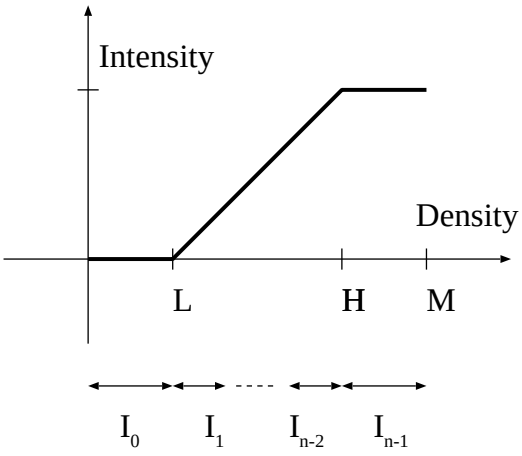


Figure 4.9: Mapping the densities onto intensities.

In practice of medical data evaluation usually a *windowing* function is used to map the relevant part of the density domain onto intensities (Figure 4.9)(for the sake of simplicity, in the further discussion the ”density” term is used for the data values of the input array, although it is not completely correct for MRI data sets). This function is defined by two parameters, a lower (L) and a higher (H) threshold. In order to handle the irrelevant part efficiently the intervals I_0 and I_{n-1} are defined as [0,L) and [H,M] respectively, where M is the maximum density value. The domain [L,H) can be subdivided using a uniform or a balanced quantization based on the histogram of the data set.

Without loss of generality, assume that the number of intervals is four ($n = 4$). In this case the intervals can be encoded on two bits, where the code “00” represents the lowest and “11” the highest interval. The binary volume containing the corresponding codes for each voxel can be stored in a 3D integer array *mask*, where an integer element contains continuously the codes of voxels along the rows parallel to the z-axis. Having a density volume of size $X \times Y \times Z$ this integer array is defined as *int mask*[*depth*][*height*][*width*], where *depth = (2 * Z + 30) div 32*, *height = Y*, *width = X* and the size of an integer is supposed to be 32 bits.

Similarly to the discussion of the previous acceleration technique first a special case is described assuming that the viewing direction is the z-axis. It is shown how to use in this special

case the binary mask for fast global and local maximum intensity projection. Afterwards the extension to arbitrary viewing directions is discussed applying a binary shear transformation.

4.2.3 Maximum intensity projection (MIP)

In order to project the maximum intensities onto the x, y plane, rays are cast into the z direction from each (x_i, y_j) grid point. Since in this special case, the sample points are located at voxel positions, the integer elements of the array *mask* store the density interval codes of the samples along the rays parallel to the z-axis. For each ray, first that interval is determined which contains the maximum density in order to avoid the resampling in the lower density intervals. The following routine compares bit patterns to find the code of the highest density interval, checking 16 samples in each step of the loop.

```
int MaxInterval(int i, int j) {
 int index = 0;
 for(int k = 0; k < depth; k++) {
  int segment = mask[k][j][i];
  if(segment) {
   int I01orI11 =
    segment & 0x55555555;
   if(I01orI11) {
    if(I01orI11 & (segment >> 1)) return 3;
    else if(index < 1) index = 1;
   }
   else index = 2;
  }
 } return index;
}
```

The variable *I01orI11* is greater than zero if the integer *segment* contains at least one “01” or “11” pattern. In this case, if the bitwise AND operation of *I01orI11* and the *segment* shifted right is greater than one then there is at least one “11” pattern found, otherwise the highest density interval code is “01”. If the value of *I01orI11* is zero then the highest density code is “00” or “10” depending on whether the value of *segment* is zero or not. Having four or more intervals a lookup table can be used in order to find the bit pattern inside an integer representing the highest density interval. The lookup table stores the index of the corresponding interval for each byte combination and the bytes in an integer are checked sequentially. After having the appropriate interval code only those samples have to be investigated, which have the same density code. For example, if the return value of *MaxInterval* is 1 the bit pattern “01” has to be searched for in the integer array *mask*. According to the offset of the found pattern the location of the corresponding voxel is determined, and the exact intensity value is compared with the current ray intensity.

```
int MIP(int i, int j) {
 int max = 0, pattern = MaxInterval(i,j);
 if(!pattern) return 0;
 for(int k = 0; k < depth; k++) {
  int segment = mask[k][j][i], pos;
  while((pos = Offset(segment, pattern)) < 32) {
   int density = volume[k * 16 + pos / 2][j][i];
   if(density > max) max = density;
```

```
   segment &= ~(0xC0000000 >> pos);
  }
 } return max;
}
```

The routine *Offset* returns with the offset of the found bit pattern passed as a second argument. Having four lookup tables of 256 byte size storing the position of the first pattern inside a byte for each interval code, this operation can be performed very fast (Figure 4.10).

```
int Offset(int segment, int pattern) {
 int pos, *lut = SelectLUT(pattern);
 pos = lut[segment >> 24];
 if(pos < 8) return pos;
 pos = lut[(segment << 8) >> 24];
 if(pos < 8) return pos + 8;
 pos = lut[(segment << 16) >> 24];
 if(pos < 8) return pos + 16;
 pos = lut[segment & 0x000000FF];
 if(pos < 8) return pos + 24;
 return 32;
}
```

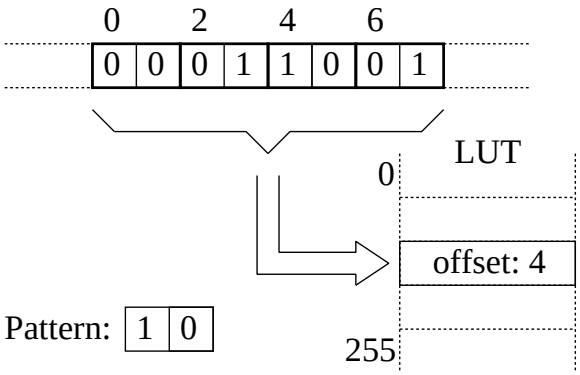


Figure 4.10: An example of a LUT entry .

Assume that, if the given byte does not contain the searched pattern then the lookup table stores 8 for this bit combination. On the other hand, if the whole segment represented by an integer does not contain the given pattern then the return value of the routine *Offset* is 32. The address of the applied lookup table is returned by the routine *SelectLUT* depending on the bit pattern required to be searched for. Note that, the processed samples are deleted from the variable *segment*, thus this lookup table technique can be used again to determine the location of the next sample.

4.2.4 Local maximum intensity projection (LMIP)

For local maximum intensity projection (LMIP) which is an extended version of MIP a similar lookup table technique can be used in order to reduce the number of samples to be taken. LMIP selects the first local maximum along a ray which is greater than a predefined threshold [50]. With an appropriate parameter setting LMIP can provide similar shading effects as the transfer

function based volume rendering and more precise geometric information like the depth and occlusion of vessels can be obtained.

In this case, just one bit is assigned to each voxel indicating whether the density is under or above the predefined threshold, thus the problem of finding the first density sample along a ray which is higher than the threshold is reduced to the problem of finding the position of the first non-zero bit inside an integer. Having a lookup table storing this position for each byte combination, the bytes of the given integer have to be checked sequentially, and the lookup table is addressed by the first non-zero byte. Taken into account the offset of the first non-zero byte and the read value the z-position of the first sample can be calculated easily. The further samples are investigated sequentially until the first local maximum is found. According to the definition of LMIP, for those rays which do not intersect any voxels of densities higher than the threshold the global maximum has to be taken. For these rays the data structure defined in the previous section can be used. Although for LMIP we cannot apply a more precise density encoding, due to the early ray termination it is usually faster than the traditional MIP.

4.2.5 Shear-warp projection

The presented MIP technique works only for a special case, when the viewing direction is parallel to the z-axis. It can be generalized to arbitrary viewing directions similarly to the rotation technique discussed in Section 4.1. In order to compute the discrete translations of the slices a Bresenham-like symmetric 3D DDA line drawing algorithm is used [54]. During the binary shear transformation, to avoid the cutting of the binary codes stored in the integer array *mask* the bits of the two bit long segments in the array *shift* have to be set to the same value. This is similar to the binary shear transformation of masks encoding several classified volumes described in Subsection 4.1.8. After having an intermediate image generated in the sheared space the final image is produced in the same way as it is described in Section 4.1.

4.2.6 Extensions

Since the rays are approximated by 3D discrete lines the algorithm does not generate exact maximum intensity projections but it can be further improved using bilinear interpolation for computing the density samples. In this case the binary density codes have to be assigned to rectangular cells on the planes perpendicular to the principal component of the viewing direction rather than to voxels. To each cell two codes are assigned representing respectively the maximum and the minimum density of the four corner voxels. In the ray casting process, first the lower bound of the global maximum is determined as the highest minimum density. Afterwards only those cells are resampled, where the encoded maximum density is not smaller than the lower bound or the current maximum. The previously presented lookup-table technique can be easily adapted to this extension.

Applying four bit density codes the shear operation is slower but since the binary volume contains more exact information about the voxel densities, much less samples have to be taken in the projection step. Ignoring the complete resampling process the highest density indices along the rays can be displayed directly yielding an image with 16 gray levels, thus the method can be used as a fast MIP previewer.

4.2.7 Implementation

The presented algorithm has been implemented in C++ and has been tested on an SGI Indy workstation. Table 4.5 summarizes the average shearing, MIP, and LMIP running times per

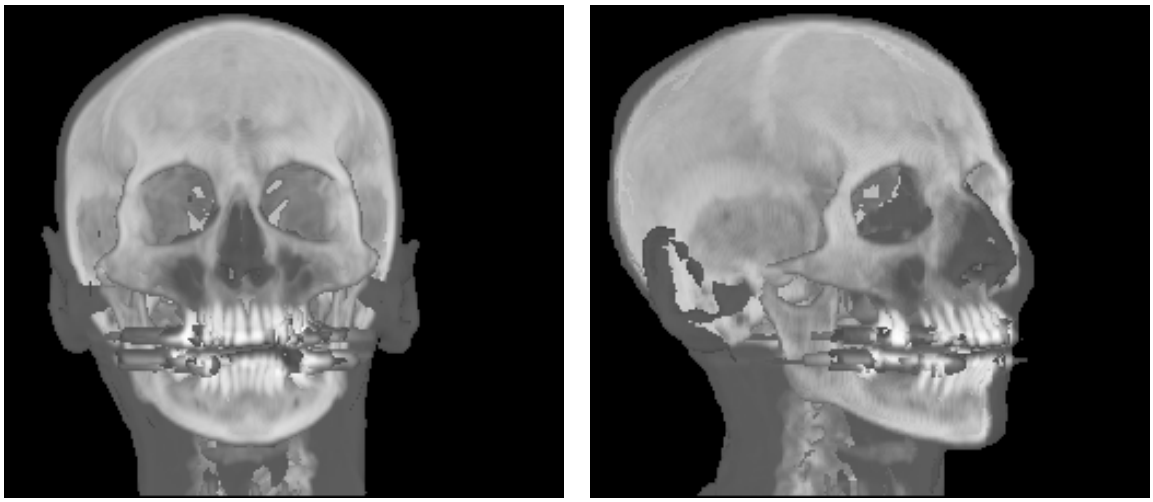


Figure 4.11: LMIP of a CT scan of a human head.

frame. We used the same data as for testing the algorithm discussed in Section 4.1, which is a CT scan of a human head available in a lower and a higher resolution.

resolution	$128 \times 128 \times 113$	$256 \times 256 \times 225$
shearing time	0.011 sec	0.104 sec
LMIP time	0.077 sec	0.432 sec
MIP time	0.193 sec	1.812 sec

Table 4.5: Running time measurements.

Since the binary shear transformation is computationally relatively cheap the effective speed of the rotation can be increased performing a couple of incremental shears between the frame generations. The average frame rate of a smooth rotation is approximately 10 Hz for the small resolution data set and 2 Hz for the high resolution volume using LMIP rendering. Figure 4.11 shows the high resolution data set rendered using LMIP. Due to an appropriate filtering, point like noise artifacts in the acquired data were not considered as local maximum values.

4.2.8 Summary

In this section a fast maximum intensity projection technique has been presented, which is also based on binary shear transformation. Due to our density encoding scheme and the applied lookup-table technique the number of density samples to be taken is significantly reduced speeding up the maximum intensity projection. The proposed method supports the local maximum intensity projection as well. The viewing rays are evaluated in front-to-back order until the first local maximum above a predefined threshold is found. In this section it has been shown that the binary shear transformation is a general acceleration tool for different volume visualization models.

4.3 Interactive iso-surface rendering

4.3.1 Introduction

In this section an interactive technique is presented which is proposed for fast shaded display of iso-surfaces [13]. The traditional way of surface rendering is the reconstruction of a triangular mesh from the volume data applying the *marching cubes* algorithm [38]. Such a mesh can be interactively rendered using the conventional graphics hardware. Nevertheless, this approach requires a time-consuming preprocessing and it does not support cutting operations. Especially in medical-imaging systems it is rather important to visualize the internal structures as well as the iso-surfaces. Usually the cross-sectional slices perpendicular to a user-defined direction are visualized for diagnostical purposes. Although the intersection of a triangular mesh and an arbitrary cutting plane could be calculated the original density values along the cutting plane are not available. Therefore, the 2D cross-sectional slices cannot be visualized.

In the last decade several direct volume-rendering techniques were published which are optimized for fast display of iso-surfaces. One alternative is to exploit frame-to-frame coherency, assuming that the volume is required to be rotated by small difference angles. Gudmundsson and Randén[23] proposed an incremental rotation technique based on this idea. Yagel and Shi[67] used a similar technique applying a hierarchical data structure for fast space-leaping.

Another approach is to extract the boundary voxels and to project them onto the image plane. Sobierajski[52] proposed a hybrid method, where the extracted surface points are converted to geometrical primitives which are rendered by a conventional graphics hardware. Saito[47] used the same strategy converting only a subset of the surface points to geometrical primitives, where the samples are selected according to a uniform distribution.

Choi and Shin[4] worked out an efficient image-based surface-rendering method which provides interactive frame rates without any specialized hardware. The limitation of their approach is the lack of the basic volume operations such as cutting.

Our fast surface-rendering method does not rely on the conventional graphics hardware to achieve high frame rates and supports interactive cutting operations as well. In order to reduce the data to be processed, the algorithm eliminates those voxels which are invisible from a specific domain of viewing directions. In contrast, the previous methods use a a view-independent extraction of boundary voxels providing weaker data reduction rate. The boundary voxels are stored in an appropriate data structure optimized for fast shear-warp projection. Since this approach is object based, mapping each voxel onto one pixel, the typical staircase artifacts can appear in the generated image. In order to reduce this aliasing not the central differences are used for normal computation but a more sophisticated gradient estimation scheme based on linear regression [43]. This technique is discussed later in Section 4.4.

4.3.2 Extraction of the potentially visible voxels

Assume that the input data is a spatial density function $f : R^3 \rightarrow R$ sampled at regular grid points, yielding a volume $V : Z^3 \rightarrow R$ of size $X \times Y \times Z$, where

$$V_{i,j,k} = f(x_i, y_j, z_k).$$

In the segmentation process a binarizing function $b : Z^3 \rightarrow \{0, 1\}$ is applied in order to define the voxels which belong to the object of interest. The value of 1 is assigned to these voxels and they are referred to as *non-empty* voxels, while to all the other voxels the value of 0 is assigned and they are referred to as *empty* voxels. Typically this function is defined as:

$$b(i,j,k) = \begin{cases} 1 \text{ if } V_{i,j,k} > t \\ 0 \text{ otherwise,} \end{cases}$$

where t is a density threshold. Generally, zero is also assigned to those voxels which are within an arbitrary cutting object. After the segmentation, the data set is reduced by eliminating the invisible voxels. Unlike the other extraction techniques, our method does not extract all the boundary voxels, but only those which are possibly visible from a certain domain of viewing angles. According to the principal component of the viewing direction 6 cases can be distinguished. Figure 4.12 depicts these cases as regions on the directional cube. Without loss of generality, assume that the principal component is the z-coordinate, thus the viewing direction is inside the domain 6. The extraction is performed according to a recursive visibility function $v : Z^3 \to \{0,1\}$ assigning 0 to the hidden voxels and 1 to the potentially visible voxels:

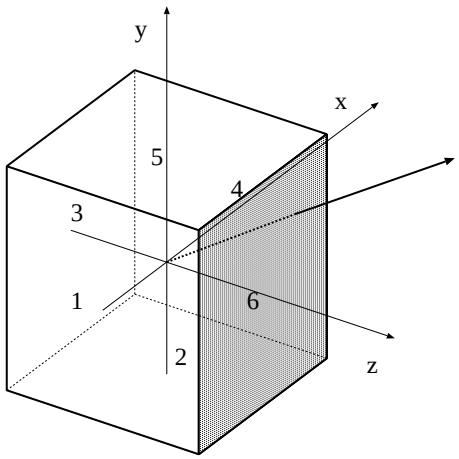


Figure 4.12: Decomposition of viewing directions into 6 regions.

$$v(i,j,k) = \begin{cases} 1 \text{ if } k = 0 \text{ or} \\ \quad \{ \exists\, l,m \mid i-1 \le l \le i+1, \\ \quad j-1 \le m \le j+1, \\ \quad v(l,m,k-1) = 1,\ b(l,m,k-1) = 0\} \\ 0 \text{ otherwise.} \end{cases}$$

A voxel at position (i,j,k) is potentially visible ($v(i,j,k) = 1$) if it belongs to the first z-slice ($k = 0$) or there exists a potentially visible, *empty* voxel at position $(l,m,k-1)$ ($v(l,m,k-1) = 1$, and $b(l,m,k-1) = 0$), where l and m are in the sets $\{i-1,i,i+1)\}$ and $\{j-1,j,j+1\}$ respectively.

Note that, if one voxel is hidden it does not necessarily mean that all the nine voxels in front of it are *non-empty* ones since an *empty* voxel can also "hide" another voxel if it is hidden. This incremental extraction can eliminate much more occluded voxels than a scanning which takes into account only the local neighborhood. Figure 4.13 demonstrates this procedure in 2D, where one pixel can be occluded by the three pixels located in front of it. Because of the recursive definition of the visibility function the rows are processed in front-to-back order. The pixels with dots represent *non-empty* voxels, while the other pixels depict the *empty* ones. Having the visibility calculation executed, the gray pixels are hidden and the white ones may be visible assuming the given range of viewing directions. Our method extracts only the potentially visible

non-empty voxels. In contrast, other similar techniques select all the exterior boundary voxels, requiring a more complicated preprocessing and providing worse data-reduction rate.

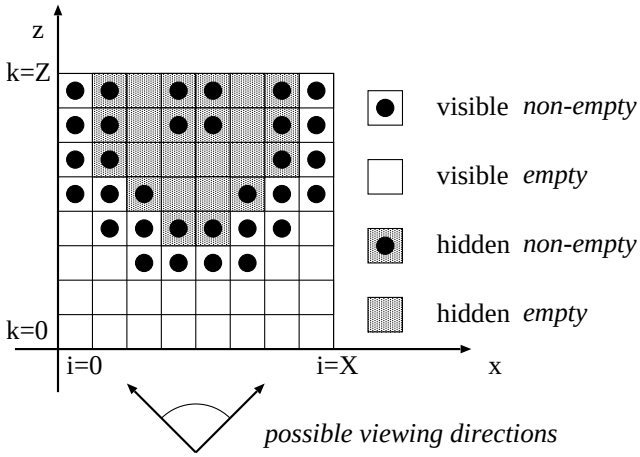


Figure 4.13: Extraction of the potentially visible non-empty voxels.

After the extraction of the boundary voxels they have to be stored in an appropriate data structure which contains all the information necessary for the rendering stage. This information includes the original data value (density), the color, the position vector, and the approximated gradient vector. The original density value can be used for the gray-scale rendering of the cutting planes, while the gradient vector is required as a surface normal for the view-dependent shading. Assuming that the light sources are rotated together with the object and the view-independent Lambertian shading model is used the gradient components do not have to be stored. In this case the shaded colors are precalculated for each extracted boundary voxel. As the gradient estimation is also performed only for the relatively small number of extracted potentially visible voxels, instead of calculating the central differences a more sophisticated derivative filter can be applied keeping the preprocessing time in a reasonable range.

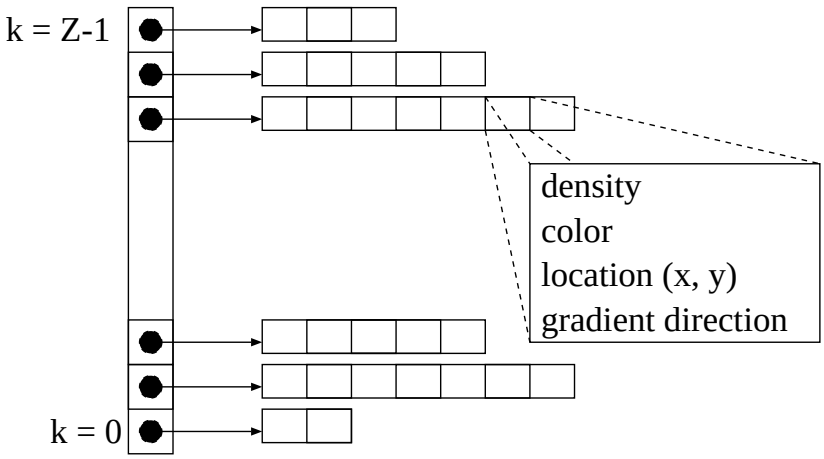


Figure 4.14: The data structure storing the boundary voxels.

In order to make the rendering step fast the surface points are stored in lists separately for each slice perpendicular to the z-axis (Figure 4.14). For efficiency reasons, these lists are represented by fixed size arrays, therefore in the preprocessing the boundary voxels are collected

in a maximized temporary array. Having a slice processed, the data fields are copied into a newly allocated array containing as many entries as the number of the boundary voxels in the given slice. Since in this data structure the z-coordinates of the surface points are stored implicitly the array elements contain only the x and y components of the voxel position.

4.3.3 Shear-warp projection

In order to efficiently render the potentially visible boundary voxels shear-warp projection[31] is used (Figure 3.1). As the boundary voxels are sorted according to z-depth, the hidden-voxel removal is performed automatically projecting the slices in a descending depth order overwriting the pixel values in the intermediate image.

The previous methods[23][52] usually use a z-buffer for this purpose, because they store all the surface points in one single list. Before the projection the depth value has to be checked in the z-buffer which requires a conditional jump decreasing the efficiency of the pipeline mechanism of the instruction execution. Furthermore, projecting each voxel to one pixel, holes may appear in the image. Applying the splatting technique with an appropriate footprint kernel, this artifact can be avoided, but it could drastically influence the performance.

For the sake of efficiency, our method also maps each voxel to one pixel, but in order to avoid holes, an intermediate image of size $(X + Z) \times (Y + Z)$ is generated, where the neighboring voxels are mapped to neighboring pixels. This mapping is very simple in the sheared space. To each voxel location the 2D offset vector of the given slice is added. This offset is calculated for each slice in advance and only once whenever the viewing direction is changed. Assuming that the principal component of the viewing direction is the z coordinate the maximum absolute translation of a boundary voxel is $Z/2$ in the direction of the x-axis or the y-axis. Therefore, the temporary image will contain all the projected boundary voxels.

The slice offsets are calculated using a Bresenham-like symmetric 3D DDA line drawing algorithm [54]. In order to compute more accurate pixel values bilinear interpolation can also be applied taking into account the exact translations of the slices. This quality improvement can drastically increase the rendering time. In a practical implementation, for fast rotation the approximating discrete rays are used (it is equivalent with a nearest-neighbor resampling) and only the final image is rendered from bilinearly interpolated samples.

Having the intermediate image generated, it has to be mapped onto the final image using a 2D warp operation. The scaling factors can be built into the warp matrix, thus the size of the final image does not necessarily depend on the size of the original volume. In fact, the intermediate image is used as a texture map as it is described in Section 4.1

If the shading model is view-dependent (like the classical Phong model) then for each extracted boundary voxel an approximated surface normal has to be stored. A normal vector is represented by two polar coordinates and each polar coordinate is quantized onto a 6 bit integer. Therefore, a surface normal can be stored in a 16 bit integer, which can be used as an address to a lookup table containing the precalculated shading factors. This lookup table is refreshed for each possible normal vector whenever the lighting conditions are changed.

The voxels intersected by a cutting plane are handled differently since they are not shaded at all. Typically the pixel color is overwritten by the density of the projected voxel. Generally, an arbitrary function can be used which maps the original data value to a color. Since the shading process has to ignore these voxels, to each pixel of the intermediate image an additional attribute has to be assigned indicating whether the corresponding voxel is a boundary voxel or a voxel intersected by the cutting plane.

4.3.4 Decomposition of the viewing directions

The presented surface-rendering method can be extended to arbitrary viewing directions since the preprocessing can be executed for any principal direction. A further opportunity of improvement is to decompose the domain of the viewing directions into 24 regions instead of 6. Figure 4.15 depicts these regions on the directional cube. Assume that the principal component of the viewing direction is the z coordinate. According to the signs of the x and y components four cases can be distinguished. For example, if both of them are positive the visibility function is defined as:

$$v(i,j,k) = \begin{cases} 1 \text{ if } k = 0 \text{ or} \\ \quad \{ \exists\, l,m \mid i-1 \leq l \leq i, \\ \quad j-1 \leq m \leq j, \\ \quad v(l,m,k-1) = 1,\ b(l,m,k-1) = 0\} \\ 0 \text{ otherwise.} \end{cases}$$

Since in this case we have a stronger condition (instead of nine, only the four voxels are checked which could hide the currently tested voxel from the specific viewing direction) defining the potentially visible surface points the number of extracted voxels will be lower than in the general case. The visibility function can be defined similarly for the other regions. In order to render the volume from an arbitrary viewing angle the preprocessing has to be performed for all the 24 regions. In the rendering phase the appropriate data structure is selected according to the current viewing direction. Although this modification increases the preprocessing time and the storage requirements, later we will show that it significantly improves the extraction rate, making the rendering process much faster.

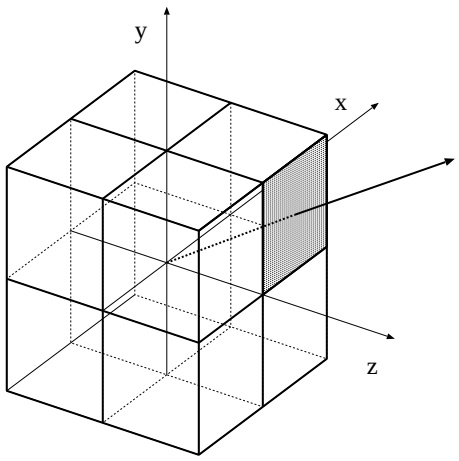


Figure 4.15: Decomposition of viewing directions into 24 regions.

4.3.5 Interactive cutting operations

A major drawback of the method presented so far is that the cutting planes have to be defined in advance, since the surface points eliminated by the cutting operation are not stored any more. Thus after the preprocessing the location and the orientation of the cutting planes cannot be modified interactively, although it would be rather important in a medical imaging application. In this section we discuss how to add this feature without significant reduction in performance.

In the preprocessing the cutting planes are not taken into account and only the potentially visible voxels are extracted. In the projection phase, due to the cutting operations internal voxels intersected by the cutting plane have to be rendered as well, thus the original volume needs to be kept in the main memory. Assume that we have only one cutting plane. The surface points and the internal voxels are projected separately, where the order depends on the current viewing direction. The cutting plane divides the space into two halfspaces. If the surface points are located in the halfspace which contains the viewpoint then they are projected after the rendering of the internal voxels intersected by the cutting plane. Otherwise the projection order is reversed.

Before the projection, the potentially visible voxels have to be checked whether they are in the positive or negative halfspace. It requires the substitution of the voxel coordinates into the implicit equation of the current plane:

$$a \cdot x + b \cdot y + c \cdot z + d = r.$$

The sign of the residual r indicates whether the given voxel has to be rendered or ignored. Since the voxels are stored sorted by the z coordinate, the whole expression need not to be evaluated for each single voxel. The subexpression $c \cdot z + d$ is evaluated only once for each z-slice. For further optimization, the surface points inside a slice can be sorted by the y coordinates as well. This does not increase the preprocessing time since performing the visibility calculation row-continuously the surface points are sorted automatically. Therefore, the elimination of the cut voxels requires practically just one multiplication and one conditional instruction additionally, thus this extension does not significantly effect the performance.

The voxels intersected by the cutting plane are projected separately. Without loss of generality, let us assume that the principal component of the plane normal is the z-coordinate. In this case all the possible discrete (x, y) pairs ($x \in \{0, 1, ..., X-1\}, y \in \{0, 1, ..., Y-1\}$) are substituted into the explicit equation of the plane and the obtained z value is used for addressing the original volume. The internal voxels are rendered using the gray-scaled data values. Before projecting these voxels onto the image plane they are checked whether they belong to the region of interest, otherwise the low density voxels representing the surrounding air could hide the surface points. In order to avoid this, an additional segmentation function can be used which is not necessarily the same as the one used for the extraction of the surface points ($b(i, j, k)$). For example, when the surface of the skull is required to be rendered the same thresholding segmentation is not usable to determine the region of interest in the slice defined by the cutting plane, since it can contain voxels with densities lower than the bone threshold.

If the purpose of cutting is the visualization of the internal surface then only the extracted surface points located in the positive (or negative) halfspace are rendered. Unfortunately the incremental extraction of the boundary voxels cannot be used in this case, thus only the local neighborhood is taken into account in the visibility calculation.

Another opportunity of improvement is the restriction of the cutting plane orientations. If only the cutting planes which are perpendicular to one of the major axes are allowed then the projection phase can be further optimized. The implementation of the cutting plane perpendicular to the z-axis is the simplest. In this special case there is no need to check each single voxel before projection, since the slices behind (or in front of) the given z depth are simply ignored.

The cutting planes perpendicular to the x-axis or to the y-axis are also supported by the proposed data structure with the following slight modification. The surface points are sorted inside the z-slices by the x and y coordinates as well, and these two lists are stored separately for each z-slice. In this case, the cutting planes can be incrementally translated, since for each list a pointer can be introduced indicating the border between the voxels behind and in front of

the current plane.

4.3.6 Implementation

The presented surface-rendering method has been implemented in C++ and it has been tested on a *Silicon Graphics Indy* workstation. The test data was the CT scan of a human head used for testing the previously presented algorithms. Table 4.6 and Table 4.7 summarize the run-time measurements for a volume with two different resolutions. The scaling factors in the warp operation are set to one, so the image sizes are 128×113 and 256×225 respectively. In the preprocessing, a thresholding function was used in order to segment the skull. Using the view-dependent Phong shading to model specular surfaces the shaded colors are calculated during the rendering process. In case of just a diffuse surface the colors are determined in advance for each boundary voxel according to the view-independent Lambertian shading model.

shading model	Lambert	Phong
preprocessing	7 sec	6 sec
intermediate image	9 msec	52 msec
final image	4 msec	4 msec
frame rates	76.9 Hz	17.9 Hz

Table 4.6: Test results for the volume of resolution $128 \times 128 \times 113$.

shading model	Lambert	Phong
preprocessing	56 sec	55 sec
intermediate image	45 msec	213 msec
final image	18 msec	18 msec
frame rates	15.9 Hz	4.33 Hz

Table 4.7: Test results for the volume of resolution $256 \times 256 \times 225$.

In the preprocessing, decomposing the domain of viewing directions into 24 regions 1.8% of the voxels were extracted from the low resolution data set. For the high resolution volume this percentage was 0.9%. For the sake of comparison, having the data only for 6 regions preprocessed the extraction rates were 3.4% and 1.8% respectively.

Beside the extraction of the boundary voxels, the preprocessing time includes the gradient estimation and the shading if the surface is diffuse. The generation of the intermediate image consists of the voxel projection in the sheared space and the real-time shading if the surface is specular. The final image is produced from the intermediate image using a 2D warp operation. Since this procedure includes the scaling and the rendering as well, its run-time performance depends linearly on the image size.

Figure 4.16 and Figure 4.17 show the rotation of an iso-surface shaded according to the Lambert and Phong models respectively. Although the rendering of the tissue boundary as a diffuse surface is approximately four times faster than rendering it as a specular surface, it assumes that the light sources are rotated together with the object. The view-dependent Phong shading gives a better spatial impression, since the object can be illuminated always from the viewing direction.

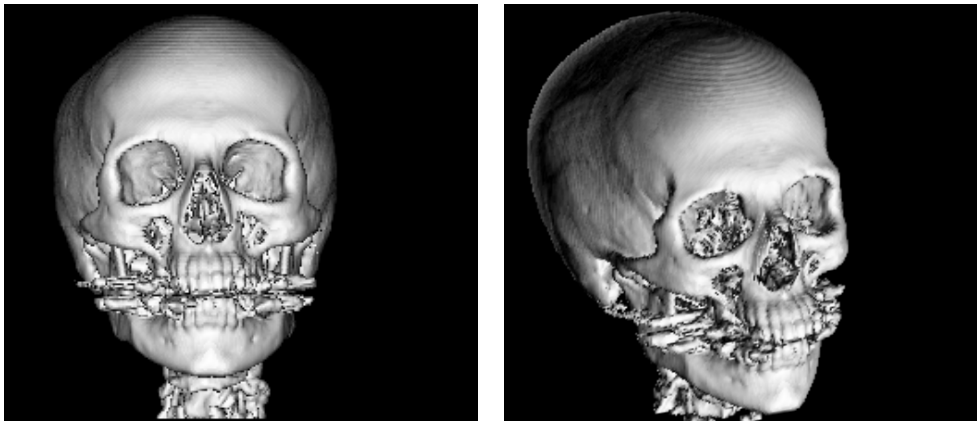


Figure 4.16: Rotation of a skull using Lambertian shading.

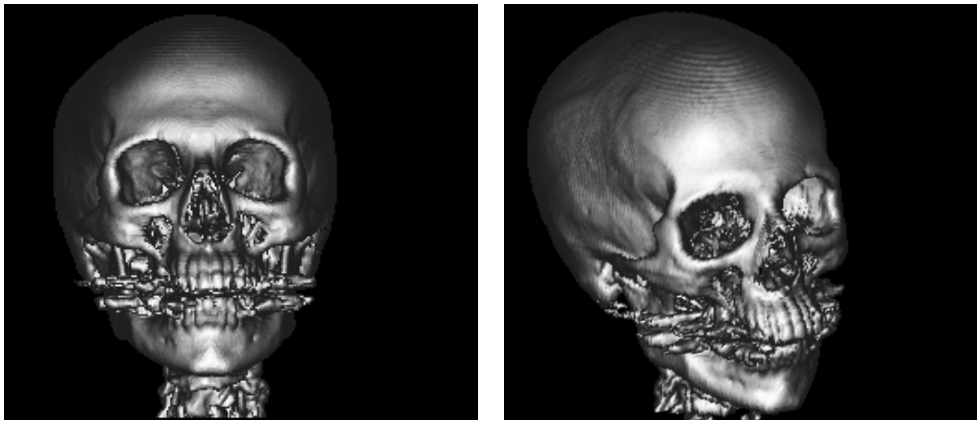


Figure 4.17: Rotation of a skull using Phong shading.

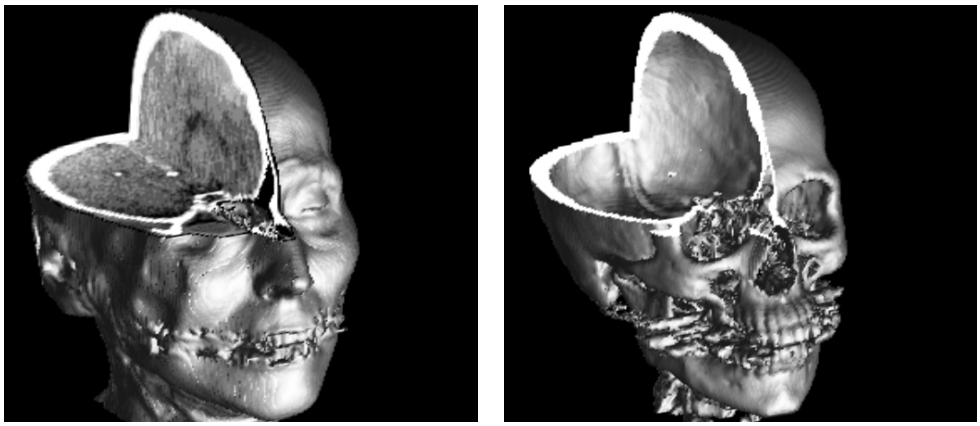


Figure 4.18: Cutting operations.

Figure 4.18 shows the application of cutting planes. In the left image the voxels intersected by the cutting planes are rendered using gray-scale density values, while on the right image they are considered transparent in order to visualize the internal surfaces.

shading model	Lambert	Phong
xy-plane	83 Hz	23 Hz
xz-plane	55 Hz	20 Hz
yz-plane	55 Hz	20 Hz
arbitrary plane	47 Hz	15 Hz

Table 4.8: Frame rates using different cutting planes (volume resolution: $128 \times 128 \times 113$*).*

Table 4.8 and Table 4.9 contain the frame rates for the two different resolution data sets, when interactive cutting is used. The implementation of the axis parallel cutting planes is optimized as it is discussed in Section 3.4. The frame rates have been measured calculating the average rendering times for all the possible translations of the planes.

shading model	Lambert	Phong
xy-plane	17 Hz	6 Hz
xz-plane	12 Hz	5 Hz
yz-plane	12 Hz	5 Hz
arbitrary plane	10 Hz	4 Hz

Table 4.9: Frame rates using different cutting planes (volume resolution: $256 \times 256 \times 225$*).*

4.3.7 Summary

In this section a fast iso-surface rendering method has been presented, which provides real-time rotation without using any specialized hardware. Therefore, it can be widely used in 3D diagnostical systems even on low-end machines.

The basic idea is the extraction of potentially visible boundary voxels. The preprocessing is performed for 6 or 24 regions of viewing directions achieving higher efficiency than other techniques which extract all the boundary voxels. The surface points are stored in a data structure which supports fast shear-warp projection. Since in the sheared space the neighboring voxels are projected to neighboring pixels holes will not appear in the intermediate image. The implementation of zooming is simple since the scaling factors are built into the warp matrix which transforms the intermediate image into the final image.

Due to the direct volume-rendering approach conventional volume operations such as cutting are also supported. The voxels intersected by an arbitrary cutting plane are rendered using the original density values, while the surface points are shaded according to the estimated normal vectors.

4.4 Normal estimation based on 4D linear regression

The fast volume rendering algorithms presented in Section 4.1 and Section 4.3 have a common drawback. Because of the limitation of shear-warp projection super sampling using trilinear interpolation is not supported. A low sampling rate can cause typical staircase artifacts especially when the estimation of the original gradients is not accurate enough. In order to compensate the disadvantages of rare resampling, in this section, a new normal estimation scheme is presented which provides a smooth approximated gradient function [43].

4.4.1 Introduction

In direct volume rendering the gradients are used for shading as surface normals. Therefore, the quality of the generated image is strongly influenced by the applied gradient estimation method. Volumetric data is usually obtained by sampling continuous objects and after the discretization the exact surface normals are not available anymore. Therefore, the inclination of the surfaces is estimated investigating a close neighborhood of a given voxel. A possible way of evaluation of different normal computation techniques is to discretize continuous geometrical models and to compare the estimated normals to the exact original ones. Unfortunately, this strategy cannot be used for practical data sets, like medical CT scans, since the exact normal vectors are not known. In a typical volume, there are no sharp edges and the surfaces are rather smooth. Therefore, one can expect that the surfaces and the contours of different organs are displayed smoothly with reduced staircase artifacts. In order to fulfill this requirement our method integrates the filtering and the normal computation into one process.

In [66] Yagel overviews several methods for discrete normal estimation and analyses their performance. Depending on the neighborhood considered, these techniques can be classified into two fundamentally different categories as *image space* and *object space* methods.

Image space techniques take only the 2D neighborhood in the projected image into account, therefore they are view-dependent. *Depth-gradient shading* [6][55] as a representative of image space methods, approximates the gradient from the z-buffer calculating the differences between the depths of the given and the neighboring pixels. This approach produces sharp contours where in the neighboring pixels different objects are visible or where there is a drastic jump between the depth values. In order to avoid this artifact *context sensitive normal estimation* can be used which takes also these object and slope discontinuities into account. The basic idea of this approach can be applied to object space techniques too[66].

Object space methods estimate the normal according to the 3D neighborhood of the given voxel. *Constant shading* [24] which is based on the cuberille method is an early example of this category. The voxels are considered to be unit cuboids, and the normals at each point of a boundary surface are the true normals of the corresponding cuboid faces. *Normal-based contextual shading* [25][3] is based on the cuberille method as well. This technique additionally takes the orientation of the adjacent visible faces into account increasing the number of possible normal vectors and giving a better impression about the inclination of a boundary surface. *Gray-level gradient shading* [26][27] is used for volumes, where each voxel represents a gray-level value. The gradient vectors are estimated according to the neighborhood of the voxels using traditional derivative filters [42]. *Contextual shading* fits a local approximate plane [2] or a biquadratic [61][62] function to the set of points that belong to the same iso-surface. These methods are time-consuming and limited to a certain neighborhood. Bryant and Krumvieda [2] solve a set of linear equations by Gaussian elimination in order to obtain an approximate tangent plane at a given surface point. Webber's technique[61][62] is similar, but in a 26-neighborhood the surface is approximated by a biquadratic function producing accurate results for objects

with C^1 continuous faces.

According to our approach the normal estimation is extended to a 4D linear regression problem and not restricted to the approximation of an iso-surface. In a local neighborhood the density function is approximated with a 3D hyper-plane taking not only the surface points but all the neighboring voxels into account, using an appropriate weighting function. Since a plane is defined by a normal vector and a translation, a 4D linear equation system is solved in order to minimize the error of the approximation. This seems to be more complicated than the previous contextual shading techniques but we will show that it leads to a computationally efficient convolution, thus the linear equation does not need to be solved using the traditional time-consuming methods of linear algebra, like Gaussian elimination. Furthermore, our technique provides not only an estimated normal vector but a translation value as well, which can be considered as a filtered value for the given voxel location. By substituting the original density with the filtered value, smooth surfaces can be displayed and the staircase artifacts can be reduced.

4.4.2 Linear regression

Assuming that the origin of the coordinate system is translated into the position of the current voxel, the density function $f(x, y, z)$ in a close neighborhood can be approximated linearly according to the following formula:

$$f(x, y, z) \approx A \cdot x + B \cdot y + C \cdot z + D. \tag{4.5}$$

This approximation tries to fit a 3D regression hyper-plane onto the measured density values assuming that the density function changes linearly in the direction of the plane normal $n = [A, B, C]$. The value of D which is the approximate density value at the origin of the local coordinate system determines the translation of the plane.

Evaluating this approximation for the voxels of the local neighborhood the error can be measured using the following mean square error calculation:

$$E(A, B, C, D) = \sum_{k=0}^{26} w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k)^2. \tag{4.6}$$

The coordinates x_k, y_k, z_k denote the components of the neighboring voxel locations in the coordinate system translated into the center of the subvolume representing the local neighborhood. The measured density value in the kth voxel position is denoted by f_k. The error of the kth sample contributes to the global mean square error with weight w_k. The weighting function is assumed to be an arbitrary, spherically symmetric function, which is monotonically decreasing as the distance from the origin is getting larger.

The k indices are assigned to the neighboring voxel locations row-continuously (Figure 4.19). For the sake of clarity but without loss of generality, we assume that only the 26-neighborhood is taken into account. In this case, the index k of voxel $V_{x,y,z}$ in the 26-neigborhood of the current voxel $V_{0,0,0}$ is defined as:

$$k = (z + 1) \cdot 9 + (y + 1) \cdot 3 + x + 1. \tag{4.7}$$

In order to minimize the 4D error function $E(A, B, C, D)$, the partial derivatives according to the four unknown variables A, B, C, D are investigated:

$$\frac{\partial E}{\partial A} = 2 \cdot \sum_{k=0}^{26} w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot x_k, \tag{4.8}$$

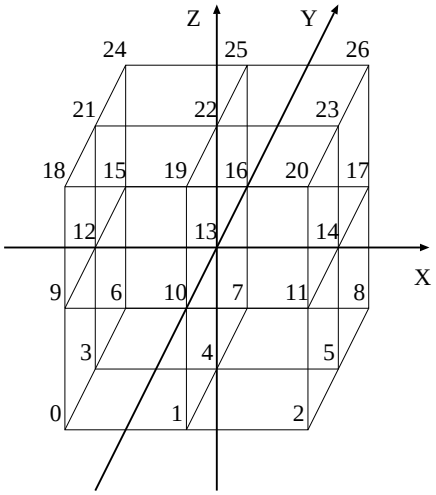


Figure 4.19: Indexing of the neighboring voxels.

$$\frac{\partial E}{\partial B} = 2 \cdot \sum_{k=0}^{26} w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot y_k, \tag{4.9}$$

$$\frac{\partial E}{\partial C} = 2 \cdot \sum_{k=0}^{26} w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot z_k, \tag{4.10}$$

$$\frac{\partial E}{\partial D} = 2 \cdot \sum_{k=0}^{26} w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k). \tag{4.11}$$

In a minimum location of the error function these partial derivatives are equal to zero. This condition leads to the following system of linear equations:

$$M \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} \sum w_k f_k x_k \\ \sum w_k f_k y_k \\ \sum w_k f_k z_k \\ \sum w_k f_k \end{bmatrix}, \tag{4.12}$$

where

$$M = \begin{bmatrix} \sum w_k x_k^2 & \sum w_k x_k y_k & \sum w_k x_k z_k & \sum w_k x_k \\ \sum w_k x_k y_k & \sum w_k y_k^2 & \sum w_k y_k z_k & \sum w_k y_k \\ \sum w_k x_k z_k & \sum w_k y_k z_k & \sum w_k z_k^2 & \sum w_k z_k \\ \sum w_k x_k & \sum w_k y_k & \sum w_k z_k & \sum w_k \end{bmatrix}.$$

Note that the elements of the coefficient matrix M are constants, thus only the right side of the matrix equation depends on the measured f_k values. Assuming that the voxels are located at regular grid points, where the sampling distance is the same in the three major directions the equation is further simplified. In this case, the coefficient matrix M is a diagonal matrix since all the elements except the diagonal ones are equal to zero because of symmetry reasons ($x_k, y_k, z_k \in \{-1, 0, 1\}$ and the weights w_k are symmetric to the origin, therefore each non-zero term in the sum has a pair with an opposite sign):

$$M = \begin{bmatrix} \sum w_k x_k^2 & 0 & 0 & 0 \\ 0 & \sum w_k y_k^2 & 0 & 0 \\ 0 & 0 & \sum w_k z_k^2 & 0 \\ 0 & 0 & 0 & \sum w_k \end{bmatrix}. \tag{4.13}$$

Such a linear equation can be solved very easily, since the inverse of the coefficient matrix is also a diagonal matrix containing in the diagonal the reciprocals of the original matrix elements. Thus the unknown vector $[A, B, C, D]$ is calculated by weighting the components of the right side. Let us introduce the following weights for each unknown variable:

$$w_A = \frac{1}{\sum_{k=0}^{26} w_k x_k^2},\ w_B = \frac{1}{\sum_{k=0}^{26} w_k y_k^2},$$
$$w_C = \frac{1}{\sum_{k=0}^{26} w_k z_k^2},\ w_D = \frac{1}{\sum_{k=0}^{26} w_k}. \tag{4.14}$$

The solution of the matrix equation leads to a simple linear convolution:

$$A = w_A \sum_{k=0}^{26} w_k f_k x_k,\ B = w_B \sum_{k=0}^{26} w_k f_k y_k,$$
$$C = w_C \sum_{k=0}^{26} w_k f_k z_k,\ D = w_D \sum_{k=0}^{26} w_k f_k. \tag{4.15}$$

Assuming that the sampling distances along the three major axes are the same the weights w_A, w_B, w_C are equal to each other. Thus these weights can be ignored since the estimated gradient $[A, B, C]$ has to be normalized anyway in order to obtain a surface normal of unit length. The gradient magnitude might also be used in the rendering stage for emphasizing the boundaries of iso-surfaces. In this case the weights w_A, w_B, w_C can be ignored as well, since only the relative differences between the gradient magnitudes are important.

Note that the value of variable D is a normalized weighted sum of the measured values in the local neighborhood thus it can be considered as a filtered value. This is the result of the approximation in the origin of the local coordinate system ($f(0, 0, 0) = D$). Together with the approximate normal components these filtered values are stored in a newly generated volume. In this volume there is a strong correlation between the data values and the corresponding normals since in the grid points the error of the linear approximation which has been assumed in the normal estimation is minimal. Therefore in the ray casting process this volume is used instead of the original one. In a typical volume-rendering application, in order to reduce the noise in the data set and to smooth the surfaces low-pass filtering is used. This filtering process is completely separated from the gradient estimation. In our approach the smoothing and the normal estimation are performed in one step in a consistent way, using the same function for weighting the contribution of the neighboring voxels.

4.4.3 Interpolation

In direct volume rendering the approximate gradient vectors are usually calculated in advance at the grid points, in a preprocessing step. In the ray casting stage, a normal vector at an arbitrary sample point is calculated from the gradients of the eight closest voxels using trilinear interpolation. Wherever there is a big difference between the gradients at the eight corner voxels of the given cubic cell the typical staircase artifacts appear.

In order to avoid this problem the linear regression can be evaluated at the sample points along the viewing rays as well yielding a continuous reconstruction of the density function. Generally, the coefficient matrix M in Equation 4.12 will not be diagonal because of the asymmetric weights (the distances from the grid points of the neighborhood are different). Furthermore the entries depend on the position. Although the solution of the linear equation requires just a matrix multiplication, the evaluation of the inverse coefficient matrix is computationally

rather expensive. This problem can be handled by dividing each cell into subcells with a regular subgrid (Figure 4.20). The inverse matrix is evaluated in advance for each corner point of the subcells. This has to be done only once for one generic cell. In the ray casting process trilinear interpolation is applied for the subcells, where the normals at the corner points are calculated using the precalculated inverse matrices. This modification provides a more accurate approximation of the density function although it increases the rendering time.

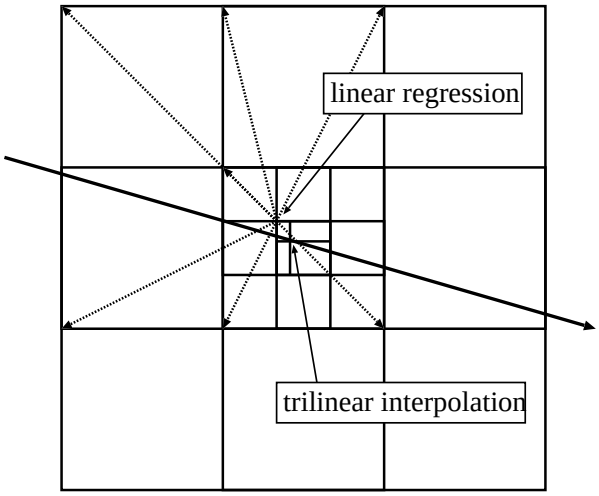


Figure 4.20: Subdivision of the original grid.

Another alternative is to use the approximating hyper-planes for density interpolation. First a density d_0 is computed from the filtered values at the eight closest grid points (which are the translations of the approximating hyper-planes), using trilinear interpolation. The obtained value d_0 cannot be larger than the maximum corner density of a cubic cell. Taking also the inclination of the surface into account another density d_1 is calculated the following way. The current sample location is substituted into the plane equations at the eight closest grid points and d_1 is trilinearly interpolated from the obtained values. Since this computation is not restricted to a cubic cell, the value d_1 might be larger than the maximum corner density depending on the influence of the neighboring cells. In order to sample the densities along the rays an arbitrary normalized weighted sum $\mu_0 \cdot d_0 + (1 - \mu_0) \cdot d_1$ can be used. Increasing the weight μ_0 the influence of the local inclination is getting stronger and setting μ_0 to one results the traditional trilinear interpolation. In our experiments we used the value $\mu_0 = 0.5$ in order to interpolate the densities with a quadratic function. It can be considered as an acceptable compromise between trilinear and B-spline interpolation.

4.4.4 The weighting function

The weighting function w_k of the convolution can be an arbitrary, spherically symmetric function which is, apart of the origin, monotonically decreasing as the distance from the origin is getting larger. For example, the reciprocal of the square of the Euclidean (or Manhattan) distance can be used for weighting the neighboring voxels:

$$w_k = \begin{cases} 0 \text{ if } k = 13 \\ \frac{1}{d_k} \text{ otherwise,} \end{cases} \tag{4.16}$$

where d_k is the distance of the kth neighboring voxel from the central voxel. Note that the

classical gradient estimation based on central differences is the special case of our method using the following weighting function:

$$w_k = \begin{cases} 1 \text{ if } k = 4 \text{ or } k = 22 \\ \text{or } k = 10 \text{ or } k = 16 \\ \text{or } k = 12 \text{ or } k = 14 \\ 0 \text{ otherwise.} \end{cases} \tag{4.17}$$

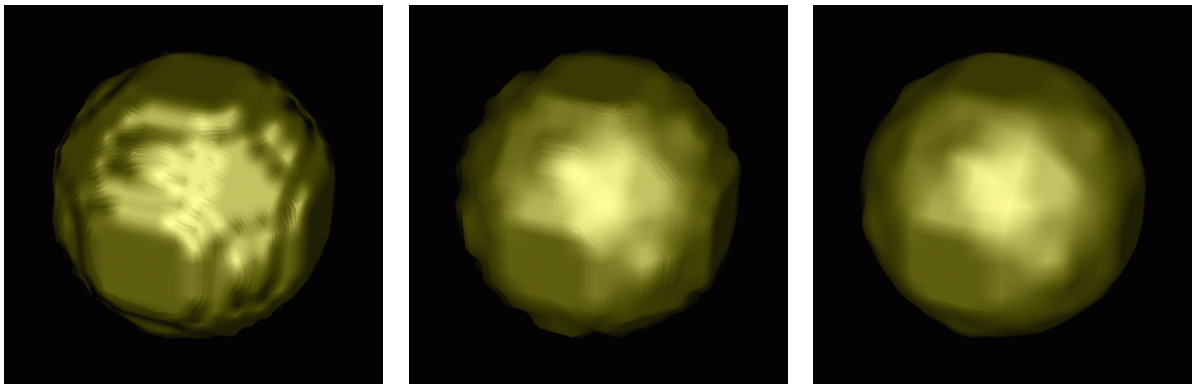
Thürmer's technique [56] which has been proposed for normal estimation in binary volumes is also a special case of our method. According to this approach the N_x, N_y, and N_z components of the estimated normal vector are calculated according to the following formula:

$$N_x = \sum_{k=0}^{26} w_k \sigma_k x_k, \; N_y = \sum_{k=0}^{26} w_k \sigma_k y_k, \; N_z = \sum_{k=0}^{26} w_k \sigma_k z_k, \tag{4.18}$$

where $\sigma_k = 1$ if the value of the kth binary voxel in the certain neighborhood is one and zero otherwise. Having a binary volume the density function f takes only the values of zero and one therefore in this special case our method provides the same normal components. It can be considered as a generalization of the previous normal estimation techniques, and can be used for gray-scale and binary data sets as well. Furthermore, our approach provides also a filtered value which is consistent to the estimated normal vector. Substituting the original densities with the filtered values the typical staircase artifacts of direct volume rendering can be reduced.

4.4.5 Implementation

The proposed normal estimation method has been tested on binary and gray-scaled data sets. Figure 4.21 shows a binary volume of resolution $20 \times 20 \times 20$ obtained by discretization of a sphere.

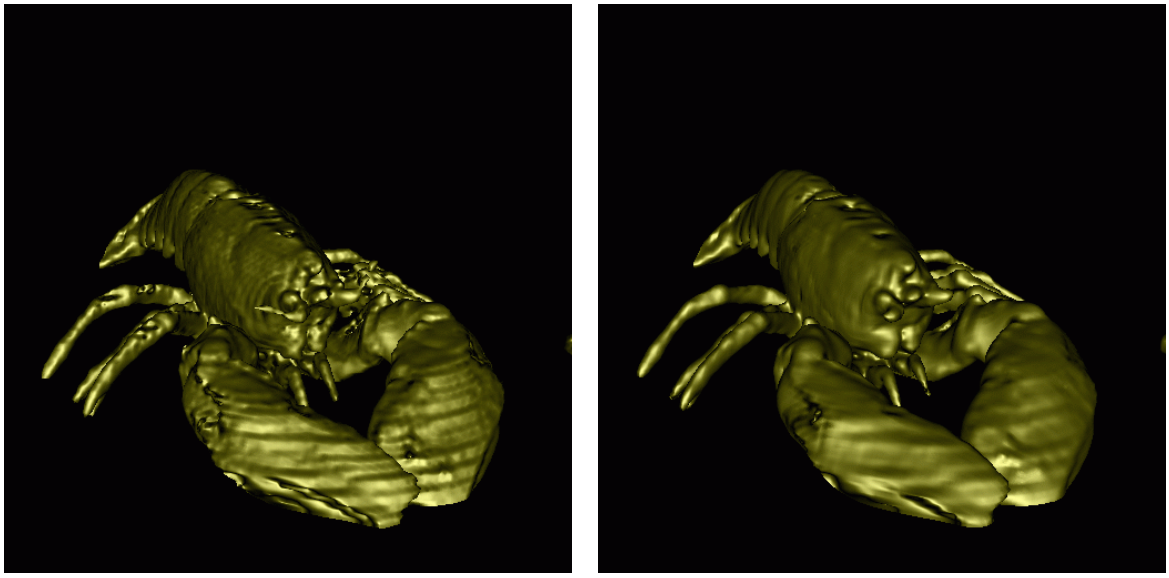


(a) (b) (c)

Figure 4.21: Normal estimation on binary volume data using central differences (a), Thürmer's method (b), and linear regression, where $\mu_0 = 0.5$ (c).

Image (a) was rendered calculating central differences to estimate the surface normals, therefore the typical staircase artifacts appear. In image (b), where normals are estimated from the 26-neighborhood of the voxels using Thürmer's method the surface is much smoother but the contour of the object has the same discontinuities as in image (a). Image (c) was rendered using our method, where the regression plane at each voxel location was calculated from the voxels

of the 26-neighborhood. According to the linear regression the original data values are allowed to be modified in order to minimize the error of the approximation. Therefore the contour of the object is smoother and approximates the original contour much better than in the previous two images. Processing binary volumes, Thürmer's technique [56] and our method provides exactly the same normal vectors at the grid points. Nevertheless, the intersection points, where the normals are evaluated using trilinear interpolation, are different since the linear regression slightly changes the original data values. Although the estimated normal components are the same the surface in the right image is much smoother since the calculated intersection points are closer to the exact intersection points of the sphere and the viewing rays. These images clearly show that in rendering binary data sets not only the estimated normal components are important but also the sample locations, where the interpolated normals are evaluated.



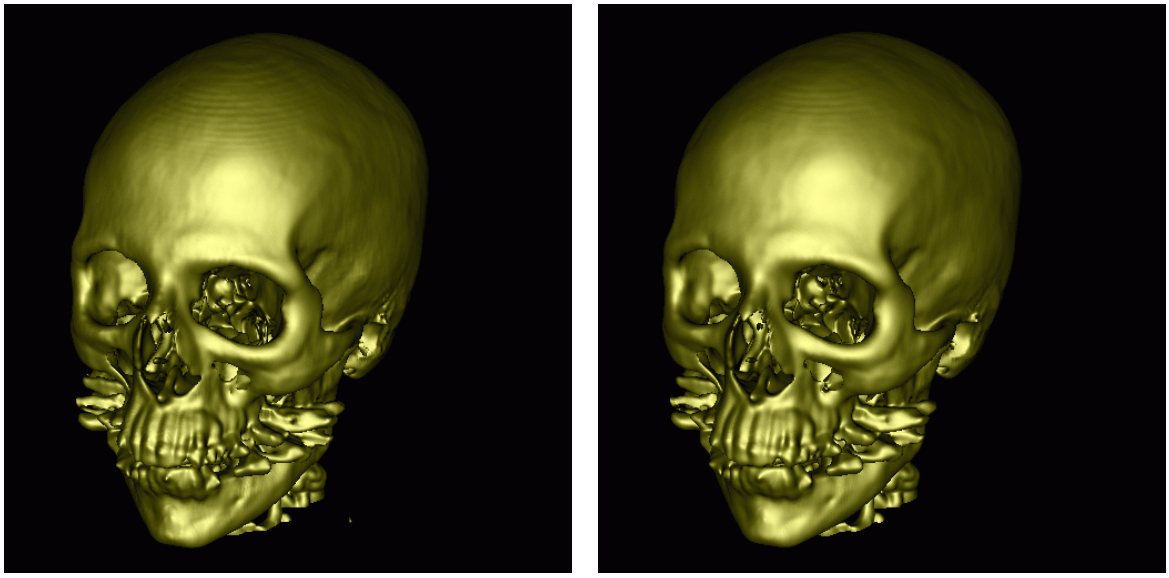
(a) (b)

Figure 4.22: A lobster rendered calculating central differences for gradient estimation (a) and using linear regression (b).

Figure 4.22 shows a gray-scale data set obtained by a CT scan of a lobster. The data set has been rendered calculating central differences (a) and using linear regression (b) for gradient estimation. Although, in image (b) some high frequency details are filtered, the contours of the different parts of the body are much sharper then in image (a), therefore they can be distinguished more easily. For example, the location and the shape of the legs can be perceived much better in the right image providing stronger spatial impression. In contrast, the left image contains some noisy regions, where the topology of the object cannot be recognized at all.

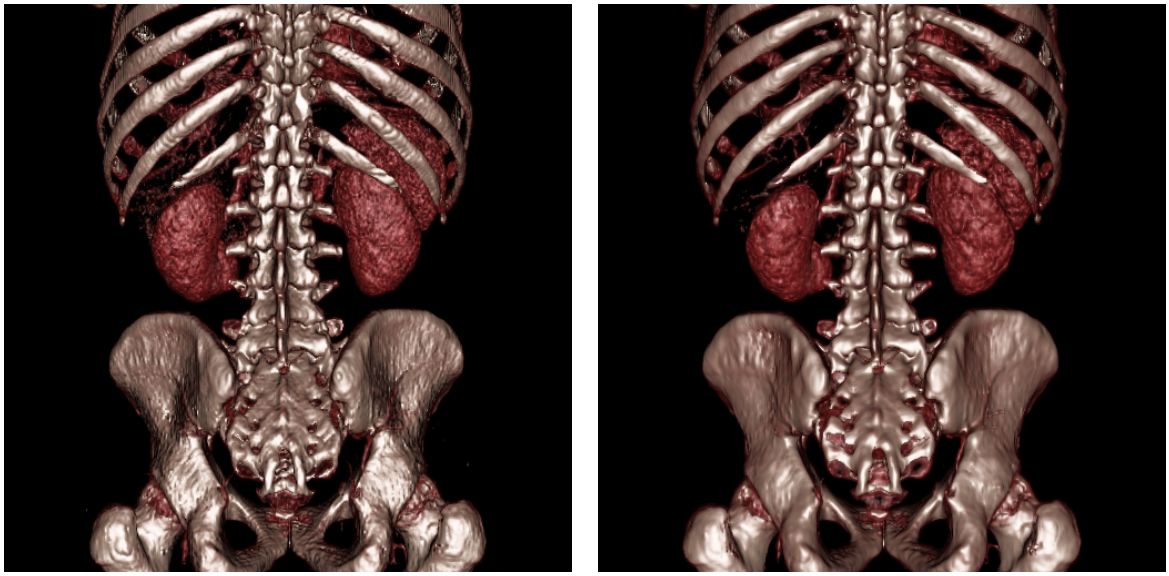
Having high resolution data sets, it is worthwhile to take a larger neighborhood into account for the linear regression calculation without significant loss of high frequency details. Figure 4.23 shows a human skull segmented from a CT scan of resolution $256 \times 256 \times 225$. The left image was rendered calculating the normals from the 3^3 neighborhood while in the right image the normals were estimated according to the 4^3 neighborhood. Note that the top of the skull in the right image is much smoother than in the left one and the staircase artifacts are less recognizable, while in high-frequency areas there is no significant difference.

Our method has been tested using also complex transfer functions. The images in Fig-



(a) (b)

Figure 4.23: Rendering of a human skull taking the 3^3 (a) and the 4^3 (b) neighborhood into account in the normal estimation using linear regression.



(a) (b)

Figure 4.24: Rendering of the kidneys and the skeleton using central differences (a) and linear regression (b) for gradient estimation.

ure 4.24 have been rendered with an opacity function emphasizing the soft tissue and the bone. Using linear regression for gradient estimation (b) rather than calculating the central differences (a) the rendered image seems to be more realistic because of the antialiasing.

4.4.6 Summary

In this section a new gradient estimation approach has been presented which is based on 4D linear regression. It has been shown that it is worthwhile to use the same function weighting the contribution of the neighboring voxels for filtering and for gradient computation yielding strong correlation between the filtered data values and the estimated normal vectors. Some previous normal computation techniques are special cases of our method thus it can be considered as a generalized solution with a clarified mathematical background. The presented technique can be used for gray-scale and binary data sets as well. Previous contextual shading methods are rather expensive computationally since they try to fit a linear or biquadratic function on the set of surface points and it requires the solution of a system of linear equations. In contrast, our approach approximates the density function itself with a 3D regression hyper-plane and it leads to a computationally efficient convolution.

4.5 Interactive volume rendering based on a "bubble model"

In this section an interactive volume rendering technique is presented which is based on a novel visualization model [11]. We call the basic method *bubble model* since iso-surfaces are rendered as thin semi-transparent membranes similarly to blown soap bubbles. The primary goal is to develop a fast previewing technique for volumetric data which does not require a time consuming transfer function specification to visualize internal structures. Our approach uses a very simple rendering model controlled by only two parameters. The rendering process is optimized exploiting that only a small part of the data contributes to the generated image. Due to the interactive display, fine tuning is also supported since the modification of the rendering parameters has an immediate visual feedback.

4.5.1 Introduction

Basically, there are two alternatives for high quality visualization of volume data sets. One alternative is iso-surface extraction using the *marching cubes* [38] surface-reconstruction method and the other one is direct volume rendering [16][34]. The first approach requires a time consuming preprocessing in order to generate a polygonal mesh. Although such a mesh can be rendered interactively using conventional 3D graphics hardware, this method is limited to certain iso-surfaces defined by modality-dependent threshold parameters. Without any a priori knowledge about the data distribution it is not obvious which iso-surfaces represent the content of the volume best without significant loss of information. Furthermore, whenever a threshold value is changed the entire reconstruction process has to be repeated.

Another alternative is direct volume rendering which is a more flexible approach. Theoretically, every single voxel contributes to the final image, therefore the internal structures can also be rendered. In practice, it is rather difficult and time-demanding to specify an appropriate transfer function. Because of the exponential attenuation, the objects which are hidden by several other semi-transparent objects are hardly recognizable. Even if low opacities are assigned to the voxels only a limited number of semi-transparent iso-surfaces can be rendered at the same time. There are techniques to determine optimal threshold parameters automatically [29], and methods for effective transfer function design also exist [20][22]. The transfer function specification, however, is still data-dependent, and often requires user interaction [30].

Physics-based direct volume rendering is also limited because of the computational cost. Without using any specialized hardware device, it is not possible to render a large volume data set interactively, although it would be rather important in transfer function specification to have an immediate feedback.

Application oriented visualization models like maximum intensity projection (MIP)[49], local maximum intensity projection (LMIP)[50], or frequency-domain volume rendering [57], [37] do not need time-consuming user interaction to specify the rendering parameters. Nevertheless, they also suffer from computational cost and they are limited to the medical imaging application field. Using MIP, some internal features can be hidden by higher density regions and using Fourier volume rendering, which is equivalent to density accumulation, similar problems arise.

The application of well-known non-photorealistic rendering (NPR) techniques [48][32][19] is a new direction in volume rendering research. Previously, these NPR methods have been proposed for polygonal surface models, therefore their application to iso-surfaces extracted from volume data seems to be obvious. Interrante [28] uses principal-direction driven 3D line integral convolution for illustrating surface shapes in volume data. Saito proposes an NPR technique for real-time previewing of volumes [47]. His approach is also restricted to an iso-surface, where

the surface is uniformly sampled and the sample points are projected onto the image plane as geometrical primitives like cross lines. The orientation of these primitives depends on the local inclination of the surface. Ebert [18] proposes a volume illustration framework combining direct volume rendering with NPR techniques. Most of the features of their general model are gradient and view-point dependent. Current volume rendering hardware devices do not support this model so interactive rendering is not possible.

In a certain sense, our method can also be considered as an NPR technique, since we do not concentrate on physically plausible rendering. Our major goal is to avoid a visual overload of the generated image and preferably, to render interactively all the important details.

4.5.2 The "bubble model"

Using a certain volume rendering application, especially in the medical imaging area, it is rather important to reduce the time of the user interaction which is necessary to tune the rendering process. For instance, a radiologist applying a 3D diagnostical system usually does not have too much time to find the most appropriate transfer function.

In order to avoid a time-consuming specification of rendering parameters and to develop a technique for fast previewing of volumetric data we use a simplified visualization model called *bubble model*. The main idea is to render several iso-surfaces as thin membranes similarly to the visual appearance of soap bubbles. We do not aim at a physically plausible model, therefore the spectral effects are neglected. The most important feature of the model is that such thin membranes do not hide too much information behind them. In traditional direct volume rendering [40], because of the exponential attenuation only a limited number of iso-surfaces can be rendered at the same time. Decreasing the opacity assigned to an iso-surface, the objects behind it become more visible but its own visual contribution is reduced as well.

Taking these aspects into account we propose the following simplified rendering model. The "surfaceness" of a voxel is characterized by the gradient magnitude at the voxel position. If the gradient magnitude is high then the voxel belongs to an iso-surface rather than a homogeneous region. The gradient vector is estimated by calculating *central differences*:

$$\nabla f(x_i, y_j, z_k) \approx \frac{1}{2} \cdot \begin{bmatrix} (f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)) \\ (f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ (f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \end{bmatrix}, \tag{4.19}$$

where $f(x, y, z)$ is the spatial density function. In order to avoid the staircase artifacts of this approximation a more sophisticated gradient estimation method [43] can also be used but it increases the preprocessing time as well. After having the gradient vectors calculated, opacities proportional to the gradient magnitudes are assigned to the voxels:

$$\alpha(i, j, k) = |\nabla f(x_i, y_j, z_k)| \cdot s, \tag{4.20}$$

where $\alpha(i, j, k)$ is the opacity of voxel $v(i, j, k)$ and s is a constant scaling factor. This idea is similar to Levoy's approach [34], who proposed a 2D opacity function weighted by the gradient magnitudes in order to enhance the iso-surfaces. In contrast, we use only a 1D opacity function depending on gradient magnitudes rather than density values. This simplification has a special visual effect and it will be exploited in the interactive rendering method discussed later. Unlike the conventional light transport equation [34] we do not assign own colors to the voxels. In this sense, the voxels do not have a light reflection contribution. We assume only a constant ambient background light, and each voxel with non-zero gradient magnitude attenuates

this background light. Thus, each pixel intensity is calculated as an accumulated transparency multiplied by the ambient light.

This so called bubble model can be considered as a simplified special case of the general optical model presented by Max [40]. We will show that the simplification has several advantages. Due to the opacity function weighted by the gradient magnitudes the number of voxels contributing to the image is reduced, therefore the visual overload can be avoided. Furthermore, the data reduction can be exploited in the optimization of the rendering procedure. Last but not least, the user interaction for tuning the rendering parameters is shorter and because of the fast display an immediate visual feedback is ensured.

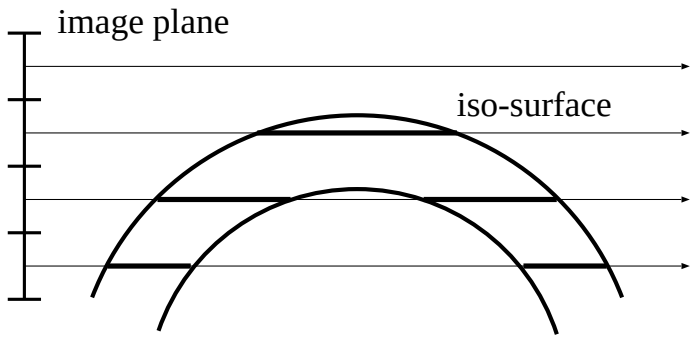


Figure 4.25: Opacity accumulation in the bubble model.

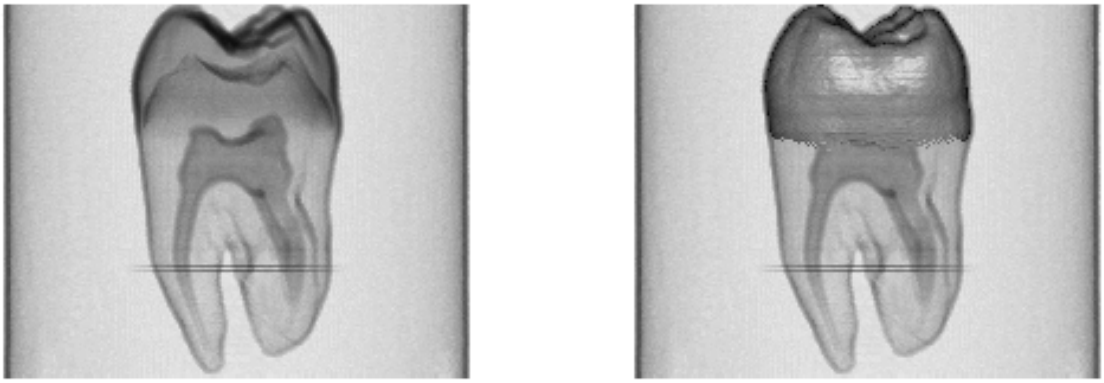


Figure 4.26: A CT scan of a tooth rendered using the bubble model (a) and the combined model (b).

The visual effect is illustrated in Figure 4.25. Those viewing rays which are nearly tangential to the iso-surface have a longer intersection segment with the region of high gradient magnitudes, therefore the corresponding accumulated transparency is lower. The rays nearly perpendicular to the iso-surface have minimal attenuation because of the short intersection segment, thus from these viewing angles the background is just slightly occluded. This model effects sharp silhouette lines at the object boundaries. This is similar to the visual appearance of a blown soap bubble, where almost just the silhouettes are visible. Nevertheless, the smooth transition at the boundaries characterizes the inclination of the surface improving the spatial impression. Figure 4.26a shows the CT scan of a tooth rendered using the bubble model. Note

that, you can see all the important details the nerves and the complete internal structure of the tooth.

The bubble model can be combined with traditional surface shaded display. In order to avoid the visual overload of the generated image we propose only one additional shaded iso-surface. The viewing rays are evaluated according to the following algorithm:

```
double RayCasting(Volume volume, Vec3D origin, Vec3D direction) {
  double transparency = 1.0;
  for(i = 0, i < i_max; i++) {
    Vec3D sample = origin + direction * i;
    Voxel voxel = volume.Resample(sample);
    if(voxel.density > threshold)
      return Shading(voxel) * transparency;
    else {
      double opacity = voxel.gradient_magnitude * scaling_factor;
      transparency *= 1.0 - opacity;
    }
  }
  return ambient_light * transparency;
}
```

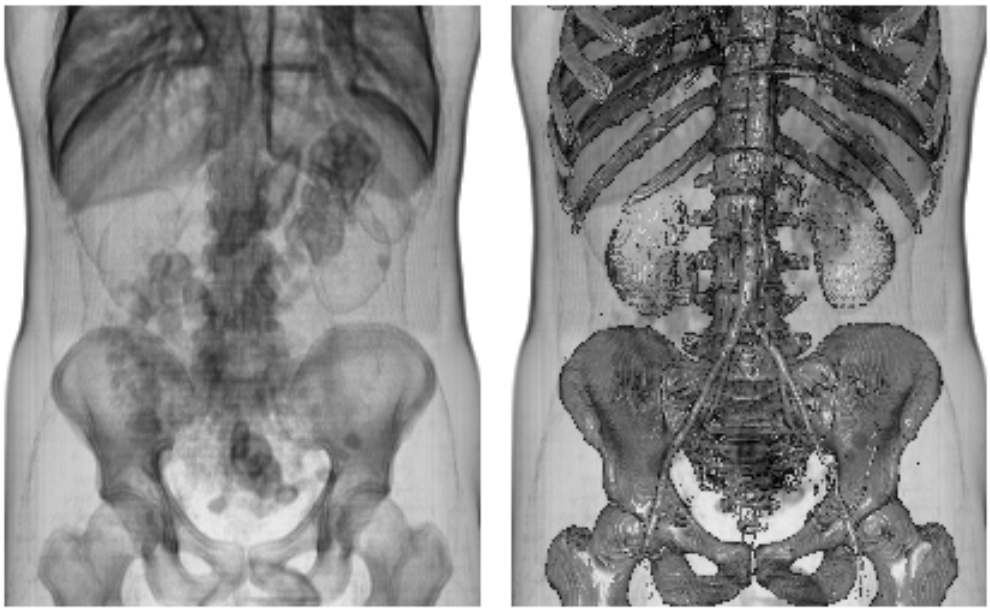
The upper formal ray-casting routine resamples the volume along a viewing ray defined by parameters *origin* and *direction.* In each *sample* point the *density* value and the *gradient magnitude* are calculated from the eight closest voxels using trilinear interpolation. If the *density* is greater than a predefined *threshold* then the lighting conditions are evaluated and the function returns the shaded color of the hit intersection point multiplied by the accumulated transparency. Otherwise the opacity of the current sample is multiplied by a *scaling factor* (denoted by s in Formula 4.20) and contributes to the accumulated transparency. If the ray does not have an intersection point with the iso-surface defined by the *threshold* then the function returns the *ambient light* multiplied by the accumulated transparency. This approach assumes that the internal structures have higher densities, like the bone in medical CT data sets. Generally, we can also use a confidence interval $[t-\epsilon, t+\epsilon]$ around threshold t to define the voxels belonging to an iso-surface.

The image in Figure 4.26b has been generated using the combined model. The upper part of the tooth has the highest density values, therefore setting an appropriate threshold it can be rendered separately using an arbitrary shading model. The root of the tooth has been visualized applying the bubble model.

Figure 4.27 shows a CT scan of a human body rendered using the bubble model (a) and the combined model (b). These images also contain almost all the internal details like the lungs, the ribs, the spine and the pelvis. In the right image it is illustrated that one additional shaded iso-surface is really a useful extension of the basic method since there can be organs with less drastic density transitions at the boundaries. For instance, the kidneys represent such a case, which can be easily rendered using surface-shaded display.

4.5.3 Interactive rendering

In this subsection we present an interactive rendering technique which supports our extended bubble model. This method is similar to the fast surface rendering algorithm presented in Section 4.3. The first step is a preprocessing, where at each voxel location a gradient vector is



(a) (b)

Figure 4.27: A CT scan of a human body rendered using the bubble model (a) and the combined model (b).

calculated. Afterwards, we extract the voxels having higher gradient magnitudes than a predefined threshold value. This results in a sparse volume which is stored in an appropriate data structure optimized for fast shear-warp projection [31].

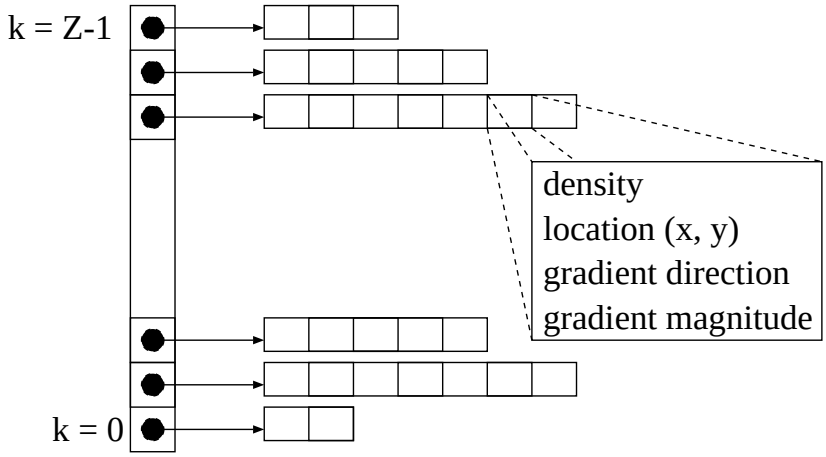


Figure 4.28: The data structure storing the sparse volume.

For the sake of clarity but without loss of generality, we assume that the principal component of the viewing direction is the z-coordinate and the resolution of the volume is $X \times Y \times Z$. In this case, the sparse volume is stored in a data structure illustrated in Figure 4.28.

The extracted voxels are stored sorted by their z-coordinates. The voxels having the same z-coordinate are stored in variable length arrays, where the length depends on the number of voxels extracted in the given z-slice. The entries contain all the information necessary for the rendering process, like the density, coordinates x and y (the z-coordinate is stored implicitly),

the gradient magnitude, and the gradient direction. The gradient direction is required for the view-dependent shading of an iso-surface. This is represented by twelve bits, where the upper and lower six bits store the two polar coordinates of the gradient direction. This is used as an address into a lookup table, which contains the precalculated shaded colors under certain lighting conditions. Whenever the viewing direction or the lighting conditions are modified the lookup table has to be refreshed.

The addresses of these arrays containing the voxels with the same z-coordinates are stored in a separate pointer array of size Z. The extracted voxels are mapped onto the image plane using an efficient shear-warp projection [31] (Figure 3.1).

The voxels are projected onto the intermediate image plane in back-to-front order. Initially, the pixel values of the intermediate image are set to the intensity of the background or ambient light. The current pixel values are multiplied by the transparency of the projected voxel. Whenever the density of the projected voxel is higher than the density threshold (defining a fully opaque iso-surface to be shaded), the current pixel value is overwritten by its shaded color. The shaded color is read from a precalculated lookup table, using the twelve-bit representation of the gradient direction as an address.

After having the intermediate image generated, the final image is produced by a 2D warp operation in the same way as it has been explained in Section 4.3.

4.5.4 Implementation

The previously presented interactive rendering technique has been implemented in C++ under Windows NT and has been tested on a 400MHz Pentium PC with 512M RAM. The interface of the application is shown in Figure 4.29. The two rendering parameters, the opacity scaling and the threshold defining a shaded iso-surface, can be controlled by two sliders. Because of the fast rendering procedure, an immediate visual feedback is ensured. The image can be rotated by moving the mouse pointer on the display window using the drag and drop convention. Table 4.10 shows the frame rates for different data sets.

volume	resolution	data reduction rate	frame rates
tooth	$256 \times 256 \times 161$	3.48 %	20.37 Hz
body	$202 \times 152 \times 255$	16.89 %	14.26 Hz
small head	$128 \times 128 \times 113$	25.19 %	9.32 Hz
big head	$256 \times 256 \times 225$	16.91 %	2.58 Hz

Table 4.10: Data reduction rates and frame rates for different data sets.

Figure 4.30 shows the front and side views of a human head rendered using the bubble model (a, c) and the combined model (b, d).

4.5.5 Summary

In this section a new interactive volume rendering technique has been presented. We proposed a simplified visualization model that we call *bubble model*, since the iso-surfaces are rendered as thin semi-transparent membranes. Our opacity function weighted by gradient magnitudes reduces the number of voxels which contribute to the final image. Such an opacity mapping has two advantages. On one hand the visual overload of the image can be avoided, without significant loss of information. On the other hand the data reduction can be exploited in the

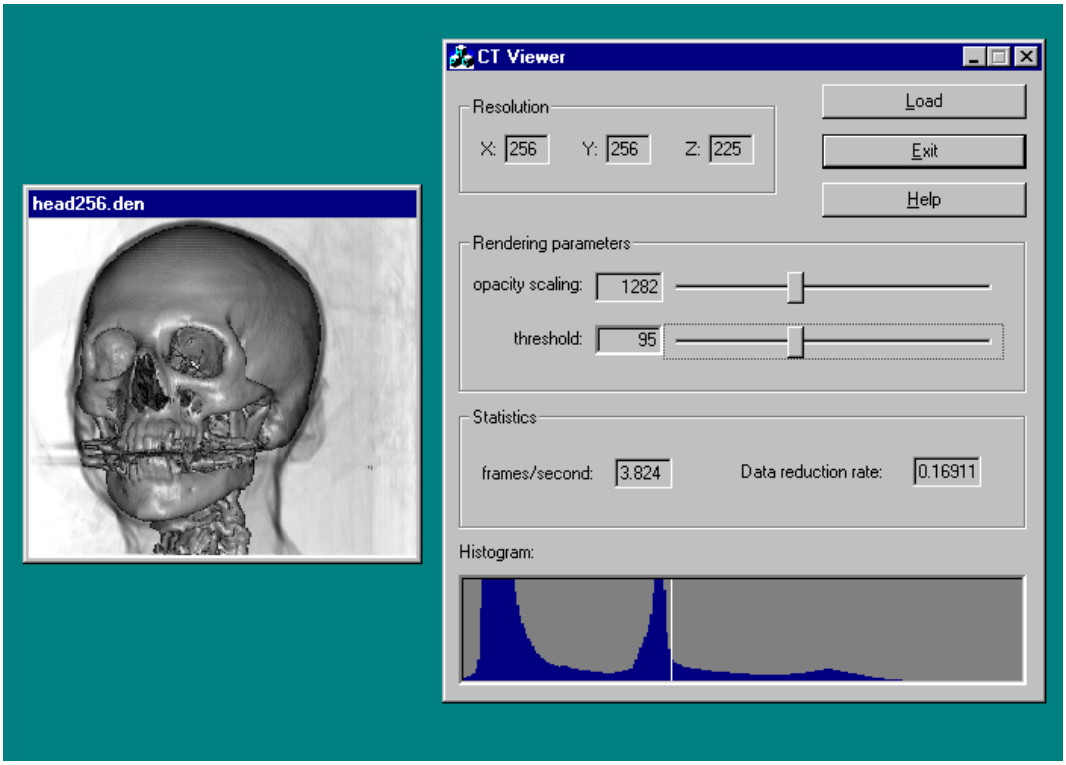


Figure 4.29: The graphics interface of the application.

optimization of the rendering process. We propose our model for fast volume previewing, which does not require a time-consuming transfer function specification. The rendering procedure is controlled by only two parameters and due to the optimization an immediate visual feedback is ensured. Since our acceleration technique is a pure software based method it does not rely on any specialized hardware to achieve interactive frame rates.

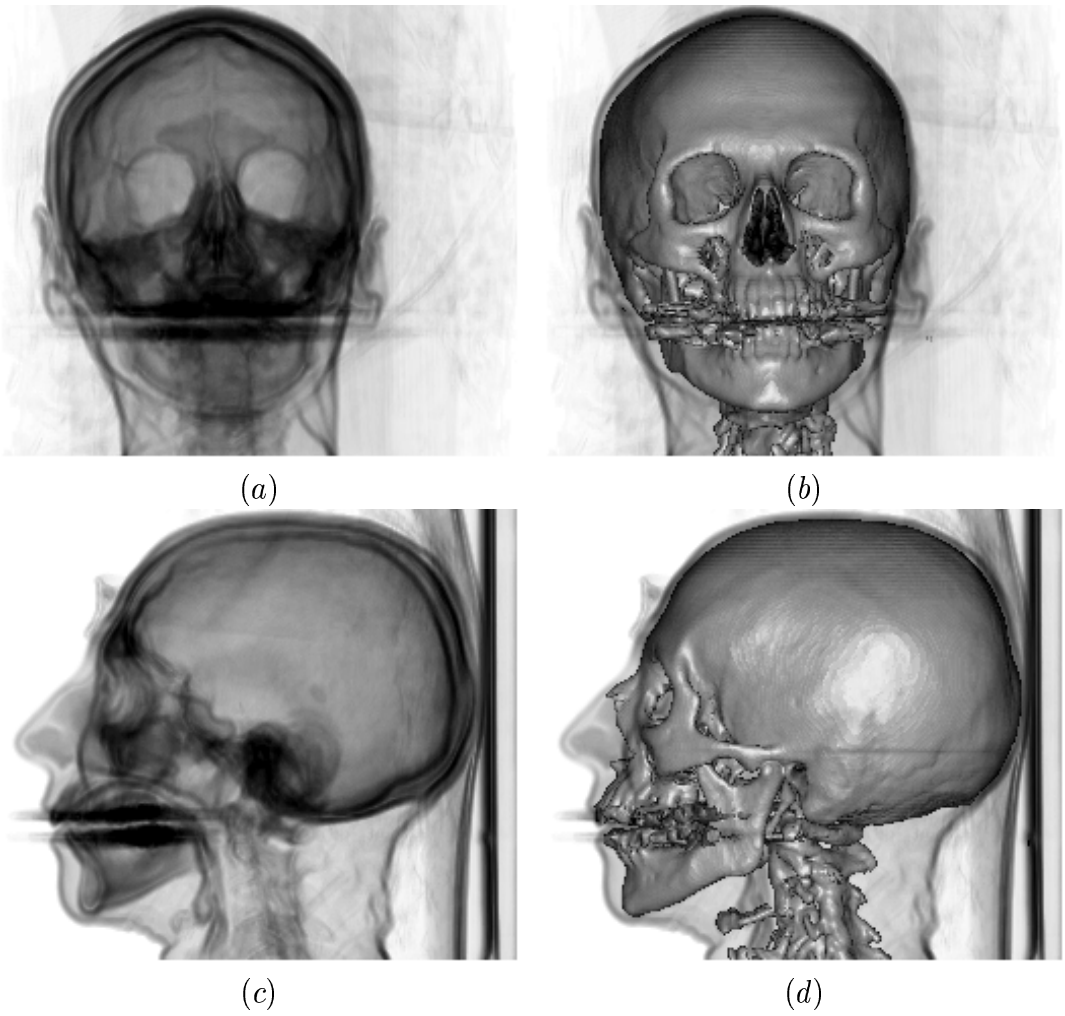


Figure 4.30: A CT scan of a human head rendered using the bubble model (a, c) and the combined model (b, d).

Chapter 5

Conclusion

In this thesis several fast volume-rendering techniques have been proposed mainly for interactive medical applications. It has been shown that it is not necessary to use any specialized hardware in order to achieve high frame rates even on low-end machines. This paper contributes the following new results:

1. In Section 4.1 a bit-parallel *binary shearing algorithm* has been presented. This is a good example for the interaction between the hardware-based and software-only research directions. Previously a similar method called *incremental alignment* has been proposed for reducing the communication overhead in a large multi-processor architecture supporting real-time volume rendering. It has been shown, that very simple operations like shifting of voxels in a binary segmentation mask can be performed efficiently in a parallel way using a conventional single-processor architecture. Exploiting the bitwise integer operations, the ALU can be used as a parallel machine processing several voxels at the same time. Using the binary shear operation together with an appropriate lookup table mechanism the empty segments along the viewing rays can be precisely skipped.

2. In Section 4.2 it has been shown that the binary shear transformation can be used not only for fast skipping of empty regions but for accelerated maximum intensity projection as well. Applying an efficient intensity encoding scheme together with the binary shearing algorithm, the rays can be encoded by a sequence of bytes. These bytes are used as addresses to lookup tables, storing the codes of the density intervals which contain the maximum density in the given ray segment. Therefore, it is easy to determine those low intensity ray segments, where the computationally expensive resampling does not have to be performed.

3. In Section 4.3 a fast direct surface-rendering technique is proposed. In order to reduce the number of voxels to be processed a recursive visibility calculation is performed. The domain of viewing directions is decomposed into different regions and for each region only those boundary voxels are extracted from the volume which are potentially visible. The extracted voxels are stored in view-dependent data structures optimized for fast shear-warp projection. The appropriate data structure is selected in the rendering process according to the current viewing direction. The extracted boundary voxels are projected onto the image plane in back-to-front to ensure hidden voxel removal. The presented technique also supports interactive cutting operations because of the direct volume-rendering approach.

4. The fast surface-rendering technique presented in Section 4.3 maps each voxel to one pixel because of efficiency reasons. Such a projection provides approximately the same

image quality as a first hit ray caster using nearest neighbor resampling. Because of the sparse resampling staircase artifacts can appear in the image. In order to compensate this drawback instead of the central differences a more sophisticated normal estimation scheme can be used. In Section 4.4 a new gradient estimation method has been presented which is based on *4D linear regression*. Since this method takes a larger voxel neighborhood into account to estimate the inclination of the surface a smooth approximated gradient function can be obtained.

5. In Section 4.5 a fully interactive volume previewing technique has been presented. It is based on a novel simplified visualization model called *bubble model*. The iso-surfaces are rendered as thin semi-transparent membranes similarly to blown soap bubbles. According to this model only those voxels contribute to the image which belong to an iso-surface. The "surfaceness" is measured by the gradient magnitude, therefore the surface voxels can be easily extracted from the original volume by a simple thresholding. This data reduction has several advantages. On one hand, the visual overload of the image can be avoided, and the occlusion of internal structures is also reduced. On the other hand, the data reduction can be exploited in the rendering process. Furthermore, because of the simplified visualization model a time-consuming transfer function design is not required since the rendering is controlled by only two parameters. These parameters can be interactively modified because of the optimized rendering procedure, thus an immediate visual feedback is ensured.

Bibliography

[1] E. Artzy, G. Frieder, and G. T. Herman. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. In *Proceedings Computer Graphics and Image Processing*, pages 1–24, January 1981.

[2] J. Bryant and C. Krumvieda. Display of 3D binary objects: I-shading. *Computers and Graphics, Vol.13, No.4*, pages 441–444, 1989.

[3] L. S. Chen, G. T. Herman, R. A. Reynolds, and J. K. Udupa. Surface shading in the cuberille environment. *IEEE Computer Graphics and Applications, Vol.5*, pages 33–43, 1985.

[4] J. Choi and Y. Shin. Efficient image-based rendering of volume data. In *Proceedings of Conference on Computer Graphics and Applications*, pages 70–78, 1998.

[5] M. A. Chupa. Marching cubes. 1998. http://www.erc.msstate.edu/˜chupa/f97/vis/lab1/.

[6] D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman. Real time discrete shading. *The Visual Computer, Vol.6, No.1*, pages 16–27, 1990.

[7] D. Cohen and Z. Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer, Vol.11, No.1*, pages 27–38, 1994.

[8] D. Cohen-Or and S. Fleishman. An incremental alignment algorithm for parallel volume rendering. In *Computer Graphics Forum (Proceedings EUROGRAPHICS '95)*, pages 123–133, 1995.

[9] B. Csébfalvi. An incremental algorithm for fast rotation of volumetric data. In *Proceedings of Spring Conference on Computer Graphics*, pages 168–174, 1998.

[10] B. Csébfalvi. Fast volume rotation using binary shear-warp factorization. In *Proceedings of Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 145–154, 1999.

[11] B. Csébfalvi and E. Gröller. Interactive volume rendering based on a "bubble model". In *Proceedings of Graphics Interface*, 2000.

[12] B. Csébfalvi, A. König, and E. Gröller. Fast maximum intensity projection using binary shear-warp factorization. In *Proceedings of Winter School of Computer Graphics*, pages 47–54, 1999.

[13] B. Csébfalvi, A. König, and E. Gröller. Fast surface rendering of volumetric data. In *Proceedings of Winter School of Computer Graphics*, pages 9–16, 2000.

[14] B. Csébfalvi and L. Szirmay-Kalos. Interactive volume rotation. *Journal Machine Graphics & Vision, Vol.7, No.4*, pages 793–806, 1998.

[15] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization '92*, pages 91–98, 1992.

[16] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Computer Graphics (Proceedings SIGGRAPH '88)*, pages 65–74, 1988.

[17] D. E. Dudgeon and R. M. Mersereau. *Multidimensional Digital Signal Processing.* Prentice-Hall, Inc., New Jersey, 1984.

[18] D. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume data. In *Proceedings of IEEE Visualization 2000,* pages 195–202, 2000.

[19] G. Elber. Interactive line art rendering of freeform surfaces. In *Computer Graphics Forum (Proceedings EUROGRAPHICS '99),* pages 1–12, 1999.

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

[21] J. L. Freund and K. Sloan. Accelerated volume rendering using homogenous region encoding. In *Proceedings of IEEE Visualization '97,* pages 191–196, 1997.

[22] I. Fujishiro, T. Azuma, and Y. Takeshima. Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In *Proceedings of IEEE Visualization '99,* pages 467–470, 1999.

[23] B. Gudmundsson and M. Randén. Incremental generation of projections of CT-volumes. In *Proceedings of the Conference on Visualization in Biomedical Computing,* 1990.

[24] G. T. Herman and H. K. Liu. Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing, Vol.9, No.1,* pages 1–121, 1979.

[25] G. T. Herman and J. K. Udupa. Display of three-dimensional discrete surfaces. In *Proceedings of the SPIE, Vol.283,* pages 90–97, 1981.

[26] K. H. Höhne and R. Bernstein. Shading 3D images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging, Vol.5, No.1,* pages 45–47, 1986.

[27] K. H. Höhne, M. Bomans, A. Pommert, M. Riemer, U. Tiede, and G. Wiebecke. Rendering tomographic volume data: Adequacy of methods for different modalities and organs. *3D Imaging in Medicine, Springer-Verlag,* pages 197–215, 1990.

[28] V. L. Interrrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Computer Graphics (Proceedings SIGGRAPH '97),* pages 109–116, 1997.

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

[30] A. König and E. Gröller. Mastering transfer function specification by using VolumePro technology. In *Proceedings of Spring Conference on Computer Graphics,* pages 279–286, 2001.

[31] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Computer Graphics (Proceedings SIGGRAPH '94),* pages 451–457, 1994. http://www-graphics.stanford.edu/papers/shear/.

[32] J. Lansdown and S. Schofield. Expressive rendering: A review of non-photorealistic techniques. *IEEE Computer Graphics and Applications, Vol.15, No.3,* pages 29–37, 1995.

[33] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Computer Graphics (Proceedings SIGGRAPH '91),* pages 285–288, 1991.

[34] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications, Vol.8, No.3,* pages 29–37, 1988.

[35] M. Levoy. Efficient ray tracing of volume data. *ACM Transations on Graphics, Vol.9, No.3,* pages 245–261, 1990.

[36] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer, Vol.6, No.1*, pages 2–7, 1990.

[37] L. Lippert and M. H. Gross. Fast wavelet based volume rendering by accumulation of transparent texture maps. In *Computer Graphics Forum (Proceedings EUROGRAPHICS '95)*, pages 431–443, 1995.

[38] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (Proceedings SIGGRAPH '87)*, pages 163–169, July 1987.

[39] T. Malzbender. Fourier volume rendering. *ACM Transactions on Graphics, Vol.12, No.3*, pages 233–250, 1993.

[40] N. Max. Optical models for direct volume rendering. *Journal IEEE Transactions on Visualization and Computer Graphics, Vol.1, No.2*, pages 99–108, 1995.

[41] N. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3D scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualization), Vol.24, No.5*, pages 27–33, 1990.

[42] T. Möller, R. Machiraju, K. Müller, and R. Yagel. A comparision of normal estimation schemes. In *Proceedings of IEEE Visualization '97*, pages 19–26, 1997.

[43] L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4D linear regression. In *Computer Graphics Forum (Proceedings EUROGRAPHICS 2000)*, pages 351–358, 2000.

[44] D. R. Ney, E. K. Fishman, D. Magid, and R. A. Drebin. Volumetric rendering of computed tomography data: Principles and techniques. *IEEE Computer Graphics and Applications, Vol.10, No.2*, pages 24–32, 1990.

[45] T. Porter and T. Duff. Compositing digital images. *Computer Graphics (Proceedings SIGGRAPH '84), Vol.18, No.3*, pages 253–259, 1984.

[46] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (Proceedings SIGGRAPH '88), Vol.22, No.4*, pages 51–58, 1988.

[47] T. Saito. Real-time previewing for volume visualization. In *Proceedings of Symposium on Volume Visualization '94*, pages 99–106, 1994.

[48] T. Saito and T. Takahashi. Comprehensible rendering of 3D shapes. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 197–206, 1990.

[49] G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection (MIP). In *EUROGRAPHICS Workshop on Rendering Techniques*, pages 51–63, 1995.

[50] Y. Sato, N. Shiraga, S. S. Nakajima, Tamura, and R. Kikinis. LMIP: Local maximum intensity projection. *Journal of Computer Assisted Tomography, Vol.22, No.6*, 1998.

[51] R. Shekhar, E. Fayad, R. Yagel, and F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings of IEEE Visualization '96*, pages 335–342, 1996.

[52] L. Sobierajski, D. Cohen, A. Kaufman, R. Yagel, and D. E. Acker. A fast display method for volumetric data. *The Visual Computer, Vol.10, No.2*, pages 116–124, 1993.

[53] K. R. Subramanian and D. S. Fussell. Applying space subdivision techniques to volume rendering. In *Proceedings of IEEE Visualization '90*, pages 150–159, 1990.

[54] L. Szirmay-Kalos. *Theory of Three Dimensional Computer Graphics.* Akadémia Kiadó, Budapest, 1995.

[55] Y. W. Tam and W. A. Davis. Display of 3D medical images. In *Proceedings Graphics Interface,* pages 78–86, 1988.

[56] G. Thürmer and C. A. Wüthrich. Normal computation for discrete surfaces in 3D space. In *Computer Graphics Forum (Proceedings of EUROGRAPHICS '97),* pages 15–26, 1997.

[57] T. Totsuka and M. Levoy. Frequency domain volume rendering. In *Computer Graphics (Proceedings SIGGRAPH '93),* pages 271–278, 1993. http://www-graphics.stanford.edu/papers/fvr/.

[58] S. S. Trivedi, G. T. Herman, and J. K. Udupa. Segmentation into three classes using gradients. In *Proceedings IEEE Transactions on Medical Imaging,* pages 116–119, June 1986.

[59] H. K. Tuy and L. T. Tuy. Direct 2D display of 3D objects. *IEEE Computer Graphics and Applications, Vol.4, No.10,* pages 29–33, 1984.

[60] J. K. Udupa. Interactive segmentation and boundary surface formation for 3D digital images. In *Proceedings Computer Graphics and Image Processing,* pages 213–235, March 1982.

[61] R. E. Webber. Ray tracing voxel data via biquadratic local surface interpolation. *The Visual Computer, Vol.6, No.1,* pages 8–15, 1990.

[62] R. E. Webber. Shading voxel data via local curved-surface interpolation. *Volume Visualization, (A. Kaufmann, ed.), IEEE Computer Society Press,* pages 229–239, 1991.

[63] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics (Proceedings SIGGRAPH '90),* pages 144–153, 1990.

[64] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *Computer Graphics, Vol.24, No.5,* 1990.

[65] R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications, Vol.12, No.5,* pages 19–28, 1992.

[66] R. Yagel, D. Cohen, and A. Kaufman. Normal estimation in 3D discrete space. *The Visual Computer, Vol.8, No.5,* pages 278–291, 1992.

[67] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *Proceedings of IEEE Visualization '93,* pages 62–69, 1993.

[68] K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Acceleration of ray casting using 3D distance transformation. In *Proceedings of the Conference on Visualization in Biomedical Computing,* pages 324–335, 1992.

Curriculum Vitae

CSÉBFALVI Balázs

November 16^{th}, 1972:	born in Budapest, Hungary
1979–1987:	elementary school in Pécs, Hungary
1987–1991:	Nagy Lajos High School in Pécs
1991–1996:	graduate studies at the Faculty of Electrical Engineering, Technical University of Budapest
June 25^{th}, 1996:	graduation as M.Sc. Engineer in Technical Informatics diploma thesis: “Design and Implementation of a Volume-Rendering Application” awarded by the Hungarian Ministry of Industry
1996–1998:	Ph.D. student at the Department of Control Engineering, and Information Technology, TU Budapest
August, 1998:	scholarship of the Austrian-Hungarian Action Fund at the Vienna University of Technology
1998–2001:	Ph.D. studies at the Institute of Computer Graphics and Algorithms, Vienna University of Technology