DISSERTATION

GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery

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,
Institut E186 für Computergraphik und Algorithmen,

und

Dipl.-Ing. Dr.techn. Markus Hadwiger,
VRVis Zentrum für Virtual Reality und Visualisierung,

und

Dipl.-Math. Dr.techn. Katja Bühler,
VRVis Zentrum für Virtual Reality und Visualisierung,

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

von

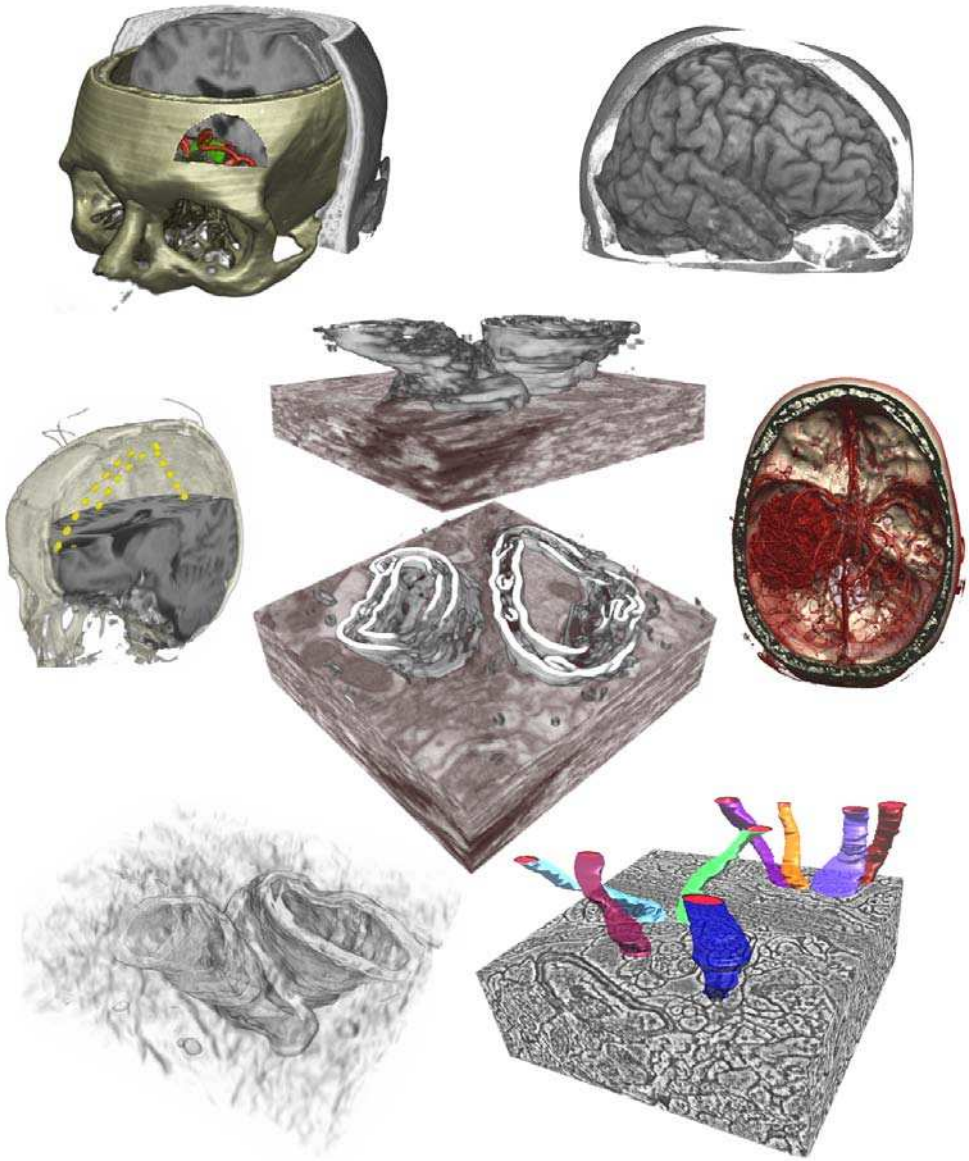
Dipl.-Ing. (FH) Johanna Beyer,

Matrikelnummer 0426197,
Gablenzgasse 99/24
1150 Wien

Wien, im Oktober 2009

DISSERTATION

GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery



Abstract

Recent advances in image acquisition technology and its availability in the medical and bio-medical fields have lead to an unprecedented amount of high-resolution imaging data. However, the inherent complexity of this data, caused by its tremendous size, complex structure or multi-modality poses several challenges for current visualization tools. Recent developments in graphics hardware architecture have increased the versatility and processing power of today's GPUs to the point where GPUs can be considered parallel scientific computing devices.

The work in this thesis builds on the current progress in image acquisition techniques and graphics hardware architecture to develop novel 3D visualization methods for the fields of neurosurgery and neuroscience.

The first part of this thesis presents an application and framework for planning of neurosurgical interventions. Concurrent GPU-based multi-volume rendering is used to visualize multiple radiological imaging modalities, delineating the patient's anatomy, neurological function, and metabolic processes. Additionally, novel interaction metaphors are introduced, allowing the surgeon to plan and simulate the surgial approach to the brain based on the individual patient anatomy.

The second part of this thesis focuses on GPU-based volume rendering techniques for large and complex EM data, as required in the field of neuroscience. A new mixed-resolution volume ray-casting approach is presented, which circumvents artifacts at block boundaries of different resolutions. *NeuroTrace* is introduced, an application for interactive segmentation and visualization of neural processes in EM data. EM data is extremely dense, heavily textured and exhibits a complex structure of interconnected nerve cells, making it difficult to achieve high-quality volume renderings. Therefore, this thesis presents a novel on-demand nonlinear noise removal and edge detection method which allows to enhance important structures (e.g., myelinated axons) while de-emphasizing less important regions of the data. In addition to the methods and concepts described above, this thesis tries to bridge the gap between state-of-the-art visualization research and the use of those visualization methods in actual medical and bio-medical applications.

Kurzfassung

Technische Fortschritte in bildgebenden Verfahren und deren weite Verfügbarkeit im medizinischen und bio-medizinischen Bereich haben zu einer noch nie dagewesenen Menge an hochaufgelösten Bilddaten geführt. Für die meisten Visualisierungs-Werkzeuge stellt jedoch die Komplexität dieser Daten, hervorgerufen durch ihre enorme Größe, Auflösung, komplexe Struktur oder Multi-Modalität große Herausforderungen dar. Die vorliegende Arbeit baut auf den aktuellen Entwicklungen im Bereich der bildgebenden Verfahren und der hohen Performanz und Flexibilität heutiger Grafikkarten auf, um neue Methoden zur 3D Visualisierung in der Neurochirurgie und Neurobiologie zu entwickeln.

Der erste Teil der Arbeit beschreibt eine Anwendung für die Planung von neurochirurgischen Eingriffen. Es werden GPU-basierte Multi-Volume-Rendering-Methoden entwickelt, um die Volumendaten mehrerer unterschiedlicher Bild-Modalitäten gleichzeitig darstellen zu können. Diese kombinierte Visualisierung erlaubt die genaue Darstellung der individuellen Anatomie, neurologischen Funktionen und Stoffwechselvorgänge der Patienten. Darüber hinaus werden neue Interaktions-Metaphern eingeführt, die es Chirurgen ermöglichen, den operativen Zugang zum Gehirn patientenindividuell zu planen und zu simulieren.

Der zweite Teil der Arbeit konzentriert sich auf GPU-basierte Volume-Rendering-Techniken für die Darstellung großer und komplexer EM Daten, wie sie im Bereich der Neurowissenschaften vorhanden sind. Ein neuer auf unterschiedliche Auflösungsstufen beruhender Volume-Ray-Casting-Ansatz ermöglicht die artefaktfreie Darstellung von Blöcken mit unterschiedlichen Auflösungsstufen. Schließlich wird *NeuroTrace* vorgestellt, ein Programm zur interaktiven Segmentierung und Visualisierung neuronaler Prozesse in EM-Daten. Um die Qualität von abgebildeten EM-Daten zu verbessern, wird eine neue bedarfsorientierte und nichtlineare Rauschunterdrückung und Kantendetektierungsmethode beschrieben, die wichtige Strukturen in EM-Daten (z.B. myelinisierte Axone) hervorhebt, während weniger wichtige Regionen des Volumens ausgeblendet werden. Zusätzlich zu den oben beschriebenen Methoden und Konzepten, versucht diese Dissertation eine Brücke zwischen dem momentanen Stand der Forschung im Bereich der Volumenvisualisierung und deren Integration in tatsächliche medizinische und biomedizinische Anwendungen zu schlagen.

Contents

Related Publications

This thesis is based on the following publications:

J. Beyer, M. Hadwiger, S. Wolfsberger, C. Rezk-Salama, and K. Bühler. **Segmentierungsfreie Visualisierung des Gehirns für Direktes Volume Rendering**. In *Proceedings of Bildverarbeitung für die Medizin 2007*, pages 333–337, 2007.

J. Beyer, M. Hadwiger, S. Wolfsberger, and K. Bühler. **High-Quality Multimodal Volume Rendering for Preoperative Planning of Neurosurgical Interventions**. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007)*, 13(6), pages 1696–1703, 2007.

J. Beyer, M. Hadwiger, T. Möller, and L. Fritz. **Smooth Mixed-Resolution GPU Volume Rendering.** In *Proceedings of IEEE International Symposium on Volume and Point-Based Graphics (VG 2008)*, pages 163–170, 2008.

W.-K. Jeong, J. Beyer, M. Hadwiger, A. Vasquez, H. Pfister, and R. Whitaker. **Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets**. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2009)*, 15(6), pages 1505–1514, 2009.

Other publications:

J. Beyer, C. Langer, L. Fritz, M. Hadwiger, S. Wolfsberger, and K. Bühler. **Interactive Diffusion Based Smoothing and Segmentation of Volumetric Datasets on Graphics Hardware**. *Methods of Information in Medicine*, pages 270–274, 2007.

Chapter 1

Introduction

Recent advances in image acquisition technology and its availability in the medical and bio-medical fields provide an unprecedented amount of high-resolution imaging data. Difficulties in handling this data often arise due to its complexity, enormous size, multi-dimensionality and the need of fusing several separate datasets into a single consistent visualization. Therefore, interactive 3D visualization of these datasets is still an active area of research, facilitated by the improvements made in current graphics hardware. GPU-based volume rendering has made great advances thanks to the ever increasing flexibility in graphics card programmability. With the advent of high-level shading languages such as Cg [53] or GLSL [72] and, more recently, with the introduction of CUDA [60], GPUs can now be seen as versatile computing devices.

This thesis focuses on the development of 3D visualization tools for the fields of neurosurgery and neuroscience, ranging from the macro- and microscopic level down to the nanoscale level. The first part of the thesis deals with concurrent GPU volume rendering of multiple medical datasets for planning of neurosurgeries. For use in clinical practice the combination of these different medical datasets such as CT, MRI, DSA or PET scans into a single visualization needs to be driven by a user-centered point of view, which closely resembles the surgeon's workflow. The second part focuses on GPU volume rendering of large and complex EM data, as required in the field of neuroscience. When dealing with this huge amount of data in the petascale range it does not suffice to simply adapt the current algorithms to be able to cope with the data size. Instead, it requires completely new ways of handling and processing this data, as well as new interaction methods.

1.1 Contribution

The main contributions of this work are as follows:

- An application for planning of neurosurgical interventions integrated into the clinical workflow:
  - Unified handling of GPU-based multi-volume ray-casting and bricking with and without segmentation masks (Chapter 4, Section 4.5). For each sample location, the volume to be sampled is either chosen depending on segmentation information, or multiple volume samples are blended. We circumvent GPU memory constraints by bricking each volume (CT, MR, DSA, PET, fMR), and downloading only active bricks into 3D cache textures (one per modality or unified). Segmentation information is represented as a bricked object ID volume over all modalities, which likewise employs a 3D cache texture.
  - *Skull peeling* (Chapter 3, Section 3.3) for selectively removing structures obscuring the brain (e.g., skin, bone) without segmentation. In contrast to opacity peeling [68], we consider registered CT and MR data at the same time for more dependable results. The impact of clipping is resolved consistently. Areas of the patient's skin and bone can be removed selectively by painting 2D clipping areas.
  - Smooth rendering of segmented object boundaries, taking into account the contributions of multiple volumes (Chapter 4, Section 4.5.5). We propose an approach that is customized for the needs of our neurosurgery pipeline that achieves better results in this case. During ray-casting, the precise transition between two adjacent materials is re-classified depending on user-specified iso-values and searching the object ID and data volumes along the gradient direction.
- An application for interactive visualization (and segmentation) of large neurobiological EM datasets, for the reconstruction of neural connections in the brain:
  - A GPU-based bricked mixed-resolution volume rendering scheme that is not restricted to downsampling at the original grid positions and does not require modifying the original sample values. In this approach we employ single-pass ray-casting to mix different levels of resolution with continuous (C^0) transitions between resolution levels.
  - An application for scalable and interactive visualization of neural processes in EM datasets. We propose a CUDA-based ray-caster which offers on-demand filtering for de-noising and edge-enhancement of structure boundaries. A local histogram-based edge metric provides better visual cues to easily find regions of interest in complex EM datasets

compared to traditional transfer functions. This functionality is integrated into *NeuroTrace*, an application that offers interactive segmentation and visualization of EM data for neuroscience.

1.2 Organization

This thesis is organized into two main parts: The first part (Chapters 2, 3, and 4) focuses on an application for neurosurgical planning, including necessary pre-processing steps, interaction metaphors and visualization methods. Chapter 2 starts with an introduction to medical imaging in neurosurgery as well as fundamentals of medical visualization and surgical planning applications. Skull Peeling, an algorithm for the fast visualization of the brain's surface without any prior segmentation, is introduced in Chapter 3. Finally, in Chapter 4, an application for high-quality multimodal volume rendering for preoperative planning of neurosurgical interventions is presented.

The second part of this thesis (Chapters 5, 6, and 7) focuses on volume rendering of large, complex, biological data. Chapter 5 gives an introduction to the fields of neuroscience and connectomics as well as large data and multi-resolution rendering. The main technical contributions of this part are a method for smooth mixed-resolution volume rendering, presented in Chapter 6, and a framework for interactive segmentation and visualization of neural processes in EM datasets, presented in Chapter 7. This thesis concludes with a summary of the presented contributions and draws conclusions in Chapter 8.

Multi-Volume Rendering for Neurosurgery

Chapter 2

Introduction

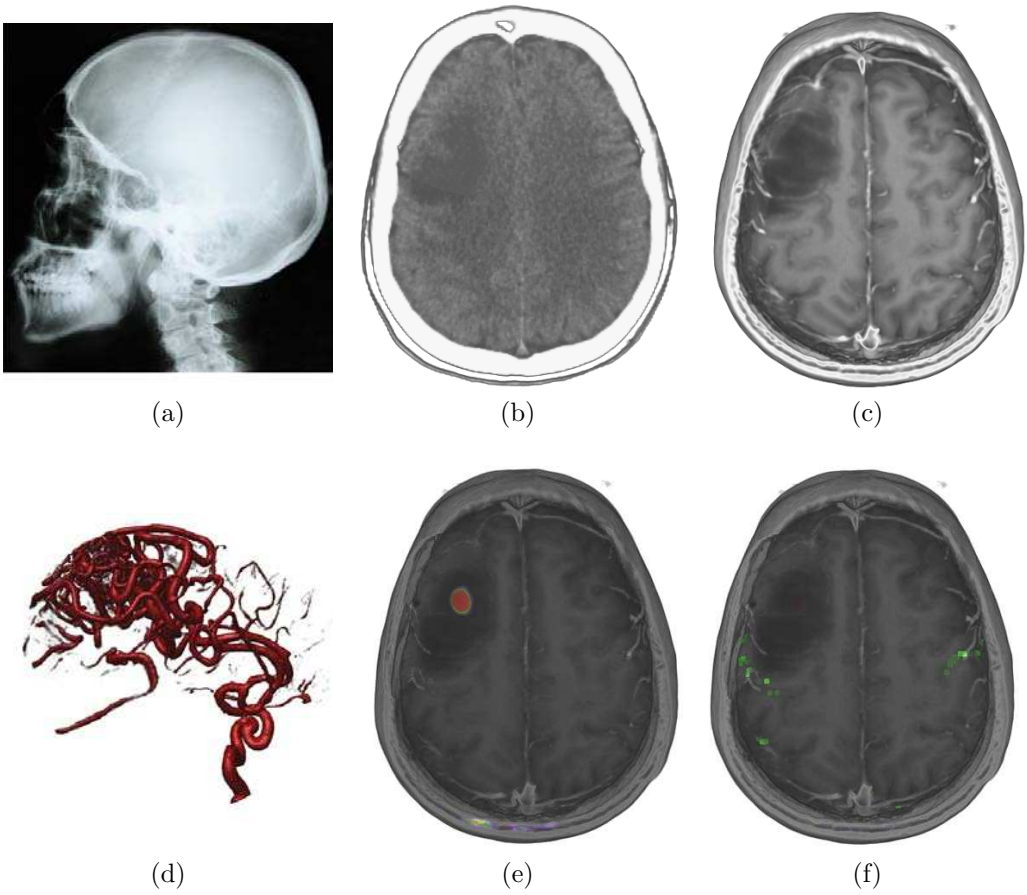
In today's clinical practice, computer aided diagnosis and surgery planning applications are becoming increasingly important. These specialized applications, which rely on high-resolution 3D images (e.g., CT or MRI scans), often require integrated volume de-noising, filtering, segmentation, and visualization possibilities. However, the combination of all these features into a single application is quite challenging. Often a compromise has to be found between quality, reliability, usability, and the amount of time the user needs.

With the advent of modern GPUs and their high computing power many algorithms can now achieve interactivity. Complex filtering and segmentation algorithms can be run in real-time, permitting the user to stop or modify the ongoing computation. Volume rendering, even of large or multiple datasets, can be performed entirely on the GPU, at interactive rates. With the newest generation of graphics hardware and the development of APIs such as CUDA [60, 31] or OpenCL [26], the flexibility for scientific programming on the GPU has achieved new heights. Medical and bio-medical applications can build on these new developments to perform compute-expensive calculations while still achieving interactive performance.

This chapter starts with Section 2.1 reviewing different medical imaging techniques. Section 2.2 presents fundamentals of image processing, registration and segmentation of medical data. Finally, the basics of medical visualization and visualization systems for neurosurgery are given in Section 2.3. For a more detailed introduction to medical visualization and image processing the reader is referred to the textbook *Visualization in Medicine* by Preim and Bartz [63].

2.1 Medical Image Data

In recent years, imaging scanners have been able to acquire images of ever increasing quality, resolution and accuracy. Especially in the field of neurosurgery it is common to use more than one modality for the exact delineation of the patient's anatomy and pathology. Each modality has different advantages and shortcomings. Figure 2.1 shows the most common scanning techniques in the field of neurosurgery which are described in the following:



(a) (b) (c)

(d) (e) (f)

Figure 2.1: Examples of the most common imaging modalities in the field of neurosurgery. (a) X-Ray image of a human head. (b) Slice of a CT image. (c) MR image. (d) Volume Rendered DSA dataset. (e) PET superimposed on an MR image. (f) fMR superimposed on an MR image.

2.1.1 X-Rays

X-Rays, the most common form of diagnostic imaging, were first discovered by Wilhelm Conrad Röntgen in 1895 and is based on measuring the attenuation of electromagnetic radiation traveling through a scanned object. X-Rays traveling through an object are absorbed and scattered depending on the object's density, with dense objects having a higher absorption rate than less dense objects. Therefore, dense objects, such as bones, get depicted as bright areas whereas less dense objects, such as air, are depicted as a dark areas.

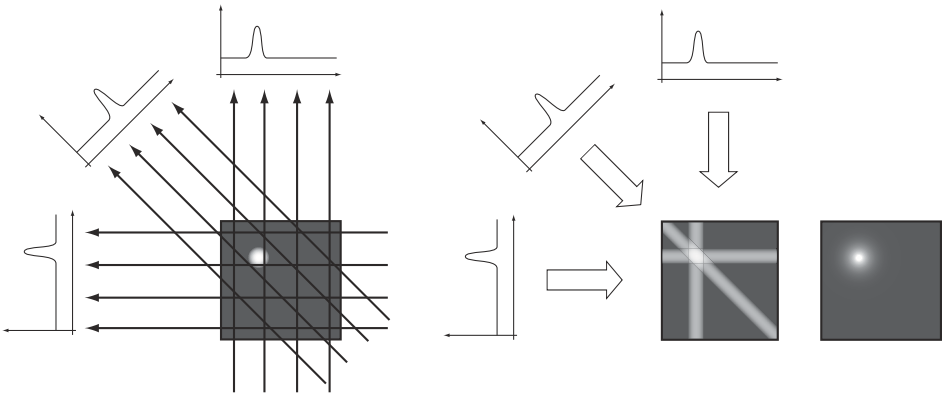
2.1.2 Computed Tomography (CT)

Computed Tomography works by taking a series of individual X-Rays from different viewpoints around the scanned object and back-projecting them into a 3D volume. An X-Ray emitter/detector pair rotates on a circular path around the scanned object and creates X-Ray projections. The projections from a full rotation are projected back, based on the Radon transform, to calculate one cross-section of the final volume. Then the emitter/detector pair proceeds to scan the next slice. Figure 2.2a depicts the CT acquisistion process and Figure 2.2b explains the CT reconstruction step in more detail. The drawback of CT imaging is its high radiation dose. However, newer spiral CT scanners allow to not only scan individual cross-sections but to continuously change the cross sections by moving the scanned object slowly through the X-Ray circle. After acquisition, the computed values are normalized into Hounsfield units, where water is represented with 0 and air is represented with -1000. For enhancing the visibility of vessels or certain tissue, contrast agents can be introduced into the scanned object to highlight these areas. In neurosurgery, CT is the medium of choice for depicting the skull and bony structures and required for intra-operative navigation.

2.1.3 Magnetic Resonance Imaging (MRI)

Magnetic resonance imaging is based on the fact that human tissues have different properties in a magnetic field. MRI works by placing the scanned subject in a strong magnetic field. Hydrogen nuclei in the scanned subject align themselves parallel or anti-parallel to this magnetic field. Next, a radio-frequency pulse is induced which causes the hydrogen nuclei to spin synchronously (i.e., in phase). After stopping the radio-frequency pulse the protons slowly de-phase. Measuring the time of this relaxation allows to measure the proton density, which is depicted in the MRI image.

In neuro-imaging MRI is usually used for depicting soft tissue such as the brain, whereas it is not well suited to display tissue with little water in it, such as bone. One main advantage of MRI compared to CT imaging is that there is no radiation used in MRI imaging which could be harmful for the patient.



(a) (b)

Figure 2.2: (a) During CT acquisition several X-Ray images from different directions around the scanned object are taken. (b) For CT reconstruction, the original views are backprojected to create the final reconstructed image.

2.1.4 Functional Magnetic Resonance Imaging (fMRI)

Functional MRI is used to depict the neural activation of certain brain areas. It detects changes in blood flow and oxygen metabolism. Therefore, fMRI is acquired while the patients are performing cognitive or behavioral tasks (e.g., talking) in the MRI scanner. The activation areas found by the fMRI are usually superimposed on an anatomic MRI image, to get a notion of the spatial orientation and location of the "active" brain areas.

2.1.5 Digital Subtraction Angiography (DSA)

Digital subtraction angiography is an example for contrast enhanced imaging. One angiographic image is taken before inducing contrast agent and one image is taken after inducing contrast agent. The image without contrast agent is then subtracted from the image with the contrast agent, resulting in an image highlighting only the area where the contrast agent was present. DSA is primarily used for imaging blood vessels and blood vessel trees and therefore very popular in cerebral imaging.

2.1.6 Positron Emission Tomography (PET)

Positron emission tomography is a nuclear medicine imaging technique that depicts the metabolic activity of tissue. It works by injecting a short-lived radioactive substance into the patient. When this substance starts to decay it emits positrons. Subsequently, when a positron interacts with an electron two gamma

photons moving in opposite directions are generated. These photons can be measured and used to localize their source location. PET has a very low spatial resolution but is very useful in depicting areas of high metabolic activity, such as certain tumors.

2.1.7 Imaging Artifacts

Medical imaging data aquired by the scanning techniques described above, is always subject to different kinds of imaging artifacts. Knowledge of the different kinds of artifacts and their occurrence is therefore vital for correct image understanding and analysis.

In the following, the most important types of imaging artifacts are listed and explained:

Undersampling occurs when a signal is sampled below the Nyquist rate. The *Nyquist rate* is defined to be more than twice the frequency of the original signal, and is necessary for a correct reconstruction of the signal. If the sampling rate is below this rate, the original signal might be mistaken for a different signal, and aliasing effects and information loss occur. Figure 2.3 shows a signal that is sampled below its Nyquist rate and the resulting erroneous signal.

Another typical artifact in medical data is the *partial volume effect*. This effect is caused by the limited resolution of the image data reconstruction and happens when two or more tissue types are present within a single voxel. The partial volume effect leads to blurring of boundaries between high and low intensities and always needs to be considered when dealing with medical imaging data.

Signal artifcats, such as metal streak artifacts in CT data, arise from the distinct properties of individual imaging modalities. For example, in CT imaging, a piece of metal might cause the attenuation measurement on a detector to be incorrect, resulting in several visible streaks in the image.

Image inhomogeneities are commonly found in MRI data, and are caused by inhomogeneities of the magnetic field produced by the scanner. These inhomogeneities cause distortions in both geometry and intensity of the MR images. Many image processing algorithms try to reduce the effect of these different kinds

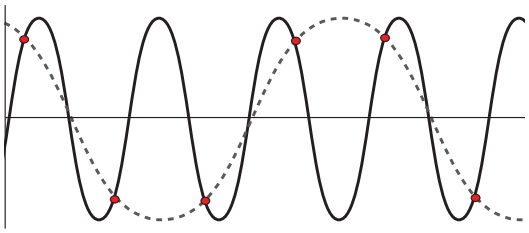


Figure 2.3: Undersampling of the original signal (black curve) at the red sampling positions leads to the erroneous reconstructed signal (dashed curve).

of imaging artifacts. However, careful parameter setting during the actual image acquisition is the most dominant factor for image quality.

2.2 Image Processing for Medical Data

Most medical visualization systems rely on some kind of preprocessing of the raw volume data. The aim is to extract clinically relevant information from the radiological image data, which helps in diagnosis or treatment planning for the patient. Usually there is an entire pipeline of different pre-processing steps, consisting of image filters for de-noising and smoothing, segmentation of important structures and registration or resampling of datasets prior to visualization.

2.2.1 Filtering and De-noising

The main objective of image smoothing in medical applications is the enhancement of image quality as a preprocessing step prior to segmentation or visualization. The aim is to reduce noise while preserving important features to improve the visualization and segmentation result.

Where some preprocessing steps only try to visually enhance the image, other methods improve the signal-to-noise ratio in the scanned images. *Histogram equalization*, for example, tries to enhance the contrast in the image by equalizing the histogram (i.e., transforming the histogram so that all histogram bins have nearly the same value). This enhances the contrast of the image visually, but does not improve the actual signal-to-noise ratio.

The amount and type of noise present in medical images has several different causes: it greatly depends on the imaging modality, its spatial resolution, slice thickness and patient movement, to name a few. Noise is considered to be in the high frequency spectrum of the image. Therefore, noise reduction techniques often try to low-pass filter the dataset to reduce the noise present.

Image filtering is a neighborhood operation in which the image values of a specified neighborhood are used to compute the output filter value of one pixel/voxel. *Linear filtering* can be thought of as a convolution of the original image with a filter kernel. A very simple example for a linear filter is averaging. Another example is *Gaussian filtering*, where the original image is convolved with a Gaussian function as shown in Equation 2.1 for the 2D case:

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

Since the Gaussian kernel is continuous, for filtering a discrete 2D image the values of the Gauss kernel are usually pre-calculated and stored in a discrete 2D filter mask of a given size (e.g., 3×3, 5×5).

A *non-linear filter*, on the other hand, is a filter that generates an output that is not a linear function of the input. The most common examples are median, bilateral or anisotropic diffusion filters. Non-linear filters are usually more expensive to compute than linear filters but achieve better filtering results.

Diffusion filtering is an edge-preserving filtering technique where the physical process of diffusion is simulated. The common notion of diffusion is a physical process that equilibrates concentration/energy differences without creating or destroying mass/energy. For example, transporting molecules from a region of higher concentration to one of lower concentration, finally resulting in a complete mixing or a state of equilibrium. For image filtering this idea is used to smooth intensity differences in adjacent pixels. Non-linear and anisotropic diffusion scale the amount of diffusion depending on the gradient magnitude of a pixel to reduce diffusion around edges. A comparison of different diffusion filtering techniques and their application to medical data is given by Suri et al. [80].

2.2.2 Segmentation

Segmentation is the process of partitioning image data into distinct regions by assigning labels to structures of interest. In medical imaging, segmented objects are usually clinically relevant structures such as organs, bone, vessels or tumors.

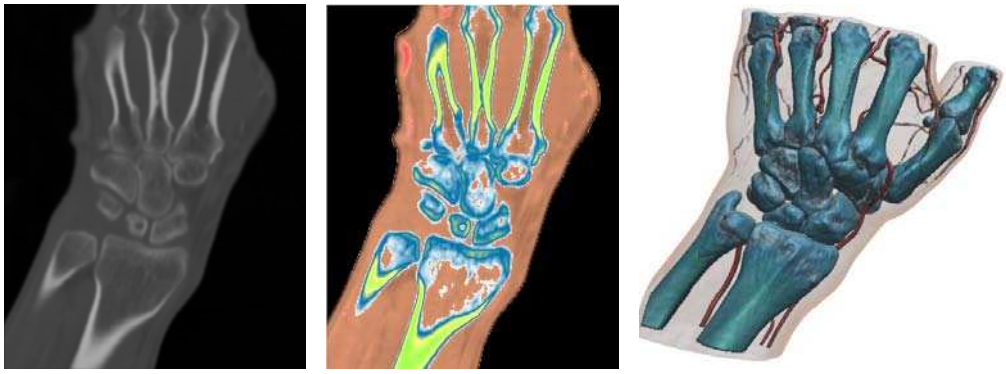
Segmentation methods can be divided into manual, semi-automatic and automatic approaches. Even though manual approaches are very time consuming, they are still common in clinical practice. Automatic approaches, on the other hand, are often very limited in their scope of application because of their highly specialized algorithms, and they might not allow interactive adjustments of global parameters.

There are several popular strategies for image segmentation ranging from edge- and region-based approaches to model-based or multi-scale approaches. An in-depth discussion of different segmentation techniques is out of scope of this thesis but the reader is referred to [62, 81].

In the context of volume rendering the result of the segmentation process is usually stored as a segmentation mask which defines the object membership of each voxel (see Figure 2.4). However segmentation results can also be stored as surface representations or as distance fields.

2.2.3 Registration and Resampling

Registration is the process of transforming separate images into the same coordinate system, so that structures in one image can be related to the corresponding structures in the registered image. The different images that need to be registered can either come from the same imaging modality but from different acquisition times, from different imaging modalities, or from an anatomical atlas. Medical image registration can either be rigid or non-rigid. *Rigid registration* allows only



(a) (b) (c)

Figure 2.4: (a) Slice view of an unsegmented hand CT dataset. (b) Slice view showing the segmented dataset. (c) Volume rendering of the segmented dataset.

for translation, rotation and scaling whereas *non-rigid deformation* allows for a broader range of modifications and deformations. For the neurosurgical planning application described in this thesis we use an external registration algorithm based on mutual information [14] and perform rigid registration (i.e., only translation and rotation).

Resampling is often a part of registration and is the process of changing the grid/voxel structure of the input data into another grid/voxel structure. In the context of medical imaging this usually means either resampling from a non-regular grid into a regular grid (e.g., resampling in 3D ultrasound) or resampling from one regular grid into another regular grid to change resolution or orientation of the grid (e.g., for registration). A major concern in image resampling is degradation of image quality. Several different methods have been developed, ranging from very fast trilinear interpolation to more advanced triquadratic or tricubic interpolation [83].

2.3 Visualization of Medical Data

The objective of medical visualization is to support doctors and medical personnel in their decision finding process. Medical visualization is a subarea of scientific visualization which tries to create images and renderings that aid the user in understanding complex or high-dimensional data. The main purpose of visualization is to gain insight into the data [56]. The user should be able to explore the dataset, declare and test hypotheses and present and illustrate the found results.

In the medical field the applications of visualization are mainly education,

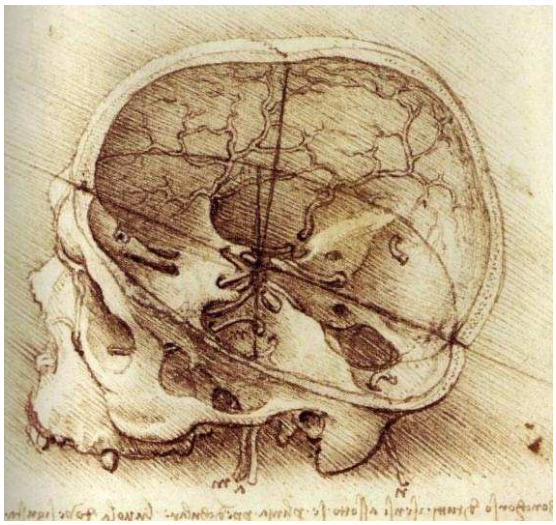


Figure 2.5: Medical illustration of Leonardo da Vinci. “View of a skull”, 1489.

diagnosis, treatment planning and intraoperative support. First medical illustrations depicting anatomical systems and pathologies date back to the 16th century and were based on sketches made during or after surgery or dissections (see Figure 2.5). Depending on the final application a medical visualization system needs to support different features such as volume illustration and labeling, support for measuring, visualization of segmentation results, surgery planning tools and the integration into intraoperative navigation systems. An overview of different medical visualization algorithms and applications can be found in [63, 37, 38].

For use in clinical practice most visualization systems are embedded into medical workstations that support the user in patient management and diagnosis and often provide additional filtering, segmentation and visualization capabilities. Figure 2.6 shows the Agfa IMPAX EE system and the integrated volume rendering and multiplanar reconstruction views presented in this thesis.

In *multiplanar reconstruction* (MPR), oblique cutting planes are positioned in the volume and displayed as slices. The standard MPR usually displays slices orthogonal to the x, y, and z axis of the dataset.

A main criterion for the acceptance of medical visualization applications in clinical use is their usability, including their support for more advanced features such as measuring tools, volume labeling, and clipping, as well as their interactivity. Therefore, all algorithms proposed in this thesis are implemented in a GPU-based framework and show interactive performance.

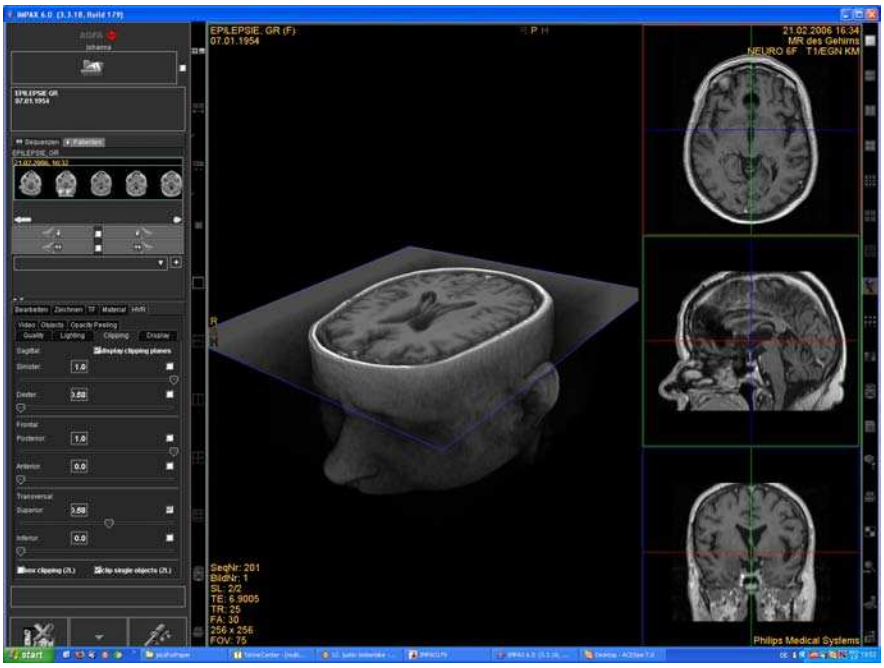


Figure 2.6: Medical workstation IMPAX EE by Agfa Healthcare.

2.3.1 Volume Rendering

Direct Volume Rendering (DVR) [18] creates images of volumetric datasets without the need to explicitly extract geometry or surfaces. It is based on an optical model that defines how a volume emits, reflects, scatters and occludes light. *Ray-casting techniques* [44] shoot viewing rays through the volume and accumulate color and opacity values while traversing the volume, as depicted in Figure 2.7. The accumulation of color and opacity is computed by evaluating the *volume rendering integral* that integrates the emission and absorption of light along the direction of light traversal [18](see Figure 2.8):

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)dt} ds \tag{2.2}$$

I_0 is the initial intensity at point s_0, where the light is entering the volume from the background. $I(D)$ is the intensity when the ray is leaving the volume at point D. The first term in Equation 2.2 represents the light from the background that is absorbed along the ray through the volume. The second term in Equation 2.2 defines the active emission and absorption of the participating medium (i.e., volume) along the remaining distance.

The volume rendering integral is usually implemented by its approximation by

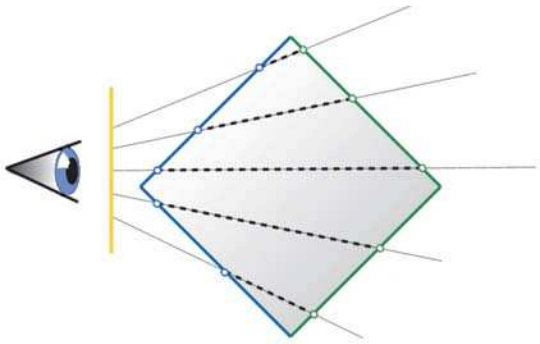


Figure 2.7: The ray-casting principle. Viewing rays are traced from the viewpoint through the volume, accumulating color and opacity values at each sample position along the ray.

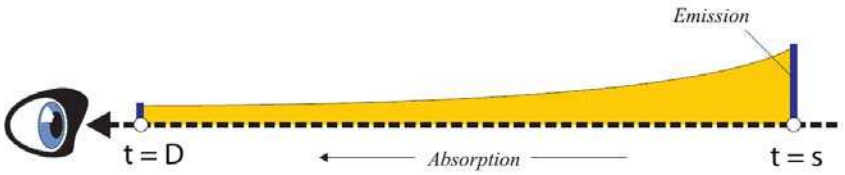


Figure 2.8: By evaluating the volume rendering integral, the amount of light that reaches the viewpoint is calculated. Image courtesy of Hadwiger et al. [29].

a Riemann sum, where the integration is split into n discrete intervals. :

$$I(D) = \sum_{i=0}^{n} c(x_i)\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j) \tag{2.3}$$

$c(x_i)$ in Equation 2.3 denotes the color at position x_i, α_i is the sample's opacity.

The mapping of scalar intensity values of the raw data to color and opacity is usually done by *transfer functions*. An example of a 1D transfer function editor can be seen in Figure 2.9. Using 1D transfer functions, every sample with the same intensity value is assigned the same color and opacity. Figure 2.10 depicts volume rendered images with and without transfer functions.

However, if spatially distinct regions in a dataset have the same intensity range it is impossible to distinguish between these regions by only using the transfer function, thus occlusions occur. Multi-dimensional transfer functions [39] try to alleviate this problem by using additional parameters for the lookup of color and opacity. To enhance boundaries between structures, for example, the intensity

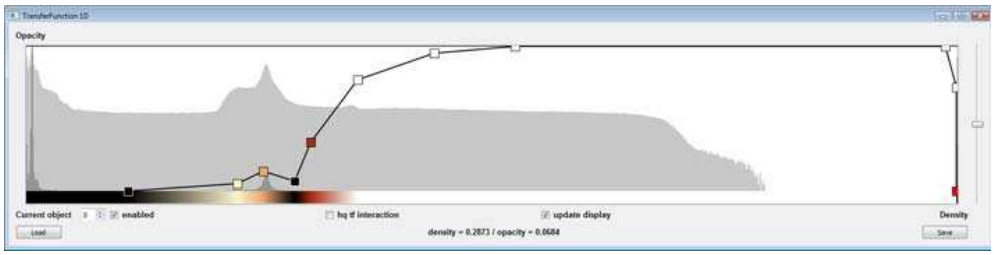
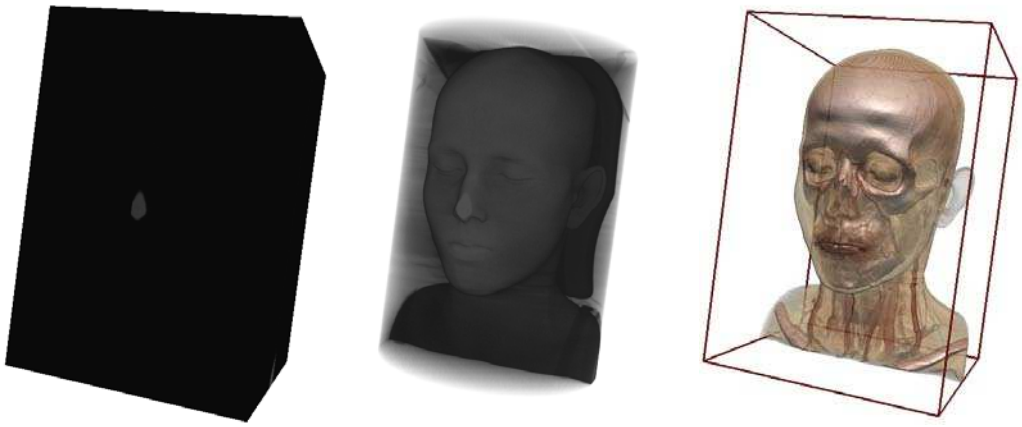


Figure 2.9: GUI transfer function editor displaying the current transfer function and the original data histogram (in log and linear format). The user can add colored transfer function nodes in the editor, with the x-axis corresponding to the original data's intensity, and the y-axis specifying the opacity of the transfer function. Colors and opacities are linearly interpolated between the nodes.



(a) (b) (c)

Figure 2.10: Volume rendering with and without transfer functions. (a) Volume rendering of original intensity values (i.e., grey values), with constant opacity. (b) Volume rendering with opacity ramp. (c) Volume rendering with transfer function from Figure 2.9.

value can be used in combination with the gradient magnitude to assign colors and opacity. A major problem of transfer functions, however, is the difficulty in designing and specifying good transfer functions, which is a research area on its own [8, 67, 76, 87].

2.3.2 Volume Visualization for Neurosurgery

In neurosurgery, surgical approaches tailored to an individual patient's anatomy and pathology have become standard. Therefore, precise preoperative planning

is necessary to achieve an optimal therapeutic effect. Volume rendering is used in many surgical planning systems to support the surgeon with high-quality 3D visualization. Especially when multiple radiological imaging modalities are used to delineate the patient's anatomy, neurological function, and metabolic processes, concurrent 3D visualization of these datsets can significantly help in developing a three-dimensional perception of the surgical approach. Traditionally this was done by just mentally fusing the different datasets. However, in neurosurgery there are many small objects and high-risk structures that must not be damaged during the surgical approach, which further emphasizes the need for computer aided surgical planning. Many different systems have been proposed for the planning and simulation of different kinds of neurosurgical procedures [4, 24, 32, 42, 59, 69, 70, 85]. Recently fiber tracking, the reconstruction of microstructural characteristics in the brain and central nervous system using diffusion tensor imaging (DTI) has become a popular area of research, leading to several neurosurgical planning tools such as virtual Klingler dissection [75].

Applications for *endoscopic approaches* to the brain simulate the reduced field of view and camera movement during endoscopic interventions [2, 42, 59]. In endoscopic approaches to the sinuses or pituitary gland a rigid endoscope is inserted into the patient's nose and advanced through the nasal airways. STEPS [59], an application for advanced virtual endoscopic pituitary surgery, not only simulates the endoscopic view, but also restricts the surgeons movement of the endoscope and simulates surgical tools for removing bones and tumors (see Figure 2.11).

More recently, Krueger et al. [42] proposed a system for simulating sinus endoscopy, focusing on GPU-based volume rendering and a realistic representation of the rendered biological structures such as mucosa.

Multi-volume rendering deals with the concurrent visualization of multiple volumes in a single rendering. Most neurosurgical planning applications heavily rely on image data from multiple modalities. Simultaneously visualizing anatomical and functional data enables the surgeon to examine the spatial relationship between brain activation and brain tissue [73, 71, 32].

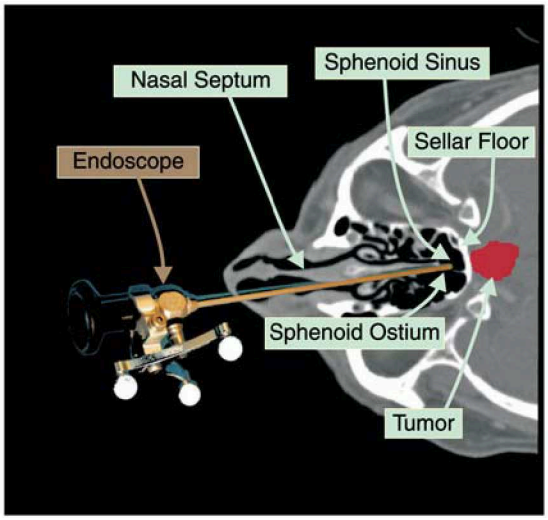
Different approaches for multi-volume rendering allow different levels of data intermixing [22]. The first distinction lies in the number of volumes that are displayed simultaneously at one sample position, either displaying only one volume per sample (i.e., one property per point) or blending multiple volumes at one sample position (i.e., multiple properties per point). Furthermore, when multiple properties are displayed per sample, property fusion can occur at different points in the volume rendering pipeline. Usually volume properties are fused either based on their raw intensity values, as gradient or material fusion, or during the shading or compositing steps. A more in-depth review of related work in the field of multi-volume rendering is presented in Section 4.2.

Recently an application for neurosurgical tumor treatment was proposed by

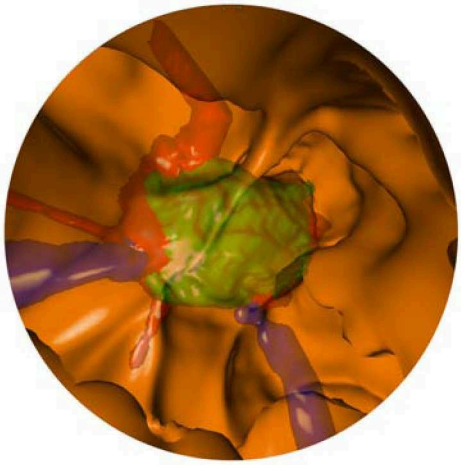
Rieder et al. [69] which uses multimodal data and offers distance-based enhancements of functional data and lesions and allows the visual exploration of the surgical approach to the structure of interest by advanced clipping and cutaway tools.

Advanced *clipping and cutaway techniques* in volume rendering allow the display of structures of interest which would otherwise be occluded. The specification of the cutting area can be done either geometrically or based on volume features. Exploded views or context-preserving techniques such as presented by Bruckner et al. [11] try to display the interior of a volumetric dataset while preserving context information (see Figure 2.12). Semantic volume rendering approaches try to support the non-expert user by allowing users the specification of volume rendering parameters in the natural language of the domain [65].

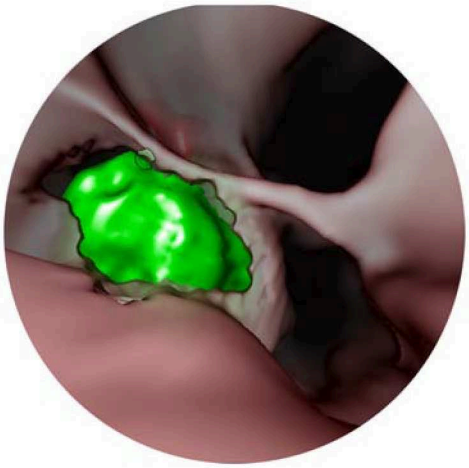
If *segmentation information* is available, clipping of segmented objects is possible [4]. Segmented objects are usually either displayed as surface mesh [24] or as voxelized, binary segmented objects [28]. While the integration of surface models into a volume rendered image needs extra care, the segmentation information can be displayed as a smooth surface. Binary segmented objects, on the other hand, allow a volumetric display of the segmented structure, including the assignment of different transfer functions and render modes.



(a)



(b)



(c)

Figure 2.11: STEPS - A virtual endoscopy training application. (a) Endoscopic approach. b) Tumor and vessel visualization in endoscopic view. (c) Tumor resection tool. Image courtesy of Neubauer [58].

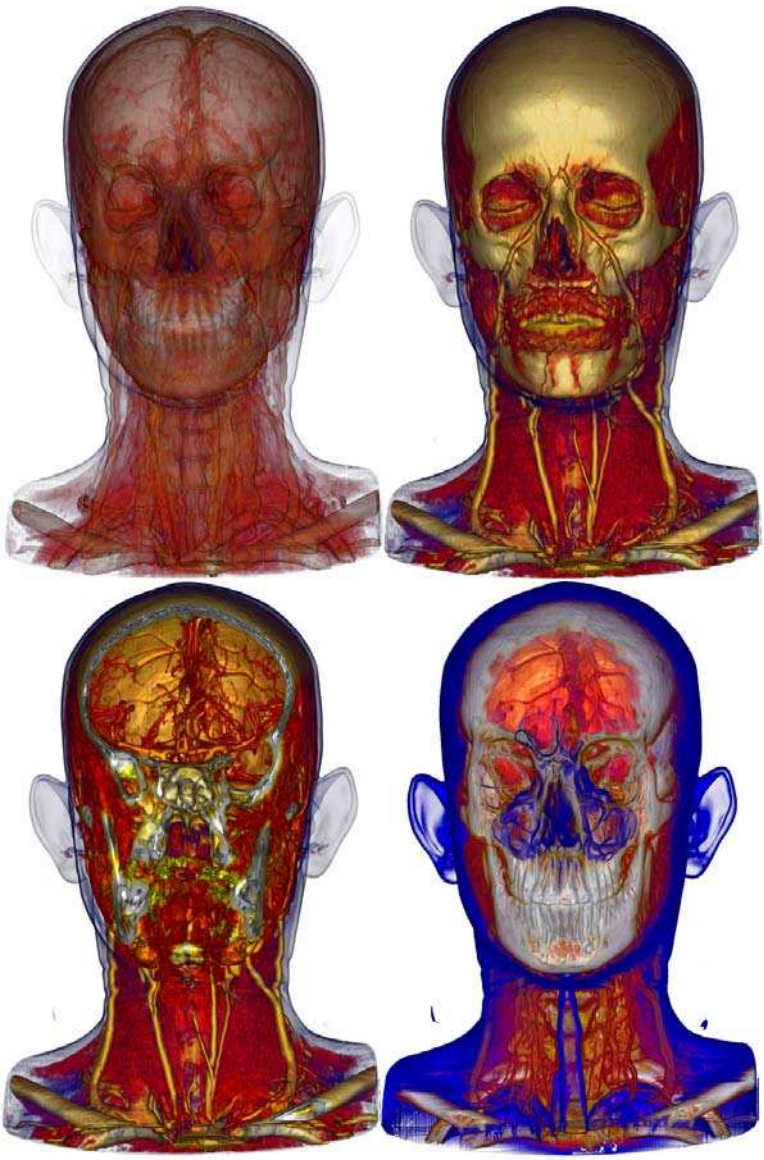


Figure 2.12: Illustrative context-preserving volume rendering tries to display the interior of volumetric datasets while preserving context information. Image courtesy of Bruckner et al. [11].

Chapter 3

Skull Peeling

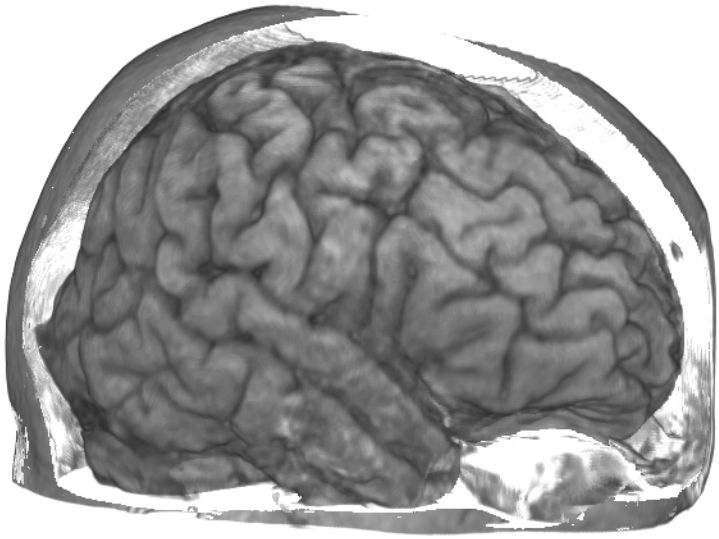


Figure 3.1: Skull Peeling, fast visualization of the brain without prior segmentation.

3.1 Introduction

Parts of this chapter are based on the papers *Segmentierungsfreie Visualisierung des Gehirns für Direktes Volume Rendering*, in Proceedings of Bildverarbeitung für die Medizin [5] and *High-Quality Multimodal Volume Rendering for Preoperative Planning of Neurosurgical Interventions.* IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007) [4].

In today's clinical practice, direct volume rendering (DVR) is already used routinely for the 3D visualization of medical volume data, such as those created by MR or CT scanners. The main advantage of DVR compared to surface rendering methods is the higher information density of the images. Transfer functions, which map the intensity values of the raw data to color and opacity, are used to create meaningful images. However, if spatially distinct regions in a dataset have the same intensity range, it is impossible to distinguish between these regions by only using the transfer function. This leads to occlusions and ambiguities in the volume rendered image. Higher dimensional transfer functions might alleviate this problem, however their design is even more burdensome than the design of 1D transfer functions.

Therefore, one of the biggest problems in DVR is the occlusion of potentially interesting areas by other structures of the same intensity/density. One example is data from an MR scan of the head, where the brain has the same intensity values as some outer tissues and, therefore, gets occluded. Due to this occlusion, visualizations of the brain usually rely on pre-segmentation methods (the so-called skull stripping), where the brain gets segmented prior to visualizing it. Unfortunately, the use of skull stripping tools in clinical practice is not always possible due to constraints in quality, robustness and available user time.

This section describes the *skull peeling* algorithm for directly displaying the unoccluded brain from MR data without the need for prior segmentation. Keeping the requirements for clinical applications in mind, our intention is to reliably visualize the brain without the need of tedious preprocessing or complex user interaction. Naturally, a manual segmentation of the brain would achieve the best visual results, but would require a much longer preprocessing time, which we want to avoid.

Our algorithm is based on the idea of opacity peeling [68], a view-dependent method for peeling away layers in DVR guided by accumulated opacity. In other words, it tries to reduce viewpoint-based occlusions by skipping over outer "uninteresting" structures to expose inner structures of a volume dataset. Although opacity peeling quickly generates meaningful images, a major problem for medical practice is its dependency on the threshold parameters. Minor changes of these settings can cause major changes in the resulting images (such as a shrinking or expanding brain). Thus, it is an important goal to improve reliability. The work described in this chapter arises from the requirements of neurosurgical applications to be able to extract and visualize the brain in an MR dataset in a fast and robust way, without much user interaction and in high quality.

Additionally, an application for the display of implanted electrodes, used for epilepsy surgery, and for the display of superficial brain tumors was developed based on the described algorithm. Using skull peeling, the surgical approach to the brain can be simulated by only removing those parts of the skull that need to be removed for surgery.

3.2 Related Work

High-quality renderings of the human brain from cranial MR scans usually require segmentation due to the brain being occluded by surrounding structures and tissue. However, this segmentation process, called skull stripping [1, 78], is not trivial and automatic methods often have problems with noise or require certain MR sequences or scanners. The interested reader is referred to Atkins et al. [1] and Song et al. [78] where different skull stripping approaches are compared and evaluated.

If the brain is rendered without prior segmentation, it is occluded by surrounding tissue of similar intensity values (e.g., skin). Adjusting only the transfer function, including multi-dimensional transfer functions [39], cannot solve this problem. Methods such as opacity peeling [68] or confocal volume rendering [57] are ray-casting-based methods that peel away outer, less important regions of a volume to visualize inner structures. These methods, however, are hard to use in clinical applications because the visual results are very sensitive to several user-defined parameters.

As described by Rezk-Salama and Kolb in [68], opacity peeling is based on ray-casting. The first step consists of stepping along the ray and accumulating color and opacity, as in standard ray-casting. However, when the accumulated opacity along a ray exceeds a threshold T_1, the color- and opacity values of this ray are stored for later display and then reset. Stepping further along the ray, when the opacity value of one single sample (i.e., the current sample) falls below a threshold T_2, the accumulation of color and opacity along the ray is started again (until threshold T_1 is exceeded again). This loop continues and permits to calculate several layers of the dataset in one render pass and display them subsequently.

Confocal Volume Rendering [57] is a viewpoint-dependent method for visualizing deep structures, where the volume gets displayed only beyond a certain depth and for a user-defined length. Bruckner et al. [11] have introduced a context-preserving DVR method, where a function of shading, gradient magnitude, distance to the viewpoint and accumulated opacity directly influences the opacity.

However, the final visualization results of these methods for selectively displaying occluded structures are often hard to predict and highly dependent on manual parameter settings. We try to alleviate this problem in our new skull peeling method.

Our visualization framework is based on DVR [44], which creates images directly from volumetric data without the need to extract any geometry first. The major advantage of DVR is the increased amount of information that can be conveyed in one image by making use of transparency, which allows users to peer inside the volume. Additionally, transfer functions and lighting can significantly enhance the 3D perception of the volumetric structure. Hardware accelerated ap-

proaches allowing interactive high quality volume visualization on standard PCs range from 2D texture mapping based algorithms [66] to more recent GPU-based ray-casting [41]. Rendering of segmented volume data imposes the additional problem of filtering object boundaries at high resolution and has been addressed by Hadwiger et al. [28]. They use GPU-based two-level volume rendering to specify transfer functions and render modes on a per object basis with trilinear object filtering.

3.3 Skull Peeling

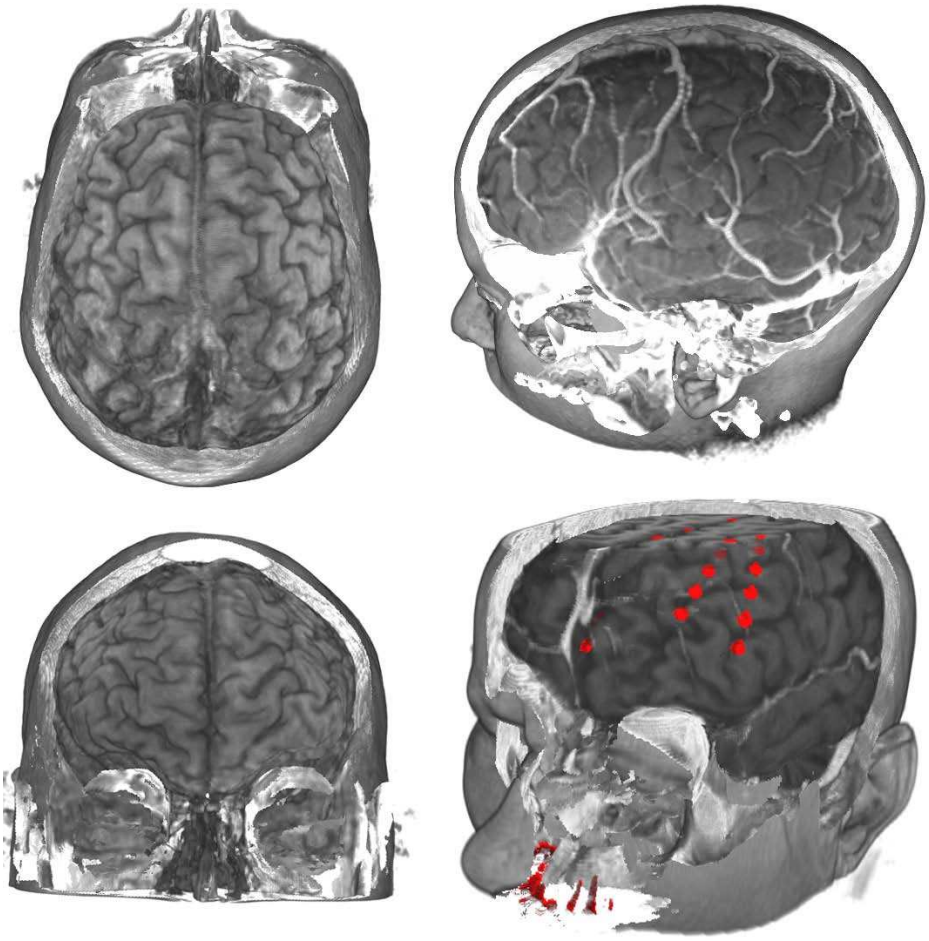


Figure 3.2: Different examples of skull peeling. The brain, superficial vessels and implanted electrodes can be displayed fast and without any prior segmentation of the brain.

The skull peeling algorithm simultaneously uses the information of registered CT and MR volumes in order to remove areas along the viewing ray which occlude

the brain in the MR volume [5]. Different examples of skull peeling can be seen in Figure 3.2.

While the brain is depicted well in MR scans, CT scans are superior in depicting bony structures with very high intensity values. We exploit this knowledge to decide automatically if a sample lies within a bony area (i.e., the value of the CT dataset is above 1000 Hounsfield units). During ray-casting, both the CT and the MR volume are sampled. When the current ray hits a bone for the first time, the accumulated opacity and color values from the MR volume are reset and the ray is advanced until the bony area is exited. At that point, accumulation starts again in order to reveal the brain. This algorithm needs no user input and works well in standard cases where the brain is surrounded by bone (see Figure 4.2a).

In many cases the need of a registered CT does not lead to an additional screening for the patient, because a CT is required anyway for intraoperative navigation. Due to this additional volume, the skull peeling algorithm is independent of the user-specified parameters of the original opacity peeling algorithm, thereby increasing the reliability and quality of the visualization.

However, when the brain is not surrounded by bone (e.g., after surgery, or when clipping planes are enabled during rendering) this algorithm would fail. The ray would hit a bone for the first time after traversing the brain and everything in front of that hitpoint would be skipped (see Figure 3.3b). We therefore added the following extensions:

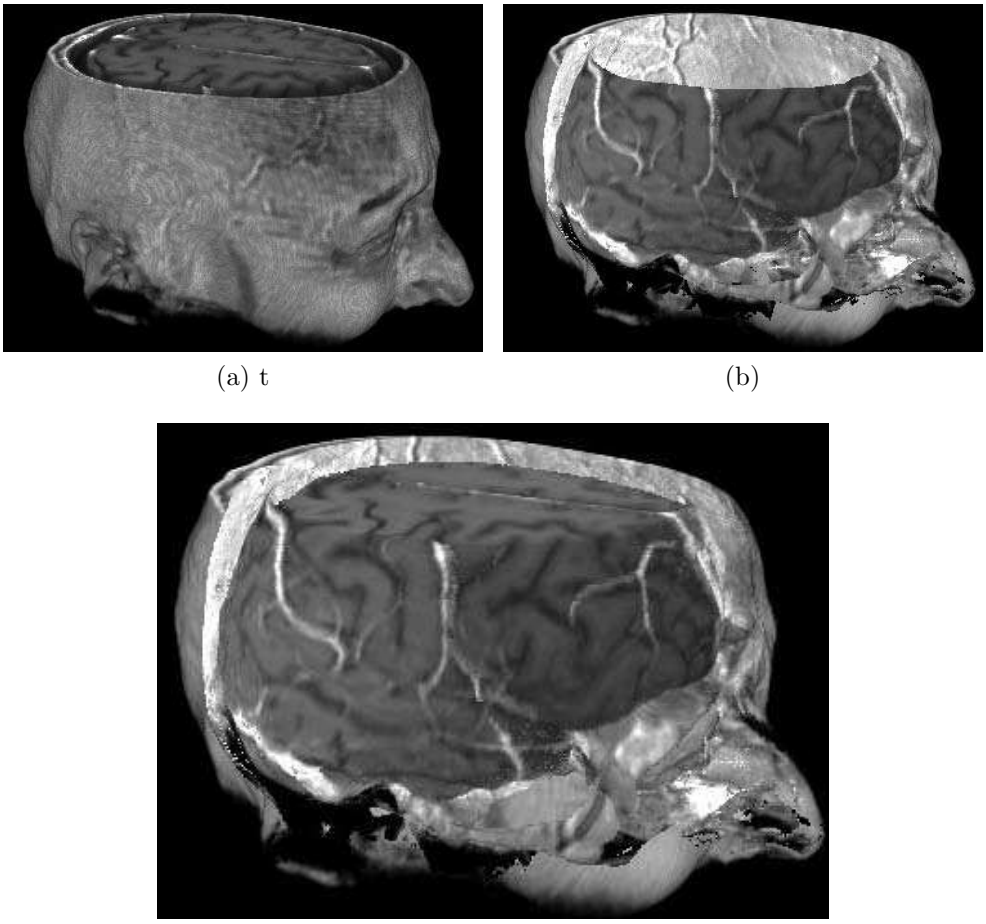
- The position of the first hitpoint of the skin (i.e., first sample with a density higher than air) is saved. If the ray does not hit a bone within a certain number of steps (defined by threshold T_{clip_1}) we assume that there is no bone in front of the brain and use the radiance accumulated from the first hitpoint.

- A second extension to the algorithm was made to improve visualization near the skull base. When looking at the brain from below, along the spinal chord, many small bone pieces occlude the view of the brain. We introduce a threshold T_{clip_2} which specifies the minimum distance two bony areas must have in order to assume the area in-between to be brain. If this distance is not reached, the area between these two bone areas is skipped and not rendered.

Both thresholds have default values that usually work very well and only need to be adjusted for special cases (e.g., looking at the brain from below). Figure 3.4 outlines the standard case of skull peeling and a case where threshold T_1 is needed.

3.4 Neurosurgical Applications

The skull peeling algorithm described above, for displaying the brain surface, was applied to several different neurosurgical application scenarios, namely epilepsy



(a) t (b)

(c)

Figure 3.3: Skull peeling of clipped volumetric datasets. (a) Original rendering. (b) Incorrect clipping, resulting from erroneously skipping parts of the brain that are not occluded by the skull. (c) Corrected clipping, after introducing the additional parameters T_{clip_1} and T_{clip_2}.

surgery planning and for planning of the surgical approach to the brain for surgery on superficial lesions.

3.4.1 Surgical Approach to the Brain

Finding the ideal position for the skin incision and subsequent bone removal is important for minimizing the invasiveness of a surgery. For this purpose, we introduce clipped skull peeling to visualize the simulated surgical approach (Figure 3.5). The user input consists of the surgeon drawing the areas for the skin incision and subsequent bone removal directly onto the volume-rendered image of the head. After skin removal, the skull is rendered with shaded DVR, as this

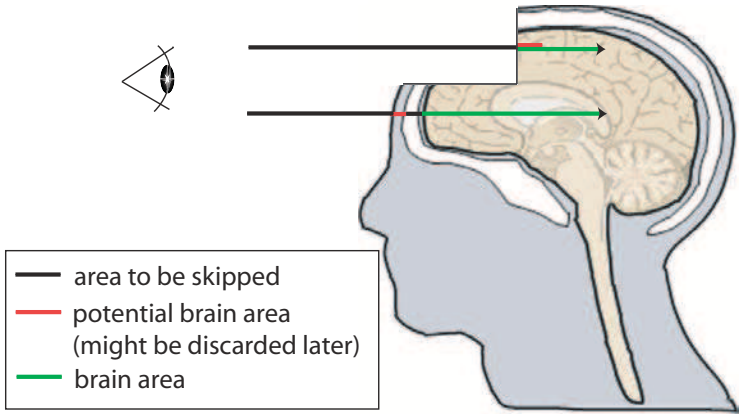
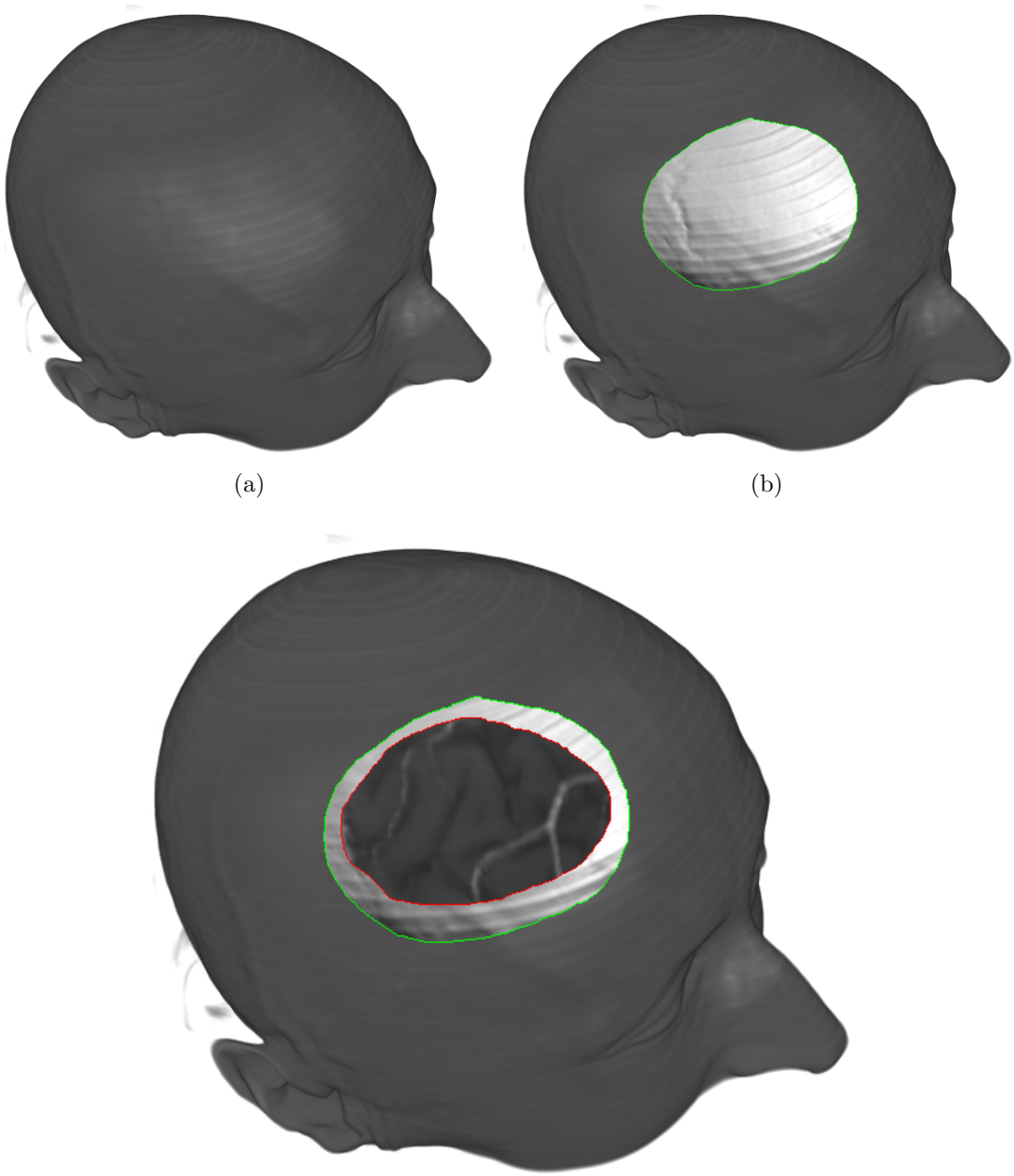


Figure 3.4: Skull peeling algorithm. The lower ray displays the standard case where the ray hits the skull prior to the brain. The upper ray depicts the case where the brain is not covered by bone.

enhances the 3D perception and helps the surgeon to find anatomical landmark points on the skull, which can be used as orientation aides during surgery. The result of clipped skull peeling is generated in three ray-casting passes by using the stencil buffer in order to restrict pixels and thus rays to one of three different cases with one specific rendering mode each: (1) Everything outside the specified clipping areas is rendered using unshaded DVR of the MR volume; (2) Inside the skin incision area the skull is displayed (shaded DVR of the CT data), but accumulation of color and opacity is started only after the threshold for bone has been exceeded; and (3) The bone removal area is skull-peeled. The assignment of these three rendering modes to their corresponding pixels is performed as follows: After clearing the stencil buffer to zero, the polygon that was drawn by the user to simulate the skin incision is rendered, increasing the stencil values of the covered pixels to one. Next, the polygon drawn for bone removal is rendered as well, which increases the stencil values of the corresponding pixels to two. However, for rendering the bone removal polygon the stencil function is set such that stencil values are only modified in areas where the stencil buffer is already one. This ensures that the bone is only removed in areas where the skin has already been removed. Then, the three ray-casting passes outlined above are performed, restricting rendering to the corresponding pixels by setting the stencil function accordingly. Note that this algorithm could easily be extended to more general view-dependent clipping methods. Simulating the surgical approach to the brain by planning the skin incision and removing the bone window can be seen in Figure 3.5.



(a) (b)

(c)

Figure 3.5: Planning the surgical approach to the brain. (a) Original volume rendering. (b) Displaying the skin incision area. (c) Removing the bone window.

3.4.2 Epilepsy Surgery Planning

To localize the source of epileptic seizures, one common method is the use of invasive electrodes. These small electrodes are implanted onto the patients brain surface during a small surgery. Next, the patient is supervised and when an epileptic seizure happens the location of the source of the seizure can be determined by the read-out of the implanted electrodes. This information can subsequently be used for surgery, where the part of the brain that causes the seizures is operated on. However, a big problem for the surgeons is to determine the exact location of the implanted electrode in regard to the brain and the brain's surface prior to surgery. For imaging of the electrodes a CT is acquired, which we use for threshold segmentation of the electrodes. Using skull peeling we can display the brain's surface along with the segmented electrodes. Since the implanted electrodes usually seem to sink into the brain's surface a little bit we can optionally improve their visibility in the skull peeled image. This is achieved by resetting the color and opacity values of the ray when first hitting an electrode, as long as the opacity along the ray has not reached a maximum before (see Figure 3.6b).

3.5 Results and Evaluation

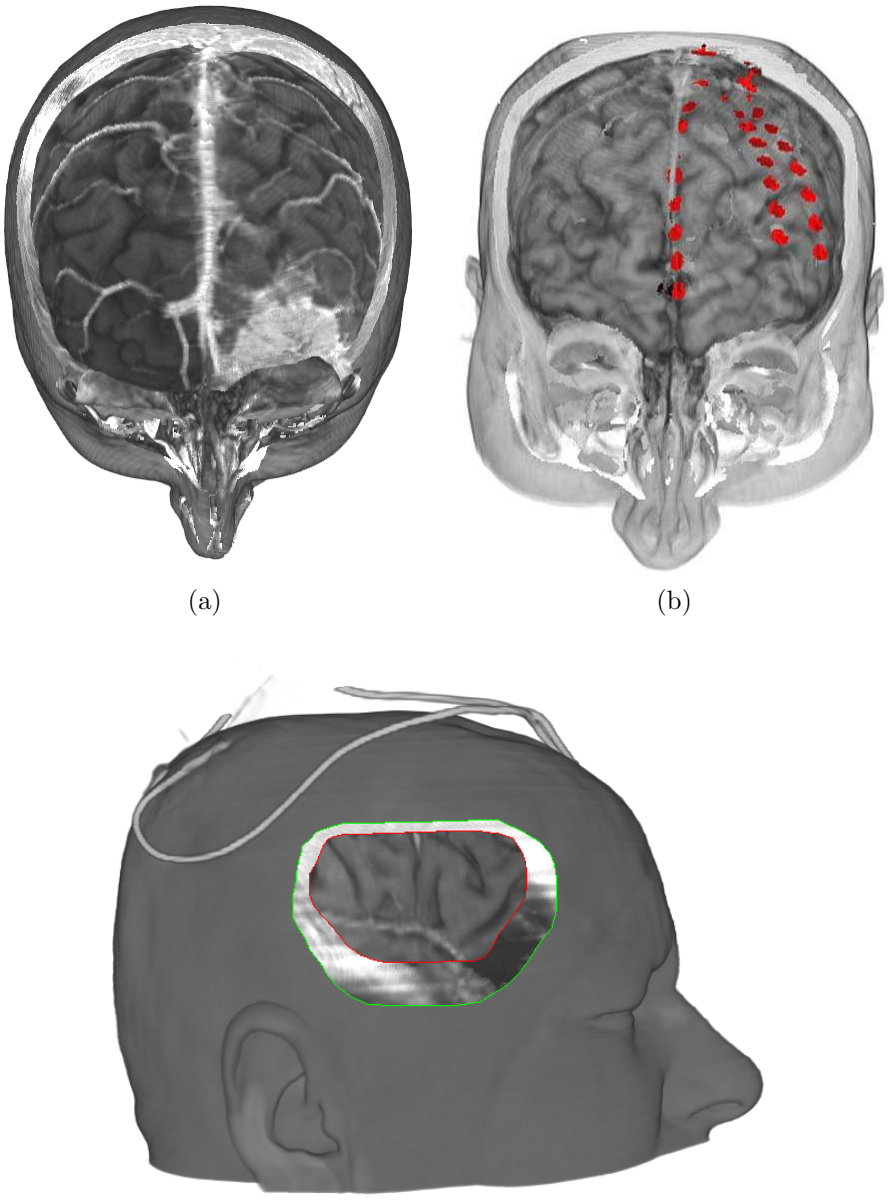
The algorithm was implemented on a Pentium IV 3,2 GHz PC with a ATI X1800 graphics card. A MR T1 dataset ($512 \times 512 \times 512$) achieves frame rates of 14 fps which illustrates the interactivity of the approach. For a more detailed evaluation see Section 4.6 in Chapter 4.

3.6 Summary and Conlusion

The proposed method allows us to visualize the exact position of superficial brain tumors as well as surrounding vessels (see Figure 3.6c). Implanted electrodes for the localization of epilepsy centers are displayed in Figure 3.6b.

Skull peeling allows for a simple and fast visualization of the brain's surface and does not require any tedious pre-segmentation of the brain. This was very positively received by the medical doctors who were using our system.

Our system is mainly designed for the use in time critical applications, where a high-quality segmentation of the brain is not feasible or possible. Skull peeling is not entirely artifact-free (especially near the silhouette) but it achieves good results in a very small amount of time. For a more detailed evaluation of the integration of our skull peeling algorithm into a 3D neurosurgical planning application see Section 4.6.



(a) (b)

(c)

Figure 3.6: Medical example applications for using skull peeling. (a) Visualization of a superficial brain tumor. (b) Displaying implanted electrodes for epilepsy surgery. (c) Simulation of the surgical approach to the brain.

Chapter 4

Preoperative Planning of Neurosurgical Interventions

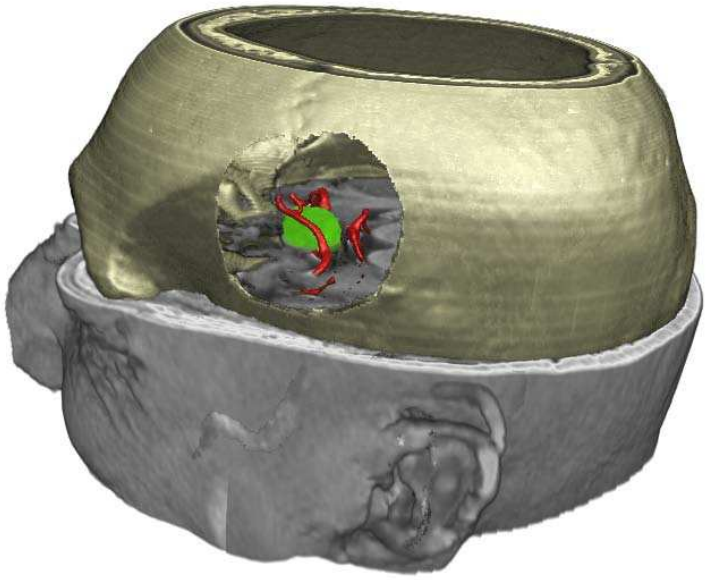


Figure 4.1: Multi-volume rendering of segmented data (green: tumor - MR, red: vessels - MRA, brown: skull - CT).

4.1 Introduction

Parts of this chapter are based on the papers *High-Quality Multimodal Volume Rendering for Preoperative Planning of Neurosurgical Interventions.* IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007) [4].

Surgical approaches tailored to an individual patient's anatomy and pathology have become standard in neurosurgery. Precise preoperative planning of these procedures, however, is necessary to achieve an optimal therapeutic effect. Therefore, multiple radiological imaging modalities are used prior to surgery to delineate the patient's anatomy, neurological function, and metabolic processes. Developing a three-dimensional perception of the surgical approach, however, is traditionally still done by mentally fusing multiple modalities. Concurrent 3D visualization of these datasets can, therefore, improve the planning process significantly.

Minimally invasive neurosurgical procedures are constantly gaining importance with the aim to minimize surgical trauma, shorten recovery times and reduce postoperative complications. For surgery of deep-seated structures, *neurosurgical keyhole procedures* are becoming standard, where a small opening in the skull is sufficient to gain access to a much larger intracranial region via an endoscope or operating microscope. In contrast, interventions in areas directly below the skull require a larger and individually tailored opening of the cranial bone. For both approaches (i.e., surgery of deep-seated structures and near the brain's surface), orientation is necessary to perform the skin incision and bone cover removal at the optimal location. For deep-seated targets, further orientation is crucial to find the structures of interest while additionally preserving the surrounding tissue. *Preoperative planning* enables the surgeon to identify anatomical landmarks and critical structures (e.g., large vessels crossing the path of the operating microscope or critical cranial nerves) and in determining the optimal position of incision prior to surgery. It is during this planning session that the physician decides upon the optimal approach by adapting the general surgical plan to the individual patient's anatomy. The medical doctor uses this knowledge during surgery to determine the current location in the skull and the subsequent optimal course of action. Therefore, the success of a surgery, especially in keyhole approaches, depends largely on accurate preoperative planning.

Up to now, the standard approach of presurgical planning is performed using stacks of raw images obtained from medical scanners such as CT (Computed Tomography) or MRI (Magnetic Resonance Imaging). In the field of neurosurgery, MR scans are the medium of choice for depicting soft tissue such as the brain, whereas CT scans are superior in picturing bony structures. Functional MR (fMR) images depict neural activity, Positron Emission Tomography (PET) shows metabolical activity and Digital Subtraction Angiography (DSA) depicts vessels in high quality. However, a mental combination of all these datasets and a correct 3D understanding by simple slice-by-slice analysis is very difficult, even for the skilled surgeon.

3D visualization alleviates this problem by enhancing the spatial perception of the individual anatomy and, therefore, speeding up the planning process. Considering the neurosurgical background, a preoperative planning application has to meet certain requirements: First of all, it should provide a *high-quality, inter-*

active and flexible 3D visualization of the volumetric dataset using direct volume rendering (DVR). Next, a preoperative planning application should offer *multimodal visualization* of datasets from different imaging modalities such as CT, MRI, fMRI, PET or DSA. *Interactive manipulation* of the visualization such as simulated surgical procedures, endoscopic views or virtual cutting planes should be available and, finally, an *intuitive workflow* is necessary, which is integrated into an application framework and ready for use by surgeons or medical staff.

In this chapter we introduce an application for planning of individual neurosurgical approaches with high-quality interactive multi-volume rendering for the concurrent and fused visualization of multimodal datasets (see Figure 4.2 for several examples).

The application supports three main tasks:

- *Planning of the surgical approach to access the brain*, by simulating the optimal skin incision and removal of the cranial bone tailored to the underlying pathology and without any prior segmentation.
- *Visualization of superficial brain areas*, including information from additional volumes such as DSA, fMR or PET to provide further insight into the individual brain anatomy, function and metabolism.
- *Visualization of deep-seated structures of the brain for (keyhole) surgery*, by including segmentation information.

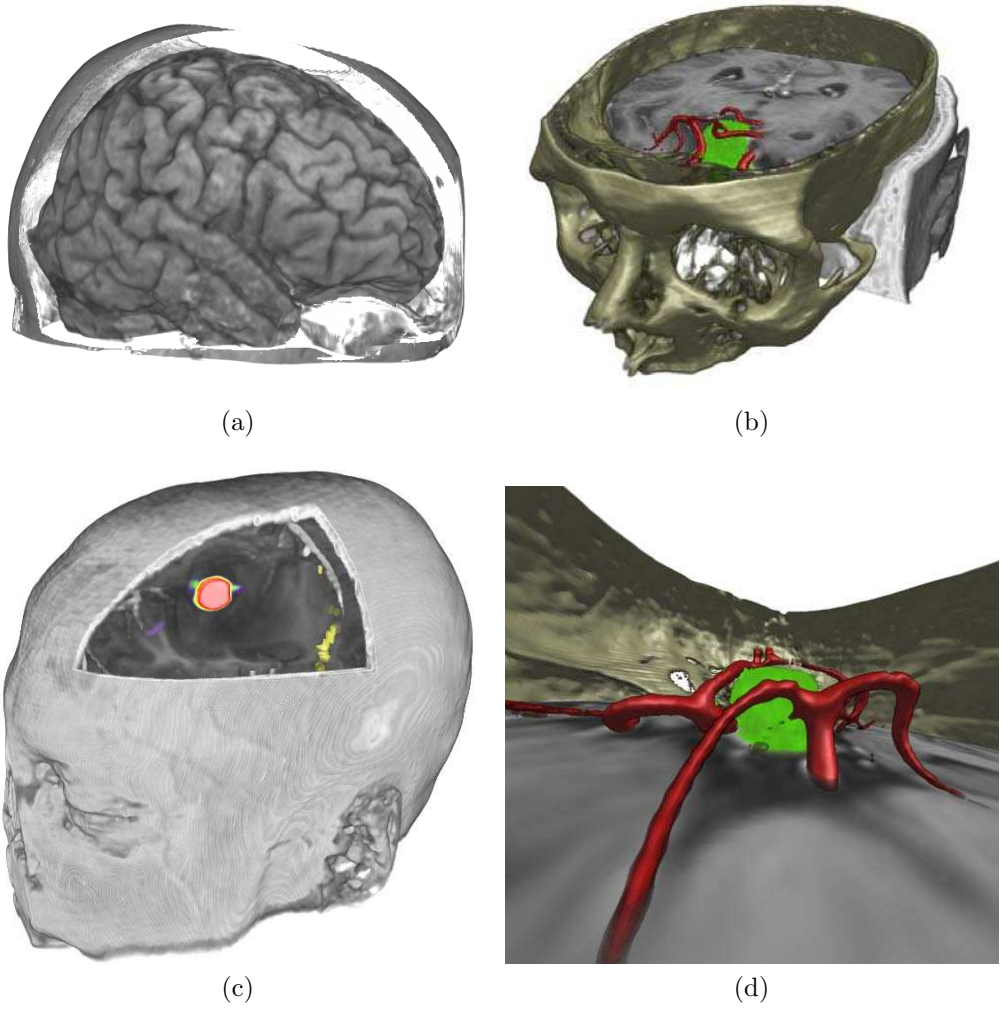
The visualization is based on direct multi-volume ray-casting on graphics hardware, where multiple volumes from different modalities can be displayed concurrently at interactive frame rates. Graphics memory limitations are avoided by performing ray-casting on bricked volumes. For preprocessing tasks such as registration or segmentation, the visualization modules are integrated into a larger framework, thus supporting the entire workflow of preoperative planning.

All visualization modules are integrated into a framework that is designed to support surgeons in the task of preoperative planning, including a preprocessing stage for registration and optional segmentation of the different datasets.

Rendering performs real-time GPU ray-casting with perspective projection, in general using a single ray-casting pass for 32-bit floating point computations and blending. We employ efficient empty space skipping, early ray termination, and bricking for memory management of multiple volumes. Ray-casting is performed through several volumes at the same time, potentially taking into account multiple volumes at a single sample location.

The technical contributions presented in this chapter are:

- Unified handling of multi-volume ray-casting and bricking, with and without segmentation masks. For each sample location, the volume to be sampled is either chosen depending on segmentation information, or multiple volume



(a) (b)

(c) (d)

Figure 4.2: (a) Visualization of the brain without prior segmentation using our skull peeling algorithm. (b) Multi-volume rendering of segmented data (green: tumor - MR, red: vessels - MRA, brown: skull - CT). (b) Multi-volume blending (black/white: brain - MR, red: metabolic active part of tumor - PET, yellow: brain areas active during speech - fMR). (c) Perspective multi-volume rendering for simulating keyhole surgery.

samples are blended. We circumvent GPU memory constraints by bricking each volume (CT, MR, DSA, PET, fMR), and downloading only active bricks into 3D cache textures (one per modality or unified). Segmentation information is represented as a bricked object ID volume over all modalities, which likewise employs a 3D cache texture.

- The integration of *Skull peeling* (see Chapter 3) for selectively removing structures obscuring the brain (skin, bone) without segmentation (Figure 3.4). In contrast to opacity peeling [68], we consider registered CT and MR data at the same time for more dependable results. The impact of clipping is resolved consistently. Layers can also be peeled away selectively by painting 2D clipping areas (Figure 4.10).

- The result of skull peeling is view-dependent. In order to employ it for powerful view-independent clipping, we first generate a view-dependent depth map, which is then transformed into volume space and resampled into a static segmentation mask.

- Smooth rendering of segmented object boundaries, taking into account the contributions of multiple volumes. In contrast to earlier work [28], we do not propose a general solution, but an approach that is customized for the needs of our neurosurgery pipeline that achieves better results in this case. During ray-casting, the precise transition between two adjacent materials is re-classified depending on user-specified iso-values and searching the object ID and data volumes along the gradient direction.

4.2 Related Work

The main advantage of DVR compared to surface rendering lies in the increased amount of information that can be conveyed in one image, as the entire volumetric dataset is used to create the final rendering.

For rendering multimodal data, several methods have been proposed [13, 96, 25, 23]. Their key differences lie in the way how the volumes are combined. Different data intermixing levels (e.g., accumulation level, illumination level, image level) and fusion methods (e.g., one or multiple properties per sample) are used depending on the characteristics of the volumes and the desired results. Manssour et al. [52] use an MRI volume to define the opacity transfer function, while a PET volume determines the color transfer function. Clusters [96] and specialized volume rendering hardware [25, 71] have also been used. Recently, Rößler et al. [71] introduced a slice-based multi-volume rendering method to display a template brain along with patient-specific fMR data, including advanced clipping techniques and render modes. All of the above methods, however, do not address the problem of high-quality rendering of segmented multimodal data.

For high-quality rendering of segmented data, object boundaries must be determined at the subvoxel level [84, 28, 89], mostly using linear or cubic filtering. Tiede et al. [84] propose a CPU-based method for threshold-segmented objects. They compare the intensity of each sample to the objects in its 2^3 neighborhood to assign the object ID. If the objects have not been segmented via thresholding,

trilinear filtering of object masks is used. They also propose to extend their approach to multimodal data. Two-level volume rendering [28] is a flexible rendering method for segmented data with trilinear object boundary filtering and per-object transfer functions and rendering modes. Preoperative planning for neurosurgery has been an active research topic for several years [77, 33, 59, 69], with the main focus often on the integration of imaging, registration, and segmentation into a planning workstation [17], but often falling short of a high-quality visualization of multi-volume datasets. The virtual tumor resection planning of Serra et al. [77] uses volume slicing with basic support for multiple volumes. Even though volume slicing approaches achieve a fast and flexible visualization, they usually do not reach the quality of ray-casting methods, especially with respect to close-up perspective views. Other approaches employ iso-surface rendering [59], or the extraction of 3D contours that can subsequently be blended into a mono-volume rendering [33]. This, however, is not optimal for versatile preoperative planning as it does not offer the amount of flexibility needed by surgeons for interactive exploration, e.g., changing the transfer function.

Rieder et al. [69] recently proposed a neurosurgical tumor treatment application which uses multimodal data, offers distance-based enhancements of functional data and lesions, and supports advanced clipping and cutaway tools to visually explore the surgical approach. An illustrative hybrid visualization method for anatomical and functional brain data was proposed by Jainek et al. [32].

We build on previous research in the area of multimodal volume rendering, and especially GPU-based ray-casting. While first approaches were based on slicing [93, 66], GPU ray-casting is now a viable and very powerful alternative [41]. The basis for our volume rendering framework is a GPU-based ray-caster [74] (requiring Shader Model 3.0) that achieves interactive frame rates also for large volumes. However, we have extended this ray-caster considerably in order to support multiple volumes, segmentation masks, flexible per-object as well as view-dependent clipping, and rendering modes tailored for neurosurgical applications.

4.3 Workflow

For daily use in clinical environments it is crucial for CASP applications (Computer Aided Surgical Planning) to be integrated directly into the clinical workflow. CASP applications should support the surgeon, who usually has a very tight schedule, by offering an intuitive, easy-to-use interface. Therefore, we have integrated our rendering framework as a plugin into the medical workstation and PACS system Impax 6.0 by Agfa Healthcare[1]. The complete workflow of our planning tool can be seen in Figure 4.3 and contains the following steps: Data acquisition, registration, segmentation, planning of the skin incision and bone removal area, brain surface visualization, and surgery planning for deep-seated

[1] http://www.agfa.com

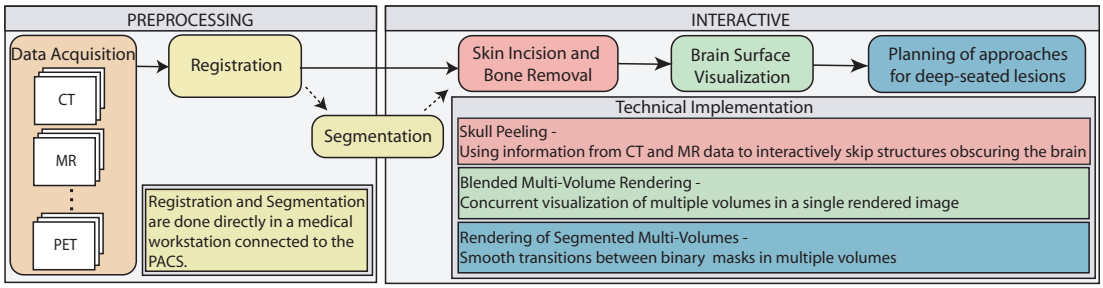


Figure 4.3: Detailed workflow of our neurosurgical planning application. Data acquisition, registration and segmentation are performed in a preprocessing step. The interactive part of the workflow consists of three main parts, described in Section 4.5.

structures, tailored to the individual anatomy. The application consists of a preprocessing stage, which was intentionally kept as short as possible, and an interactive stage for rendering. After acquisition of the radiological images, the different datasets are registered and segmented directly in the medical workstation before starting the interactive visualization. Segmentation is not mandatory but helps later on by giving additional information during the visualization of deeper structures. Optionally, diffusion-based smoothing or segmentation [6] can be employed. Using skull peeling (Section 4.5.2), the surgeon can take a look at the brain's surface and optionally define the area on the patient's head for the skin incision and bone window removal. The superficial brain can be displayed via direct multi-volume rendering (Section 4.5.3), showing the brain (MRI) along with other structures such as vessels (DSA), implanted electrodes (CT), or functional brain areas (fMRI, PET). Next, the surgeon may navigate the viewpoint through the keyhole in the cranial bone to access deeper brain structures, using multi-volume rendering of segmented data (Section 4.5.4). Additionally, binary segmented objects can be displayed in high quality and without staircase artifacts by an on-the-fly subvoxel classification algorithm based on ray-casting (Section 4.5.5). During the entire interactive visualization process, tools for further exploration of the data are available to the surgeon, such as MPR (multiplanar reconstruction) views, clipping geometry, or endoscopic views.

4.4 Preprocessing Stage

In the registration module, the different multimodal datasets are aligned and resampled. We use an automatic registration algorithm based on mutual information [14] and perform rigid registration (i.e., only translation and rotation). The algorithm usually converges after a few seconds. If the result is not satisfactory, the user can improve the registration manually by interacting with three orthog-

onal slice views. The segmentation module implements manual segmentation, thresholding, watershed segmentation based on markers [21] and diffusion-based segmentation [6]. Watershed based on markers is a semi-automatic segmentation algorithm where the user has to draw initial markers for the different objects into the volume data. For each voxel its most probable membership to an object is calculated. The algorithm completes within seconds, however, manual refinement of the initial markers is necessary most of the time. The binary segmented objects are saved in an additional segmentation volume that defines the object membership of each voxel.

4.5 Visualization Modules

The three visualization modules illustrated on the right-hand side of Figure 4.3 constitute the main part of our application. Section 4.5.1 provides a technical introduction to our multi-volume rendering system, which is the basis for all visualization modules. Specifics of the modules are then described in Section 4.5.2 for *skull peeling* in order to perform view-dependent clipping of "uninteresting" parts of a volume, Section 4.5.3 for *multi-volume blending* in order to visualize unsegmented data, and Section 4.5.4 for *segmented multi-volume rendering* for visualizing segmented objects from multiple modalities. Rendering smooth boundaries of segmented volumes is described in Section 4.5.5. General interaction tools which support the surgeon in the exploration and planning process are explained in Section 4.5.6.

4.5.1 Multi-Volume Rendering

A very important issue in multi-volume rendering is how the contributions of different modalities are combined. Our rendering pipeline offers several options that are part of a single consistent framework. Contributions of multiple volumes are combined on a per-sample basis during ray-casting. For a given sampling location, either a single volume is sampled and mapped to optical properties via a transfer function, or the contributions of multiple volume samples taken at the same relative location in each modality are blended *after* separate transfer functions have been applied.

A single object ID volume guides these choices for combining multiple volume contributions. Table 4.1 gives an overview of the most important texture types that are used in our implementation. Each voxel is assigned the ID of the segmented object it belongs to (*tex_objectID*), and each object is assigned the volume it belongs to using a 1D mapping (*tex_volumeID*). For example, "bone voxels" would be assigned to the CT volume, whereas "brain voxels" would be assigned to the MR volume. Each object can use its own transfer function, which is determined using its object ID [28]. The mapping of object ID to volume ID

implicitly solves the problem of using different transfer functions for different volumes, as long as each object uses only a single transfer function. Additionally, special *blending object IDs* known by the ray-casting shader are used to indicate that for this object a given configuration of multiple volumes should be blended at each sampling location, where each volume uses its own transfer function and blending weight. We currently support only a fixed number of useful configurations, such as blending the MR and the DSA volume used in the visualization module described in Section 4.5.3. However, adding additional configurations is easy, and our system could easily be extended to full generality using additional look-up textures.

Table 4.2 illustrates the basic texture look-ups used by multi-volume rendering. The transfer function TF_j in Table 4.2 is determined by the object ID, which either means $j = o(\mathbf{x})$ for regular object IDs, or a set of predefined j's is used by the shader because $o(\mathbf{x})$ is a blending object ID and thus uses multiple transfer functions. Each object can further have its own set of up to six axial clipping planes, whose positions are obtained in the shader by sampling two 1D look-up textures for minimum and maximum (x, y, z) coordinates, respectively (Table 4.1). Note that segmentation information is not strictly required. If no object ID volume is available, all parts of our pipeline will implicitly assume that all voxels belong to the same default object. In this case, the only option for combining different volumes is blending them.

Data and Memory Management: Another important challenge in multi-volume rendering is volume data management and coping with memory consumption. We use a bricked volume rendering scheme that subdivides each volume into equally-sized (e.g., 16^3 or 32^3) bricks and maintains 3D brick cache textures. For rendering, active bricks of the volume are held in GPU cache memory, and small 3D layout textures are used for address translation [30] between "virtual" volume space and actual cache texture coordinates. Figure 4.4 shows an overview of our system. One cache stores segmentation information (an object ID per voxel),

Texture	Dim.+ Type	Function
tex_objectID	3D (I)	map sample position to object ID
tex_volumeID	1D (I)	map object ID to volume ID
$\mathrm{tex\_volume}_i$	3D (I)	texture cache for volume i
tex_clipmin	1D (RGB)	map object ID to clip planes (min)
tex_clipmax	1D (RGB)	map object ID to clip planes (max)
tex_tf	2D (RGBA)	packed 1D textures TF_j for all j

Table 4.1: Basic multi-volume rendering textures. Texture type I is single-channel (intensity), and RGB is three-channel, to store three axial clipping plane positions (min or max for x, y, and z, respectively).

obtain ... at $\mathbf{x}$		as
object ID	$o(\mathbf{x})$	$=$ sample(tex_objectID, $\mathbf{x}$)
volume ID	$i(\mathbf{x})$	$=$ sample(tex_volumeID, $o(\mathbf{x})$)
volume scalar	$s_i(\mathbf{x})$	$=$ sample(tex_volume$_{i(\mathbf{x})}$, $\mathbf{x}$)
transfer function	$\mathrm{TF}_j(\mathbf{x})$	$=$ sample(tex_tf, $(s_i(\mathbf{x}), j)$)

Table 4.2: Basic multi-volume rendering quantities and texture look-ups.

with its corresponding layout texture. Each data volume (e.g, CT, PET) has its own layout texture, and either also uses an individual cache texture, or references bricks in a larger unified cache for multiple modalities. Individual caches are less flexible because their size must be chosen at startup, whereas a unified cache allows to change the memory limit dynamically for each modality. However, in this case the GPU must support the texture dimensions required by the larger unified cache. In general, ray-casting is performed in a single rendering pass [74] through "virtual" volume space. At each sampling location, one or multiple address translations are performed, and the sample from each modality is obtained from the corresponding cache texture. Although address translation is a relatively fast process, it can optionally be reduced by using a global cache layout for all volumes. This, however, leads to a higher number of active bricks and thus requires bigger caches, since in this case culling cannot be performed independently for each modality.

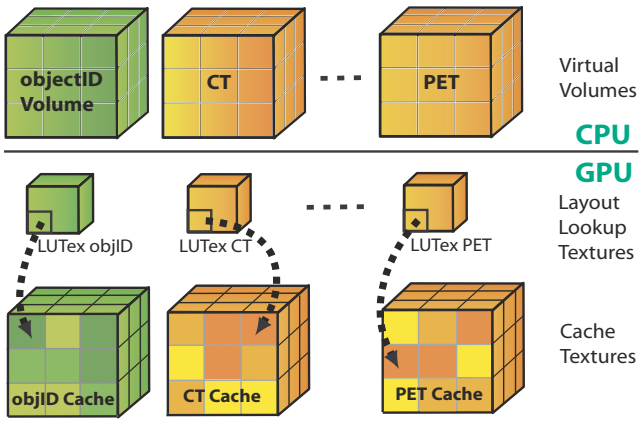


Figure 4.4: Bricked memory management for multi-volume rendering. No volume is required to be resident in GPU memory in its entirety. Ray-casting is performed in "virtual" volume space, with addresses translated to actual cache texture coordinates using layout look-up textures.

4.5.2 Skull Peeling - Surgical Approach to the Brain

For simulating the surgical approach to the brain we integrated Skull Peeling, as described in Chapter 3 into the system. Additionally, the user can now store the result of the view-dependent skull-peeling procedure as a segmentation mask. This is crucial for neurosurgical intervention planning as it enables viewing the skull peeled area from any arbitrary angle.

View-Independent Skull Peeling: The skull peeling algorithm is inherently view-dependent. This, however, implies that the peeled volume is static with respect to the image plane instead of the volume, and the part of the volume that is peeled away changes whenever the view is changed. We extend skull peeling to a powerful view-independent approach for volume clipping. During ray-casting, a depth image is generated that stores the depths where rays first hit a part of the volume that is not peeled away. In order to generate a segmentation mask that corresponds to the peeled area, each voxel position is transformed according to the view transform into an (x, y) position and corresponding depth z. Comparing the voxel's transformed depth with the depth stored in the depth image at (x, y) determines whether the voxel is included or excluded from the skull peeling segmentation mask, which is equivalent to a voxelized selection volume [92]. This process is very similar to shadow mapping [95, 20]. The generated mask allows to switch back to standard volume rendering with segmented masks, toggling the visibility of the peeled area on demand while it stays constant with respect to the volume, even when the view is changed. Therefore, view-independent skull peeling is the first step for multi-volume brain surface visualization (Section 4.5.3) as it offers an unobscured view to the brain. An example can be seen in Figure 4.5.

4.5.3 Multi-Volume Blending – Brain Surface Visualization

Combining multiple modalities in a single rendering can significantly increase the understanding of the actual clinical situation. Surgery at the brain's surface primarily includes tumor resection, epilepsy surgery, and vessel surgery (arteriovenous malformation, AVM). For these cases, after virtually removing the bone cover, the surgeon wants to see the brain surface with additional information such as DSA or functional data (i.e., PET or fMRI). For this task, we employ a visualization approach that combines different modalities without requiring segmentation masks. A major motivation for this is that PET data is very diffuse and cannot be segmented well, fMRI data is almost binary, and DSA data can be visualized clearly with a simple ramp transfer function. Also, avoiding the need for segmentation significantly speeds up the preprocessing phase. Therefore, a method that combines multiple volumes based on property fusion is a very good choice for visualizing the brain surface with information from multiple modalities.

Our algorithm only needs two or more registered volumes as input. It is flexible

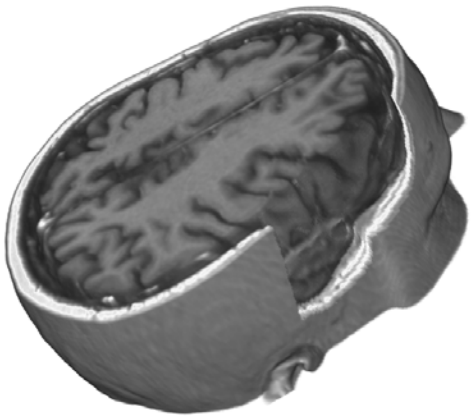


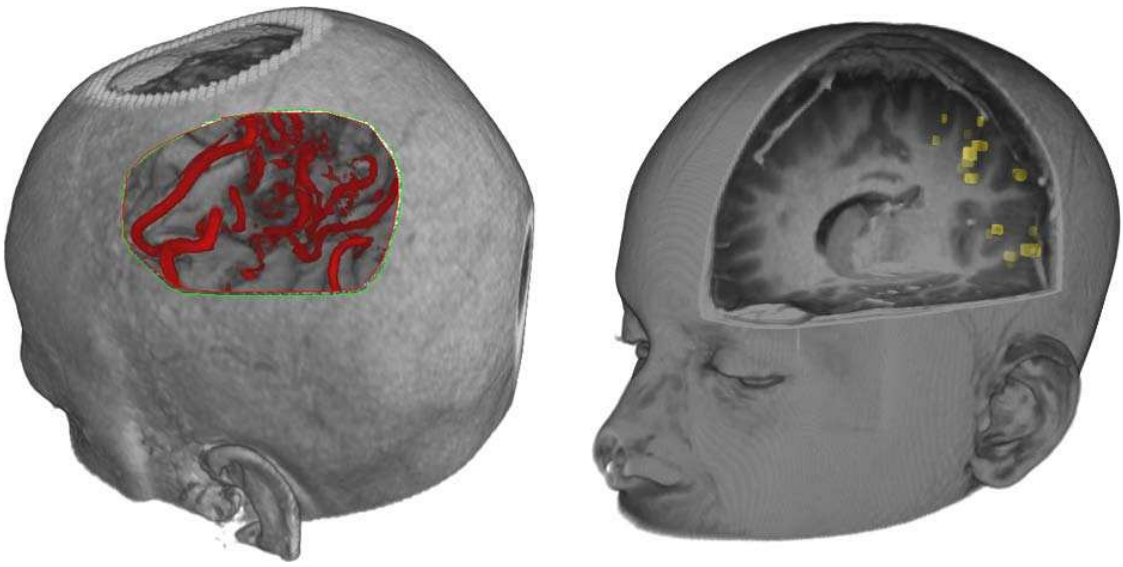
Figure 4.5: View-independent skull peeling allows to plan the surgical approach to the brain in detail, by enabling the user to position the patient's head according to the surgical approach.

with regard to the number of volumes that can be visualized concurrently, since our bricking scheme (Section 4.5.1) makes the approach scalable. Each volume has its own individual transfer function. The volume contributions are combined during ray-casting by applying the transfer functions of all volumes at each sample location and combining the classified values.

We have implemented different combination techniques ranging from simple linear blending to more specialized combination modes. For blending DSA data with MR or CT, for example, it is sufficient to only display the DSA data whenever its opacity value lies above a certain level. Otherwise, the other volumes are blended and displayed. For visualizing functional data such as PET along with anatomical data, the PET values can be used for color classification while the anatomical data determines the opacity values. Figure 4.6 shows examples of multi-volume rendering by blending.

4.5.4 Segmented Multi-Volume Rendering – Deep Lesions

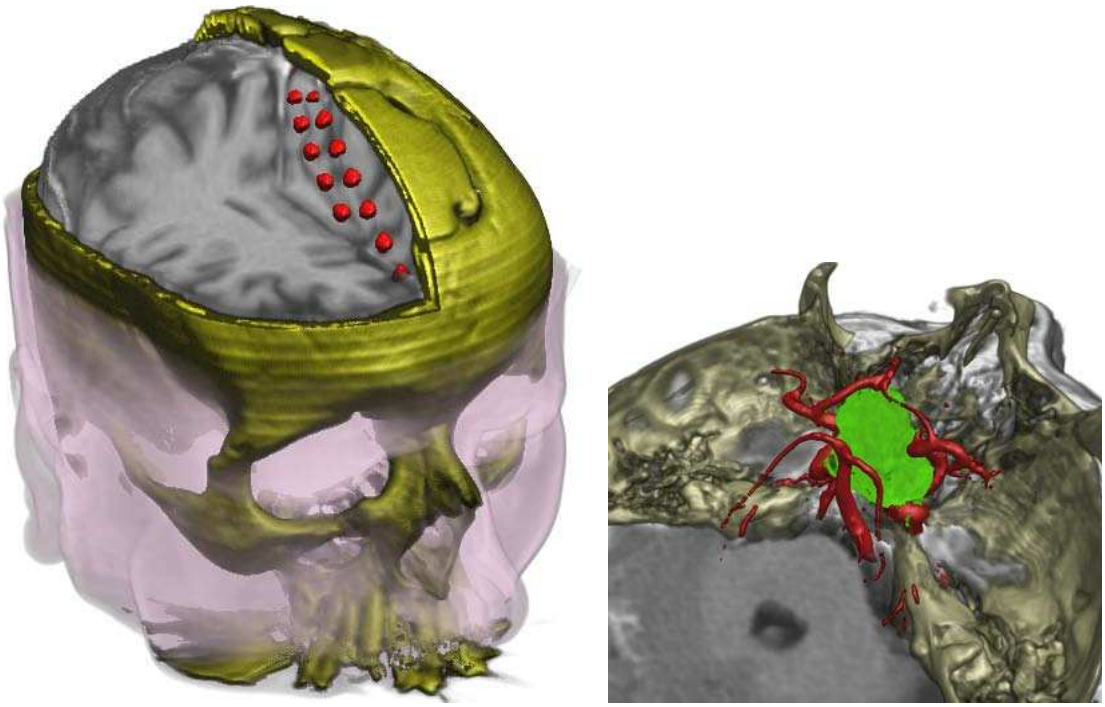
For thorough planning of surgery on deep-seated processes, after skull peeling and brain surface visualization certain structures need to be emphasized, e.g., a tumor or the optical nerve. This is achieved by a prior segmentation of the structures of interest, and individually applying multiple optical properties such as transfer functions and rendering modes. In the following, we assume that an object ID volume is available, which is the combination of multiple binary segmented objects. Our rendering algorithm for segmented volumes is conceptually based on two-level volume rendering [28], which allows to define a separate rendering mode and transfer function for each individual object. However, we use



(a) (b)

Figure 4.6: Multi-volume rendering by blending different modalities. (a) Blending MR and DSA. (b) Blending MR and fMR.

ray-casting instead of slicing, which allows for much more flexibility such as the mask smoothing described in Section 4.5.5. During rendering, the object ID of a sample determines the corresponding rendering mode and transfer function. All 1D transfer functions are stored in a single 2D texture with one transfer function per row. All 2D transfer functions are stored in a single 3D texture. For multi-volume rendering, the user additionally chooses a specific volume for each object ID in order to select the modality that depicts the underlying kind of data best. For example, choosing an MRI as underlying data of a segmented bone is not a good choice, since bone is not depicted well by this modality. Using MRI data for the "brain object," and CT data for the "bone object" surrounding the brain, however, is a very good choice. During rendering, a small 1D look-up texture is used to fetch the corresponding volume ID for each object ID. The shader then simply samples the volume texture corresponding to the volume ID of the sample. Figure 4.7 depicts examples of multi-volume rendering for segmented data, which also shows that it is possible to specify per-object clipping planes. The clipping plane equations are obtained from two 1D textures, as outlined in Section 4.5.1, and the shader simply discards fragments that should be clipped. A combination of multi-volume blending with segmented data is also possible, which is determined by special blending object IDs, as described in Section 4.5.1. In this case, each object can have as many transfer functions as there are volumes, and the result is blended per sample after all transfer functions have been applied.



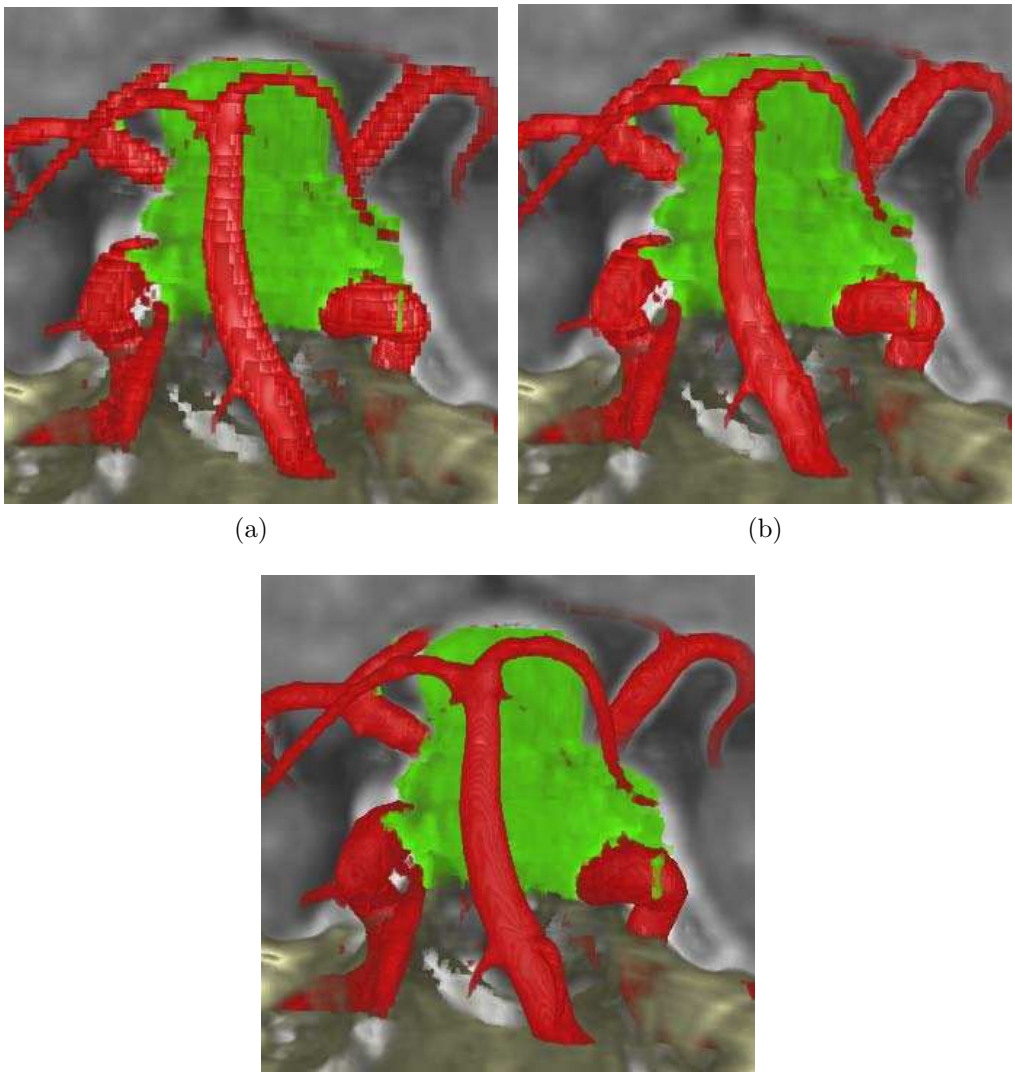
(a) (b)

Figure 4.7: Multi-volume rendering of segmented data. (a) CT and MR data for visualization of implanted electrodes for epilepsy surgery. (b) CT, MR and MRA data for tumor resection planning.

4.5.5 Smooth Rendering of Segmented Multi-Volume Data

The main visual problem of rendering binary segmented objects are the staircase artifacts that appear at object boundaries (Figure 4.8a). Especially small objects with narrow diameters such as vessels get a ragged appearance with clearly discernable object boundaries of voxel size. For high-quality visualization, the real boundary of the object is needed with subvoxel accuracy. Approaches such as trilinear filtering of object boundaries improve visual appearance (Figure 4.8b) and work for all kinds of segmented objects [28]. However, trilinear interpolation does not completely remove all artifacts (especially in close-up views), as it takes into account only the binary segmentation information and not the underlying data values. Our approach for smooth rendering of segmented data (Figure 4.8c) is based on the assumption that the object boundary can be described by an iso-value. Values above the specified iso-value belong to the object whereas values below or equal are outside. This works well for segmented vessels as well as other structures of interest in neurosurgery, such as the bone or implanted electrodes. We take advantage of this existing iso-value boundary to adjust the object ID of each sample on-the-fly during rendering.

The algorithm works as follows: First, for each object that should use improved



(a) (b)

(c)

Figure 4.8: Comparison of different volume rendering methods for displaying binary segmented objects. (a) Unfiltered object boundaries. (b) Trilinearly filtered object boundaries. (c) Our smooth boundary rendering algorithm.

smooth boundaries, the iso-value corresponding to its boundary must be specified by the user. Then, for each sample during ray-casting, we take a number of steps (defined by a user-adjustable parameter) in the positive and negative gradient direction to check if there is another object nearby. The gradient direction is used since new structures are most likely to appear in the direction of the greatest change of intensity. If a sample in the gradient direction belongs to a different object than the original sample, the original sample is treated as *boundary sam-*

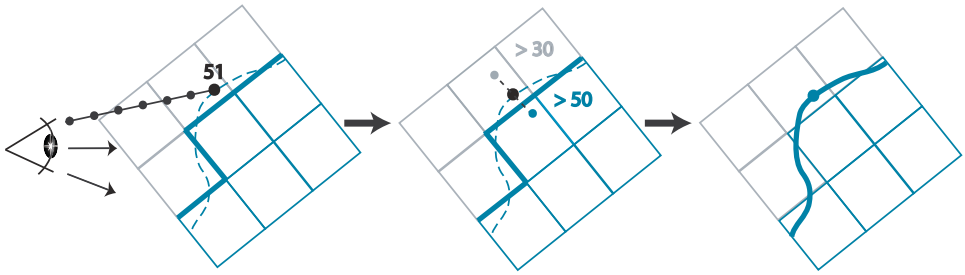


Figure 4.9: Smooth object boundary rendering. The density of a sample is compared to the iso-values of neighboring objects in gradient direction. The sample is assigned to the object with the closest iso-value.

ple, as below. If the sample is not a boundary sample, standard ray-casting for segmented objects is performed as described above. For a boundary sample, the following steps are applied: The intensity of the current boundary sample (I_{cur}) is compared to the predefined boundary iso-value of the current object (Iso_{cur}) as well as to that of the adjacent object found (Iso_{adj}). If the intensity value corresponds to the iso-value of the adjacent object, the current sample is re-classified by changing its object ID (oID_{cur}) to the adjacent object's ID (oID_{adj}). Figure 4.9 depicts the different steps of the algorithm, and Equation 4.1 summarizes the reclassification step:

$$oID_{cur} = \begin{cases} oID_{adj} & \text{if } (I_{cur} > Iso_{adj}) \wedge ((Iso_{adj} > Iso_{cur}) \vee (I_{cur} < Iso_{cur})) \\ oID_{cur} & \text{else.} \end{cases} \tag{4.1}$$

Tiede et al. [84] have presented a similar approach based on threshold-segmented objects and their corresponding min and max threshold values. Their approach, however, only takes into account the eight surrounding voxels of each sample to reassign object memberships, whereas we search a user-defined length along the gradient direction. This gives us the possibility to adapt the boundary to our needs. We can, for example, increase the boundary iso-value of a segmented vessel on-the-fly to show only the interior of the vessel, or lower the iso-value to display the vessel and its vascular hull. The main advantage of our algorithm is that even inexact segmentation masks (e.g., slightly too small or too large masks) can be rendered correctly and with a smooth appearance because the object boundary is adapted to the actual underlying data. When extending this algorithm to multiple volumes one has to be careful to always use the correct volume for iso-value comparison. When comparing the current sample's intensity value to the adjacent object's iso-value, the adjacent object's ID has to be used to fetch the correct volume for getting the intensity at the current sample. Figure 4.8 shows the visual result of our algorithm compared to standard rendering of segmented masks.

4.5.6 Interaction Aides

Various features have been implemented to support the surgeon in the task of preoperative planning:

Transfer Function Specification: Colors and opacities are specified over the intensity range in a standard graphical TF editor, in which it is possible to load a set of predefined transfer functions and adapt them manually to the individual dataset.

Flythrough Navigation / Microscopic & Endoscopic View: The datasets can be explored by flythrough navigation. To simulate the operating microscope the viewpoint for rendering can be set inside the volume and moved around interactively. An endoscopic lens with an adjustable field of view can be simulated by perspective ray-casting.

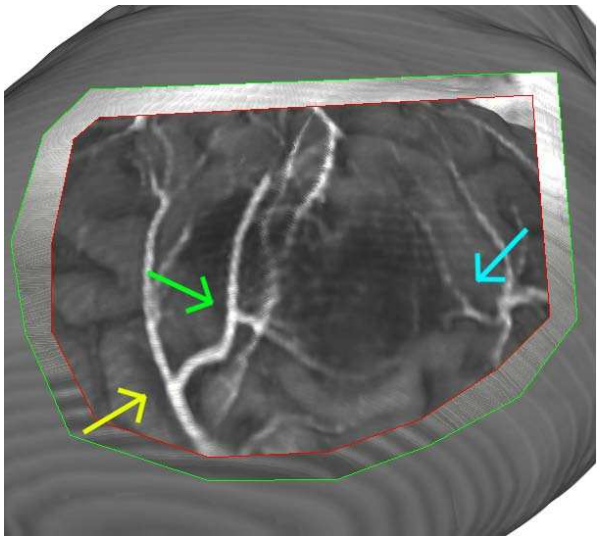
Slice Views: Next to the 3D visualization window an MPR (multiplanar reconstruction) can be displayed. The MPR consists of three orthogonal slice views (axis aligned) displaying the raw data as well as the segmented objects.

Integration into a PACS: The whole framework is integrated into Agfa's Impax 6.0 medical workstation. Impax 6.0 offers a plugin interface which allows to extend the basic functionality of the workstation to meet the individual demands of the users. Therefore, by integrating our visualization framework, all other features of the medical workstation (e.g., additional segmentation possibilities, data access) can be used in combination with our neurosurgical planning application.

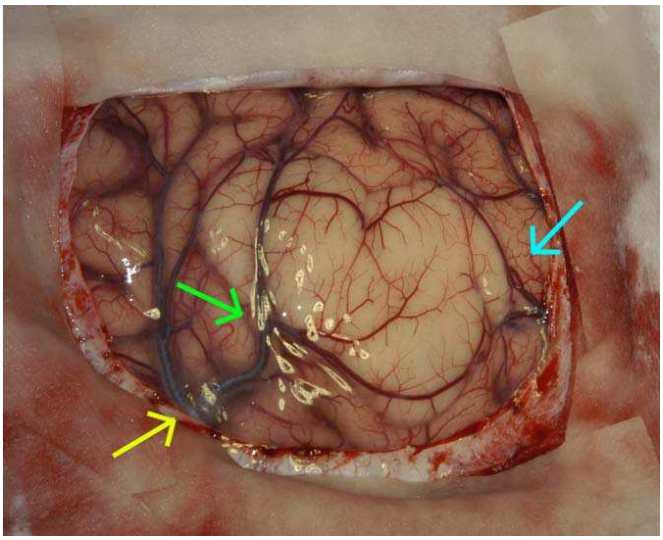
4.6 Results and Evaluation

We demonstrate the usefulness of our application by presenting two distinct planning cases as they were performed by a neurosurgeon. The cases consist of a tumor resection at the frontal lobe near the brain's surface and a tumor resection near the pituitary gland. The first patient underwent CT, MR, PET and fMRI scans, as the tumor was in the vicinity of the motor language area. First the datasets were registered and resampled to have the same volume dimensions (256^3). After initial exploration of the datasets (e.g., via skull peeling) and positioning of the patient's head as done in real surgery, skin incision and bone removal were performed tailored to the individual anatomy. Next, the datasets (MR, PET, fMRI) were visualized by multi-volume blending where the MR data depicts the anatomy, PET shows the metabolic active parts of the tumor and the fMRI data shows the brain areas involved in language function, which must be kept intact during surgery. Screenshots from different stages of the planning

process are depicted in Figure 4.12. The virtual anatomic structures such as gyri, sulci and blood vessels were found to correlate well with the intraoperative view (Figure 4.10), thus allowing the surgeon to preoperatively plan the resection borders. Concurrent visualization of fMR data helped in identifying critical "no-touch" areas at the left resection border. PET data revealed a focus of high metabolic activity in the right part of the tumor where consequently a separate specimen was taken and sent to histology during surgery, leading to additional irradiation treatment.



(a)



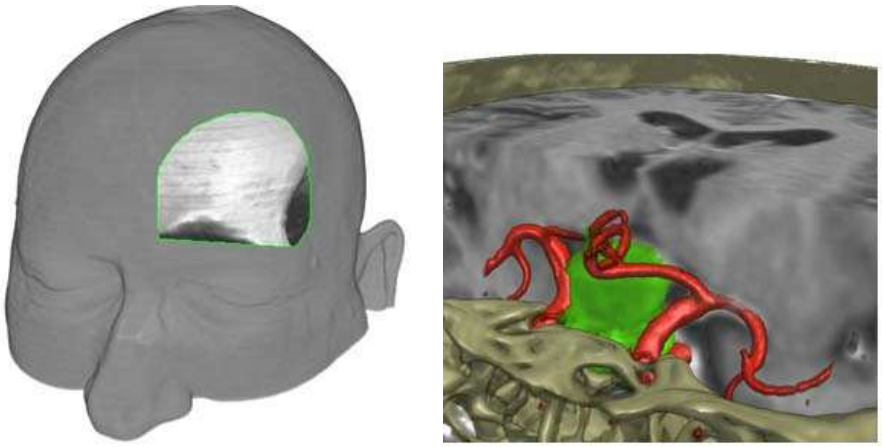
(b)

Figure 4.10: Comparison of a skull peeled image (a) with the corresponding intraoperative view (b). Arrows show points of correspondence.

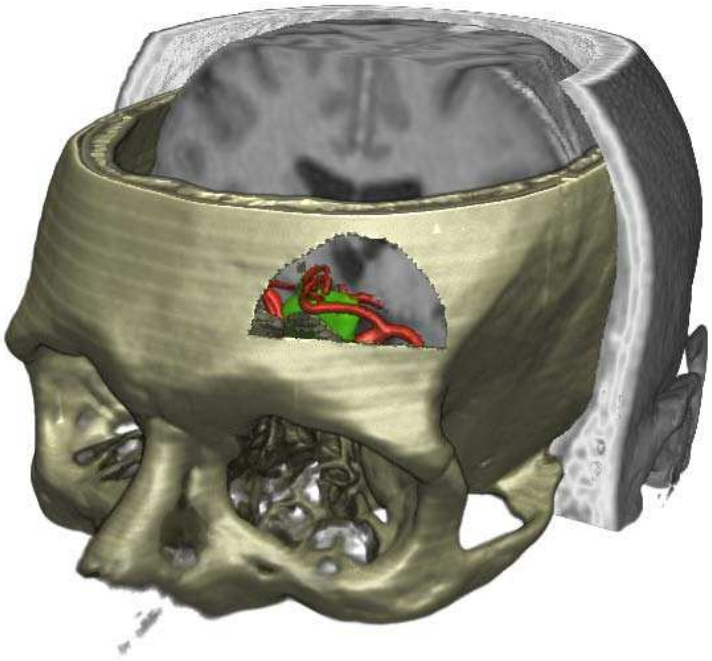
Preprocessing: (avg 10 min for 3 volumes w/o manual segmentation)	
Collection of image data from interdisciplinary PACS network (radiology, nuclear medicine, neurology)	2-10 min
Image registration	2 min / volume
Threshold segmentation for bone/vessels	1 min / object
Manual segmentation of tumor	> 5 min
Volume rendering initialization	1 min
Initial setup at interactive stage: (avg 2 min for 3 volumes and 2 objects)	
Skull peeling	immediate
Multi-volume blending (TF and blending factor design)	15 sec / volume
Multi-volume rendering of segmented masks (setting of volume, transfer function and render mode)	15 sec / object
Smooth mask rendering (iso-value adjustment)	15 sec / object
Interactive exploration	optional
Loading setup of archived case	< 10 sec

Table 4.3: User effort for initial case setup (parameter setting).

The second patient had a deep-seated lesion near the pituitary gland and underwent CT, MR and MR angiography scans (dataset size $512 \times 512 \times 164$). In this case, after registration, tumor and vessels were segmented by thresholding. Next, the surgeon used our multi-volume rendering for segmented masks to gain insight into the individual anatomy (i.e., position of critical vessels in relation to the tumor). After he had a clear perception of the location of the tumor and other anatomical landmarks he then used the skull peeling algorithm to plan the optimal position of the surgical approach (Figure 4.11). During development we kept a tight feedback loop with the department of neurosurgery at the General Hospital Vienna to iteratively refine the system. The skull peeling algorithm was received very well as it offered a direct view of the brain's surface instantly, without tedious preprocessing. The main drawback is the need of a registered CT dataset which might not always be available. Multi-volume rendering by blending also convinced because of its instant visualization without requiring segmentation and its ease of use. However, the optimal method to blend/combine the different volumes (e.g., linear blending, taking the first volume to define opacity and a second to define color) depends strongly on the type of datasets that are visualized. Therefore, an automatic setting of the combination method depending on the types of datasets could further improve the usability of the entire system. The multi-volume rendering of segmented masks was again perceived as very helpful by the surgeon. A drawback, however, is the amount of parameters that need to be set for each mask individually (assigning a volume to the mask, choosing the render mode and transfer function, setting the iso-value for smooth object bound-



(a) (b)



(c)

Figure 4.11: Planning of a right subfrontal approach for pituitary tumor resection. (a) Skin incision. (b) Operating microscope view. (c) Keyhole approach planning.

aries). Naturally, these parameters offer a very high flexibility for visualizing the datasets, however they also reduce the ease of use of the application. According to the surgeon, the most tedious part of the workflow consists of collecting and registering all the different datasets prior to visualization, especially fMR, PET and DSA data. All issues considered, our surgery planning application was very well perceived and is now used almost daily in clinical practice.

Additional User Effort: In the routine clinical setting, preprocessing has been shown to take an average of 10 minutes (Table 4.3), and initial case setup for visualization up to 3 minutes. After initial exploration, all parameters of a case can be saved in an archived casefile. Case exploration is interactive and re-loading of archived cases takes less than 1 minute. The 3D cases are routinely prepared by young residents and later demonstrated to and discussed with the performing neurosurgeon: For the resident, this has the advantage of teaching and training neuroanatomy of the oncoming surgical approach, for the advanced surgeon time expenditure is thus very small. The cases are chosen by the surgeons either on the basis of anatomical difficulty, individual variations in anatomy, or simply for the convenience of a preoperative 3D visualization.

Visualization Method	Case Study 1 CT, MR, fMR, PET (each $256 \times 256 \times 256$)		Case Study 2 CT, MR, MRA (each $512 \times 512 \times 164$)	
# volumes	2	3	2	3
Skull Peeling	9.5 fps	9 fps	5 fps	4.5 fps
Multi-Volume (MV) Blending	23 fps	12 fps	18 fps	12 fps
Segmented Multi-Volumes	35 fps	27 fps	24 fps	20 fps
Smooth Segmented MVs	15 fps	5 fps	6 fps	4 fps

Table 4.4: Frame rates of the different visualization methods for two case studies. (viewport 512×512).

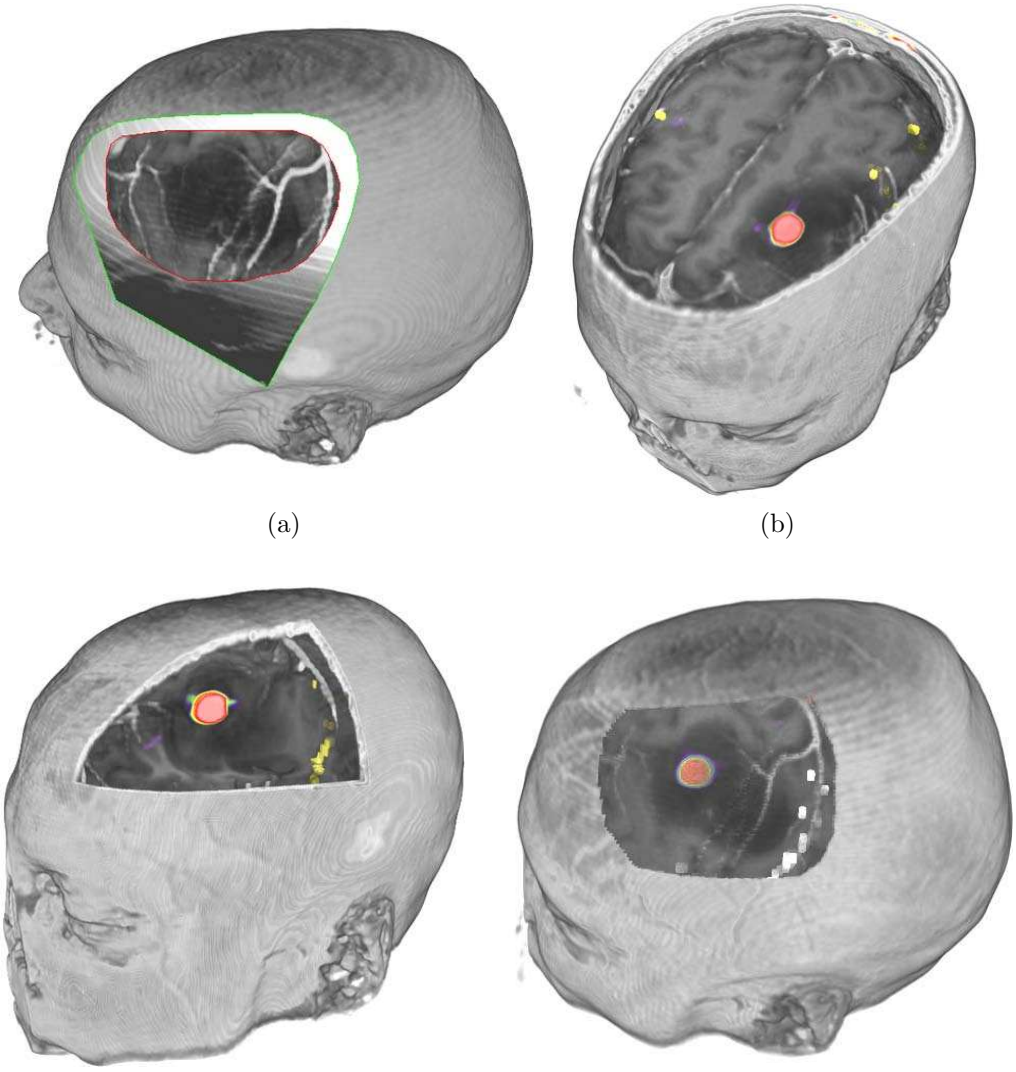
Performance: All our visualization algorithms run at interactive frame rates. Naturally, the frame rates vary depending on the transfer functions and render modes that are used (e.g., unshaded DVR, shaded DVR). Table 4.4 gives an overview of the frame rates of both case studies for the different visualization methods. Timings are for a Pentium 4, 3.2 GHz with 3 GB RAM and an ATI Radeon X1800 graphics card.

Multi-volume rendering by either blending or segmented masks achieves the highest frame rates. After activating our algorithm for rendering smooth object boundaries, however, the frame rate drops significantly due to the complex shader for rendering smooth boundaries. This problem can be alleviated by automatically switching back to normal rendering during user interaction (e.g., while rotating the view). The frame rates of the skull peeling algorithm can be explained by the costly branching-statements that have to be performed in the shader to cover all special cases for peeling the skull correctly. On the whole, the frame rates were found to be adequate by the users, since minor viewpoint changes are usually sufficient during preoperative planning.

Memory Usage: The actual memory footprint of our system depends on the cache sizes chosen, and on whether texture dimensions need to be padded to power-of-two dimensions or not. For example, Case Study 2 (Table 4.4) can use caches of size $512^2 \times 128$ each for the CT and MR volumes and of size $512^2 \times 64$ for the MRA volume (all 16-bit voxels), and a $512^2 \times 256$ cache for the object IDs (8-bit voxels). This yields a memory consumption of 224MB, which would allow this configuration to be rendered even on a 256MB graphics card. In comparison, the original volumes sum up to 287MB, but if padding to power-of-two dimensions is required (e.g., Radeon cards) would actually consume 448MB of GPU memory, which requires at least a 512MB graphics card. However, it is important to note that our caching scheme works better with larger volumes (e.g., $512^2 \times 300$ and upward), and especially helps to alleviate power-of-two requirements.

4.7 Summary and Conclusion

Our multi-volume rendering system for preoperative planning of neurosurgical interventions was directly inspired by the needs of neurosurgeons to visualize multimodal data fast, in high quality, and with as little user interaction as possible. The surgical approach to the brain is simulated by interactively removing surrounding tissue such as skin and bone from MR data by making use of additional information present in a registered CT dataset. Further we developed multi-volume rendering techniques that work either purely on the data or include additional segmentation masks. Rendering of segmented objects was improved by an algorithm for smooth rendering of object boundaries. To encourage use in daily clinical practice, we integrated our multi-volume visualization system into a medical workstation which offers registration, segmentation and interactive data exploration possibilities. As 3D visualization has become well accepted among neurosurgeons, the next logical step would be to connect our system to a neuronavigation system for tracking of the intraoperative position of the surgeon's instruments. Visualizing the multimodal datasets in parallel to the real surgery could further help the surgeon in identifying structures of interest which are not visible during surgery (e.g., functional areas, optimal skin incision line).



(a) (b)

(c) (d)

Figure 4.12: Workflow for planning a tumor resection near the brain's surface. (a) Planning the surgical approach. (b,c,d) Multi-volume blending for visualization of superficial structures. The visualization includes MR (black/white), PET (red) and fMR (yellow and white) data. The PET transfer function shows an area of high metabolic activity within a low grade glioma. The fMR spots delineate areas activated during speech.

Volume Rendering of Large Complex Biological Data

Chapter 5

Introduction

In recent years, new imaging and scanning technology have lead to the acquisition of tremendously high resolution imaging datasets. High-resolution EM (Electron Microscopy) images support neuroscientists in the reconstruction of the neural circuitry of the nervous system to get a detailed "wiring diagram" of the brain. This new area of bioscience research, called Connectomics, heavily relies on visual inspection of high-resolution large-scale imaging data.

The enormous size of these datasets pose several challenging problems for 3D visualization. Many traditional visualization techniques are not feasible for large data and might result in severe bottlenecks in data storage, memory, transfer bandwidth and processing. Therefore, new scalable techniques for visualization are needed that are designed specifically for handling of large data. This includes, for example, the need for on-demand computation of derived data instead of pre-calculation and storage of the calculated data. These new scalable techniques, however, often come at additional computational costs. Therefore, optimization methods are required, that optimize and reduce the required computational cost, memory or bandwidth.

Different methods have been proposed for volume rendering of large data, based mainly on multi-resolution techniques, hardware-assisted volume rendering and distributed rendering systems on CPU or GPU clusters.

This chapter starts with an introduction to the field of Connectomics, Electron Microscopy and brain anatomy in Section 5.1. Fundamental work in the area of large data volume rendering is reviewed in Section 5.2. More information on volume rendering of large data can be found in the textbook *Real-Time Volume Graphics* by Engel et al. [18].

5.1 Connectomics

Connectomics [79] is an active research area in the field of bioscience, which aims to develop methods, from data acquisition to analysis, to reconstruct the detailed neural circuitry of the nervous system. Connectomics relies heavily on recent advances in EM scanning technology as well as methods for handling enormous amounts of data for analyzing, segmenting and visualizing neural connections.

The goal of connectomics (i.e., to determine the connectivity and structure of the nervous system) is based on the realization that the brain's connectivity is intricately linked to the brain's function. Therefore, the connectome (i.e., the complete description of the structural connectivity of an organism's nervous system) will enable scientists to come closer to an understanding of how the brain works.

5.1.1 Brain Anatomy

The microanatomy of the nervous system consists primarily of a huge number of interconnected cells (i.e., neurons). *Neurons* are electrically excitable cells that process and transmit information by electrochemical signaling. They are supported by glial cells, which are non-neuronal cells, responsible for providing nutrition, form myelin and participate in signal transmition in the nervous system. Neurons consist of several parts: The central part, which also contains the nucleus is called *soma*. *Dendrites* are heavily branching cellular extensions of neurons. The dendritic tree of a neuron is where the majority of the input to the neuron occurs. The *axon* is a very long and fine extension of the neuron, and is responsible for carrying nerve signals away from the soma. *Synapses* form connections to target neurons and are responsible for transmitting information to them, using neurotransmitter chemicals. A schematic view of a neuron is depicted in Figure 5.1.

The adult human brain consists of approximately 100 billion neurons and 250 trillion neural connections. One cubic millimeter of cerebral cortex contains an estimated number of 50,000 neurons with 300 million interconnctions. In addition to this extremely complex structure, neurons and their interconnections are highly specific. There are many different kinds of neurons, varying in their shape, neurochemistry and function. Furthermore, the size of neuronal cells can be very small, such as dendritic spines of roughly 50nm in diameter. Typical optical microscopes can only attain resolutions of about 200 nanometers per pixel. Therefore, it is necessary to use high-resolution EM technologies for reconstructing the connectome.

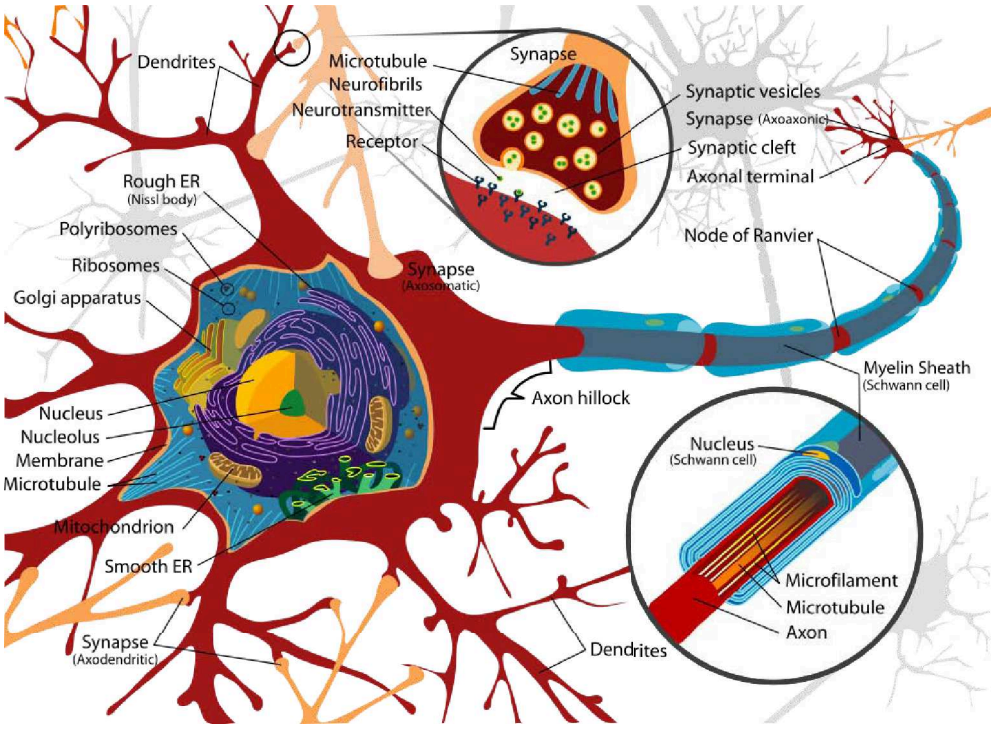


Figure 5.1: Schematic view of a neuron, displaying it's long elongated structure for transmitting electrical signals to other neurons. (Image from Wikipedia.)

5.1.2 Electron Microscopy Data

Electron microscopy uses a particle beam of electrons to create highly-magnified images. Modern EM technologies are able to attain resolutions of up to 3-5 nanometers, with a slice thickness of 30 nanometers. The increased magnification factor of electron microscopes compared to light microscopes is due to the fact that the wavelength of an electron is much smaller than the wavelength of a photon of visible light. A high-resolution EM dataset is depicted in Figure 5.2.

There are several different kinds of electron microscopes:

Transmission Electron Microscope (TEM)

TEM uses an electron gun to fire an electron beam through the specimen, as seen in Figure 5.3. Next, the electron beam, carrying the information about the structure of the specimen, gets magnified by an objective lense and is projected onto a fluorescent viewing screen, where it can be photographed.

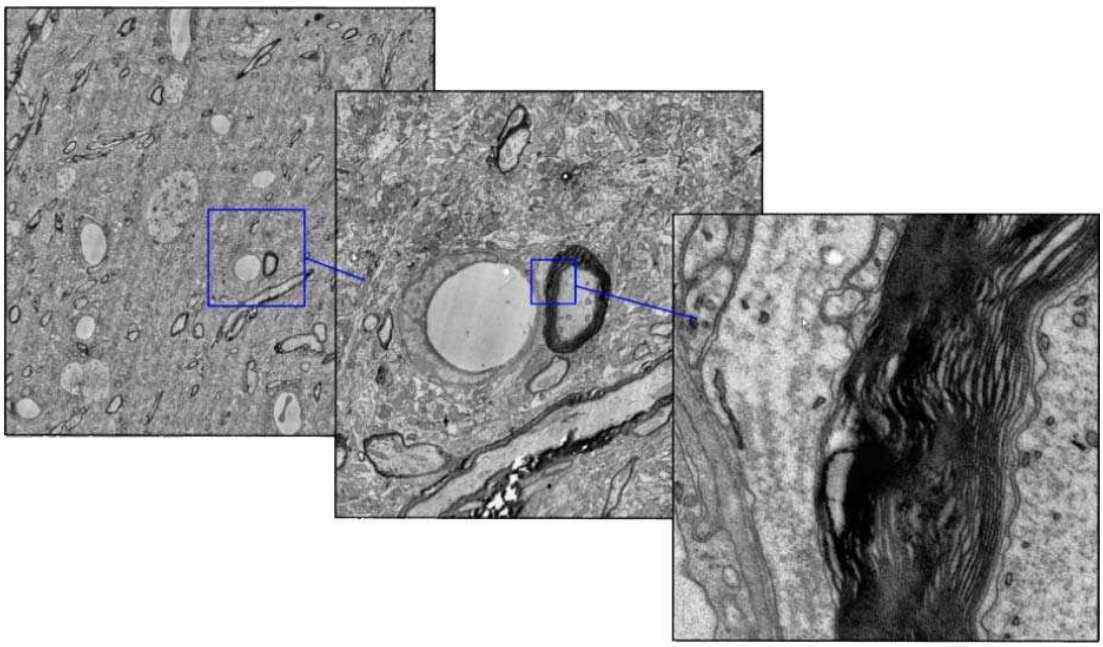


Figure 5.2: High-resolution EM dataset of a mouse brain. The dense and complex structure of the data poses significant challenges for segmentation and visualization algorithms. Image courtesy of Hanspeter Pfister, Harvard University.

Scanning Electron Microscope (SEM)

SEM, on the other hand, is based on raster scanning a specimen with a focused electron beam across a rectangular area. When the electron beam hits the specimen, some electrons and electromagnetic radiation gets reflected and can subsequently be captured to create the final image. Even though the spatial resolution of SEM is poorer than TEM, it is able to create image tiles of several centimetres in size and has a greater depth of view. An example of an automated SEM is Harvard's ATLUM, depicted in Figure 5.4, which can produce up to 11 gigabytes of data per second.

Due to the high-resolution of EM imaging, a cube of brain tissue of one millimeter length will result in up to one petabyte of raw data. This tremendous amout of data poses additional challenges of efficiently storing, processing and retrieving the data. New image acquisition pipelines need to be developed, which are able to deal with streams of data to maximize throughput.

5.1.3 Processing Pipeline

In connectomics, to be able to visualize neural processes in a large single volume from the acquired large-scale EM data, a multitude of pre-processing tasks need to be performed [9]. The first step consists of the acquistion of individual image

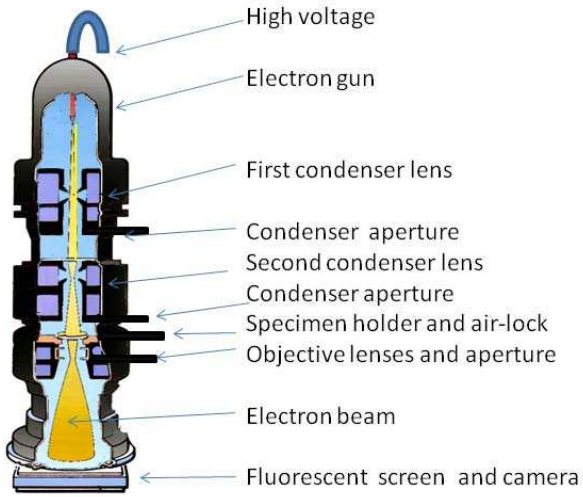


Figure 5.3: Schematic view of a Transmission Electron Microscope. (Image from Wikipedia.)

tiles (e.g., by the ATLUM microscope). Individual tiles usually have a resolution between 1000×1000 and $100,000 \times 100,000$ and need to be aligned and stitched together to form one single image [15]. To create the final 3D volume, the individual slices also have to be registered along the z-direction [10, 50]. To improve image quality, usually there are also some image processing steps (e.g., de-noising, histogram-equalization) incorporated into the pipeline. Finally, the neural processes can be reconstructed [16] using volume segmentation, labeling, axon tracking [49, 35] and interactive visualization methods.

5.2 Large Data Volume Rendering

Due to the ever increasing size of volumetric datasets, volume rendering of large-scale data has been an active area of research for several years. High-resolution volumetric datasets are now common in fields such as medicine and biology as well as in material sciences, geoscience and archeology. However, traditional techniques for volume rendering are often not feasible for large-scale datasets. Nowaydays, volumes can range from several mega- or gigabytes up to the petascale (10^{15}) range. This huge amount of data can cause severe bottlenecks in data transfer bandwidth, memory and processing. In addition, the still common 32-bit CPUs can only address up to 4 gigabytes of memory, requiring to switch to 64-bit CPUs for addressing a larger amount of memory.

Several different approaches have been developed for volume-rendering of large-scale data, mainly building on multi-resolution, data compression and packing techniques.

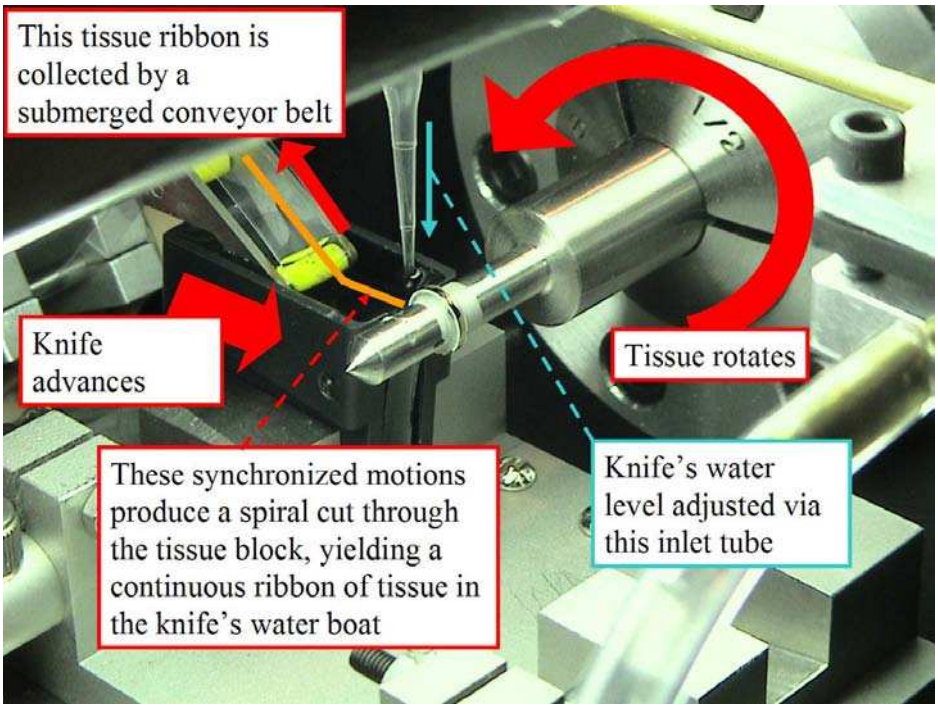


Figure 5.4: ATLUM - Automatic serial sectioning for large volume nanoscale imaging. A diamond knife is used to cut a solidified tissue sample into very thin slices. Subsequently, the resulting tissue ribbon is imaged to acquire a high-resolution EM dataset. Image courtesy of Harvard University, Center for Brain Science.

5.2.1 Multi-Resolution Techniques

Multi-resolution techniques for volume rendering try to reduce bandwidth requirements and memory storage on the GPU by adapting and reducing the resolution of the volume that is being rendered based on LOD (level of detail) selection schemes.

Bricking is the process of splitting up a single volume into several individual volume bricks. Usually, each brick is rendered separately and composited together in the final step of the rendering pass [43]. Another approach is to use a bricked volume layout together with texture packing to be able to render the entire volume in a single pass [74]. Bricking by itself does not reduce overall memory consumption, rather it is usually necessary to store an additional brick boundary per brick, to circumvent artifacts during rendering. However, it can reduce memory consumption on the GPU by downloading only currently active bricks (i.e., the current working set of bricks) to GPU memory.

Octrees are one of the most prominent examples for a multi-resolution volume hierarchy and have been used extensively for volume rendering [7, 27, 43, 91]. Octrees store the high-resolution bricks in their leaves, while inner nodes store

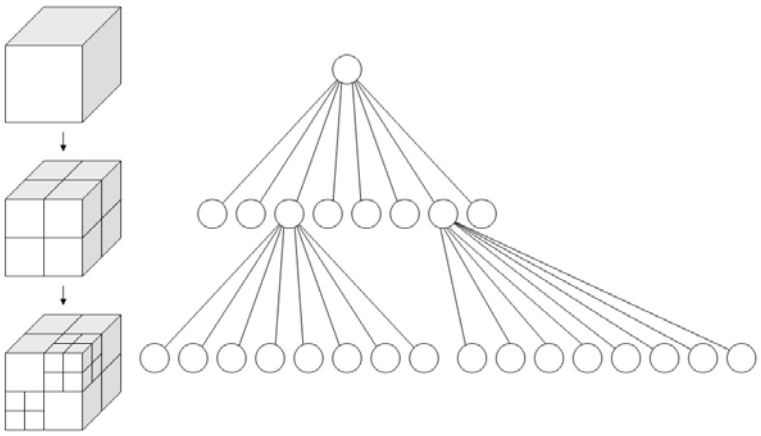


Figure 5.5: Multi-resolution octree. The leaf nodes of the bottom level of the octree store bricks in the original resolution of the volume, while nodes in upper levels store lower resolution bricks. (Image from Wikipedia.)

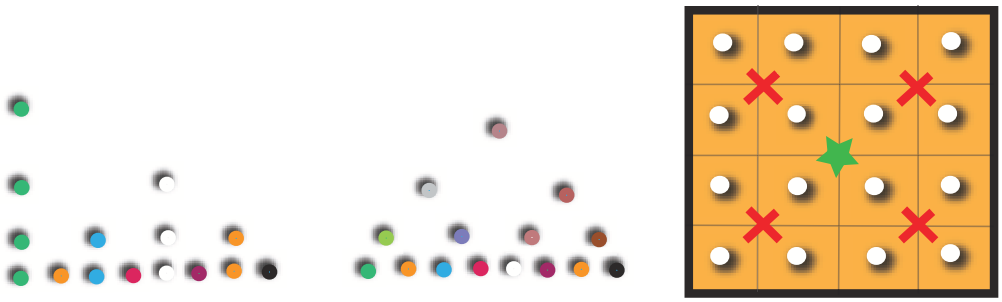
the down-sampled volume at an eigth of the resolution of their eight child nodes. Figure 5.5 displays the tree structure of an octree.

During rendering, a *Level-of-detail (LOD) scheme* is used to determine the resolution of the individual bricks that are being rendered. LOD selection algorithms are usually either based on the current viewpoint or eye/camera distance, a region of interest, the data error, or the transfer function [46]. In the majority of cases a combination of the above methods is used for LOD selection.

For creating down-sampled versions of the high-resolution data, most multi-resolution volume rendering techniques use simple subsampling at the original sample positions of the data to create the data hierarchy [43, 91]. Other approaches set the new sample position between the high-resolution sample points [47, 3] (e.g., as done for averaging). Figure 5.6 compares the different approaches.

One common problem in multi-resolution volume rendering is the appearance of artifacts at resolution boundaries. Several methods have been proposed for smooth transitions between blocks of different resolution [91, 3].

In addition to the multi-resolution hierarchy for handling the raw data, large-scale volume rendering also requires efficient utilization of the different layers in the computer's memory hierarchy. On standard PCs, different types of memory (e.g., disk, RAM, GPU memory) have different storage capacities, memory bandwidth and latencies (i.e., the time between a data read request and its delivery). Therefore, a hierarchical data management scheme is often necessary for high-performance volume rendering. Figure 5.7 displays an exemplary out-of-core data managment structure for volume rendering of large data (as implemented in *NeuroTrace*, Chapter 7).



(a) (b) (c)

Figure 5.6: (a) Multi-resolution hierarchy of a 1D signal. The bottom line depicts the original signal, the upper lines display the down-sampled signal, using subsampling. (b) Multi-resolution hierarchy of a 1D signal using averaging for down-sampling. (c) 2D example of down-sampling by averaging. White dots depict the original sample positions, red crosses represent the first down-sampled layer, and the green star depicts the second down-sampled layer.

5.2.2 Compression and Packing Techniques

Compression techniques aim at minimizing memory and bandwidth requirements of the raw data by storing the data in a compressed format that is de-compressed during rendering. Compression techniques include OpenGL's built-in texture compression as well as wavelet compression, discrete cosine transform or vector quantization.

To achieve higher compression rates than directly compressing the original data, compression is usually achieved in two steps. First a transform is applied to the original data, resulting in different transform coefficients describing the data. In the second step the resulting coefficients, or the most important subset of them, are stored in a compressed way (e.g., entropy encoding).

Wavelet compression is based on the fact that images are usually comprised of areas with high frequency and homogenous regions of low frequency. In wavelet compression, the original signal is projected onto a series of basis functions, so called wavelets. Compression is achieved by discarding wavelet coefficients of low importance. Hierarchical wavelet representations have been used for multi-resolution volume rendering [27].

Texture packing techniques try to pack the required volume blocks for rendering into GPU memory as compact as possible. An additional small lookup or indexing structure is used to translate between the virtual volume and the packed volume in the texture cache. This allows to only download those parts of the volume to the GPU that are required for the current frame, saving GPU memory. A drawback of texture packing, however, is the need for an additional

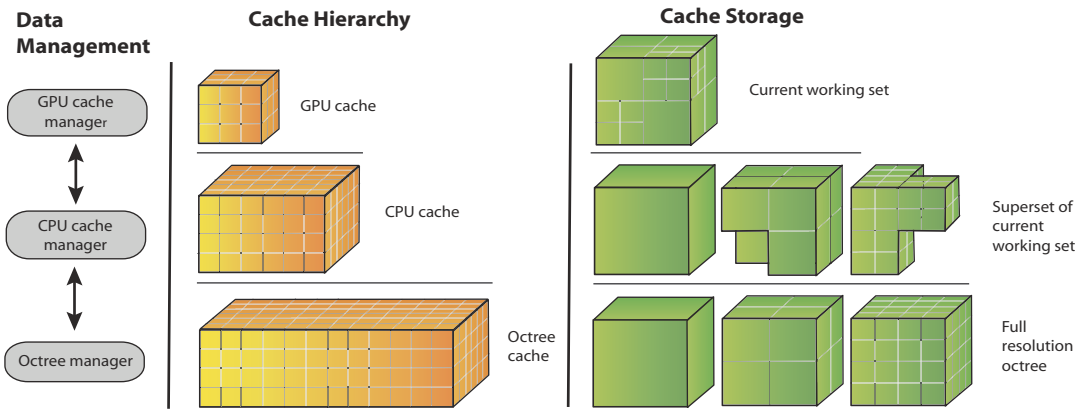


Figure 5.7: Out-of-core cache memory hierarchy. The GPU cache is the smallest cache, holding only the current working set of blocks. The CPU cache holds a super-set of the working set, while the full resolution octree resides in the octree cache on disk/network storage.

texture lookup in the indexing structure. These additional texture fetches can impose a considerable overhead and slow down the overall performance of the volume renderer. Figure 5.8 shows the memory improvement of using a texture packing scheme. In Chapter 6 a texture packing scheme for mixed-resolution volume rendering is presented in more detail.

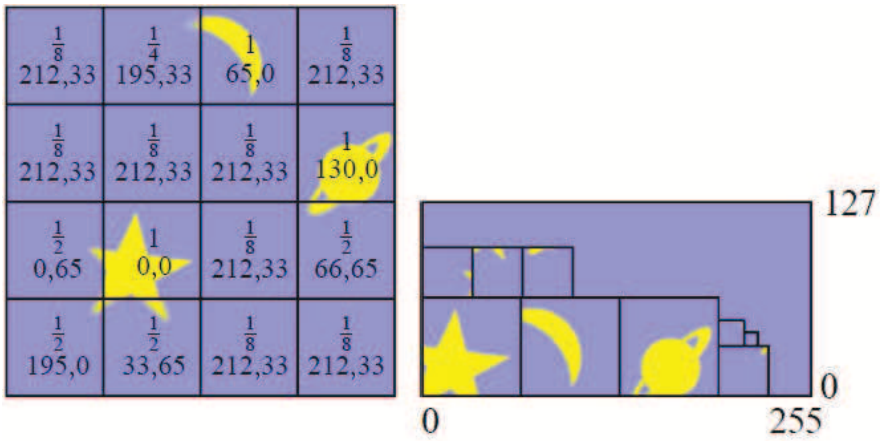


Figure 5.8: Adaptive texture maps for efficient texture packing. Left: Index texture, storing scale factors and coordinates for texture fetch. Right: Packed data texture. Image courtesy of Kraus and Ertl [40].

Chapter 6

Smooth Mixed-Resolution GPU Volume Rendering

Figure 6.1: Mixed-resolution volume rendering of a large medical dataset. High-resolution bricks are colored in green.

6.1 Introduction

Parts of this chapter are based on the paper *Smooth Mixed-Resolution GPU Volume Rendering*. IEEE International Symposium on Volume and Point-Based Graphics (VG '08) [3].

The sizes of volumetric data sets, as for example generated in medical imaging, industrial scanners, or scientific simulations, are increasing at a very rapid rate. Although volume rendering approaches exploiting graphics processing units (GPUs) achieve real-time frame rates, GPU texture memory is still limited and increasing in size more slowly than data sizes do. Therefore, many GPU volume rendering schemes incorporate multi-resolution volume bricking or data compression techniques to cope with large volumes. Multi-resolution schemes circumvent the memory constraints of GPUs by downsampling the volume or parts of it to a lower resolution. To reduce visual artifacts in the final image, level-of-detail (LOD) selection techniques can be employed to steer the selection of areas for downsampling.

Most multi-resolution schemes for GPU-based volume rendering restrict the sampling positions of the downsampled grid to a subset of the original sample positions. This, however, restricts the choice of downsampling filters and thus the attainable quality or smoothness of lower resolutions. A major reason for this restriction is that it simplifies the generation of a continuous function when different resolution levels are mixed. Many approaches use only nearest-neighbor downsampling, and moreover require higher resolution samples to be overwritten with values interpolated from lower resolution levels [91]. This implies that even in the highest resolution level many samples are not identical to the original volume and thus inaccurate.

We propose a mixed-resolution volume ray-casting approach that enables more flexibility in the choice of downsampling positions and filter kernels, allows freely mixing volume bricks of different resolutions during rendering, and does not require modifying the original sample values. Our approach can be used to render volumes larger than GPU memory with ray-casting in a single rendering pass, mixing different levels of resolution with continuous transitions between resolution levels. A C^0-continuous function is obtained everywhere with hardware-native filtering at full speed by simply warping texture coordinates of samples in transition regions. It operates on a per-sample basis and solely modifies the coordinates of texture fetches in a thin transition region between bricks of different resolutions. Thus, it is fast and requires only minor modifications to existing ray-casters, and could also be applied to renderers using texture slicing. We place the sample positions of a downsampled level half-way between samples of the higher-resolution level above it. This is the natural choice for downsampling filters that weight an even number of samples in order to compute a lower-resolution sample. In this work, we employ a $2 \times 2 \times 2$ averaging filter, but our sampling scheme facilitates higher-order filters as well (e.g., cubic splines). Additionally, we propose a simple but powerful, flat texture packing scheme that supports mixing different resolution levels in a single 3D volume cache texture with a very simple and fast address translation scheme. Although this packing constrains full scalability, it enables mixing different resolution levels in GPU-based ray-casting with only a single rendering pass. We demonstrate our approach on large real-world data,

obtaining a continuous scalar function and shading at brick boundaries, using single-pass ray-casting at real-time frame rates.

The major advantages of our approach are that: (1) bricks of different resolutions are stored in a single 3D cache texture and can be freely mixed during rendering; (2) address translation between volume space and the cache texture is extremely simple; (3) the transitions between bricks of different resolution levels are C^0-continuous, which is achieved by simply modifying the texture coordinates of chosen samples in the fragment shader; thus (4) actual interpolation can use the hardware-native tri-linear filtering at full speed; and (5) downsampling positions do not need to be aligned with the original sample positions, which allows for a wider range of filter kernels and thus more flexibility.

6.2 Related Work

Multi-resolution approaches for volume rendering try to circumvent memory restrictions of current hardware usually by breaking down a single large volume texture into several smaller ones (i.e., bricks). For low-resolution representations either the number of bricks is reduced (as in hierarchical bricking schemes) or the texture size of the bricks is reduced (as in flat bricking schemes). The breaking down of a larger texture into several smaller ones is called *bricking* and usually requires duplication of texels at brick boundaries.

LaMar et al. [43] were one of the first to introduce a hierarchical (octree) bricking scheme for hardware-assisted volume rendering. They propose a selection filter for downsampling bricks depending on their distance from the viewpoint and the view frustum, however they do not account for continuous transitions between different levels of detail during rendering. Weiler et al. [91] ensure continuous level transitions in their octree-based multi-resolution scheme by adapting the brick borders of the finer resolutions, throughout all hierarchy levels. They are, however, restricted to downsampling on the original grid points. Other hierarchical approaches including LOD selection algorithms were proposed by Boada et al. [7] and Guthe et al. [27], who use a hierarchical wavelet representation and screen-space error estimation for LOD selection. Entezari et al. [19] use Cartesian, FCC and BCC lattices for downsampling the data using different sampling densities. Their approach, however, does not support mixing of different resolution levels. A method for iso-surface reconstruction of multi-resolution volume data was proposed by Westermann et al. [94]. They create a hierarchical octree using averaging and focus on fixing cracks in the surface at transitions between different resolution levels.

A flat bricking scheme for multi-resolution data with continuous resolution transitions was proposed by Ljung et al. [47]. Their approach does not need sample replication at brick boundaries, as they perform interbrick interpolation directly during rendering. This, however, requires complex fragment shaders

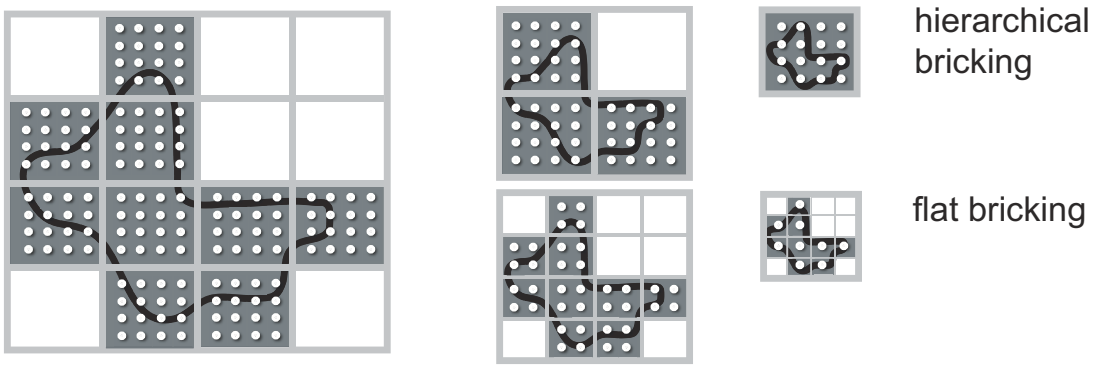


Figure 6.2: 2D example of hierarchical bricking vs. flat bricking. The resolution is reduced from left (original resolution) to right (lowest resolution). Culled bricks are marked in white.

to manually perform the correct interpolation. LOD selection is based on the transfer function [48].

Most multi-resolution approaches render each brick individually, storing them in different textures. Our work, however, is based on a scheme similar to adaptive texture maps introduced by Kraus et al. [40], where data bricks of different resolution are packed into a single texture. An additional index texture is used for address translation. In contrast to Kraus et al. [40], however, we maintain our packed data and index texture dynamically.

Our volume visualization framework builds on previous research in the area of hardware assisted volume rendering using commodity GPUs. While first approaches were based on texture slicing [93, 66], GPU ray-casting is now a viable and very powerful alternative [41].

6.3 Mixed-Resolution Volume Rendering

Most multi-resolution volume rendering methods are based on hierarchical bricking schemes where the brick size in voxels is kept constant from level to level, and the spatial extent of bricks increases from high to low resolution until a single brick covers the entire volume (Figure 6.2, top row). Conversely, flat bricking schemes (Figure 6.2, bottom row) keep the spatial extent of bricks constant and successively decrease the brick size in voxels. A major advantage of flat bricking schemes is that the culling rate is much higher, illustrated by the number of white bricks in Figure 6.2, because the granularity of culling stays constant irrespective of actual brick resolutions. This not only reduces the required texture memory, as more bricks can be culled, but also allows for a much more fine-grained LOD or shader selection per brick [45]. However, flat multi-resolution techniques have a bigger memory overhead when samples are replicated at brick boundaries,

because for decreasing brick sizes the overhead of duplicated voxels increases. This overhead can be removed by avoiding sample duplication, but only at the cost of highly increased run-time filtering complexity and cost [47]. We employ flat multi-resolution bricking with sample duplication, but reduce the run-time overhead significantly by using hardware filtering and only warping the texture coordinates of samples where necessary.

6.3.1 Volume Subdivision for Texture Packing

The original volume is subdivided into equally-sized bricks of size n^3 in a preprocess, where n is a power of two, e.g., $n = 32$. During this subdivision, the minimum and maximum value in each brick are stored for culling later at run time, and lower-resolution versions of each brick are constructed. For the latter we compute the value of the new sample at the center of eight surrounding higher-resolution samples as their average, but higher-order filters could also be used. We limit the number of resolution levels to minimize the overhead of duplicated boundary voxels, and also to allow tight packing of low-resolution bricks in the storage space reserved for high-resolution bricks (Section 6.3.2). By default we use only two resolution levels, e.g., 32^3 bricks with a downsampled resolution of 16^3. For fast texture filtering during rendering, voxels at brick boundaries are duplicated. In principle, duplication at one side suffices for this purpose [91], e.g., storing $(32 + 1)^3$ bricks. However, in the high-resolution level we duplicate at both sides, because the space for a single $(32 + 2)^3$ brick provides storage for eight $(16 + 1)^3$ bricks. Coincidentally, this often does not impose additional memory overhead. The brick cache texture (Section 6.3.2) always has power-of-two dimensions for performance reasons, and a cache of size 512^3, for example, can hold the same number of 34^3 and 33^3 bricks.

Although this approach is not fully scalable, it is very simple and a good trade-off that is not as restrictive as it might seem. Because culling is very efficient in a flat scheme, fewer bricks need to be resident in GPU memory. Even without culling, if the size of the brick cache texture is 512x512x1024 (256 mega voxels), for example, and two resolution levels are used (brick storage size 34^3), 15x15x30 bricks fit into the cache. This yields a possible data set size of about 1.7 giga voxels, e.g., 960x960x1920, if all bricks actually need to fit into the cache. Due to culling, the real data set size can typically be much larger. Additionally, for very large data three levels could be used. For example, increasing the allocated space for each brick from $(32 + 2)^3$ to $(32 + 4)^3$, both 16^3 and 8^3 bricks can be packed tightly, including boundary duplication for filtering. Using three levels with storage for $(32+4)^3$ bricks, 14x14x28 bricks would fit into the cache, yielding a data set size of 10.7 giga voxels, e.g., 1792x1792x3584, and more when bricks are culled.

6.3.2 Mixed-Resolution Texture Packing

For rendering, a list of active bricks is determined via culling, using, e.g., the transfer function or iso value, and clipping plane positions to determine non-transparent bricks that need to be resident in GPU memory. The goal is to pack all active bricks into a single 3D brick cache texture (Figure 6.3, right). In the beginning, all cache space is allocated for high-resolution bricks. If the number of active bricks after culling exceeds the allocated number, individual bricks are chosen to be represented at lower resolution. In this case, the effective number of bricks in the cache is increased by successively mapping high-resolution bricks in the cache to eight low-resolution bricks each, until the required overall number of bricks is available. This is possible because the storage allocation for bricks has been chosen in such a way that exactly eight low-resolution bricks fit into the storage space of a single high-resolution brick, including duplication of boundary voxels, as described in the previous section.

After the list of active bricks along with the corresponding resolutions has been computed, the layout of the cache texture and mapping of brick storage space in the cache to actual volume bricks can be updated accordingly, which results in an essentially arbitrary mixture of resolution levels in the cache. The actual brick data are then downloaded into their corresponding locations using, e.g., `glTexSubImage3D()`. During rendering, a small 3D layout texture is used for address translation between "virtual" volume space and "physical" cache texture coordinates (Figure 6.3, top left), which is described in the next section.

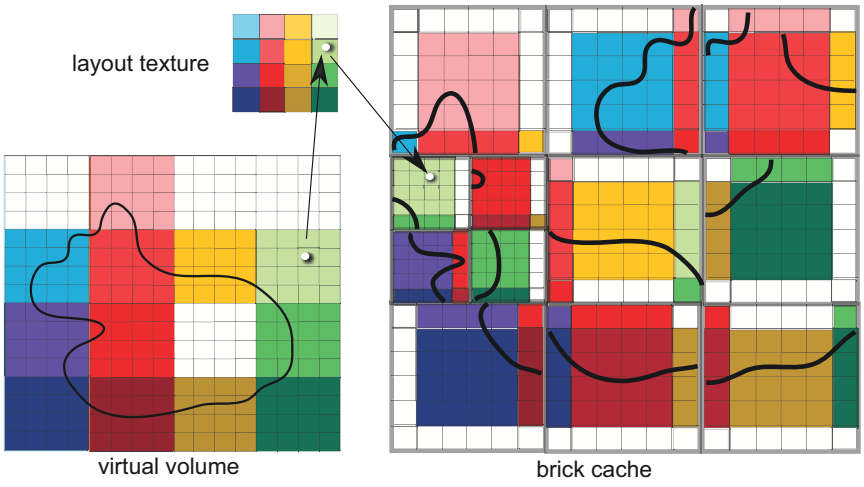


Figure 6.3: Mixed-resolution texture packing and address translation from virtual volume space to physical cache texture space via the layout texture. Resolution levels are mixed by packing low-res bricks tightly into high-res bricks.

6.3.3 Address Translation

A major advantage of our texture packing scheme is that address translation can be done in an identical manner irrespective of whether different resolution levels are mixed. Each brick in virtual volume space always has constant spatial extent and maps to exactly one brick in physical cache space. "Virtual" addresses in volume space, in $[0,1]$, corresponding to the volume's bounding box, are translated to "physical" texture coordinates in the brick cache texture, also in $[0,1]$, corresponding to the full cache texture size, via a lookup in a small 3D layout texture with one texel per brick in the volume. This layout texture encodes (x,y,z) address translation information in the RGB color channels, and a multi-resolution scale value in the α channel, respectively.

Conceptually, a volume space coordinate $\mathbf{x}_{x,y,z}$ that is located within brick $\mathbf{brick}_{virtVol}$ in virtual volume space, is translated to cache texture coordinates $\mathbf{x}'_{x,y,z}$ by first subtracting the position of the brick's origin ($\mathbf{brickPos}_{virtVol}$) from $\mathbf{x}_{x,y,z}$, yielding the offset of $\mathbf{x}_{x,y,z}$ within the brick $\mathbf{brick}_{virtVol}$. Next, this offset needs to be scaled according to the resolution of the brick in the brick cache (i.e., using the multi-resolution scale factor $\mathbf{t}_w$). Taking into account the duplicated border of a brick in the brick cache, $\mathbf{t}_w$ has to be added to get the correct offset of $\mathbf{x}'_{x,y,z}$ within its brick in the brick cache ($\mathbf{brick}_{brickCache}$). Finally, adding the position of the brick's origin in the brick cache ($\mathbf{brickPos}_{brickCache}$) results in the final coordinate $\mathbf{x}'_{x,y,z}$.

$$\mathbf{x}'_{x,y,z} = (\mathbf{x}_{x,y,z} - \mathbf{brickPos}_{virtVol}) \cdot \mathbf{t}_w + \mathbf{t}_w + \mathbf{brickPos}_{brickCache}, \tag{6.1}$$

The detailed implementation for the translation from a volume space coordinate $\mathbf{x}_{x,y,z} \in [0,1]^3$ to cache texture coordinates $\mathbf{x}'_{x,y,z} \in [0,1]^3$ in the fragment shader is as follows:

$$\mathbf{x}'_{x,y,z} = \mathbf{x}_{x,y,z} \cdot \mathbf{bscale}_{x,y,z} \cdot \mathbf{t}_w + \mathbf{t}_{x,y,z}, \tag{6.2}$$

where $\mathbf{t}_{x,y,z,w}$ is the RGBA-tuple from the layout texture corresponding to volume coordinate $\mathbf{x}_{x,y,z}$, and **bscale** is a constant fragment shader parameter containing a global scale factor for matching the different coordinate spaces of the volume and the cache. When filling the layout texture, the former is computed as:

$$\mathbf{t}_{x,y,z} = (\mathbf{b}'_{x,y,z} \cdot \mathbf{bres}'_{x,y,z} - \mathbf{o}_{x,y,z} + \mathbf{t}_w)/\mathbf{csize}_{x,y,z} \tag{6.3}$$

$$\mathbf{t}_w = 1.0, \tag{6.4}$$

for a high-resolution brick, where $\mathbf{b}'$ is the index of the brick in the cache, $\mathbf{bres}'$ is the storage resolution of the brick, e.g., 34^3, and **csize** is the cache texture size in texels to produce texture coordinates in the $[0,1]$ range. For a low-resolution brick, this is computed with $\mathbf{t}_w = 0.5$. The offset $\mathbf{o}_{x,y,z}$ is computed as:

$$\mathbf{o}_{x,y,z} = \mathbf{b}_{x,y,z} \cdot \mathbf{bres}_{x,y,z} \cdot \mathbf{t}_w, \tag{6.5}$$

where **b** is the index of the brick in the volume, and **bres** is the brick resolution in the volume, e.g., 32^3. The global scale factor **bscale** is computed as:

$$\mathbf{bscale}_{x,y,z} = \mathbf{vsize}_{x,y,z}/\mathbf{csize}_{x,y,z}, \tag{6.6}$$

where **vsize** is the size of the volume in voxels.

6.4 Smooth Mixed-Resolution Interpolation

The most fundamental operation in direct volume rendering using ray-casting or texture slicing is taking individual samples along viewing rays into the volume. This (re-)sampling requires interpolating between the discrete samples (voxels) at the grid positions. In order to avoid discontinuities, this interpolation must reconstruct a continuous function. When only a single resolution level is used, piecewise tri-linear interpolation yields a continuous function, which is extremely fast on current GPUs, where it can be performed automatically when a 3D texture is sampled in the fragment shader.

However, when multiple resolution levels (i.e., bricks of different resolutions) are mixed, obtaining a continuous function usually requires matching sample positions and modification of original sample values [91]. Figure 6.4 shows the sample positions we are using in high-resolution (yellow) and low-resolution (blue) bricks, respectively. In order to allow a sample offset between resolution levels and avoid modifying original samples, we employ the following approach for (re)sampling in the fragment shader:

- Sampling within bricks (at least 0.5 voxels away from the boundary) is performed as usual, with hardware-native tri-linear interpolation at the sample's texture coordinates.
- Sampling at the boundary between bricks of different resolution warps the texture coordinates of samples within 0.5 voxels from the brick's boundary in the brick of higher resolution. Coordinates are warped according to special interpolation primitives described in detail below.

Note that simply warping texture coordinates implies that the actual interpolation is still carried out by the hardware at full speed. Also, everything is performed on a per-sample basis, i.e., no explicit vertices, vertex attributes, or actual interpolation primitives are used. The primitives that ensure a continuous function in transition regions are only implicit, and solely determine how texture coordinates must be warped.

6.4.1 Smooth Transition Interpolation

Figure 6.4 illustrates the different cases of interpolation between bricks of different resolution in 2D. The extension to 3D is conceptually straight-forward

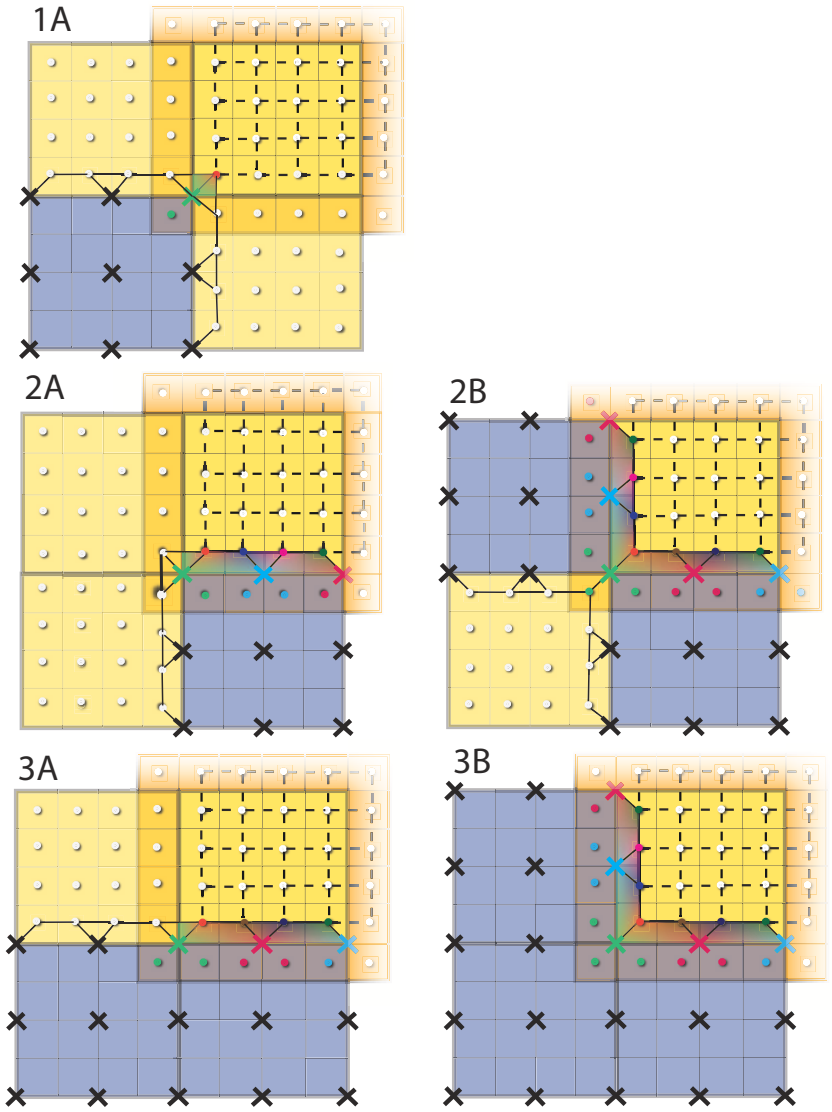


Figure 6.4: Basic configurations of mixed-resolution interpolation in 2D transition regions. High-resolution bricks are shown in yellow with white dots at the sample positions, low-resolution bricks in blue with crosses at the sample positions. The figure focuses on the top-right high-resolution brick (shown with its border of duplicated voxels in orange), and the interpolation functions in its 0.5 texel border.

and described after the 2D case below. The actual interpolation functions that have to be used in order to obtain a C^0-continuous scalar function depend on the configuration/adjacency of low-res and high-res bricks, all of which are depicted in Figure 6.4, apart from symmetry. Samples in low-res bricks are shown

as black/colored crosses in blue bricks in Figure 6.4, and high-res samples as white/colored dots in yellow bricks. When two bricks of different resolution levels are adjacent to one another, the transition region that requires warping of texture coordinates is a band of 0.5 voxels inside the brick of higher resolution. A smooth function is guaranteed by modifying the texture coordinates in this band in such a way that the hardware-native tri-linear interpolation actually carries out a smooth warping between the two sample grids.

Consider the transition region shown in Figure 6.4 (2A). We have to obtain a smooth interpolation between the high-res samples inside the high-res brick above the brick boundary (colored dots above the colored crosses), and the low-res samples on the brick boundary (colored crosses). We apply two different, smooth interpolation functions: (1) trapezoids between two low-res and two high-res samples; and (2) triangles between one low-res sample and two high-res samples. Trapezoids can be interpolated using bi-linear interpolation, and triangles with linear (barycentric) interpolation. However, for efficiency both must be mapped to hardware bi-linear interpolation inside a square of four samples. Using bi-linear texture fetches in the transition region with the warping of texture coordinates described below, the result of bi-linear texture interpolation is the same as interpolating within these trapezoids and triangles.

Figure 6.5 depicts this mapping for a single trapezoid, which can be used for any trapezoid in Figure 6.4 by using the appropriate coordinate offsets. In order to obtain the desired interpolation function with hardware-native bi-linear interpolation, coordinates within the trapezoid must be mapped to a square in such a way that the same interpolation function results. Inside a trapezoid with the coordinate extents given in Figure 6.5, the following mapping can be used:

$$u' = (u+v)/(1+2v), \tag{6.7}$$

$$v' = 2v. \tag{6.8}$$

The two top samples of the square in Figure 6.5 are original, unmodified high-res samples. In contrast, the two bottom samples of that square must be modified. However, these samples are duplicated voxels outside the high-res brick, i.e., their modification does not change original sample values. This is illustrated by the colored dots within the orange duplication border in Figure 6.4. These samples must be set to the values of the low-res grid at the bottom of the trapezoid in Figure 6.5. This modification only has to take place whenever the brick cache changes, i.e., is performed entirely independent from the volume rendering fragment shader. The details are explained in Section 6.4.3.

Figure 6.6 depicts the analogous mapping for a triangle, which can be used for any triangle in Figure 6.4 by using the appropriate coordinate offsets. Inside a triangle with the coordinate extents given in Figure 6.6, the following mapping can be used to map a triangle of height 0.5 to a triangle of height 1.0 embedded in a square such that bi-linear interpolation within that square again yields the

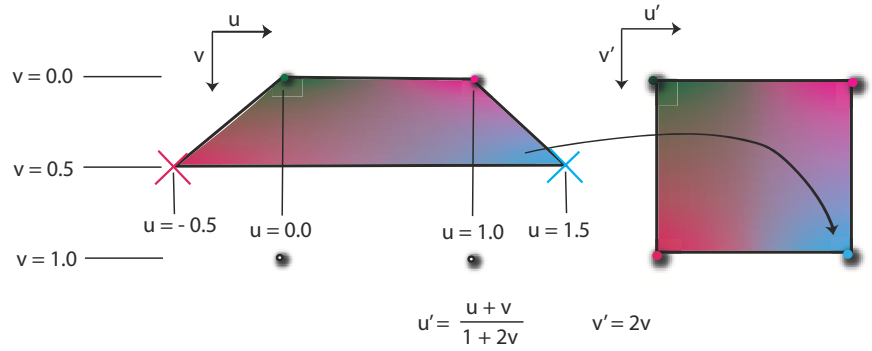


Figure 6.5: Mapping texture coordinates within a trapezoid to a square, such that hardware bi-linear interpolation in the latter yields the same result as (bi-)linear interpolation in the former.

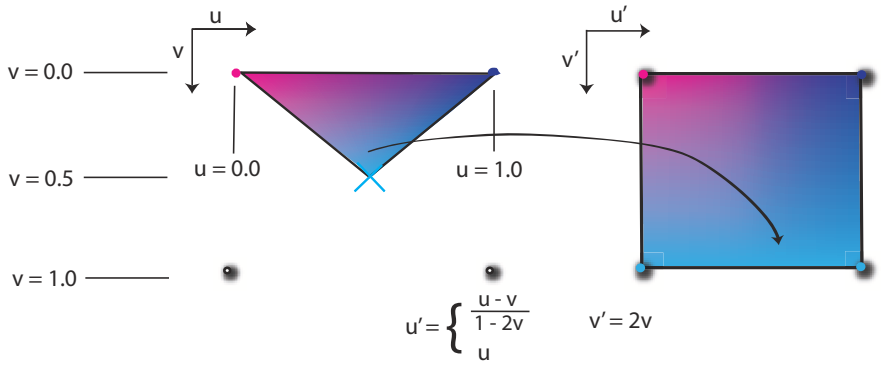


Figure 6.6: Mapping texture coordinates within a triangle to a square, such that hardware bi-linear interpolation in the latter yields the same result as (bi-)linear interpolation in the former.

desired interpolation function:

$$u' = \begin{cases} (u-v)/(1-2v) & v \neq 0.5 \\ u & v = 0.5 \end{cases} \tag{6.9}$$

$$v' = 2v. \tag{6.10}$$

Again, the two top samples of the square in Figure 6.6 are original, unmodified high-res samples, and the two bottom samples of that square result from modification of the voxels in the duplicated border. They are both set to the value at the apex of the triangle in Figure 6.6.

In order to apply the two mappings above to any trapezoid or triangle in Figure 6.4, we start with texture coordinates $(\bar{u}, \bar{v}) \in [0, 1]$, where $[0, 1]$ maps to an entire brick. We then shift the coordinates toward the center of the voxels in the high-resolution brick and map $[0, 1]$ to the size of a single voxel, instead of

the entire brick, to obtain the local coordinates (u, v):

$$u = fract(\bar{u} \cdot b_w - 0.5), \tag{6.11}$$
$$v = fract(\bar{v} \cdot b_h - 0.5), \tag{6.12}$$

where b_w and b_h are the width and height of the brick in voxels, respectively.

Extension to 3D

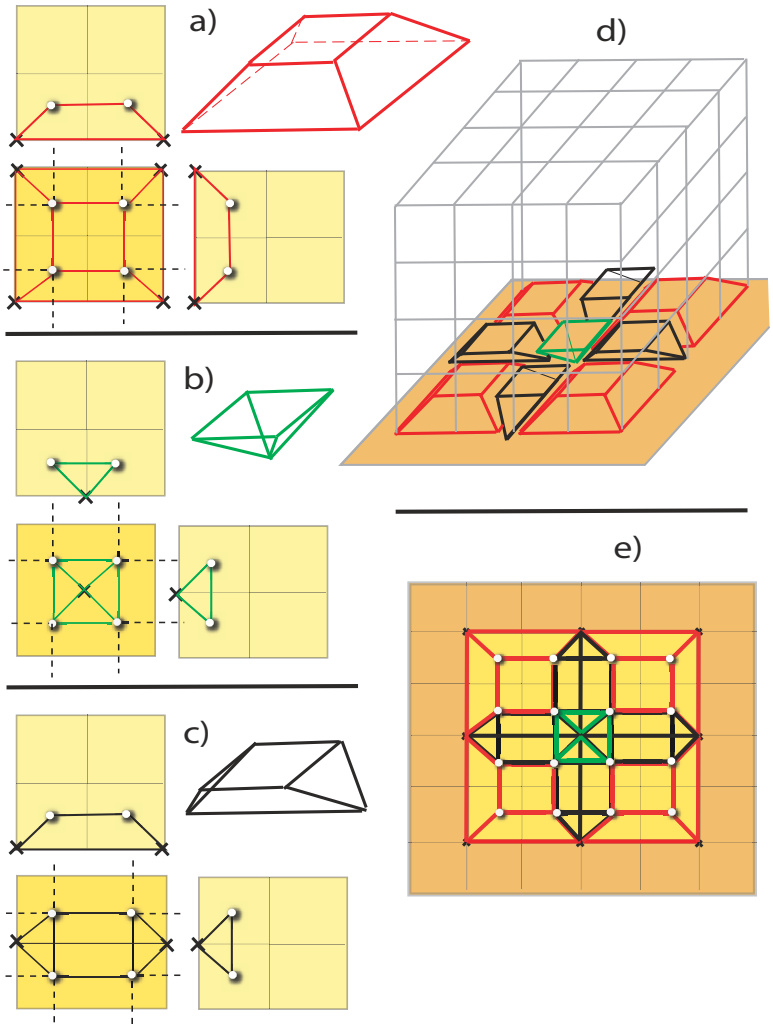


Figure 6.7: Smooth mixed-resolution interpolation in 3D. In contrast to the 2D case, in 3D three different primitives have to be used (a, b, c). However, all fragment shader computations can be performed in the 2D front and side projections shown. In 3D, the primitives fit together as shown in d, e.

Fortunately, the 3D case is almost a direct extension of the 2D case. In 3D, three different primitives must be used for interpolation, which are shown on the left-hand side of Figure 6.7 (as top, front and side view). They fit together as illustrated on the right-hand side of Figure 6.7 (as 3D and top-down view). These primitives and the interpolation within them can be computed from projections into 2D: The first primitive (Figure 6.7a) is a truncated pyramid that can be constructed from two trapezoid projections. Computing the interpolation uses Equations 6.7 and 6.8. The second primitive (Figure 6.7b) is a pyramid that can be constructed from two triangle projections. Computing the interpolation uses Equations 6.9 and 6.10. The third primitive (Figure 6.7c) can be constructed from one trapezoidal projection and one triangular projection. Computing the interpolation uses Equations 6.7, 6.8, 6.9, and 6.10. Therefore, checking the current sample's position in the 2D projections (front and side views in Figure 6.7a, b, and c, allows to unambiguously identify the primitive the current sample point belongs to.

Primitive type precedence

As can be seen in Figure 6.4 for the 2D case, special care has to be taken at the corners of bricks (i.e., corners and edges in 3D). Depending on the resolution level of the surrounding bricks different primitives have to be used for correct interpolation. Case 2A in Figure 6.4, for example, uses a triangle at the leftmost part of the lower border, because the adjacent brick on the left is in high resolution. In case 2B, however, the adjacent brick on the left is in low resolution, which results in using a trapezoid at the leftmost part of the lower border of the high-res brick. In 3D, there are even more configurations how primitives can be combined in the corner of a brick, as shown in Figure 6.8a, b, and c. In this figure, we focus on the lower left corner in the back of the brick, and assume that there is a low-resolution brick directly below the displayed brick. The figure shows the possible configurations of the different primitives to correctly interpolate samples at the lower left corner, in the back. Figure 6.8a depicts the case where all three surrounding faces of the corner are adjacent to low-res bricks, in Figure 6.8b two faces are adjacent to low-res bricks, and in Figure 6.8c only the bottom face is adjacent to a low-res brick. If a high-res brick is only adjacent to a low-res brick at an edge, there would be no face primitive (truncated pyramid) needed, but an additional edge primitive instead. If the low-res brick is only adjacent at the high-res brick's corner, additional corner primitives would be necessary. Consequently, if present, a face primitive always overrides an edge primitive, which in turn overrides a corner primitive.

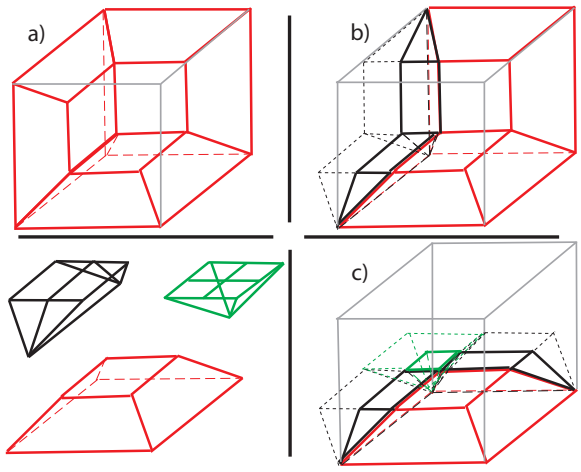


Figure 6.8: Different configurations of face, edge and corner primitives, depending on the adjacency configuration of high-res and low-res bricks. Accordingly, the lower left back corner is composed of different primitive types (a, b, c).

6.4.2 Volume Rendering Fragment Shader Modification

As described above, for smooth interpolation sampling in the fragment shader must be modified in order to warp the texture coordinates of samples in transition regions between bricks of differing resolution. This is done as follows:

1. Determine whether the sample position is in a high-res brick. If so, determine if the sample position is within the 0.5 texel border that is adjacent to a low-res brick.

2. If it is, determine the primitive type the sample position projects to via the two 2D projections (Figure 6.7) orthogonal to each relevant border. For example, for a face orthogonal to the z axis, the (x, z) and the (y, z) projections are used.

3. If the sample is contained in more than one border (e.g., at an edge), determine the primitive type that must be used according to the primitive precedence described above.

4. Warp texture coordinates in the two 2D projections according to Equations 6.7 – 6.10, and composite the results for the final 3D coordinate, e.g., (x, z) and (y, z) to (x, y, z).

In order to distinguish all possible configurations of adjacency of high-res and low-res bricks efficiently (see Figure 6.4 for the 2D cases), we create a bit state of

the 26-neighborhood of each high-res brick whenever the cache layout changes. A bit is set when a low-res brick is adjacent to a face, an edge, or a corner, respectively. For simplicity, this bit state is supplied to the fragment shader as an additional small 3D lookup texture with one texel per brick, similar to the 3D layout texture. This texture contains a 32-bit integer value for each brick, encoding the 26 neighborhood bits.

6.4.3 Brick Cache Fixup

In addition to adapting texture coordinates for sampling in the fragment shader, selected sample values must be modified in the brick cache in order to compute the smooth interpolation functions described in Section 6.4.1 between the correct source sample values. That is, whenever high-resolution bricks are adjacent to low-resolution bricks in volume space, the voxels in the duplicated border of high-resolution bricks might need to be modified in order to perform the correct interpolation. However, this modification solely depends on the layout and resolutions of bricks in the cache, and thus only needs to be performed whenever the cache changes, e.g., due to a transfer function change. It is also completely independent from the volume rendering fragment shader and therefore does not influence rendering performance.

Depending on the location of the low-resolution neighbor, we either have to adjust the face, edge, or corner of the adjacent high-resolution brick.

Figure 6.9 shows the 2D case where the left border of the high-resolution brick needs to be modified because it is adjacent to a low-resolution brick in volume space. Therefore, the duplicated border voxels of the high-resolution brick have to be set to the same value as the nearest low-resolution sample of the adjacent brick (shown by the replicated crosses in the high-resolution brick).

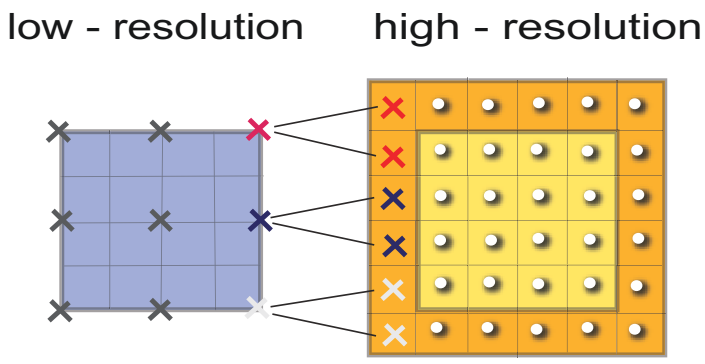


Figure 6.9: Fixup of the high-resolution brick's duplicated border to the nearest sample of its low-resolution neighbor in volume space, for smooth hardware-native interpolation.

To efficiently implement this fixup step on the GPU, an additional reverse layout texture for address translation between the brick cache and virtual volume space is necessary. Using this reverse address translation, we can look up the volume coordinates for a given brick in the fragment shader that does the fixup. Note that this is not the rendering fragment shader, but an additional shader that is only invoked once whenever the cache changes.

Whenever the brick cache is updated, all high-res bricks that are adjacent to a low-res brick are marked. In order to modify sample values in the duplicated borders where necessary, we rasterize all marked bricks slice-by-slice and perform the fixup on each slice separately, either copying a 2D slice buffer back into the 3D cache texture or directly rendering into its slices. In the shader we check for each fragment if it is located on the rasterized brick's boundary. For all boundary voxels, a reverse lookup is performed to fetch the coordinates in virtual volume space. Translating those volume coordinates back to the brick cache automatically fetches the correct neighbor of the rasterized brick. Now we only have to check if this neighbor is a low-resolution brick. If this is the case, we overwrite the current sample's value with the sample from the low-resolution brick (using nearest neighbor interpolation for the texture fetch in the low-resolution brick). This simple scheme works well for fragments on the brick's border that are positioned on the brick's face (i.e., there is only one direct neighbor). For voxels on the edges and corners we have to check all adjacent bricks and perform the texture fetch on the first low-resolution brick that we encounter. To fetch all adjacent bricks, the current sample is first transformed to virtual volume coordinates and then translated by one voxel in the virtual volume space, depending on the adjacent brick we want to fetch. Performing a lookup from volume space back to the cache brick yields again the correct adjacent brick.

6.5 Results and Evaluation

We have tested our mixed-resolution volume rendering approach on large real world data. Figure 6.10 shows highly magnified views (unshaded and shaded DVR) of a medical abdomen data set of size 512x512x1112 (16-bit voxels). Artifacts at brick boundaries of different resolution are clearly visible using standard volume rendering (Figure 6.10, upper images). Using our approach, however, a smooth and continuous function can be obtained (Figure 6.10, lower images).

Figure 6.11 shows an iso-surface rendering (first-hit ray-casting) of a high-resolution industrial CT scan of a metal ring (1518x1518x232, 16-bit voxels). The magnified views show the junction of two high-res and two low-res bricks. Again, disturbing artifacts at the brick boundaries are visible in the standard volume rendering (Figure 6.11, left) whereas our method (Figure 6.11, right) obtains a smooth function. Table 6.1 lists the frame rates of both data sets, as tested on a Core 2 Duo with 3 GHz, 4 GB RAM and an NVidia Geforce GTX 280. For

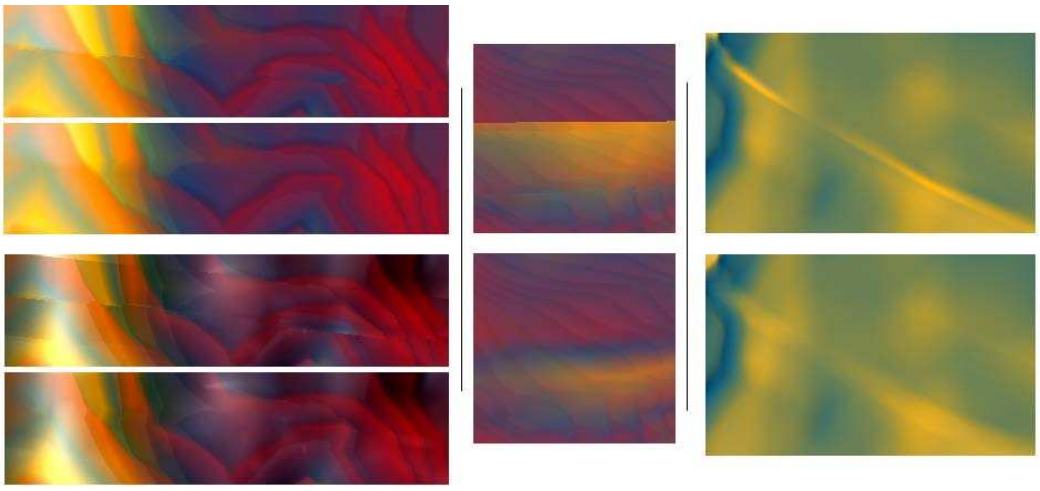


Figure 6.10: Transitions between different resolution levels in an abdomen data set (512x512x1112). The upper images show the original discontinuous transitions, whereas the respective lower images show the smooth transitions obtained by our method.

shaded DVR, additional samples are needed for on-the-fly central-difference gradient calculation. For smooth, shaded DVR, this requires six additional texture coordinate warping and subsequent texture fetching steps, therefore resulting in lower frame rates than smooth, unshaded DVR.

Calculating the brick cache fixup step does not impose an additional limitation of the frame rates, as this step is only performed whenever the cache itself changes, e.g., after a transfer function change. However, the brick cache can be updated several times a second, if necessary.

Data set	Fig.	Resolution	Cache Size	Transition Bricks	Render Mode	Discontinuous	Smooth
Abdomen	6.10	$512^2 \times 1112$	$512^2 \times 512$	29%	unshaded	60 fps	33 fps
					shaded	41 fps	9 fps
Ring	6.11	$1518^2 \times 232$	$512^2 \times 256$	22%	unshaded	65 fps	35 fps
					shaded	45 fps	6 fps

Table 6.1: Frame rates of our mixed-resolution approach with and without smooth transitions between resolution levels obtained by warping texture coordinates in the fragment shader. Measured for a 512×512 viewport on a Geforce GTX 280.

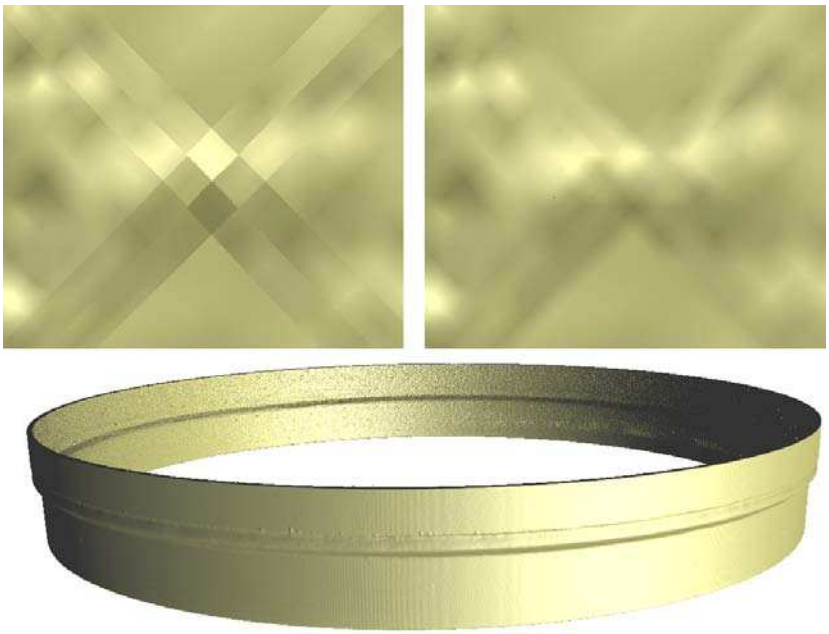


Figure 6.11: High-resolution industrial CT scan of a metal ring (1518x1518x232). Zoom-in: Junction of two low-res and two high-res bricks. Discontinuous (top left) vs. continuous (top right) resolution transition between bricks.

6.6 Summary and Conclusion

In this chapter, a mixed-resolution volume rendering approach for high-quality rendering of large data was introduced. We use a downsampling scheme where the samples are shifted by half a voxel in each dimension, permitting more flexibility in the choice of downsampling filter kernels. Our approach offers C^0-continuous transitions between different resolution levels by special handling of high-resolution brick boundaries which are adjacent to low-resolution bricks. We do this by warping the texture coordinates which are used for hardware-native trilinear interpolation of the sample's value during ray-casting. Prior to rendering, the duplicated voxels in the border outside high-res bricks at each high-res/low-res boundary are adjusted in the corresponding high-resolution brick in a brick cache fixup step.

All the necessary steps to ensure continuous transitions between resolution levels are implemented on the GPU and offer interactive frame rates. Furthermore, we have described an efficient texture packing scheme that allows to dynamically store bricks of different resolutions in the same large 3D cache texture.

Chapter 7

Visualization of Neural Processes in EM Datasets

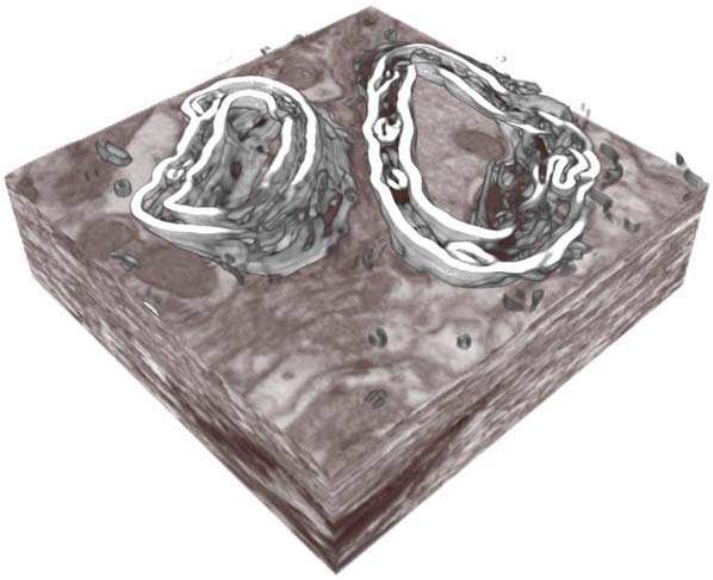


Figure 7.1: Enhanced visualization of neural processes in EM data.

7.1 Introduction

Parts of this chapter are based on the paper *Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets*. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008) [34].

In this chapter, *NeuroTrace*, a system for high-quality 3D visualization and semi-automatic segmentation for neural processes is presented. In recent years EM (Electron Microscopy) technology has enabled bioscientists to scan an unprecedented amount of large-scale high-resolution data, allowing them to reconstruct the complex neural interconnections in the nervous system.

Connectomics [79] is an emerging research area in bioscience, which aims to determine the complete, detailed wiring diagram of neural circuits, with the goal of understanding the function of the brain. Interactive segmentation and visualization is a main requirement by neuroscientists, as it enables them to follow neural processes interactively through the entire volume.

Modern electron microscopes can attain resolutions of three to five nanometers within a slice, with 30 nanometers slice thickness. For the first time, this has made the reconstruction of small neural processes such as synapses or narrow dendritic spines feasible. Automated scanning devices, such as the ATLUM (Automatic Tape-Collecting Lathe Ultramicrotome) at the Harvard Center for Brain Science can produce up to 11 gigabytes of raw data per second. Scanning a $1mm^3$ sample of brain tissue, for example, would result in roughly one petabyte of raw data.

This enormous amount of data, however, poses several challenging problems for data storage, retrieval, processing, segmentation and visualization. EM data is very complex, and segmentation is usually a difficult and time-consuming manual process. Additionally, the complex structure of neural tissue makes the 3D visualization of EM datasets very challenging. Conventional transfer functions alone are not able to clearly depict neural processes and their interconnections, resulting in cluttered images. In addition to that, new scalable methods for visualization are needed, which are able to maintain interactive performance while dealing with this tremendous amount of data.

NeuroTrace is a system for segmentation and visualization of large-scale EM data, designed to be scalable to large data and data-parallel hardware architectures. The segmentation of neural processes is based on a semi-automatic multiphase level set segmentation [88] with 3D centerline tracking along the main axis of the neural structure. The visualization system employs on-the-fly de-noising and edge-detection. A local histogram-based edge metric is used to enhance important structures while reducing the clutter in the rendered image. Additionally, the segmented neural structures are displayed as implicit surfaces in the volume rendered image. To achieve interactive frame rates all methods are implemented on the GPU.

This chapter starts with reviewing related work for 3D visualization of microscopy images (Section 7.2). In Section 7.3 *NeuroTrace* is presented, an application that supports segmentation and visualization of neural processes in EM data. In the scope of this thesis, however, we will focus mainly on the enhanced 3D visualization capabilities of EM data in *NeuroTrace*. Section 7.4 presents a

volume rendering framework which supports on-demand filtering for de-noising and detection of structure boundaries in complex EM datasets. We employ a local histogram-based edge metric to visually enhance the boundaries of neural processes and find regions of interest in the volume. To support processing of large-scale datasets we implemented the algorithm entirely on the GPU and use a dynamic caching system to ensure scalability. Finally, Section 7.5 presents the results of our algorithm and Section 7.6 concludes this chapter.

7.2 Previous Work

Volume rendering of microscopic structures is a very recent area of research, enabled by new developments in microscopic data aquisition techniques. Mayerich et al. [55] presented segmentation and subsequent visualization of microvascular structures and their relationships. However, the resolution of their microscopic images is still two orders of magnitude lower than in EM data. We employ GPU-based ray-casting of volumes and implicit surfaces [74] using a bricking scheme for large data [3] implemented in CUDA.

Enhancing edges or structure boundaries has always been important in volume rendering, and is typically achieved using higher-order transfer functions [44]. Kindlmann and Durkin [36] proposed a boundary emphasis function for enhancing edges in volume rendering. Caban and Rheingans [12] have recently introduced texture-based tranfer functions based on first-, second-, and high-order local (histogram) statistics. However, these methods are not effective in dealing with noise in EM images. For enhancing cell boundaries in 2D TEM images Tasdizen et al. [82] adapted coherence enhancing diffusion by constructing their diffusion tensor based on the Hessian matrix.

The rendering framework presented here, employs a general filtering and denoising step with a neighborhood size that can be changed interactively. Viola et al. [90] have presented early GPU implementations of non-linear filters such as median, or bilateral filters.

Martin et al. [54] define a set of brightness, color, and texture cues for constructing a local boundary model. To enhance edges during ray-casting we extended their 2D boundary detection framework using local histogram comparisons.

7.3 NeuroTrace

NeuroTrace is an application which provides interactive segmentation and high-quality 3D volume visualization of high-resolution electron microscope datasets. *NeuroTrace* focuses on maintaining the scalability of the system to large-scale datasets and high-performance parallel computing architectures. A screenshot of *NeuroTrace* is displayed in Figure 7.2.

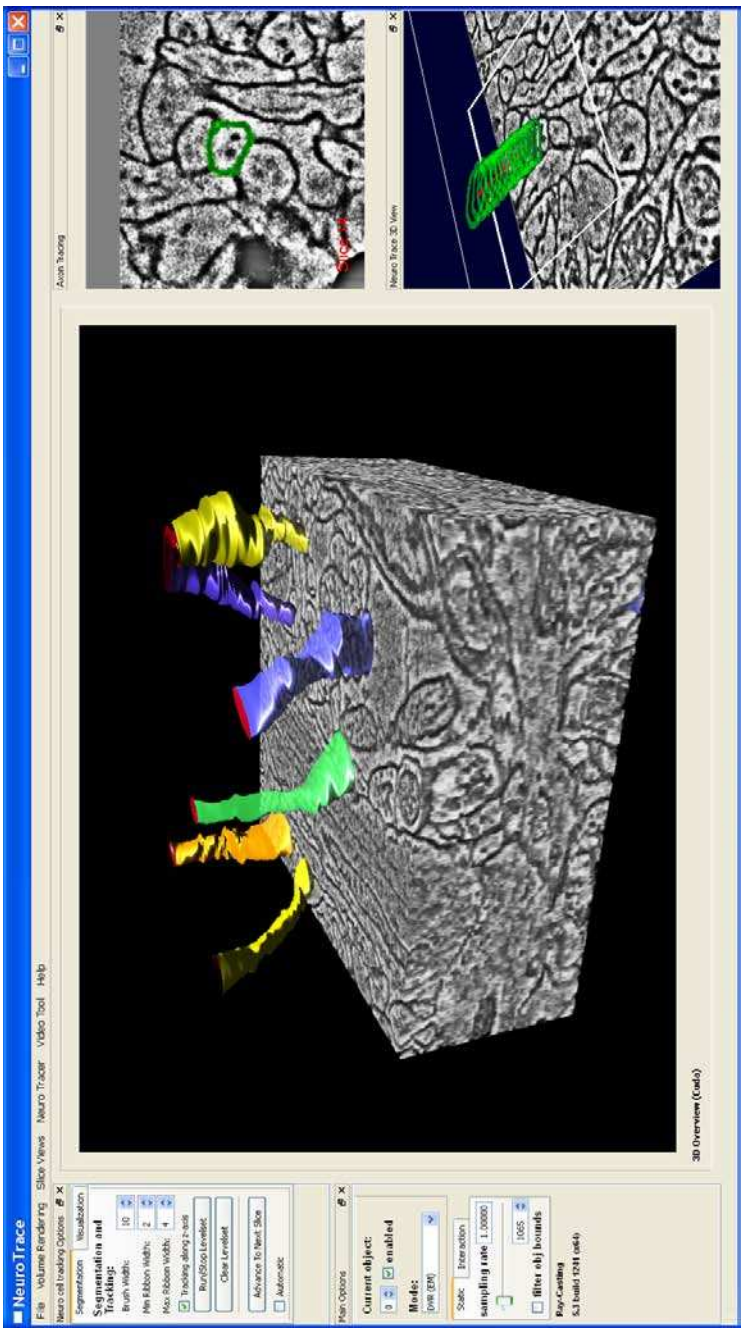


Figure 7.2: *NeuroTrace* allows neuroscientists to interactively explore and segment neural processes in high-resolution EM data. The large view shows a 3D volume rendering of the original EM data in combination with implicit surface ray-casting of segmented structures. The smaller views on the right depict the ongoing segmentation of an axon.

7.3.1 Pre-Processing

NeuroTrace is embedded into a workflow that supports bioscientists in their research. After acquisition of the EM datasets the individual image tiles are first aligned and stitched into a single large high-resolution slice image. Next, the slices are aligned and registered along the z-direction to create a 3D volume. These pre-processing steps are performed outside of *NeuroTrace*, prior to starting the application.

7.3.2 NeuroTrace Workflow

Figure 7.3 illustrates the workflow in *NeuroTrace*, supporting integrated and interactive visualization and segmentation of neural processes.

The workflow in *NeuroTrace* starts with the visual inspection of the 3D EM volume, prior to segmentation. The objective of the user is to gain an overview of the dataset and identify regions of interest that contain important structures. We support this step by enhancing boundaries of important structures in the volume (e.g., myelinated axons) while simultaneously removing background noise. To better delineate structures of interest we propose on-the-fly noise removal and edge enhancement, as described in Section 7.4.

To specify a rectangular ROI (region of interest) as starting point for the segmentation, the user can navigate a view-aligned clipping plane through the volume and define the ROI center by clicking on any point on the plane. Next, the user can start the semi-automatic segmentation of a neural process by roughly painting the outline of the process in the 2D view of the rectangular ROI. This initializes the active ribbon segmentation step, which automatically detects the 2D boundary of the neural process that is to be segmented. After the cell boundary has been detected, the segmentation algorithm proceeds to the next slice by using a centerline tracking method [34].

At all times, the ongoing segmentation process is displayed interactively in 3D, allowing the user to inspect and modify the segmentation at any time. The

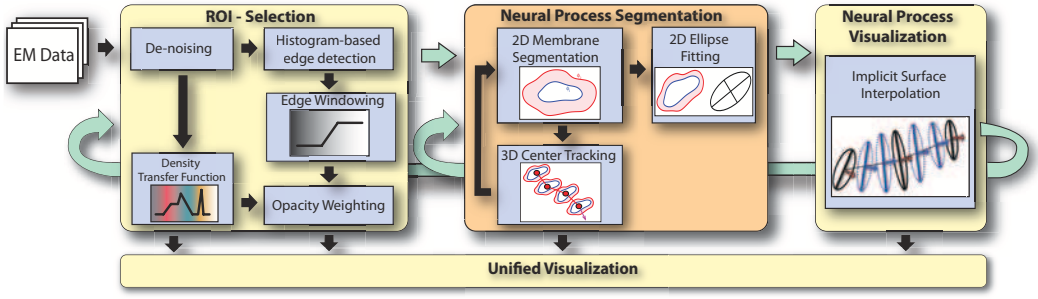


Figure 7.3: Pipeline diagram of our integrated, interactive workflow for visualizing and segmenting neural processes.

segmented axons, the most important neural processes in our case, have long, thin, elongated structures, which can be approximated by a list of elliptical cross-sections. Therefore, in each slice, we approximate the actual boundary of the axon by fitting an ellipse. This allows us to store the segmentation information in a very compact and memory efficient way. Each axon is represented by a list of ellipses, which are transferred to the volume renderer. Next, during ray-casting, each sample along the ray is tested for potential intersections with the iso-surfaces that delineate the segmented structures. If an intersection with an implicit surface is detected, the sample point is colored accordingly, and the accumulation of the color and opacity along the ray continues.

7.3.3 NeuroTrace Framework

NeuroTrace is implemented in C++, OpenGL, NVIDIA CUDA and Qt. We use CMake as a cross-platform make tool to be able to easily port the existing application to different operating systems. At the moment, we have built *NeuroTrace* for Windows and Linux.

The *NeuroTrace* framework consists of four main conceptual modules:

- *NeuroTrace Core*: This module is the core of the *NeuroTrace* framework and responsible for data handling and storage, including out-of-core data loading, octree generation and cache management.
- *NeuroTrace Segmentation Module*: This module implements the segmentation functionality of *NeuroTrace*, the active ribbon segmentation algorithm.
- *NeuroTrace Visualization Module*: This module is responsible for 3D visualization, including direct volume rendering of EM data, on-demand denoising and edge-detection, and implicit surface ray-casting of segmented processes (see Section 7.4).
- *NeuroTrace GUI*: This module provides the user interface to the entire application which is completely decoupled from the actual core, segmentation and visualization modules.

Data management is handled in the core. We use a separate loader thread for opening and loading datasets. Whenever the thread has finished loading a new block of data, it is interactively added to the visualization. This streaming approach allows us to speed up the initial startup time of the volume renderer by loading and displaying low-resolution blocks first, while the loader thread in the background still reads in the higher resolution blocks. The encapsulation of all data access functions into our data management layer will allow us to switch from local data storage to a network-based file system without any changes in

the visualization and segmentation modules.

The *segmentation module* accesses the original data via the *NeuroTrace* core module. The segmentation results get propagated back and passed on to the interactive visualization system. The segmentation module is treated as a black box by the *NeuroTrace* framework, meaning that the actual implementation of the segmentation algorithms is not known outside of the module.

The *visualization module* also accesses the data via the *NeuroTrace* core module. It offers 2D slice and 3D rendering at interactive frame rates, transfer functions, clipping planes, etc. In the case of binary segmentation information, the visualization module also offers two-level volume rendering for segmentation masks.

The *user interface* was developed with the Qt framework. We have implemented a flexible layout using dockable views, which can individually be adjusted by the user, depending on his/her primary requirements. We have implemented a transfer function editor for 1D as well as for 2D transfer functions. Additionally, we offer several adjustable 2D slice as well as 3D views which can be individually activated or hidden, depending on user preferences. We have intentionally kept the user interface decoupled from the rest of the application to be able to switch from one GUI framework to another, if necessary in the future.

Additionally, we keep two tables of all visualization parameters that can be changed in the GUI. One table at the user-interface side, and one table at the renderer-side. This allows us to synchronize the two tables at an arbitrary time in a thread-safe way. Using a separate thread for the user interface keeps the application responsive, even during time-consuming processing tasks.

7.4 Volume Visualization

Volume rendering of high-resolution EM data poses several challenges. EM data is extremely dense and heavily textured, exhibits a complex structure of interconnected nerve cells, and has a low signal-to-noise ratio. Therefore, standard volume rendering results in cluttered images that make it hard to identify regions of interest or to observe an ongoing segmentation.

The visualization approach presented here, supports the inspection of data prior to segmentation, for identifying regions of interest (ROIs), as well as the visualization of the ongoing and final segmentation (see Figure 7.3). To improve the visualization of the raw data prior to segmentation, we have implemented on-the-fly nonlinear noise removal and edge enhancement to support the user in finding and selecting ROIs. Using a local histogram-based edge metric, which is only calculated on demand for currently visible parts of the volume and cached for later reuse, we can enhance important structures (e.g., myelinated axons) while

fading out less important regions. During ray-casting we use the computed edge values to modulate the current sample's opacity with different user-selectable opacity weighting modes (e.g., min, max, alpha blending).

7.4.1 On-demand Filtering

The main motivations for on-demand filtering (i.e., noise removal and edge detection) are the flexibility offered by being able to change filters and filter parameters on the fly while avoiding additional disk storage and bandwidth bottlenecks for terabyte-sized volume data. We perform filtering only on blocks of the volume that are visible from the current viewpoint, and store the computed data directly on the GPU for later reuse. We have implemented a caching scheme for these precomputed blocks on the GPU to avoid costly transfers to and from GPU memory while at the same time avoiding repetitive recalculation of filtered blocks. During visualization we display either the original volume, the noise-reduced data, the computed edge values, or a combination of the above.

Our on-demand filtering algorithm consists of several steps: (1) Detect the visible blocks for the current viewpoint (see Figure 7.4). (2) Build the list of blocks that need to be computed. (3) Perform noise removal filtering on selected blocks and store them in the cache. (4) Calculate the histogram-based edge metric on selected blocks and store these blocks in the cache. (5) High-resolution ray-casting combining edge values and original data values. The detection of visible blocks (Step 1) is done either in a separate low-resolution ray-casting pass or included in Step 5.

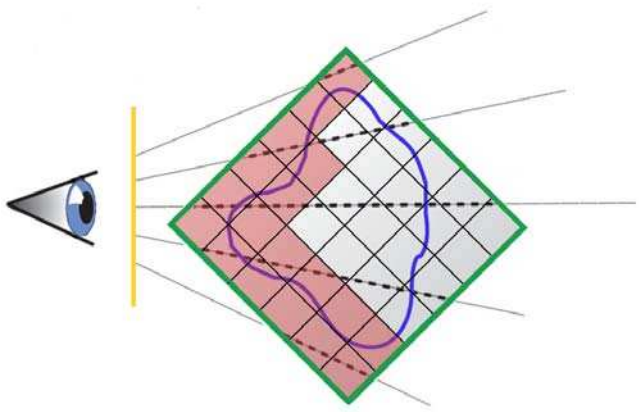


Figure 7.4: Detection of visible blocks (red) from the current viewpoint during ray-casting. Only visible blocks that are not currently in the cache need to be filtered and stored in the GPU cache.

7.4.2 Noise Removal

Since EM data generally exhibits a low signal-to-noise ratio we have integrated an on-demand noise removal filter step into our pipeline prior to calculating the local histogram-based edge metric. We perform the filtering only on those blocks that were marked as visible and are not present in the cache yet. We have implemented 2D and 3D Gaussian, mean, non-linear median, bilateral [86], and anisotropic diffusion filters [61] with user adjustable neighborhood sizes. Especially non-linear filters have shown good noise removal properties without degrading edges in the EM data [82]. Our main objective, however, was to develop a general framework for noise removal, where additional filters can be added easily. The result for each processed block is stored in the cache and used as input for the edge detection algorithm.

7.4.3 Local Histogram-based Edge Detection

We use a local histogram-based edge metric to modulate the opacity of the EM data during ray-casting. Boundaries in the volume get enhanced while more homogenous regions are supressed. This helps the user in navigating through the unsegmented dataset and in finding regions where a segmentation should be started. The edge metric is computed only for visible blocks that are not stored in the cache yet.

Our edge detection algorithm is based on the work of Martin et al. [54] who introduced edge and boundary detection in 2D images, based on local histograms. They did a thorough evaluation of different brightness, color, and texture cues for constructing a local boundary model, which was subsequently used to detect contours [51] in natural images.

In our local histogram-based edge detection approach we take a block neighborhood around each voxel to calculate the brightness gradient for different directions. We separate the voxel's neighborhood along the given direction into two halves and calculate the histogram in each half-space. The different possible directions of the half-spaces used for histogram calculation are depicted in Figure 7.5. Finally, the histogram difference is calculated using the χ^2 distance metric [64]. A high difference between histograms indicates an abrupt change in brightness in the volume, i.e., an edge. The maximum difference value over all directions is saved as the edge value in the cache block. As the neighborhood size for the histogram calculation can be adjusted to match the resolution level of the current input data, this approach scales to large data and to volume subdivision schemes like octrees. Again, we have kept the implementation of our edge detection framework as modular as possible to support adding different edge detection algorithms in the future. During volume rendering, we fetch at each sample location the corresponding edge value and use it to modulate the sample's opacity and/or color. Optionally, the user can first use a windowing function on

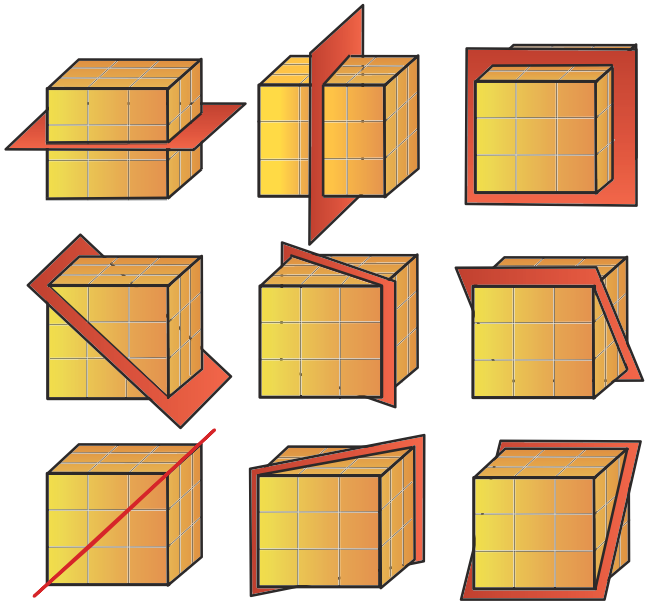


Figure 7.5: Possible configurations for separating a 3×3 voxel neighborhood into two half-spaces, used for histogram calculation and comparison.

the calculated edge values to further enhance the visualization.

7.4.4 Dynamic Caching

To improve the performance of our edge-based visualization scheme, we have implemented a dynamic caching scheme for storing blocks computed on-the-fly. Two caches are allocated directly on the GPU, one cache to store de-noised volume blocks and the second cache to store blocks containing the calculated edge values. First, the visibility of all blocks is updated for the current viewpoint in a first ray-casting pass and saved in a 3D array corresponding to the number of blocks in the volume. Next, all blocks are flagged as either: (1) visible, present in cache; (2) visible, not present in cache; (3) not visible, present in cache; or (4) not visible, not present in cache. Visible blocks that are already in the cache (flagged with (1)) do not need to be recomputed. Only blocks flagged with (2) need to be processed. Therefore, indices of blocks flagged with (2) are stored for later calculation (see Section 7.4.5). During filtering/edge detection the computed blocks are stored in the corresponding cache. A small lookup table is maintained for mapping between block storage space in the cache to actual volume blocks as described in Chapter 6. Unused blocks are kept in the cache for later reuse (flagged with (3)). However, if cache memory gets low, unused blocks are flushed from the cache and replaced by currently visible blocks.

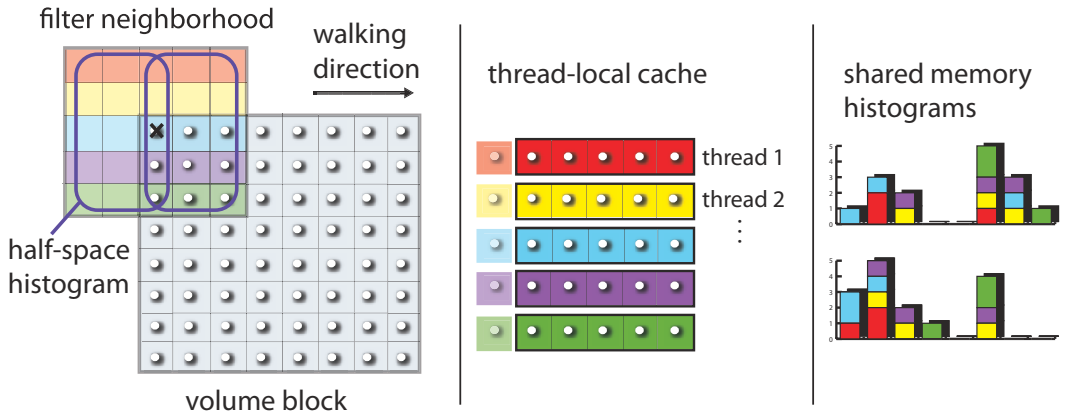


Figure 7.6: Local histogram-based edge detection in volume blocks using CUDA. (a) Neighborhood required for local histograms. (b) Fetching only one new sample per thread at each step to update the neighborhood. (c) Shared histograms for calculation of the χ^2 difference.

7.4.5 GPU Implementation

After detecting which blocks need processing, a CUDA kernel is launched with a grid size corresponding to the number of blocks that need to be processed. For simplicity we explain the implementation of our filtering and edge detection algorithm in 2D. The extension to 3D is straightforward.

To calculate filter/edge values in each block, we start a CUDA kernel with a CUDA block size that corresponds to the user specified neighborhood size, but of one dimension lower than the actual neighborhood (e.g., for a 3D neighborhood a 2D CUDA block is started, for a 2D neighborhood a 1D CUDA block is started). Figure 7.6 depicts the case where the edge detection of a block uses a 5×5 neighborhood. In this case the kernel is started with five concurrent threads. Next, the threads iterate over the entire block that needs to be filtered and calculate the filter/edge values for each voxel. Each thread is responsible for only one part of the filter's neighborhood, as depicted by the colored areas in Figure 7.6. To reduce redundant texture fetches each thread locally caches its previously fetched values. The size of this thread-local array corresponds to the neighborhood size of the filter. Therefore, at each step a thread only needs to perform one texture fetch, and store the value in its local cache (Figure 7.6b).

To calculate the local-histogram based edge metric, all samples in a voxel's neighborhood need to be assigned to one of the two local histograms (for both half-spaces), as depicted in Figure 7.6c. The histograms are stored in shared CUDA memory and used for the final calculation of the χ^2 histogram difference. The main steps for each thread are: (1) Update the histogram of the first half-

space ($histogram_{left}$) by removing the sample that has left the filter neighborhood and adding the last sample from $histogram_{right}$. (2) Remove the sample that has left the filter neighborhood from the thread-local cache. (3) Fetch the sample that has entered the filter neighborhood from the volume texture and store it in the thread-local cache. (4) Update the histogram of the second half-space ($histogram_{right}$) by removing the sample that is now in $histogram_{left}$ and adding the sample that has just been fetched from the volume texture.

All threads are synchronized after they have performed the above steps using atomic CUDA operations for updating the shared histograms. Now the χ^2 histogram difference for the current neighborhood can be computed and stored in the cache.

To implement the de-noising filters we use the same basic strategy. For Gaussian filters we transfer a 1D look-up table of the weights to the GPU to speed up the calculation. For bilateral filtering we use the same look-up table to calculate the geometric closeness function, whereas the photometric similarity function is calculated on-the-fly in the CUDA kernel. For median filtering we implemented bitonic sort on the GPU to find the median value of the filter neighborhood. Anisotropic diffusion filtering is the most complex filter in our framework. It requires a second filter cache to allow ping-pong swaps between source and destination. Also, costly neighborhood lookups in the source cache are needed to compute the boundary values of the destination blocks.

If the noise removal step is performed prior to the edge detection, the local histogram calculation uses the values from the filtered block cache as input values instead of the original volume texture. Since the calculation of edge-blocks requires samples from the block's neighborhood (depending on the edge filter's neighborhood size), special care has to be taken that all necessary de-noised values are available when computing an edge-block. One way to handle this is to add a duplicated border around each de-noised block, so that all neighborhood samples that are required for edge detection are already stored in the corresponding de-noised block. Another way to deal with this problem is to detect the additional blocks that would have to be de-noised. Now if a sample outside the block boundary of a de-noised block is needed, a neighborhood lookup can be performed to fetch the correct block and the corresponding sample.

7.5 Results and Evaluation

The prefiltering and edge-detection methods (Figure 7.7) were both implemented entirely in CUDA and achieve interactive frame rates. Filtering blocks on-demand and caching them for later reuse allows the user to change filters and filter settings interactively. Especially de-noising prior to calculating the edge metric improved the results considerably. The best results were achieved using anisotropic diffusion filtering. For our local histogram-based edge metric we found a histogram with

64 bins to be sufficient for our data. Also, a simple average-based histogram difference operator showed good results compared to the computationally more complex χ^2 distance metric. For our caching scheme we used 8^3 sized blocks, but this can be adjusted according to the resolution of the data. At the moment our implementation of the cache is based on CUDA arrays, but in the future we would like to use 3D textures to improve tri-linear filter performance during ray-casting.

The dimension of EM data is highly anisotropic, with z-slice distances that can be a factor of 10 or more larger than the pixel resolution. This poses real problems for volume visualization, since the visible edges from axons are shifted considerably between slices. However, promising new advances in electron microscopy might reduce the minimum z-slice distance significantly. Even though our filtering and edge detection method works better than traditional transfer functions, the results are sometimes still ambiguous and confusing, requiring closer inspection of the 2D slice views to identify the ROI.

7.6 Summary and Conclusion

To support neuroscientists in their effort to study the connections of the mammalian or human brain, scalable algorithms for the visualization of large-scale, high-resolution EM data is of immediate interest.

In this chapter *NeuroTrace*, a novel interactive segmentation and visualization system for neural processes in EM volumes was introduced. The main contribution in this chapter is a volume rendering method with on-the-fly filtering and edge detection, a scalable implementation of these methods on the GPU, and its integration into the *NeuroTrace* framework.

Using *NeuroTrace*, neuroscientists are able to work on the reconstruction of the neural circuitry, hopefully one day leading to a better understanding of the brain's intricacies and function.

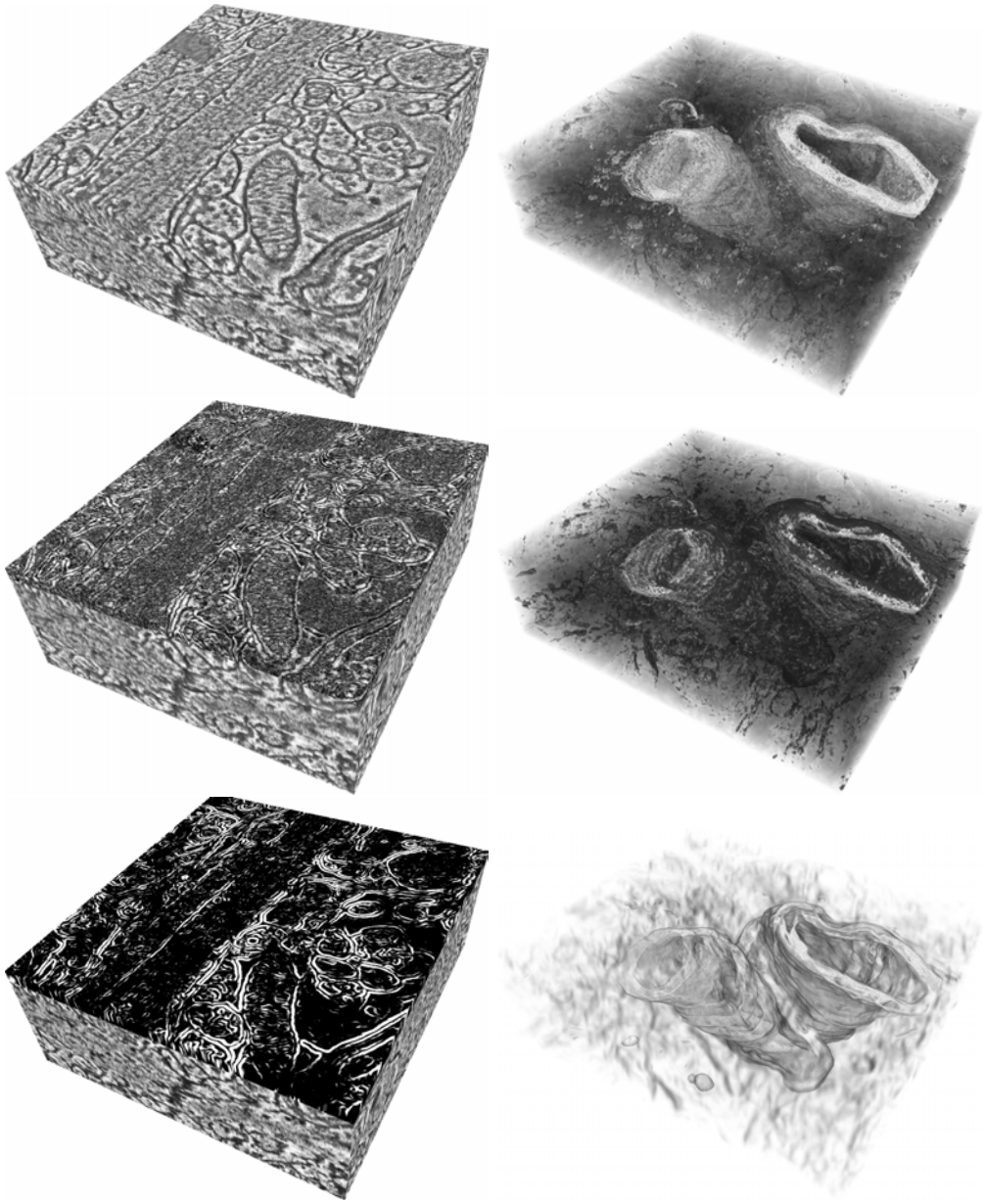


Figure 7.7: Left: Volume Slab visualization; Top: Original data; Middle: Gradient magnitude displayed on the top slice; Bottom: Local-histogram edges; Right: Volume Rendering; Top: Original data; Middle: Gradient-magnitude shaded; Bottom: Pre-filtering and edge enhancement with opacity weighting.

Chapter 8

Summary and Conclusions

Today's GPUs and high-performance parallel computing architectures offer never before achieved flexibility and computing power for scientific programming. For the first time, this allows us to include sophisticated, high-quality, interactive volume rendering techniques for large and complex data into real-world applications.

This thesis has focused on two different application areas: surgical planning in medicine, and connectomics research in neurobiology. Several different visualization methods for rendering of multi-modal, complex, and large data have been presented.

In today's medicine, diagnosis and treatment planning heavily relies on recent advances in image scanning technologies. The visual combination of images from several different scanning modalities into a fused visualization helps doctors to get a better understanding of the individual patient's anatomy and pathology. However, standard visualization systems need to be adapted and integrated into the clinical workflow to be useful for doctors and medical personnel. In addition to concurrent multi-volume rendering, intuitive interaction metaphors are needed, which support doctors in their decision finding process.

In this thesis an application for precise preoperative planning of neurosurgical interventions and approaches to the brain has been presented, which uses multiple radiological imaging modalities to delineate the patient's anatomy, neurological function, and metabolic processes.

For interventions in areas directly below the skull, we addressed the problem of opening the cranial bone tailored to the individual anatomy. The optimal surgical approach to the brain is simulated by allowing the doctor to interactively specify parts of the patient's skin and skull that should be removed.

In addition, a multi-modal volume rendering framework was presented for planning surgeries on deep-seated structures, where a small opening in the skull is sufficient to gain access to a much larger intracranial region via an endoscope or operating microscope. To enhance the spatial orientation of doctors in regard to structures of interest and critical areas within the brain, multi-volume rendering techniques were introduced, which work either purely on the data or include additional segmentation masks. The visual appearance of rendered segmented objects was improved by an algorithm for smooth rendering of object boundaries.

In the field of Connectomics and neurobiology, the current visualization research focus is on the development of scalable methods for volume rendering of tremendously large and complex data. The latest generation of electron microscopes can provide huge amounts of high-resolution data. Segmentation and visualization of the complex neural tissue aquired by electron microscopy, however, is very challenging and still on-going research.

The second part of this thesis has dealt with volume rendering techniques and interaction metaphors for large data. A new volume ray-casting approach was introduced, which circumvents artifacts at block boundaries of different resolution. Volumes larger than GPU memory can be rendererd in a single ray-casting pass, mixing different levels of resolution with continuous transitions between resolution levels.

Finally, *NeuroTrace*, an application for interactive segmentation and visualization of neural processes in EM volumes was introduced. The visual appearance of volume rendered EM data was improved considerably, addressing one of the most common problems in volume rendering of complex data - visual clutter. A novel on-demand filtering and edge detection method, based on a high-quality volume rendering framework, was integrated into *NeuroTrace* and presented.

This thesis presented significant progress in applying novel volume rendering techniques to problems in different scientific fields. By building on the flexibility and parallel processing power of current GPUs, new interactive volume visualization methods could be developed, supporting medical doctors and neuroscientists in their work. However, several challenges still remain, requiring further research in scalable techniques and interaction metaphors. Especially user-centered techniques for exploring, segmenting, and analyzing large datasets, which scale to the multi-resolution representation of large data, are on-going research challenges.

Acknowledgments

This thesis and the related publications represent the work I carried out at the VRVis Research Center in cooperation with the Institute of Computer Graphics and Algorithms at the Vienna University of Technology since the fall of 2005. This work would never have been possible without the support of many co-workers, collaborators, friends and family. The neurosurgery project was carried out as a collaboration with the Neurosurgery Department at the Medical University of Vienna, who also provided the medical datasets. The neuroscience project was a joint effort with the Harvard Initiative on Innovative Computing and the Harvard Center for Brain Science, which provided the EM datasets.

First of all, I would like to thank my advisors Markus Hadwiger, Katja Bühler and Meister Eduard Gröller for their constant support and guidance. I especially want to thank Markus for sharing his passion for visualization research, for creating a relaxed, fun and passionate working atmosphere and for his helpful comments and programming expertise. I want to extend my thanks to Katja, for giving me the opportunity to work at the VRVis and for introducing me to the world of research.

Many thanks go to my colleagues Laura Fritz, Thomas Höllt, Philipp Muigg, Florian Schulze, Jiri Hladuvka and Sebastian Zambal and the students Jana Banova and Andreas Ritzberger for many fruitful discussions and for always allowing me to bounce ideas off them. This thesis is based on the helpful reviews and proofreading of my co-workers and room mates in the loudest laughing room at VRVis. Laura, Thomas and Markus, thank you so much! I also want to extend my gratitude to Philipp, for supporting me during my first presentation at Vis 2007, and preventing me from running off and hiding in the basement when the laptop crashed, the beamer malfunctioned and my presentation froze.

Many thanks go to Dr. Stefan Wolfsberger, who is the most enthusiastic volume rendering user I have ever met and who made the neurosurgical planning tool what it is today. I also want to extend my thanks to my international collaborators Christof Rezk-Salama, Torsten Möller and my colleagues at Harvard, Won-Ki Jeong, Amelio Vasquez, Duncan Mak and, of course, Hanspeter Pfister and our collaborating neuroscientists Bobby Kasthuri, Davi Bock, Jeff Lichtman and Clay Reid. Finally, I want to express my gratitude to my family, for their encouragement and support over all these years and their trust in my decisions.

Bibliography

[1] M.S. Atkins, K. Siu, B. Law, J.J. Orchard, and W.L. Rosenbaum. Difficulties of T1 brain MRI segmentation techniques. In *Proceedings of SPIE Medical Imaging 2002*, volume 4684, pages 1837–1844, 2002.

[2] D. Bartz, W. Straßer, Ö. Gürvit, D. Freudenstein, and M. Skalej. Interactive and multi-modal visualization for neuroendoscopic interventions. In *Data Visualization (Proceedings of Symposium on Visualization 2001)*, pages 157–164, 2001.

[3] J. Beyer, M. Hadwiger, T. Möller, and L. Fritz. Smooth mixed-resolution GPU volume rendering. In *Proceedings of IEEE International Symposium on Volume and Point-Based Graphics (Volume Graphics 2008)*, pages 163–170, 2008.

[4] J. Beyer, M. Hadwiger, S. Wolfsberger, and K. Bühler. High-quality multimodal volume rendering for preoperative planning of neurosurgical interventions. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007)*, 13(6):1696–1703, 2007.

[5] J. Beyer, M. Hadwiger, S. Wolfsberger, C. Rezk-Salama, and K. Bühler. Segmentierungsfreie Visualisierung des Gehirns für Direktes Volume Rendering. In *Proceedings of Bildverarbeitung für die Medizin 2007*, pages 333–337, 2007.

[6] J. Beyer, C. Langer, L. Fritz, M. Hadwiger, S. Wolfsberger, and K. Bühler. Interactive diffusion based smoothing and segmentation of volumetric datasets on graphics hardware. In *Methods of Information in Medicine*, volume 46, pages 270–274, 2007.

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

[8] C.P. Botha and F.H. Post. New technique for transfer function specification in direct volume rendering using real-time visual feedback. In *Proceedings of SPIE Medical Imaging 2002*, volume 4681, pages 349–356, 2002.

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

[10] L.G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24:325–376, 1992.

[11] S. Bruckner, S. Grimm, A. Kanitsar, and M.E. Gröller. Illustrative context-preserving volume rendering. In *Proceedings of EG/IEEE-VGTC Symposium on Visualization (EuroVis) 2005*, pages 69–76, 2005.

[12] J.J. Caban and P. Rheingans. Texture-based transfer functions for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008)*, 14(6):1364–1371, 2008.

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

[14] M. Capek, L. Mroz, and R. Wegenkittl. Robust and fast medical registration of 3D-multi-modality data sets. In *Proceedings of Medicon 2001*, pages 515–518, 2001.

[15] J. Dauguet, D. Bock, R.C. Reid, and S.K. Warfield. Alignment of large image series using cubic b-splines tessellation: application to transmission electron microscopy data. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2007*, pages 710–717, 2007.

[16] W. Denk and H. Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biol*, 2(11):1900–1909, 01 2004.

[17] S.P. DiMaio, N. Archip, N. Hata, I.F. Talos, S.K. Warfield, A. Majumdar, N. McDannold, K. Hynynen, P.R. Morrison, W.M. Wells, D.F. Kacher, R. Ellis, A.J. Golby, P.M. Black, F.A. Jolesz, and R. Kikinis. Image-guided neurosurgery at Brigham and Women's Hospital: The integration of imaging, navigation and interventional devices. *IEEE Engineering in Medicine and Biology Magazine*, 25(5):67–73, 2006.

[18] K. Engel, M. Hadwiger, J.M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-time volume graphics*. A. K. Peters, Ltd., 2006.

[19] A. Entezari, T. Meng, S. Bergner, and T. Möller. A granular three dimensional multiresolution transform. In *Proceedings of EG/IEEE VGTC Symposium on Visualization (EuroVis) 2006*, pages 267–274, 2006.

[20] C. Everitt, A. Rege, and C. Cebenoyan. Hardware Shadow Mapping. Technical report, NVIDIA Corp., 2000. `http://developer.nvidia.com/attach/8456`.

[21] P. Felkel, R. Wegenkittl, and M. Bruckschwaiger. Implementation and complexity of the watershed-from-markers algorithm computed as a minimal cost forrest. *Computer Graphics Forum (Proceedings of Eurographics 2001)*, 20(3):26–35, 2001.

[22] M. Ferré, A. Puig, and D. Tost. Rendering techniques for multimodal data. In *Proceedings of 1st Ibero-American Symposium on Computer Graphics (SIACG) 2002*, pages 205–313, 2002.

[23] M. Ferré, A. Puig, and D. Tost. A framework for fusion methods and rendering techniques of multimodal volume data. *Computer Animation and Virtual Worlds*, 15:63–77, 2004.

[24] D.T. Gering, A. Nabavi, R. Kikinis, L.J. O'Donnell, W.E.L. Grimson F.P.M. Black, and W.M. Wells III. An integrated visualization system for surgical planning and guidance using image fusion and an open MR. *Journal of Magentic Resonance Imaging*, 13:967 – 975, 2001.

[25] A. Ghosh, P. Prabhu, A.E. Kaufman, and K. Mueller. Hardware assisted multichannel volume rendering. In *Proceedings of Computer Graphics International 2003*, pages 2–7, 2003.

[26] Khronos OpenCL Working Group. The OpenCL specification 1.0, October 2009. `http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf`.

[27] S. Guthe and W. Strasser. Advanced techniques for high-quality multiresolution volume rendering. *Computers & Graphics*, 28:51–58, 2004.

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

[29] M. Hadwiger, P. Ljung, C. Rezk-Salama, and T. Ropinski. Advanced illumination techniques for GPU-based volume raycasting - course notes (web version: `http://www.voreen.org/241-SIGGRAPH-Asia-08-Course.html`). In *Proceedings of SIGGRAPH Asia 2008*, 2008.

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

[31] Tom Halfhill. Parallel Processing with CUDA. *Microprocessor Report*, 2008.

[32] W.M. Jainek, S. Born, D. Bartz, , W. Straßer, and J. Fischer. Illustrative hybrid visualization and exploration of anatomical and functional brain data. *Computer Graphics Forum (Proceedings of EG/IEEE-VGTC Symposium on Visualization 2008)*, 27(3):855–862, September 2008.

[33] P. Jannin, O.J. Fleig, E. Seigneuret, C. Grova, X. Morandi, and J.M. Scarabin. Multimodal and multi-informational neuro-navigation. In *Proceedings of Computer Assisted Radiology and Surgery (CARS) 2000*, pages 167–172, 2000.

[34] W.-K. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R. Whitaker. Scalable and interactive segmentation and visualization of neural processes in EM datasets. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2009)*, 15(6):1505–1514, 2009.

[35] E. Jurrus, M. Hardy, T. Tasdizen, P.T. Fletcher, P. Koshevoy, C.-B. Chien, W. Denk, and R.T. Whitaker. Axon tracking in serial block-face scanning electron microscopy. *Medical Image Analysis (MEDIA)*, 13(1):180–188, February 2009.

[36] G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of IEEE Volume Visualization 1998*, pages 79–86, 1998.

[37] J. Klein, D. Bartz, O. Friman, M. Hadwiger, B. Preim, F. Ritter, A. Vilanova, and G. Zachmann. Advanced algorithms in medical computer graphics. In *Annex to the Conference Proceedings of Eurographics 2008*, pages 25–44, 2008.

[38] J. Klein, O. Friman, M. Hadwiger, B. Preim, F. Ritter, A. Vilanova, G. Zachmann, and D. Bartz. Visual computing for medical diagnosis and treatment. *Computers & Graphics*, 33(4):554–565, 2009.

[39] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[40] M. Kraus and T. Ertl. Adaptive texture maps. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware 2002*, pages 7–15, 2002.

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

[42] A. Krueger, C. Kubisch, B. Preim, and G. Strauss. Sinus endoscopy - application of advanced GPU volume rendering for virtual endoscopy. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008)*, 14(6):1491–1498, 2008.

[43] E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of IEEE Visualization 1999*, pages 355–362, 1999.

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

[45] F. Link, M. Koenig, and H.O. Peitgen. Multi-resolution volume rendering with per object shading. In *Proceedings of Vision, Modeling and Visualization 2006*, pages 185–191, 2006.

[46] P. Ljung. *Methods for Direct Volume Rendering of Large Data Sets*. PhD thesis, Linköping University, Department of Science and Technology, 2006.

[47] P. Ljung, C. Lundström, and A. Ynnerman. Multiresolution interblock interpolation in direct volume rendering. In *Proceedings of EG/IEEE-VGTC Symposium on Visualization (EuroVis) 2006*, pages 259–266, 2006.

[48] P. Ljung, C. Lundstrom, A. Ynnerman, and K. Museth. Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics 2004*, pages 25–32, 2004.

[49] J.H. Macke, N. Maack, R. Gupta, W. Denk, B. Schölkopf, and A. Borst. Contour-propagation algorithms for semi-automated reconstruction of neural processes. *Journal of Neuroscience Methods*, 167(2):349–357, 01 2008.

[50] J. Maintz and M. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.

[51] M. Maire, P. Arbelaez, C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2008*, pages 1–8, 2008.

[52] I.H. Manssour, S.S. Furuie, S.D. Olabarriaga, and C.M.D.S. Freitas. Visualizing inner structures in multimodal volume data. In *Proceedings of SIBGRAPI 2002*, pages 51–58, 2002.

[53] W.R. Mark, R.S. Glanville, K. Akeley, and M.J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. In *Proceedings of SIGGRAPH 2003*, pages 896–907, New York, NY, USA, 2003. ACM.

[54] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):530–549, 2004.

[55] D. Mayerich, L. Abbott, and J. Keyser. Visualization of cellular and microvascular relationships. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008)*, 14(6):1611–1618, 2008.

[56] B.H. McCormick. Visualization in scientific computing. *ACM SIGBIO Newsletter*, 10(1):15–21, 1988.

[57] R. Mullick, R.N. Bryan, and J. Butman. Confocal volume rendering: Fast segmentation-free visualization of internal structures. In *Proceedings of SPIE Medical Imaging 2000*, volume 3976, pages 70–76, 2000.

[58] A. Neubauer. *Virtual Endoscopy for Preoperative Planning and Training of Endonasal Transsphenoidal Pituitary Surgery*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2005.

[59] A. Neubauer, S. Wolfsberger, M. Forster, L. Mroz, R. Wegenkittl, and K. Bühler. STEPS - an application for simulation of transsphenoidal endonasal pituitary surgery. In *Proceedings of IEEE Visualization 2004*, pages 513–520, 2004.

[60] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture, programming guide version 2.0. Website, September 2009. `http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf`.

[61] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. *IEEE Transactions in Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[62] D.L. Pham, C. Xu, and J.L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–228, 2000.

[63] B. Preim and D. Bartz. *Visualization in medicine. Theory, algorithms and applications*. Morgan Kaufmann Series in Computer Graphics, 2007.

[64] J. Puzicha, J.M. Buhmann, Y. Rubner, and C. Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *Proceedings of International Conference on Computer Vision (ICCV) 1999*, pages 1165–1172, 1999.

[65] P. Rautek, S. Bruckner, and M.E. Gröller. Semantic layers for illustrative volume rendering. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007)*, 13(6):1336–1343, 2007.

[66] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware 2000*, pages 109–118, 2000.

[67] C. Rezk-Salama, M. Keller, and P. Kohlmann. High-level user interfaces for transfer function design with semantics. 11(5):1021–1028, 2006.

[68] C. Rezk-Salama and A. Kolb. Opacity peeling for direct volume rendering. 25(3):597–606, 2006.

[69] C. Rieder, F. Ritter, M. Raspe, and H.O. Peitgen. Interactive visualization of multimodal volume data for neurosurgical tumor treatment. *Computer Graphics Forum (Proceedings of EG/IEEE-VGTC Symposium on Visualization 2008)*, 27(3):1055–1062, 2008.

[70] C. Rieder, M. Schwier, H.K. Hahn, and H.O. Peitgen. High-quality multimodal volume visualization of intracerebral pathological tissue. In *Eurographics Workshop on Visual Computing for Biomedicine 2008*, pages 167–176, 2008.

[71] F. Rößler, E. Tejada, T. Fangmeier, T. Ertl, and M. Knauff. GPU-based multi-volume rendering for the visualization of functional brain images. In *Proceedings of SimVis 2006*, pages 305–318, 2006.

[72] R. Rost. *OpenGL Shading Language*. Pearson Education Inc., 2004.

[73] T. Schafhitzel, F. Rößler, D. Weiskopf, and T. Ertl. Simultaneous visualization of anatomical and functional 3D data by combining volume rendering and flow visualization. In *Proceedings of SPIE Medical Imaging 2007*, pages 650902 1–9, 2007.

[74] H. Scharsach, M. Hadwiger, A. Neubauer, and K. Bühler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *Proceedings of EG/IEEE-VGTC Symposium on Visualization (EuroVis) 2006*, pages 315–322, 2006.

[75] T. Schultz, N. Sauber, A. Anwander, H. Theisel, and H.P. Seidel. Virtual Klingler dissection: Putting fibers into context. *Computer Graphics Forum (Proceedings of EuroVis 2008)*, 27(3):1063–1070, 2008.

[76] P. Sereda, A. Vilanova, and F.A. Gerritsen. Automating transfer function design for volume rendering using hierarchical clustering of material boundaries. In *Proceedings of EG/IEEE-VGTC Symposium on Visualization (EuroVis) 2006*, pages 243–250, 2006.

[77] L. Serra, R.A. Kockro, C.G. Guan, N. Hern, E.C.K. Lee, Y.H. Lee, C. Chan, and W.L. Nowinski. Multimodal volume-based tumor neurosurgery planning in the virtual workbench. In *Proceedings of MICCAI 1998*, pages 1007–1015, 1998.

[78] T. Song, E.D. Angelini, B.D. Mensh, and A. Laine. Comparison study of clinical 3D MRI brain segmentation evaluation. In *Proceedings of IEEE Engineering in Medicine and Biology Society 2004*, pages 1671–1674, 2004.

[79] O. Sporns, G. Tononi, and R. Kötter. The human connectome: A structural description of the human brain. *PLoS Computational Biology*, 1(4):e42+, September 2005.

[80] J.S. Suri, D.W., J. Gao, S. Singh, and S. Laxminarayan. A comparison of state-of-the-art diffusion imaging techniques for smoothing medical/non-medical image data. In *Proceedings of International Conference on Pattern Recognition (ICPR) 2002*, pages 508–517, 2002.

[81] J.S. Suri, S.K. Setarehdan, and S. Singh. *Advanced algorithmic approaches to medical image segmentation: State of the art applications in cardiology, neurology, mammography and pathology.* Springer, 2002.

[82] T. Tasdizen, R. Whitaker, R. Marc, and B. Jones. Enhancement of cell boundaries in transmission micropscopy images. In *Proceedings of IEEE International Conference on Image Processing (ICIP) 2005*, volume 2, pages 129–132, 2005.

[83] P. Thévenaz, T. Blu, and M. Unser. *Image interpolation and resampling.* Academic Press, Inc., 2000.

[84] U. Tiede, T. Schiemann, and K.H. Höhne. High quality rendering of attributed volume data. In *Proceedings of IEEE Visualization 1998*, pages 255–262, 1998.

[85] C. Tietjen, T. Isenberg, and B. Preim. Combining silhouettes, surface, and volume rendering for surgery education and planning. In *Proceedings of EG/IEEE-VGTC Symposium on Visualization (EuroVis) 2005*, 2005.

[86] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision (ICCV) 1998*, pages 839–846, 1998.

[87] F.-Y. Tzeng, K.-L. Ma, and E. Lum. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization 2003*, pages 505–512, 2003.

[88] A. Vazquez-Reina, E. Miller, and H. Pfister. Multiphase geometric couplings for the segmentation of neural processes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2009*, pages 2020–2027, 2009.

[89] F. Vega Higuera, P. Hastreiter, R. Naraghi, R. Fahlbusch, and G. Greiner. Smooth volume rendering of labeled medical data on consumer graphics hardware. In *Proceedings of SPIE Medical Imaging 2005*, pages 13–21, 2005.

[90] I. Viola, A. Kanitsar, and M.E. Gröller. Hardware-based nonlinear filtering and segmentation using high-level shading languages. In *Proceedings of IEEE Visualization 2003*, pages 309–316, 2003.

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

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

[93] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH 1998*, pages 169–178, 1998.

[94] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, 1999.

[95] L. Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH 1978*, pages 270–274, 1978.

[96] B. Wilson, E.B. Lum, and K.-L. Ma. Interactive multi-volume visualization. In *Proceedings of International Conference on Computer Science 2002*, pages 102–110, 2002.

Curriculum Vitae

Personal

Name	Johanna Beyer
Date of Birth	October 16th, 1981
Place of Birth	Vienna, Austria

Contact

Address	Gablenzgasse 99/24 1150 Vienna, Austria
E-Mail	beyer@vrvis.at
Phone	+43 650 7574751

Education

10/2004	Ph.D. student in Computer Science, Vienna University of Technology, Austria
10/2000-07/2004	Dipl. Ing. (FH) in Medical Software Engineering, Upper Austrian University of Applied Sciences, Austria
08/1999-06/2000	Fred C. Beyer High School, Modesto, CA, USA
09/1991-06/1999	Secondary School, BG & BRG8 Albertgasse, Vienna, Austria
09/1987-06/1991	Elementary School, Zieglergasse, Vienna, Austria

Work Experience

03/2009-07/2009	Research Fellow, Initiative on Innovative Computing (IIC), Harvard University, USA
02/2008-	Researcher, Visualization Group, VRVis Research Center, Vienna, Austria
10/2005-02/2008	Junior Researcher, Medical Visualization Group, VRVis Research Center, Vienna, Austria
10/2004-10/2005	Research Assistant, Medical Visualization Group, VRVis Research Center, Vienna, Austria
09/2003-06/2004	Internship & Diploma Thesis, Simpleware Ltd, Exeter, United Kingdom
08/2001	Internship Telemed Communication Services, Innsbruck, Austria

Professional Activities

Conference Reviewing	IEEE Visualization Conference, EG/IEEE Symposium on Visualization, IEEE/EG International Symposium on Volume and Point-Based Graphics, International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Central European Seminar on Computer Graphics
Journal Reviewing	IEEE Transactions on Visualization and Computer Graphics, The Visual Computer, Computer Assisted Radiology and Surgery

Publications

Journal and Reviewed Conference Publications

- W.-K. Jeong, J. Beyer, M. Hadwiger, A. Vasquez, H. Pfister, and R. Whitaker. Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2009)*, pages 1505–1514, 2009.

- J. Beyer, M. Hadwiger, T. Möller, and L. Fritz. Smooth Mixed-Resolution GPU Volume Rendering. In *Proceedings of IEEE International Symposium on Volume and Point-Based Graphics (VG 2008)*, pages 163–170, 2008.

- J. Beyer, M. Hadwiger, S. Wolfsberger, and K. Bühler. High-Quality Multimodal Volume Rendering for Preoperative Planning of Neurosurgical Interventions. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007)*, pages 1696–1703, 2007.

- J. Beyer, M. Hadwiger, S. Wolfsberger, C. Rezk-Salama, and K. Bühler. Segmentierungsfreie Visualisierung des Gehirns für Direktes Volume Rendering. In *Proceedings of Bildverarbeitung für die Medizin*, pages 333–337, 2007.

- J. Beyer, C. Langer, L. Fritz, M. Hadwiger, S. Wolfsberger, and K. Bühler. Interactive Diffusion Based Smoothing and Segmentation of Volumetric Datasets on Graphics Hardware. *Methods of Information in Medicine*, pages 270–274, 2007.

Other Publications

- S. Wolfsberger, J. Beyer, M. Hadwiger, T. Czech, K. Bühler, and E. Knosp. 3D Visualization for Preoperative Planning of Approaches to the Sella. *8th Congress of the European Skull Base Society*, 2007.

Awards and Scholarships

Scholarships

- Marshall Plan Scholarship, 2008

- Leistungsstipendium, Vienna University of Technology, 2004/2005

Awards

- Medvis Award (Karl-Heinz-Höhne-Preis), second place, 2008

- Best Applications Paper Award, IEEE Visualization Conference 2007

- Best Student Project Award, Upper Austrian University of Applied Sciences 2003/2004