Eberhard-Karls-Universität Tübingen

Wilhelm-Schickard-Institut für Informatik / Graphisch-Interaktive Systeme

---

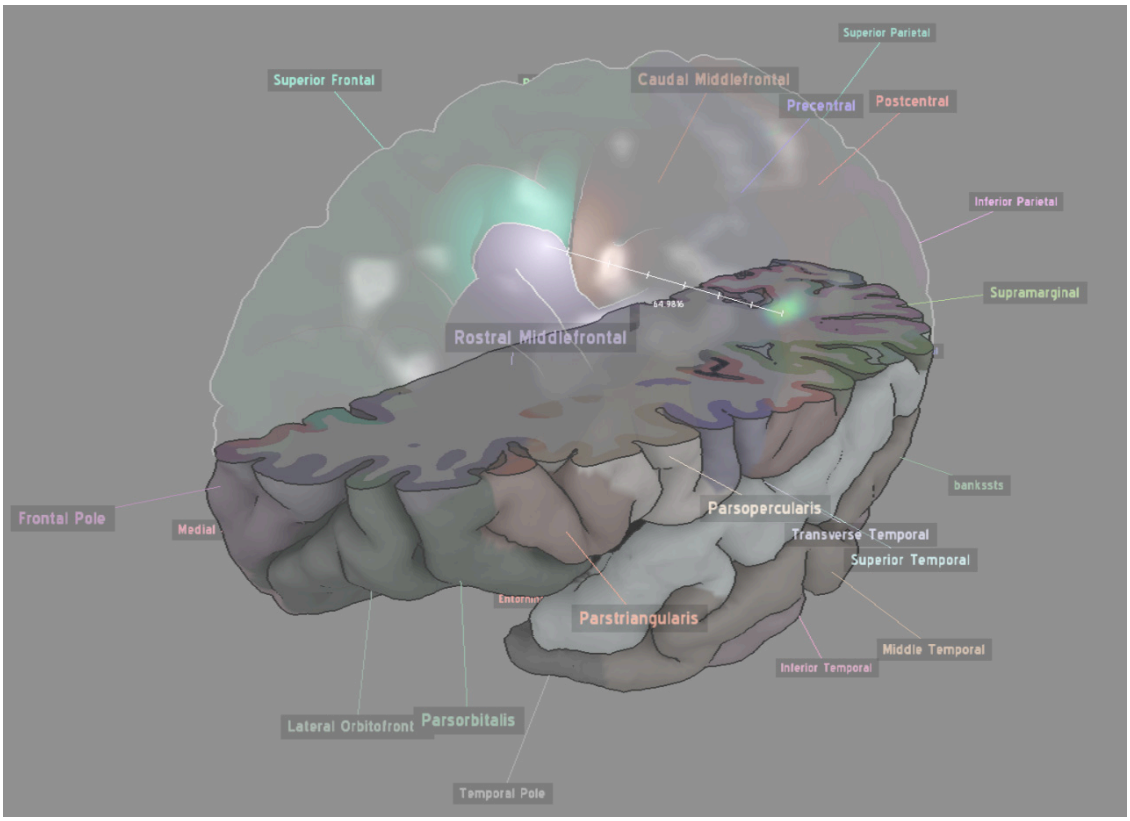
Illustrative Visualization of Brain Structure and Functional MRI Data

Werner Jainek

Oktober 2007

Betreuer: Dr. rer. nat. Jan Fischer

1. Gutachter: Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer
2. Gutachter: Prof. Dr. Hanspeter A. Mallot



http://jainek.de/projects/iBrain/

Acknowledgements

Front and foremost, I want to thank Dr. Jan Fischer for his outstanding commitment and his valuable advice and suggestions during our numerous discussions.

Further, I want to thank Prof. Wolfgang Straßer for his strong support throughout this thesis, and Prof. Hanspeter Mallot for his valuable insights into various neurological aspects.

My special thanks go to Silvia Born and Prof. Dirk Bartz for providing the MRI and fMRI data that has been used in this thesis.

Finally, I want to thank Christian Krämer and Tatjana Müller for proof-reading this thesis.

Contents

1 Introduction

1.1 Understanding the Brain

Understanding the relationship between brain activities and human behavior has been a long-standing goal of cognitive neuroscience. Ever since Gall has proposed in 1820 that all behavior emanates from the cortex, there has been an ongoing debate as to how the cortex is organized.[1] Gall himself postulated that the cortex is not a single organ, but is composed of several distinct organs that are each responsible for different kinds of personality traits, such as ideality, spirituality, hope, and so forth. Shortly after, this view has been challenged by Flourens, who took the completely opposite approach. Rather than strictly divide the cortex into separate organs, he argued instead that it has a uniform nature, where every part is able to perform exactly the same functions as the other parts.

In the following decades, support for both theories came from various experiments. For example, inflicting damage on the cortex of test animals in order to selectively disable certain traits was unsuccessful and therefore in favor of Flourens' theory. Other experiments, however, were able to trace sensory and motor functions to different parts of the cerebral cortex, which seemed in favor of Gall's theory.

It was mainly the work of Broca, Wernicke, Sherrington and Cajal that led to a new view of functional organization in the cortex which is central to our understanding of the brain today. This view integrates various aspects of both previously existing theories:

- Neurons are the basic signaling units of the brain.

- They are arranged in functional groups that are specialized to perform elementary processing operations (*functional specialization*). This is in contrast to Gall's theory, where functional groups are responsible for entire traits, not just for elementary operations. It is also in contrast to Flourens' theory, in which no specialization is assumed at all.

[1] The historical aspects presented here are described in more detail in [Kandel, 1991], [Savoy, 2001] and [Hirsch, 2006].

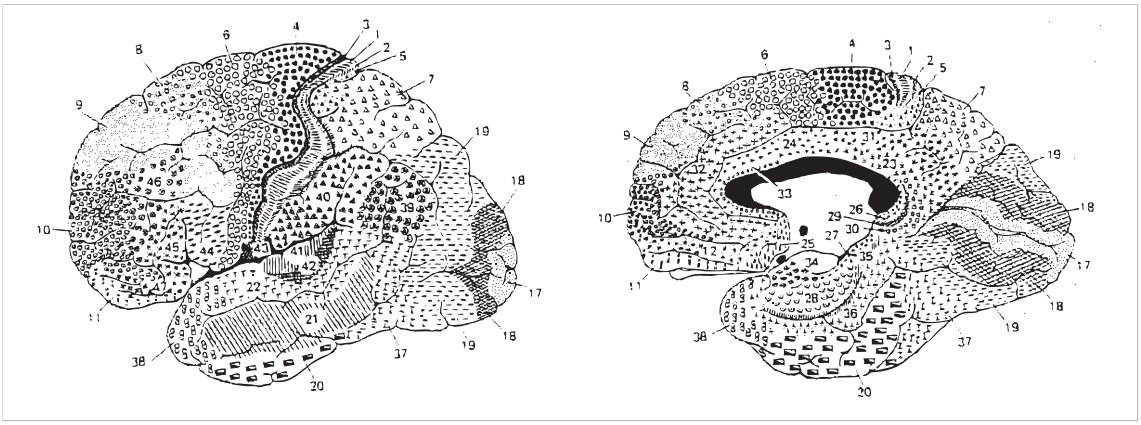


Figure 1.1: Different cortex areas identified by Brodmann. The classification is the result of comparing nerve cell structures from different parts of the cortex. (Adapted from [Brodmann, 1909])

- Neurons from different functional groups connect to one another in a precise fashion to build neural pathways that facilitate complex behavior (*functional integration*).

- The neural processing is organized into both serial and parallel pathways.

It was in this context that Brodmann started to examine cytological samples of the cortex in the early 20th century. He divided the cortex into distinct areas, based on their cytological differences. His findings are shown in Figure 1.1. Other than this, early mappings of cortex functions were mostly based on knowledge gained from patients that had lesions in different parts of the brain. In addition, direct neural stimulations of the cortex became a standard procedure in brain surgery to avoid impairment of important functional areas. The drawing shown in Figure 1.2 summarizes the knowledge obtained using these techniques up to the year 1957. Seen from today, it is remarkable how accurate that knowledge already was [Savoy, 2001].

One central problem with the data acquisition methods described before is that all gathered data stems from brains that are actually damaged. Being able to study healthy brains under controlled conditions is something that has been become possible only recently with the advent of new technological advances (see [Savoy, 2001] for a longer introduction into each technique):

- Transcranial magnetic stimulation (TMS) and transcranial electrical stimulation (TES) can be used to stimulate cortex areas without performing surgery. Also, TMS can create temporal lesions to disable certain functional regions.

Figure 1.2: Knowledge about functional specialization in the cortex as of 1957. (Image courtesy of [Polyak, 1957].)

The drawbacks of this technique are its low spatial resolution and safety concerns due to the danger of induced seizures.

- Electroencephalography (EEG) and magnetoencephalography (MEG) are non-invasive recording methods for brain activity that have an extremely high temporal resolution. Electrical currents inside the brain are measured on the brain surface either directly, or though the magnetic fields they produce. Since the measurements happen on the brain surface, this technique suffers from a low spatial resolution.

- In positron emission tomography (PET), a short-lived radioactive tracer (e.g., O^{15}) is administered to the subject. Radioactive decay leads to the emission of positrons, which in turn collide with nearby electrons shortly after to produce two opposing gamma rays. When the subject is placed in a scanner, both rays will simultaneously hit opposing parts of the device. This information is collected over a time frame of 30 to 90 seconds and is then used to reconstruct a three-dimensional activation volume. Due to the involved radiation, the number of tests on subjects are limited. Also, the long time it takes for one scan to complete limits the experiments to block designs. (Block designs will be explained later.)

- Functional magnetic resonance imaging (fMRI) is an adaptation of magnetic resonance imaging (MRI), where small magnetic field inhomogeneities are used as an indication of brain activity. While the details are explained in Ap-

pendix A, it can be noted that this technique has significant advantages over the ones presented before. In contrast to EEG/MEG, it has a higher spatial resolution, albeit a lower temporal resolution. Still, the temporal resolution of about 20-30 scans per minute is much higher than the 0.5 to 1.5 scans per minute that PET provides. Also, safety concerns are very minimal. So, with fMRI, it became possible for the first time to run *repeated* experiments with moderate spatial *and* temporal resolution, allowing the implementation of various test paradigms.

1.1.1 Brain Mapping Today

The amount of research in brain mapping has increased significantly since the inception of PET, MRI and especially fMRI (see [Fox, 1997] and [Bandettini, 2007]). Today, fMRI is used in many different areas, such as neurosurgery, psychiatry, the studying of memory in aging and dementia, the study of language systems, epilepsy studies, studies of visual pathways, studies of the auditory cortex, pain perception and many more. [Faro and Mohamed, 2006, Baert et al., 2000, Pekar, 2006]

Not only the number of applications, but also the applied methods have been and still are constantly improving: fMRI scanners gain higher temporal and spatial resolutions; new experiment paradigms apart from block- and event-related-designs (described later) are developed; various fMRI pulse sequences have been devised to not only measure the BOLD effect (see Section 2.2) but also diffusion coefficients, temperature, changes in the blood volume (VASO), changes in perfusion (ASL), among others. Finally, there is significant research in fMRI post-processing methods. While massively univariate analyses (e.g., statistical parametric mapping, see Section 2.2) are still very common, new methods such as multivariate analysis, machine learning and pattern classification are being adapted. See [Bandettini, 2007] for a longer overview of these techniques.

The unprecedented scientific activity in the area of brain mapping is driven by the desire to understand what brain regions are involved in different behaviors on the one hand, and by the technological advances on the other hand.

1.1.2 A Multitude of Data / Visualization Requirements

While the possibilities of gathering multi-modal data are growing, there is a need to integrate the different modalities into a coherent visualization. For example, when performing EEG, MEG, PET or fMRI studies to determine brain activation sites, it is common to also perform structural MRI scans that will act as a frame of reference for the activation data. Both types of scans (activation and structural) represent complex three-dimensional data that, in their simplest form, are displayed

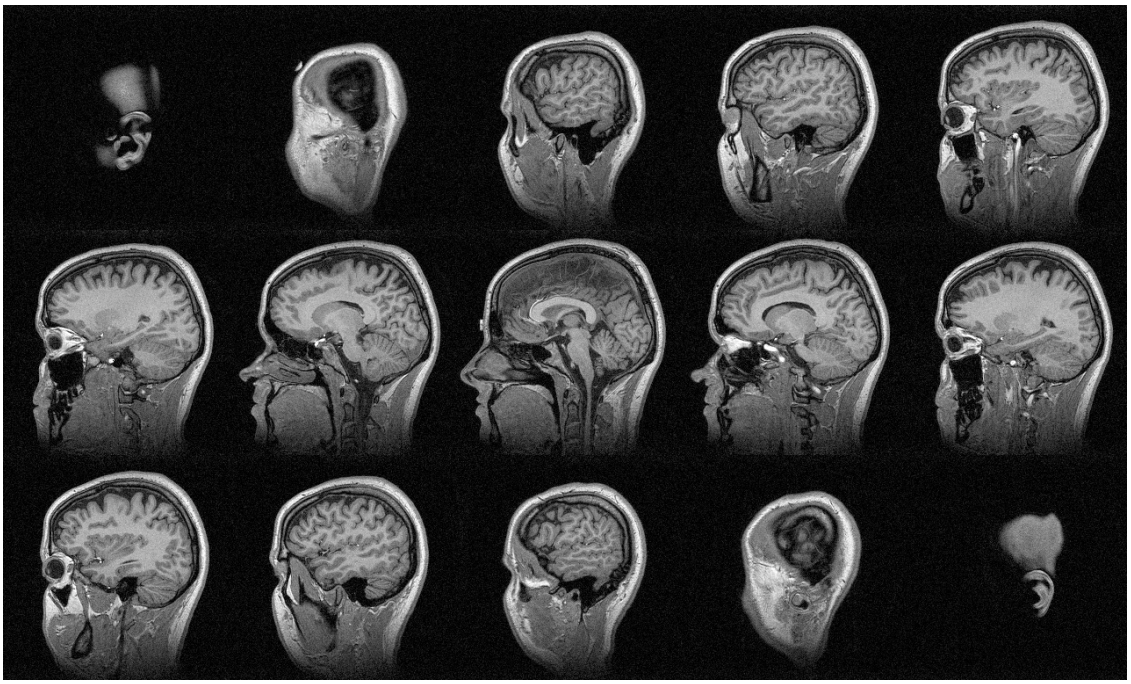


Figure 1.3: MRI scan of the author. Only 15 out of a total of 128 slices are shown.

in the form of two-dimensional slices (see Figure 1.3). The activation data is usually overlaid on top of the structural data as can be seen in Figure 1.4.

While some scientists consider this the only visualization that properly represents the "true" data and resent the idea of further post-processing, it is clear that it has several disadvantages:

- The most obvious shortcoming is that the three-dimensional structure — even for a single data set — is not easy to grasp by simply looking at the two-dimensional slices. It is further complicated by the fact, that the distance between two slices (in world units) does not necessarily have to be equal to the distance between two pixels (see Appendix A).
- The previous point is especially true for the highly complex three-dimensional folding structure of the cortex.
- The more additional structures such as tumors or blood vessels are displayed on the two-dimensional slices, the harder it becomes to integrate this data into a coherent understanding of the three-dimensional structure.
- Interaction of the observer with the displayed data is very limited. For example, it might be beneficial to cut the volume into slices that are not orthog-

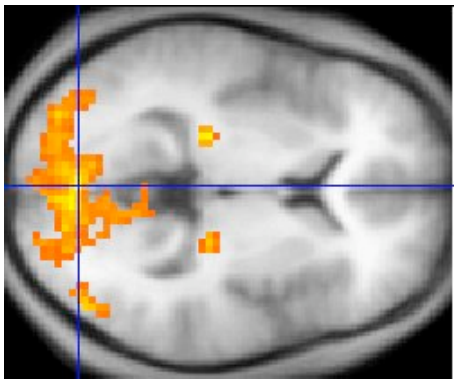


Figure 1.4: Processed fMRI data displayed on top of a structural MRI scan. The blue cross denotes the site with the highest activation. (Image taken from [Wikipedia, fMRI].)

onal to one of the main axes (i.e., the resulting view is not saggital, coronal or axial). Specifying this slice orientation, however, by simply looking at the two-dimensional slices is not intuitive.

The inherent three-dimensional complexity of the data is undoubtedly more properly captured by a three-dimensional visualization. By further providing the observer with various interaction tools that change the appearance of the visualization in real-time, better exploration of different aspects of the data that are of interest becomes possible.

In the past, real-time three-dimensional display of such data has been very restricted due to technical limitations. With todays hardware advances, however, the technological grounds are laid out for new and creative visualizations. These technological improvements will be described in the next section.

1.2 Modern Computer Graphics

In the beginning, 3D graphics accelerator cards had a fixed functionality. Their pipeline implemented various aspects of a typical rendering system in hardware, such as lighting and transformation of vertices, rasterization of triangles, depth buffer checking, and so forth [Woo et al., 1997]. To achieve different render results, some inputs to that pipeline were adjustable by the user, e.g., the light color, the type of shading that was to be used, etc. Injecting custom code into that pipeline, however, was not possible. On the negative side, this meant that there was only a limited number of visual styles that could be rendered. The positive side was, of course, the provided speed.

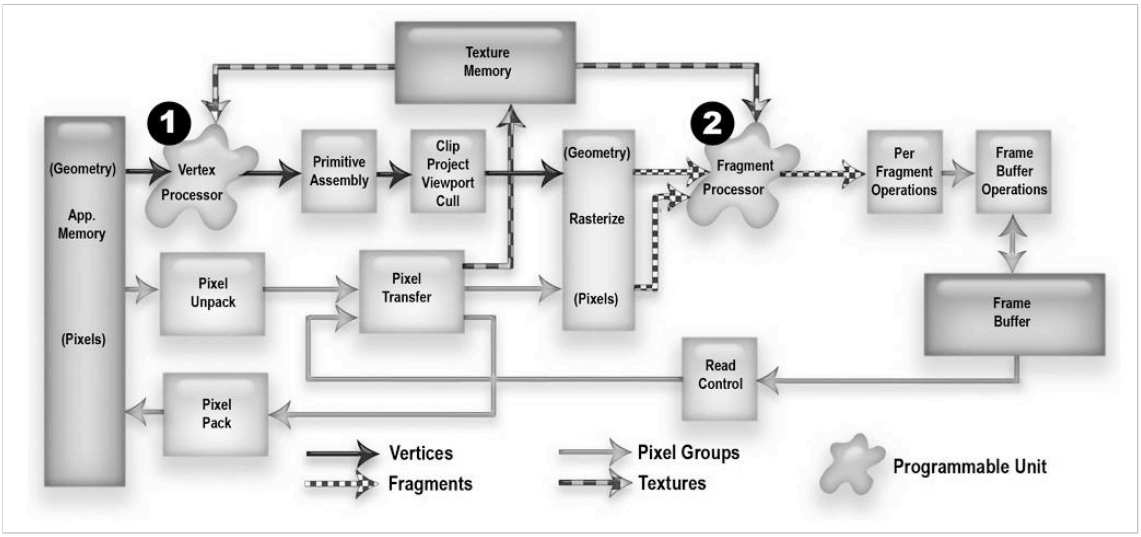


Figure 1.5: Programmable graphics pipeline. Two key stages of the pipeline (denoted by 1 and 2) are programmable. (Image courtesy of [Rost, 2006].)

1.2.1 The Programmable Graphics Pipeline

As graphics cards got faster, more and more features were added that could be adjusted by the user to diversify the rendering output. Finally, recent generations of graphics cards offer programmability of key stages in the rendering pipeline, providing a standardized programming language: The OpenGL Shading Language (GLSL) [Rost, 2006]. Figure 1.5 shows the two key stages in the pipeline that have been replaced by programmable units: the vertex processor and the fragment processor. The programs that are uploaded to the graphics card are typically called *shaders*. Newer graphics cards developments are pointing to ever more programmability of the pipeline (e.g., trough geometry shaders [Brown and Lichtenbelt, 2007]) on the one hand, and to more uniformity of the pipeline on the other hand, i.e., all stages of the pipeline can be programmed with the same language and different stages are in fact executed by the same generic processing units on the graphics card.

This new flexibility has led to a renaissance of custom rendering algorithms and visual outputs:

> "No longer restricted to the graphics rendering algorithms and formulas chosen by hardware designers and frozen in silicon, software developers are beginning to use this programmability to create stunning effects in real time." *Randi J. Rost in OpenGL Shading Language*

1.2.2 Non-Photorealistic Rendering

One main goal of the graphics community has always been to render synthetic images that are indistinguishable from real scenes. For this, various global illumination algorithms [Dutre et al., 2002] have been devised that approximate the equation of transfer through the use of various simplifying assumptions (see Appendix B), e.g. radiosity [Sillion and Puech, 1994], "classical" raytracing [Whitted, 1980], path tracing [Kajiya, 1986] and photon mapping [Jensen, 2001]. Some of these algorithms have now been adapted to run on 3D graphics cards (see [Coombe et al., 2005] and [Purcell et al., 2005]) and there are even successful efforts to build custom raytracing hardware [Schmittler et al., 2002].

Recently, however, there has been growing interest in a field called *non-photorealistic rendering*, or NPR in short. For good overviews, see [Gooch and Gooch, 2001] and [Strothotte and Schlechtweg, 2002]. The goal here is not to render physically accurate pictures of a scene, but stylized versions thereof. These styles range from halftoning, ink and line drawings to pencil and charcoal drawings and to painterly drawings (see [Sousa, 2003] for a broad overview). The general motivation of using such rendering styles (other than their novelty and visual appeal) is to present the scene in such a way that the information important for the application at hand is optimally conveyed to the observer. As such, these techniques have been successfully applied to various fields such as architecture , technical drawings, archeology and anatomy [Sousa, 2003].

With the advent of programmable graphics hardware, many NPR techniques have been implemented on the GPU (graphics processing unit) in order to provide interactive, stylistically rendered scenes [Sousa, 2003]. When integration of computer generated content with real world footage is desirable, NPR techniques can be applied to seamlessly blend both sources into an augmented reality scene [Fischer et al., 2007].

1.2.3 Endless Possibilities

As the trend of allowing more programmability of the GPU continues, the graphics card more and more becomes a general co-processor, or *stream processor* (see [Buck et al., 2004]) composed of many small processing units working in parallel. The implementation of complex pipelines, where the output of one render pass is stored inside a texture and is then used as an input for the next render pass, can be regarded as technical details of a high-level algorithm. These algorithms do not necessarily have to be render algorithms. In fact, the area of general purpose GPU programming (GPGPU) already uses such complex pipelines to implement general

algorithms such as sorting [Purcell, 2005], fourier transformations and dynamics simulations [Luebke et al., 2004] to name just a few.

Abstraction from the details of the algorithm implementation (i.e., the exact render passes and output textures) will finally benefit regular render algorithms as well [Ragan-Kelley et al., 2007], as continuous re-implementations of the render pipeline infrastructure becomes unnecessary and therefore, development time is shortened.

The render pipeline in this work is already developed with this mindset. Output textures are simply treated as temporary "variables" that store intermediate computation results and are combined through a series of passes to produce the final image.

1.3 The Goal of this Thesis

Today, there is an unprecedented amount of research in the field of human brain mapping. As described in Section 1.1, one of the main tools in this endeavor is magnetic resonance imaging (MRI) and its specialization, functional magnetic resonance imaging, or fMRI for short. After the data gathered with these tools has been processed, there is a need to visualize the results in an intuitive and uncluttered manner.

At the other end, programmable computer graphics hardware has reached a technical stage of development that allows the implementation of highly complex rendering algorithms with real time output. In particular, rendering styles are no longer limited to "classical" shading but allow for more abstract and illustrative visualizations. A few examples have been given in Section 1.2.

In this thesis, a new visualization style is developed. It aims at conveying both structural and activation information about the brain at the same time. It does so by means of an illustrative rendering style that is implemented on the graphics card to achieve a real time rendering output. This way, the user is presented with an interactive scene that can be explored and manipulated with real time feedback.

There are unique challenges in trying to display both MRI and fMRI data together in a three-dimensional visualization while keeping it clean and uncluttered at the same time. These challenges have been approached before, e.g., in [Stokking et al., 2001] and [Schafhitzel et al., 2007]. The ideas behind the approach taken in this work will be described in full detail in Chapter 3: Finding the Right Visualization.

First, however, the next chapter will give an overview of the pre-processing steps applied to the MRI and fMRI data that is used in this work.

2 Brain Analysis Pipeline

The technical details of the acquisition process for MRI and fMRI data are very complex. A general overview is given in in Appendix A. After the data has been acquired, it has to be processed in order to extract the information that is needed in the context of the developed visualization pipeline. This processing is performed by using two existing software packages:

- FreeSurfer is used to extract information from the MRI data (see below for references).
- SPM is used to process the fMRI data in order to retrieve brain activation patterns (see below for references).

2.1 MRI Data and FreeSurfer

The different algorithms that are used in FreeSurfer have been published in several papers: [Segonne et al., 2004], [A.M.Dale et al., 1999], [Fischl et al., 1999], [Fischl et al., 2001], [Fischl et al., 2004], [Fischl et al., 2002]. Here, only a broad outline is given.

The FreeSurfer pipeline is divided into two processing streams: a surface based stream and a volume based stream. The surface-based stream extracts the meshes of the gray and white matter boundaries. The volume based stream performs the segmentation of the brain volume into anatomical regions (white matter, gray matter, hippocampus, etc.). Further, the gray matter is parcellated into cortex regions based on gyri.

The *surface based processing stream* is visually summarized in Figure 2.1. It consists of the following stages:

- *Talairach Registration* — The volume is registered with Talairach space [Talairach and Tournoux, 1988]. Although the coordinate space defined by Talairach & Tournoux is still used, the original brain used in their atlas has been replaced, since it did not present a good average of the general population [Brett et al., 2002]. Instead, the MNI standard brain [Brett, 2002] that is the result of registering and averaging the brains of 152 individuals is used in most applications.

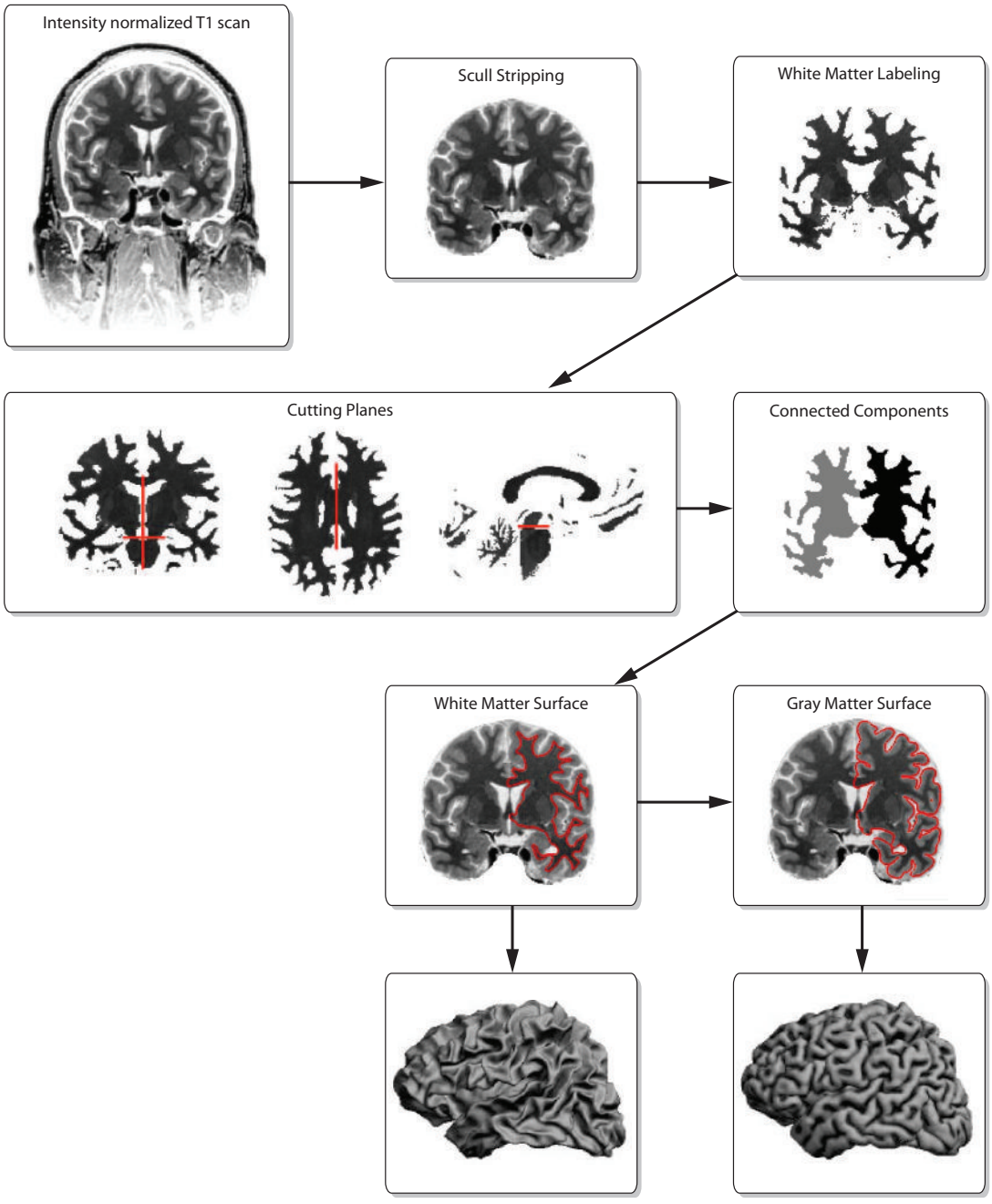


Figure 2.1: Surface based processing stream as implemented in FreeSurfer. (Individual images courtesy of [A.M.Dale et al., 1999].) Note that the intensities in all images except for the final two have been inverted for display purposes.

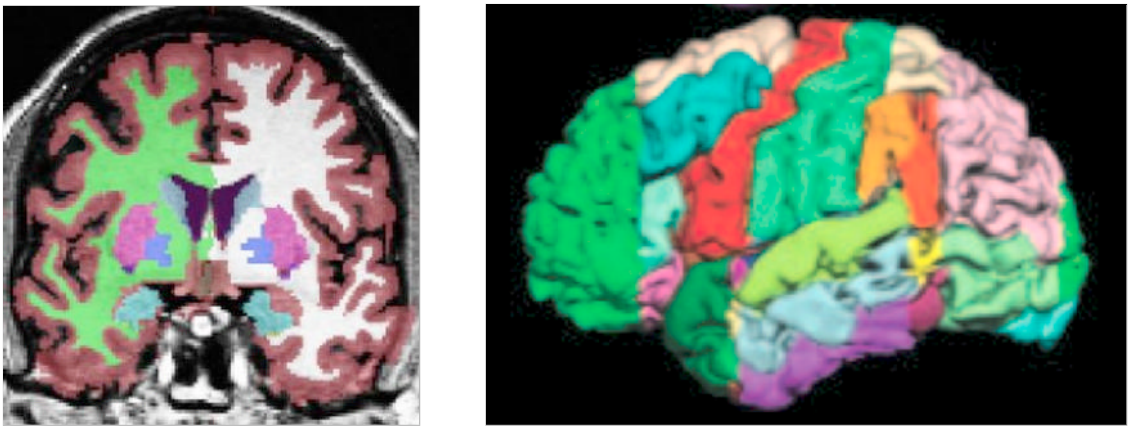
- *Intensity Normalization* — T1-weighted scans contain artifacts due to magnetic field inhomogenities (since they are not accounted for like the T2-weighted images using EPI, see Appendix A). These inhomogenities causes the same tissue type (e.g., white matter) to have different intensities across the volume. This is corrected by estimating the bias introduced by the magnetic field using white matter intensities.
- *Skull Stripping* — The skull is removed from the data set by means of hybrid algorithm as described in [Segonne et al., 2004]. First, a watershed segmentation is performed (see Section 8.3.2). Then, a deformable template model is used to correct inaccuracies of the previous step.
- *White Matter Labeling* — Voxels that represent white matter are determined through their intensities and neighbor constraints.
- *Cutting Planes* — The cortex hemispheres, the cerebellum and the brain stem are separated by finding appropriate cutting planes.
- *Connected Components* — The white matter voxels belonging to each cortex hemispheres are classified with a connected components procedure. (This is possible since the white matter represents a connected mass).
- *Surface Tesselation, Refinement and Deformation* — First, a white matter surface is constructed for each hemisphere by tesselation of the previously classified white matter voxels. This tesselation is then smoothed using a deformable surface algorithm that is guided by local MRI intensity values. The resulting white matter surface is then expanded to produce the cortex surface.

The *volume based processing stream* first performs affine registration of the volume with talairach space. Then, an initial volumetric labeling is performed and the magnetic field inhomogenities are corrected.[1] In the next step, a high dimensional nonlinear alignment of the volume to the actual Talairach atlas is performed. After that, both the cortical and the subcortical voxels are labeled according to the atlas and further refined. Results for the subortical and cortical labeling can be seen in Figure 2.2 (a) and (b), respectively.

2.2 fMRI Data and SPM

The fMRI data used in this work has been gathered through a BOLD T2* EPI scan and then processed using SPM. Note that SPM denotes both a technique *and*

[1] As described by Fischl, the Talairach registration and the correction of inhomogenities uses different algorithms than the surface based processing stream, since they evolved separately from each other.



(a) (b)

Figure 2.2: (a) Subcortical segmentation. (b) Cortical segmentation. (Images taken from the [Free Surfer Wiki, 2007].)

a software package for Matlab [SPM, Web] that is used for the processing in this work. Appendix A explains what a T2* EPI scan is. Further introductions and explanations of these topics can be found, e.g., in: [Kim and Bandettini, 2006], [Aguirre, 2006] and [Friston et al., 2004]. Only a basic overview of important concepts is given here.

Haemodynamic Response to Neural Activity. When neurons in the brain become active, oxygen in their surrounding is consumed. As an immediate effect, the veneous blood oxygen level drops in those areas. Shortly after (2-3 seconds), this is compensated by an increased blood-flow in these areas and the blood oxygen level raises again. The increased blood-flow does not match the actual utilization of oxygen, so the blood oxygen level raises above the baseline. When the blood-flow decreases again, so does the oxygen level until it reaches the baseline again (after about 20 seconds after the first activation, [Haeger and Ress, 2002]). A schematic picture of the oxygen level changes as a function of time is shown in Figure 2.3. This function is called the *haemodynamic response function*, or HRF. It is important to note that [Logothetis et al., 2001] have shown that the utilization of oxygen actually correlates with synaptic activity of the neurons and is therefore a good measure of brain activity.

Blood-Oxgen-Level Dependent (BOLD) fMRI. When put into a strong magnetic field of a MRI scanner, the magnetic properties of oxyhaemoglobin and deoxy-

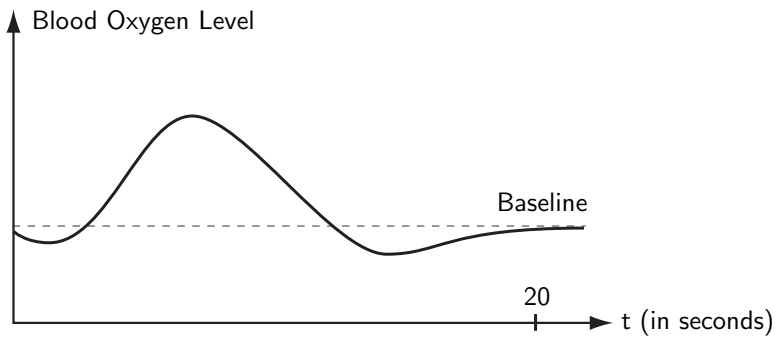


Figure 2.3: The haemodynamic response function. When a brain area becomes active, the oxygen level drops at first. It is quickly compensated by increased blood flow which actually leads to an increased oxygen level. Finally, the blood oxygen level returns to the baseline after approximately 20 seconds.

haemoglobin are different. While oxyhaemoglobin leves the magnetic field intact, deoxyhaemoglobin creates small local magnetic field inhomogenities. Therefore, if the oxygen level in the blood drops, stronger magnetic field inhomogenities can be registered. When the oxygen level raises, the magnetic field becomes more homogeneous. Using a T2* EPI scan (see Appendix A), these inhomogenities can be detected over time (where each scan of the entire brain takes about 2-3 seconds). The recorded signal is called the *blood-oxygen-level dependent fMRI* signal. Figure 2.4 shows the recorded signal for one time point.

Test Paradigms: Block Design. Simple *block design* paradigms present the subject with one type of stimulus (A) for an extended period (block) of time and record the respective BOLD fMRI signal. Then, another stimulus (B) is presented in another block and the signal is recorded as well. These two blocks are alternated for the length of one trial (e.g., ABBAABA...). In the end, the recorded signals for both stimuli are averaged and compared to one another, e.g., by subtracting one average from the other (A-B). Finally, it can be determined if there is a significant difference between both stimuli for each part of the brain. One of the drawbacks of this technique is that there is no randomization of stimuli, so they become predictable by the test subject.

Test Paradigms: Event-related Design. In *event-related design* paradigms, two or more types of stimuli can be interleaved arbitrarily and presented to the subject. The exact stimulus onset times (SOTs) of for each type of stimulus (A, B, ...) are recorded separately. Given that a specific region inside the brain reacts to one type of stimulus, the expected behavior of this region over time can be predicted

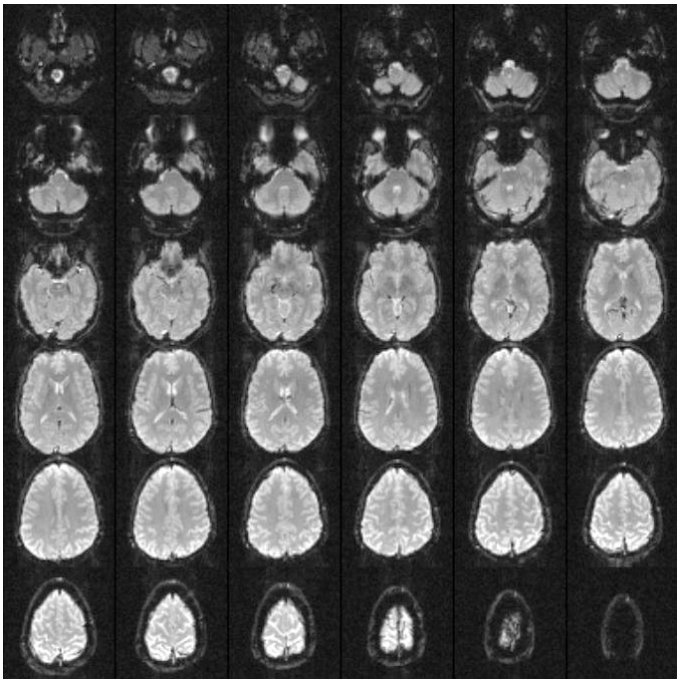


Figure 2.4: Recorded BOLD fMRI signal of the entire brain volume from one of the data sets that is used in this work.

by convoluting the HRF with the SOTs of that stimulus. Different brain regions can then be tested against these expected behaviors. This is achieved by using a technique called SPM.

Statistical Parametric Mapping (SPM). This technique is described in detail in [Friston et al., 2004]. The general idea is to estimate the effect of different stimuli on different brain voxels individually. For each stimulus type i, a function f_i is computed that is the convolution of the stimulus onset times for stimulus i with the haemodynamic response function. Then, for each voxel, the linear combination of these functions that approximate the recorded signal s with the least error e is determined. If the signals are written as vectors, the following equation:

$$\mathbf{s} = \mathbf{F} \cdot \mathbf{b} + \mathbf{e}$$

is called the *general linear model* (GLM) that has to be solved for $\mathbf{b}$. Here, $\mathbf{F}$ is a matrix whose columns are composed of the functions f_i. Typically, there are more columns than stimuli in the experiment, since both inaccuracies in the stimulus onset times and general brain activity changes have to be accounted for. After the GLM has been solved for $\mathbf{b}$ with a minimal error $\mathbf{e}$ (in a standard least squares sense) for every voxel of the brain independently, inferences based on statistical tests (e.g.,

T- or F-tests) can be made for every voxel. This allows to test different hypothesis about the interaction of the presented stimuli. In the end, brain activations maps can be created. Since each voxel is processed individually, this approach is called *massively univariate*.

Data preprocessing. For SPM to yield correct results, several pre-processing steps have to be performed on the fMRI data: realignment, spatial normalization and spatial smoothing. These steps are described in detail in [Friston et al., 2004]. Further, the fMRI data is co-registered with the anatomical MRI scan, so that the activation patterns can be correctly overlaid on the structural data (see Figure 1.4).

Part I

Visualization Pipeline

3 Finding the Right Visualization

3.1 Choosing a Visualization Style

As mentioned in the introduction, visualizing both the structural brain data and the activation data at the same time is a unique challenge. This is because many different aspects of the data have to be communicated to the user in order to provide a meaningful visualization. Important aspects include

- the shape of the brain surface,
- the extent of different cortex areas,
- the cortex folding structure and thickness,
- structures inside the brain, e.g., the hippocampus, the amygdala, etc.,
- and various activation areas as computed from the fMRI data.

Each of these structures is defined by highly complex three-dimensional data that possibly overlaps the other structures. Even if only a subset of these structures had to be visualized, doing so while keeping a clear and concise output would already be a very difficult task. The main reason for this is that during the visualization process, the three-dimensional data is projected on a two-dimensional surface, thereby discarding an entire dimension. The human observer then has to reconstruct the three-dimensional scene in order to understand the shape and relation of the presented objects. This problem is exactly what the field of computer vision tries to solve with various algorithms, and the tremendous amount of research in this area is a good hint at the inherent complexity of the matter. In fact, in and on itself, reconstructing a three-dimensional scene from a two-dimensional projection is an ill-posed problem and can only be solved in an unambiguous way by using prior knowledge about the scene [Ma et al., 2003].

Luckily, the human brain excels at this task. It uses many different visual clues and prior knowledge in order to processes the two-dimensional projection of the environment on the retina and to reconstruct the three-dimensional scene [Mallot, 1998]. One such clue would be the shading of the projected objects under the current lighting condition; another one would be the texture and materials of

those objects. Even when the visual input is limited to only one type of visual clue (e.g. surface contours), it can already be enough for the human observer to derive the three-dimensional shape of the presented objects to a certain degree [Ullmann, 2000]. Therefore, the approach taken in this thesis is to integrate the multiple complex three-dimensional brain data as outlined above into one visualization style by using different visual clues for some of these data sets.

Defining the Visualization Style

Each of the visualization requirements outlined at the beginning of the chapter are now revisited, trying to find the right visual clues to convey the respective information:

- *Background* — Choosing the background color can be regarded as defining the lighting conditions of the scene. A very bright background corresponds to a brightly lit scene and successive visualizations are limited to subtracting brightness from the previous steps. A very dark background, on the other hand, corresponds to a faint lighting of the scene and successive visualizations can only add brightness to the scene. This is why a medium gray background color will be used in this thesis. It allows for both the addition and subtraction of brightness by the various parts of the visualization pipeline.

- *Brain Surface* — The brain surface is important for all other displayed structures, since they all reside inside the volume it defines. Put differently, the brain surface acts as a context, or frame of reference for the other parts. For this reason, the surface should always be visible to some extent. At the same time, however the chosen visualization has to allow for the other structures to be blended on top of it (e.g., the fMRI activation areas). This is why the style of the brain surface should not particularly stand out but should instead be modest. Using *outlines* and *contours* for the brain surface meet both criteria, as they are both unobtrusive and leave large areas of the surface completely transparent.

- *Surface Areas* — In order to distinguish the different brain areas on the surface, the space between the outlines is filled with colors, using different colors for different areas. Just adding uniform colors, however, counteracts the impression of a three-dimensional surface that has been created by using contours alone. So, a little *shading* is also added to the surface, *subtracting brightness* and sacrificing full transparency in certain parts.

- *Cortex Folding Structure* — The brains folding structure is highly complex. Displaying this information for the entire brain at once can be very distracting and overwhelming. So instead, this information has to be explored by the user, i.e., a clipping plane is introduced that intersects the brain volume and

the full folding information is shown only on the two-dimensional cut. By adjusting the clipping plane, the user can explore the entire folding structure.

In order to provide a better sight of the clipping plane, the surface parts above it (both the silhouettes and the shading) will be drawn almost completely transparent. To keep the frame of reference, the outer contour of the surface will still be drawn very prominent.

- *Inner Brain Structures* — Although inner brain structures such as the hippocampus or the amygdala, or even tumor sites, will not be included in this work, they can be easily integrated on the previously described clipping plane. Also, drawing three-dimensional semi-transparent isosurfaces of one or two of these structures is probably possible without introducing too much complexity in the final output.

- *Activation Areas* — All previous visualization steps have mostly darkened the final output. This is actually very important, since it allows for the use of bright colors to display the fMRI activation areas.

3.2 Deriving a Visualization Method

After the desired outcome has been specified in the previous section, this section describes the technical constraints and decisions that were made in order to achieve that output.

Volumes or Meshes?

The brain surface described above will be rendered using a mesh. Although NPR rendering techniques for volume rendering have been devised (e.g., [Csebfalvi et al., 2001], [Treavett and Chen, 2000]), the ones available for meshes are far more numerous. Also, meshes are able to produce crisp output even at high resolutions, whereas volumes are limited by the resolution of the volume texture. Finally, meshes can be rendered much faster on current GPUs.

Whenever a two-dimensional boundary inside a three-dimensional volume can be easily defined, as is the case for the brain surface, it is probably best to use meshes to render that boundary. In the case of fMRI activations, however, such a surface is not easy to define. An iso-surface could be created for such a volume, but by doing so, important information about the spatial extent and decay of the activation is lost. This is why volume rendering will be used to display the various activation areas.

Mixing Non-Photorealistic Rendering of Meshes with Volume Rendering

Many non-photorealistic rendering techniques require access to information about neighboring pixels on the screen in order to determine the current pixel's value. In order to implement these techniques (and the technique used in this work), it is common to first render the entire scene into a texture and to then use that texture in a second render pass to access neighboring pixel values. This approach, however, makes is hard to interleave volume and NPR mesh rendering in a single render pass. Therefore, in this work, the entire scene is split up into different layers which are rendered in individual render passes and are later composed to produce the desired output. Before these render layers are described, it is important to note that the volume rendering can be arbitrarily split up into many smaller segments as long as emission and absorption values are stored for each segment (see Appendix B for more details).

Render Layers

To simplify the following derivation, it is assumed that the scene camera uses an orthographic projection. Switching to a perspective projection, however, does not invalidate any of the made assumptions.

Figure 3.1 shows a schematic two-dimensional cut through a scene. In that figure, the camera is placed in the upper right corner. Its viewing direction is denoted by the arrows. The camera viewing volume is also shown. This is the part of the scene that will be rendered into the framebuffer. At the far end of that viweing volume (as seen from the camera) lies the far clipping plane. It is chosen so that all objects in the scene are closer to the camera than the plane itself. (Note, however, that it is not always the case that the entire scene geometry is contained in the camera viewing volume as depicted in Figure 3.1.) The scene itself is composed of a brain surface mesh and a volume that holds the fMRI activation areas. The volume bounds are shown as a circle here (a sphere in three dimensions) even tough the volume will later reside inside a cube. For the current context, the exact shape of the volume is irrelevant. Also, since the volume holds fMRI activation areas, it can be assumed that all non-empty parts of the volume actually lie within the bounds defined by the surface mesh. Finally, the scene is divided by a clipping plane. Only the part of the clipping plane that falls inside the camera viewing volume is shown.

This scene is now split up into different render layers, using the surface mesh and the clipping plane as a guide. The first two render layers are shown in Figure 3.2. For the first layer, Lower Surface, the upper part of the surface is clipped away and only the lower part is rendered. This results in an image where only those parts of the surface are seen that are marked with the thick blue lines. The depth

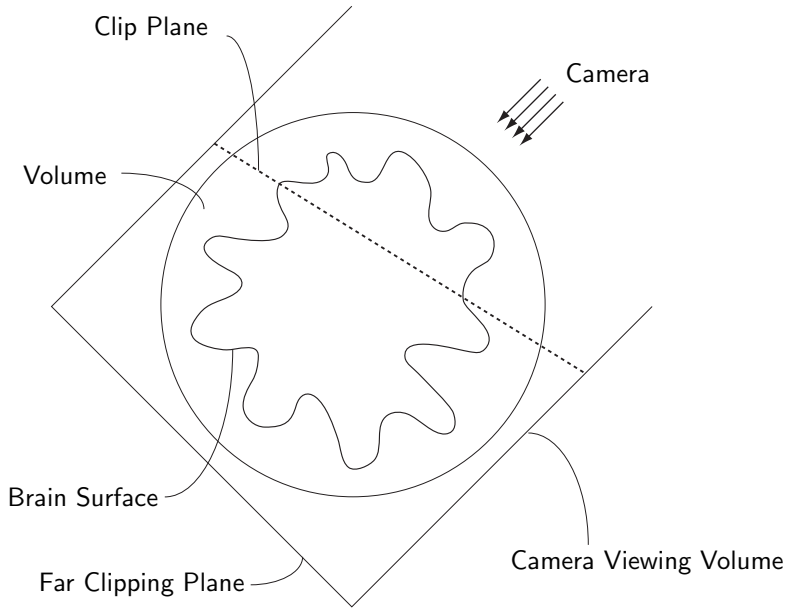


Figure 3.1: Schematic display of the scene.

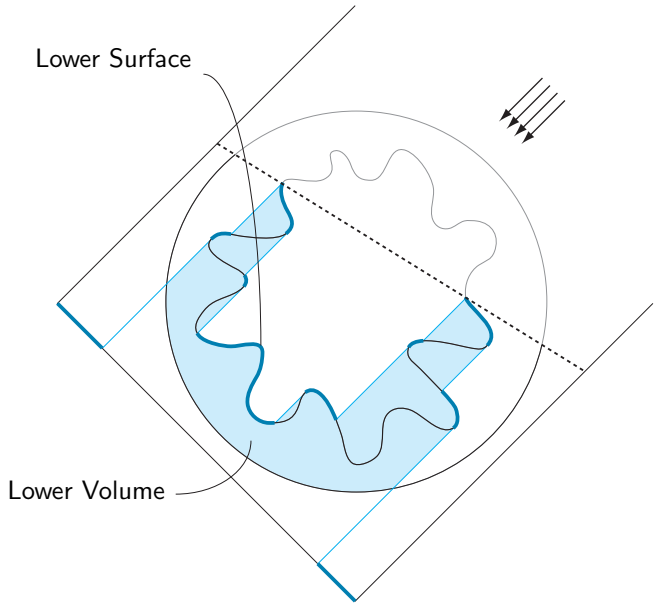


Figure 3.2: The upper part of the surface is clipped away. Only the lower part of the surface is rendered. The depth of this surface as seen from the camera is then used in the volume rendering pass to draw only the the parts that are shaded in blue.

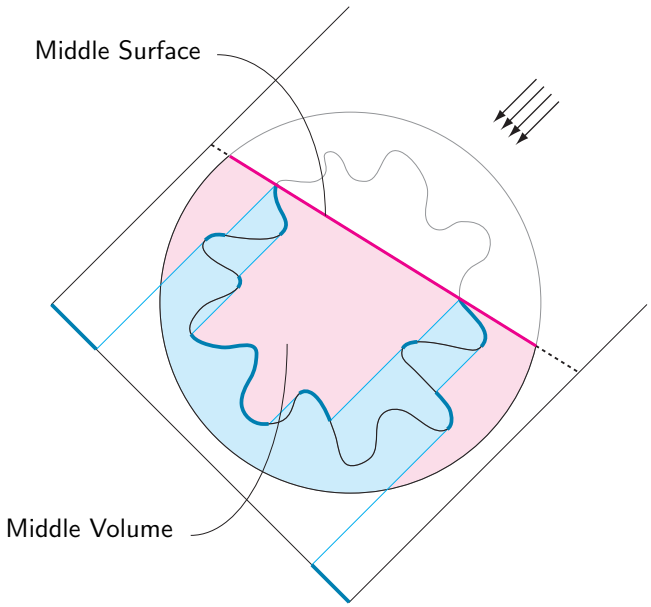


Figure 3.3: The Middle Volume is defined by the thick blue lines on the far side and by the clipping plane on the near side. The Middle Surface is simply the clipping plane itself.

of these surface areas as seen from the camera can be used to render the parts of the volume that lie behind these areas. This is the second layer, LOWER VOLUME, and is shaded in light blue in Figure 3.2. Note that when actually composing these layers together, LOWER VOLUME has to be drawn before LOWER SURFACE.

The next two layers are depicted in Figure 3.3. The first, MIDDLE VOLUME, draws all parts of the volume that lie both *above* the thick blue lines and *below* the clipping plane. The next layer, MIDDLE SURFACE draws the clipping plane itself (denoted by the thick red line).

Finally, the last two layers are shown in Figure 3.4. The first is UPPER VOLUME, that draws the entire volume that lies above the clipping plane. The second is UPPER SURFACE that draws those parts of the surface which lie above the clipping plane. Note that it is safe to render the entire upper volume at once, since it can be assumed that all activation areas lie below the UPPER SURFACE.

On a final note, this render algorithm does not take into account multiple foldings of the brain surface along the same viewing ray from the camera. All surface folds that lie below the topmost surface point for a given ray are simply ignored. In the context of this thesis, this is actually desired behavior, since the entire folding

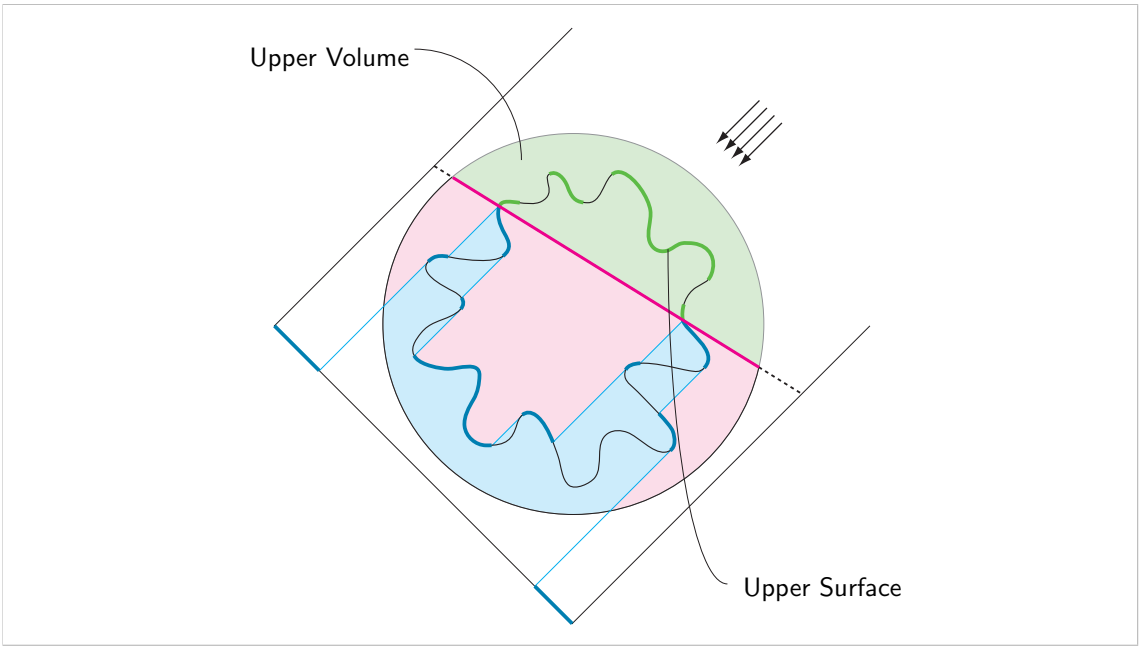


Figure 3.4: The last two layers complete the scene. The Upper Volume can be rendered in one pass, since it can only contains activation areas that lie below the Upper Surface.

structure of the brain surface is too complex to show it in its full detail all the time.

In the following chapters, the details for the individual volume and surface render passes are presented. They are composed into a pipeline that produces the final output in Chapter 7.

4 Mesh Rendering

4.1 Mesh Preprocessing

The brain surface output generated by FreeSurfer is a mesh consisting of vertex positions and triangles (faces). As such, it does not allow for an efficient way to traverse the surface defined by the mesh, e.g., finding all triangles adjacent to a given vertex is an $O(n)$ operation (n being the number of triangles). To speed up the algorithms presented in this chapter, the mesh is therefore first loaded into a half-edge data structure [Overmars, 1996]. This data structure is designed to allow for efficient mesh traversal queries like the one mentioned above (which now becomes a $O(t)$ operation, t being the number of triangles adjacent to the given vertex).

Surface Normals. The first algorithm that uses this half-edge data structure is the one that computes the surface normals: The mesh's triangles are interpreted as being an approximation of a curved surface. For each vertex, a normal is defined that denotes the tangent plane of the curved surface at that position. In order to compute these normals, first all triangles of the mesh are traversed. For each triangle, the normal of the plane defined by the three triangle vertices is computed and stored. Then, all vertices of the mesh are traversed, and for each vertex the surface normal is calculated by averaging the normals of the surrounding triangles and re-normalizing the result.

Intersection Queries. Mesh traversal queries are not the only type of queries that will be used in the following. A second, very important type will be intersection queries. These are, for example:

- Given a point in space and a direction, what is the shortest distance between the surface and the point along that direction?
- Given a point in space, what is the direction to the surface point that has the shortest distance to the given point?

Trivial algorithms for this type of queries again have a complexity of $O(n)$. By using a kd-tree data structure, however, the complexity can be reduced to $O(\log n)$. The creation and traversal of the kd-tree is explained in Chapter 12.

4.2 Ambient Occlusion

When using shading to convey the shape of an object, it can be assumed that the more natural the shading of the object appears under a given lighting condition, the better the grasp of its three-dimensional shape will be. In this work, a uniform gray color background is used, on which all renderings are composed. This probably corresponds best with a dim and uniform lighting situation in which light is coming from all directions with equal intensity, i.e. a diffuse lighting environment. Indeed, it has been shown [Langer and Bülthoff, 1999] that shape recognition based on shading in diffuse lighting conditions is superior to shape recognition in directional lighting conditions.

Given this lighting situation, computing the full global illumination for the scene will result in the most natural shading results. Assuming a diffuse mesh surface, the global illumination can be effectively calculated using a radiosity approach [Sillion and Puech, 1994]. In the context of this thesis, however, only an ambient occlusion term will be calculated for each vertex of the mesh to approximate the global illumination. The reason is that computing the ambient occlusion is relatively easy to implement and that it can be regarded as the first step of a full radiosity solution.

4.2.1 Derivation

Given a surface point $\mathbf{x}$, the amount of radiance leaving the surface in direction ω can be computed using the rendering equation for vacuum conditions [Kajiya, 1986]:

$$L(\mathbf{x}, \omega) \;=\; L_e(\mathbf{x}, \omega) + \int_\Omega f_r(\mathbf{x}, \omega \leftarrow \omega')\, L(\mathbf{x}, \omega')\, |\mathbf{n} \cdot \omega'|\, d\omega', \tag{4.1}$$

where L_e denotes the emitted radiance, $\mathbf{n}$ is the normal at the surface point, Ω denotes the upper hemisphere with respect to that normal and f_r is the bi-directional reflectance distribution function (BRDF). Assuming that the surface does not emit light and that it reflects light in a perfectly diffuse manner (lambertian reflection), then the BRDF becomes constant, i.e. $f_r = \frac{1}{\pi}$, and the render equation simplifies to

$$L(\mathbf{x}, \omega) = \frac{1}{\pi} \int_\Omega L(\mathbf{x}, \omega') |\mathbf{n} \cdot \omega'| d\omega'.$$

Finally, instead of aiming for a full global illumination, the incoming radiance term inside the integral is approximated by:

$$L(\mathbf{x}, \omega') \approx L_b(\omega') \cdot V(\mathbf{x}, \omega'),$$

where L_b is the radiance of the background and V is the visibility of the background from point $\mathbf{x}$ in direction ω'. Depending on whether the line of sight is occluded

by other geometry, this value is either 1 or 0. Since a completely diffuse lighting environment is assumed, L_b is constant and the approximation of the rendering equation becomes:

$$\begin{aligned} L(\mathbf{x}, \omega) &\approx L_b \cdot \frac{1}{\pi} \int_\Omega V(\mathbf{x}, \omega') |\mathbf{n} \cdot \omega'| d\omega' \\ &= L_b \cdot A(\mathbf{x}), \end{aligned} \tag{4.2}$$

where A is the *ambient visibility* term at point $\mathbf{x}$. It is 0 when the hemisphere above $\mathbf{x}$ is fully occluded, 1 when it is not occluded at all, and in between otherwise. Note that the *ambient occlusion* is simply 1 minus the ambient visibility.

4.2.2 Implementation

A straightforward approach to compute the ambient occlusion for every vertex is to use Monte Carlo integration of Equation 4.2. Starting with

$$A(\mathbf{x}) = \int_\Omega \underbrace{V(\mathbf{x}, \omega) \cdot \frac{1}{\pi} |\mathbf{n} \cdot \omega|}_{f(\omega)} d\omega, \tag{4.3}$$

a probability density function (pdf) that is constant over the hemisphere, $p_u(\omega) = \frac{1}{2\pi}$ can be included to get

$$\begin{aligned} A(\mathbf{x}) &= 2\pi \int_\Omega f(\omega) \cdot \frac{1}{2\pi} d\omega \\ &= 2\pi \cdot E_{p_u}[f], \end{aligned}$$

where $E_{p_u}[f]$ is the expected value of the function f with respect to p_u. By randomly drawing directions ω_i, $i = 1 \dots N$, according to p_u, i.e., uniformly over the hemisphere, the ambient visibility can be approximated by the following discrete summation:

$$\begin{aligned} A(\mathbf{x}) &= 2\pi \cdot \frac{1}{N} \sum_{i=1}^{N} f(\omega_i) \\ &= \frac{2}{N} \sum_{i=1}^{N} V(\mathbf{x}, \omega_i) |\mathbf{n} \cdot \omega_i|. \end{aligned} \tag{4.4}$$

Uniformly Sampling the Hemisphere. In order to evaluate Equation 4.4, seeing as p_u is constant, the hemisphere over point $\mathbf{x}$ has to be sampled uniformly. This is achieved by first sampling the hemisphere in a local tangent plane coordinate system, where $\mathbf{n}$ corresponds to the z-Axis and both the x- and y-Axis lie on the

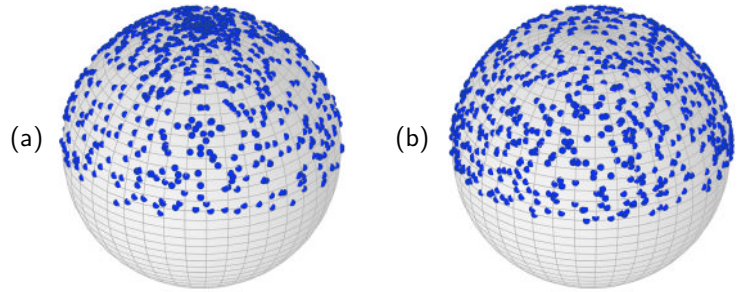


Figure 4.1: (a) Uniform sampling of both polar coordinates results in more samples at the pole of the hemisphere than at the equator. (b) Uniform sampling of the height (z-axis) and ϕ results in evenly distributed samples across the hemisphere.

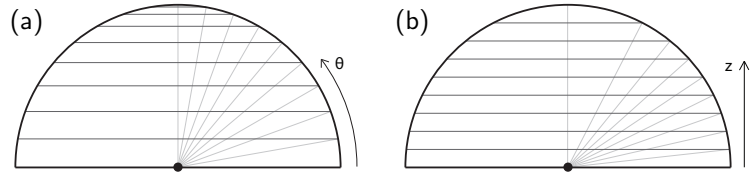


Figure 4.2: (a) θ is chosen uniformly between 0 and $\pi/2$. The resulting horizontal slices are not distributed uniformly across the z-axis, which leads to an over-sampling of the pole of the hemisphere. (b) θ is chosen such that the resulting slices are uniformly distributed along the z-axis.

tangent plane (with an arbitrary orientation). In this coordinate frame, given two random numbers a, b drawn uniformly from $[0, 1)$, they could be used to generate the polar coordinates $\theta = \frac{\pi}{2}a$ and $\phi = 2\pi b$. This simple scheme, however, results in a clustering of the directions at the pole of the hemisphere, which can be seen in Figure 4.1 (a). The reason is that the uniform distribution of the θ-values results in more sampling of the slices at the pole of the hemisphere (Figure 4.2 (a)), whereas the correct slice distribution would be uniform across the z-axis (Figure 4.2 (b)).

So, in order to correctly sample the hemisphere, the z-value of the direction vector is set directly to a, thereby ensuring a uniform slice distribution. This value is then used to compute the radius r of the slice: $r = \sqrt{1 - a^2}$. On this slice, a random direction is chosen by setting $\phi = 2\pi b$. Finally, the local direction vector can be computed:

$$\omega_{\text{local}} = \begin{pmatrix} r \cdot \cos\phi \\ r \cdot \sin\phi \\ a \end{pmatrix}.$$

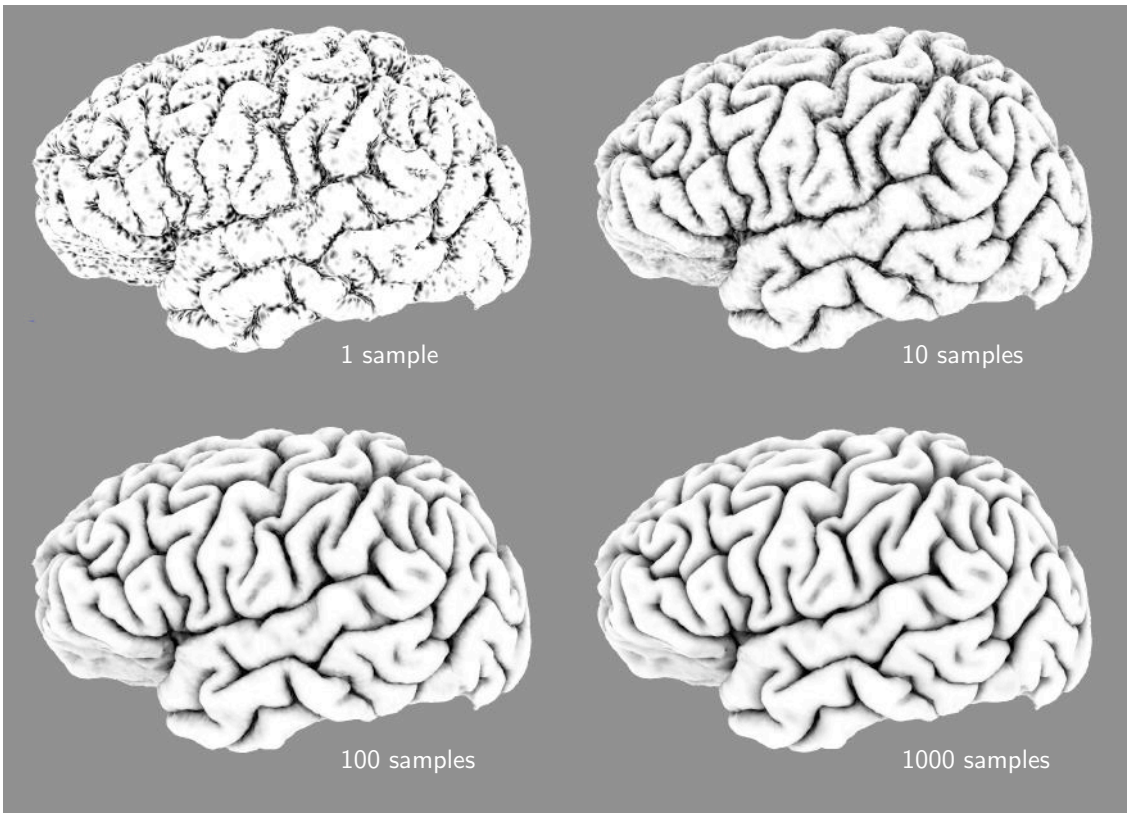


Figure 4.3: Ambient occlusion results for 1, 10, 100 and 1000 samples per vertex when using uniform sampling. The more samples, the better the approximation of the ambient visibility integral becomes and the less noise is present in the output.

Results of this sampling can be seen in Figure 4.1 (b). To transform the direction vector from the local tangent plane coordinate system to world coordinates, it has to be multiplied by the orientation matrix $L = \left(\mathbf{x}_{\text{world}}\, \mathbf{y}_{\text{world}}\, \mathbf{n}\right)$, so $\omega_{\text{world}} = L \cdot \omega_{\text{local}}$.

Evaluating the Visibility Function. In order to determine the value of $A(\mathbf{x}, \omega_i)$ for a given random direction ω_i, a ray is initialized with starting position $\mathbf{x}$ and direction ω_i. The previously constructed kd-tree data structure is then queried for the closest intersection of that ray with the stored geometry. If an intersection is found, the value of the visibility function is 0, otherwise it is 1.

Results. Figure 4.3 shows the results for the ambient visibility function when using 10, 100 and 1000 samples for the computation. The images are generated by storing the pre-computed ambient visibility values for each vertex in their color component and by using a special fragment shader that simply passes that value to the output.

Speeding Up the Computation. It is possible to achieve a faster convergence of the Monte Carlo integration by not uniformly sampling the hemisphere of directions. Looking back at Equation 4.3:

$$A(\mathbf{x}) = \int_{\Omega} \underbrace{V(\mathbf{x}, \omega)}_{g(\omega)} \cdot \underbrace{\frac{1}{\pi} |\mathbf{n} \cdot \omega|}_{p(\omega)} d\omega,$$

it is important to note that for directions that are almost perpendicular to the surface normal, $p(\omega)$ is close to 0. So whatever the value of $g(\omega)$ is for those directions, its contribution to the integral is very small. As a corollary, its contributed variance is also small. Therefore, to speed up the convergence of the Monte Carlo integration, it would be beneficial to spend less rays for sampling the lower parts of the hemisphere than for the upper parts. Indeed, if samples were to be drawn according to $p(\omega)$ (which is a valid pdf, see Appendix C), exactly the desired distribution of directions would be achieved.

With these observations, the ambient visibility can be written as:

$$\begin{aligned} A(\mathbf{x}) &= E_p[g] \\ &= \frac{1}{N} \sum_{i=1}^{N} g(\omega_i) \\ &= \frac{1}{N} \sum_{i=1}^{N} V(\mathbf{x}, \omega_i), \end{aligned} \tag{4.5}$$

where this time, the ω_i are drawn according to p. This can be achieved by first uniformly sampling the disk at the base of the hemisphere and then projecting that point upwards on the hemisphere [Pharr and Humphreys, 2004]. For random numbers $a, b \in [0, 1)$, the radius $r = \sqrt{a}$ and the angle $\phi = 2\pi b$, the local coordinates become $x = r \cdot \cos\phi$ and $y = r \cdot \sin\phi$. After projecting the point on the hemisphere, the direction vector becomes:

$$\omega_{\text{local}} = \begin{pmatrix} x \\ y \\ \sqrt{1 - x^2 - y^2} \end{pmatrix}.$$

Figure 4.4 shows the distribution of samples for this technique and Figure 4.5 shows the rendering results for 1, 10, 100 and 1000 samples per vertex. Figure 4.6 shows the direct comparison between uniform sampling and the sampling according to p presented here.

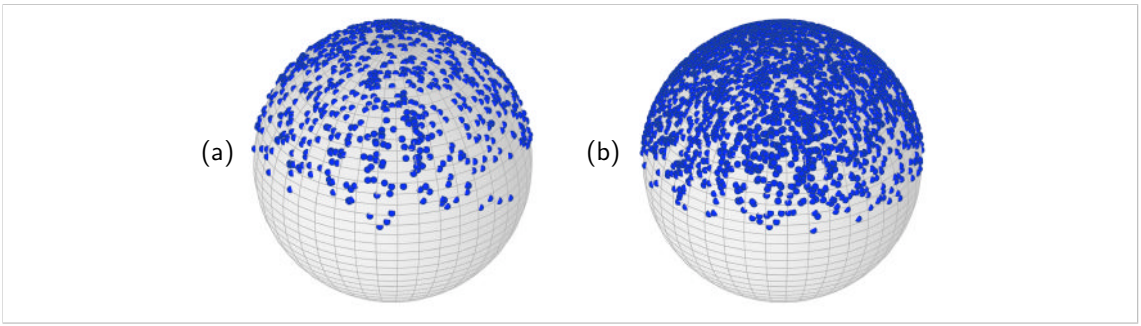


Figure 4.4: Sampling the hemisphere according to $p(\omega)$ with (a) 1000 and (b) 3000 samples. Although there are more samples at the pole, the distribution is different from the one shown in Figure 4.1 (a).

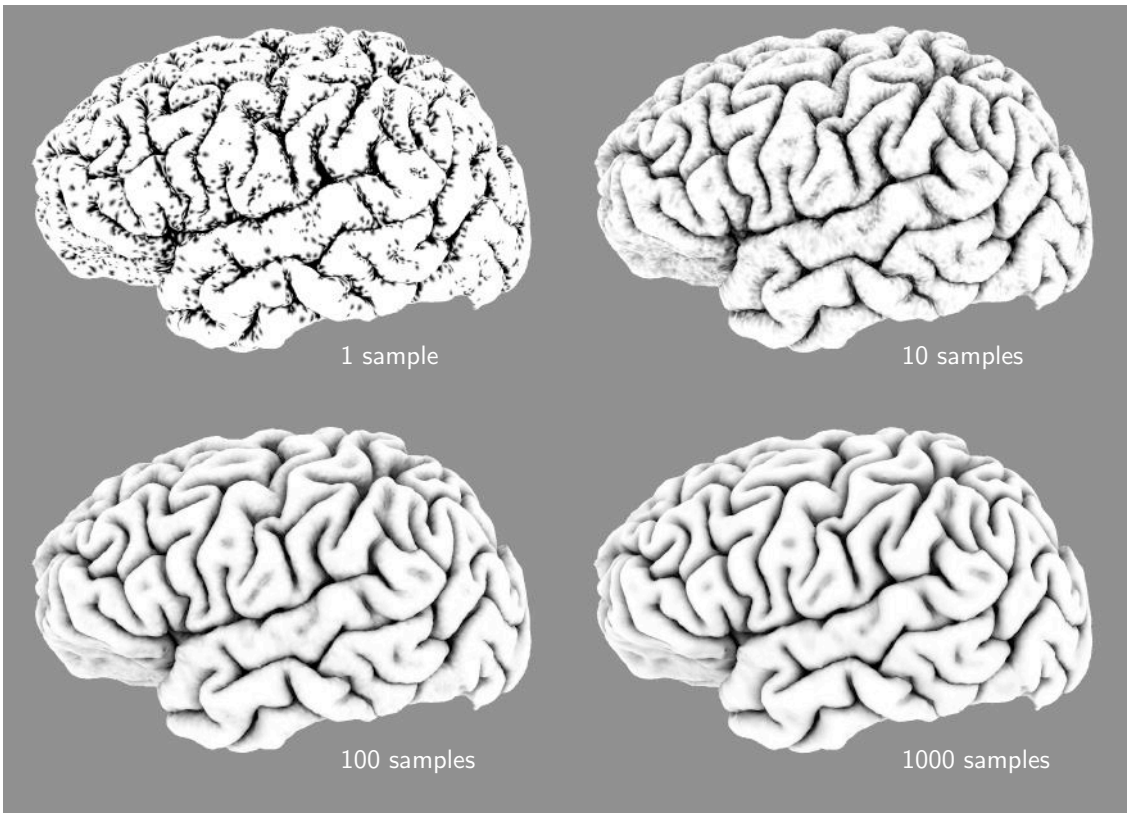


Figure 4.5: Ambient occlusion results for 1, 10, 100 and 1000 samples per vertex when using the sampling distribution shown in Figure 4.4. Note that using 1 sample per vertex results in ambient visibility values for each vertex that are either 1 or 0, which is in contrast to Figure 4.3, where they are scaled by a cosine factor.

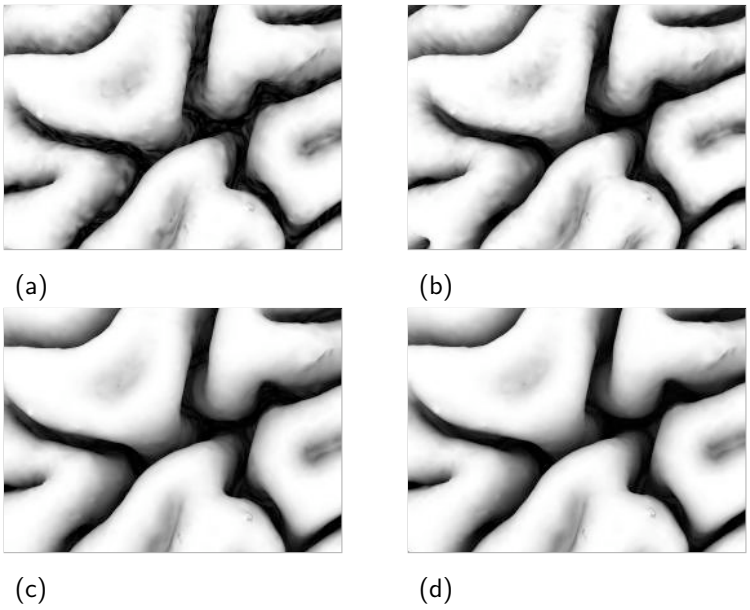


Figure 4.6: Comparison of uniform sampling (a), (c), and sampling according to $p(\omega)$ (b), (d), for 100 and 1000 samples per vertex, respectively. Especially the dark areas in (a) and (c) contain a significant amount of noise when compared to (b) and (d).

Combining Ambient Occlusion with Diffuse Shading. To combine the contribution of the ambient light and a single directional light source, a value of $L_b = 0.5\frac{W}{m^2\,sr}$ was chosen for the background radiance in Equation 4.2. The directional light source is set to contribute a maximum of $0.5\frac{W}{m^2\,sr}$ to the emission of a surface point. This is true if the surface is perpendicular to the light direction. Otherwise it is weighted by the typical (for diffuse surfaces) cosine factor. The ambient light and the point light's contribution can add up to at most $1\frac{W}{m^2\,sr}$ which makes it very convenient to implement inside a fragment shader. (Generally, the radiance emitted from a point can be anywhere in $[0, \infty)$, but the construction of the scene limits the radiance to a value between [0,1]. This has the advantage that no tone mapping is required in the shader.) Figure 4.7 shows a sample rendering.

4.3 Surface Area Coloring

For orientation purposes, different cortex areas will be presented with a different color. FreeSurfer already exports both cortex labels for each vertex and the color used for each label, so all that remains to be done is to load these labels and colors at startup and to render them accordingly.

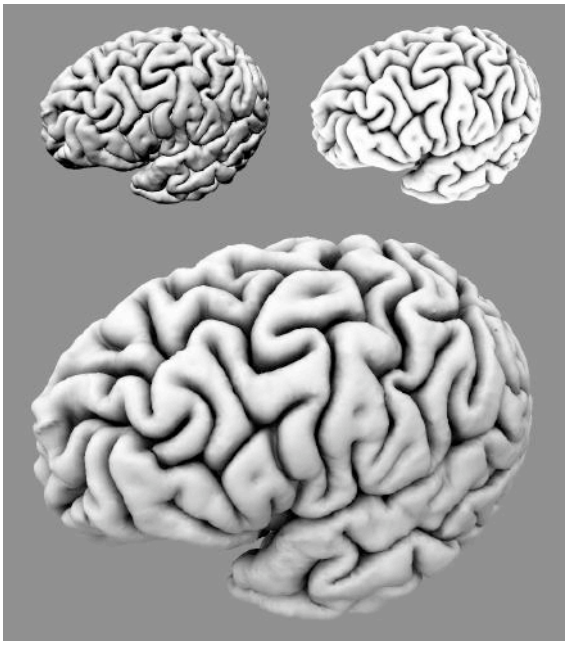


Figure 4.7: Directional lighting (upper left) is combined with ambient occlusion (upper right) to produce the final shading.

Regarding the actual render pass, there are two possible ways to draw the colors. The first is to attach the label number to the vertex and look up the corresponding color inside the fragment shader (e.g., by reading from a texture). When using smooth shading, however, interpolation between nearby vertices with different label numbers, say 3 and 10, results in ugly border artifacts: fragments between the two vertices cycle through all label numbers between 3 and 10, all of which might have completely different colors assigned to them.

The second approach is to send the actual colors for every vertex to the graphics card. This way, interpolation between adjacent vertices with different label numbers results in a linear blend between two colors—clearly a better behavior. A rendering of the labels is shown in Figure 4.8.

Render Pass: MESH BELOW 1

The ambient occlusion and diffuse shading from the previous section and the surface coloring from this section can be combined into one render pass. The surface coloring results are stored in the RGB component of the render target texture, the shading is stored in the alpha component (see Figure 4.9). A user defined clipping plane is passed to the fragment shader in order to discard all fragments that lie

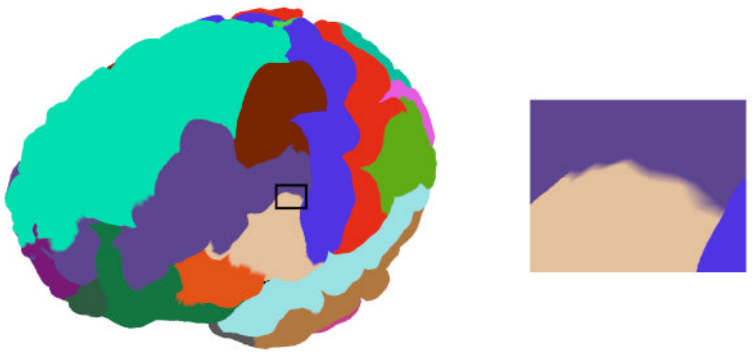


Figure 4.8: Brain surface colored according to FreeSurfer labels and their corresponding colors. The magnified portion shows how the colors are blended at the border of two surface areas.

above the clipping plane. See Chapter 6 for more information.

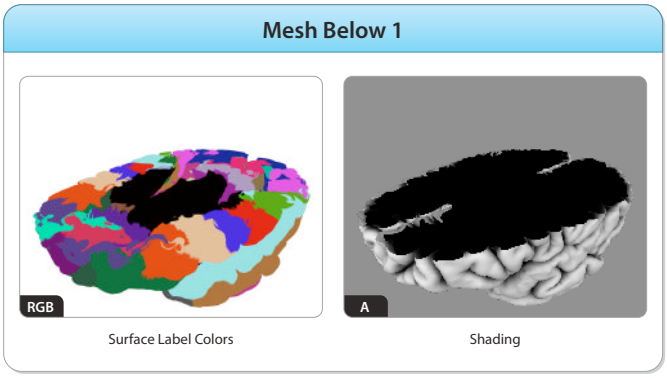


Figure 4.9: Output of the MESH BELOW 1 render pass.

Render Pass: MESH ABOVE 1

Same as above, except this time the fragment shader is set to discard all fragments that lie below the clipping plane (see Figure 4.10).

4.4 Silhouettes

The non-photorealistic rendering of the brain surface used in this thesis was originally inspired by [Fischer et al., 2005]. Figure 4.11 shows an image rendered with

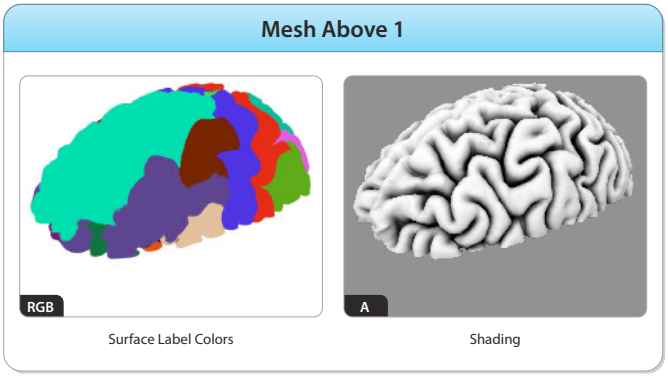


Figure 4.10: Output of the MESH ABOVE 1 render pass.

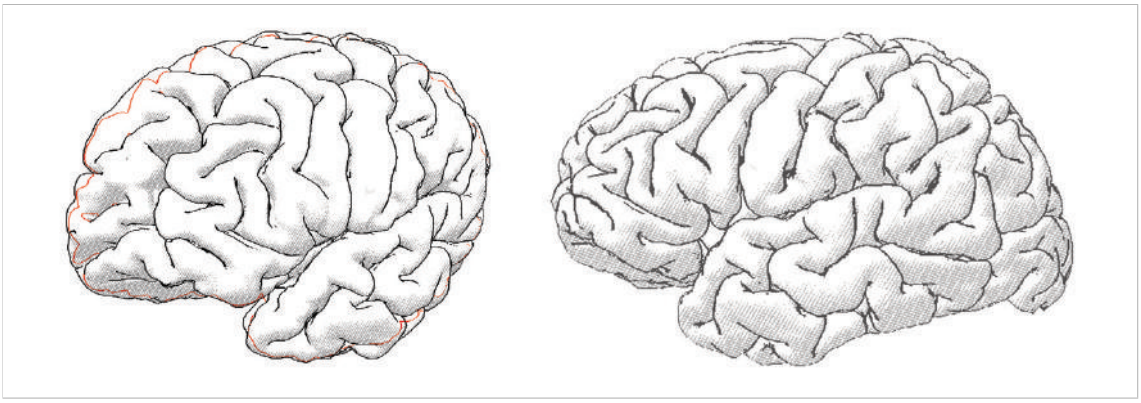


Figure 4.11: Brain surface rendered in the style presented in [Fischer et al., 2005].

that technique. One important aspect of this style is the rendering of mesh silhouettes as described in [Mitchell et al., 2002]. The detection of silhouettes is based on normal and depth discontinuities of the rendered object.

Depth Discontinuities. To determine depth discontinuities, the object is first rendered with a fragment shader that is responsible for storing the depth of each pixel in some channel of the render target. This *depthmap* D is shown in Figure 4.12 (a). In a second step, a screen sized quad is rendered, textured with the result from the previous pass. The fragment shader for this quad determines the gradient vector ∇D of the depth map by computing the partial derivatives $\frac{\partial D}{\partial x}$ and $\frac{\partial D}{\partial y}$. The higher the magnitude $|\nabla D|$ of this gradient vector is, the greater the depth discontinuity. A user definable gradient magnitude threshold determines whether or not a specific pixel is considered part of the silhouette. Figure 4.12 (b) shows the depth gradient

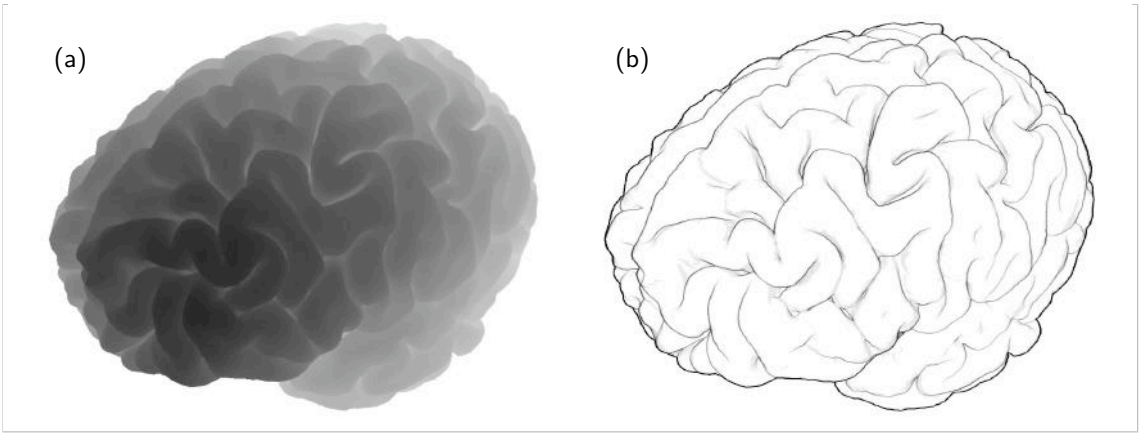


Figure 4.12: (a) Depth map of the brain surface. The darker a pixel, the closer it is to the camera. (b) Magnitudes of the depth map gradient. The darker the line, the higher the magnitude.

magnitudes.

Normal Discontinuities. Typical cases where the detection of depth discontinuities alone does not produce all desired outlines are object creases. Here, the depth gradient magnitude is close to 0, but the surface normals change rapidly. Detection of these normal discontinuities is performed similar to the steps above. First, the object's normal vectors are stored in the RGB component of a texture by using a special fragment shader (see Figure 4.13 (a)). Then, the *normal similarity* s is computed by a fragment shader during the rendering of a screen-sized quad (the texture of the previous step being the input for this step). For each pixel, its normal $\mathbf{n}$ and the normals $\mathbf{n}_u, \mathbf{n}_d, \mathbf{n}_l, \mathbf{n}_r$ of the neighboring pixels are read from the texture. The normal similarity is the sum of the dot products between the the current normal and the neighboring ones: $s = (\mathbf{n}\cdot\mathbf{n}_u) + (\mathbf{n}\cdot\mathbf{n}_d) + (\mathbf{n}\cdot\mathbf{n}_l) + (\mathbf{n}\cdot\mathbf{n}_r) = \mathbf{n}\cdot(\mathbf{n}_u+\mathbf{n}_d+\mathbf{n}_l+\mathbf{n}_r)$. Again, a user defined threshold determines whether or not the current pixel is considered as part of the outline. Figure 4.13 (b) shows computed normal similarities.

Render Pass: MESH BELOW 2

The first steps of determining depth and normal discontinuities can be combined into one render pass, storing the normals in the RGB components, and the depth in the alpha component. Since the result of this pass is later used for depth comparisons, the typical 8 bits per channel are not satisfactory, so this texture uses 32 bits per channel. This added precision will later be useful for storing additional information in each of the RGB channels.

As in the previous render passes, coordinates for the user defined clipping plane

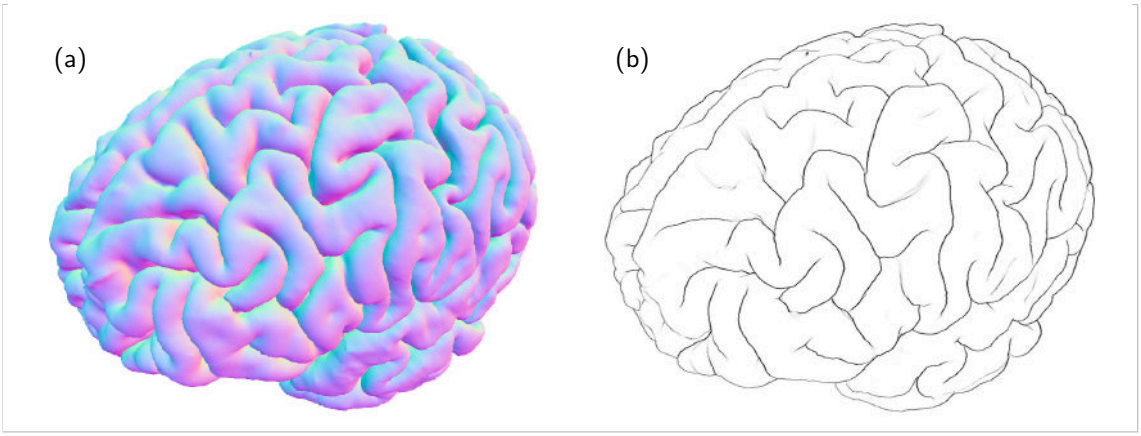


Figure 4.13: (a) Normals of the brain surface stored as RGB color components. (b) Normal similarities. The darker a pixel, the lower the similarity.

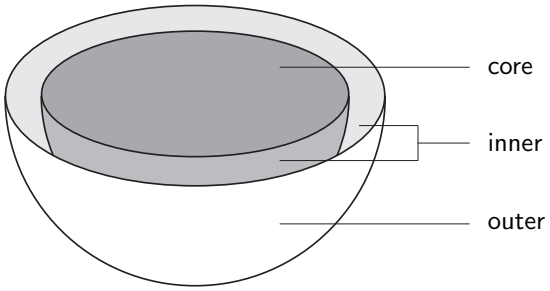


Figure 4.14: Classification of pixels into outer, inner and core pixels.

are passed to the fragment shader to discard all fragments that are above that plane. This reveals some inner parts of the outer surface and some parts of the white matter/gray matter boundary, i.e. the inner surface. In a later stage, it will become necessary to distinguish between three different pixel categories (see Figure 4.14):

- *outer pixels* — these are pixels that belong to the outer part of the outer surface,
- *core pixels* — pixels belonging to the inner part of the inner surface,
- *inner pixels* — pixels that lie in between, i.e. they belong either to the inner part of the outer surface or to the outer part of the inner surface.

This information is stored in the length of the normal. So, instead of always

storing a unit length normal in the RGB component, it is scaled depending on the category of the current pixel. Outer pixels are not scaled at all, core pixels are scaled by a factor of 0.7 and inner pixels are scaled by 0.4. This is implemented by using two slightly different fragment shaders for the outer and inner surface. Both shaders determine whether the current fragment belongs to the front or to the back side of the surface.[1] Depending on the outcome, the shader for the outer surface then scales the normal by 1.0 or 0.4, whereas the shader for the inner surface scales the normal by 0.4 or 0.7. See Figure 4.15 for the outputs of this pass.

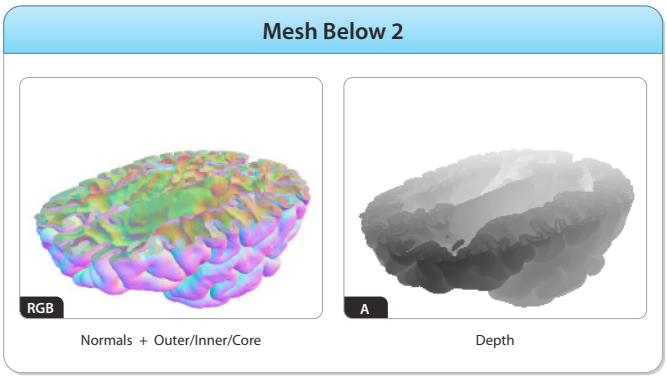


Figure 4.15: Output of the MESH BELOW 2 render pass.

Render Pass: MESH BELOW 3

The second step of determining depth discontinuities as well as the second step of determining normal discontinuities are combined in this render pass. The magnitude of the depth gradient is stored in the red component, the normal similarity in the green component. Before the normal similarity can be computed, however, the normals read from the texture (five in total—one for the current pixel and four for the neighbors) have to be re-normalized, i.e. brought to unit length. By inspecting the length each normal had before normalization, the category of each pixel can be determined. This information is then stored in the blue component (which will be called *pixel category map* C). For convenience, a simple alpha mask is also stored in the alpha component (see Figure 4.16)

In the case of rendering brain meshes, it turned out that detecting normal discontinuities already accounts for nearly all the relevant silhouettes. Both techniques

[1] As a technical note, it can be conveniently determined inside a fragment shader whether a given fragment belongs to the front or the back side of a polygon by querying the special input variable gl_FrontFacing. At the time of this writing, however, driver support for this feature is not present for the used ATI graphics card, so a slightly more complex solution is used: the vertex shader writes different values to gl_FrontColor and gl_BackColor, which the fragment shader then reads through gl_Color.

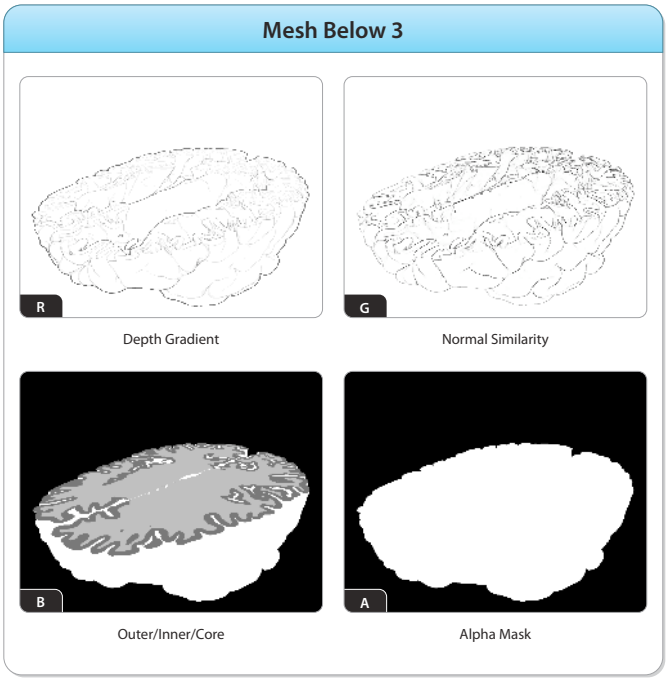


Figure 4.16: Output of the MESH BELOW 3 render pass.

(normal and depth discontinuity detection), however, performed poorly at robustly detecting the rims of the surface right below the clipping plane. While depth discontinuity detection should succeed in most of the cases, sometimes the inner and outer surface are very close together. In order to still be able to detect the rims, the depth threshold would have to be set very low which in turn would lead to unpleasant artifacts at other parts of the surface. So, an alternative solution was devised: instead of performing edge detection on the depthmap, it is performed on the pixel category map C. There is no need to do this in yet another render pass, since all the necessary information is already available in this pass.

While performing the edge detection, another important piece of information can be extracted: does the rim belong to the outer surface, or does it belong to the inner surface? If the former holds true, the outline is stored in the red component at full intensity and at half intensity otherwise (see Figure 4.17).

Render Pass: MESH ABOVE 2

This is the same as the render pass for the lower part of the mesh, except for the fact that this time, the fragment shader discards all fragments below the clipping plane. Also, there is no need to distinguish between different categories of pixels

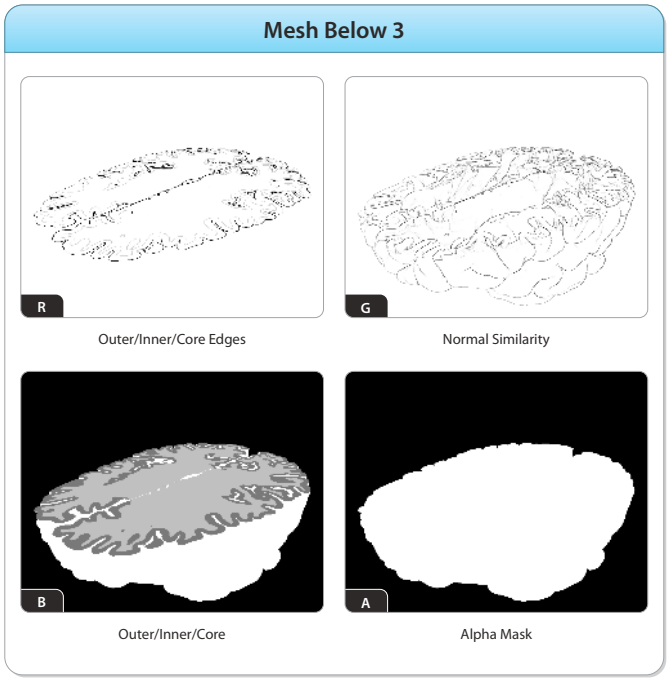


Figure 4.17: Revised output of the MESH BELOW 3 render pass.

(outer, inner, core), since the upper part of the mesh is rendered in a much simpler manner. As before, a texture with 32 bits per channel is used (see Figure 4.18).

Render Pass: MESH ABOVE 3

Again, this pass is very similar to "Mesh Below 3". However, since no pixel categories have been computed in the previous pass, obviously no edge detection on the pixel categories is performed. The red channel instead holds the result of an edge detection performed on the alpha mask. The blue channel that used to hold the pixel categories is now empty (see Figure 4.19).

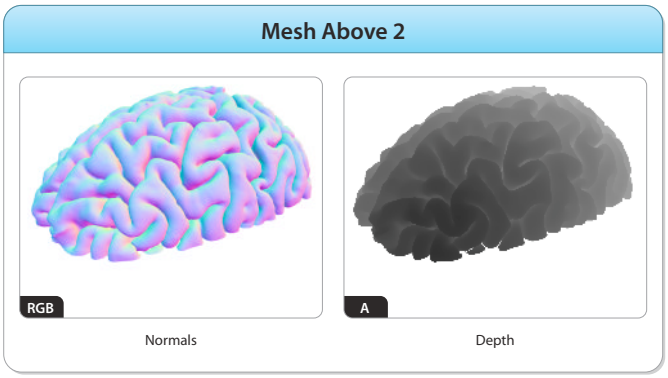


Figure 4.18: Output of the MESH ABOVE 2 render pass.

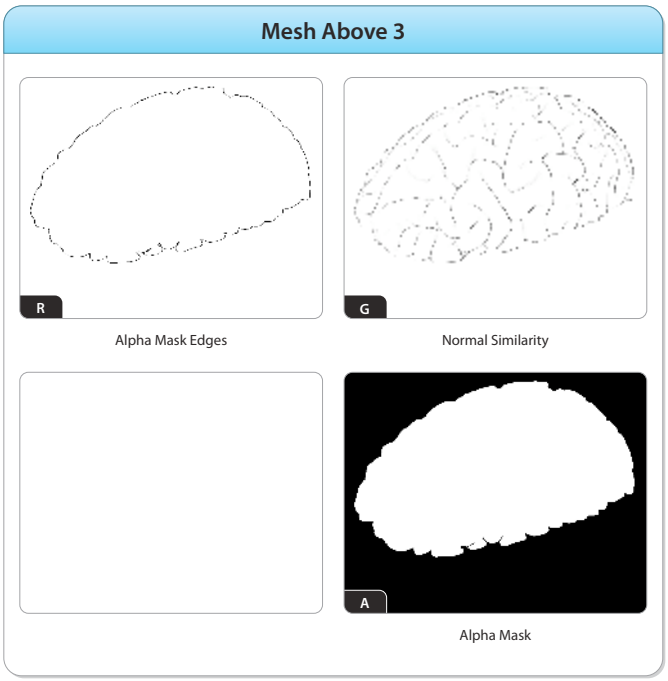


Figure 4.19: Output of the MESH ABOVE 3 render pass.

5 Volume Rendering

In rendering algorithms, a simplifying assumption commonly made to speed up the rendering process is that objects in the scene can be represented through hard object boundaries (meshes) and that the space between those meshes is made of vacuum, i.e., does not contain any participating media.

When dealing with fMRI activation volumes, deciding on a hard object boundary is not straightforward. One approach would be to construct isosurfaces from the activation volume, but in doing so, a lot of information would be lost. For example: Where is the center of the activation? How fast is the spatial decay of the activation?

On the other hand, when dropping *all* simplifying assumptions, the rendering equation—or more general: the *equation of transfer* [Arvo, 1993]—becomes too complex to compute in real time on today's hardware. However, in order to be able to perform basic volume rendering, there are certain assumptions that can be made that allow to convert the general equation of transfer into a more simple *emission-absorption equation* (see Appendix B for all assumptions and a complete derivation):

$$L(\mathbf{x},\omega) = \int_a^b \underbrace{\exp\left[-\int_a^t \sigma(\mathbf{x}-z\omega)dz\right]}_{\text{path absorption}} \underbrace{\epsilon(\mathbf{x}-t\omega,\omega)}_{\text{emission}}\, dt, \tag{5.1}$$

where $\epsilon(\mathbf{x},\omega)$ denotes the emission (in $\frac{W}{m^3 sr}$) and $\sigma(\mathbf{x})$ denotes the absorption (in $\frac{1}{m}$) at point $\mathbf{x}$ respecively, and a and b denote the distance of the volume boundaries from $\mathbf{x}$ along the direction vector $-\omega$ (see Figure 5.1).

By discretizing this equation into N steps, where each step has a width of $\Delta t = \frac{b-a}{N}$, the equation can be rewritten as:

$$\begin{aligned} L(\mathbf{x},\omega) &= \sum_{i=0}^{N} \left(\prod_{j=i+1}^{N} A_j\right) \cdot \epsilon_i \cdot \Delta t \\ &= \Delta t\epsilon_N + A_N(\cdots A_2(\Delta t\epsilon_1 + A_1(\Delta t\epsilon_0))), \end{aligned}$$

$$A_i = \exp(\Delta t \cdot \sigma_i), \tag{5.2}$$

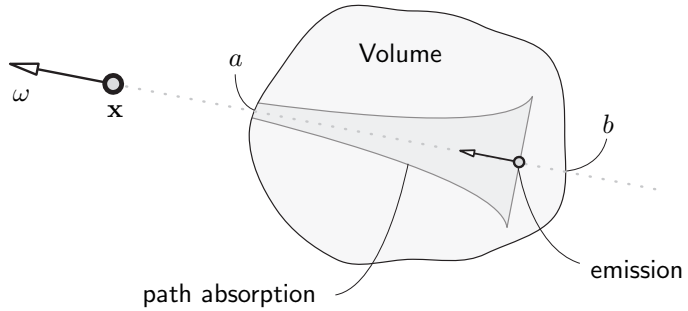


Figure 5.1: Schematic display of Equation 5.1. The emission and path absorption are shown for an arbitrary point inside the volume. Note that in volume rendering, a and b are typically chosen so that they lie outside of the volume. This does not change the computed radiance, since, by definition, no participating medium exists on the outside.

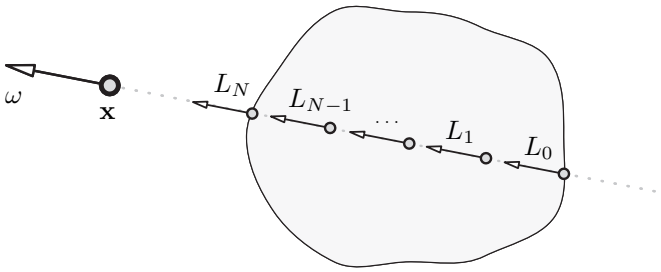


Figure 5.2: The discrete points used to approximate the volume integral.

where ϵ_i and σ_i denote the emission and absorption at the discretized points, respectively. The exact derivation can again be found in Appendix B. Casting Equation 5.2 in a recursive form leads to:

$$\begin{aligned} L_i &= \Delta t \cdot \epsilon_i + A_i \cdot L_{i-1}, \\ L_0 &= \Delta t \cdot \epsilon_0, \end{aligned} \tag{5.3}$$

where L_N is the radiance leaving the border of the volume and is therefore the same as $L(\mathbf{x}, \omega)$ (see Figure 5.2).

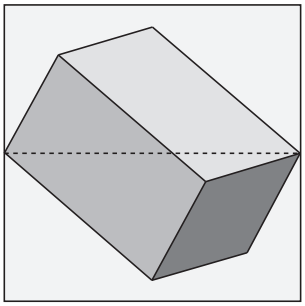


Figure 5.3: The volume (depicted with arbitrary dimensions) is rotated so that its maximal diameter (the dotted line) aligns with a horizontal cut through one of the volume slices (the outer square).

5.1 Volume Rendering on the GPU

The recursive formulation of Equation 5.3 lends itself to two different implementations of volume rendering on the GPU. The first is to use textured quads that are spaced apart by Δt and oriented towards the camera, rendered back to front ([Cabral et al., 1994]). The second is to use raycasting, i.e., to use a special fragment shader that uses an internal loop to traverse a volume texture in steps of size Δt ([Hall and Watt, 1991, Weiler et al., 2006]). In this thesis, the first approach is used and described in the following.

Volume Slices. Given a volume of width w, height h and depth d that is centered at the origin, the first task is to create a set of N quadratic slices that form a cube—the *slice cube*—that encompasses the entire volume. Since these slices will later be arbitrarily rotated to face the camera, they have to be bigger than the actual volume dimensions. Put more specifically, the minimal diameter of the slice cube has to be at least as big as the maximal diameter of the volume. The minimal diameter of the slice cube is simply the length s of one of its sides. The maximal diameter of the volume is the maximal distance of any two opposing corners of the volume (see Figure 5.3). Therefore, the slice size can be computed by:

$$s = \sqrt{w^2 + h^2 + d^2}.$$

And since N slices are created, the step size can be computed by $\Delta t = \frac{s}{N}$. Finally, all slices are positioned perpendicular to the z-Axis and spaced Δt apart. This is depicted in Figure 5.4.

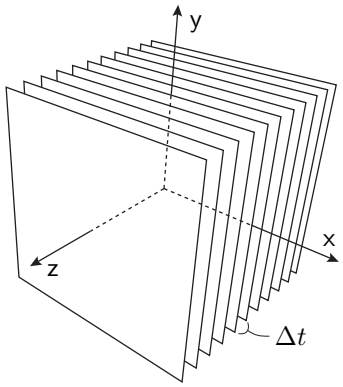


Figure 5.4: Alignment of the volume slices.

Volume Texture. The volume texture is a three-dimensional array of voxels (volume elements), which are the three-dimensional equivalent to pixels (picture elements). In this implementation, each voxel can have a R,G,B and an alpha component that define various aspects of a three-dimensional volume. As with two-dimensional textures, the resolution of any of the volume texture's dimensions does not have to correlate with any of the volume's dimensions in world coordinates, i.e., the voxel spacing does not have to be the same for each dimension. For example, even though the width of the volume might be smaller than its height and depth, the volume texture can have the highest resolution in the x-dimension and lower resolutions in the y- and z-dimensions. This decoupling of volume size and resolution is actually very important when dealing with MRI data. As described in the introduction, the resolution on the x/y-plane is higher than the resolution along the z-axis, i.e., the volume is non-isotropic.

Texture Coordinates. In order to map the volume texture to the actual volume, each corner vertex of each slice is assigned a three-dimensional texture coordinate. As is usual for textures, the texture coordinates are in the range $[0, 1]^3$. If the texture is accessed outside of this range, a user specified border color is returned. Indeed, since the previously created volume slices are bigger than the actual volume, each corner will have a texture coordinate that lies outside of $[0, 1]^3$.

Later on, the volume slices will be rotated to face the camera. When doing so, the texture coordinates have to be transformed as well. To simplify this transformation, the texture coordinates of each vertex are initially set to equal the world space coordinates of that vertex. So, for example, if the vertex is at position

$\mathbf{p} = (100\ 50\ 30)^\top$, then its texture coordinate is set to $\mathbf{t} = (100\ 50\ 30)^\top$ as well. During the rendering of the volume slices, these assigned texture coordinates have to be transformed by a *texture matrix* so that they lie in the correct range. (Since OpenGL provides direct support for texture matrices, no custom scheme has to be implemented for the described functionality.)

An alternative approach would have been to intersect each slice with the volume after each rotation, compute its intersection shape and render that [McReynolds and Blythe, 2005]. This approach has the advantage of only drawing fragments that are actually inside the volume. It is more complex than the method presented here, however, and on today's hardware the fragment processing has reached very high speeds, thereby diminishing the need for this optimization.

Texture Transformations. The texture matrix in the context of this implementation takes a coordinate defined in the volume's local coordinate system and transforms that into the texture coordinate system. The matrix will be called $\mathbf{T}_{\text{Local}\to\text{Tex}}$, where the letter T is chosen to symbolize that the matrix is applied to texture coordinates. The first step in computing this matrix is to transform the volume's corners from the range $[-\frac{w}{2}, \frac{w}{2}] \times [-\frac{h}{2}, \frac{h}{2}] \times [-\frac{d}{2}, \frac{d}{2}]$ to the range $[-\frac{1}{2}, \frac{1}{2}]^3$ by using a scaling matrix $\mathbf{T}_{\text{Scale}}$ defined as follows:

$$\mathbf{T}_{\text{Scale}} = \begin{pmatrix} 1/w & 0 & 0 & 0 \\ 0 & 1/h & 0 & 0 \\ 0 & 0 & 1/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The second step is translating the range $[-\frac{1}{2}, \frac{1}{2}]^3$ to $[0, 1]^3$:

$$\mathbf{T}_{\text{Translate}} = \begin{pmatrix} 1 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 1/2 \\ 0 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

By concatenation of these steps, the texture matrix becomes:

$$\mathbf{T}_{\text{Local}\to\text{Tex}} = \mathbf{T}_{\text{Translate}} \cdot \mathbf{T}_{\text{Scale}}.$$

Position and Orientation of the Volume. The volume can be translated ($\mathbf{G}_{\text{Translate}}$) and rotated ($\mathbf{G}_{\text{Rotate}}$) arbitrarily by the user, thereby transforming the vertex coordinates of the slices from the local coordinate system to the world coordinate system:

$$\mathbf{G}_{\text{Local}\to\text{World}} = \mathbf{G}_{\text{Translate}} \cdot \mathbf{G}_{\text{Rotate}}.$$

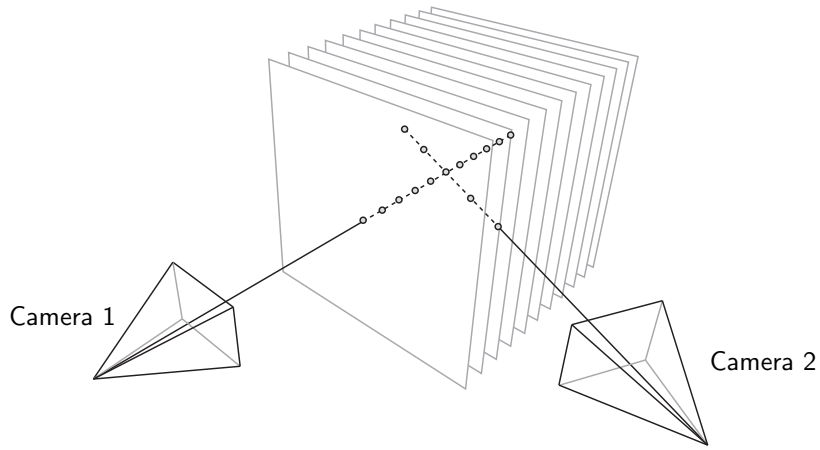


Figure 5.5: The viewing direction of Camera 1 is orthogonal to the volume slices. The viewing direction of Camera 2 is closer to being parallel to them. This results in both fewer samples and greater sampling distances.

The letter G is chosen to symbolize, that this matrix affects the actual geometry, i.e., slice vertex positions. In later equations, the symbol $\mathbf{G}$ will be used as a shorthand notation for $\mathbf{G}_{\text{Local}\rightarrow\text{World}}$. This matrix does not affect the texture coordinates in any way, so no adjustment to the texture matrix is necessary.

Orienting the Slices to Face the Camera. Figure 5.5 shows two cameras—one with a viewing direction orthogonal to the slices and one where the viewing direction is closer to being parallel to the slices. In the latter case, the assumptions made for Equation 5.3 are clearly violated, since a viewing ray emanating from the camera intersects consecutive volume slices with a distance far greater than Δt. Note that even when the slices are perfectly orthogonal to the camera's viewing direction, Δt is an approximation for the slice distance that only holds exactly for the centers of the slices. In general, the approximation error increases as the viewing ray direction and the plane normals are closer to being orthogonal to each other. Therefore, in order to minimize the approximation error, the volume slices are rotated so that they always face the camera.

To derive the rotation matrix by which the slices are transformed, the target orientation of the slices has to be constructed. Given the world space position of the camera $\mathbf{c}$ and the volume center $\mathbf{v}$, the target z-Axis becomes:

$$\mathbf{z}_c = \frac{\mathbf{c} - \mathbf{v}}{||\mathbf{c} - \mathbf{v}||}.$$

The target x-axis $\mathbf{x}_c$ and y-axis $\mathbf{y}_c$ have to be chosen to be orthogonal to $\mathbf{z}_c$ but can otherwise be arbitrarily rotated. The exact orientation is irrelevant, since the slice size was chosen to always be greater than the volume's maximal diameter. The target orientation is therefore:

$$\mathbf{C} = (\mathbf{x}_c \ \mathbf{y}_c \ \mathbf{z}_c),$$

and the relative rotation between $\mathbf{C}$ and the current slice orientation $\mathbf{G}$ can be expressed as:

$$\mathbf{R}_{\text{World}} = \mathbf{C} \cdot \mathbf{G}^{-1}, \tag{5.4}$$

where the subscript "World" denotes that $\mathbf{R}$ is the relative orientation with respect to the world coordinate system. In order to express this rotation in the local volume coordinate system, the following transformations have to be applied:

$$\begin{aligned}\mathbf{R}_{\text{Local}} &= \mathbf{G}^{-1} \cdot \mathbf{R}_{\text{World}} \cdot \mathbf{G} \\ &= \mathbf{G}^{-1} \cdot (\mathbf{C} \cdot \mathbf{G}^{-1}) \cdot \mathbf{G} \\ &= \mathbf{G}^{-1} \cdot \mathbf{C}\end{aligned}$$

Finally, both the geometry matrix and the texture matrix can be adjusted:

$$\begin{aligned}\mathbf{G}^* &= \mathbf{G}_{\text{Local}\to\text{World}} \cdot \mathbf{R}_{\text{Local}} = \mathbf{C} \\ \mathbf{T}^* &= \mathbf{T}_{\text{Local}\to\text{Tex}} \cdot \mathbf{R}_{\text{Local}}\end{aligned} \tag{5.5}$$

Results. Figure 5.6 shows a rectangular volume rendered with different orientations and world space positions.

5.2 Volume Shader and Transfer Functions

In order to implement Equation 5.3:

$$\begin{aligned}L_i &= \Delta t \cdot \epsilon_i + A_i \cdot L_{i-1}, \\ L_0 &= \Delta t \cdot \epsilon_0,\end{aligned}$$

on the GPU, the following steps have to be taken:

- The framebuffer background is cleared with an RGBA value of $(0, 0, 0, 0)$.
- The volume slices created in the previous chapter are rendered in a back-to-front order.

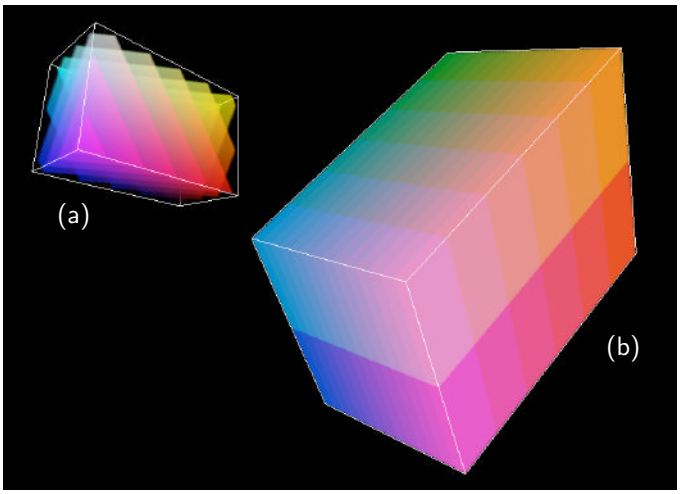


Figure 5.6: (a) Only few slices have been used to render the volume so that their alignment is visible. (b) The same volume with a different orientation and rendered using 180 slices. Also, texture blending has been turned off, which reveals the underlying volume texture. The volume texture dimensions appear to be (14, 2, 6), but this is only because there is a 1 voxel boundary around the entire texture to avoid blending artifacts. So the texture size is (16, 4, 8).

- OpenGL blending is turned on and the blending equation is set to (GL_ONE, GL_ONE_MINUS_SRC_ALPHA). This means that the color output from the fragment shader (in the following also called *volume shader*) is simply added to the framebuffer, whereas the previously stored framebuffer value is scaled by 1 minus the volume shader's alpha output.
- The volume shader writes $1 - A_i$ into the alpha channel of the output. This way, the OpenGL blending mechanism actually implements the second summand from Equation 5.3. An alternative would have been to set the blending equation to (GL_ONE, GL_SRC_ALPHA) and to write A_i into the output directly. The way it is set up at the moment, however, makes it easy to think of the alpha value as being the opacity of the current fragment.
- The volume shader writes the value of $\Delta t \cdot \epsilon_i$ into the color channel of the output. Since this value is simply added to the framebuffer, the first summand of Equation 5.3 is therefore also implemented. Note that although Equation 5.3 in its current form only accounts for one wavelength of light (or light color), L, ϵ and A can be treated as vectors containing values for different wavelengths, e.g for red, green and blue.

Transfer Functions. Both the absorption σ and the emission ϵ vary throughout the volume and are therefore stored in a volume texture. This texture is accessed by the volume shader. Although RGB absorption and emission values could be stored

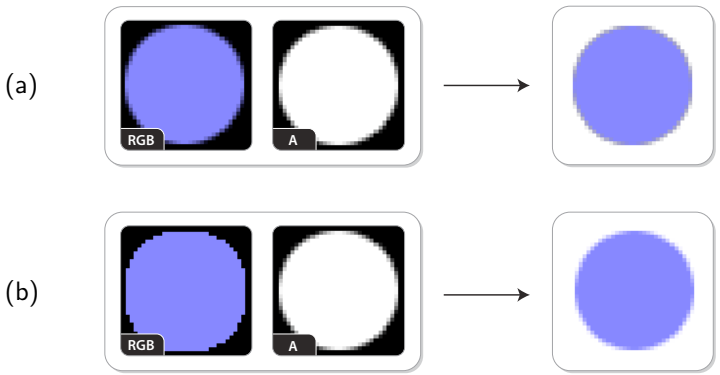


Figure 5.7: (a) The alpha-channel has been pre-multiplied into the color channel. When the image is blended with a white background, a black border appears. (b) When colors are not pre-multiplied, blending the image with the white background produces the correct result.

directly inside this volume texture, it is more common to just store a single value and to use a *transfer function* to map this value (and potentially its derivatives) to RGB values ([McReynolds and Blythe, 2005]). This transfer function is passed to the volume shader in form of a texture. The transfer function used in this work has been manually chosen and refined to produce nice fMRI renderings. It is shown in the next section.

The Pre-Multiplied Alpha Problem. Figure 5.7 (a) shows the result of blending a two-dimensional texture with the depicted RGB and alpha channels on a white background (the blending function being exactly the one that has been described above). The same situation in three dimensions can be seen in Figure 5.8 (a). The reason for this behavior in both the two- and the three-dimensional case is that the color value of pixels (or voxels) at the border has already been multiplied by the alpha value in order to blend it with the black background. When it is later blended with the white background, a black border remains. To correct this behavior, all pixel colors have to be stored without pre-multiplying them with their corresponding alpha value. This can be seen in Figure 5.7 (b) for the two-dimensional case. For three dimensions, the results are shown in 5.8 (b).

Note that in volume rendering, there is another artifact called *color bleeding* [Wittenbrink et al., 1998] that occurs when the colors of the volume change rapidly. This artifact was not present in the context of this work, however, so there was no need to implement any countermeasures.

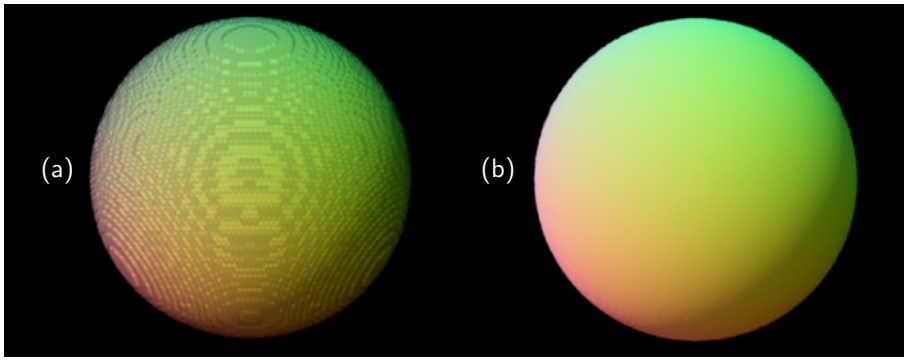


Figure 5.8: (a) Volume rendered with pre-multiplied alpha. The volume texture (256^3 voxels) is linearly interpolated, so the "block"-artifacts are solely due to the pre-multiplied color values. (b) Exactly the same scene, except that the colors are not stored pre-multiplied. In both (a) and (b), the volume renderer was set up to calculate the volume gradient vector for each rendered pixel and to use that to produce the diffuse shading seen in the images.

5.3 Volume Rendering Passes

A total of three volume render passes are performed to correctly interleave the meshes and the volume of the scene. No intermediate textures are used to hold the result of each pass, since each pass renders directly into the final output texture of the pipeline. Therefore, their exact order is described in Chapter 7. This section only describes the general setup of the vertex and fragment shaders.

fMRI Data. First, each of the fMRI data's dimensions are rounded up to the closest power-of-two. Then, the data itself is resampled using bi-cubic interpolation to fit into the new dimensions. This data is finally loaded into the red component of a volume texture. The transfer function used for the emission and absorption is shown in Figure 5.9. Another approach would have been to leave the data un-scaled but instead embed it in the bigger texture by adding a padding at the borders.

The Clipping Plane. As in the previous geometry rendering steps, the user defined clipping plane is passed to the fragment shader. Depending on the pass, fragments above or below the clipping plane are discarded. However, hard clipping of fragments during volume rendering results in ugly artifacts. This can be seen in Figure 5.10 (a). So instead of this hard clipping, a "soft" clipping is used, where fragments that lie above or below the clipping plane by an amount of less than Δd are not discarded, but are gradually faded away. This is achieved by linearly scaling down their absorption and their emission until they reach 0. Ideally, Δd should depend on the step size Δt, but in the current implementation it is simply

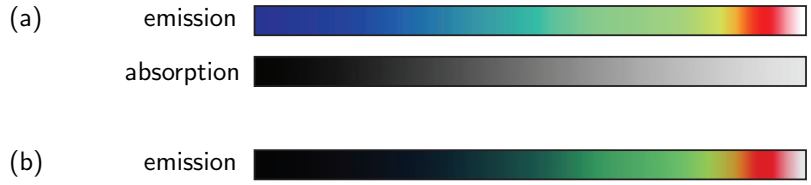


Figure 5.9: (a) The emission and absorption transfer function used for fMRI volume rendering. (The darker the value, the lower the absorption.) Emission values are stored in the RGB components, absorption values in the A component of the texture. (b) An alternative emission transfer function that would lead to the artifacts described in Section 5.2.

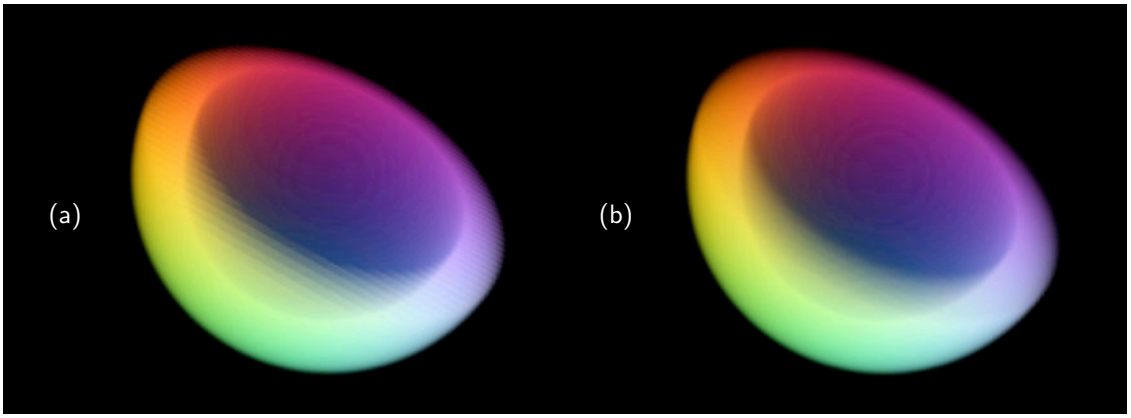


Figure 5.10: (a) The volume is rendered with 100 slices and hard clipping is used to discard fragments lying above the clipping plane. This reveals the edges of the used slices. The more slices are used, the less noticeable this artifact becomes. (b) The same number of slices is used as before. Instead of using hard clipping, however, soft clipping is used to linearly "fade out" fragments close to the clipping plane.

chosen by hand. The result of using this soft clipping is shown in Figure 5.10 (b). Results for a more complex volume can be seen in Figure 5.11.

Depth Clipping. In some volume rendering passes, it is necessary to discard fragments that lie below or above a certain depth value (as seen from the camera). To this end, a previously rendered depth texture D is made available to the fragment shader. It is assumed that the the range $[0,1]^2$ of the texture actually corresponds to the current viewport dimensions $[0, \text{width}] \times [0, \text{height}]$. If the passed texture is one of the previously rendered textures (e.g., the output of render pass Mesh Below 1), then this assumption certainly holds. Additionally, the current viewport width and height are passed to the fragment shader. When processing a

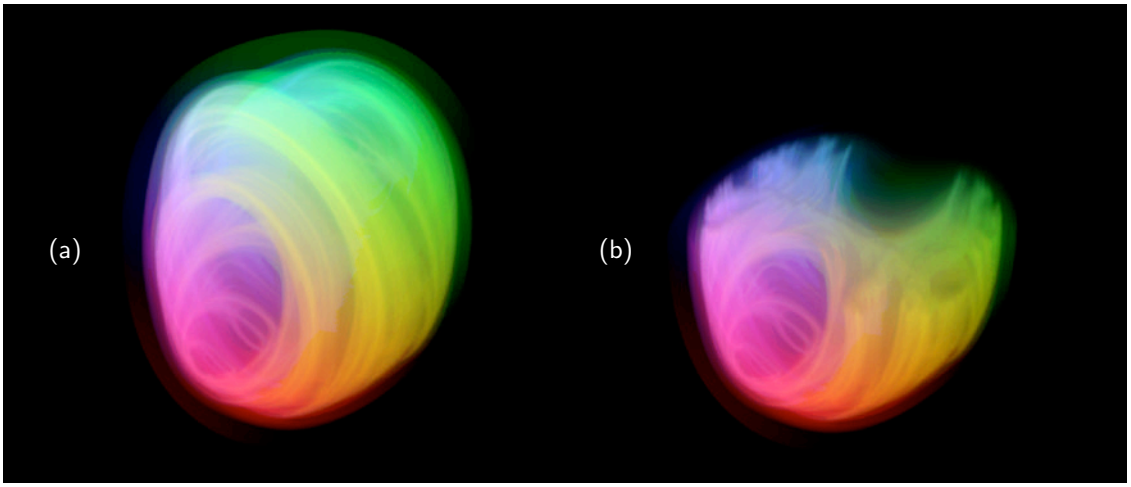


Figure 5.11: (a) Julia-Fractal stored in a volume texture with 512^3 voxels. (b) Soft clipping of the Julia-Fractal.

fragment, its coordinates are available to the shader as frag_x, frag_y (both in pixel units) and frag_z (in the range $[0, 1]$). The shader then computes the coordinates for the depth texture lookup by:

$$t_x = \frac{\text{frag}_x}{\text{width}}, \quad t_y = \frac{\text{frag}_y}{\text{height}}.$$

Finally, the comparison result between $D(t_x, t_y)$ and frag_z determines whether the current fragment is discarded or not.

6 The Clip Plane

As mentioned in the previous chapters, the application developed in this thesis supports a user definable clip plane. To this end, a plane equation is passed to the various fragment shaders, which then perform the actual clipping. An alternative would have been to perform the clipping in software. Since each mesh consists of about 300.000 triangles, however, this approach is not feasible.

Definition. The clipping plane is defined by a normal vector $\mathbf{n} = (n_x\ n_y\ n_z)^\top$ and a distance d from the origin. All points $\mathbf{p} = (x\ y\ z)^\top$ lying on the plane satisfy the three-dimensional plane equation:

$$n_x \cdot x + n_y \cdot y + n_z \cdot z = d,$$

or, more simply:

$$\mathbf{n} \cdot \mathbf{p} = d.$$

If a point lies above the clipping plane, $\mathbf{n} \cdot \mathbf{p} > d$ holds true and $\mathbf{n} \cdot \mathbf{p} < d$ if it lies below. Note that for both $\mathbf{n}$ and $\mathbf{p}$, world coordinates are used. (And it is assumed that $\mathbf{n}$ has unit length.)

Clipping Inside the Fragment Shader. What is actually passed to the fragment shader are the three normal vector coordinates and d. In order to determine whether a given fragment lies above or below the plane, it needs access to the current fragment's world position. This information is computed in the corresponding vertex shader by transforming the current vertex by the model matrix. The result is then passed to the fragment shader. (OpenGL does not distinguish between model and view matrices, but instead combines them into the modelview matix. For this reason, the model matrix has to be explicitly passed to the vertex shader by the application as a uniform variable.)

6.1 The Visible Clip Plane

In addition to the abstract clipping plane above that only manifests itself by missing fragments of meshes and volumes, a real clip plane is also rendered. This is

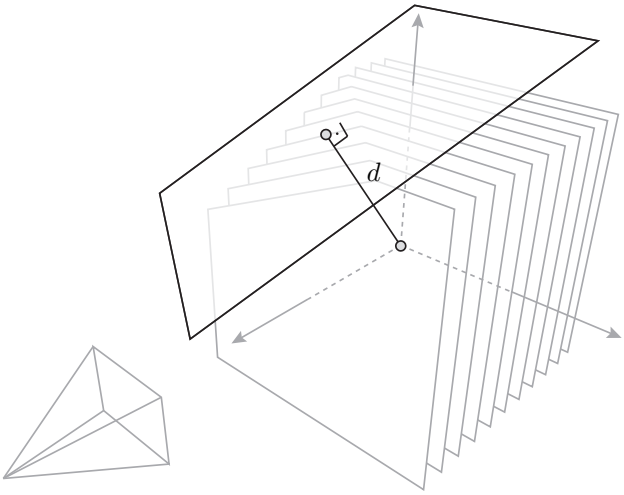


Figure 6.1: A user adjustable clip plane with distance d from the volume center. While the volume slices are always rotated to face the camera, the clip plane stays fixed.

done by aligning a square to have the same orientation and distance from the origin as the abstract clipping plane. This square will be used to render both the MRI data as well as gray matter color labels. Therefore, the size and placement of this square has to be chosen in such a way that it always encompasses the entire brain geometry at the currently set clipping plane. If it is chosen big enough, the actual rotation of the square on the clipping plane is irrelevant. Also, volume texture coordinates have to be assigned to the four corners of the square, since access to the MRI volume texture is obviously needed. These are exactly the properties of the individual slices used for the volume rendering, so it is easy to implement this square as just another one of those slices. The only difference is that the square has a user defined distance d from the volume center and is not automatically rotated to face the camera. See Figure 6.1 for a schematic picture.

6.2 Gray Matter Coloring

As described in Section 4.3: Surface Area Coloring, the outer and and inner surfaces are already colored according to the surface labels. When using a clipping plane to cut off parts of the surfaces, however, two holes appear (see Figure 6.2). In reality, these holes would be filled with white and gray matter. Therefore, to create the impression of a solid object, a cap will be rendered onto these holes by using

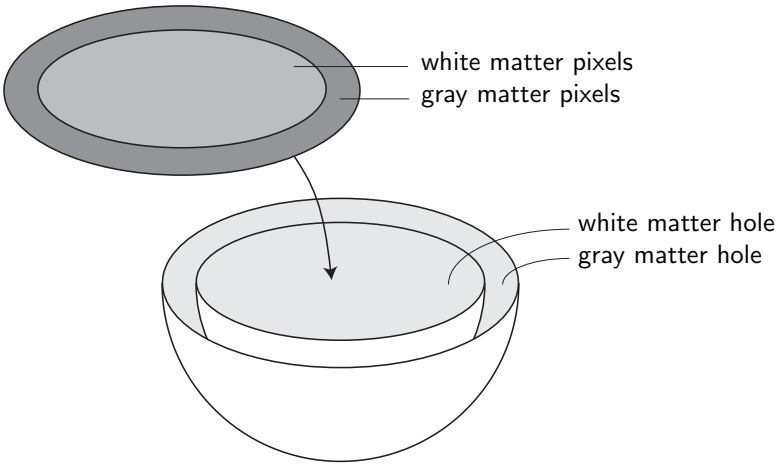


Figure 6.2: When the upper parts of the outer and inner surfaces are clipped away, two holes appear. A cap is rendered on top in order to "fill" these holes.

the visible clip plane from the previous chapter. When doing so, all pixels on this plane that belong to the gray matter—the *gray matter pixels*—are colored with the same colors as the ones used in the surface coloring. The *white matter pixels* have a uniform color in the final visualization. Therefore, there is no need to render them in this pass, since they correspond to the *core pixels* of render pass Mesh Below 3 and are handled in the final composition (see Chapter 7). However, if sub-cortical structures such as the hippocampus or the amygdala were to be included on the clip plane, this pass would be the right place to render them.

Accessing Gray Matter Volume Labels and Colors. When rendering the clip plane, access to the MRI volume labels produced by FreeSurfer is needed. These are stored in a volume texture with 256^3 texels. As with surface labels before, unpleasant banding artifacts at label borders would appear through texture interpolation if actual label numbers were stored in the texture. So, the label colors are stored directly in the RGB component. This way, the clip plane fragment shader can simply access the volume texture to determine the color of a given pixel. A rendering of the clip plane, overlayed on a rendering of the surface mesh is shown in Figure 6.3. It is very noticeable that the surface mesh boundaries and the volume color boundaries do not match.

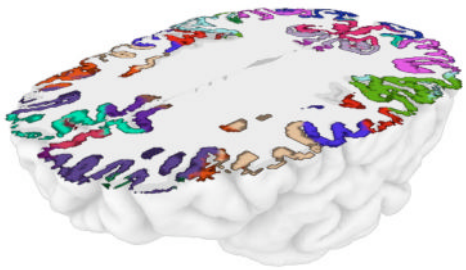


Figure 6.3: Coloring of the gray matter pixels on the clip plane. Below, the rendered surface mesh is shown.

Improving the Rendering Quality. The inherent problem of the previous approach is that the volume texture used to determine gray matter pixel colors has a much lower resolution than the actual rendered image. In fact, by zooming in on a small area of the surface, no matter how high the resolution of the volume texture is, at some point it will always be too low to accurately match the boundary of the surface meshes. The solution devised in this thesis is as follows:

- Before the volume texture is processed to store RGB colors, the labels are expanded: For every voxel, all direct neighbors are examined. If any of these neighbors has no assigned label yet, it is assigned the label of the current voxel. This process is repeated a few times. The output of this stage can be seen in Figure 6.4 (a). Note that this approach can be considered as being similar to a morphological dilation [Haralick et al., 1987].

- The output texture of render pass Mesh Below 3 (see Figure 4.4) is passed to the fragment shader. There is used to determine the pixel category of the current fragment (which is stored in the blue component of the texture). Only if it is an *inner pixel*, the gray matter color is read from the volume texture and written to the output. The result can be seen in Figure 6.4 (b). Now, both the surface boundaries and the gray matter color boundaries are aligned perfectly.

It might be argued that the first step of expanding the surface labels introduces anatomical inaccuracies in the rendered output. While this certainly holds true to some extent, the same would also hold true for the reconstructed surface meshes, so this is not just a problem of this rendering step. Also, the accuracy will only get better as the used volume textures gain higher resolutions. Even then, the presented technique is advantageous as it ensures that no visual glitches appear.

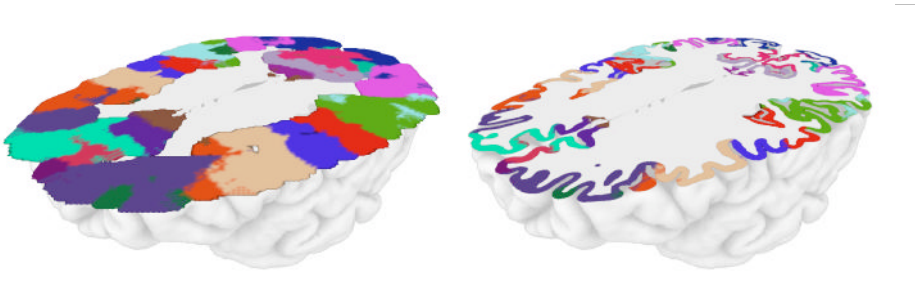


Figure 6.4: Improving the gray matter boundary. (a) Expanded labels. (b) Only those pixels marked as *inner pixels* in render pass MESH BELOW 3 are rendered.

Render Pass: CLIP PLANE

The gray matter colors are determined inside a fragment shader as described above and stored in the RGB component. Further, the raw MRI data is also read from the volume texture and stored in the alpha component (see Figure 6.5). Although this data is not directly used to produce the final image, it can be later superimposed on the rendering to gain additional insights on inner brain structures.

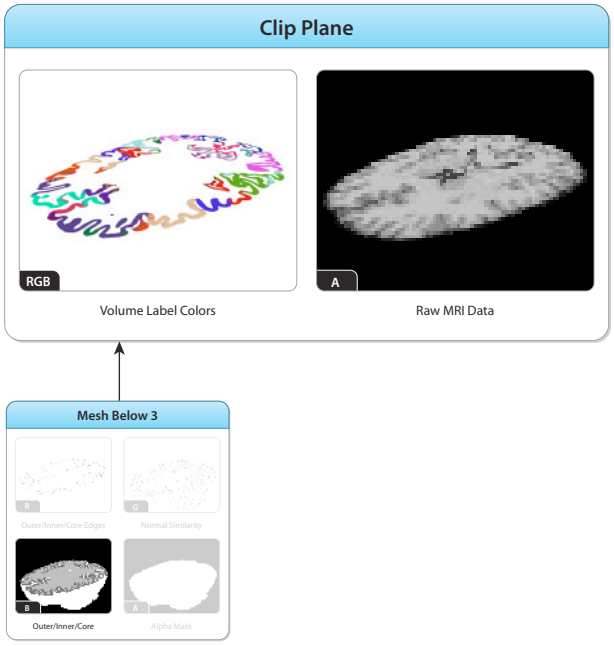


Figure 6.5: Output of the CLIP PLANE render pass.

7 Compositing the Brain Image

In the previous chapters, the basic building blocks of the visualization algorithm have been presented. In this chapter, they are combined to produce the final rendering of the human brain. For each composition pass, two screenshots will be presented in this chapter: one where the camera is above the clipping plane, and one where it is below. Additionally, all inputs for the currently described pass are shown. Lowercase greek letters ($\alpha, \beta, \ldots$) will be used to denote parameters that can be adjusted in the code to fine-tune the visual output.

All composition passes except for the first will follow the same general outline:

- The value already stored in the framebuffer is treated as the incoming radiance for the currently processed fragment.
- The shader computes an absorption factor that is used to attenuate the incoming radiance.
- The shader computes the emission for the current fragment, which is added to the framebuffer.

As before, in Chapter 5, this behavior is achieved by setting the OpenGL blending function to (GL_ONE, GL_ONE_MINUS_SRC_ALPHA).

Also, there are two kinds of composition passes:

1. *Volume Passes*—these are passes that use the volume rendering shader described in Chapter 5. They always have the fMRI volume texture as an input, as well as the transfer function texture.
2. *Image Space Passes*—these are passes that render a screen sized rectangle in order to compose results from previous passes (e.g., the brain surfaces, the clip plane, etc.) into the framebuffer.

Image Space Pass: BACKGROUND

First, the framebuffer is initialized to a gray background. This will serve as the canvas for the coming rendering passes. As noted previously, this gray background can be interpreted as resulting from a uniform diffuse background lighting environment.

Volume Pass: LOWER VOLUME

For this pass, the volume shader is set to only render fragments that lie below a given depth value. This depth value is read from the alpha component of MESH BELOW 2. This way, only those parts of the volume are rendered that will later be obscured by the lower brain surface. (See Figure 7.1.)

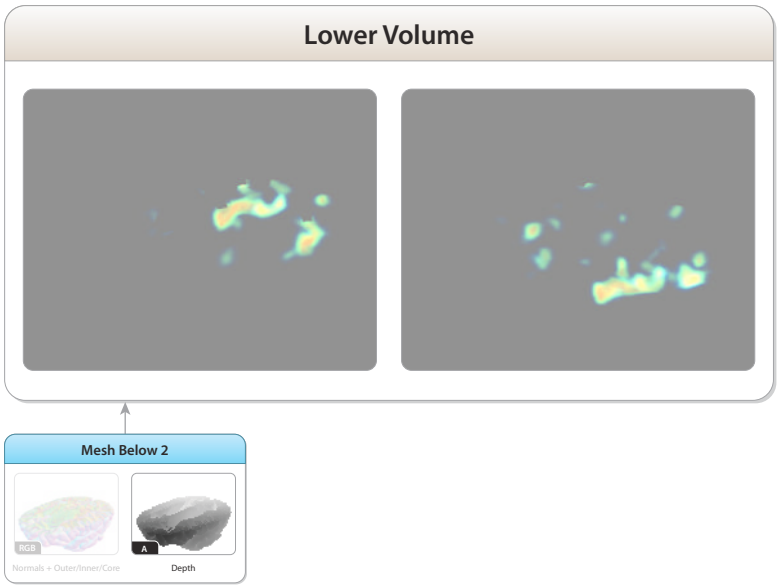


Figure 7.1: Output from the LOWER VOLUME render pass.

Image Space Pass: LOWER SURFACE

This pass is responsible for composing the lower brain surface, i.e., the part of the surface that lies under the clipping plane, into the framebuffer (see Figure 7.2). It does this by performing the following steps for each pixel:

- If the alpha mask stored in MESH BELOW 3 is 0 at the position of the the pixel, it is discarded.

- *Absorption:* The category (outer, inner, core) of the current pixel is determined by reading the blue component of MESH BELOW 3. Depending on the category, a different pre-defined absorption factor (α_{outer}, α_{inner}, α_{core}) is used. This provides a convenient way to fine tune the resulting brain image. In the rendering shown below, only the outer pixels are set to absorb light, as the absorption of the inner and core pixels is handled at a later stage. The exact values are $\alpha_{\text{outer}} = 0.95$, $\alpha_{\text{inner}} = 0$, $\alpha_{\text{core}} = 0$.

- *Emission:* Only the outer pixels actually emit light in this pass. To determine the actual color, first MESH BELOW 1 is accessed to read the surface label color $\mathbf{c} = (c_r\ c_g\ c_b)^\top$ from the RGB component and the ambient occlusion shading result a from the alpha component. If $\mathbf{b}$ denotes the background color used in the first pass, then the final emission $\mathbf{e}$ is computed by:

$$\mathbf{e} \;=\; [\beta(\mathbf{c}-\mathbf{b})+\mathbf{b}] \;\cdot\; [1-\gamma(1-a)]\,,$$

  where both β and γ lie in the range $[0,1]$. The idea behind this formula is as follows: β determines how much the surface label color will differ from the actual background color. In this work, since a gray background is used, it can be intuitively understood as a saturation factor—the higher β, the higher the saturation. γ on the other hand determines how strong the ambient occlusion shading manifests itself. The bigger γ is, the stronger the surface color is darkened due to ambient occlusion. In the rendering shown in Figure 7.2, both β and γ equal $\frac{1}{4}$.

- *Silhouettes:* Silhouettes for the outer and inner surfaces could be drawn in this pass, but they are postponed until after the next pass. This is possible, because no silhouettes are drawn on the inside surfaces.

- Additionally, this pass also reads the depth value stored in the alpha component of MESH BELOW 2 and writes it to the z-buffer.

Volume Pass: MIDDLE VOLUME

This volume rendering pass is responsible for drawing all volume elements that

a) lie above the z-value written into the z-buffer by the previous pass, and

b) lie under the clipping plane.

While a) is taken care of by the standard OpenGL depth checking, b) is handled by the soft clipping described in Chapter 5. See Figure 7.3 for the output of this pass.

Image Space Pass: MIDDLE SURFACE

In this pass, the clip plane is composed into the framebuffer. Also, silhouettes for both the outer surface (which have been postponed before) and for the clip plane will be added (see Figure 7.4). The computations performed for each pixel are:

- As before, the texture from MESH BELOW 3 is queried for the stored alpha mask and the pixel is discarded if it falls outside of that mask. However,

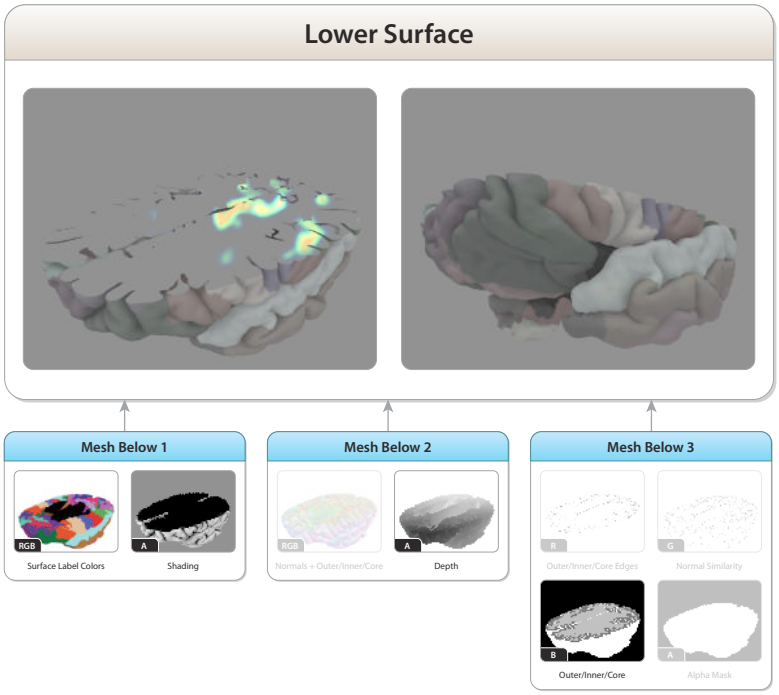


Figure 7.2: Output from the LOWER SURFACE render pass.

there is one additional challenge at this stage: Further below, silhouettes will be composed onto the image. The discontinuity detection algorithm used to create these silhouettes produces lines that are two pixels wide. (This is because the discontinuity actually occurs *between* two pixels and will therefore be detected on both the left and right, or top and bottom side the discontinuity.) These thick edges are preserved in the output, since 1 pixel edges are too thin for this visualization style. Therefore, not only the alpha value of the current pixel is determined, but also those of the neighboring pixels. If any of them has a value over 0, the current pixel is not discarded. (This is again very similar to a morphological dilation, see [Haralick et al., 1987].)

- *Absorption:* If the current pixel is an inner or core pixel, the absorption is set to δ_{inner} and δ_{core}, respectively. Otherwise it is set to zero. Note that conceptually, the α_{inner} and α_{core} absorption factors from the LOWER SURFACE pass account for the absorption of the brain surface walls, whereas δ_{inner} and δ_{core} from this pass account for the absorption of the clip plane (which is the reason why there is no δ_{outer}). In the rendering shown below, both δ_{inner} and δ_{core} are set to 0.85.

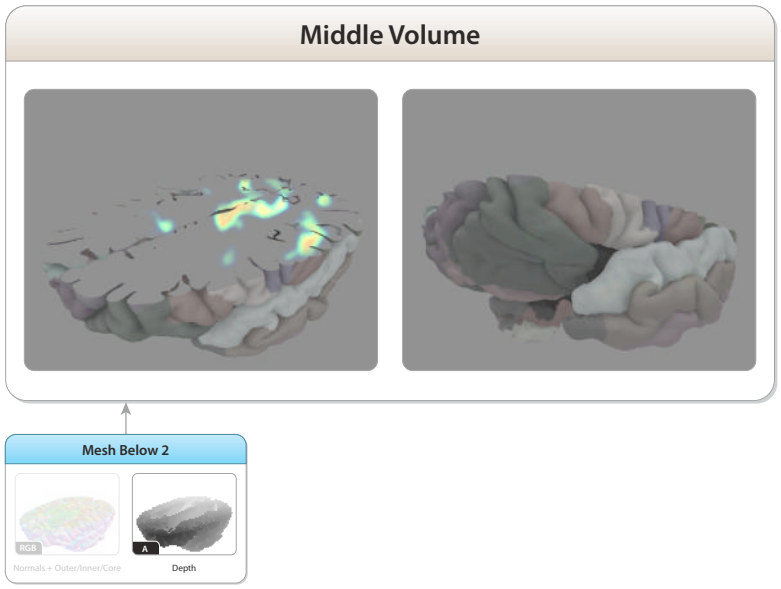


Figure 7.3: Output from the MIDDLE VOLUME render pass.

- *Emission:* The clip plane emission is computed by taking the background color $\mathbf{b}$ as a base and darkening it by a factor ζ. ζ is chosen as 0.6 for inner pixels and by 0.8 for core pixels. Then, for inner pixels, the label color stored in the RGB component of CLIP PLANE is read, de-saturated by a factor η that is set to 0.8 in this work, and finally added to the emission.

- *Silhouettes:* Silhouettes are read from both the red and green channel of MESH BELOW 3. If a silhouette exists at the currently processed pixel, any previous computations of absorption and emission are discarded and replaced by an absorption of 1 and an emission equal to the silhouette color (dark gray in the rendering below). It turned out that the silhouette formed by the white matter / gray matter boundary actually degraded the clarity of the final output, so it was discarded in the rendering seen below. (Discarding the white matter / gray matter silhouette is very easy, since it has been stored at half intensity in render pass MESH BELOW 3.)

Volume Pass: UPPER VOLUME

This is the final volume rendering pass. It is responsible for drawing all fragments that lie above the clipping plane. Again, soft clipping as described in Chapter 5 is used. See Figure 7.5 for the output of this pass.

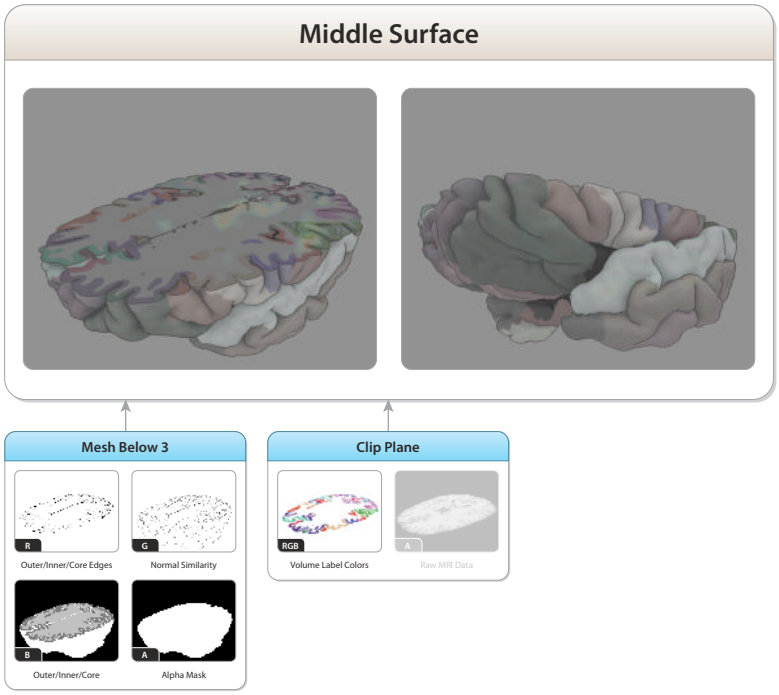


Figure 7.4: Output from the MIDDLE SURFACE render pass.

Image Space Pass: UPPER SURFACE

This pass is responsible for compositing both the shading and the silhouettes of the upper brain surface into the framebuffer. As such, it has to perform tasks that are very similar to the ones described in LOWER SURFACE and MIDDLE SURFACE. The few differences are:

- A white color is used for the silhouette that belongs to the object boundary.
- The brain surface does not absorb any light and emits only a very faint color.
- Silhouettes on the surface are not drawn directly, but only have the effect of setting the emission of the current pixel to zero.

The result of this step, and therefore the final result of the composition pipeline is shown in Figure 7.6.

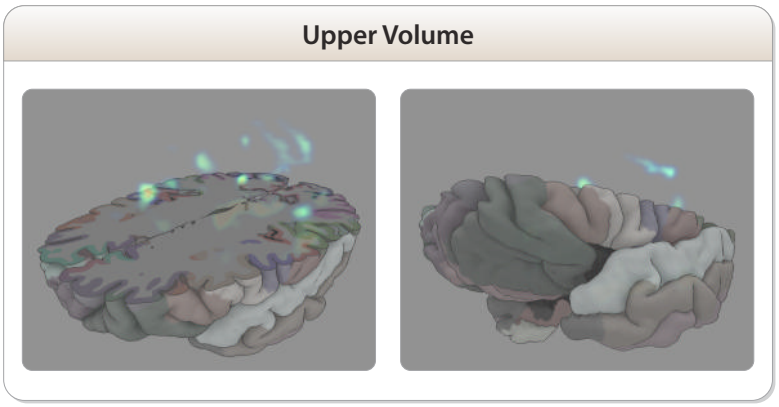


Figure 7.5: Output from the UPPER VOLUME render pass.

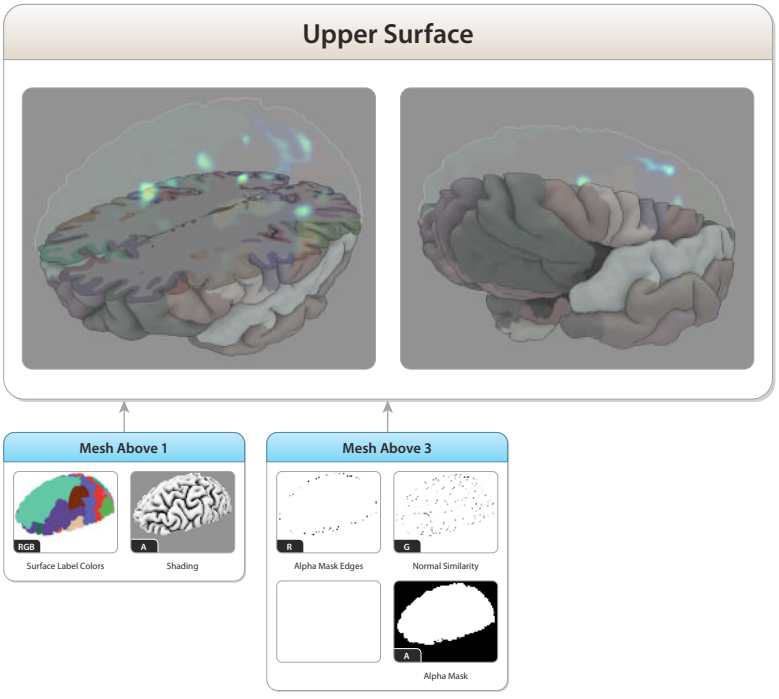


Figure 7.6: Output from the UPPER SURFACE render pass.

8 User Interaction

The output of the rendering pipeline is presented to the user inside a window and he/she is given various tools to interact with the presented scene. These range from simple manipulation tools that allow adjustment of camera and clipping plane parameters to more complex ones like fMRI and surface point selection.

8.1 Camera Control

The camera is defined by a target position $\mathbf{t}$; a yaw ϕ and a pitch θ, both in $[0, 2\pi]$, that determine its orientation; a target distance d and a field of view $f \in [\frac{1}{180}\pi, \frac{179}{180}\pi]$. (A field of view of 180° has no meaning in the context of a perspective projection. Neither does a field of view of 0°; therefore the minimal and maximal value is set to 1° and 179° respecively). For non-quadratic render targets, an aspect ratio $r = \frac{\text{width}}{\text{height}}$ is also stored. When a frame is rendered, the camera viewing direction can be computed through ϕ and θ, and the actual camera position can be computed by starting at $\mathbf{t}$ and going back d units along that computed vector. The *camera viewing plane* is the plane passing through $\mathbf{t}$ that is perpendicular to the viewing direction.

Basic Controls. The user can use the mouse to move the camera target position $\mathbf{t}$ either on the x/y-plane or in z-direction. Zooming in and out on the scene is accomplished by simply changing the target distance d. The camera's yaw and pitch can also be independently increased or decreased by moving the mouse left/right or up/down. By computing the camera orientation from these values, no discontinuities at the poles are produced and therefore, Gimbal lock [Shoemake, 1985] is avoided.

From Perspective to Orthographic Projection. The camera is initialized with a field of view of 60°. When narrowing the field of view while keeping the target distance fixed, a smaller portion of the actual scene will be rendered. This can be counterbalanced by simultaneously zooming out, i.e. increasing the target distance. The combined effect is that the "perspectiveness" of the scene gradually decreases until, in the end, orthographic projection is reached. (For the projection to be truly orthographic, the target distance would have to be infinite and the field of view would have to be 0. Flipping to a true orthographic projection matrix once a field

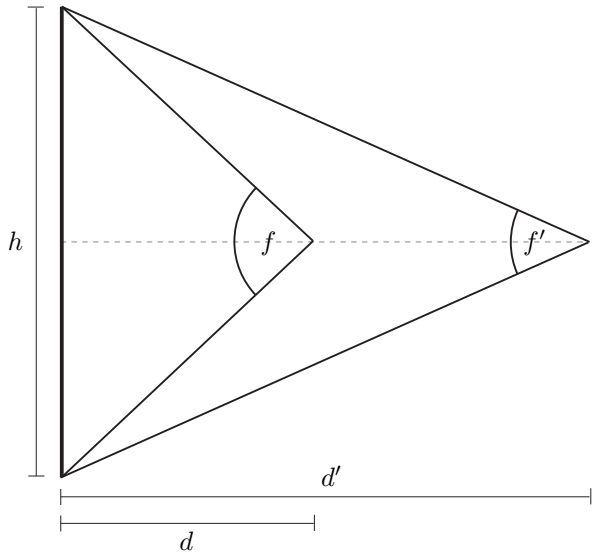


Figure 8.1: Two camera configurations, (f, d) and (f', d'), that have the same camera viewing plane height h.

of view of 1° is reached would be possible if perfect orthographic projection was needed.)

The user is given the possibility to gradually increase or decrease the "perspectiveness" by simply dragging the mouse up and down while pressing a modifier key. If the field of view is to be changed by an amount of Δf, then first, the current height h of the camera viewing plane in world coordinates is computed:

$$h = 2 \cdot d \cdot \tan(f/2).$$

If this height is to stay the same after the field of view changed to $f' = f + \Delta f$, the new camera distance d' has to be set to (Figure 8.1):

$$d' = \frac{h}{2 \cdot \tan(f'/2)}.$$

Figure 8.2 shows the scene when rendered with different field of view values.

Automatic Near and Far Clipping Plane Adjustment. Before each frame is rendered, the bounding box of the scene is used to adjust the near and far clipping planes of the camera. This is necessary in order to ensure that no depth-buffer accuracy is wasted and as a consequence, that no z-fighting occurs. This is done by

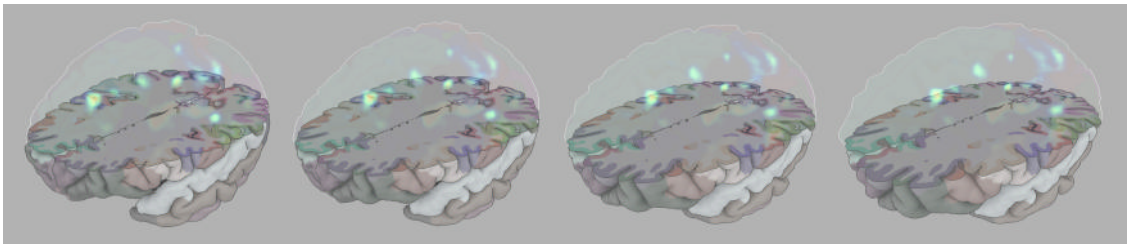


Figure 8.2: Renderings with increasing field of view from left to right. The leftmost rendering has a field of view of 1°, thereby having almost orthogonal projection.

encompassing the scene bounding box with a bounding sphere (with radius r_{sphere} that is half the diameter of the bounding box). The center of that sphere is projected on the camera viewing direction, and the distance d_{sphere} from the camera is computed. The near and far clipping planes are then set to $d_{\text{sphere}} - r_{\text{sphere}}$ and $d_{\text{sphere}} + r_{\text{sphere}}$, respectively.

8.2 Clip Plane Control

In order to set the parameters of the clipping plane, the user can go into *Clip Plane Mode*. While in this mode, initiating a mouse dragging operation changes to the *Clip Plane Rotation Mode*. By further pressing or releasing a modifier key, the *Clip Plane Height Adjustment Mode* is activated or deactivated.

Clip Plane Rotation Mode. As described in Chapter 6, all points $\mathbf{p}$ lying on the clip plane satisfy the equation:

$$\mathbf{n} \cdot \mathbf{p} = d,$$

where $\mathbf{n}$ is the plane normal and d is the plane's distance from the origin. A simple way to rotate the clipping plane in response to the user input would be to simply rotate its normal $\mathbf{n}$ while keeping the distance d fixed. The specifics of how to translate mouse movements to actual rotations will be defined later. For now, it is important to point out, that this simple method can result in unintuitive behavior for some scene configurations. Figure 8.3 depicts one such situation. Starting with a scene as seen in Figure 8.3 (a), the user decides to tilt the plane towards himself (depicted by the black arrow). If the user's mouse movement translates to a rotation of $\alpha = 30°$, then Figure 8.3 (b) shows the resulting clip plane position, where the outcome shown in Figure 8.3 (c) is probably what the user expected. While the clip plane has an infinite extent in theory, the user's indirect view of the clip plane is limited to the extent of the intersected scene. With no further point of reference,

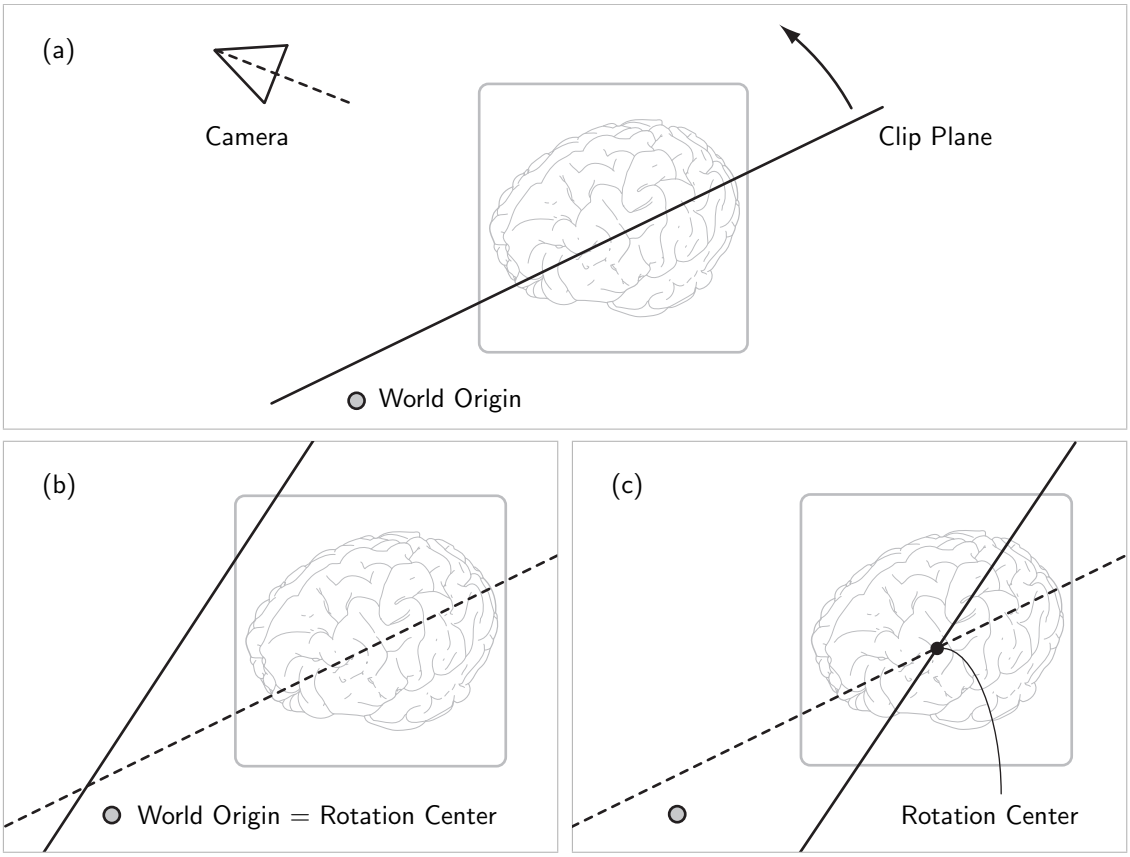


Figure 8.3: (a) The scene is viewed from the camera and the user drags the mouse to tilt the clip plane by 30°. (b) and (c) show possible outcomes, each using a different center of rotation.

the most reasonable assumption on the users part is that the clip plane will rotate around the center $\mathbf{c}_v$ of the visible part of the plane.

A second problem with the simple approach outlined above is that once the plane's distance from the origin is greater than the distance of the scene geometry to the origin, successive rotations of the clipping plane have no visual effect. This is shown in Figure 8.4. To the user, the plane will seem "lost".

To address both of these problems, the following approach is taken:

- The point $\mathbf{c}_g$ that lies at the center of the scene geometry and the point $\mathbf{c}_r$ that lies on the clipping plane and is closest to $\mathbf{c}_g$ are computed (see Figure 8.5 (a)).
- When the user starts to rotate the clip plane, the point $\mathbf{c}_r$ is used as the

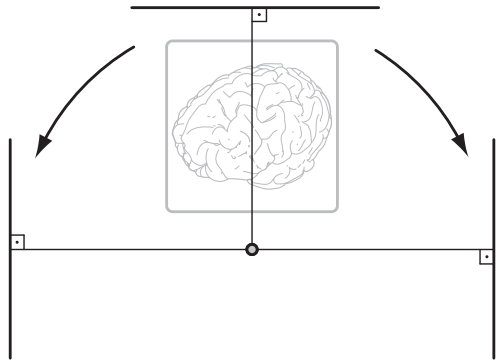


Figure 8.4: In the depicted scene, no matter how the user rotates the clipping plane, it will always lie completely outside of the scene. It is "lost".

rotation center. After every incremental rotation, the clip plane equation is updated to give the user immediate feedback. The rotation center $\mathbf{c}_r$, however, remains fixed until the user stops the rotation process by either releasing the mouse button or pressing the modifier key to enter *Clip Plane Height Adjustment Mode*. Only then is the rotation center $\mathbf{c}_r$ updated to reflect the changed clip plane equation (see Figure 8.5 (b)).

The implications of this approach are two-fold. First, the way that the rotation center $\mathbf{c}_r$ is computed ensures that it will be close to the center of the visible portion of the clip plane (if the plane is visible at all), which seems like an intuitive rotation center. Second, even if the distance of the clipping plane to the origin is greater than that of the scene, there is always a rotation the user can apply to make the plane visible again. Also, after every rotation, the distance between $\mathbf{c}_r$ and $\mathbf{c}_g$ is guaranteed to decrease (see Figure 8.5 (b)). This way, even if the clip plane starts far away from the scene center, after a few rotations, the rotation center $\mathbf{c}_r$ will be very close to $\mathbf{c}_g$. The situation of a "lost" clip plane is therefore largely avoided.

Rotation Axes. When the user drags the mouse to the left or to the right, the clip plane normal is rotated around the camera's local up-axis; when the mouse is dragged up or down, it is rotated around the camera's local right-axis (see Figure 8.6).

Clip Plane Height Adjustment Mode. When entering height adjustment mode, dragging the mouse up or down translates the clip plane along the plane's normal in a positive or negative direction, respectively. After every translation, the rotation

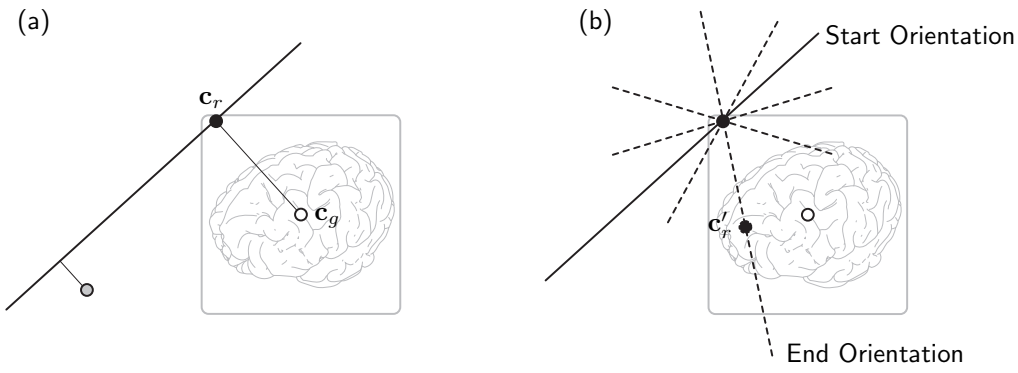


Figure 8.5: (a) The center of the scene geometry, $\mathbf{c}_g$ is projected on the clip plane to compute the rotation center $\mathbf{c}_r$. (b) When the user initiates a rotation, the previously computed rotation center is used for successive rotations (dashed lines). When the user ends the rotation, the new center of rotation $\mathbf{c}'_r$ is computed.

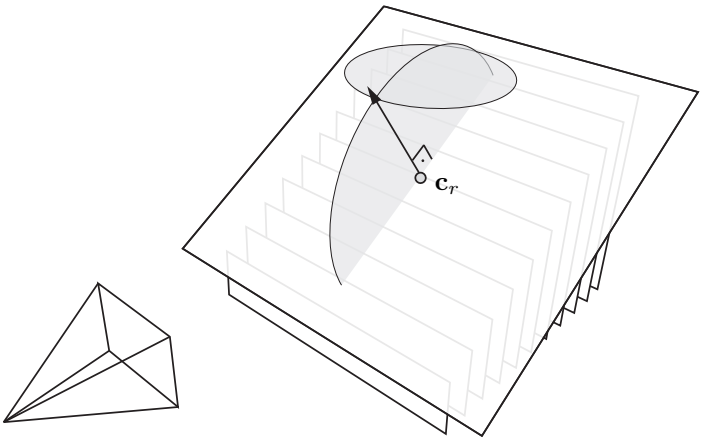


Figure 8.6: Rotation of the clip plane normal (black arrow) around the x- and y-axes of the camera, centered at $\mathbf{c}_r$.

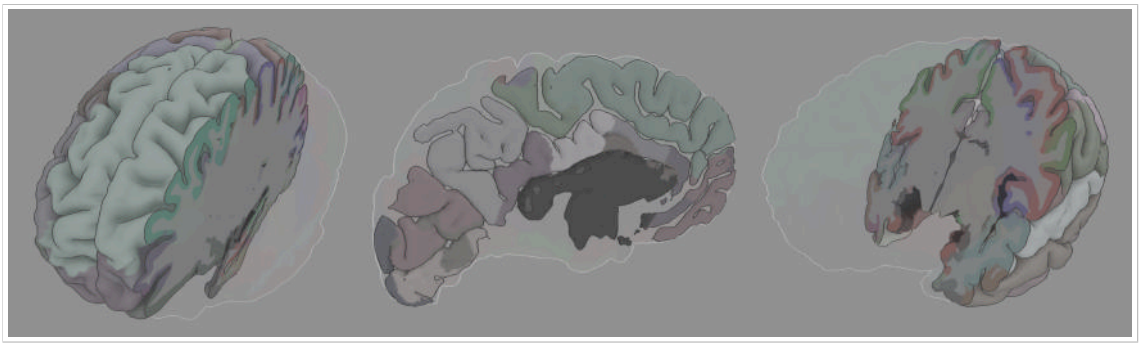


Figure 8.7: Different orientations and heights of the clip plane. In the middle picture, only one brain hemisphere is rendered.

center $\mathbf{c}_r$ is updated to reflect the new clip plane position.

Results. Figure 8.7 shows different orientations and heights of the clip plane.

8.3 fMRI Selection

When looking at fMRI activation volumes like the one shown in Figure 8.8, the human observer is capable of distinguishing different activation areas. Some, like the one labeled with "A", have an easily recognizable boundary and point of maximum activation. Other areas, like the one labeled with "B" or "C", have more complex boundaries and a point of maximum activation is not easy to spot. If different observers were asked to draw borders for such areas, it is likely that they would come up with different segmentations.

The goal of this chapter is to devise a mechanism by which the user can easily select different activation areas by pointing and clicking with the mouse. After an area has been selected, the point of maximum activation inside that area can be determined and used for further operations (e.g., to compute its distance to a cortex surface point).

As noted before, areas similar to B have somewhat ambiguous borders, so it can not be expected to find an algorithm whose outcome always matches the user's intentions. In order to implement the proposed selection mechanism, the following two steps are taken:

1. Activation areas inside the fMRI volume are determined and all voxels are classified with respect to the area they belong to. A simple algorithm is de-

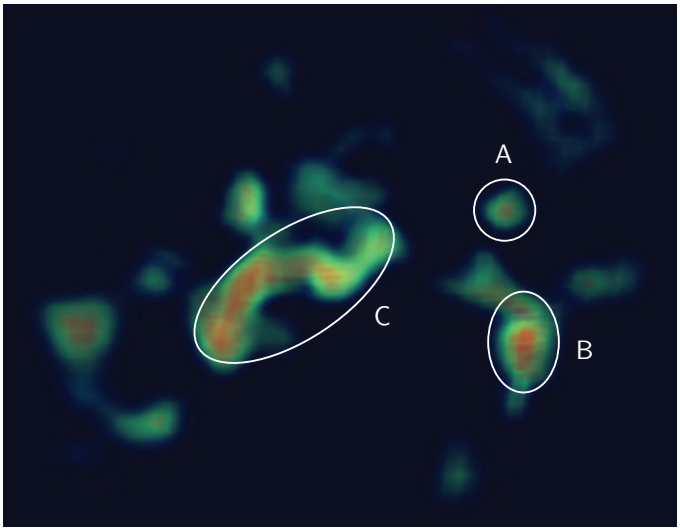


Figure 8.8: Different activation areas. Some have easily determinable boundaries (A), some are more complex (B, C).

scribed in Section 8.3.1 and an improvement is described in Section 8.3.2: The Watershed Algorithm.

2. When the user clicks on one part of the projected fMRI volume, the closest activation area is chosen. This will be the topic of Section 8.3.3: Volume Rendering for Picking.

8.3.1 A Simple Segmentation Approach

Given an fMRI activation volume, a simple approach for finding the different activation areas would be the following:

A) Find the voxel with the highest value that does not already belong to an area. Create a new area consisting only of that voxel. Figure 8.9 (a) shows this for the two-dimensional case.

B) Grow that area to all sides by recursively inspecting all neighboring voxels v of the area. If the intensity value of the currently inspected voxel v is less than any of its neighbors that also belong to the area, include v in the area. Stop, when no more such voxels exist for the current area (see Figure 8.9 (b)).

C) If there are still voxels left in the volume that belong to no area, go back to A. Otherwise, all areas have been found (see Figure 8.9 (c)).

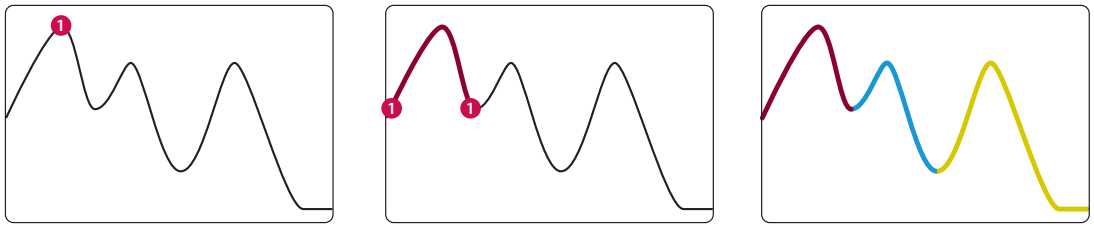


Figure 8.9: Naive segmentation algorithm. (a) Starting from the highest point, a new area, "1", is created that is expanded by descending from the hilltop. (b) When reaching a local minimum, the algorithm stops. This process is repeated until all areas have been found (c).

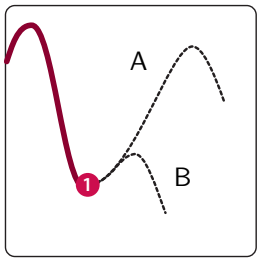


Figure 8.10: Two possible continuations of the curve. In case A, a new area should probably be created, while in B, the current area (**1**) should be further expanded.

There is one major drawback when using this algorithm, however. When a situation as the one depicted in Figure 8.9 (b) is encountered, there is no way of knowing if the local increase in intensity is due to a small irregularity of the signal, or if it marks the beginning of another area (see Figure 8.10). Whether the first or the latter is true mainly depends on the height of encountered "hill" relative to the current position. Since the design of the algorithm does not provide this information at this stage, there is no choice but to stop processing the current area and creating a new one. This naturally results in an over-segmentation of the volume. The *Watershed Algorithm* addresses precisely this problem [Roerdink and Meijster, 2000].

8.3.2 The Watershed Algorithm

Typical descriptions of the watershed algorithm start by first negating the signal, i.e., all maxima are turned into minima and vice versa. The segmentation process is then described by a slowly raising water-level. The following description does not rely on this watershed allegory.

A) A horizontal scan-line is run from the top to the bottom of the signal. Once it hits the top of a hill, a new area is created (Figure 8.11 (a)) and the intensity value, or "height" of the hill-top is stored. (Note that the algorithm is described in terms of a one-dimensional signal. For three-dimensional signals, the scan-line corresponds to a hyperplane in four dimensions.)

B) As the scan-line moves downward, existing areas are expanded as long as the intensity values are decreasing, and newly encountered hill-tops generate new areas. See Figure 8.11 (b).

C) If two area-borders collide as the scan-line descends, it is decided whether or not those areas are merged. In contrast to the simple algorithm before, the height of the hills on both sides of of the collision point are already known. Looking at the smaller hill, if its relative height h to the collision point is smaller than a user defined threshold τ, the areas are merged (see Figure 8.11 (c)). Otherwise, they are kept separate (Figure 8.11 (d)).

D) Once the scan-line reaches the bottom, the algorithm stops (see Figure 8.11 (e)).

Note that the threshold τ described here is otherwise known as the *pre-flood height*. If τ is set to zero, this algorithm produces the same output as the simple segmentation algorithm described above. The higher τ is, the more previously separate areas are merged together. Finding a good value for τ depends highly on the nature of the data, but for the available fMRI data used in this thesis, a value of approximately $\frac{1}{25}$ has been found to give good results. (It is assumed that fMRI activation values lie in the range $[0, 1]$). Figure 8.12 shows the resulting areas for different threshold values. In those pictures, the scan-line has not been run all the way to the bottom, since this would have included all black voxels that lie between the activation areas. Instead, the scan-line was halted when it reached the height $\frac{1}{25}$.

8.3.3 Volume Rendering for Picking

Each of the areas computed by the watershed algorithm is assigned a number in the range $[1, 255]$, 0 being reserved for voxels that belong to no area.. For each voxel, its corresponding area number is then stored in the alpha component of the fMRI volume texture. The moment the user clicks on the screen, the entire fMRI volume is rendered into an off-screen texture. This is done using a special fragment shader that writes area numbers to the output texture. This texture is then accessed at the location that corresponds to the user's click location and the area number is determined. The picking render pass is shown in Figure 8.13. A few details of this pass are worth noting:

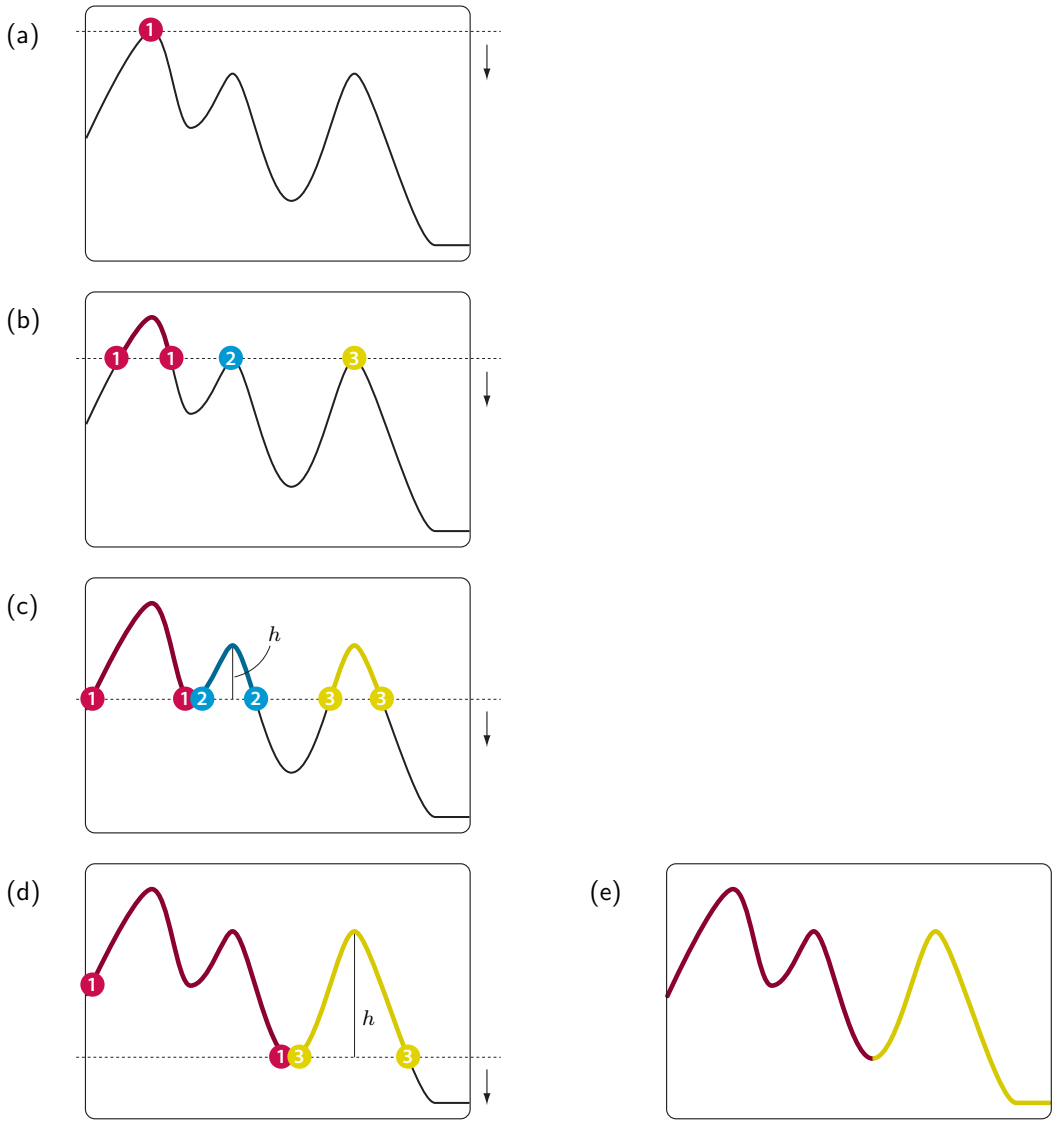


Figure 8.11: Watershed Algorithm. (a) The scan-line encounters a point that belongs to no area, so a new area, **1**, is created. (b) Area **1** has been expanded and two new areas have been created. (c) Areas **1** and **2** collide. It is checked, whether the relative height h of area **2** to the collision point is below a user-defined threshold. In this case it is, so area **2** is subsumed into **1**. (d) Areas **1** and **3** collide. The height of **3** relative to the collision point is compared to the threshold. In this case, it is bigger, so the two areas are left separate. (e) The scan-line has reached the bottom and the resulting areas are shown.

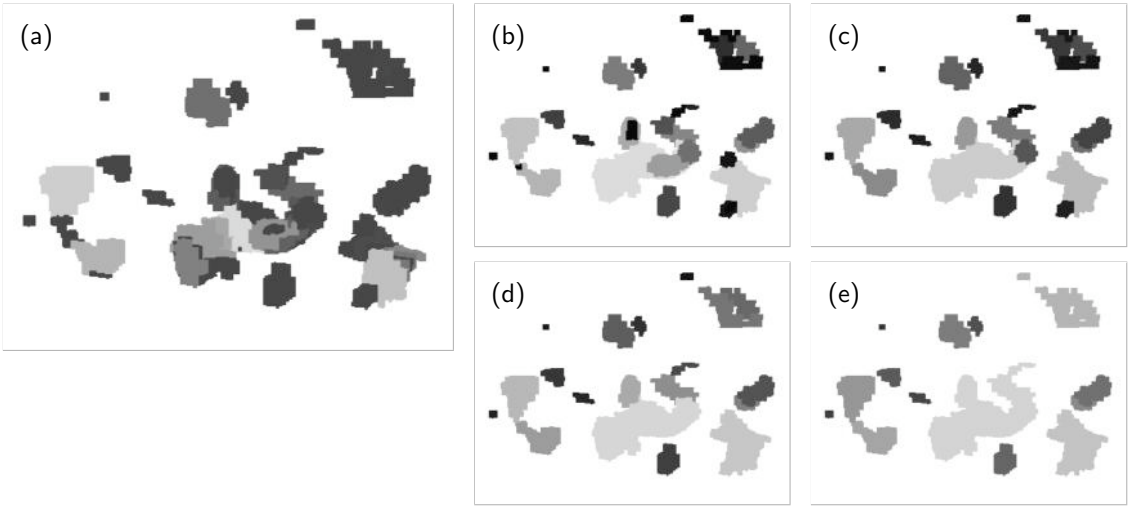


Figure 8.12: (a) Watershed threshold τ is set to 0, which results in an over-segmentation. In (b) trough (e) the threshold is set to $\frac{1}{50}$, $\frac{1}{25}$, $\frac{1}{10}$ and $\frac{1}{5}$, respectively.

- When accessing the volume texture, no filtering is performed. While the previously used volume shaders all used tri-linear filtering to blend between the voxels of the volume, this approach is not feasible here. The reason is the same as the one described for the surface labels, i.e., if two neighboring voxels have area numbers n_1 and n_2, interpolation between those values will produce area numbers over the entire range $[n_1, n_2]$. The solution devised for the surface labels (storing colors directly instead of the label numbers) does not work here either, since the actual area numbers have to be read from the texture later on.

- The absorption factor of each voxel is set to one as long as it belongs to any area. If it is not associated with an area, the absorption is set to zero. With this setup, voxels that are closer to the camera completely obscure the ones that are further away, while voxels that belong to no area are simply omitted.

8.4 Surface Point Selection and Display

There are various reasons why a user might want to select a specific point on the brain surface. One scenario, which has been implemented in the context of this thesis (and which is described in the next section), is to measure distances between the center of a previously selected fMRI activation area and a given surface point.

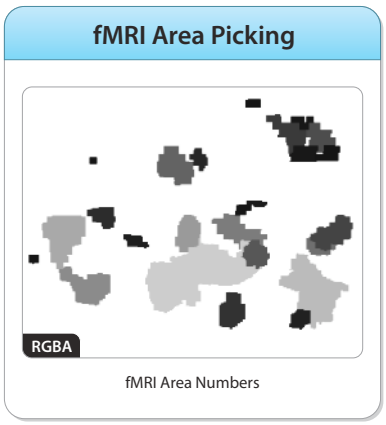


Figure 8.13: Output from the FMRI AREA PICKING pass.

Surface Point Selection Mode

In order to select a point on the surface, the user enters *Surface Point Selection Mode*. While in this mode, moving the mouse over the brain surface constantly updates the selected surface point. A mouse click permanently sets the surface point and ends the current mode. Visual feedback (which will be described below) is given to the user throughout this entire process.

In order to determine the position of the surface point, a ray is created that has its origin at the camera position. The ray direction is first computed in the camera local coordinate system in such a way that the ray passes through the pixel the mouse currently points at. This direction is then transformed into world coordinates. Finally, the kd-tree data structure is used to find the closest intersection of the constructed ray with the brain surface.

Surface Point Display Style

To display the computed point on the surface, an appropriate rendering style has to be devised. A simple approach would be to render a marker such as a a dot or a cross on the surface as seen in Figure 8.14. The relative position of both objects, however, can not be easily determined by looking at a rendered image, especially when the marker lies on the part of the surface that is clipped away (as seen in Figure 8.14 (b)).

To address this problem, the *interaction* between the surface and the marker is strengthened. (At the moment, occlusion is the only form of interaction.) The marker point will act as a point light source with a very quick falloff that can be

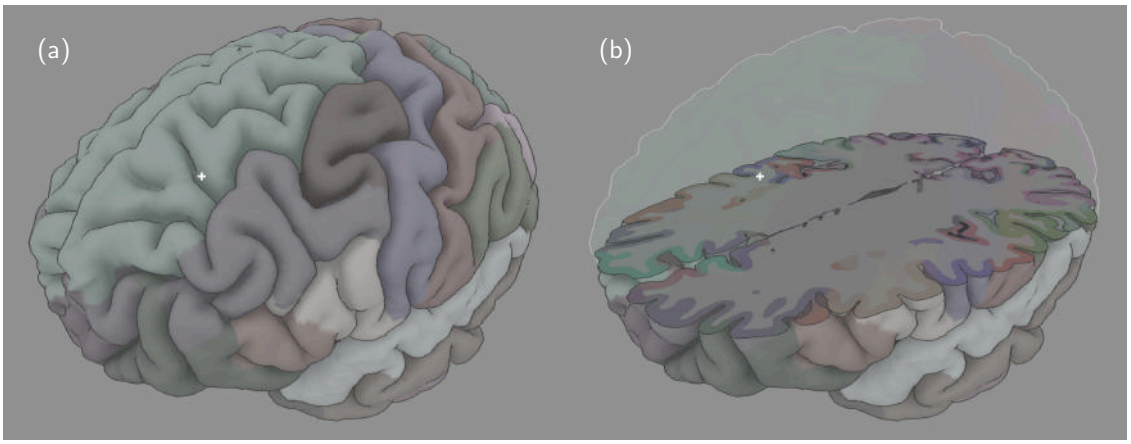


Figure 8.14: Surface point displayed as a marker. The relative position of both objects can not be easily determined.

adjusted by the user. Figure 8.15 shows results for this method. Note that the point light is allowed to "shine trough" the object in order to increase the surface area affected by the light. Strengthening the interaction through the use of light has the following advantages:

- The fact that the surface is illuminated at all is a strong clue that the marker is actually very close to it. Also, the very bright illumination at the center strongly suggests that the point is actually attached to the surface itself.
- Even when the marker lies on the surface that is clipped away, the fact that the marker's surroundings are illuminated and therefore become visible, allows for an easy identification of the surface point's position.
- Marker positions that are only one pixel apart on the screen may be far apart from each other in world coordinates. This is extremely well captured by this rendering style, as can be seen in Figure 8.16.

Integrating the Point Light into the Render Pipeline

The straightforward way of implementing the point light would be in another render pass. This however, would lead to a lot of duplicated work, since the pass would need information that is already available inside the other shaders (e.g., the silhouettes for both the upper and lower surface). So instead, the point light shading is integrated into the existing pipeline as follows:

1. For every visible surface point, the amount by which it is lit has to be stored for later access. This can be any value ℓ in the range $[0, 1]$. The passes in which the brain surface is actually rendered are Mesh Above/Below 1 and

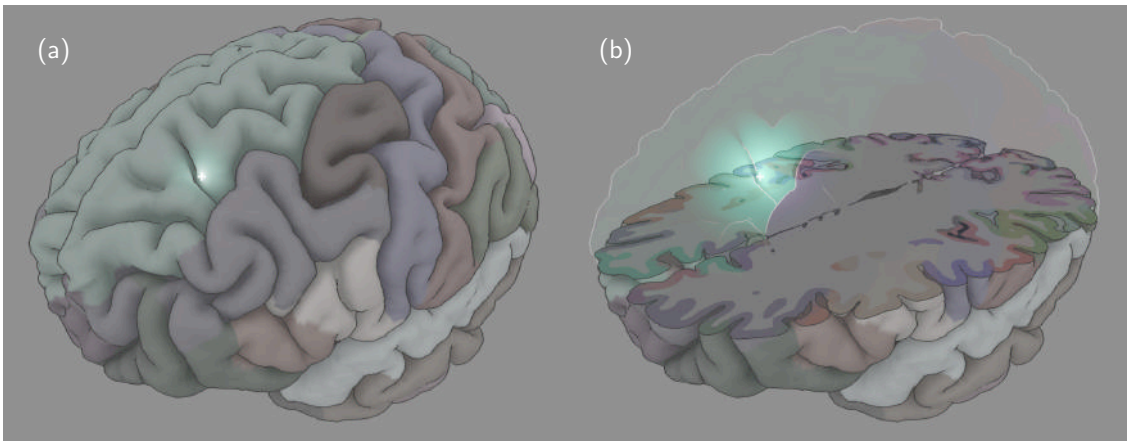


Figure 8.15: The surface point acts as a light source that illuminates the surrounding surface.

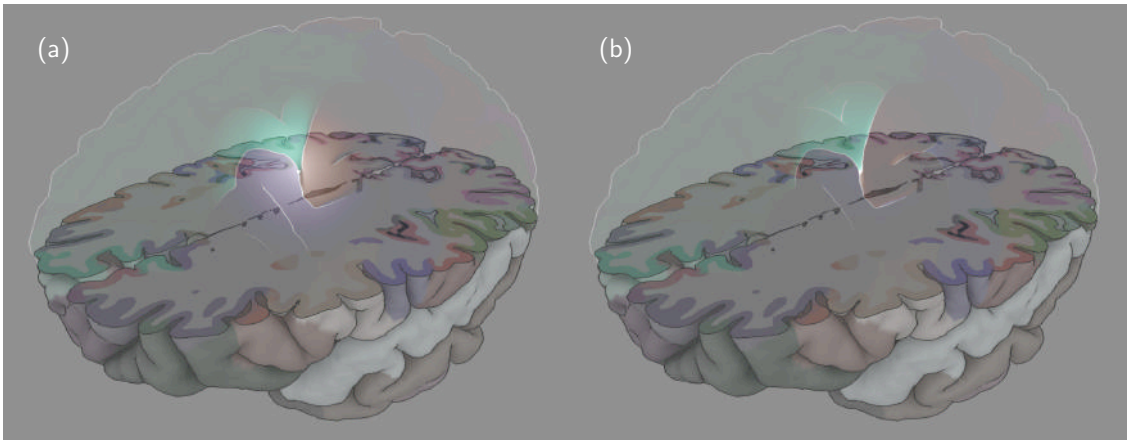


Figure 8.16: The surface point moves 1 pixel to the right from (a) to (b). In (a), it lies on top of the gyrus, in (b) it has "descended" into the sulcus. The shading reflects this very well.

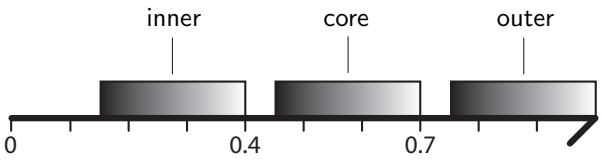


Figure 8.17: The normal length encodes both the category of a pixel (outer, inner, core) and the amount by which it is lit (as depicted with the gradient boxes).

Mesh Above/Below 2, so ℓ should be stored in one of those output textures. Since all channels are already occupied by other values, the information has to be encoded. The length of the normal stored in the RGB component of Mesh Above/Below 2 has been used before to encode the category of a pixel. The length s was set to 1.0, 0.7 and 0.4 for outer, core and inner pixels, respectively. This length is now additionally modulated by ℓ as follows:

$$s' = s - 0.25 \cdot (1 - \ell).$$

The resulting encoding scheme is shown in Figure 8.17.

2. Since the normal length is computed in the Mesh Above/Below 3 pass anyway, this is the best place to decode the previously stored value ℓ as well. After it has been decoded, it is stored in the alpha mask of Mesh Below 3 and for Mesh Above 3, it is stored in the empty blue component (see Figure 8.18).

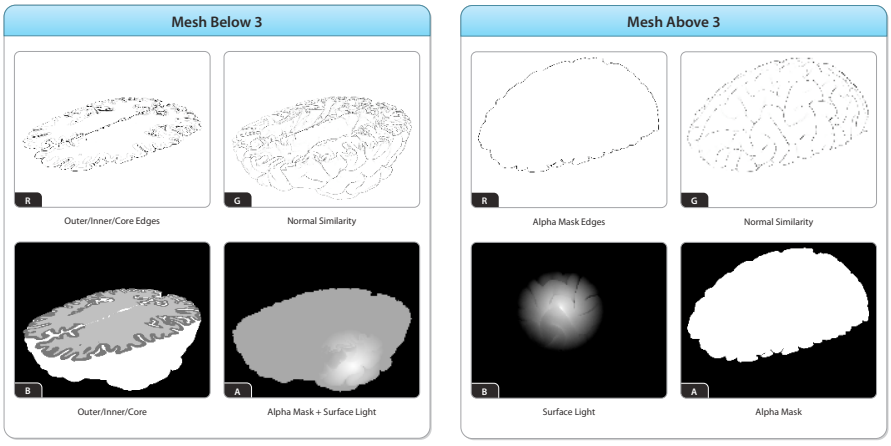


Figure 8.18: Output from the modified Mesh Below 3 and Mesh Above 3 render passes.

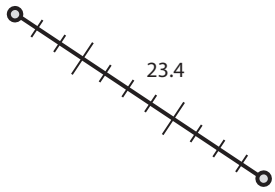


Figure 8.19: A simple ruler with a start and end point, major and minor tick-marks and a text label showing the length of the ruler.

3. Finally, inside the LOWER SURFACE and UPPER SURFACE composition passes, this value is read and used to modulate the surface shading and silhouette drawing. For the LOWER SURFACE pass, all that is done is to brighten the computed emission color $\mathbf{c}$ of the current fragment:

$$\mathbf{c}_{\text{whitened}} \;=\; (1-\ell)\cdot\mathbf{c} \;+\; \ell\cdot\mathbf{c}_{\text{white}},$$

where $\mathbf{c}_{\text{white}}$ is simply the color white. For the UPPER SURFACE pass, in addition to the previous step, the absorption of the surface is increased according to ℓ. Also, the silhouettes that have been previously drawn only very faintly and gray, are now drawn very strong and in white. The final result can be seen in Figures 8.15 and 8.16.

8.5 Ruler

In this work, a ruler is used to measure the distance between the currently selected fMRI area and the surface point chosen by the user. This ruler, however, can be used more generally to measure distances between any two points in the scene.

Parts of the Ruler. The ruler consists of a straight line rendered from the start to the end point. Further, major and minor tick-marks are shown along the ruler as seen in Figure 8.19. The spacing of the major marks, the number of minor tick-marks and their respective sizes can be adjusted by the user. When the ruler's length changes, new tick-marks are created and old ones are deleted as needed. Finally, the ruler also shows a small text label in the center that displays the ruler's current length.

Orthogonal Tick-Marks. In order for the tick-marks to be always visible, regardless of the camera's or the ruler's orientation, they are drawn orthogonal to the

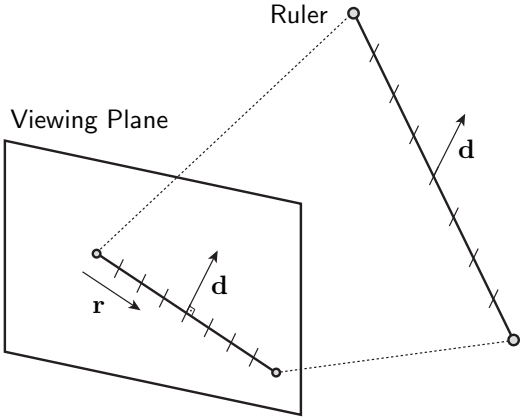


Figure 8.20: The tick-marks of the ruler are oriented such that when projected on the viewing plane, they are orthogonal to the projected ruler direction $\mathbf{r}$.

ruler's direction vector, as seen from the camera. More specifically, both the start and end point are projected onto the camera's viewing plane and the direction vector $\mathbf{r}$ between those projected points is computed. If the camera's view direction is given as $\mathbf{v}$, then the orthogonal direction $\mathbf{d}$ needed for the tick-marks is simply $\mathbf{d} = \mathbf{r} \times \mathbf{v}$. This direction is determined at the start of every frame, after which start and end positions of all tick-marks are updated (see Figure 8.20). If $\mathbf{p}_r$ is the position of a tick-mark on the ruler, then its start and end positions, $\mathbf{p}_s$ and $\mathbf{p}_e$, become:

$$\mathbf{p}_s = \mathbf{p}_r + s \cdot \mathbf{d},$$
$$\mathbf{p}_e = \mathbf{p}_r - s \cdot \mathbf{d},$$

where s is the tick-size specified by the user.

Label Placement. The label that shows the length of the ruler is positioned at the center of the ruler. In order to ensure that the ruler is not obscured by the label or vice versa, the label is offset by a certain amount o in the same orthogonal direction $\mathbf{d}$ that has been computed above. By taking into account the label width w and height h (which depends on the text that is currently displayed) the amount o by which the label is offset, is set to the label's diameter, i.e.:

$$o = \sqrt{w^2 + h^2}.$$

Render Pass: RULER

The ruler, together with its tick-marks and label, is rendered into the scene by means of an additional render pass during the composition pipeline. Figure 8.21 shows an example of the final output. In order to achieve this result, the ruler render pass is inserted between MIDDLE SURFACE and UPPER VOLUME of the pipeline. The ruler fragments are drawn in different strengths depending on their position relative to the clipping plane.

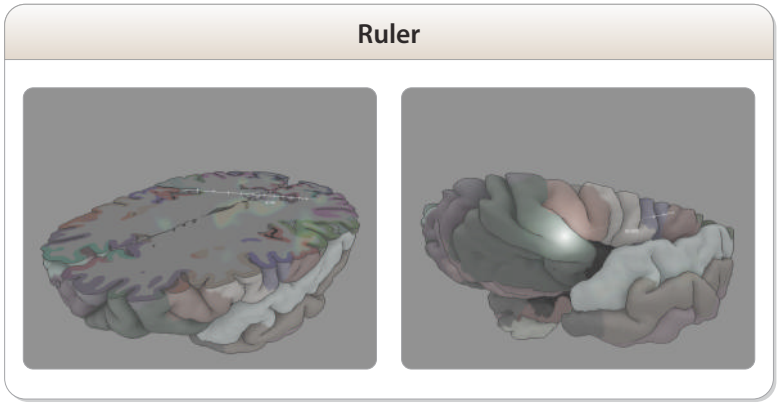


Figure 8.21: Output from the RULER render pass.

9 Text Labels

In the system developed in this thesis, the different areas on the brain surface are rendered with different colors. In order to ease the identification of these areas, the user can turn on text labels in the rendering. An example can be seen in Figure 9.1. Note that only one hemisphere of the brain is rendered in order to allow for inner parts of the surface to be visible as well.

9.1 Label Positions

Each label is composed of three parts: the label text itself, the half-transparent text background, and a label stick (Figure 9.2). Two points have to be specified for each label. The first is the *stick point* **s**, which is where the label is "stuck" into the surface. The second is the *label point* **p**, which is where the actual text label is placed. This section describes how these two points are computed.

Stick Point. Figure 9.3 shows a schematic two-dimensional cut through the brain surface. When computing the stick point by simply choosing the average positions of all vertices belonging to the green label, the stick point would lie close to the depicted point **a**. This is less than ideal, however, since seen from the outside, the point doesn't seem to lie at the center of the brain area at all. A much better choice for the stick point would be the depicted point **b**. To compute this point, the following approach is taken:

1. An outer hull of the brain mesh - the *brain hull* - is computed. This is done by first constructing an alpha shape [Bernardini and Bajaj, 1997], [Edelsbrunner and Mücke, 1992],[1] and then computing the surface normals of that shape as described in Chapter 4. A potenitally better way of constructinc the hull surface would have been to use an active contour model as described in [Xu and Prince, 1998], [Kass et al., 1988]. However, in this thesis, the alpha-shape hull has been used due to the significantly lower implementation effort.

[1] To construct the alpha shape, an algorithm from the CGAL library [Overmars, 1996] is used. Due to the nature of this algorithm, the orientation of the surface faces is not consistent at first, i.e., some are oriented clock-wise, and some are oriented counterclock-wise. So, the first step is to unify the face orientations.

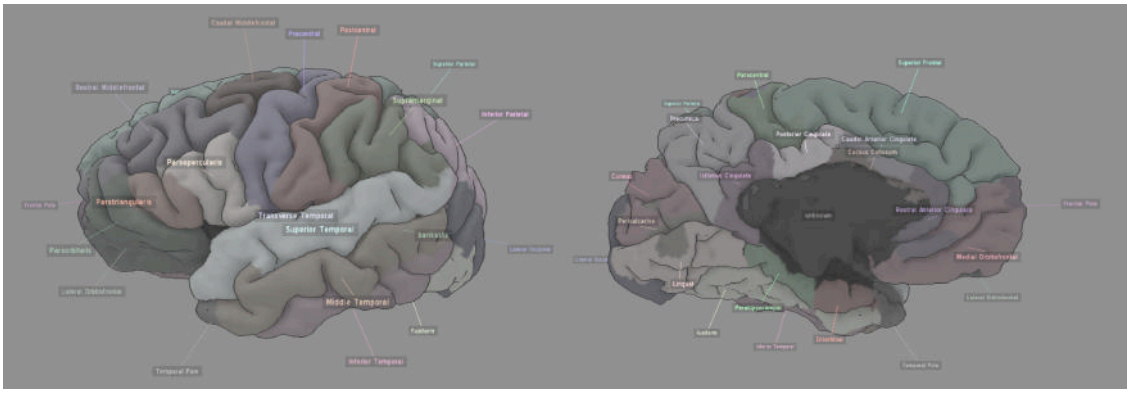


Figure 9.1: Text labels are shown to ease the identification of the colored surface areas.

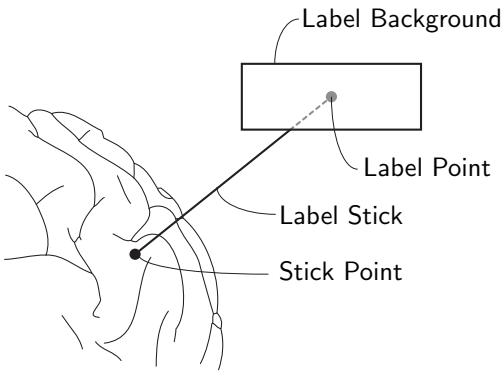


Figure 9.2: Parts of a label.

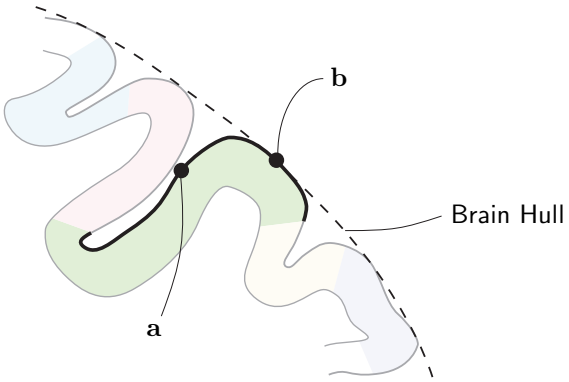


Figure 9.3: Potential stick point positions on a two-dimensional cut through the brain surface. Simply computing the average position of all vertices belonging to the green surface area could result in a point similar to **a**. **b** would be the better choice for the stick point.

2. A kd-tree is constructed for the computed brain hull.

3. For each vertex position $\mathbf{v}_p$, the kd-tree is queried for the closest point $\mathbf{v}_\alpha$ that lies on the hull. Then, all vertices are sorted with respect to their distance to the hull.

4. The 30% closest vertices are selected and the average of their $\mathbf{v}_\alpha$ points is computed. This average point is then again projected on the hull to get the point $\hat{\mathbf{v}}_\alpha$. By only taking the 30% closest vertices, this step prevents the average point to drift to a less optimal point on the hull in cases, where large parts of an area lie on deep sulci (see Figure 9.3).

5. $\hat{\mathbf{v}}_\alpha$ can not be used as the stick point directly, since it might not lie on the surface mesh. So, a ray is constructed and intersected with the kd-tree holding the original brain surface mesh. This ray is initialized as follows:

   a) First, the normal $\hat{\mathbf{n}}_\alpha$ of the brain hull at $\hat{\mathbf{v}}_\alpha$ is computed. This is done by determining the hull triangle this point belongs to and computing the barycentric coordinates of the projected point in the hull triangle. These coordinates are then used to linearly interpolate the normals stored in the vertices of the triangle. Finally, the resulting normal has to be re-normalized.

   b) The ray's origin is set to $\hat{\mathbf{v}}_\alpha + t \cdot \hat{\mathbf{n}}_\alpha$, where t is positive and is chosen so that the resulting point is guaranteed to lie outside of the brain mesh.

   c) The ray's direction is set to $-\hat{\mathbf{n}}_\alpha$.

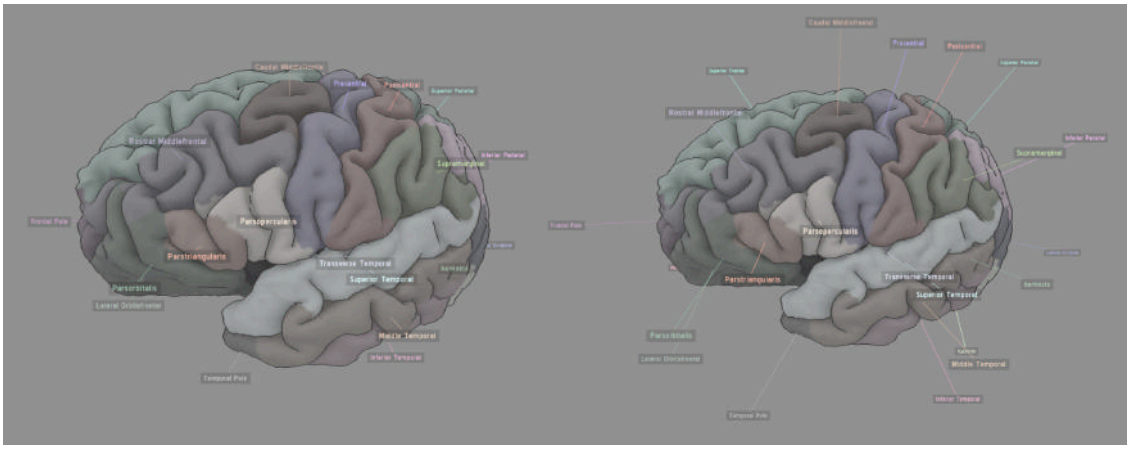


Figure 9.4: Two renderings with different label distances from the surface.

Finally, the stick point is set to the closest intersection point of that ray with the brain surface.

Label Point. The label point is computed by taking the previously computed stick point $\mathbf{s}$ and translating that along the alpha-shape normal $\hat{\mathbf{n}}_\alpha$:

$$\mathbf{p} = \mathbf{s} + d \cdot \hat{\mathbf{n}}_\alpha.$$

Ideally, the optimal value for the label distance d from the surface should also be computed automatically. At the moment, however, it is initially set to a manually chosen value and the user is given the possibility to adjust it as needed. Figure 9.4 shows examples with different label distances.

9.2 Label Rendering

Label Text. The label text is rendered centered at the label point $\mathbf{p}$. In order to center the text, both its height and its width (which depends on the actual text characters) are computed. The text color is chosen to match the color used to render the corresponding brain surface area.

Label Background. The label background is rendered as a half-transparent black colored rectangle that is positioned slightly behind the label text as seen from the camera (thus, its actual position is updated at the beginning of each frame). It's width w and height h are computed by taking the label width and height and adding a small margin.

Label Stick. The label stick uses the same color as the one used for the text. Every frame, the stick's end point has to be adjusted so that it doesn't occlude parts of the label text or label background. This is achieved by first computing the direction vector between **s** and **p** and projecting it onto the camera plane. This direction vector (together with the label background dimensions) is used to determine where the stick enters the background when viewed from the camera and the stick is shortened accordingly.

Render Pass: LABELS

In order to blend the labels into the final output, a new rendering pass is added to the end of the composition pipeline. For this pass, depth checking is activated, so that labels lying behind the brain surface are not drawn.

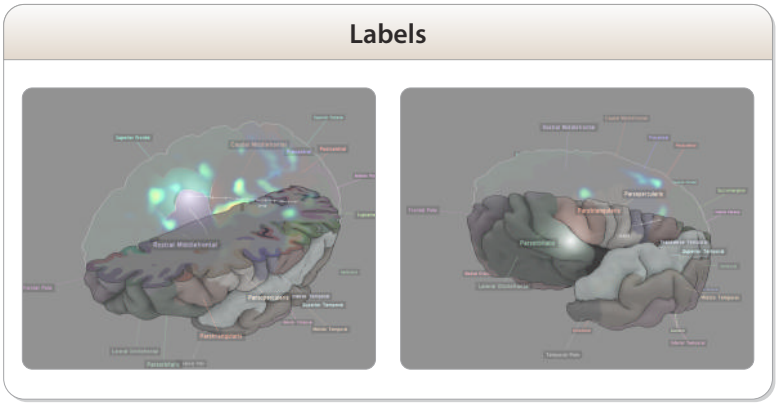


Figure 9.5: Output from the LABELS render pass.

10 The Complete Pipeline

In Chapters 4, 5 and 6, the presented render passes have used textures to store their output. In Chapter 7, they have been put together to render the final image. In the last two chapters, various additions and modifications to the pipeline have been made to incorporate the concepts needed for user interaction and orientation. In this chapter, the final modified pipeline is presented as a whole.

Figure 10.1 shows, how all render passes work together to produce the final result. First, all passes with a blue header are processed. Some passes need input from other passes, which is denoted by the arrows. After the blue passes are finished, the main composition pipeline is executed (brown passes). Starting from the top with the Background pass and finishing at the bottom with the Labels pass, the final output is computed step by step.

The FMRI Area Picking pass is only executed when the user clicks with the mouse to select an area of activation.

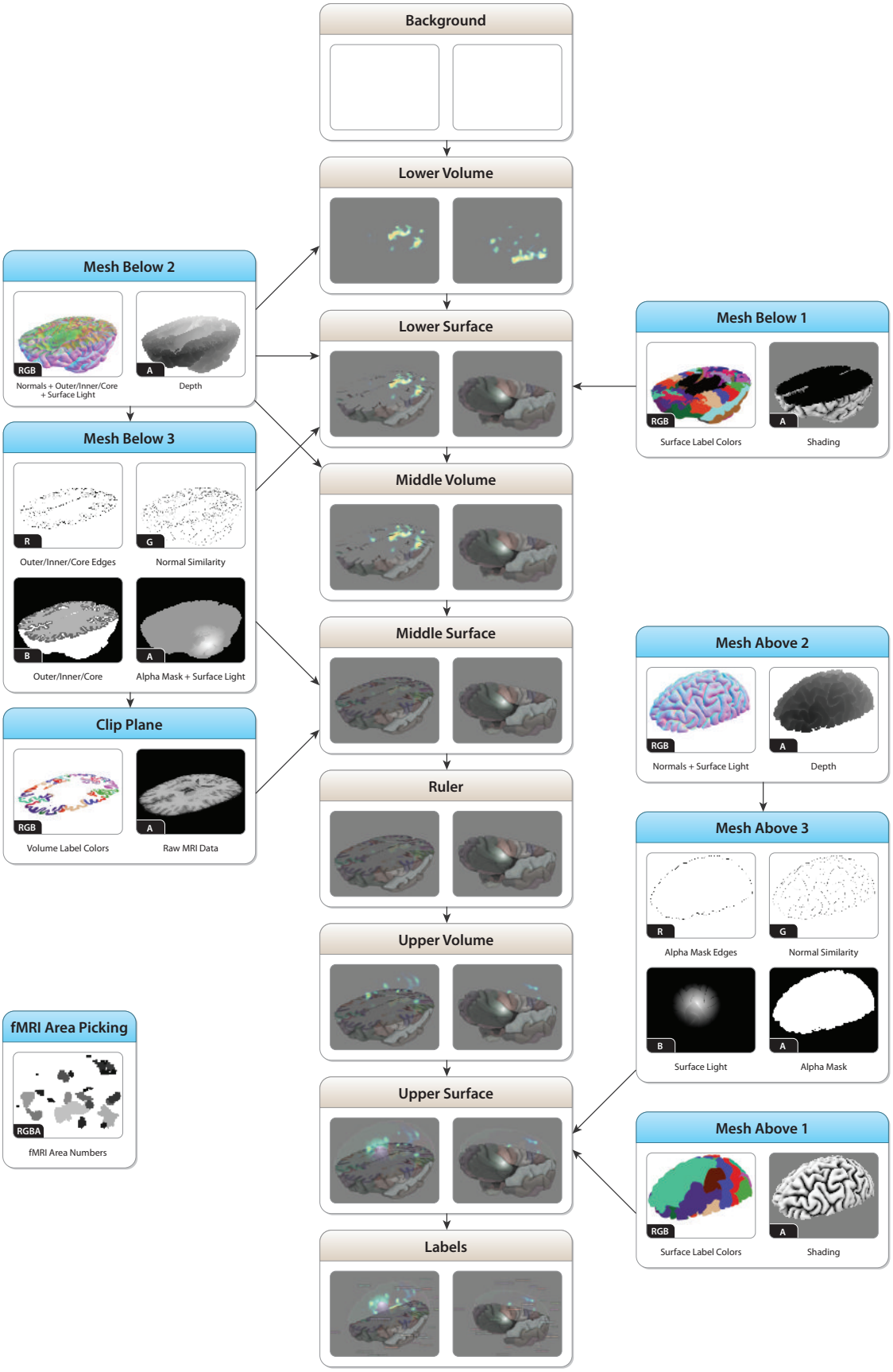


Figure 10.1: The complete render pipeline. The description is given in the text.

Part II

Implementation Details

11 General Program Structure

11.1 iBrain

The software that was developed in the context of this thesis is called *iBrain*. It is implemented in C++ and uses various libraries that are described further below. The prominent features are:

- Cross-platform operability — iBrain has been developed and tested on both Mac OS X and Windows XP. Both Xcode and Visual Studio 2005 Express project files are provided. Compilation of iBrain on Linux platforms has not been tested but should be straightforward.
- Modular design — The software is built on different modules that are kept largely separate to reduce dependencies and allow for their easy replacement.
- Test Scenes have been created for many of the techniques presented in this work. This allowed for an isolated study and fine-tuning of each technique before integrating it into the overall pipeline.

11.2 Used Libraries

The following libraries have been used in the development of iBrain:

- Qt [Qt, Web] — a cross-platform windowing toolkit, which is used for the main user interface elements and the cross-platform render window creation.
- Ogre3D [Junker, 2006] — a cross-platform object-oriented graphics engine. Among other features, it provides a scenegraph implementation and a resource management system. In theory, both Direct3D and OpenGL can be used to render the final output, but all rendering operations developed in this thesis are based on OpenGL shaders. Therefore, output is limited to OpenGL rendering.
- CGAL [Overmars, 1996] — the computational geometry algorithms library (CGAL) provides the half-edge data structure that is used in this work for mesh storage. Among other algorithms, it has the capability to compute alpha-shapes [Bernardini and Bajaj, 1997], [Edelsbrunner and Mücke, 1992], which are used in Chapter 9.

11.3 Program Flow

First, a given MRI and fMRI data set has to be processed by FreeSurfer and SPM, respectively. In a further preprocessing step, the ambient occlusion values are computed by iBrain and stored for each hemisphere.

Every time iBrain is launched, the following steps are performed:

- The surface meshes are loaded together with the surface labels and ambient occlusion values for each vertex.
- A kd-tree is created for the surface meshes.
- The activation volume (fMRI) is loaded and segmented into regions by using the watershed algorithm.
- The structural volume (MRI) is loaded together with the segmentation labels. These labels are then expanded as described in Chapter 6.
- The text labels for both hemispheres are computed as described in Chapter 9.
- All meshes and volumes are uploaded to the graphics card memory.
- The main render loop starts. In each frame, the entire pipeline shown in Chapter 10 is processed to produce the visualization output.

12 The Kd-Tree Structure

This work uses ray/mesh intersections in Chapter 4 in order to compute ambient occlusion values for the brain surface. In Chapter 8, they are used when the user selects a brain surface point. In Chapter 9 closest point queries are performed on the brain hull mesh in order to compute label positions. To efficiently implement these queries, a kd-tree data structure is used [Bentley, 1975, Jansen, 1986].

A kd-tree represents a partitioning of objects in k-dimensional space. It is a specialization of a BSP tree in that only axis aligned planes are used to partition the space, and not arbitrary ones. In this thesis, a three-dimensional kd-tree has been implemented that partitions mesh triangles. This allows for a fast execution of the described mesh intersection and nearest point queries.

12.1 Constructing the Tree

The tree is constructed by first creating a bounding box that encompasses the entire mesh. The first node of the kd-tree is initialized with this bounding box and is set to hold all triangles of the mesh. Then the following algorithm is used to create the tree, starting with the previously created node:

A) If the node holds less than 20 triangles, the algorithm stops.

B) Otherwise, the current node is split up. To this end, the optimal axis and position for the splitting plane have to be determined. This is done by examining each axis individually and evaluating different split plane positions along that axis. For each of those positions, it is determined how many triangles lie "below" the plane and how many lie "above" (with respect to the splitting dimension). With this information, a cost function as presented in [MacDonald and Booth, 1990] is computed for each position. The general idea is to assign a low cost to (and therefore favor) split positions that come close to the following situation:

   1. One of the resulting child nodes is small and contains numerous triangles, whereas
   2. the other child is large and contains a small amount of triangles.

This way, when a ray is later used to intersect the mesh and it is determined that it doesn't hit the bounding box of the first child (which is probable, since it was chosen small), many triangles can be discarded at once. On the other hand, if it hits the second child (which is more probable), only few triangles have to be actually tested.

C) The previously determined axis and split position with the lowest cost are chosen to create two child nodes. All triangles inside the current node are assigned to one of these child nodes, depending on whether they intersect the child's bounding box. Note, however, that the same triangle can be included in both nodes if spans the bounding box of both nodes.

D) The algorithm is executed recursively on both child nodes, unless a maximum tree depth is reached. For the brain surface meshes used in this thesis, a maximal tree depth of 20 resulted in an average of approximately 10 triangles per leaf node and an average tree depth of 18.

The successive splitting of each node can be seen in Figure 12.1. Note that only the leaf nodes actually contain references to the triangles. In Figure 12.2, all bounding boxes of the kd-tree nodes up to a certain depth are shown.

12.2 Intersection Queries

If a point $\mathbf{p}$ and a direction vector $\mathbf{d}$ are given, then the closest point on the surface mesh in that direction can be effectively computed by using the kd-tree. The general algorithm proceeds as follows, starting at the root node of the kd-tree:

A) The initial distance (t) from $\mathbf{p}$ to the mesh is set to infinity.

B) If the ray intersects the bounding box of the current node, the child nodes are recursively tested for intersections with the ray. Otherwise, the current node and all the triangles that it contains are ignored.

C) If a leaf node is encountered, all the triangles it contains are intersected with the ray. Each intersection distance is compared to t, and if it is smaller, t is updated to hold the new value. Also the triangle that resulted in the closest hit is stored.

In order to speed up the intersection algorithm, even if the ray intersects a bounding box of a node as described in B), it is not always necessary to recursively check both child-nodes for intersections. Depending on the orientation of the ray, one child can sometimes be discarded. Another speed up can be gained by using a technique called mailboxing [Amanatides and Woo, 1987], which ensures that any given triangle is actually only tested once for an intersection with the ray. This

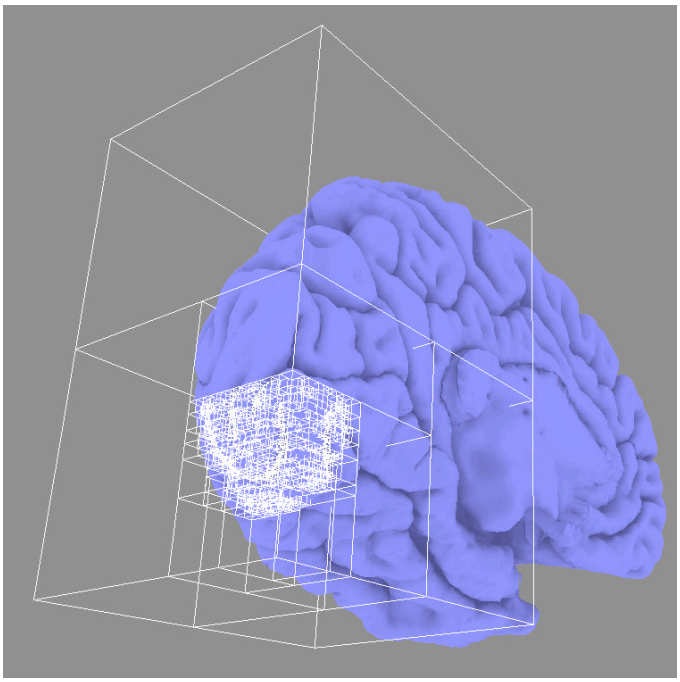


Figure 12.1: The root node of the tree is successively subdivided in order to reduce the number of triangles per node. In this picture, a depth-first traversal has been performed on the kd-tree. The traversal has been halted after the first 2000 child nodes were visited. The bounding boxes of all visited nodes (interior and child) are shown.

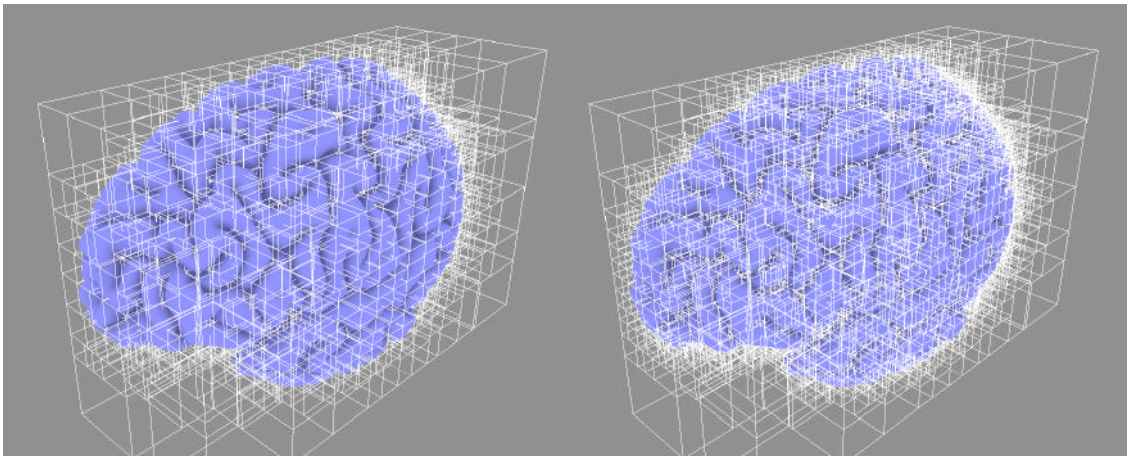


Figure 12.2: Bounding boxes of all kd-tree nodes down to a depth of 10 (left) and 12 (right).

is not implemented in this work, however, since the intersection tests needed in Chapter 8 already performed fast enough to allow for real-time user interaction.

To intersect each individual triangle, the fast ray/triangle intersection algorithm found in [Pharr and Humphreys, 2004] was used.

12.3 Nearest Point Queries

Given an arbitrary point $\mathbf{p}$, the corresponding point on the mesh with the minimal distance to $\mathbf{p}$ can be effectively computed using the previously constructed kd-tree. For this, the kd-tree is traversed, starting with an initial minimum distance of ∞. The algorithm is designed so that the following invariant holds true: "The current node is worth investigating, i.e., parts of the current node lie within the previously found minimum distance to $\mathbf{p}$." So starting with the kd-tree root node (for which the invariant trivially holds true), the following steps are performed:

A) If the current node is an interior node (i.e., not a leaf node), the child node with the bounding box that is closer to $\mathbf{p}$ is determined and made the current node. (Note that the previously stated invariant automatically holds true for this child node as well). The child node that is farther away is stored in a *far stack* (lifo) for eventual later processing.

B) If the current node is a leaf node, the distance of $\mathbf{p}$ to each triangle is computed (see description below). If any triangle is closer than the previously found minimal distance, it is updated accordingly. Next, the topmost node on the far stack is examined. If its distance to $\mathbf{p}$ is smaller than the previously computed minimum distance, it is made the current node. Otherwise, it is discarded and the next node on the stack is checked. If no such node exists, the algorithm terminates.

In order to speed up the algorithm, for every comparison of distances, squared distances are used. (Since $f(x) = x^2$ is a strictly increasing function, this decision does not affect the outcome of these comparisons.)

Nearest Point on a Triangle

For a given point $\mathbf{p}$, to compute the closest point on a given triangle, the following approach is taken. If the triangle has corners $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ and the unit length direction vectors between to corners are written as $\overrightarrow{\mathbf{ab}}$, the plane defined by that triangle has the following normal:

$$\mathbf{n} = \overrightarrow{\mathbf{ab}} \times \overrightarrow{\mathbf{ac}}.$$

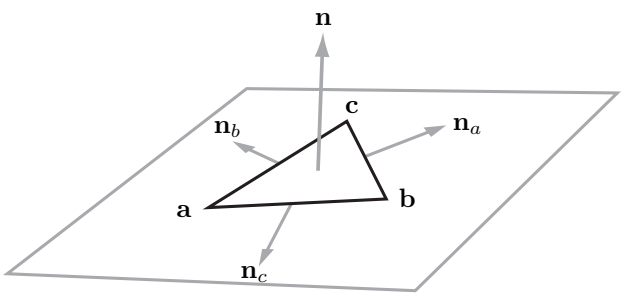


Figure 12.3: A triangle with the normal and the plane it defines. The edge normals $\mathbf{n}_a$, $\mathbf{n}_b$ and $\mathbf{n}_c$ are orthogonal to both $\mathbf{n}$ and the respective edges.

First, for each edge of the triangle, the *edge normal* that is orthogonal to both the edge and the plane normal is computed (see Figure 12.3). If $\mathbf{a}$ and $\mathbf{b}$ are the corners of the edge, the edge normal $\mathbf{n}_c$ becomes:

$$\mathbf{n}_c = \overrightarrow{\mathbf{ab}} \times \mathbf{n}.$$

Each edge normal defines an *edge plane* that is perpendicular to the plane defined by the triangle itself. For each edge, it is determined if the given point $\mathbf{p}$ lies on the positive or on the negative side of the corresponding edge plane. If it always lies on the negative side, the point lies directly above or below the triangle, so it can be simply projected on the triangle plane to find its minimal distance (and therefore, the nearest point on the triangle). If $\mathbf{p}$ lies on the positive side of at least one edge plane, the following steps are performed to compute the minimal distance:

- Each edge for which $\mathbf{p}$ lies on the positive side of the edge plane, $\mathbf{p}$ is projected on the line defined by the edge. So, e.g., for the edge from $\mathbf{a}$ to $\mathbf{b}$, the projected point becomes:
$$\mathbf{p}' = \mathbf{a} \; + \; \left\langle (\mathbf{p} - \mathbf{a}),\; \overrightarrow{\mathbf{ab}} \right\rangle \cdot \overrightarrow{\mathbf{ab}}.$$
- If $\mathbf{p}'$ lies between $\mathbf{a}$ and $\mathbf{b}$, the minimal distance for the current edge is given by $\|\mathbf{p} - \mathbf{p}'\|$, otherwise it given by the minimal distance to the edge endpoints, i.e., either $\|\mathbf{p} - \mathbf{a}\|$ or $\|\mathbf{p} - \mathbf{b}\|$.
- After the minimal distance to each edge has been determined, the minimal distance to the triangle can be easily computed by taking the minimum of those distances.

As mentioned above, squared distances are used in all computations in order to speed up the nearest point search.

Part III

Results & Discussion

13 Results

The main goal of this thesis has been to devise a visualization system for the simultaneous display of MRI and fMRI data. In order to be successfully used by brain mapping researchers, it has to be able to

1. import MRI and fMRI data,

and to convey the following information to the user (see Chapter 3):

2. The shape of the cortex surface,
3. the range and boundaries of different cortex areas,
4. the cortex folding structure and thickness,
5. the shape and position of subcortical structures,
6. and the fMRI activation areas.

Further, the user has to be able to interact with and gather information from the scene. This includes the means to

7. navigate the scene,
8. change the clip plane orientation,
9. select different fMRI activation areas,
10. measure their distances to the surface,
11. and to determine the names of different cortex areas.

All these aspects will now be revisited in the context of a sample data set. A rendering showing most of the features mentioned above is shown in Figure 13.1.

1. The Data. The structural data was gathered through a T1-weighted scan in a 1.5 Tesla MRI scanner with 176 slices and a resolution of 256×256 voxels per slice. The functional data was gathered every 3 seconds through a T2*-weigthed BOLD EPI scan with a resolution of 64×64×64 voxels per time step during an event-related experiment that lasted 500 seconds in total. Both, the MRI and the fMRI data have been processed as described in Chapter 2.

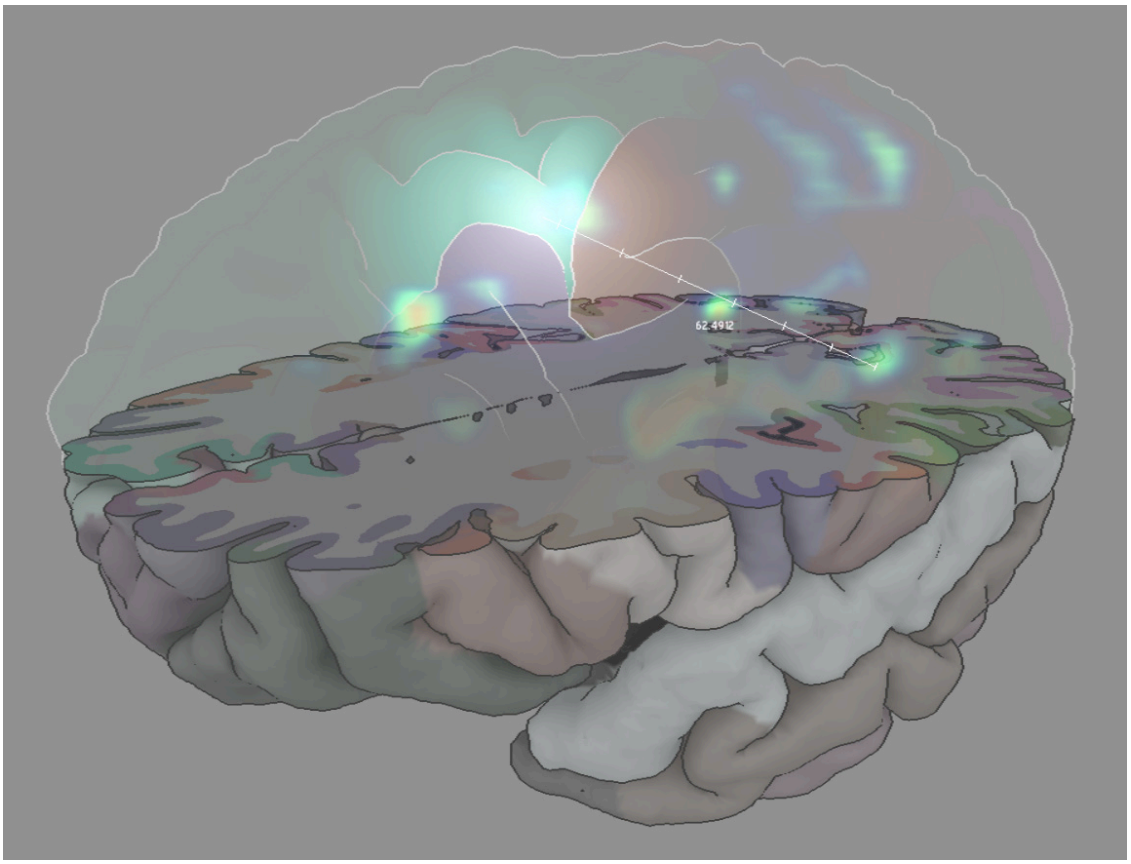


Figure 13.1: A typical output of the rendering pipeline. A surface point is selected and the distance from an fMRI activation area to that point is measured with the shown ruler.

2. Cortex Surface. As noted before, the cortex surface acts as as a frame of reference for the other parts of the visualization and should therefore be visible at all times. In the rendering shown in Figure 13.1, this certainly holds true for the part below the clip plane. For the part above the clipping plane, only the outer hull of the cortex is drawn in a white outline. This reveals the other structures that lie below the surface. When a surface point is selected, however, the cortex surface surrounding that point is shown, since it is necessary to set the point's position in relation to the cortex surface. For both the lower and upper parts of the surface, a subtle ambient occlusion shading is incorporated into the rendering to give a better impression of its three-dimensional shape.

3. Cortex Areas. The different cortex areas are made distinguishable through the use of colors. In the parts below the clipping plane, modest colors have been chosen so that the attention is in order to not draw too much attention to them. However, if a surface point is selected, it becomes a focus of interest. Therefore, brighter and stronger colors are used in its surroundings.

4. Cortex Folding Structure. The cortex folding structure is highly complex. Rendering a semi-transparent cortex surface that would reveal the entire folding structure would lead to a cluttered output. Yet, when locating fMRI activation areas, the folding structure becomes important as a frame of reference. To solve this dilemma, the folding structure is only revealed in its entirety on the clip plane and the user is given the possibility to manually orient it to suit his needs. Figure 13.2 shows such a scenario, where the same activation area is viewed from different positions and with different clip plane orientations.

5. Subcortical Structures. At the moment, no subcortical structures (e.g., tumor sites) are rendered due to the lack of such data. In theory, however, such structures can be easily integrated into the existing pipeline — either as semi-transparent meshes or as additional volume textures. If the resulting image becomes too crowded, these structures can be confined to the clipping plane (just as the cortex folding structure before). An example, or "proof of concept" is shown in Figure 13.3, where the raw MRI data is blended onto the clip plane.

6. fMRI Activation Areas. The cortex coloring, the shading and the folding structure all use modest colors, have low contrasts and mostly darken the final output (i.e., the resulting colors are darker than the uniform gray background). This leaves room for bright and strong colors to be used for the fMRI activation areas. The result is shown in Figure 13.1.

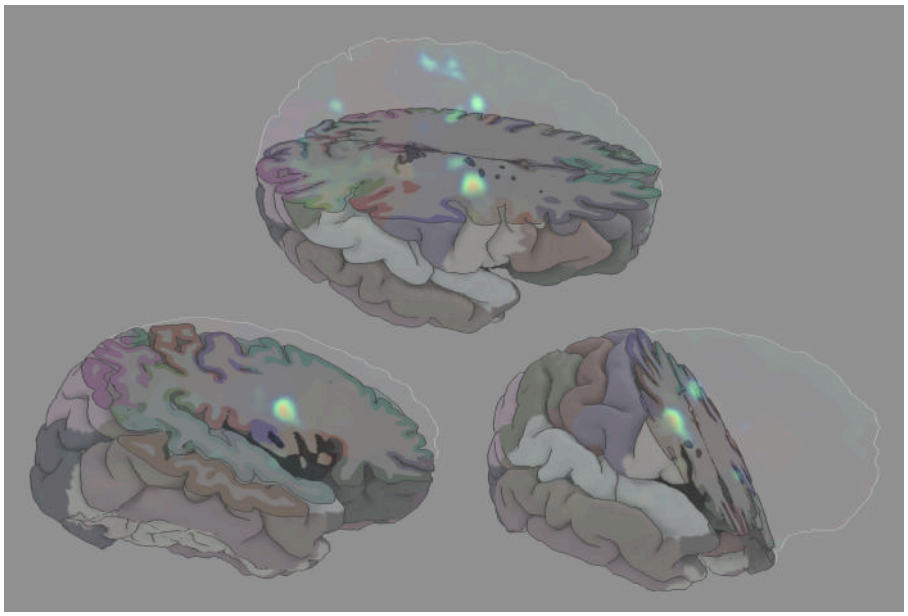


Figure 13.2: The same activation area, viewed from three different camera orientations. In each view, the clip plane is tilted to reveal different parts of the cortex folding structure surrounding the activation area.

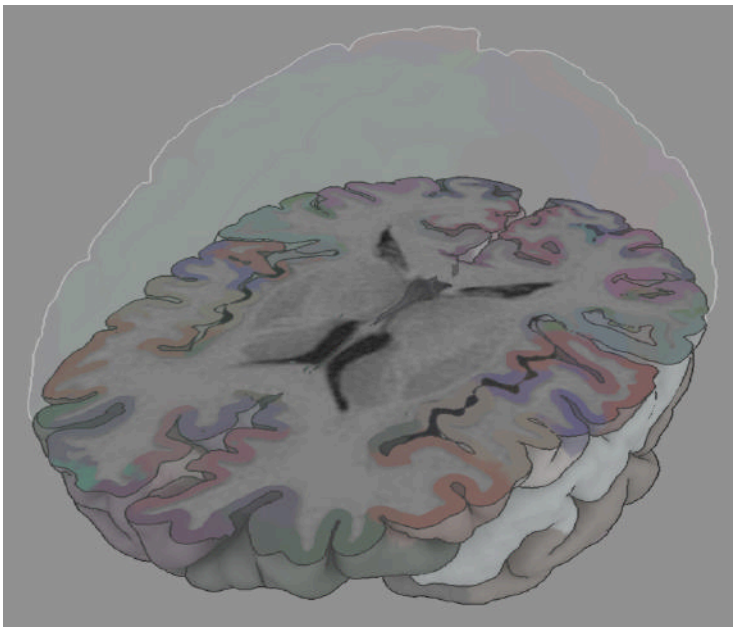
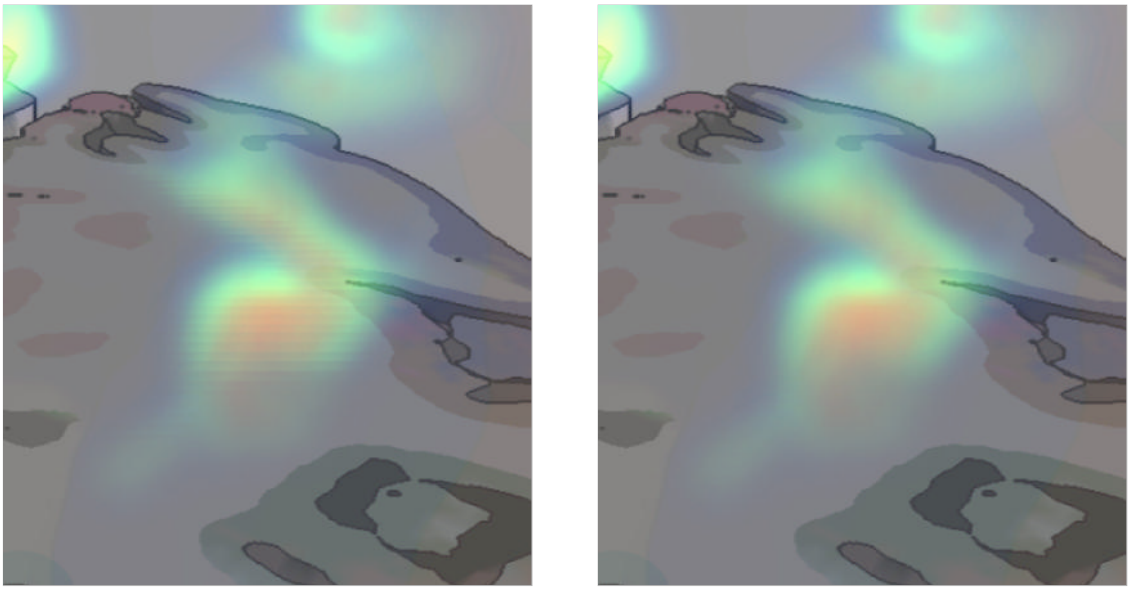


Figure 13.3: Subcortical structures can be blended on top of the clip plane. This image is a "proof of concept" showing the raw MRI data.



(a) (b)

Figure 13.4: (a) When the clipping plane passes right through an activation area, hard clipping of the volume slices results in artifacts. (b) Soft clipping as described in Chapter 5 removes those artifacts.

If the clipping plane cuts directly through an activation area, artifacts like the ones shown in Figure 13.4 (a) become visible. With soft clipping as described in Chapter 5, the artifacts vanish (see Figure 13.4 (b)).

By default, the cortex opacity for the surface lying below the clipping plane is set to a high level (0.85). There are some occasions when a lower opacity setting might be beneficial, e.g., when no clip plane is used at all. For this reason, this value is exposed in the GUI and can be adjusted by the user as needed (Figure 13.5).

7. Scene Navigation. The scene camera can be moved and rotated arbitrarily in the scene. Additionally, the user can continuously change the degree of "perspectiveness" of the camera (see Section 8.1). When investigating activation sites, it can sometimes be helpful to "zoom in" on the activation area at hand. Figure 13.6 (a) shows such a situation. Through the use of a high degree of perspectiveness and a closely positioned camera, the surroundings of the activation area can be studied in detail. Parts that are further away are much smaller in size and are therefore less obtrusive. On the other hand, an orthogonal projection as shown in Figure 13.6 (b) is better suited to compare the measured distances to other parts of the brain and

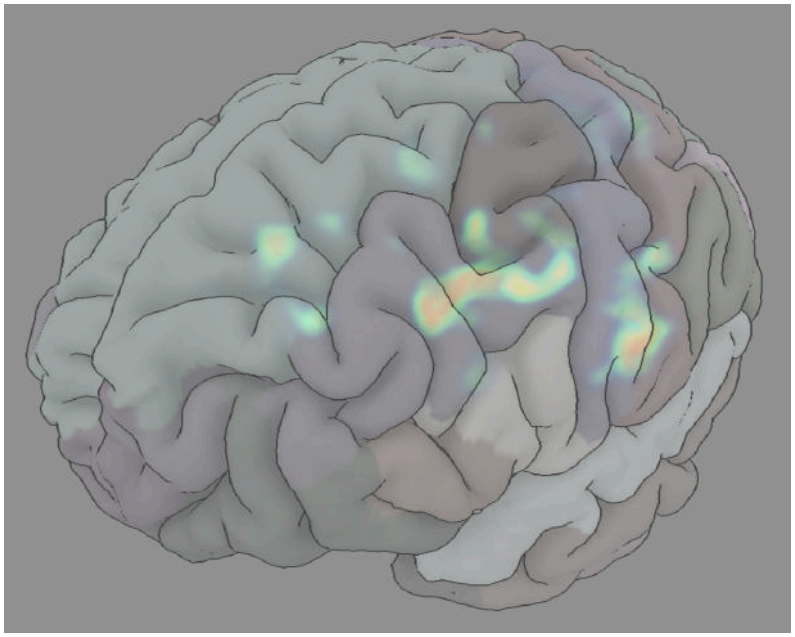


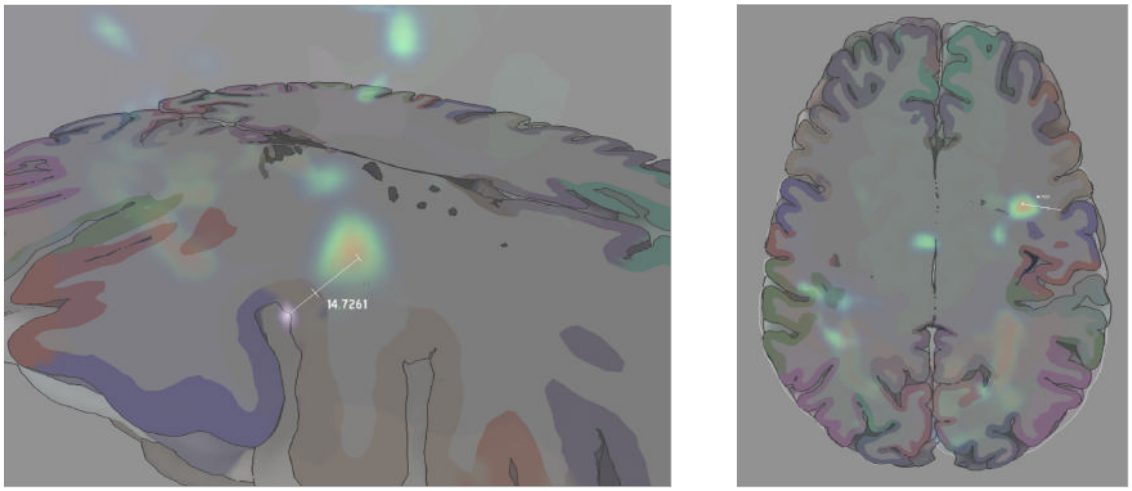
Figure 13.5: When no clip plane is used, setting the transparency of the cortex surface to a higher level reveals the activation areas beneath.

to put them into context.

8. Clip Plane Control. The various uses and benefits of an user adjustable clip plane have already been shown in Figures 13.2 and 13.6. In the current implementation, there is also preliminary support for arbitrarily shaped clipping objects (e.g., spheres) that can be combined with the normal clipping plane to define more complex clipping regions (see Figure 13.7).

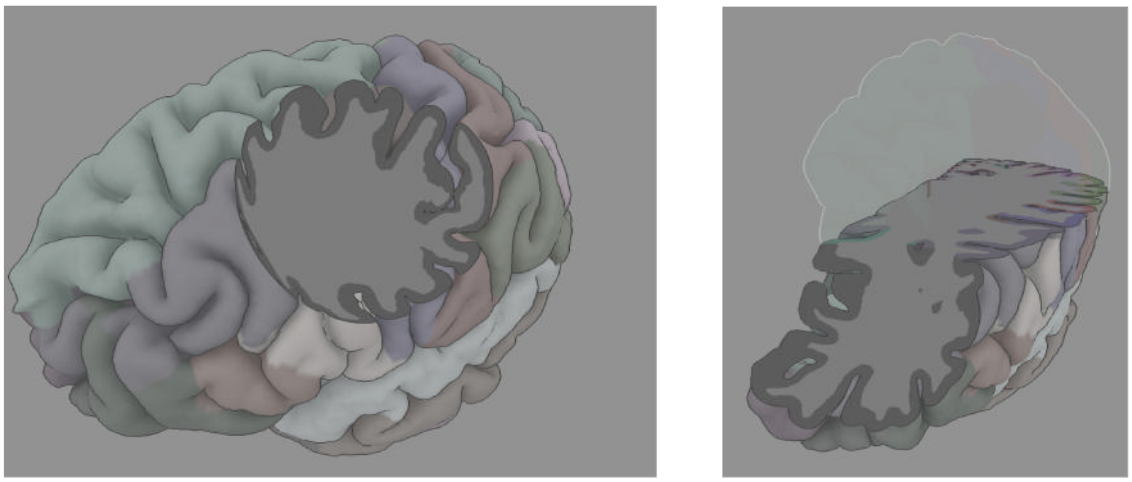
9. fMRI Area Selection. Different activation areas can be selected with the mouse as described in Section 8.3. Optionally, the currently selected area can be highlighted by only using colors for this area, while keeping the others monochrome. Again, this removes clutter from the visual output and serves as a focusing mechanism. Examples can be seen in Figure 13.8.

10. Distance Measurement. Distances from the currently selected fMRI area to the cortex surface can be displayed using a ruler (described in Section 8.5). The end point of the ruler on the surface can be selected by the user with the mouse (as described in Section 8.4). While the user is in the process of moving the mouse cursor over the surface to select a point, he is given immediate feedback of the end result. Figures 8.15 and 8.16 in Section 8.4 show how important the



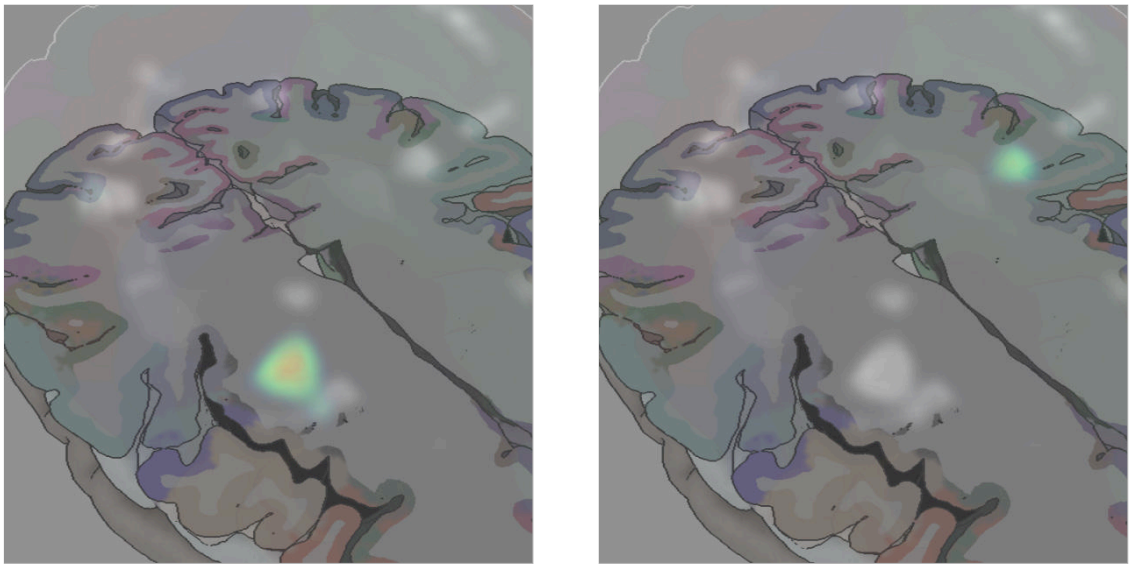
(a) (b)

Figure 13.6: (a) The camera is close to the scene and a high degree of "perspectiveness" is chosen. The user is able to focus on the surroundings of the studied activation area. (b) An orthogonal projection can be useful when the distance of the activation area to the cortex has to be set in relation to the entire brain dimensions. In this image, the camera viewing direction is also set to be orthogonal to the clip plane. This way, line segments of equal length on the clip plane have the same length on the projected image, regardless of their orientation.



(a) (b)

Figure 13.7: Preliminary support for arbitrarily shaped clipping objects. (a) A sphere is used as the clipping object. The user can place the sphere at an arbitrary surface position using the mouse. (b) Both the user defined plane and the sphere are used as clipping objects.



(a) (b)

Figure 13.8: The user can select the activation area he wants to focus on by clicking on it with the mouse. Then, only the currently selected activation area is drawn in colors. The other areas are drawn in grayscale.

surface shading used in this work is for accurately conveying the spatial relation between the selected point and the cortex surface. Example uses of the distance measurement have already been shown in Figures 13.1 and 13.6.

11. Labels. When identification of a given cortex area becomes necessary the user can activate the rendering of surface labels for each hemisphere. An example is shown in Figure 13.9. If he does so, only one hemisphere is rendered in order to show labels for the inner side of the hemisphere (Figure 13.10). (Selectively turning on and off the rendering of each hemisphere can be controlled by the user even if no labels are rendered.)

Finally, Figure 13.11 is an image that combines many of the techniques implemented in this work to show labels, a selected fMRI area, a selected surface point and the corresponding ruler.

13.1 Performance Measurements

To measure the performance of the devised rendering system, two test scenes are used. Scene A is the one seen in Figure 13.1, whereas Scene B is the one seen in

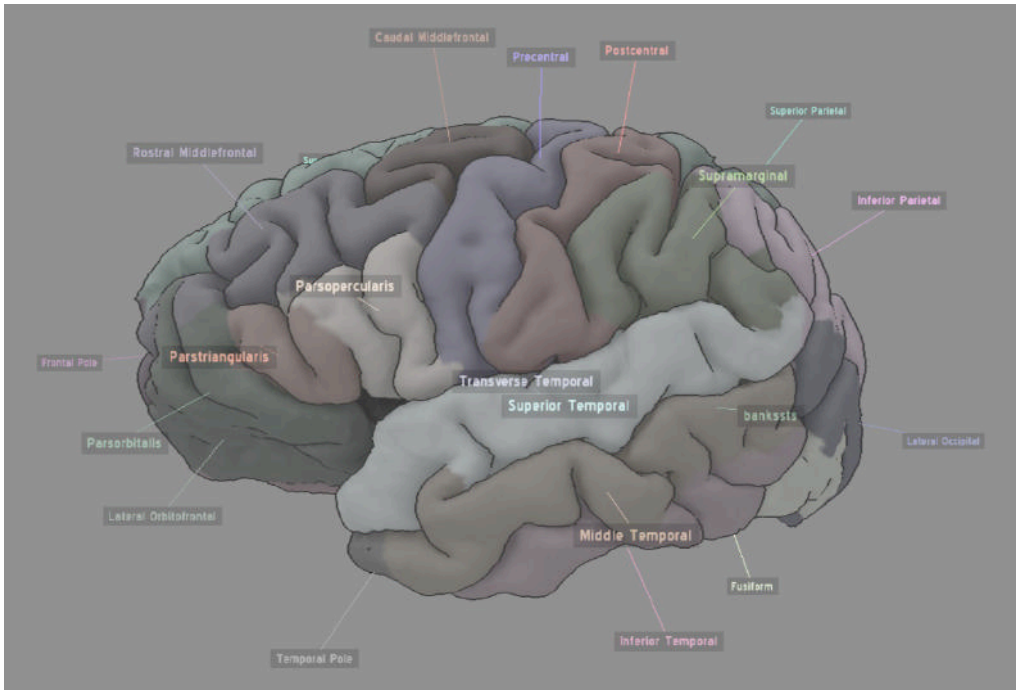


Figure 13.9: Labels for the outer parts of the left hemisphere.

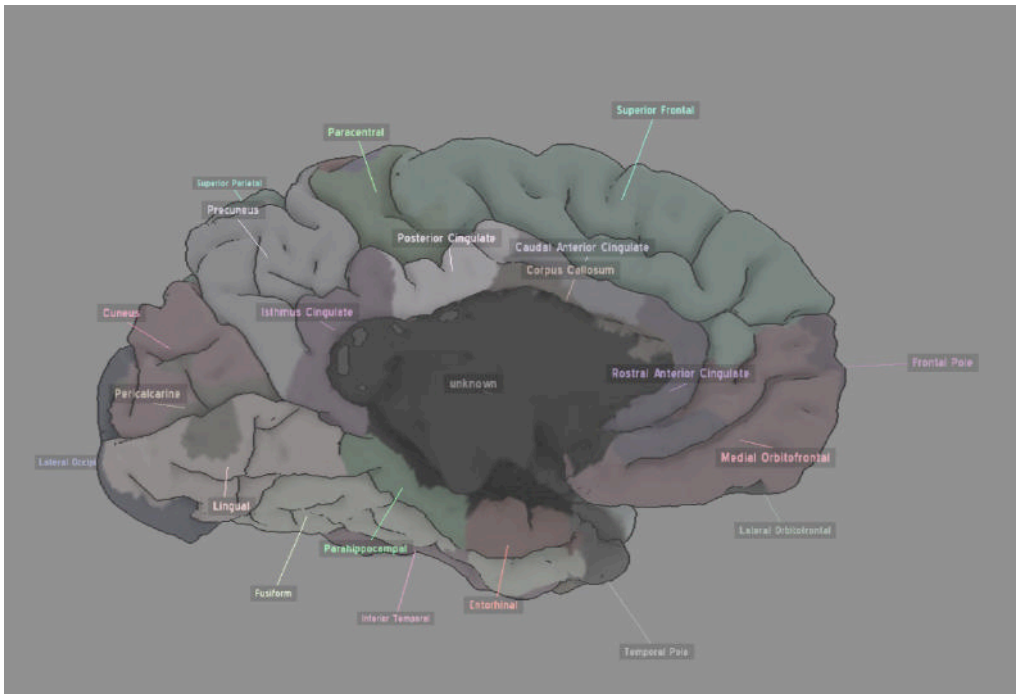


Figure 13.10: Labels for the inner parts of the left hemisphere.

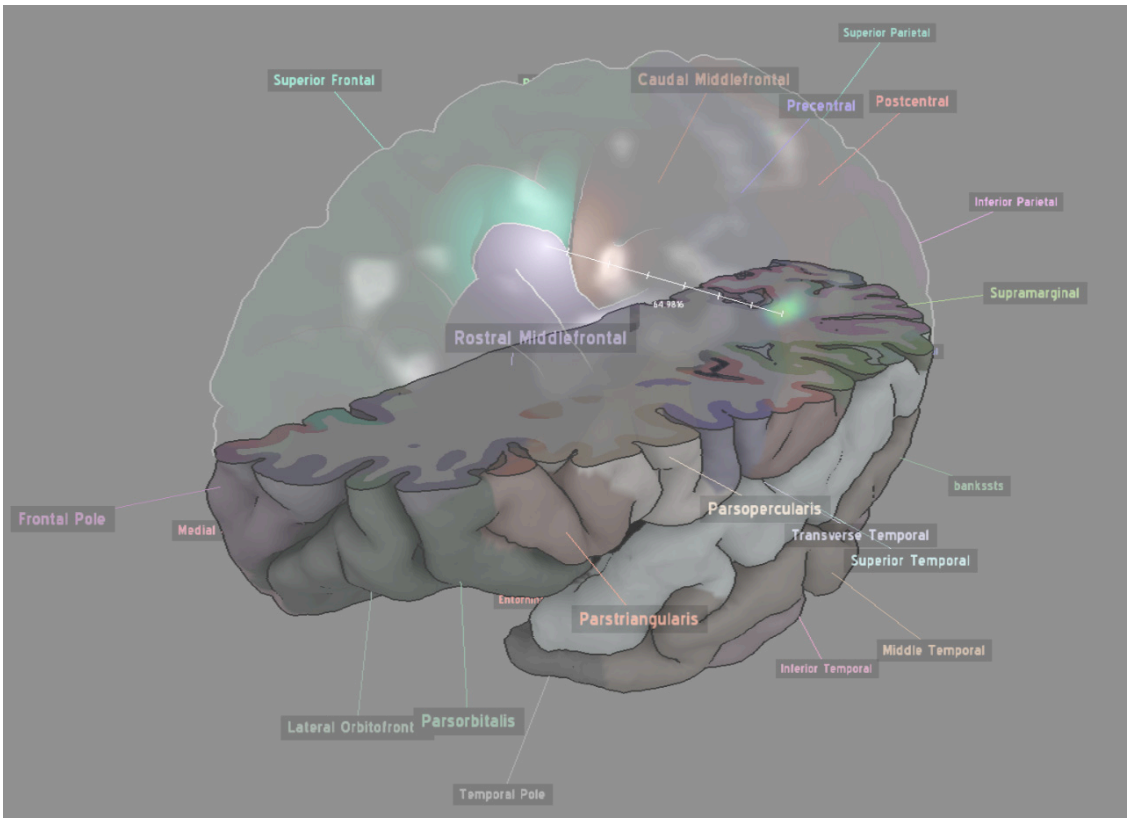


Figure 13.11: An image that combines many of the implemented techniques in this work.

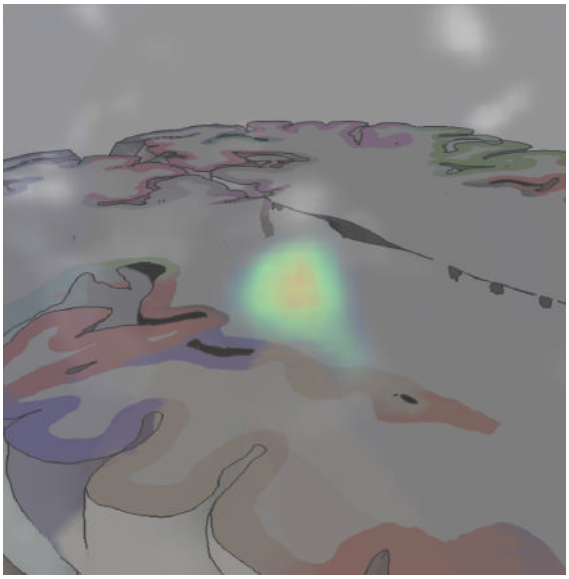


Figure 13.12: A close up view of an activation area. In this image, the entire fMRI volume is rendered with only 90 slices. On the one hand, this degrades the quality of the output to some extent; on the other hand, it improves the rendering speed.

Figure 13.11.

Scene A is composed of two cortex hemispheres, each consisting of two meshes: one that represents the cortex surface and one represents the white matter/gray matter boundary. Both meshes consist of approximately 300.000 triangles. Therefore, one render pass draws approximately 1.2 million triangles. There is a total of 4 render passes in the pipeline that draw these meshes, so in each frame, 4.8 million triangles are drawn for the cortex. In Scene B, only one hemisphere is rendered, so only 2.4 million triangles are drawn each frame.

To render the fMRI activation volume, 180 slices are used in each volume pass to produce high quality outputs. There is a total of three volume render passes, so 540 slices are drawn each frame. When using only 90 slices in each pass, the render quality is still acceptable (see Figure 13.12), so frame rates for this setting are also measured. Finally, when examining cortex labels, it can be beneficial to completely turn off the volume rendering to reduce the scene complexity. Since this is a common usage scenario, frame rates are measured for this setting as well.

All measurements were performed on a Windows XP test machine with an AMD Athlon XP 2600+ processor, 1.5 GB RAM and an ATI Radeon X1950 Pro (8x AGP) graphics card. Results were gathered for two output resolutions: 1024^2 pixels and

512^2 pixels. (This actually denotes the size of the the internal textures used to store the outputs of the render passes and not the output resolution on the screen, which was always set to 1280×1024 pixels.) The measurement results can be seen in Table 13.1.

Table 13.1: Benchmarks for different texture resolutions and scenes.

Resolution	No. Volume Slices	Scene A (fps)	Scene B (fps)
1024×1024	180	1.5	1.6
	90	2.6	2.9
	–	11.3	18.9
512×512	180	4.4	5.2
	90	6.6	8.5
	–	13.2	24.4

Two interesting observations can be made:

1. When using 180 volume slices, the frame rate between Scene A and Scene B differs only slightly, even though there are 2.4 million less triangles in Scene B. This suggests that the volume rendering passes are the main bottleneck of the render pipeline.

2. Indeed, when looking at the performance of the render pipeline when no volume rendering is performed at all, the reduction in the amount of rendered triangles has a significant impact on the render speed.

Seeing that the volume rendering is the main bottleneck of the pipeline, a quick way to improve the rendering speed is to combine the render passes Lower Volume and Middle Volume into one volume render pass. This is possible in this specific visualization, since no interior walls of the brain surface are rendered (see Chapter 7). With this optimization, the measured frame rates become the ones presented in Table 13.2.

Further Performance Considerations

- The preprocessing of the MRI data using FreeSurfer takes approximately 60 hours on a PowerBook G4, 1.5 Ghz PowerPC processor with 1.5 GB RAM. The preprocessing of the fMRI data using SPM takes less than one hour on the same machine.

Table 13.2: Benchmarks for the optimized render pipeline that contains only 2 volume rendering passes.

Resolution	No. Volume Slices	Scene A (fps)	Scene B (fps)
1024×1024	180	2.1	2.3
	90	3.5	4.0
512×512	180	5.7	7.0
	90	7.9	10.9

- The watershed algorithm that is used to perform the fMRI activation volume segmentation (see Section 8.3) takes approximately 5 seconds on the Windows XP test machine.
- The kd-tree construction for both brain hemispheres (see Chapter 12) takes approximately 22 seconds. Once created, it contains approximately 600.000 triangles in 385.000 nodes with an average tree depth of 18.
- The ambient occlusion computation (see Chapter 4) takes approximately 60 minutes for each hemisphere. In this time, 1000 samples are created for each of the 150,000 vertices in the mesh, so a total of 150 million kd-tree interesection queries are performed. This leads to an average of approximately 41,000 intersection queries per second.
- While the user is in the process of selecting a point on the brain surface (e.g., for distance computations), an intersection query is executed on the kd-tree every frame. Assuming a high frame rate of 40 frames per second and taking the above average of 41,000 possible kd-tree intersection queries per second, this means that from the time it takes to render one frame, about about 1/1000 is spent on the intersection query. Therefore, it is safe to say that this form of user interaction has practically no impact on the rendering speed.

14 Discussion

The simultaneous display of brain structure and activation areas potentially leads to complex and cluttered outputs. This problem has been addressed in this thesis by a three-fold approach:

1. Illustrative rendering styles are used for various objects in the scene. This makes it possible to highlight only those aspects of an object that are most important for conveying the needed information. The use of silhouettes for the cortex surface as the main mechanism to convey the surface structure is one example for this.

2. In the absence of activation data, the rendered output has a low contrast and features mostly gray to dark colors. This allows the actual activation areas to stand out through the use of bright colors when they are actually shown.

3. Not all the information is visible all the time. Instead, the user is given various tools that allow him/her to focus on and reveal additional parts of the scene. This includes the adjustable clipping plane, the selection of a specific fMRI area and the selection of surface points.

While this approach is theoretically sound and has been successfully tested on two sample data sets, further "real world" tests are needed to ensure a seamless workflow for the user.

Also, there are a few improvements that were not implemented due to time constraints. For one, the interaction with the scene becomes more seamless as higher frame rates are provided. The measurements in the previous chapter have shown that the use of a texture resolution of 512^2 with a low number of volume slices (90) leads to approximately 8 frames per second. While interaction with the program under this frame rate is feasible, the desirable high quality setting produces only 2 frames per second. One way to improve upon this is to merge some of the render passes of the visualization pipeline. The current pipeline evolved from a fairly general approach of combining NPR techniques with volume rendering, and while some parts have been later tuned for the specific visualization presented in this work, this is certainly not true for all of them. Further, reducing the number of triangles used to render the brain surfaces should only have a modest impact on the rendering quality. Finally, it would be interesting to see, how the devised rendering style can be implemented using raycasting instead of slice-based volume

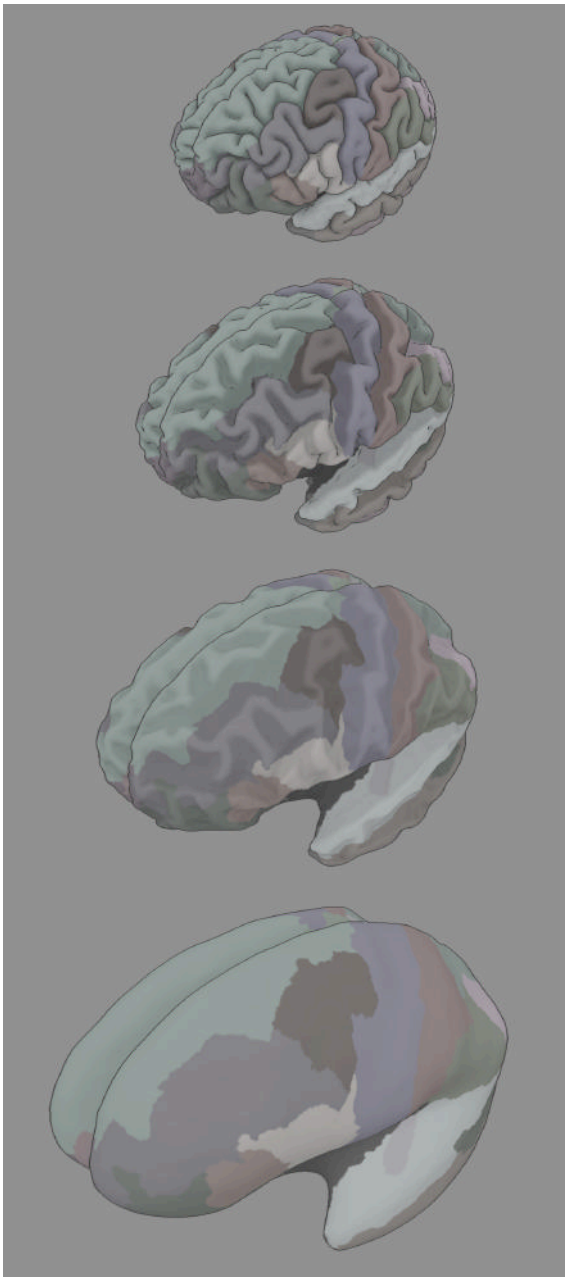


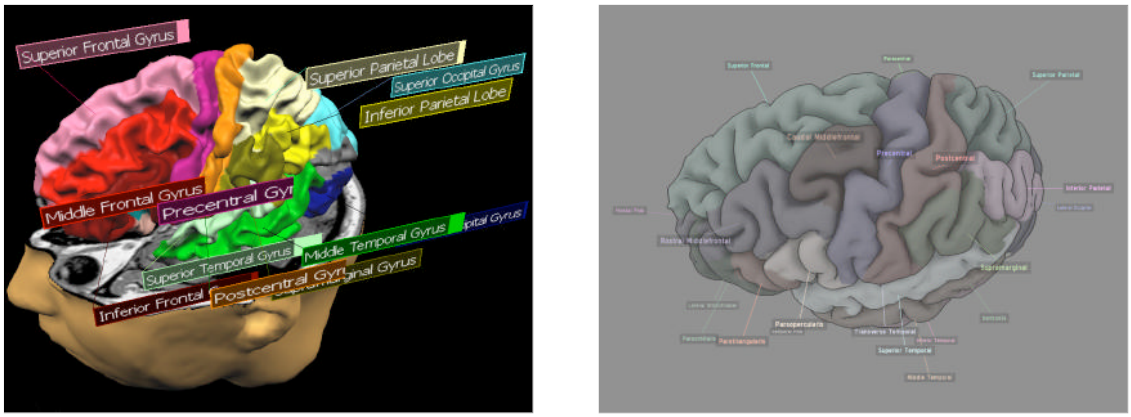
Figure 14.1: The brain surface is gradually inflated. This is an interactive process that can be controlled by the user with real time feedback. (The data for the inflated cortex is exported from FreeSurfer.)

rendering and how this would affect the achieved frame rate.

The visualization style itself could be improved through the display of additional subcortical structures as suggested before. Also, if the clipping plane cuts through an activation area, the part of the activation area that lies directly on the clip plane could be drawn in a stronger way. This would lead to a stronger interaction of these two objects and therefore, to a better understanding of their relation to one another. Finally, depth perception in the scene could be enhanced by introducing "fog", i.e., parts of the scene that are further away could be drawn with less intense colors that gradually fade to the background color. Another way to enhance depth perception would be to support the use of special hardware, such as stereo goggles.

An application area that is outside the scope of this thesis is the display of inflated brain surfaces, onto which the various activation areas have been projected. Although some information is lost in this process (e.g., the exact location and shape of the activation areas), the resulting scene is less complex and easier to comprehend. It would be ideal, if a seamless and gradual transition between both display styles could be interactively controlled by the user. Preliminary support for this scenario has already been implemented in the current work. Figure 14.1 shows the process of gradually inflating a cortex surface. Since the inflation is implemented in the vertex shader, the entire process happens in real time. The next step is to provide a similar mechanism for the activation areas, although this might necessitate more complex algorithms.

Finally, the devised rendering style might be suitable for other application contexts as well. One example would be the use in educational software (see Figure 14.2).



(a) (b)

Figure 14.2: (a) Screenshot of a program called BrainTutor [BrainTutor, Web] from the same company that offers the BrainVoyager software package. (b) The visualization technique devised in this work, rendered from a similar camera perspective. For this application area, stronger colors would probably be more appropriate.

Part IV

Appendix

A Brain Data Acquisition

The technology behind MRI and fMRI scans is very complex. Here, only the basic concepts will be presented. For more details, see [Pinus and Mohamed, 2006], [Savoy, 2001] and [de Haan, 2000].

A.1 Structural Data

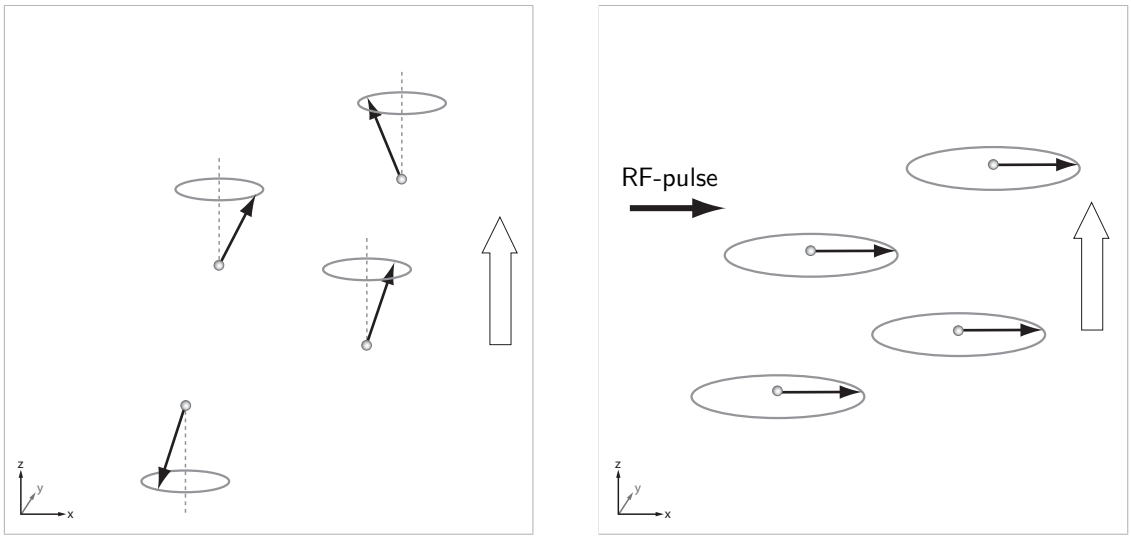
Hydrogen nuclei (H^+) inside molecules are electrically charged and spin around their axis. This produces a small magnetic field that can be represented by a vector, encoding both its strength and direction. This vector is called the magnetic dipole moment, or MDM.

If the hydrogen nuclei are put into an outer magnetic field, a fraction of the MDM's aligns in parallel with the magnetic flied, while a smaller fraction aligns anti-parallel. The net effect is a positive magnetic field produced by the MDM's in the same direction as the outer magnetic field. In general, the stronger the outer magnetic field is, the bigger is the fraction of MDM's that align themselves parallel or anti-parallel to that field. Also, the MDM's of the hydrogen nuclei start to precess, i.e., they engage in a circular motion around the axis of the outer magnetic field (see Figure A.1 (a)). The frequency of this precession depends on both the type of nucleus and on the strength of the outer magnetic field — the higher the strength, the higher the frequency.

If a radio frequency pulse (RF pulse) is applied in a 90° angle, that exactly matches the precession frequency, the MDM's are "tipped" into the x/y-plane (this assumes that direction of the outer magnet field represents the z-direction). Also, the MDM's are now precessing in phase (see Figure A.1 (b)). The net effect of the individual MDM's is now a magnetic field that has no z-component and that circulates in the x/y-plane.

After the RF pulse stops, a process called *relaxation* begins. It has two components:

1. As the MDM's realign themselves with the outer magnetic field, the amplitude in z-direction of the combined magnetic field of the MDM's grows. This is known as T1-relaxation.



(a) (b)

Figure A.1: (a) When a strong magnetic field is applied (thick arrow), a fraction of the the magnetic diploe moments of the hydrogen atoms align themselves with the magnetic field and start to precess around the field axis. (b) A radio frequency pulse is applied wich tips the MDM's into the x/y-plane. Also, they are now precessing in phase.

2. The x/y-amplitude of the magnetic field decreases due to dephasing of the MDM's. This dephasing is caused by a) spin-spin interactions with surrounding nuclei and b) local magnetic field inhomogenities. The latter is a direct result of the fact that the frequency of precession depends on the magnetic field strength. The decay of the x/y-amplitude is called T2*-relaxation.

Both T1- and T2*- relaxation rates depend on the surrounding material, so different brain tissues have different relaxation rates. Depending on whether T1- or T2*-relaxation is of interest, the measurement time after the application of the RF-pulse is chosen in such a way, that the highest contrast between different brain tissues is achieved. These measurements are then called T1- or T2*-weighted.

In T2*-relaxation, the dephasing induced by magnetic field inhomogenities is usually considered an artifact. To reduce this effect, additional RF-pulses are applied to the signal after the original 90° RF-pulse. Each successive pulse is rotated by 180°, where each pulse has the effect of reversing the recess orientation of the MDM's. By measuring the signal in between the applied RF-pulses (where the MDM's are now in phase in spite of the magnetic field inhomogenities), the decay of the signal in the x/y plane that is solely due to the spin-spin interactions of the MDM's can be measured. This is called T2-relaxation, and the succession of the

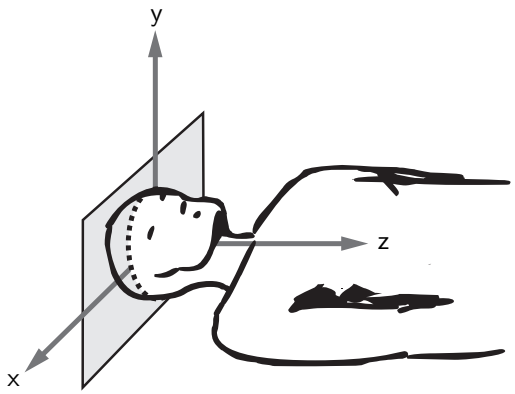


Figure A.2: The subject is placed into the scanner and slices are collected through the use of three gradients: the slice select gradient along the z-axis, the phase encoding gradient along the y-axis and the readout gradient along the x-axis.

RF-pulses is called a *spin-echo pulse cycle*.

Three-dimensional Images

Three-dimensional images of brain data are retrieved by placing the subject in a scanner (Figure A.2) with a strong magnetic field in the z-direction. Then, a *slice-select gradient* is applied on the magnetic field in z-direction, which results in different precessing frequencies of the MDM's along that axis. A RF-pulse then selects a specific slice by narrowing the range of frequencies it contains. Now, all MDM's are precessing in phase. A second gradient along the y-axis (the *phase-encoding gradient*) is turned on for a brief period and then switched off. The result is that all MDM's have the same frequency on the current slice, but have a phase that linearly increases along the y-axis. Finally, during readout, the *readout gradient* is switched on along the x-axis. This way, all MDM's have a different phase and frequency. The recorded signal is then Fourier-transformed to retrieve the intensities of each point on the slice. This process is then repeated for every row and for every slice. Examples of these slices can be seen in Figure 1.3. Note that the resolution along each axis depends on the used gradients. Therefore, is is possible (and common) to have different resolutions along different axes. Such volumes are called *non-isotropic*.

A.2 Activation Data

When performing fMRI scans that measure the BOLD effect (see Chapter 2), small magnetic field inhomogenities that change over time have to be detected. The scanning process has to be both fast (to achieve a high temporal resolution) and it has to detect and preserve field inhomogenities. Therefore, the previously described T2*-relaxation is used. To gain a high temporal resolution, a variant of the spin-echo pulse cycle is used, where, after each RF-pulse of 180°, the phase-encoding gradient is applied in addition to the readout gradient. This way, more than one row of a slice can be determined during one pulse cycle. In fact, as many rows as there are 180°RF-pulses can be determined in succession.

The technique described above is referred to as T2* echo-planar imaging (T2* EPI). A scan of the entire brain volume can be seen in Figure 2.4.

B The Equation of Transfer

This chapter gives an introduction to the general *equation of transfer*, which is the base for many photo-realistic rendering techniques devised in computer graphics. After a derivation that is closely based on the derivation found in [Arvo, 1993], the assumptions and resulting simplifications that are needed to implement real-time volume rendering on today's hardware are presented.

B.1 Derivation

To render realistic images of the physical environment, an accurate model of light and its interactions with matter is needed. Transport theory as described in [Duderstadt and Martin, 1979] provides a model that can be adapted for this purpose. In this context, the following assumptions are being made:

- Photons are treated as particles. This has the disadvantage that the model is not able to capture effects that require wave optics to describe them, such as interference, diffraction or dispersion. Some of these effects can be easily added later on, however.

- All photons travel at the same speed. This assumption holds for scenes that only have monochromatic light sources and have no materials that change the frequency of light. By evaluating the model independently for different wavelengths - and thus photon speeds - arbitrary spectra for light sources can be simulated.

- The position $\mathbf{x} \in R^3$ and direction $\omega \in S^2$ of a photon fully describe its state. Each photon can therefore be described by a point in the five-dimensional *phase space* $R^3 \times S^2$.

- Photons are both small and numerous so that their statistical distribution over the phase space can be treated as a continuum.

- The only events that can change the phase space distribution are 1. the emission of photons, 2. the streaming of photons, i.e. their movement along their propagation direction without interaction with the medium they travel through, and finally 3. the collision of photons with medium particles. The latter can be further subdivided into 3a. the absorption of photons and 3b.

the scattering of photons. There are no self-collisions of photons. Also, the same-speed assumption implies that all collisions with the medium have to be elastic, i.e. they only change the propagation direction of photons, but not their speed.

- The streaming of photons is always in straight lines, so a varying index of refraction inside a medium can not be modeled. One example for a scene where a varying index of refraction is important is the "flickering air" above a highway road on extremely hot or extremely cold days.

- Collisions of photons with the medium are not explicitly modeled. Both the absorption and scattering of photons are instead described statistically by *coefficients of absorption* $a(\mathbf{x})$ and *scattering* $b(\mathbf{x})$ respectively. These coefficients are allowed to change as a function of position $\mathbf{x}$. They describe the probability that a photon is absorbed (or scattered) per unit travel distance. (For non-isotropic media they are also allowed to change as a function of propagation direction ω.) Given that a photon traveling in direction ω undergoes a scattering event at position $\mathbf{x}$, the probability density that after the scattering event the photon is traveling in direction ω' is given by the *phase function* $P(\mathbf{x}, \omega, \omega')$.

- The phase space is in a steady state, i.e. the distribution of photons over the phase space does not change over time. Since photons are continuously moving, being created, absorbed and scattered, yet their distribution does not change, the phase space is described as being in a dynamic equilibrium. Simply put, this assumption means that the light has been "turned on" and enough time has already passed for the photons to have reached every distant corner of the scene.

Despite the many assumptions that have been made about the nature of light and its interactions with the medium it traverses, the particle transport model has proven that it is able to reproduce many physical phenomena to a high degree of photo-realism.

In the following, a mathematical expression for the distribution of photons over the phase space is derived. More concise, if v is the photon velocity and $n(\mathbf{x}, \omega)$ is the number of photons at position $\mathbf{x}$ traveling in direction ω per unit volume per unit solid angle, then the *phase space flux* $\phi_p(\mathbf{x}, \omega) = v \cdot n(\mathbf{x}, \omega)$ describes the number of photons passing through a plane at position $\mathbf{x}$ that is orthogonal to the propagation direction ω per unit area per unit solid angle per unit time. If the number of photons is multiplied by the energy E that each photon carries, the following simple relationship between the phase space flux ϕ_p and the radiometric measure of radiance L can be stated: $L(\mathbf{x}, \omega) = E \cdot \phi_p(\mathbf{x}, \omega)$. See [Arvo, 1993] for

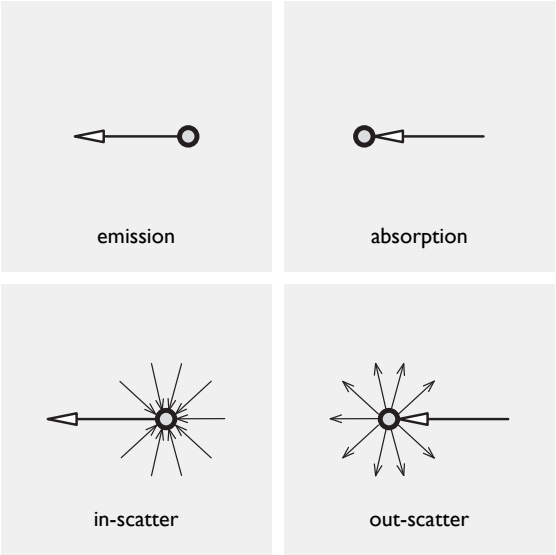


Figure B.1: The events that change the amount of radiance at a given position and direction.

a longer treatment of these relationships. With these insights, the goal of finding the distribution of photons over the phase space is equivalent to finding an expression for the radiance at an arbitrary position $\mathbf{x}$ in an arbitrary direction ω. This expression should have the form $L(\mathbf{x}, \omega) = \ldots$.

To get to such an expression, first the amount of change of radiance at a given position and towards a given direction will be described. Since the only events the particle transport model allows are streaming, emission, absorption and scattering of photons, and since streaming does not change the amount of radiance, the change of radiance can be written as:

$$(\omega \cdot \nabla)L(\mathbf{x}, \omega) = E(\mathbf{x}, \omega) + A(\mathbf{x}, \omega) + S(\mathbf{x}, \omega), \tag{B.1}$$

where E accounts for the emitted radiance, A accounts for the absorbed radiance and S accounts for the scattered radiance. S is usually further subdivided into the in-scattered radiance S_{in} and out-scattered radiance S_{out}. See Figure B.1 for a picture of these events.

The next step is to find an expression for each of the terms on the right hand side of Equation B.1. The emission of photons is the simplest term if the *radiance source function* L_e is introduced, that denotes how many photons are generated at position $\mathbf{x}$ in direction ω per unit volume per unit solid angle per unit time. With that, E becomes:

$$E(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega). \tag{B.2}$$

The absorption term can be described by taking the incoming radiance at position $\mathbf{x}$ from direction ω and multiplying that by the absorption coefficient. Since absorption decreases the amount of radiance, a minus sign has to be included in the equation:

$$A(\mathbf{x}, \omega) = -\, a(\mathbf{x}) \cdot L(\mathbf{x}, \omega). \tag{B.3}$$

For the amount of light that is in-scattered into direction ω, the incoming radiance at position $\mathbf{x}$ from all possible directions ω' has to be considered. The scattered amount of that radiance is then determined by multiplying it with the scattering coefficient. Finally, the phase function is used to determine what fraction of the scattered light is actually scattered into the direction ω:

$$S_{in}(\mathbf{x}, \omega) = \int_{S^2} P(\mathbf{x}, \omega', \omega) \cdot b(\mathbf{x}) \cdot L(\mathbf{x}, \omega')\ d\omega'. \tag{B.4}$$

The amount of out-scattered radiance can be determined similarly by considering the incoming light at position $\mathbf{x}$ from direction ω and integrating over outgoing directions ω':

$$S_{out}(\mathbf{x}, \omega) = -\int_{S^2} P(\mathbf{x}, \omega, \omega') \cdot b(\mathbf{x}) \cdot L(\mathbf{x}, \omega)\ d\omega'. \tag{B.5}$$

The only term in Equation B.5 that depends on the integration variable is the phase function. Taking into account that the phase function is a probability density over the sphere of directions and must therefore satisfy $\int_{S^2} P(\mathbf{x}, \omega, \omega')\, d\omega' = 1$ for arbitrary $\mathbf{x} \in R^3$ and $\omega \in S^2$, the out-scattering term simplifies to:

$$S_{out}(\mathbf{x}, \omega) = -\, b(\mathbf{x}) \cdot L(\mathbf{x}, \omega). \tag{B.6}$$

Finally, by defining the *extinction coefficient* $c(\mathbf{x}) = a(\mathbf{x}) + b(\mathbf{x})$, Equations B.3 and B.6 can be combined and Equation B.1 can be written as:

$$\begin{aligned}(\omega \cdot \nabla) L(\mathbf{x}, \omega) \ &= L_e(\mathbf{x}, \omega) \\ &\quad + \int_{S^2} P(\mathbf{x}, \omega', \omega) \cdot b(\mathbf{x}) \cdot L(\mathbf{x}, \omega')\ d\omega' \\ &\quad - c(\mathbf{x}) L(\mathbf{x}, \omega).\end{aligned}$$

If the emission and in-scattering term are further combined into the *source term* $Q(\mathbf{x}, \omega)$, the equation becomes:

$$(\omega \cdot \nabla) L(\mathbf{x}, \omega) = Q(\mathbf{x}, \omega) - c(\mathbf{x}) L(\mathbf{x}, \omega). \tag{B.7}$$

This equation is called the *transfer equation in integro-differential form* — "integro" because it has an integral involving the radiance (hidden inside the Q term) and "differential" because it has a differentiation operation $(\omega \cdot \nabla)$ involving the

radiance. This equation, however, only defines the directional change of radiance at a specific point. To fully describe the behavior of light, boundary conditions have to be introduced. Boundaries in this context are surfaces that can not be penetrated by photons. For every point $\mathbf{b}$ on such a surface, the surface normal at that point is denoted by $\mathbf{n_b}$ and for every outgoing direction $\omega \in \Omega = \{\omega \mid (\omega \cdot \mathbf{n}_b) > 0\}$ the following boundary condition is defined:

$$L_B(\mathbf{b}, \omega) = L_{eB}(\mathbf{b}, \omega) + \int_\Omega f_r(\mathbf{x}, \omega', \omega) \cdot L(\mathbf{x}, \omega') \cdot (\omega' \cdot \mathbf{n}_b)\ d\omega', \tag{B.8}$$

where L_{eB} describes the emission of photons from the surface and f_r is the BRDF of the surface. Together, Equations B.7 and B.8 form a complete description of the phase space distribution of radiance.

For the design of an algorithm that computes the radiance at a given position and direction, however, it turns out that a reformulation of the equation of transfer into a pure integral form is better suited. [Arvo, 1993] shows how this reformulation can be accomplished starting from Equations B.7 and B.8. Here, only the final formula will be presented. To that end, two more terms have to be defined. The *optical distance* l_c between two points $\mathbf{x}$ and $\mathbf{x}'$ that counts the number of scattering and absorption events is defined as:

$$l_c(\mathbf{x}, \mathbf{x}') = \int_0^{||\mathbf{x}-\mathbf{x}'||} c(\mathbf{x} - z\mathbf{d})\ dz, \tag{B.9}$$

where $\mathbf{d} = \frac{\mathbf{x}'-\mathbf{x}}{||\mathbf{x}'-\mathbf{x}||}$, and the *path absorption* β [1] that describes the attenuation of radiance between two points $\mathbf{x}$ and $\mathbf{x}'$ is:

$$\beta(\mathbf{x}, \mathbf{x}') = \exp(-l_c(\mathbf{x}, \mathbf{x}')). \tag{B.10}$$

With these definitions, the *transfer equation in integral form* can be formulated:

$$L(\mathbf{x}, \omega) = \beta(\mathbf{x}, \mathbf{b}) L_B(\mathbf{b}, \omega) + \int_0^{||\mathbf{x}-\mathbf{b}||} \beta(\mathbf{x}, \mathbf{x} - t\omega)\, Q(\mathbf{x} - t\omega, \omega)\ dt, \tag{B.11}$$

where $\mathbf{b}$ is the first encountered surface point in direction $-\omega$ starting from position $\mathbf{x}$. Figure B.2 shows a schematic sketch of this equation.

[1] The name path *extinction* would have been a better fit for this function, since it hints at the fact that both absorption and scattering events lead to the attenuation of radiance.

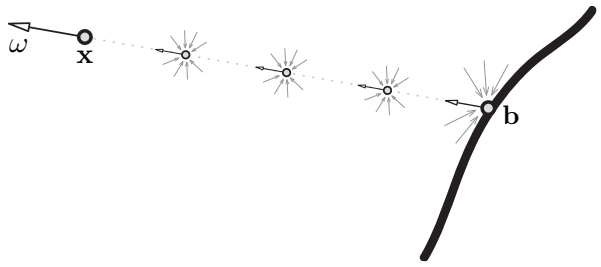


Figure B.2: A visualization of the equation of transfer in integral form. To determine the radiance at point $\mathbf{x}$ pointing into direction ω, the first point on the boundary, $\mathbf{b}$, is determined. Then, the radiance at every point between $\mathbf{x}$ and $\mathbf{b}$ that is sent into direction ω is collected and attenuated by the path absorption β from that point to $\mathbf{x}$.

B.2 Emission-Absorption Model

The equation of transfer presented in Equation B.11 can not be implemented efficiently on today's hardware. One way to simplify this equation is to assume that the participating medium does not scatter light, i.e., that $b(\mathbf{x}) = 0$. Equation B.11 then simplifies to:

$$L(\mathbf{x},\omega) = \beta(\mathbf{x},\mathbf{b})L_B(\mathbf{b},\omega) + \int_0^{||\mathbf{x}-\mathbf{b}||} \beta(\mathbf{x},\mathbf{x}-t\omega)\, L_e(\mathbf{x}-t\omega,\omega)\; dt. \tag{B.12}$$

For notational simplicity, it is further assumed that there are no boundaries in the scene, i.e., that it only consists of the participating medium. With that, and writing out the path absorption β, the *emission-absorption equation* can be written as:

$$L(\mathbf{x},\omega) = \int_0^{\infty} \underbrace{\exp\left[-\int_0^t a(\mathbf{x}-z\omega)\,dz\right]}_{\text{path absorption}} \cdot \underbrace{L_e(\mathbf{x}-t\omega,\omega)}_{\text{emission}}\; dt. \tag{B.13}$$

In the following, a will be renamed to σ and the radiance source function for the volume, L_e will be renamed to ϵ in order to avoid confusion with L_{eB}, which is the radiance source function for surfaces.

For practical purposes, the participating medium always has a limited spatial extent. So, for a given point $\mathbf{x}$, the distance of the closest point $t_{\min}$ and the distance of the farthest point $t_{\max}$ of the medium along direction $-\omega$ can be computed. With these definitions, the emission-absorption equation becomes (see Figure B.3):

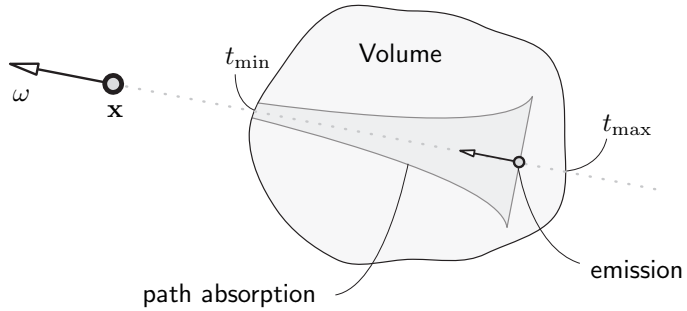


Figure B.3: Schematic display of Equation B.14. The emission and path absorption are shown for an arbitrary point inside the volume.

$$L(\mathbf{x},\omega) = \int_{t_{\min}}^{t_{\max}} \exp\left[-\int_{t_{\min}}^{t} \sigma(\mathbf{x}-z\omega)\,dz\right] \cdot \epsilon(\mathbf{x}-t\omega,\omega)\;dt. \tag{B.14}$$

B.3 Discretization

In order to evaluate the integrals in Equation B.14, both have to be discretized. This is done by evaluating the integrand at $N+1$ equally spaced points $\mathbf{x}_i$ with $i = 0 \ldots N$, where the emission and absorption at these points will be denoted by σ_i and ϵ_i. The step size is then $\Delta t = \frac{t_{\max}-t_{\min}}{N}$.

If the inner integral is evaluated with the same step size and at the same points $\mathbf{x}_i$ as the outer integral, the discretization of Equation B.14 becomes:

$$\begin{aligned} L(\mathbf{x},\omega) &= \sum_{i=0}^{N} \exp\left[-\sum_{j=i+1}^{N} \sigma_j \cdot \Delta t\right] \cdot \epsilon_i \cdot \Delta t \\ &= \sum_{i=0}^{N} \left(\prod_{j=i+1}^{N} \exp\left[-\sigma_j \cdot \Delta t\right]\right) \cdot \epsilon_i \cdot \Delta t. \end{aligned} \tag{B.15}$$

Defining $A_j = \exp(-\sigma_j \cdot \Delta t)$, the equation can be written more compact as:

$$L(\mathbf{x},\omega) = \sum_{i=0}^{N} \left(\prod_{j=i+1}^{N} A_j\right) \cdot \epsilon_i \cdot \Delta t. \tag{B.16}$$

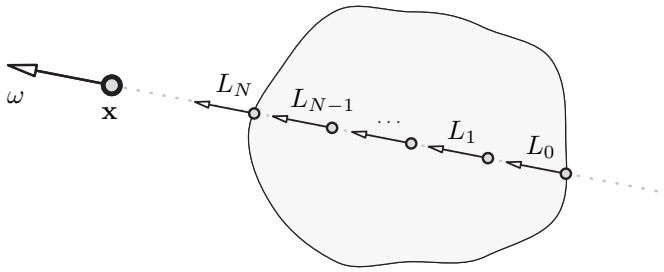


Figure B.4: The discrete points used to approximate the volume integral.

In order to efficiently implement this equation, some reorganization is required. First, the sum and products are expanded:

$$\begin{aligned} L(\mathbf{x},\omega) = \Delta t \cdot \big[\quad & (A_N \cdot A_{N-1} \cdots A_2 \cdot A_1) \cdot \epsilon_0 \\ & + (A_N \cdot A_{N-1} \cdots A_2) \cdot \epsilon_1 \\ & \vdots \\ & + (A_N) \cdot \epsilon_{N-1} \\ & + \epsilon_N \big] \end{aligned} \tag{B.17}$$

Now, the terms can be reordered to give:

$$\begin{aligned} L(\mathbf{x},\omega) &= \Delta t \cdot [\ \epsilon_N + A_N(\ldots A_2(\epsilon_1 + A_1(\epsilon_0)))\ ] \\ &= \Delta t\epsilon_N + A_N(\ldots A_2(\Delta t\epsilon_1 + A_1(\Delta t\epsilon_0))) \end{aligned} \tag{B.18}$$

Casting Equation B.18 in a recursive form leads to:

$$\begin{aligned} L_i &= \Delta t \cdot \epsilon_i + A_i \cdot L_{i-1}, \\ L_0 &= \Delta t \cdot \epsilon_0, \end{aligned} \tag{B.19}$$

where L_N is the radiance leaving the border of the volume and is therefore the same as $L(\mathbf{x},\omega)$ (see Figure B.4).

B.4 Arbitrary Order of Volume Rendering

Equation B.19 is used in Chapter 5 to describe a volume rendering approach on the GPU. In Chapter 3, it has been noted that the rendering order of different parts of the volume is irrelevant. As long as the radiance contribution and the path

absorption for each part is stored, the final result can easily be computed.

In order to prove this claim, using the same discrete points from the previous chapter, for any numbers m, n with $0 \leq n < m \leq N$, the path absorption $A_{m,n}$ between $\mathbf{x}_n$ and $\mathbf{x}_m$ is defined as:

$$A_{m,n} \;=\; \prod_{j=n}^{m} A_j, \tag{B.20}$$

and according to Equation B.16, the contributed radiance $L_{m,n}$ of that segment is:

$$L_{m,n} \;=\; \sum_{i=n}^{m} \; A_{m,i+1} \;\cdot\; \epsilon_i \;\cdot\; \Delta t. \tag{B.21}$$

For an arbitrary chosen k with $n < k < m$, the following relationship can be formulated:

$$\begin{aligned} L_{m,n} \;&=\; \sum_{i=n}^{m} \; A_{m,i+1} \cdot \epsilon_i \cdot \Delta t \\ &=\; \underbrace{\sum_{i=k+1}^{m} A_{m,i+1} \cdot \epsilon_i \cdot \Delta t}_{L_{m,k+1}} \;+\; \sum_{i=n}^{k} \; A_{m,i+1} \cdot \epsilon_i \cdot \Delta t \\ &=\; L_{m,k+1} \;+\; \sum_{i=n}^{k} \; A_{m,k+1} \cdot A_{k,i+1} \cdot \epsilon_i \cdot \Delta t \\ &= L_{m,k+1} \;+\; A_{m,k+1} \cdot L_{k,n}. \end{aligned} \tag{B.22}$$

Since $L_N = L_{N,0}$ holds true, this relationships shows that the computation of the final radiance can indeed be split up arbitrarily into many smaller segments.

C Functions over the Sphere

In this thesis, various calculations use functions over the hemisphere. This appendix explains the mathematical background in more detail.

Functions over the sphere are often defined as a mapping of polar coordinates $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ to their respective function values. For hemispheres, θ lies in $[0, \frac{\pi}{2}]$, where θ is 0 at the pole, and $\frac{\pi}{2}$ at the equator. Each pair of (θ, ϕ) values has a corresponding direction vector ω in Eucledian coordinates:

$$\omega = \begin{pmatrix} \sin\theta \cdot \cos\phi \\ \sin\theta \cdot \sin\phi \\ \cos\theta \end{pmatrix},$$

so each function $f(\theta, \phi)$ can also be written as $f(\omega)$.

When integrating a function over the hemisphere, the integration domain is denoted by Ω:

$$\int_{\Omega} f(\omega)\, d\omega. \tag{C.1}$$

Here, $d\omega$ is the differential solid angle in direction ω, i.e., the differential area of the surface patch that is located on the unit hemisphere in direction ω. This differential solid angle can also be expressed in polar coordinates as follows (see [Glassner, 1995]):

$$d\omega = d\theta \cdot (\sin\theta \cdot d\phi)$$

Uniform Probability Density Function. A uniform probability density function over the hemisphere has to integrate to one. If $p_u(\omega) = c$ is this function, then c

has to be determined:

$$\begin{aligned}
&\int_\Omega p_u(\omega)\, d\omega \\
&= c \cdot \int_0^{\frac{\pi}{2}} \int_0^{2\pi} \sin\theta \cdot d\phi \cdot d\theta \\
&= c \cdot \int_0^{\frac{\pi}{2}} \left(\int_0^{2\pi} d\phi \right) \cdot \sin\theta \cdot d\theta \\
&= c \cdot 2\pi \cdot \int_0^{\frac{\pi}{2}} \sin\theta \cdot d\theta \\
&= c \cdot 2\pi \cdot \left[-\cos\theta \right]_0^{\frac{\pi}{2}} \\
&= c \cdot 2\pi
\end{aligned}$$

Therefore, the uniform probability density function over the hemisphere is given by:

$$p_u(\omega) = \frac{1}{2\pi}. \tag{C.2}$$

Cosine Probability Density Function. For the speedup of the ambient occlusion calculation in Chapter 4, a pdf p that is proportional to $\cos\theta$ is needed. The normalization factor c can be computed just like before:

$$\begin{aligned}
&\int_\Omega p(\omega)\, d\omega \\
&= c \cdot \int_0^{\frac{\pi}{2}} \int_0^{2\pi} \cos\theta \cdot \sin\theta \cdot d\phi \cdot d\theta \\
&= c \cdot \int_0^{\frac{\pi}{2}} \left(\int_0^{2\pi} d\phi \right) \cdot \cos\theta \cdot \sin\theta \cdot d\theta \\
&= c \cdot 2\pi \cdot \int_0^{\frac{\pi}{2}} \cos\theta \cdot \sin\theta \cdot d\theta \\
&= c \cdot 2\pi \cdot \frac{1}{2} \left[-\cos^2\theta \right]_0^{\frac{\pi}{2}} \\
&= c \cdot \pi
\end{aligned}$$

So, the probability density function over the hemisphere that is proportional to $\cos\theta$ is given by:

$$p(\omega) = \frac{1}{\pi} \cos\theta \tag{C.3}$$

Bibliography

[Aguirre, 2006] Aguirre, G. K. (2006). Experimental Design and Data Analysis for fMRI. In Faro, S. H. and Mohamed, F. B., editors, *Functional MRI*, chapter 3, pages 58–74. Springer Science + Business Media, Inc.

[Amanatides and Woo, 1987] Amanatides, J. and Woo, A. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland.

[A.M.Dale et al., 1999] A.M.Dale, Fischl, B., and M.I.Sereno (1999). Cortical Surface-Based Analysis I: Segmentation and Surface Reconstruction. *NeuroImage*, 9(2):179–194.

[Arvo, 1993] Arvo, J. (1993). Transfer Functions in Global Illumination. In *ACM SIGGRAPH '93 Course Notes - Global Illumination*, pages 1–28.

[Baert et al., 2000] Baert, A. L., Sartor, K., and Youker, J. E. (2000). *Functional MRI*. Springer-Verlag Berlin.

[Bandettini, 2007] Bandettini, P. (2007). Functional MRI today. *International Journal of Psychophysiology*, 63:138–145.

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517.

[Bernardini and Bajaj, 1997] Bernardini, F. and Bajaj, C. L. (1997). Sampling and Reconstructing Manifolds using Alpha-Shapes. In *Proc. 9th Canadian Conf. Computational Geometry*, pages 193–198.

[BrainTutor, Web] BrainTutor (Web). http://www.brainvoyager.com/braintutor.html.

[Brett, 2002] Brett, M. (2002). *http://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach*.

[Brett et al., 2002] Brett, M., Johnsrude, I., and Owen, A. (2002). The problem of functional localization in the human brain. *Nature Reviews Neuroscience*, 3(3):243–9.

[Brodmann, 1909] Brodmann, K. (1909). *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Barth, Leipzig.

[Brown and Lichtenbelt, 2007] Brown, P. and Lichtenbelt, B. (2007). EXT_geometry_shader4. *OpenGL Extension Registry, http://www.opengl.org/registry/specs/EXT/geometry_shader4.txt*.

[Buck et al., 2004] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. (2004). Brook for GPUs: stream computing on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 777–786, New York, NY, USA. ACM.

[Cabral et al., 1994] Cabral, B., Cam, N., and Foran, J. (1994). Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA. ACM Press.

[Coombe et al., 2005] Coombe, G., Harris, M. J., and Lastra, A. (2005). Radiosity on graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 179, New York, NY, USA. ACM.

[Csebfalvi et al., 2001] Csebfalvi, B., Knig, A., Grller, E., Mroz, L., and Hauser, H. (2001). Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum (Proceedings of EuroGraphics '01)*, 20(3):452–60.

[de Haan, 2000] de Haan, B. (2000). *fMRI Guide, http://www.sph.sc.edu/comd/rorden/fmri_guide/index.html*.

[Duderstadt and Martin, 1979] Duderstadt, J. J. and Martin, W. R. (1979). *Transport Theory*. John Wiley & Sons, New York, NY.

[Dutre et al., 2002] Dutre, P., Bala, K., and Bekaert, P. (2002). *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, USA.

[Edelsbrunner and Mücke, 1992] Edelsbrunner, H. and Mücke, E. P. (1992). Three-dimensional alpha shapes. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization*, pages 75–82, New York, NY, USA. ACM.

[Encarnação et al., 1996] Encarnação, J., Straßer, W., and Klein, R. (1996). *Graphische Datenverarbeitung 1*. R. Oldenburg Verlag, München Wien.

[Faro and Mohamed, 2006] Faro, S. H. and Mohamed, F. B. (2006). *Functional MRI: Basic Principles and Clinical Applications*. Springer Science + Business Media, Inc.

[Fischer et al., 2005] Fischer, J., Bartz, D., and Straßer, W. (2005). Illustrative Display of Hidden Iso-Surface Structures. In *Proceedings of IEEE Visualization*, pages 663–670.

[Fischer et al., 2007] Fischer, J., Flohr, D., and Straßer, W. (2007). Seamless tangible interaction through selective stylization. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 11, New York, NY, USA. ACM.

[Fischl et al., 2001] Fischl, B., Liu, A., and Dale, A. M. (2001). Automated Manifold Surgery: Constructing Geometrically Accurate and Topologically Correct Models of the Human Cerebral Cortex. *IEEE Transactions on Medical Imaging*, 20(1):70–80.

[Fischl et al., 2002] Fischl, B., Salat, D., Busa, E., Albert, M., Dietrich, M., Haselgrove, C., van der Kouwe, A., Killiany, R., Kennedy, D., Klaveness, S., Montillo, A., Makris, N., Rosen, B., and Dale, A. M. (2002). Whole brain segmentation: Automated labeling of neuroanatomical structures in the human brain. *Neuron*, 31;33(3):341–55.

[Fischl et al., 1999] Fischl, B., Sereno, M. I., and Dale, A. M. (1999). Cortical Surface-Based Analysis II: Inflation, Flattening, and Surface-Based Coordinate System. *NeuroImage*, 9(2):195–207.

[Fischl et al., 2004] Fischl, B., van der Kouwe, A., Destrieux, C., Halgren, E., Segonne, F., Salat, D., Busa, E., Seidman, L. J., Goldstein, J., Kennedy, D., Caviness, V., Makris, N., Rosen, B., and Dale, A. M. (2004). Automatically parcellating the human cerebral cortex. *Cereb Cortex*, 14(1):11–22.

[Fox, 1997] Fox, P. T. (1997). The Growth of Human Brain Mapping. *Human Brain Mapping*, 5(1):1–2.

[Free Surfer Wiki, 2007] Free Surfer Wiki (2007). http://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki.

[Friston et al., 2004] Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J.-P., Frith, C. D., and Frackowiak, R. S. J. (2004). Statistical parametric maps in functional imaging: A general linear approach. *Human Brain Mapping*, 2(4):189–210.

[Gelder and Kim, 1996] Gelder, A. V. and Kim, K. (1996). Direct volume rendering with shading via three-dimensional textures. In *VVS '96: Proceedings of the 1996 symposium on Volume visualization*, pages 23–ff., Piscataway, NJ, USA. IEEE Press.

[Glassner, 1995] Glassner, A. S. (1995). *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA.

[Gooch and Gooch, 2001] Gooch, B. and Gooch, A. (2001). *Non-Photorealistic Rendering*. A. K. Peters, Ltd., Natick, MA, USA.

[Haeger and Ress, 2002] Haeger, D. J. and Ress, D. (2002). What does fMRI tell us about neuronal activity? *Nature Reviews Neuroscience*, 3:142–151.

[Hall and Watt, 1991] Hall, P. M. and Watt, A. H. (1991). Rapid volume rendering using a boundary-fill guided ray cast algorithm. pages 235–249.

[Haralick et al., 1987] Haralick, R. M., Sternberg, S. R., and Zhuang, X. (1987). Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):532–550.

[Hegeman et al., 2005] Hegeman, K., Ashikhmin, M., and Premoze, S. (2005). A lighting model for general participating media. In Lastra, A., Olano, M., Luebke, D. P., and Pfister, H., editors, *Proceedings of the 2005 Symposium on Interactive 3D Graphics, SI3D 2005, April 3-6, 2005, Washington, DC, USA*, pages 117–124. ACM.

[Hirsch, 2006] Hirsch, J. (2006). Brain Mapping for Neurosurgery and Cognitive Neuroscience. In Faro, S. H. and Mohamed, F. B., editors, *Functional MRI: Basic Principles and Clinical Applications*, chapter 7, pages 139–182. Springer Science + Business Media, Inc., New York, NY, USA.

[Jansen, 1986] Jansen, F. W. (1986). Data structures for ray tracing. In *Proceedings of a workshop (Eurographics Seminars on Data structures for raster graphics*, pages 57–73, New York, NY, USA. Springer-Verlag New York, Inc.

[Jensen, 2001] Jensen, H. W. (2001). *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA.

[Junker, 2006] Junker, G. (2006). *Pro OGRE 3D Programming (Pro)*. Apress, Berkely, CA, USA.

[Kajiya, 1986] Kajiya, J. T. (1986). The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA. ACM.

[Kandel, 1991] Kandel, E. R. (1991). The Brain and Behavior. In Kandel, E. R., Schwarz, J. H., and Jessell, T. M., editors, *Principles of Neural Science*, chapter 1, pages 5–18. McGraw Hill, 4th edition.

[Kass et al., 1988] Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331.

[Kim and Bandettini, 2006] Kim, S.-G. and Bandettini, P. A. (2006). Principles of Functional MRI. In Faro, S. H. and Mohamed, F. B., editors, *Functional MRI*, chapter 1, pages 3–23. Springer Science + Business Media, Inc.

[Langer and Bülthoff, 1999] Langer, M. S. and Bülthoff, H. H. (1999). Perception of shape from shading on a cloudy day. *Technical Report No. 73*, Max-Planck-Institut für biologische Kybernetik.

[Lensch et al., 2002] Lensch, H., Gösele, M., Bekaert, P., Kautz, J., Magnor, M., Lang, J., and Seidel, H.-P. (2002). Interactive Rendering of Translucent Objects. *Proc. IEEE Pacific Graphics 2002,* Beijing, China, pages 214–224.

[Logothetis et al., 2001] Logothetis, N. K., Pauls, J., Augath, M., Trinath, T., and Oeltermann, A. (2001). Neurophysiological investigation of the basis of the fMRI signal. *Nature*, 412:150–157.

[Luebke et al., 2004] Luebke, D., Harris, M., Krüger, J., Purcell, T., Govindaraju, N., Buck, I., Woolley, C., and Lefohn, A. (2004). GPGPU: general purpose computation on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, page 33, New York, NY, USA. ACM.

[Ma et al., 2003] Ma, Y., Soatto, S., Kosecka, J., and Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag.

[MacDonald and Booth, 1990] MacDonald, D. J. and Booth, K. S. (1990). Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166.

[Mallot, 1998] Mallot, H. A. (1998). *Sehen und die Verarbeitung visueller Informationen: Eine Einführung*. Vieweg, Braunschweig, Wiesbaden.

[McReynolds and Blythe, 2005] McReynolds, T. and Blythe, D. (2005). *Advanced Graphics Programming Using OpenGL (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Mitchell et al., 2002] Mitchell, J. L., Brennan, C., and Card, D. (2002). Real-time image-space outlining for non-photorealistic rendering. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*, pages 239–239, New York, NY, USA. ACM.

[Mueller et al., 1999] Mueller, K., Möller, T., and Crawfis, R. (1999). Splatting without the blur. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 363–370, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Overmars, 1996] Overmars, M. H. (1996). Designing the computational geometry algorithms library cgal. In *FCRC '96/WACG '96: Selected papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering*, pages 53–58, London, UK. Springer-Verlag.

[Pekar, 2006] Pekar, J. J. (2006). A brief introduction to functional MRI. *Engineering in Medicine and Biology Magazine, IEEE*, 25(2):24–26.

[Pharr and Humphreys, 2004] Pharr, M. and Humphreys, G. (2004). *Physically based rendering from theory to implementation*. The Morgan Kaufmann series in interactive 3D technology. Elsevier/Morgan Kaufmann., Amsterdam.

[Pinus and Mohamed, 2006] Pinus, A. B. and Mohamed, F. B. (2006). fMRI Scanning Methodologies. In Faro, S. H. and Mohamed, F. B., editors, *Functional MRI*, chapter 2, pages 24–57. Springer Science + Business Media, Inc., New York, NY, USA.

[Polyak, 1957] Polyak, S. (1957). *The Vertebrate Visual System*. University of Chicago Press, Chicago.

[Purcell, 2005] Purcell, T. (2005). Sorting and searching. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 79, New York, NY, USA. ACM.

[Purcell et al., 2005] Purcell, T. J., Donner, C., Cammarano, M., Jensen, H. W., and Hanrahan, P. (2005). Photon mapping on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 258, New York, NY, USA. ACM.

[Qt, Web] Qt (Web). http://trolltech.com/products/qt.

[Ragan-Kelley et al., 2007] Ragan-Kelley, J., Kilpatrick, C., Smith, B. W., Epps, D., Green, P., Hery, C., and Durand, F. (2007). The lightspeed automatic interactive lighting preview system. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 25, New York, NY, USA. ACM.

[Roerdink and Meijster, 2000] Roerdink, J. B. T. M. and Meijster, A. (2000). The watershed transform: definitions, algorithms and parallelization strategies. *Fundam. Inf.*, 41(1-2):187–228.

[Rost, 2006] Rost, R. J. (2006). *OpenGL Shading Language (2nd Edition)*. Addison-Wesley Professional.

[Savoy, 2001] Savoy, R. L. (2001). History and future directions of human brain mapping and functional neuroimaging. *Acta Psychologica*, 107(1-3):9–42.

[Schafhitzel et al., 2007] Schafhitzel, T., Rößler, F., Weiskopf, D., and Ertl, T. (2007). Simultaneous visualization of anatomical and functional 3D data by combining volume rendering and flow visualization. *Medical Imaging 2007: Visualization and Image-Guided Procedures.*

[Schmittler et al., 2002] Schmittler, J., Wald, I., and Slusallek, P. (2002). SaarCOR: a hardware architecture for ray tracing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 27–36, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Segonne et al., 2004] Segonne, F., Dale, A. M., Busa, E., Glessner, M., Salvolini, U., Hahn, H., and Fischl, B. (2004). A Hybrid Approach to the Skull Stripping Problem in MRI. *NeuroImage*.

[Shoemake, 1985] Shoemake, K. (1985). Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA. ACM.

[Shreiner et al., 2005] Shreiner, D., Woo, M., Neider, J., and Davis, T. (2005). *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 2 (5th Edition)*. Addison-Wesley Professional.

[Sillion and Puech, 1994] Sillion, F. X. and Puech, C. (1994). *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Sousa, 2003] Sousa, C. (2003). Indexed Taxonomies of Non-Photorealistic Rendering. *Course Notes for SIGGRAPH 2003*.

[SPM, Web] SPM (Web). http://www.fil.ion.ucl.ac.uk/spm/.

[Stokking et al., 2001] Stokking, R., Zuiderveld, K. J., and Viergever, M. A. (2001). Integrated volume visualization of functional image data and anatomical surfaces using normal fusion. *Human Brain Mapping*, 12(4):203–18.

[Strothotte and Schlechtweg, 2002] Strothotte, T. and Schlechtweg, S. (2002). *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Talairach and Tournoux, 1988] Talairach, J. and Tournoux, P. (1988). *Co-Planar Stereotaxic Atlas of the Human Brain: 3-Dimensional Proportional System: An Approach to Cerebral Imaging*. Thieme, New York, NY, USA.

[Treavett and Chen, 2000] Treavett, S. M. F. and Chen, M. (2000). Pen-and-ink rendering in volume visualization. *Proceedings of IEEE Visualization '00*, pages 203–10.

[Ullmann, 2000] Ullmann, S. (2000). *High-Level Vision: Object Recognition and Visual Cognition*. The MIT Press.

[Weiler et al., 2006] Weiler, M., Kraus, M., Guthe, S., Ertl, T., and Straßer, W. (2006). *Scientific Visualization: The Visual Extraction of Knowledge from Data*, chapter Ray Casting with Programmable Graphics Hardware, pages 115–130. Springer Berlin Heidelberg.

[Westover, 1990] Westover, L. (1990). Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA. ACM Press.

[Whitted, 1980] Whitted, T. (1980). An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349.

[Wikipedia, fMRI] Wikipedia (fMRI). http://en.wikipedia.org/wiki/fmri.

[Wittenbrink et al., 1998] Wittenbrink, C. M., Malzbender, T., and Goss, M. E. (1998). Opacity-weighted color interpolation, for volume sampling. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 135–142, New York, NY, USA. ACM.

[Woo et al., 1997] Woo, M., Neider, J., and Davis, T. (1997). *OpenGL Programming Guide (2nd ed.): The Official Guide to Learning OpenGL Version 1.1.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Xu and Prince, 1998] Xu, C. and Prince, J. L. (1998). Snakes, Shapes, and Gradient Vector Flow. *IEEE Transactions on Image Processing*, 7:359–369.