FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Noise and Artifact Reduction in Interactive Volume Renderings of Electron-Microscopy Data-Sets.

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Computergraphik/Digitale Bildverarbeitung**

eingereicht von

**Andreas Ritzberger**
Matrikelnummer 0527000

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. Dr.techn. Markus Hadwiger, Dipl.-Ing. (FH) Dr.techn. Johanna Beyer

Wien, 15.05.2010

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

# Widmung

*Gesegnet sind die, die geben können, ohne sich daran zu erinnern*
*und die, die nehmen können, ohne es zu vergessen.*
*-Melvin Schleeds*

Mit dieser Widmung möchte ich mich bei meinen Eltern bedanken, die mich all die Jahre selbstlos und tatkräftig in meiner schulischen und studentischen Laufbahn unterstützt haben. Ohne euch wäre ich nicht dort wo ich heute bin!

Danke!

# Erklärung zur Verfassung der Arbeit

**Andreas Ritzberger, Zinckgasse 22/Top 53, 1150 Wien**

Hiermit erkäre ich, dass ich diese Arbeit selbsändig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollsändig angegeben habe und dass ich die Stellen der Arbeit - einschlielich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. Mai 2010

(Unterschrift Verfasser)

# Abstract

Connectomics is an emerging area of neuroscience that is concerned with understanding the neural algorithms embeded in the neural circuits of the brain by tracking neurons and studying their connections. From all the available scanning technologies only electron microscopy (EM) can provide sufficient scanning resolutions in order to identify neural processes. EM data sets, however, suffer from bad signal-to-noise ratio and artifacts introduced to the data set during the sectioning and digital reconstruction process of the scanned specimen. In this thesis we present two different approaches that generally allow noise and artifact reduction on volumetric data sets and which can be used to increase the visual quality of direct volume renderings (DVRs) of EM data sets. The fist approach we developed was an interactive, on-the-fly filtering framework that allows a user to filter even very large volume data set with resizable 3D filter-kernels. For comparison, we implemented an average, a Gaussian, and a bilateral filter. The second approach we investigated is a semi-automatic one that allows a user to select regions within a data set. Similar regions are then retrieved by our algorithm using multiresolution histograms and the user can remove these regions from the rendering. By selecting and hiding regions containing noise or artifacts, the desired noise- and artifact-reduction can be achieved. We are going to show that both methods we investigated are suitable for removing noise and artifacts in EM data sets.

# Kurzfassung

Connectomics ist ein aufstrebender Fachbereich der Neuro-Wissenschaften, der danach strebt, die in den neuronalen Schaltungen des Gehirns versteckten Algorithmen zu verstehen. Das wird zu erreichen versucht, indem man die Neuronen im Gehirn verfolgt und deren Verbindungen untersucht. Von all den verfügbaren bildgebenden Verfahren bietet lediglich ein Elektronen Mikroskop die nötige Auflösung, um neuronale Prozesse abzubilden. Datensätze, die mit einem Elektronen-Mikroskop erstellt wurden, weisen jedoch sowohl einen schlechten Signal-Rausch-Abstand als auch Artefakte auf, die während der Vorbereitung und der Abtastung einer Probe entstehen. In dieser Arbeit stellen wir zwei Ansätze vor, den Signal-Rausch-Abstand zu verbessern und die Artefakte zu reduzieren. Der erste Ansatz, der entwickelt wurde, reduziert das Rauschen, indem die Datensätze mit einem 3D Filter geglättet werden. Das Besondere an diesem Ansatz ist, dass er interaktiv ist und die Datensätze „on-the-fly“ gefiltert werden. Zum Vergleich wurden ein Box-, ein Gauss- und ein bilateraler Filter implementiert. Der zweite Ansatz, den wir untersucht haben, arbeitet semi-automatisch. Er erlaubt es einem Benutzer, Regionen innerhalb eines Datensatzes zu selektieren und diese und ähnliche Regionen dann, mit Hilfe von Histogrammen der unterschiedlichen Auflösungsstufen dieser Regionen, auszublenden. Wenn nun Regionen selektiert und ausgeblendet werden, die Rauschen oder Artefakte beinhalten, wird die angestrebte Reduktion dieser beiden Störfaktoren erreicht. Wir werden zeigen, dass beide vorgestellten Methoden dazu geeignet sind, in, von einem Elektronen-Mikroskop erstellten, Datensätzen, den Signal-Rausch-Abstand zu erhöhen und die Artifakte beim Volumen Rendering zu reduzieren.

# Contents

# Chapter 1

# Introduction

*Every contrivance of man, every tool, every instrument, every utensil, every article designed for use, of each and every kind, evolved from a very simple beginning.*

*-Robert Collier*

Understanding the human brain has been a scientific goal for centuries. From healing diseases to inventing new types of artificial intelligence (AI), a complete understanding of the mechanics underlying the brain functions is crucial - and not yet accomplished. In almost all areas of biology, there is the need to find connections between biochemical functions and molecular structures [BD06]. Chalfie et al. [CSW+85] showed in their work that the need for structural connectivity information exists in systems biology as well. They reconstructed the nervous system of a *C. elegans* which allowed them to present a detailed description of the functionality of its nervous system. Having a complete connectivity map for a particular part of the nervous system of a more advanced specimen like a mouse for example so that its neural algorithms can be understood is however a daunting task [Mar02]. For a single mouse cortical column, the number of neurons will be $10^4$ larger than the number of neurons and neural connections in the C. elegans [BS98, Fia02]. The science investigating these neural connections is called *Connectomics* [STR05].

Connectomics is an emerging and active part of neurobiology [JBH+09] that aims to understand the neural algorithms embedded in the neural circuits by following the neurons and studying their connections. From all available scanning technologies

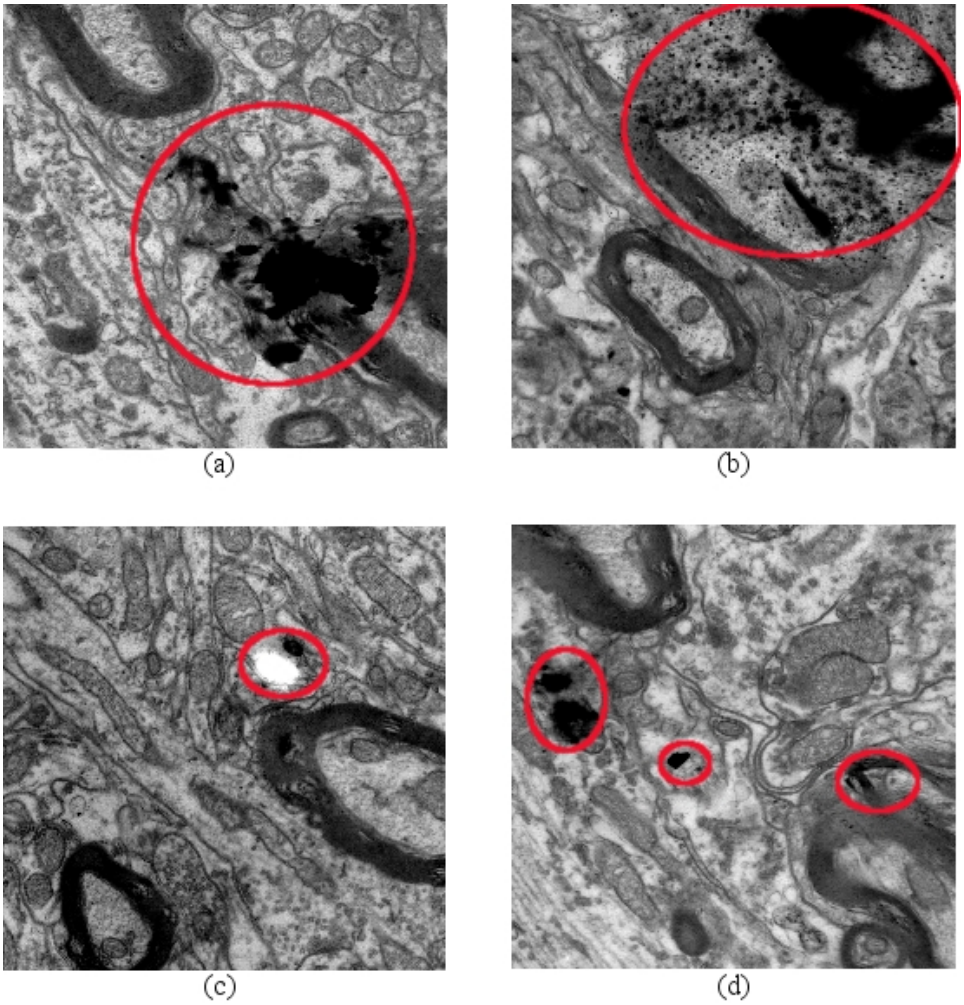


**Figure 1.1:** *Different examples of artifacts introduced to EM data sets during data set generation. (a) Shows an artifact with uniform shape while (b) shows a large uniform artifact with several smaller, blurry ones. (c) indicates that the artifacts not necessarily have to be black. The white spot in this image is an artifact as well. (d) shows three small artifacts spread across the image. The artifacts are marked by red ellipses.*

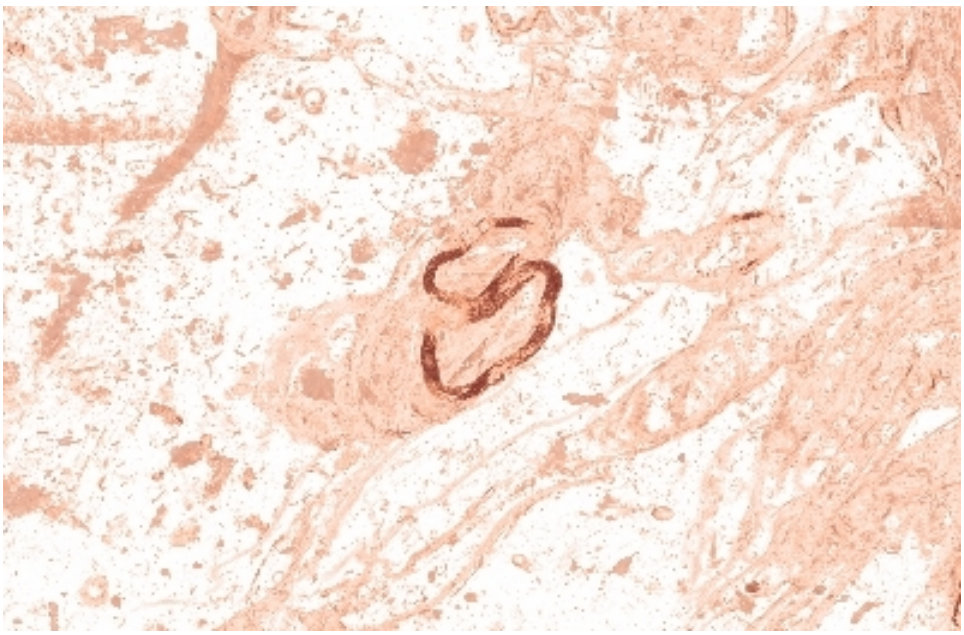

**Figure 1.2:** *Direct volume rendering of an EM data set using a 1D transfer function. The same intensities of the axons and the artifacts make it almost impossible for the user to distinguish between the structures by using just a 1D transfer function.*

only electron microscopy (EM) can provide sufficient scanning resolutions to identify neural processes [JBH+09]. Since resolutions up to 3*nm* can be achieved with recent EM technologies [JBH+09] the storage capacity necessary to accommodate a scan of a tissue sample of several nanometers in size can be up to several terabytes [JBH+09]. Despite the high resolution of EM scans, EM data sets suffer from poor signal to noise ratio [BD06]. Further, the process of combining EM scans to data sets may introduce unwanted artifacts (see Section 2.3.2). Figure 1.1 shows four different examples of artifacts introduced to EM data sets during the imaging and reconstruction stage. This figure shows that these artifacts may vary in size, shape and intensity.

Besides the computational issue of handling large neurological data sets, one must not forget the problems which users evaluating these data sets may experience. Manually searching these data sets for neural connections [JBH+09] and finding neural connections or distinguishing relevant from nonrelevant data within a Terabyte-sized, noisy scan is a tiresome task and can easily be compared to searching a needle not only in a haystack but more accurately in the whole barn. Although the EM data sets used for Connectomics are of three-dimensional nature and could

therefore be considered ideal for volume rendering [DCH88], the current practice of the researchers when they are trying to identify structures of interest is solely based on examining 2D slices of an EM data set. This may be due to the fact that the complex structure of nerve cells makes direct volume rendering (DVR) difficult, especially when using transfer functions based solely on image intensity and gradient [JBH+09]. DVR is a volume rendering technique that maps the intensity of each sample point of a volume data set to a specific color and opacity provided by the transfer function (see Section 2.1). Figure 1.2 shows a direct volume rendering of an EM data set using a 1D transfer function based on image intensity. The tube-shaped structures in that figure are the myelin sheaths of the axons, which have almost the same intensity as the noise and the artifacts introduced to the data set during data set generation. Thus, using this type of volume rendering for identifying structures is not yet feasible.

With this thesis we aim at improving the visual quality of direct volume rendering of EM data sets using 1D transfer functions based on image intensity. To accomplish this, we investigate two different approaches to reduce noise and artifacts in EM data sets. The first approach reduces image noise uniformly throughout the data set. The noise-reduction is accomplished using digital filtering. Digital filtering, in the context of digital images or digital volume data sets, is an image processing technique (see Section 2.2) that applies a 2D filter to an image that enhances or hides certain properties of the image. In our case, digital filters are used for smoothing an EM data set. Smoothing in this case means that the filters are used to reduce details within the data set. The amount of smoothing depends on the filter size. To perform filtering operations on volumetric data sets, we use three-dimensional filter kernels. The key element of our filtering approach is, that the data sets are filtered on the fly. This means that the filtering of the data set is repeated for each frame. On the fly filtering of the data set avoids the extra storage space needed to store prefiltered data sets. Besides the reduction of required storage space, on the fly filtering was chosen because it provides the flexibility to interactively select the type, the size and the dimensionality of the filter that is applied to the data set. This enables a user to interactively compare different filter types at different degrees of smoothing. The second approach we investigated is a semi-automatic one that allows the user to choose which structures of a data set he wants to see. After the user selects such a structure within the data set, multiresolution-histograms [HGN01a] are used to find similar structures which are then selected as well. The user choses whether

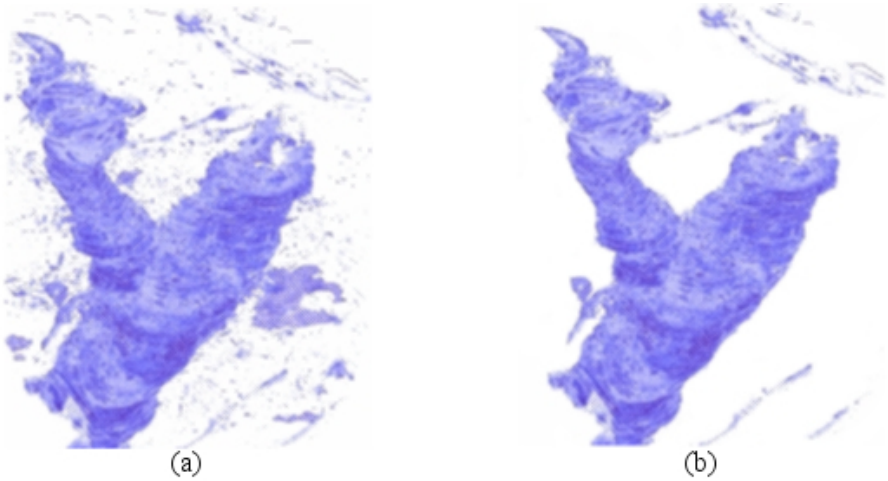


**Figure 1.3:** *Volume rendering of an EM data set without and with filtering. (a) original volume rendering. (b) Volume rendering of the data set after noise removal and artifact reduction.*

selected structures are visible and the rest of the data set is hidden or vice versa. The two approaches presented in this thesis have been integrated in an existing volume visualization framework called HVR.

## 1.1 Motivation

Connectomics is a scientific area with ambitious goals. If successful, the research done in Connectomics may be a great step towards understanding biological neural structures and eventually the human brain itself. This knowledge may lead to advancement not only in neurology and medicine but in creating artificial neural networks and new forms of AI as well and we appreciate the opportunity to not only be a part of these developments but to contribute in a useful manner using the knowledge of our own field of research. With our work, we want to facilitate the exploration of EM data sets. We want to reduce image noise and artificial artifacts within the EM data set so that the researchers can focus on exploring the parts of the data set they are interested in. Figure 1.3 (a) shows an unprocessed EM data set volume rendered with a 1D transfer function. The same as it is the case in Figure 1.2, the axons are surrounded by noise and artifacts. Image 1.3 (b) shows the desired outcome when rendering the same EM data set and using the same transfer function, but including a noise reduction step before rendering. We

hope that our work contributes to an increase of acceptance of volume rendering in Connectomics. We further hope that the researches may experience the advantages of directly studying the neural connections in 3D over manually segmenting the neural connections in each 2D slice of an EM data set.

## 1.2 Goals

Besides the ultimate longtime goal of proving the feasibility of volume rendering in Connectomics, this thesis focuses on improving the user experience when viewing direct volume renderings of EM data sets by increasing the visual quality of the renderings. The main goal hereby is to counteract the issues direct volume rendering encounters when being used on EM data sets. Namely these issues are the low signal to noise ratio, which renders the traditional use of DVR and 1D transfer functions virtually useless, and the large amount of, vastly nonessential, data that has to be explored in order to track neural connections throughout the volume.

Since evaluating the visual appearance of the renderings produced by our approaches goes beyond the scope of this thesis we focus our evaluation on the implementation issues of our methods. We will further show the noise reduction capabilities of our two approaches by providing respective screenshots for comparison.

## 1.3 Organization

The remainder of this thesis is organized as follows. Chapter 2 lays the technical foundation for this thesis. In Section 2.1 the concepts of direct volume rendering are explained. Section 2.3 shortly explains the theory behind electron microscopy, explains the two major types of electron microscopes and shows how EM data sets are formed. Section 2.2 gives a short introduction to image processing while Section 2.4 explains the reasons for using general purpose GPU computing and gives a short introduction to NVIDIA's general purpose GPU API called CUDA. Chapter 3 gives an overview over research related to volume rendering of EM data sets, filtering volume data on the GPU, and using histograms for finding structures in volumes or images. An introduction to the HVR_framework is given in Chapter 4 since both modules developed for this thesis were incorporated into this framework. Chapter 5

presents our filtering module and explains the theory behind the implemented filters. It further provides detailed information on the implementation of these filters. Chapter 6 presents our volume exploration module. Section 6.1 explains the theory behind the way similar structures are found. Section 6.2 explains the implementation of the volume exploration module from a conceptional point of view and gives insight on its user interface and usage. The results and the evaluation of the two approaches investigated in this thesis are provided in Chapter 7. A summary and a conclusion of this thesis as well as the outlook on future enhancements to this work are presented in Chapter 8.

# Chapter 2

# Fundamentals

*He who has not first laid his foundations may be able with great ability to lay them afterwards, but they will be laid with trouble to the architect and danger to the building.*

*-Niccolo Machiavelli*

This chapter gives an overview over the topics this thesis is associated with. We provide an overview and a short explanation of each topic to lay down the fundamentals to comprehend, observe and evaluate the techniques and results we present during the remainder of this thesis. Within the next sections we explain the basics of direct volume rendering and image processing, we shortly explain electron microscopy and the generation of EM data sets and we explain the principles of general purpose GPU computation using NVIDIA's CUDA as an example.

## 2.1 Volume Rendering

Volume rendering is a part of computer graphics. Computer graphics investigates the pictorial synthesis of computer based models, while image processing (see Section 2.2) on the other hand treats the reverse process [FVDFH95]. Volume rendering is accepted to be widely applicable for viewing medical data [LL94] and in computer-aided medical treatment visualization plays an important role [ZB09]. All techniques that deal with the visualization of volumetric data sets are considered

to be volume rendering. These volumetric data sets digitally represent a 3D-object by 3D sample points which usually are organized in regular grids [Max95] and stored as intensity values. As analogy to pixels in 2D imaging, the sample points in volumetric data sets are often referred to as „voxels“. Levoy was the first to present volume rendering as an alternative to constructing a mesh of polygons out of the sample points of a volume data set [Lev90]. He presented the idea of omitting intermediate geometric representations and obtaining an image by shading all sample points and projecting them onto the image plane. In accordance to the explanation of volume rendering given by Levoy, Max [Max95] defined DVR as follows:

> *Direct Volume Rendering refers to techniques which produce a projected image directly from the volume data, without intermediate constructs such as surface polygons.*

The idea of directly rendering volume data sets without intermediate polygon meshes is now known as direct volume rendering (DVR). Nowadays, the terms „volume rendering“ and „direct volume rendering“ are used interchangeably, since DVR has become the most popular volume rendering technique.

Implementation wise, DVR can be classified into three main types [EKE01]:

- **image based** DVR: The appearance of the volume is evaluated for every pixel of the image the volume is rendered to. The most prominent example for image based DVR is ray casting [RPSC99]. The basic idea for using ray casting for volume rendering is that for each pixel of the image plane, a ray is shot into the volume. Along each ray, the volume is traversed and the colors and opacities of the voxels that are hit by the ray are accumulated to obtain the final color for a pixel.
- **object based** DVR: In contrast to image based DVR, the voxels of a volume project their color onto the image plane. The whole volume has to be traversed to obtain the final pixel color. Examples for object based DVR are splatting [ZvBG01] techniques or shear-warp rendering [LL94].
- **texture based** DVR: A volume is represented by textured 2D-planes that are combined using hardware accelerated blending. The volume can either be

represented by three stacks of planes using 2D texture maps or one stack of planes using a 3D texture map. For the first approach, the planes of each stack are aligned to one coordinate axis while the planes in the second approach are view port aligned.

The common theme of all three DVR types is, that they all aim at evaluating the volume rendering integral [KVH84, MHB+00] for each pixel [EKE01]:

$$I_\lambda(x,r) = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt}ds \tag{2.1}$$

Equation 2.1 shows the formal description of the volume rendering integral where $I_\lambda$ is the amount of light or intensity of a light source that is received at location $x$ on the image plane. The variable $\lambda$ represents the wavelength of the light source. The direction of the ray of light that hits the image plane and that originates from the light source is represented by $r$. $L$ is the length of this ray. $C_\lambda$ is the light of wavelength $\lambda$ that is reflected and/or emitted at location $s$ in direction $r$. The densities of the volume's particles are given by $\mu$. These particles may reflect or emit light towards the observer. Since an analytic computation of Equation 2.1 is generally not possible [Max95], practical volume rendering algorithms use a discretized volume rendering integral as compositing equation:

$$I_\lambda(x,r) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\alpha(s_i)\cdot\prod_{j=0}^{i-1}(1-\alpha(s_j)) \tag{2.2}$$

Equation 2.2 shows the discretized version of the volume rendering integral. The colors $C(s_i)$ and opacities $\alpha(s_i)$ are calculated for each interval $i$. $\Delta s$ is the interval width and the opacity is given by $\alpha$. Functions that assign colors and opacities to intensity values are referred to as transfer functions.

The DVR type of choice on modern consumer graphics hardware is GPU based ray casting as it was originally proposed by Krüger and Westermann [KW03]. They proposed to implement a ray caster fully in the fragment shader of the GPU. This approach became feasible with the introduction of programmable vertex and fragment shaders and uses the distinct advantages a GPU has over a CPU when it comes to graphical tasks. These advantages are a massively parallel

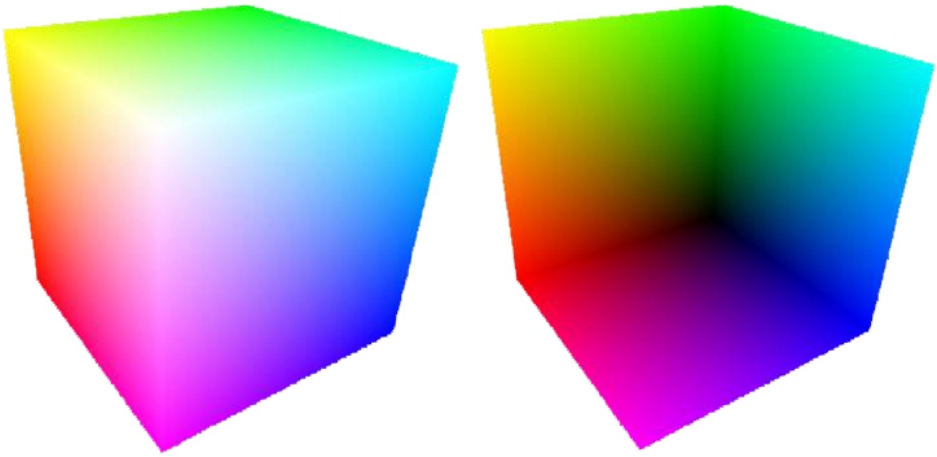

**Figure 2.1:** *This figure [Sch05] shows the front and back faces of the bounding box rendered to textures. The texture coordinates of each point are encoded in the color channel of the respective texture. Subtracting these textures yields the viewing vectors for the ray casting step.*

architecture, fast memory access and fast vector operations. The GPU ray casting algorithm proposed by Krüger and Westermann will be explained in more detail since it is the basis for the ray casting algorithm that is used in this thesis to render the EM data sets. First of all, the volume data set is stored in a 3D texture map to take advantage of the graphics card's built-in trilinear filtering capability. At any point within the volume and for any desired resolution, trilinear interpolated intensity values are provided. Then a bounding box is created around the volume. In the first rendering pass, the front faces of this bounding box are rendered to a 2D RGB texture. In the vertex shader, the *xyz* coordinates of the vertices are stored in the RGB values of the texture. This results in a 2D texture map that has the same resolution as the view port and contains the colored rendering of the bounding box. Every colored pixel in the 2D texture map represents an entry point for a ray that is cast into the volume. The coordinates of the entry points of the rays are therefore encoded in the color channel of this 2D RGB texture. The next step in the ray casting algorithm is to retrieve the directions of the rays. This is done by rendering the back faces of the bounding box to a 2D RGBA texture, in contrast to the 2D RGB texture of the previous step. In the vertex shader, the *xyz* coordinates of the vertices are then subtracted from the RGB values of the texture map obtained in the previous step. The resulting vectors are stored in the RGB values of the RGBA texture. The length of each vector is stored in the respective $\alpha$ channel. In total we have two texture maps that are needed for rendering: The texture map of the

entry points and the texture map of the direction vectors. Figure 2.1 shows the rendered front and back faces of the bounding box of the volume that is rendered. The original colors of the vertices have been replaced with their texture coordinates. When these two textures are subtracted, the viewing vectors for the ray casting step are obtained. The entry-point texture map and the direction texture are congruent and provide a starting point and a direction for every fragment to be rendered. The actual ray casting step is performed by another fragment shader in a third render pass. Here, for each fragment a ray is cast from the respective starting point along the direction vector. In a predefined step size, the intensities of the 3D texture are retrieved and accumulated using a transfer function. The ray is terminated when its length equals the length of the stored direction vector or the accumulated opacity of the ray is larger than a given threshold. The termination of such a ray because its opacity exceeds a given threshold is called „early ray termination“. To increase performance and to remove sampling artifacts, various techniques like empty space skipping or hitpoint refinement can be used [Sch05].

## 2.2 Image Processing

The scope of computer imaging is as vast as the scope of computer graphics. While computer graphics investigates the pictorial synthesis of computer based models, computer imaging deals with obtaining information or modifies the visual appearance of already created images. Image processing is an area of computer imaging where a human being is involved in the visual loop [Umb05]. Although the images are processed by digital computers, the processing is steered and examined by people. Digital image processing therefore concerns the transformations of an image to a digital format and its processing by digital computers [Pit00]. The major topics in image processing are the restoration and the enhancement of digital images. Image restoration techniques include the reduction of image noise and image distortions that occur due to misalignment of the optical systems of cameras. Enhancement techniques aim at improving the images visually, e.g. by the use of contrast stretching [Umb05] or sharpening [Umb05]. In the filtering framework developed for this thesis we use image restoration techniques to remove the noise in EM data sets. The operations that are used for reconstruction or restoration purposes either aim at an image's topography or the statistics of the distribution of image intensities. The use of digital filters exploit an image's topography, thus sharpening

is a proper example for this kind of operations. Contrast stretching is an example for the use of statistical information on image intensities because it depends on the use of intensity histograms. These intensity histograms store the distribution of the image intensities.

### 2.2.1 Noise Reduction

Noise is defined as brightness variations in regions that are ideally uniform [Rus06] and results from misalignment or defects of the optical systems of the imaging devices. The amount of noise within an image can be expressed by the signal-to-noise ratio. The signal-to-noise ratio is the ratio between two contrasts. The first one is the contrast of an image that is solely due to structural differences in the image. The second one is the contrast that is due to the noise level in the image. The smaller the signal-to-noise ratio, the harder it gets to distinguish between signal and noise. Thus detecting image structures gets harder the smaller the signal-to-noise ratio is. In the context of this thesis, we examine the kind of noise that is introduced to EM data sets. This noise is introduced during the preparation of a specimen for the EM scan. For a detailed description of electron microscopy see Section 2.3. Noise is introduced to EM data sets due to variations in the field emission current at the tip of the electron microscope. These variations occur because the electron emissions from the tip can shift from one atom of the tip to another. This shift of the electron beam produces a change of several percent in electron beam current [GNE+03]. This change in electron beam current results in a change of intensities in the emissions that are reflected from the scanned specimen on to the detectors of the electron microscope. Different intensities of the emissions result in different intensities in the EM data set. No matter where the noise comes from, the basic assumption to noise reduction is, that pixels are smaller than important details of an image and that for most pixels, their neighbors represent the same structure [Rus06]. By averaging these neighbors, the influence of outliers to an image is reduced. This assumption is the basis of digital filters.

### 2.2.2 Histograms

A useful initial characterization of an image is often based on statistical information on the intensity distribution of the image's pixels [SOS00]. Intensity histograms

provide the means to efficiently store and evaluate the intensity distributions of an image and offer a compact representation of the global intensity distribution of an image. An intensity histogram is constructed by examining the intensity values of each pixel of a given image and counting the number of pixels for each possible intensity value. Intensity histograms usually are displayed as a plot that shows the number of image pixels for each of these possible intensity values [MS00]. On the histogram plot, each intensity value is represented by a so called histogram bin. The height of the bin represents the number of pixels that have the same intensity. By choosing a greater width of a bin, intensity values can be summarized to one bin. The most basic initial characterizations one can obtain from intensity histograms are about the image brightness and image contrast. The characterization of an image's brightness can be obtained as the result from averaging the possible intensity values weighted with the height of the respective histogram bin. In a range from $[0, 255]$ intensity values, an average of 0 represents a uniformly black, or in terms of brightness, a uniformly dark image. An average of 255 on the other hand represents a fully white or bright image. The contrast of an image is expressed by the range of intensities the histogram bins cover in a histogram plot. This range is called the dynamic range and is an indicator for the contrast of an image. The greater this range is, the higher is the image's contrast. Besides information on average image brightness and image contrast, there are other useful properties of intensity histograms. The location of the majority of occupied bins for example offers insight on the illumination of an image. When the majority of the occupied bins is concentrated on the lower end of the dynamic range, it is an indicator that the image is too dark while a concentration on the other end of the dynamic range is an indicator that it is too bright. The most commonly examined histogram feature are peaks in the bin distribution of the histogram [SOS00]. A peak is a concentration of bins that exceeds the base line of the other bins. In the context of this thesis examining the peaks of the intensity histograms of images from EM scans of neural tissue, strong peaks in the lower end of the dynamic range may indicate either the presence of an axon or a strong artifact introduced by the EM.

In image processing, operations that aim at altering the appearance of an image based on a desired change in the intensity histogram are called histogram transformations. Histogram transformations are operators that generate a new output histogram by modifying the profile of the input histogram. The intensity values of

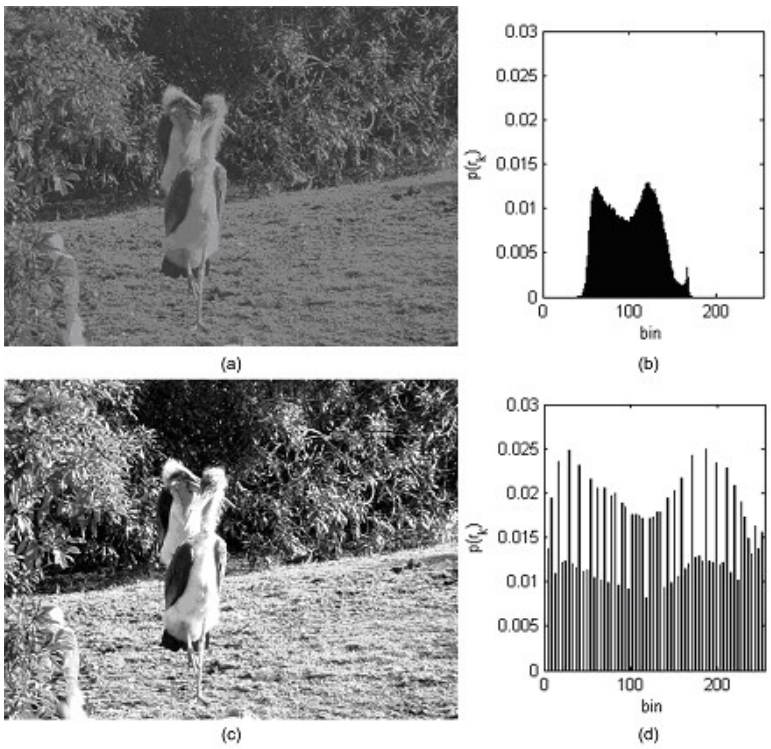


**Figure 2.2:** *Histogram equalization used for contrast enhancement. a) Original Image. b) Histogram of the original image. c) Image after histogram equalization. d) Equalized histogram. Image from [Qur05]*

the image are changed so that the desired output histogram is matched. Examples for histogram transformations are histogram expansion [Hum77] and histogram equalization [KLK+97]. Figure 2.2 shows the usage of histogram equalization for contrast enhancement on an image. Image 2.2 a) is an image of low contrast and is used as input image for the histogram equalization. Since in low-contrast images the concentration of the bins in a relatively small part of the dynamic range of the image can be observed, the low contrast can be seen as well in its histogram as it is shown in Image 2.2 b). After histogram equalization, the bins of the new histogram shown in Image 2.2 d) are spread over the whole dynamic range of the image. The higher contrast of the image after histogram equalization is shown in Image 2.2 c). While histogram transformations can be used to improve the visual quality of images, in this thesis we exploit the statistical informations provided by intensity histograms (see Section 6.1) to achieve noise reduction and an improvement of the visual quality of our renderings.

## 2.3 Electron Microscopy

Depending on eyesight and illumination, the smallest distance that can be distinguished by man lies between 0.1 and 0.2*mm* [WC09]. This distance is, so to speak, the resolution of the human vision. All techniques that provide the means to observe smaller distances can be comprehended as microscopy. The maximal magnification of a common visible light microscope (VLM) is given by $\delta = 0.61\lambda$ where $\delta$ is the maximal magnification and $\lambda$ the wavelength of the used light source. The magnification of the VLM therefore directly depends on the wavelength of the light source. Generally formulated, when using radiation for magnification, the degree of magnification depends on the wavelength of the source-radiation. The idea to use electrons to magnify the smallest of structures comes from the understanding that the wavelength of electrons are even smaller than atoms and that these wavelengths depend on the energy of the electrons [WC09]. Theoretically, resolutions as small as the wavelength of an electron are possible. Although this kind of resolution is not yet reached, electron microscopy is the only current available technique which provides enough resolution to follow axons and dendrites [BD06]. This is why they are used to produce the data sets used in Connectomics and our own research.

### 2.3.1 Basic Types of Electron Microscopes

The data sets we work on in this thesis were generated by a scanning electron microscope (SEM). The resolution of a SEM lies in the *nm*-$\mu m$ scale while its magnification range reaches from 10 to 10.000 times [GNE+03]. When a specimen is to be examined by a SEM, the area containing this specimen is irradiated with a finely focused electron beam. This beam either sweeps the specimen in a regular grid across its surface so that an image can be produced, or it remains static over a certain position to analyze the specimen at that position. In contrast to the transmission electron microscope, that is described later on, the electron ray of a SEM does not carry the whole image of a specimen but rasterizes it. The signals emitted by a SEM are secondary and backscattered electrons, x-rays and other photons of various energies. The secondary and backscattered electrons are the most interesting signals for imaging since their variance mainly depends on the specimens topography [GNE+03]. When the secondary electron emission is confined to a very small

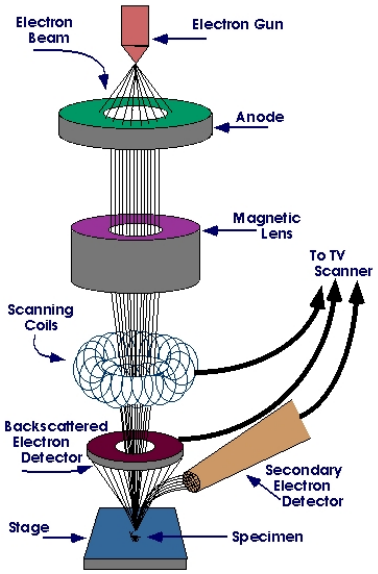


**Figure 2.3:** *A schematic view of a SEM can be seen in this figure. Diagram courtesy of Iowa State University SEM Homepage (http://www.mse.iastate.edu/microscopy/path2.html)*

volume near the beam, the impact area permits obtainable resolutions of nearly the size of the focused electron beam.

The basic components of a SEM are the lens system, the electron gun, the visual system and the associated electronics. When a beam is emitted by the electron gun it is focused by the lens system. The interaction of this primary electron beam with the specimen results in the scattering of this primary electron beam into the different signals that then can be measured by specialized detectors. Figure 2.3 shows a schematic setup of a SEM. Although the SEM has a poorer resolution than the transmission electron microscope it allows a larger specimen size. Transmission electron microscopy (TEM) was the first successful attempt to use electron beams for magnifications far beyond the capabilities of VLMs [WC09]. The components of the TEM are mostly the same as a SEM. The main difference between these two electron microscopes (EMs) lies in the way the images are generated. While the SEM samples a specimen at a regular grid, the high voltage beam of the TEM carries the whole image of the specimen. A TEM, similar to a SEM, consists of an illumination system, an objective lens stage and an imaging system. The objective lens stage and the specimen holder are the heart of a TEM because that is where all the interaction between electrons and the specimen take place. This region of the TEM is about 10*mm* in size which also represents a hard limit for specimen sizes.

### 2.3.2 3D Data Set Generation

In contrast to other imaging technologies like CT or MRI, which are able to non destructively obtain volume data from a specimen, electron microscopy can only obtain images of surfaces that can influence the electron beam. To obtain a volumetric data set from a specimen, this specimen has to be sectioned. Sectioning is the process of dividing a specimen into layers which are scanned separately and ultimately leads to the destruction of the specimen. To obtain volumetric data of organic tissue, thick blocks (several $100\mu m$) of tissue are frozen so they are not distorted during the sectioning process [BD06]. A diamond blade then cuts away thin sections from the frozen tissue which are approximately $50nm$ thick. These sections are then transfered and positioned on a grid as preparation for imaging with the EM. The thickness of these sections is responsible for the fact that the depth resolution in EM data sets is lower than the horizontal and vertical resolution. Besides the disadvantage of a low depth resolution, sectioning is tedious and prone to error [BD06]. The most prominent problems with sectioning are the possible loss of sections and the uneven section thickness mentioned above. These problems lead to image distortions and uneven illumination of the sections. The transfer of the sections from the cutting to the imaging stage may damage the sections and introduce debris which lead to the noise and artifacts we aim at removing in our work. These problems make automated alignment of the sections and therefore the digital reconstruction of the specimen difficult. To get the final data set, the single images have to be registered to each other. In order to register two images, a transformation must be found to correlate a point in the first image to one point in the second one [Bro92, MV98]. The main problems when registering two images are noise and geometric distortions, which are both present in EM data sets. Inconsistencies between two images that are supposed to be aligned are called misalignments. When the type of misalignment is known, it is the task of the registration algorithm to counteract these. When type and degree of misalignment is not known, like it is the case in EM data sets, the primary approach to register the images is called point mapping. This method consists of three stages. First, image features are calculated in one of the images that are to be registered. The second stage computes image features in the other images and determines spacial mapping functions according to these features. The third stage uses these functions to chose how the images are aligned.

## 2.4 GPGPU

Harvesting the computational powers of graphics processing units (GPUs) for general purpose computations like physically-based simulations or database queries is called general purpose GPU (GPGPU) computation. The reason to use GPUs for general purpose computations as well as an overview over NVIDIA's CUDA, a GPGPU API, is given in the following.

### 2.4.1 Why GPGPU

The main feature of GPUs that is exploited in GPGPU computations is their parallel processing capability. GPUs are designed for high performance parallel processing because graphical tasks, especially evaluating vertices or pixels, are highly parallelizable tasks, which GPUs are designed to handle. In recent years GPUs evolved into extremely flexible and powerful parallel processors [Har05]. The introduction of programmable fragment and vertex shaders and the recent introduction of dedicated GPGPU APIs and their support of high level programming languages facilitate the interaction with the GPUs. Precision wise, 32 bit floating point operations throughout the pipeline are common at present. The performance of recent GPUs is demonstrated by the specifications of recent consumer graphics cards. According to NVIDIA the GeForce 295 GTX can reach up to 596 GFLOPS/sec with a peak memory bandwidth of 223,7 GB/sec. The annual performance increase of GPUs and graphics cards in general exceeds Moore's law because the innovation in graphics hardware is driven by the multi-billion dollar video game market [OLG+07].

### 2.4.2 CUDA

Before GPGPU APIs were available, general purpose algorithms for GPUs had to be expressed as graphics primitives, textures and triangles using a graphics API. The first step towards programmable GPGPU APIs and towards NVIDIA's CUDA was set in 2004 by introducing a stream programming model that aims at moving away from graphics APIs by using a cross compiler to compile extended C code to a shading language [BFH+04].

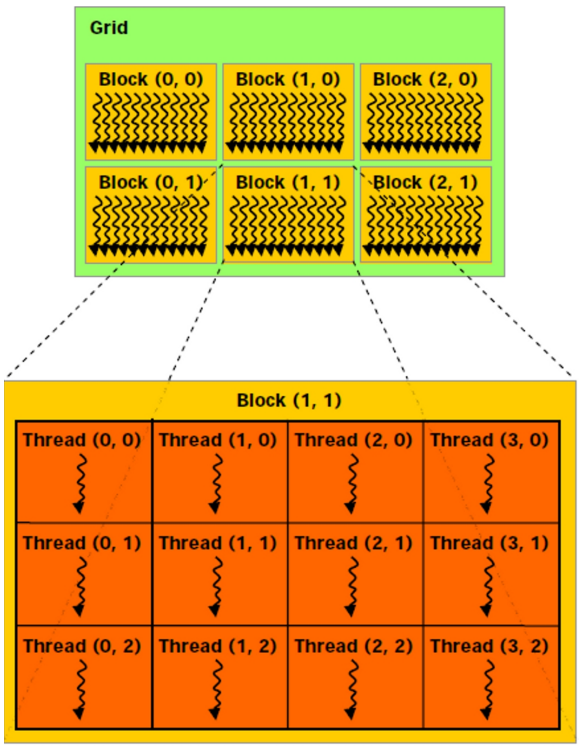


**Figure 2.4:** *Architecture of a CUDA kernel from [NVI10a]. One CUDA kernel runs one CUDA grid which contains multiple CUDA blocks. These CUDA blocks consist of CUDA threads which are executed in parallel.*

Instead of using a compiler to compile C code to a shader language, the software part of CUDA (short for Compute Unified Device Architecture) is co-designed with the actual graphics hardware [NVI10b]. Syntax wise, CUDA programs are written in a version of C that is enhanced with a small set of extensions which grant access to the GPU and the memory of a graphics card. The CUDA programming model distinguishes between CPU (host) and GPU (device) code. Data exchange works in both directions and is not limited to graphical data structures. Arrays, single variables or structures may be exchanged. The parallel portion of an application is handled on the device by a CUDA kernel. Only one CUDA kernel is executed at a time but many CUDA threads are executed in each of these kernels in parallel. Each of these threads runs the same code. To allow cooperation between the threads, the kernels are divided into logical structures called CUDA blocks. Each block contains an equal amount of threads and allows these threads to exchange data using a fast shared memory. The blocks can only exchange data among themselves using the slow global memory. Further, these blocks are organized in one CUDA grid per

kernel. Within hardware specific limitations, block and grid size are arbitrary. It is however recommended [NVI10a] that the block and grid size are chosen so that there are at least as many blocks as there are multiprocessors on the device. Grids may be organized as 1D or 2D arrays of blocks while the blocks can be organized in 1D to 3D arrays of threads. Each thread has access to its location within the block, the size of the block as well as the location of the block within the grid. With this information one can determine the location of a thread within a kernel. Figure 2.4 shows the interrelationships of threads, blocks and grids. Besides proper planing of a parallel algorithm, efficient usage of the memory bandwidth of the different memory types of CUDA is essential to the overall performance of a kernel [NVI10a]. Besides a small register for each thread, the fastest memory in the CUDA architecture is the shared memory that is available to all threads within a block. The important parts of designing a parallel algorithm in CUDA are to limit interaction with slow memory types like global memory and to avoid bank conflicts in shared memory. These bank conflicts can drastically reduce the performance of the shared memory. Bank conflicts occur when two different threads of the same block try to access the same memory bank of their assigned shared memory. These bank conflicts may double the execution time of a kernel. Figure 2.5 shows the different types of memories in CUDA and their scope. Per-thread local memory is the fastest one and takes zero clock cycles per instruction [NVI10a] but is only accessible for one thread. Shared memory is faster than global memory but can only be accessed within one block. Global memory is the slowest memory and can be accessed GPU wide.

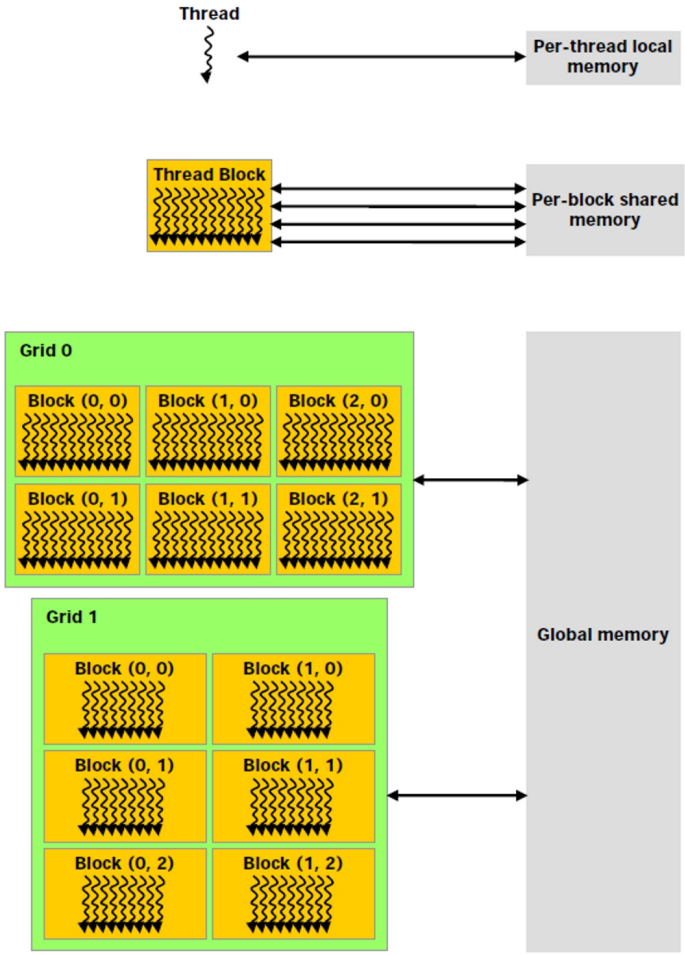


**Figure 2.5:** *Different memory types and their designated use in the CUDA architecture [NVI10a]. Per-thread local memory is the fastest but smallest memory type. Per-block shared memory is accessed as fast as per-thread local memory. Successive CUDA kernels can access the same global memory. Global memory is the slowest memory type.*

# Chapter 3

# Related Work

*So, let us not be blind to our differences - but let us also direct attention to our common interests and to the means by which those differences can be resolved.*

*-John F. Kennedy*

In this chapter, we present and introduce the research topics this thesis is built upon. Although the main goal of our work is to find the means to reduce noise and artificial artifacts in EM data-sets on-the-fly, our work falls into the domain of multiple research areas. This is mainly due to the fact that we decided to evaluate and compare two different approaches to that goal, namely filtering and multiresolution-histogram matching. To our knowledge, there has been no attempt to use multiresolution-histograms to find or eliminate certain structures of EM data-sets or any other kind of volumetric data-sets. Besides presenting research on the topic of noise reduction in volumetric data-sets, we further present research on the topic of texture and structure matching since this is the research area multiresolution-histogram matching origins from. We further present work related to DVR of electron microscopy or neurological data and the usage of general purpose GPU computing for volume rendering.

## 3.1 Noise Reduction in Volumetric Data Sets

Jeong et al. [JBH+09] were the first to propose using image-processing filtering-techniques to reduce the noise in EM data-sets for improving the quality of volume rendering. They further suggested a CUDA implementation of those filtering techniques to enable on-the-fly filtering of the EM data-set. The main goal of their work was to use edge detection prior to a semi automatic segmentation method to segment neurons in EM data-sets of neural tissue. They use the filtering techniques to reduce the noise in the EM data-set and thus improve the results of their edge detection method. The ideas presented in their work were the basis for the development of our filtering framework. Like EM data-sets, data-sets retrieved by electron tomography suffer from very bad signal to noise ratio [FH01]. This bad signal to noise ratio results from issues with the preparation process of a specimen. To achieve a combination of efficient noise reduction and excellent signal preservation, Frangakis et al. propose an approach based on non-linear anisotropic diffusion [FH99]. Their use of neural networks and the required preparation, training and preprocessing time that comes along with that makes their approach not suitable for achieving our goals. In contrast to their proposal, our filtering framework provides effective noise reduction that is simple to use and does not require any preprocessing time.

## 3.2 Texture and Structure Matching

With their work on multiresolution-histograms Hadjidemetriou et al. laid the foundation for our usage of multiresolution-histograms. Instead of finding noise or other structures in a volumetric data-set, they aimed at retrieving similar textures from a database of Brodatz textures [HGN04]. The main idea behind their work is that the combination of histograms of different resolution levels of the original image can encode spacial information and therefore is superior for matching in comparison to regular histograms. For a more detailed explanation of their proposal see Section 6.1. Carlotto et al. [Car84] propose a texture matching technique where the texture is measured by the local histogram computed in a window that slides over a given image. For each shift of this window, the local histograms are updated and compared to a precomputed sample distribution of textures of interest. They suggest to use the

comparison of local histograms on top of other image measuring techniques like gradient magnitude or gradient level. However, they consider local histograms of a gray scale image not suited as sole measuring technique because these local histograms capture gray level variations but do not represent textural properties. Although they have achieved a classification success rate of 96%, their approach can not be applied to our problem. This is due to the fact that the computational costs of shifting a window accross a volume, as well as continously calculating and updating the local histograms, render this approach useless when interactive frame rates are desired. In their work on texture classification, Ojala et al. [OPM02] use uniform patterns as an operator to detect micro structures in images like edges or lines. They enhance this operator by using discrete occurrence histograms of these uniform patterns. They use these occurrence histograms together with the uniform patterns and show that these are powerful texture features. This is due to the fact that occurrence histograms together with the uniform patterns can estimate the underlying distribution of the micro structures they want to detect. Like Hadjidemetriou et al. they use different spacial resolutions to increase the accuracy of their method. Instead of using different resolutions of the original image however, they use different resolutions of the uniform patterns. A different take on histograms is proposed by Liu and Wang. They introduce a new kind of histogram, namely a spectral histogram as a feature statistic for texture classification [LW03]. These spectral histograms consist of the responses obtained by a variety of filters. They encode the local structure as well as the global appearance of an image. To yield better optimization performance, they provide a selection algorithm to chose the filter types that are best suited for the respective texture they want to match. The number of filters they use depends on the texture and therefore may differ from texture to texture. Although the quality of their presented results seems suitable as alternative to our multiresolution-histogram matching, the differing number of used filters and the thereby differing execution times prohibit Lius and Wang's approach from being used for our purposes because it is not interactive. A survey of other texture classification techniques using image processing and filtering techniques is given by Randen and Husoy [RH99].

## 3.3 GPGPU Ray Casting

Although numerous work has been done on GPU implementations of ray casting algorithms for volume rendering [KW03, Sch05, RV06], there is little to no literature on how to translate these ray casting algorithms from shader languages like GLSL to GPGPU API's like CUDA. In our research we did not find hard evidence if this translation is a straight forward one by simply treating CUDA-threads like fragments. However, we have found research done on a CUDA implementation of a ray caster that is used to render unstructured grids. In [MRB+08], Maximo et al. propose a CUDA implementation of the VF-Ray algorithm proposed as CPU implementation in [RMB+07]. Except the face reduction and projection step, the ray casting algorithm itself works similar to that introduced by Krüger and Westermann [KW03] for the DVR of structured grids. The only difference between these two ray casters is, that VF-Ray intersects and retrieves the values from the visible faces while the ray that is cast in Krügers ray caster calculates the values from the voxels of the volume. The fact that the VF-Ray method assigns one CUDA-thread to each pixel that needs to be rendered as well as the performance of the VF-Ray presented in [MRB+08] suggest that a translation of Krügers algorithm to CUDA can be done by assigning one CUDA-thread to each pixel that needs to be rendered. Our own implementation of ray-tracing in CUDA showed an unexpected outcome. Translating the code from our GLSL implementation to CUDA almost halved its performance.

# Chapter 4

# Overview over the used Volume Rendering Framework

*There is only one way in which a person acquires a new idea; by combination or association of two or more ideas he already has into a new juxtaposition in such a manner as to discover a relationship among them of which he was not previously aware.*

*-Francis A. Carter*

The „HVR framework“ developed by Hadwiger at the VRVIS Vienna [Had04] is written in C++ and is especially designed for volume rendering and volume visualization. Its main advantage lies in its expandability allowing a variety of implementations for different volume-rendering techniques as well as different shading languages like GLSL or CG. Examples for supported volume rendering techniques are object-aligned slices [EKE01], axis-aligned slices [EEH+00] as well as GPU-based ray casting using 3D textures instead of slices. GPU-based ray casting was made possible by the invention of programmable shaders and their support of 3D textures. Thus the HVR framework supports implementations of GPU-based ray casting using either GLSL, CG or, most recently, NVIDIA's CUDA parallel computing architecture (see Section 2.4.2). According to Fung and Mann [FM08] image processing techniques can be significantly sped up by using the parallel computation capabilities of modern graphics-card. This speedup is necessary in our goal to provide on-the-fly noise reduction on EM data sets. Because of this fact

and the ease with which data can be exchanged between CPU and GPU when using CUDA , the CUDA based implementation of a ray caster (see Subsection 4.2) was chosen as a basis for the noise reduction step (see Section 5). In order to explain the various changes made to the HVR framework during the work on this thesis, we give a short overview over the key elements of the HVR framework. The main element is an interface that provides access to all the other components. The texture manager manages the volume data sets CPU wise, is responsible for creating the textures on the GPU and handles the caching of the CPU volume data for the GPU. This caching is necessary in order to display volumes that otherwise would not fit into the graphics-card memory (see Subsection 4.1). Any implementation of the ray casing-class contains a GPU based ray caster that may be individually enhanced by additional functionality, like „alpha-blending“ [WVW94]“, „MIP [JHK+99] (*maximum intensity projection*)“ or, in our case, image processing techniques. The further is a class that is responsible for exchanging data and instructions between any given GUI and the ray caster.

## 4.1 Caching Large Datasets

In contrast to the performance enhancement techniques presented in Section 2.1, like early ray termination or empty space skipping, caching does not aim at optimizing rendering performance. It aims at ensuring that even large volume data sets, which otherwise would not fit into the graphics-card memory, can be displayed. The implemented caching algorithm is based on the algorithm proposed by Beyer et al. [BHMF08]. Since the functionality of the texture manager and its caching technique have been vastly used and slightly altered in the course of this thesis (see Section 6.2.1), they are going to be explained more thoroughly. In order to be cached, the volume is subdivided into equally sized subvolumes that are called „tiles“. The cache itself is a 3D texture that is dimensioned so that it can hold a predefined number of these tiles. For fast texture filtering when stored in the cache, border voxels are attached to the tiles. Therefore, the subvolumes that are stored in the cache are bigger than the original tiles and are called „blocks“. The cache therefore is called „block-cache“. For each render pass a list of active blocks is determined by culling the original volume against a transfer function or clipping planes. The blocks contained in that list are stored in the block-cache. Due to viewport changes and only when viewport culling is enabled, blocks that were

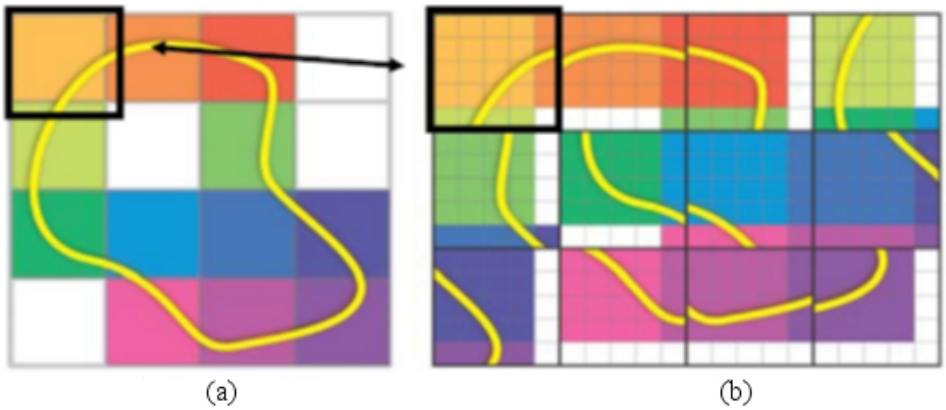


**Figure 4.1:** *Storage of tiles of a volume in a block-cache. a) Original location of the tiles in the volume. b) Random location of the tiles in the block-cache. For fast texture-filtering, border voxels are attached to the tiles before they are stored. Image courtesy of the VRVis Research Center*

culled away previously may become visible and vice versa. When this occurs, invisible blocks are removed from the cache while newly visible ones are inserted. Thus, blocks adjacent to each other in block-cache space may not be adjacent in volume space. Block-cache space and volume space are both local coordinate spaces in the range of $[0,1]$. A small 3D address-translation texture is used to translate volume-space coordinates into block-cache space-coordinates.

$$x'_{x,y,z} = \frac{x_{x,y,z} \cdot vsize_{x,y,z} + t_{x,y,z}}{csize_{x,y,z}} \tag{4.1}$$

$$t_{x,y,z} = b'_{x,y,z} \cdot bres'_{x,y,z} - b_{x,y,z} \cdot bres_{x,y,z} \tag{4.2}$$

Equation 4.1 shows the retrieval of texture space block-cache coordinates $x'_{x,y,z}$ from the volume-space coordinates $x_{x,y,z}$. $vsize_{x,y,z}$ is the volume size in voxels and $csize_{x,y,z}$ is the cache size in voxels. Dividing by $csize_{x,y,z}$ brings the result of $x_{x,y,z} \cdot vsize_{x,y,z} + t_{x,y,z}$ from block-cache space to block-cache texture space. $t_{x,y,z}$ stores the offset from $x_{x,y,z}$ to the block-cache coordinates $x'_{x,y,z}$. The offset $t_{x,y,z}$ is stored as RGB tuple in the address-translation texture and is calculated according to equation 4.2. $b'_{x,y,z}$ is the block-position in the cache while $bres'_{x,y,z}$ represents the block-size. $b_{x,y,z}$ and $bres_{x,y,z}$ respectively are the tile position in the volume and the tile size.

The texture manager administers both, the block-cache texture as well as the 3D address-translation texture. When blocks have to be exchanged in the block-cache, blocks not needed are removed and replaced by new blocks. The address offsets in the address-translation texture then have to be updated as well. Figure 4.1 shows an illustration of tiles of a volume being stored at random locations in a block cache. For the sake of simplicity, we only show a 2D representation of the volume and the block cache.

## 4.2 CUDA Ray Caster of the HVR Framework

The CUDA ray caster can be used as any other ray caster within the framework and contains all the render modes that use CUDA. A render mode is an option for a ray caster that influences the outcome of the rendering. DVR and MIP for example can be options for the same implementation of a GPU ray caster. Each render mode may implement its own ray-casting technique or share it with other render modes. The concept of a GPU ray caster in CUDA is similar to the one described in 2.1, using 2D textures for the starting positions of the rays as well as their direction. The output is written to another buffer called the „output buffer“.

Implementing the ray-casting step in CUDA is similar to an implementation in a fragment shader. For each CUDA thread - similar to a fragment in a fragment shader - a ray is cast into the volume. Beginning at the starting position given by its location from the texture of entry points, the ray is cast along the direction given by its location from the texture containing the directions. The ray may be terminated by either exiting the volume, reaching a voxel which meets a given abort criterion or reaching a predefined opacity-threshold when integrating the voxels' opacity along the ray. The opacity and color of a voxel are set by the texture containing the transfer function. When the ray is terminated, the combined color and opacity is written to the output buffer. Casting the rays for each texel in the output buffer is done in parallel. That means that all rays are cast simultaneously.

Before actually casting the rays, the CUDA grid size as well as the CUDA block size have to be defined. The grid size is given by the screen dimensions divided by the dimensions of the blocks where all blocks are equally sized. For example having a screen size of $512x512$, and wanting a block size of $8x8$, the grid size would be

64$x$64. Both, grid size and block size are two dimensional values. The block size is chosen to maximize parallelization by balancing the workload evenly between all the blocks within the grid. When the grid and block sizes are determined, all textures and arrays needed are bound to textures. Binding the textures as well as determining the block and grid sizes sizes is done on CPU. When the CUDA kernel containing the ray casting step is started, all textures and values assigned to CUDA are accessible to the kernel.

# Chapter 5

# Filtering Module - Noise Reduction by Filtering

*I was like a boy playing on the sea-shore, and diverting myself now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.*
*-Isaac Newton*

The first technique we investigated in our search for the means to reduce the noise and artifacts in EM data sets is called filtering and will be described in this chapter. We developed a filtering module as enhancement for the HVR framework that allows on-the-fly noise reduction in volumetric data sets of any kind. Although the filtering step for every frame is done prior to the DVR of the data set, filtering and DVR together still achieve interactive frame rates. This is achieved by using a caching algorithm described in Section 5.4 that makes sure that only parts of the volume have to be filtered that firstly are visible and secondly have not been filtered before. Generally spoken, filtering is to selectively choose specific frequencies from all the frequencies available in a given signal. In this work, the expression filtering is used in terms of smoothing a signal by removing high-frequency fluctuations. These high-frequency fluctuations are called noise. Removing this noise results in a smoother, less detailed signal. Digital 2D-images or digital 3D-volumes are discrete signals in the spacial domain. The filters described in this work are discrete filters applied to the image in the spacial domain. Filters applied to the frequency

domain are not examined in this work. To filter a discrete signal in discrete spacial domain one usually iterates over all values of the signal which are then replaced by the combined values of their neighborhood. The way these values are weighted and combined depends on the filter type while the size and shape of their neighborhood is defined by the filter kernel. The filter kernel can be understood to be a matrix that is overlaid on every voxel of a volume. The kernel contains multiplication factors that are applied to the voxel and its neighbors. When all values have been multiplied, the value of the voxel, the filter is applied to, is replaced by the sum of the products. Since filter kernels can further be seen as the translation of continuous filter functions to the spatial domain, different filter kernels represent different functions and thus lead to different results when applied. The amount of smoothing that is achieved by a filter is determined by the size of the filter kernel because the larger a filter kernel gets, the more values are weighted and combined. The more values are used, the less influence one individual pixel has on the final result and the more uniform the filtered image becomes. In this work all filter kernels have a symmetric shape and all filter sizes are given in voxels. This means that every discrete value that is filtered lies in the center of the filter kernel. Examples for symmetrical filter kernels are a 5$x$5 filter kernel in 2D or a 5$x$5$x$5 filter kernel in 3D. Since EM data sets are extremely dense, noisy and heavily textured [JBH+09] these filtering methods are used for reducing the noise. Besides reducing noise for a smoother and more visually appealing DVR of the data set, this noise reduction can be used prior to other techniques which aim at enhancing the volume as well. As an example of these techniques which benefit from pre-filtering the volume, Jeong et. al. propose a „Local Histogram based Edge Detection“ method to enhance the volume's raw data [JBH+09]. We chose on-the-fly filtering as approach to filter our data sets because it allows us to steer the filtering process more flexibly. We can interactively change filter types and filter sizes and this allows us to try and compare different settings of our filtering module. Thus, we do not have to load prefiltered data sets in order to study the effects different filters have on a data set. Further, our on-the-fly filtering approach is much more memory efficient than saving all prefiltered data sets because we simply do not need any other data set than the original one. To control the filtering process the user can interact with the data set by selecting one of the provided filtering techniques and setting the dimensionality and the size of the filter kernel. The user may further adjust the rendering of the EM data set by adjusting the transfer function. Besides adjusting the visibility of certain intensities and controlling the smoothing of the data set, the user has no further control over the

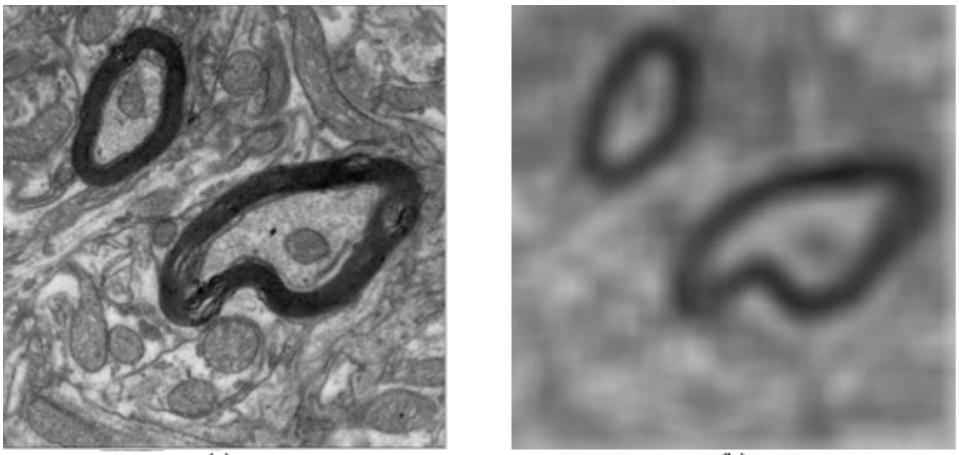


**Figure 5.1:** *Artifacts of average filter. a) original image b) averaged with a 25x25 kernel size. Note the axis aligned artifacts.*

visibility of structures within the data set. This gives the user the ability to achieve noise reduction within a maximum of three mouse-clicks. Although knowledge on the properties and achievable results of the different filter types and different kernel sizes increases the efficiency with which the user can perform the desired noise reduction, our filtering module supports the trial and error approach for less experienced users as well. This is mainly achieved by immediately reacting on selecting different filter types or kernel sizes. Depending on filter type and kernel size, applying the changes, re-filtering the data set and rendering the volume is done at $15-20fps$. Using NVIDIAs CUDA for implementing the different filter types in our filtering module, we have 3 different filter types implemented so far. We have chosen CUDA as a basis for our implementation because GPGPU is known to greatly speed up parallelizeable image processing techniques like filtering [FM08]. In the following we explain the theoretical background and the properties of the implemented filters. After the introduction of the filters, we explain their CUDA implementation with respect to their variable kernel sizes.

## 5.1 Average Blur

The most basic filter of the ones implemented in the filtering module is the „average filter“. The average filter belongs to the category of uniform filters which all

have in common that all the values within the filter kernel support have the same weight. When filtering an image with an average filter, the surrounding pixels of the filtered one are accumulated and normalized. Normalization is done by dividing the accumulated value by the total number of accumulated pixels to ensure that the output value of the filter is not larger than the largest possible pixel value. The main advantage of the average filter is, that it is fast and easy to compute. However, when using large filter kernels, average filters generally introduce axis aligned streaks and artifacts. Figure 5.1 shows one layer of an EM data set of neural tissue that is filtered with a $25x25$ average filter. Figure 5.1 a) is the original layer while Figure 5.1 b) shows the same layer after applying the average filter. Note the axis-aligned artifacts introduced to image b) due to a too large filter size.

$$A[I]_p = \frac{1}{n} \sum_{q=1}^{n} I_q \tag{5.1}$$

Equation 5.1 shows the formal description of a 1D average filter. $A[I]_p$ is the averaged value of an image $I$ at the pixel position $p$. $n$ is the filter kernel size and $I_q$ is the value of the image $I$ at pixel position $q$.

## 5.2 Gaussian Blur

Gaussian blur is one of the most widely used smoothing operations in image processing [WM98]. This results from the fact that besides being very „smooth“, the circular symmetry of the filter allows edges and lines to be treated similarly in each direction [WM98]. This is in contrast to the non-isotropic, box shaped average filter. The Gaussian blur therefore provides a consistent low pass filter regardless of the image's orientation. Because of its isotropy, the Gaussian blur is often used to de-noise an image prior to other image processing techniques like edge detection. Figure 5.2 shows that pre-filtering a volume with a Gaussian blur before applying an edge detection algorithm, significantly reduces the number of edges resulting from image noise. Since edges can be viewed as discontinuities in the intensity of an image, these discontinuities can be associated with various derivatives of the image function [SB91]. Using the derivatives to find edges however introduces noise to the result [TP86]. By using a Gaussian blur prior to edge detection and therefore low pass

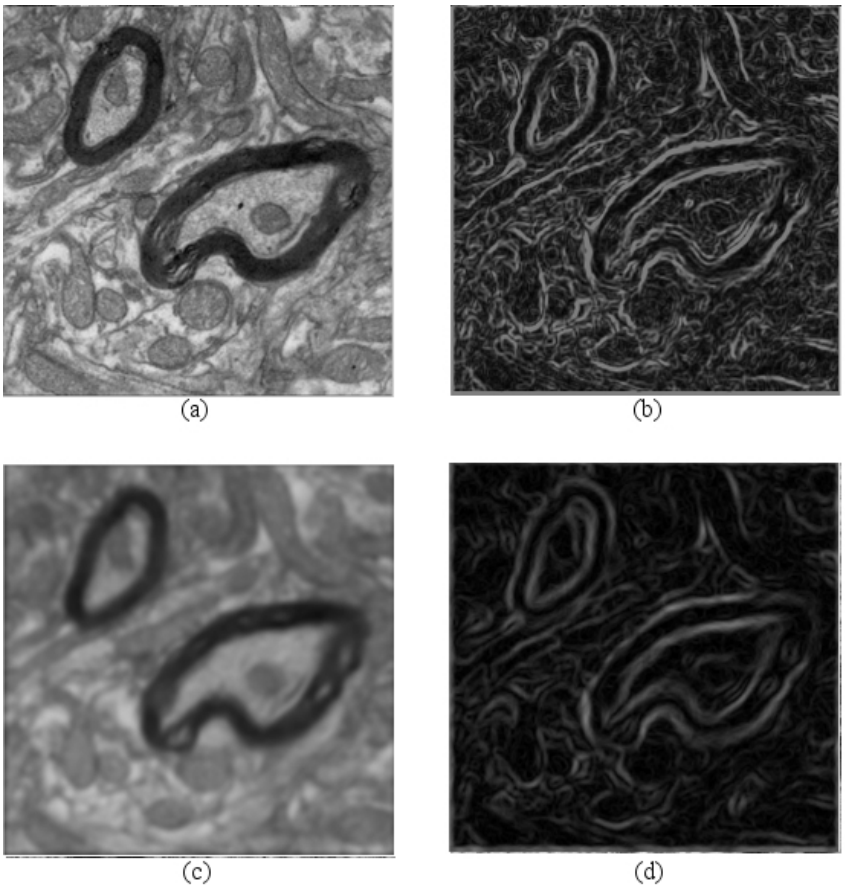


**Figure 5.2:** *Impact of smoothing an EM data set with a Gaussian blur. (a) Original Slice of an EM data set. (b) Results of an edge-detection algorithm on image (a). (c) Slice of the same data set filtered with a 5x5 Gaussian blur. (d) Applying edge detection to a pre-filtered image significantly reduces the amount of edges in the final rendering.*

filtering the image, the introduction of noise can be prevented [SB91].

$$GB[I]_p = \sum_{q \in S} G_\sigma(\|p-q\|) I_q \tag{5.2}$$

$$G_\sigma(x) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}} \tag{5.3}$$

Equation 5.2 shows the application of a Gaussian filter to an image. $S$ is the image, $p$ is the location of the center pixel, $q$ the location of a pixel in the range of the filter kernel and $I_q$ is the intensity of the pixel $q$. $G_\sigma$ is the filter weight resulting

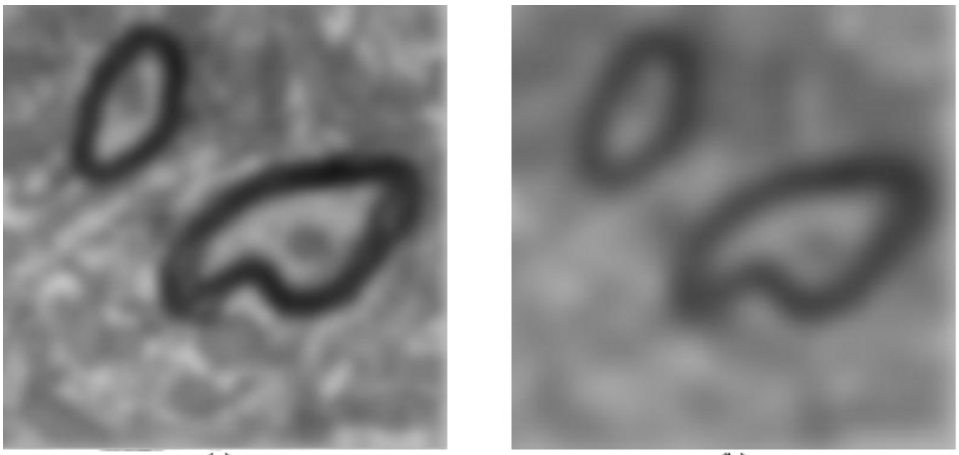


**Figure 5.3:** *Same layer of an EM data set filtered with a Gaussian blur using two different $\sigma$ values. a) Using a $\sigma$ value of* 6*. b) Using a $\sigma$ value of* 12*.*

from the Gaussian filter function that has the radius $\sigma$. $GB[I]_p$ is the result of the Gaussian blur for pixel $p$. The calculation of a 1D Gaussian filter weight $G_\sigma$ for the distance $x$ is shown in Equation 5.3. Equation 5.2 helps to understand that $x = \|p - q\|$. The value $\sigma$ is the standard deviation of the Gaussian distribution and determines the radius of the Gaussian filter curve. The larger $\sigma$ is set, the wider the Gaussian filter curve gets and the more pixels can be taken into account for filtering. Although it is possible to have a $\sigma$ that produces a larger radius than the chosen filter kernel size, in our work we chose $\sigma$ to match our kernel sizes to ensure that the contribution of the voxels, that have the greatest distance to the center of the filter, is minimal. Figure 5.3 shows two images of the same layer of an EM data set. Figure 5.3 (a) is filtered with a Gaussian blur that uses a $\sigma$ of 6 while Figure 5.3 (b) is filtered using a $\sigma$ of 12. In Figure 5.3 (b), the Gaussian filter curve includes more pixels which results in a greater smoothing than in Figure 5.3 (a). Implementation wise, performing a Gaussian blur on an image using a 2D Gaussian filter can lead to rather long execution times [WM98]. In order to speed up these execution times, the Gaussian filter's linear separability can be exploited. That means that multi-dimensional Gaussians can be separated into one-dimensional Gaussian filter vectors [VVYV98]. One for each direction. These one-dimensional Gaussian filter vectors can be calculated using Equation 5.3.

## 5.3 Bilateral Filtering

Bilateral filtering addresses the issue of smoothing edges or lines when applying a Gaussian blur to an image. A Gaussian blur can be viewed as a weighted average of the pixels in the neighborhood in which the weights decrease with the distance from the filter's center [TM98]. Thus nearby pixels are expected to be of similar intensity. This assumption fails, of course, in case of lines or edges. To preserve these edges while concurrently removing noise from an image, Tomasi et. al. introduced the concept of bilateral filtering. The main idea behind bilateral filtering is to take the differences of image intensities into account, as well as traditional spacial distances as they are used for Gaussian filtering.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p-q\|) G_{\sigma_r}(|I_p - I_q|) I_q \tag{5.4}$$

Equation 5.4 shows, that the bilateral filter is an enhancement to the Gaussian filter presented in Equation 5.2. $G_{\sigma_s}$ in this case calculates the space weight while $G_{\sigma_r}$ calculates the range weight. Both weights are calculated according to Equation 5.3. $I_p$ is the intensity at pixel $p$ while $I_q$ is the intensity of pixel $q$. $W_p$ is the normalization factor that is calculated by summing up the Gaussian as well as the bilateral filter weights. While the filter coefficients of a Gaussian filter decrease with the distance of a pixel to its center, the filter coefficients of a bilateral filter decrease with the difference between the pixel's intensity and the intensity of the center pixel. This way pixels that have intensity values that are similar to the intensity value of the center pixel are weighted with a higher value. This results in the smoothing of mostly uniform areas while edges and lines are preserved. The bilateral filtering of a noisy signal around an edge is shown in Figure 5.4. This Figure shows that by combining a spatial filter with an intensity filter, the bilaterally filtered value of a pixel is influenced mainly by pixels that are spatially close and that have a similar intensity. Figure 5.4 further shows that this influence of distance and similarity results in the smoothing of uniform areas and the preservation of edges that were mentioned above.

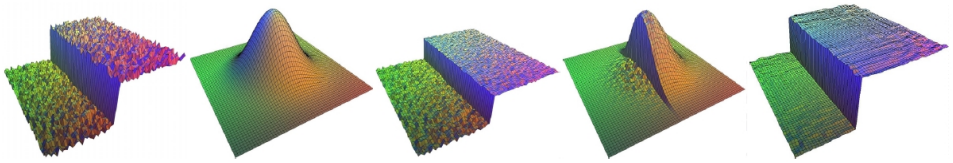

**Figure 5.4:** *Image [DD02] showing the influence of spatial distance and intensity similarity on bilateral filters. From left to right: Input image, filter kernel in spatial domain, filter kernel in intensity domain, combination of spatial and intensity filter kernels, output image.*

## 5.4 Implementation

Before being concerned with the actual filtering process, we had to ensure that only visible parts of the volume are filtered. Filtering the whole volume for every frame would on the one hand exceed the memory capacity of our graphics card and would drastically increase the render time for each frame. The block cache described in Section 4.1 does provide a visibility check to make sure that only visible blocks are rendered. We decided to implement an additional cache to this block cache that uses a smaller block size. That way we have more control over the parts of the volume that are filtered and minimize computational overhead. Each element of this new cache, similar to the block cache, represents a visible sub-volume of the EM data set. All of these elements are of equal size and are called „bricks“. These bricks are smaller than the blocks and thus provide a much more accurate visibility check. Hence less parts of the volume have to be filtered. The visibility check is performed for every frame. Every frame when the cache is full invisible bricks are removed from the cache and the newly visible ones are added. When the cache still has capacity, new bricks are simply added. After performing the visibility check, all newly added bricks are filtered. In order to filter the bricks present in the cache using CUDA the first step is to determine how to set up the CUDA block size to ensure that the parallel computation capability of the GPU is used efficiently. In our approach we have set the CUDA block size so that each CUDA thread represents a location within the filter kernel. This means that the CUDA block size and the filter kernel size are equal and that each thread always has the same offset to the center of the filter. The center of a CUDA block is the center of the filter kernel and can be related to the voxel from the brick that is filtered. Each CUDA block processes exactly one brick of the cache. When the CUDA block starts filtering a brick, each thread loops over all voxels of the brick. This is done by starting a loop for the $x$-, $y$- and $z$-axis of the coordinates of the voxels The offset of each thread from the center

of the CUDA block is added to the starting coordinate of the loops. This way, every time one thread calculates the filtered value of a voxel, the other hreads calculate the filtered value of the neighboring voxels simultaneously. After the filtered values are calculated, one thread combines the values and writes them back to the brick. Thus, every step of the loops, one voxel is filtered. This calculation process can be understood as moving the brick beneath the filter. Figure 5.5 illustrates the parallel calculation of the filter values for each step of the loops. The rectangles marked by dotted lines represent the the voxels of the brick. Rectangles filled by a stripe pattern represent the border voxels of the brick. The blue raster represents the filter kernel and the numbers in brackets represent the offsets of the threads from the center of the CUDA block. Each step of one of the loops increases the coordinates of the respective axis for all threads. This means that in every step of the loops the brick is shifted by one voxel and the filter values are calculated for the new voxels.

Since the bricks stored in the cache not necessarily have to be adjacent in the volume, it has to be accounted for how to filter the border voxels of a brick. Using the voxels of neighboring bricks in the cache or assuming the missing voxels to be of low density results in filtering artifacts. Thus the brick has to be large enough to store border voxels from its neighboring bricks as well. To prevent any filtering artifacts, this border has to be half the filter kernel size in each direction.

Program 5.1 shows a `C` style pseudo implementation for filtering a brick in CUDA. Note that this code is executed by each CUDA thread of the CUDA kernel. The actual position of a voxel of a brick in the volume is given by `sample_pos`. This position is obtained by adding the internal brick location of a voxel `brick_pos` to the offset a thread has in the volume `thread_volume_offset`. This is repeated for all voxels of brick. The brick size is given by `brick_size`. The intensity of a voxel `tex_density` at position `sample_pos` is then multiplied with the respective filter weight for the current CUDA thread index `threadIdx`. Each CUDA thread then adds its calculated intensity value to the final filter value. After making sure that all weighted intensities are added to the final value by using CUDAs `__synchronise()` command, only one CUDA thread normalizes the filter value, by dividing it by the sum of its filter weights, and writes it back to the brick at position `output_pos`.

We have now laid out the foundation for filtering a brick in CUDA. The remainder of this chapter describes the peculiarities of the implemented filter types. The

```
1  // For each CUDA thread:
2
3  for(brick_pos.x=0; brick_pos.x<brick_size.x; brick_pos.x++)
4  {
5    for(brick_pos.y=0; brickPos.y<brick_size.y; brick_pos.y++)
6    {
7      for(brick_pos.z=0; brick_pos.z<brick_size.z; brick_pos.z++)
8      {
9        sample_pos = brick_pos + thread_volume_offset;
10
11       tex_density = getVolumeIntensity( sample_pos );
12
13       tex_density *= getFilterWeight(filter_type, threadIdx );
14       filter_value += tex_density;
15
16       __synchronise();
17
18       if(threadIdx == 0)
19       {
20         normalize( &filter_value );
21         writeOutput(output_pos, filter_value);
22       }
23     }
24   }
25 }
```

**Program 5.1:** *C style pseudo implementation of our CUDA filtering algorithm.*

average filter is the least complex one of the implemented filter types. To perform averaging, each CUDA thread adds the intensity of its corresponding voxel to the CUDA block wide shared storage variable. After this is done and the CUDA threads are synchronized, one CUDA thread performs the normalization step to get the final filtering-value. In case of an average filter, the `getFilterWeight` method of Program 5.1 returns 1. Once the filter value is obtained it is stored in the corresponding brick of the brick cache. Since the bilateral filter differs only slightly from the Gaussian filter, the implementation of the Gaussian filter is explained in more detail in the following. The Gaussian filter weights only depend on the distance from the kernel center to every other location within the filter kernel and thus can be precomputed. Because of the separability of the Gaussian filter, we only need to precompute a 1D Gaussian filter instead of a 3D Gaussian filter. In case of a desired 3x3x3 filter the naive approach of calculating one weight for each position of the filter kernel results in 27 calculations. Using separable filters and exploiting the symmetry of the Gaussian filter, only 2 filter weights need to be calculated. Generally, the storage size needed to store a symmetric 1D Gaussian filter kernel is

`ceil(s/2)` where `ceil` rounds a decimal number to the next higher `integer` value and `s` is the desired filter kernel size.

$$W_x = G_{\sigma_s}(|c_x - t_x|) \tag{5.5}$$

$$W_y = G_{\sigma_s}(|c_y - t_y|) \tag{5.6}$$

$$W_z = G_{\sigma_s}(|c_z - t_z|) \tag{5.7}$$

$$W_{xyz} = W_x \cdot W_y \cdot W_z \tag{5.8}$$

The Gaussian filter weights can be calculated in parallel on the GPU using CUDA. This is due to the fact that Gaussian filter weights are computed based on one distance value. Since the distance values between the filter's center and its surrounding voxels are not related to each other, they can be calculated independently and in parallel. The calculated filter weights remain in global memory on the graphics card for further reuse during the actual filtering. The distance from CUDA threads to the center of the CUDA block are the input for the Gaussian filter function. The use of separated 1D filter kernels requires to treat the coordinates of center and CUDA threads separately for each direction. For three dimensions three filter weights are used, one for each direction. Equations 5.5-5.7 explain the calculation of the filter weights for each direction where $cx, cy, cz$ represents the center coordinates and $tx, ty, tz$ represents the thread coordinates. Equation 5.8 shows that the final weight $W_{xyz}$ at CUDA thread position $xyz$ results from multiplying the direction weights $W_x$, $W_y$ and $W_z$. The weighted intensity is given by multiplying the original intensity at the CUDA thread-position with the final filter weight $W_{xyz}$. All filtered values from CUDA threads within the CUDA block are summed up to get the final filtered-value of the CUDA block. The implementation of the bilateral filter is an enhancement to the Gaussian filter and re-uses its precomputed filter weights. The bilateral filter introduces a second weight besides the final filter weight of the Gaussian filter. This filter weight is calculated by the same function $G_{\sigma_r}$ as the Gaussian filter weight. The difference lies in the chosen input parameter. While the Gaussian filter weights

depend on the distance of two spacial locations, the bilateral filter weight depends on the difference between two intensity values. This difference is calculated by subtracting the smaller intensity value from the bigger one and normalizing them to the range of $[0, 1]$. When combining bilateral and Gaussian filter weights for calculating the final filter value the $W_p$ factor from Equation 5.4 is used for normalization. Due to the separability of the Gaussian filter, we use only the 1D Gaussian function to calculate the Gaussian filter weights. Since this function was imlemented to tage the radius $\sigma$ as input, this function is re-used for the bilateral filter weight because the difference of two 1D intensity values is per definition one dimensional. The radius $\sigma$ is set to be 1.0 since this is maximal distance between intensities that can occur in our data sets. The bilateral filter weights are not precomputed but calculated on-the-fly for each CUDA thread because it is not known in advance which intensities are present in the volume. The final filter value is calculated by multiplying the intensity value of the voxel that is filtered first by the Gaussian filter weight and then by the bilateral filter weight.

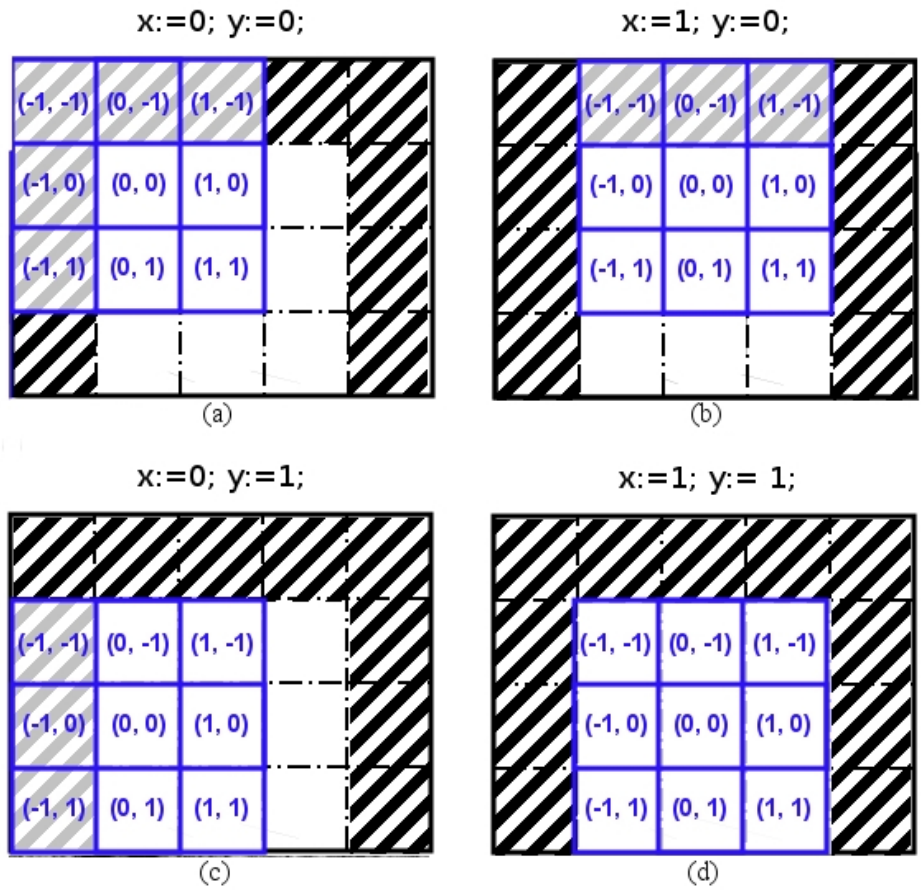


**Figure 5.5:** *Filtering of a part of a brick. Rectangles that have a dotted line as border are the voxels of the brick. Rectangles filled with a stripe pattern are the border voxels of the brick. The blue raster is the filter kernel and the number in brackets are the CUDA thread offsets to the filter kernel-center. From (a) to (d) the brick is shifted by one voxel by looping over the voxels coordinates. CUDA thread offsets are added to the voxel-coordinates of the loops to calculate filtered values for all voxels within the filter kernel.*

# Chapter 6

# Volume Exploration Module - Data Reduction by Picking

*If you limit your choices only to what seems possible or reasonable, you disconnect yourself from what you truly want, and all that is left is a compromise.*

*-Robert Fritz*

With the module presented in this chapter, we enable the user to define which structures of the volume he considers to be either noise or relevant. Instead of contextless smoothing operations on the entire volume, like it is the case with our filtering module presented in Chapter 5, we give the user the means to select the structures of the volume he wants to see and hide the rest. Finding structures in data of any kind is the domain of pattern recognition systems. Pattern recognition systems compare formally described structures statistically to a given data set [Fuk90]. The general goal of these pattern recognition systems is to automate the decision making process in matching and comparing data of any kind. Pattern recognition is not limited to images. Pattern recognition systems may be used for finding and recognizing a structure in an image and automatically mark it or buying stocks by analyzing a complex pattern of information. Using automated pattern recognition systems, however, requires the individual parametrization of any desired type of structure a user may want to find.

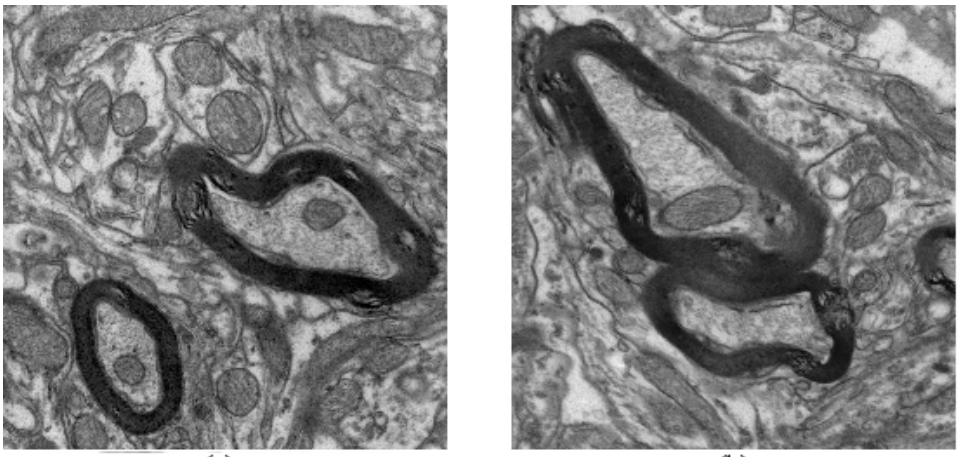


**Figure 6.1:** *The dark ellipsoid like shapes in both images (a) and (b) are myelin sheaths of two different axons. The variance in size and shape of axons makes it hard to develop automated segmentation algorithms.*

In case of volume data sets of neural tissues like the EM data sets we use in this thesis we do not have single images to compare patterns but a 3D volume to find these patterns in. Structures and shapes may vary drastically in neural tissue which can be observed in Figure 6.1. This figure shows the same two axons in two different regions of the volume. Note that even the same axon may vary its shape drastically within the same data set. Finding axons in an EM data set would require matching the points of interest acquired from the desired structures against the whole EM data set to find similar ones [Rip08]. To account for the possible variance in shape and scale of the structures within the volume, the patterns as well as the data set have to be processed and searched at each different possible scale [AAR04]. Searching the whole EM data set to find voxels that are connected to resemble a given pattern and repeating that procedure for each possible scale of these patterns would at least be time consuming if not impossible. As solution for this problem we propose a user guided out-of-core solution that allows a user to select or discard structures from a volume data set by simply clicking on a DVR of this data set. By clicking on the data set, the user marks the structures he wants to suppress and our algorithm finds and suppresses similar structures as well. This can be done either on a fully opaque rendering of the data set or on a DVR that already uses a transfer function to mask certain intensities. With each click, more structures are found and suppressed and the rendering of the data set is constantly adjusted to show the data set without the suppressed structures. For the user's convenience we

implemented the functionality to undo any selection or reset all selections.

In the remainder of this chapter we describe the theory behind our matching algorithm. We further explain the technical conversion of this theory to our „Volume Exploration module“ (VE module) and give a short introduction to the usage and the user interface of this module.

## 6.1 Structure Recognition in Theory

Effectiveness and efficiency are key requirements in user interface design [Opp02]. To accomplish efficient interaction with the data set we believe that providing the user with instant feedback on his actions is crucial. Since we want a user guided solution for our structure-recognition technique we need instant feedback on his selections. We provide this instant feedback by using a fast and easy to compute matching algorithm. Histograms have been widely used for structure recognition tasks [AHP04, LVB+93] because they are easy and fast to compute and provide a significant data reduction in comparison to the original image data. They are robust to noise and to local image transformations [HGN01a]. EM data sets are noisy by nature and the shape and transformations of regions of interest may vary drastically even within the data set. Histograms seem to be the ideal choice for matching structures within EM data sets. Histograms however are unable to encode spatial correlations [HGN01b]. In other words, two images may have the same histogram although their composition is completely different. Hadjidemetriou et al. propose a new type of histogram that encodes spatial correlations by using histograms of different resolution levels of the original image [HGN01b]. In their paper they call this type of histogram a „multiresolution histogram“ and define it as follows:

> A multiresolution histogram is the set of intensity histograms of an image at multiple resolutions.

Analyzing histograms of different image resolutions allows to discriminate between different images even if the histograms of their original images are equal. Multiresolution histograms therefore are sensitive to changes in image structure and image transformation, which makes them useful for matching operations. Figure 6.2 shows the image pyramids and histograms for two images having the same histogram at the

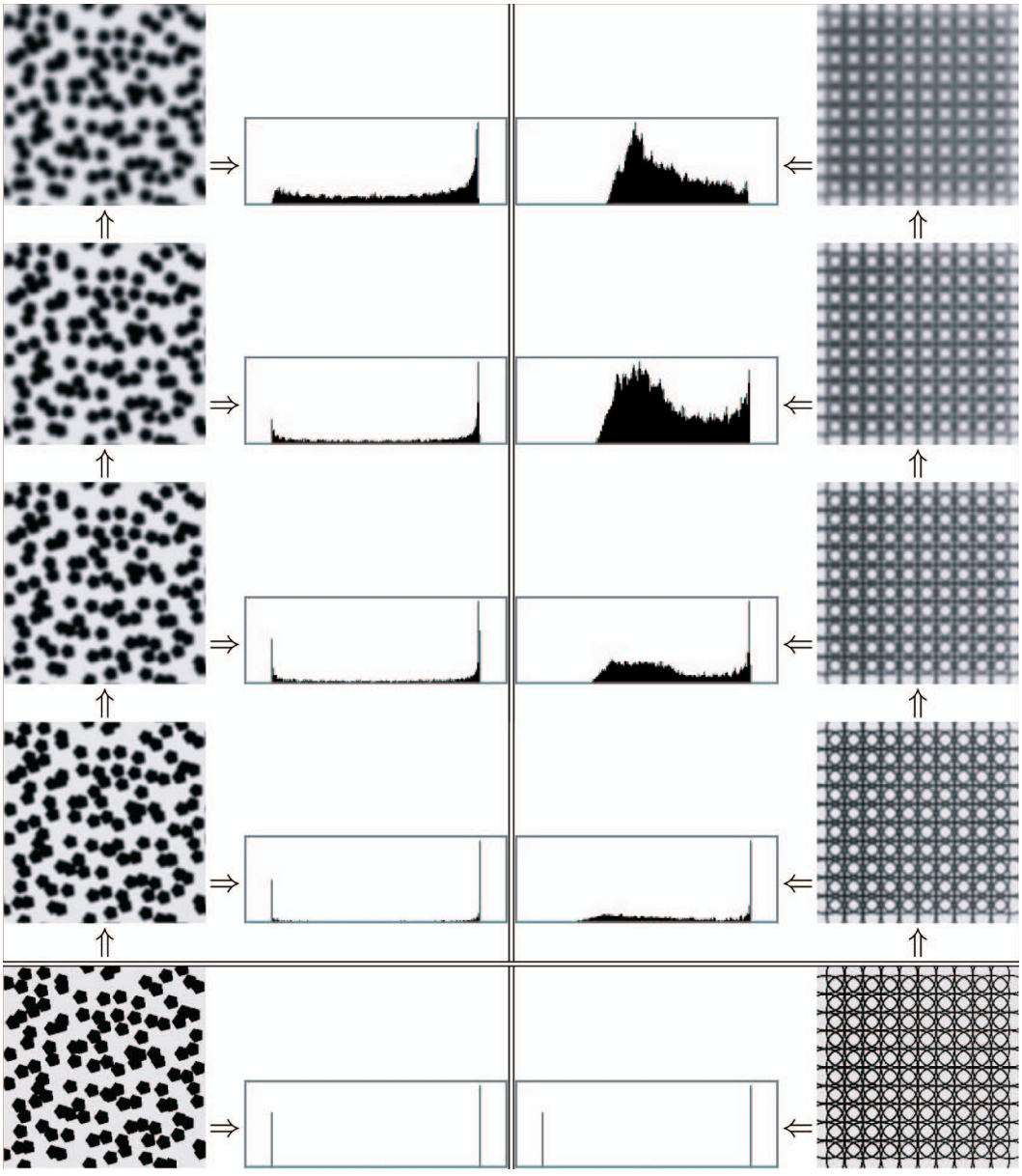

**Figure 6.2:** *Examples of two multiresolution histograms [HGN04]. The bottom row shows the original images and their identical histograms. The other rows show the two images at decreasing resolutions. The lower the resolution becomes, the more the histograms differ.*

initial resolution. Note the increasing difference of the histograms at each resolution step. The first step of Hadjidemetrou's algorithm is to calculate the image pyramids of the images they want to match. An image pyramid is created by low pass filtering an image with a two dimenstional filter, a Gaussian blur for example [OABB85]. The filtered image then is subsampled by removing every other row and pixel to obtain an image of half the size of its predecessor. This process is repeated until the desired number of resolution-steps for the image pyramid is obtained. This process is expendable to 3D volumes by using 3D low pass filters and removing every other image layer in $z-direction$ in addition to every other voxel. To decrease execution time when calculating the image or the volume pyramid, calculating the filter values for pixel or voxels, that are removed anyway, can be omitted. An image pyramid as well as a volume pyramid consists of a base image, or a base volume respectively and a series of successively smaller sub-images or sub-volumes each of half the resolution of their predecessor. To form a multiresolution histogram as it is proposed [HGN01b], an intensity histogram is computed for each level of the image pyramid. In order to compare the multiresolution histograms, Hadjidemetriou et al. calculate the Manhattan distances of the corresponding histograms at each resolution level. The sum of all Manhattan distances calculated for two multiresolution histograms is then used as a feature for comparison. The Manhattan distance (or L1 distance) can be understood as the distance between two points on a regular grid where only paths parallel to the axes can be taken [SPHC02]. Formally, the L1 distance is calculated according to Equation 6.1 where $n$ is the number of bins of the histograms, $L1$ is the L1 distance and $a$ and $b$ are the two histograms we want to calculate the distance of and $n$ is the bin size of the histogram. The L1 distance is known to be better suited for the use with histograms than the Euclidean distance [DPVN08] since it provides a more accurate difference in high-dimensional data.

$$L1 = \sum_{i=1}^{n} |a_i - b_i| \tag{6.1}$$

In a later work, Hadjidemetriou et al. proposed an improved way of measuring the distance between multiresolution histograms [HGN04]. Instead of using the sum of all the L1 distances of every level of the multiresolution histogram, they propose an implementation that caters more to the dissimilarities of the single levels of the multiresolution histograms. For this reason they developed a new type of multiresolution histogram as well as a new way of measuring the distance. The

main differences between the original proposal and its enhancement are the way the multiresolution histograms are built and how the distances between two multiresolution histograms are calculated. To form the multiresolution histograms, no longer the intensity histograms for each resolution level are calculated, but the differences between the histograms of consecutive image resolutions. These differences are called difference histograms and are computed by taking two histograms of consecutive resolution levels and subtracting the histogram-values for each bin. An image pyramid consisting of $n$ levels thus has $n-1$ difference histograms. Further, not the intensity histograms for each resolution level are used to calculate the difference histograms but the cumulative histograms of these intensity histograms. Cumulative histograms are constructed by consecutively summing up all histogram bins of one intensity histogram. This gives a histogram that has a plot that has increased bin heights for every step on the intensity-value axis.

$$CH_i = \sum_{j=1}^{i} H_j \tag{6.2}$$

Equation 6.2 shows the formal description of the construction of a cumulative histogram. The cumulative histogram $CH$ at intensity value $i$ is the sum of all the histogram values of histogram $H$, that range from the lowest intensity $j$ to the actual intensity $i$. The cumulative histogram can be seen as a histogram that stores, for each intensity value, the probability to find a pixel that has an equal or less intensity . In order to calculate the cumulative histograms and then the difference histograms, every original intensity histogram is normalized using its L1 length as normalization-factor. Normalizing the intensity histograms makes the multiresolution histograms independent of image size and resolution [HGN04].

$$vf = \sum_{i=1}^{n} dh_i \tag{6.3}$$

To calculate the distance between the multiresolution histograms, all difference histograms are summed up to form a feature vector according to Equation 6.3. $vf$ is the feature vector, $dh_i$ is the vector representation of a difference histogram at resolution level $i$ and $n$ is the total number of resolution levels of the multiresolution histogram. The distance between two multiresolution histograms is the L1 distance of their feature vectors.

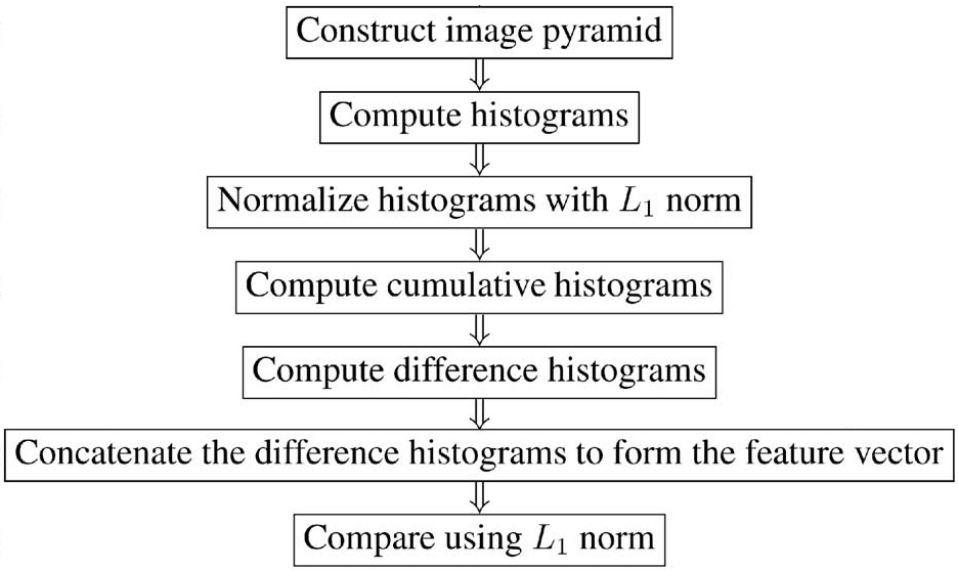


**Figure 6.3:** *Steps of the matching algorithm [HGN04]. After constructing the image pyramid, the histograms are generated and normalized. Then the cumulative- and difference histograms are generated and renormalized. After construction of the feature vector they are used for comparison using the L1 distance.*

To give a short summary on the multiresolution histogram matching-algorithm, we list each step of the algorithm in consecutive order:constructing the image pyramid and computing the intensity histograms, normalizing the intensity histograms using L1 norm, computing cumulative- and difference histograms, combining the difference histograms to form the feature vector and calculating the distance between two feature vectors using the L1 distance. Figure 6.3 presents these steps in a graphical manner.

The matching-algorithm proposed [HGN04] aims at comparing images within a database to find similar ones. Our goal however is not to find images in databases but to find similar structures within the same EM data set. Therefore we have to expand multiresolution histogram matching from 2D image pyramids to 3D volume pyramids. In contrast to comparing images within a database, the sizes and locations of the structures a user might want to eliminate are not known in advance and may not only differ from data set to data set but within the same data set as well. To overcome this issue we decided to divide the volume into smaller sub-volumes of equal size, similar to the bricks used in Section 5.4. This however results in issues that are discussed more thoroughly in Section 7.2. Each of these bricks is selectable and has its own multiresolution histogram. When selected, the multiresolution

histogram of the selected brick is compared to the multiresolution histograms of all the other bricks to identify and either show or eliminate similar bricks. The remainder of this chapter explains how we expanded multiresolution histogram matching from finding 2D images in image databases to finding sub volumes in a 3D volume data set.

## 6.2 Implementation

The implementation of the multiresolution histogram matching-algorithm is a translation of the single steps of the matching algorithm presented in Section 6.1. At first, the volume pyramid is constructed by using the graphics cards memory and CUDA. Then the histograms and multiresolution histograms are calculated in a preprocessing step and stored for reuse. Storing the multiresolution histograms for reuse means that they only have to be computed when a data set is loaded to the HVR framework for the first time. When a data set is loaded for the first time, the multiresolution histograms are calculated and stored on disk. When the same data set is loaded again, the multiresolution histograms are read from local memory. This speeds up the preprocessing step of the VE module. After activating histogram-picking in the HVR framework, a user can pick any visible brick of the volume. Once a brick has been picked, its multiresolution histogram is compared to the ones of all the other bricks. When the distance between two multiresolution histograms is within a user chosen L1 distance, the bricks in question are marked and can either be set to be visible or hidden to the user. The remainder of this section describes the construction of the volume pyramid and the class layout of all the classes implemented for the multiresolution histogram matching algorithm. We further explain how a brick is picked in the DVR of a data set and give a short overview over the usage and the user interface of the VE module.

### 6.2.1 Construction of Volume Pyramid

The construction of the volume pyramid uses the texture manager described in Section 4.1 as well as the capability of CUDA to access, modify and copy data directly on the graphics card. Similar to a texture handled by the texture manager, the volume pyramid is stored as one 3D cache-texture and one address-texture per

resolution level. Having different cache- and address-textures for each resolution level avoids introducing additional coordinate transformations as it would be the case with mipmapping extended to 3D [Wil83, BIP99]. Since no hardware 3D mipmapping exists yet, these transformations would have to be done by software. Performing these transformations in software is slower than using different cache-textures because these additional transformations would have to be calculated for each voxel along a ray of the ray caster in order to get the right voxel of the desired mipmap-level. By using different cache- and address-textures, no additional transformations are introduced and besides exchanging the adress and cache textures for each resolution level, the code of the CUDA ray caster remains unchanged. This makes our approach to volume pyramids easily adaptable to any CUDA ray caster using the implementation introduced in Section 4.2. Since each cache texture represents the same volume at different resolutions, we construct each of these cache textures to have the same layout. This way, the texture manager does not have to handle different cache layouts. However, the texture manager had to be changed to handle multiple instances of cache textures instead of one. Another issue, that using multiple cache textures at different resolutions introduces, is that although the layout of the blocks within each cache is the same, the address textures for each resolution level have to differ. This can be understood by examining Equations 4.1 and 4.2. There it is evident, that the texture-space block-cache coordinates $x'_{x,y,z}$ depend on volume and cache size as well as tile and block size. When bisecting a volume, volume size and tile size are both half their original size in order to keep the same volume size to tile size ratio. The size of a block in the cache texture however is set by the size of the respective tile combined with the border voxels. Since the number of border voxels stays the same for each resolution step, the cache texture for each resolution is slightly bigger than half the size of the original one. Due to that disproportion of volume and cache size, the address texture has to be calculated separately for each resolution level. Reducing the resolution of the blocks from one cache texture to another cache texture is done directly on the graphics-card. Since CUDA does not allow direct texture-modification, the blocks have to be copied to an intermediate CUDA array before being copied to their designated texture. This copy process is done in its own CUDA kernel in which the resolution reduction takes place as well. For every resolution step of the volume pyramid, the lower-resolution blocks are calculated from the original cache texture. Equation 6.4 shows the retrieval of the correct voxel for any resolution level. In this equation, $block_{threadIdx_{x,y,z}}$ represents the index of the voxel of the

block at $threadIdx_{x,y,z}$ which is the actual CUDA thread index in the block. To get the position of the voxel in the cache, the offset $startPos_{x,y,z}$ has to be added to the CUDA thread index. This offset is the index of the original block in the cache and the factor $stepsize$ is calculated according to Equation 6.5. This offset increases with each resolution level and by multiplying the CUDA thread index with it, the distance between the voxels that are stored for each resolution level increases as well. The resolution level of the new block is given by $level$. For example at resolution level 1 (starting at level 0 for the original resolution), every second voxel is copied into the new block.

$$block_{threadIdx_{x,y,z}} = startPos_{x,y,z} + threadIdx_{x,y,z} \cdot stepsize \tag{6.4}$$

$$stepsize = 2^{level} \tag{6.5}$$

The histograms of each resolution of each brick are calculated from the respective cache textures in an additional step. Each brick and its different resolution levels are extracted from the cache textures and the histograms are calculated by looping over the intensity values for each resolution. Our reasons of treating the volume pyramid generation and the histogram calculation separately is that the whole process of constructing the different cache textures of the different resolutions fully takes place on the GPU. The histograms however need to be present in the host environment to perform the matching calculations. Although the histograms can be calculated on the graphics card by CUDA, they still need to be copied back to the host which is time consuming and slows down the generation of the volume pyramid.

### 6.2.2 Class Layout

The classes described in the following manage histograms as well as multiresolution histograms and handle the picking process. After loading or calculating each histogram for each resolution level of a brick, each histogram is stored in an object of the `VE_Histogram` class. These `VE_Histogram` objects are then added to an object of the `VE_MultiHistogram` class, which is the implementation of a multiresolution histogram as it is proposed by Hadjidemetriou et al. [HGN04].

When all VE_Histograms are added, the VE_MultiHistogram is added to the VE_MultiHistogramManager. The VE_MultiHistogramManager contains all the logic for comparing the multiresolution histograms and setting the visibility of the bricks. In the following the properties and functionality of these classes are described in more detail.

#### 6.2.2.1 VE_Histogram

The VE_Histogram contains all the functionality to handle one single histogram. After a histogram is stored in this class, it is automatically normalized with the L1 norm. From this normalized version the cumulative histogram is computed and stored together with the original histogram. Further, this class contains functionality to obtain the distance between two histograms. The histograms to calculate the distance from may be intensity, cumulative or difference histograms or any other multi-dimensional vector. The distance functions are implemented as member functions of the VE_Histogram class, taking two multi-dimensional vectors as input. Even though the L1 distance measurement is better suited for histograms [DPVN08], for comparison the Euclidean distance measurement can be used as well. In addition to distance measurement, the VE_Histogram class also contains the means to calculate a difference histogram out of two other histograms. This function is a static implementation as well, and allows two multi-dimensional vectors as input and returns the difference histogram as output. This strategy, similar to the distance calculation, allows the construction of a difference histogram out of any kind of multi-dimensional vector.

#### 6.2.2.2 VE_MultiHistogram

The VE_MultiHistogram class stores one VE_Histogram for every resolution level of the data set and represents the multiresolution histogram for one pickable brick. It provides functionality to add or remove VE_Histograms as well as it generates the feature vector for the distance measurement method described by Hadjidemetriou et al. [HGN04]. The feature vector is adjusted whenever a VE_Histogram is added or removed. While the VE_Histogram class can calculate the distance between two single-resolution histograms, the VE_MultiHistogram class calculates the distance between two multiresolution

histograms. Currently, there exist two different implementations to retrieve the distance between two multiresolution histograms. The first one implements the method originally proposed by Hadjidemetriou et al. [HGN01b]. Here the L1 distances between the histograms of the same resolution level is calculated for each resolution. The distances for each resolution level then are summed up to get the final distance. The second implementation is presented in [HGN04] and uses the L1 distance of the feature vectors of two multiresolution histograms. Since both implementations require the L1 distance between multidimensional vectors, the implementation of the L1 distance in the `VE_Histogram` class is used. For comparison, the user may change the kind of distance measure he wants to use.

#### 6.2.2.3 VE_MultiHistogramManager

The `VE_MultiHistogramManger` contains the actual implementation of the algorithm proposed by Hadjidemetriou et al. [HGN04]. It contains one list of all `VE_MultiHistograms` of the bricks within the volume. Any brick within that list can be found by its coordinates within the volume. When a brick is picked, the according multiresolution histogram is found using these coordinates.

The `VE_MultiHistogramManager` further maintains two arrays. One array contains the visibility information for each brick of the volume and the other array contains color information for each brick. This color information is used to mark selected bricks in the DVR of the volume. Each frame, a texture is generated for each of these arrays. In the DVR, every voxel along each ray is tested against these textures and has either its' visibility or color changed when the textures contain that specific information. The first action after picking a brick is to assign a color to it by saving the color at the respective location in the color array. Then the distance between the multiresolution histogram of the picked brick and all the other multiresolution histograms are calculated. If one of these distances falls below a user set threshold, the brick is marked in the visibility array. Any distance measurement supported by the `VE_MultiHistogram` class can be chosen to calculate that distance. So far, both multiresolution histogram comparison methods introduced by Hadjidemetriou et al. are supported. After the visibility of all bricks is determined, the visibility array as well as the color array can be obtained from the `VE_MultiHistogramManager` by us-

ing the respective methods and can be assigned to the textures that are used in DVR. Each voxel in these textures represents exactly one brick of the volume. Thus, these textures have as many voxels as there are bricks in the volume. The `VE_MultiHistogramManger` further contains functions to facilitate interacting with it. It provides functionality to undo a selection, clear all selections as well as remove individual picked bricks. The usage of these functions is further explained in Subsection 6.2.3. The `VE_MultiHistogramManger` further contains the functionality for automatically storing and loading a multiresolution histogram as it was introduced at the beginning of this chapter. When a data set is loaded, the `VE_MultiHistogramManager` checks whether histograms are stored for this data set. If the histograms are stored, they are loaded from local memory directly into the `VE_MultiHistogramManager` and the process of obtaining the histograms from the volume texture on the GPU is omitted. When the histograms are stored locally, this caching method speeds up the startup time of the HVR framework by a factor of 100.

### 6.2.3 User Interface and Usage

This section describes the interaction possibilities with the VE module. The main interaction between the user and the VE module is selecting a brick in the volume. This selection is done by clicking on the DVR of the data set. From this clicking position in 2D screen space, a ray is cast in viewing direction through the volume. The first non transparent voxel that is hit by the ray belongs to the brick that is selected. Since volume and brick size are known, the 3D volume coordinates of this brick can be calculated using the volume coordinates of the hit voxel. The 3D volume coordinates are then used as input for the `VE_MultiHistogramManager` to find the respective multiresolution histogram and start the events leading to coloring picked bricks and modifying the visibility of similar ones. The retrieval and opacity change of similar bricks as well as the coloring of the selected ones is performed at interactive framerates. This allows immediate reaction to the users interaction with the VE module.

After having picked a brick, the user interface provides the following interaction possibilities with the VE module:

- **Undo button**: When having picked a brick unintentionally, the user can

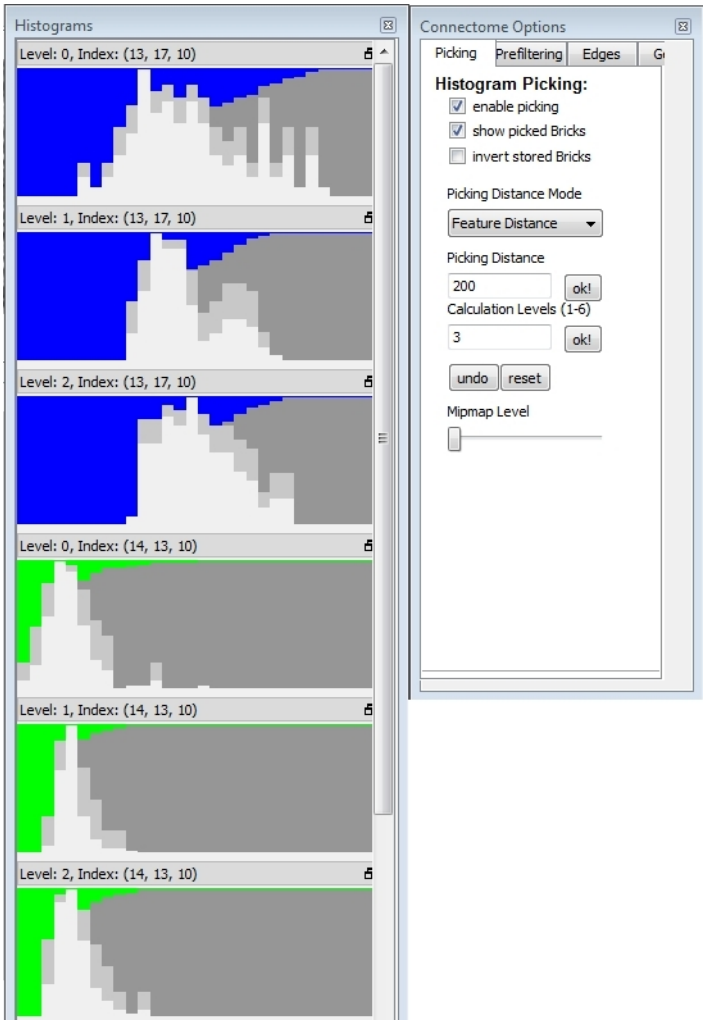


**Figure 6.4:** *User interface of the VE module. The user chooses the type of distance measurement, the distance threshold (that is a L1 distance of two multiresolution histograms) for the measurement, and the number of resolution levels that are taken into account. He can toggle the visual markings of picked bricks and can invert the visibility of the matching bricks.*

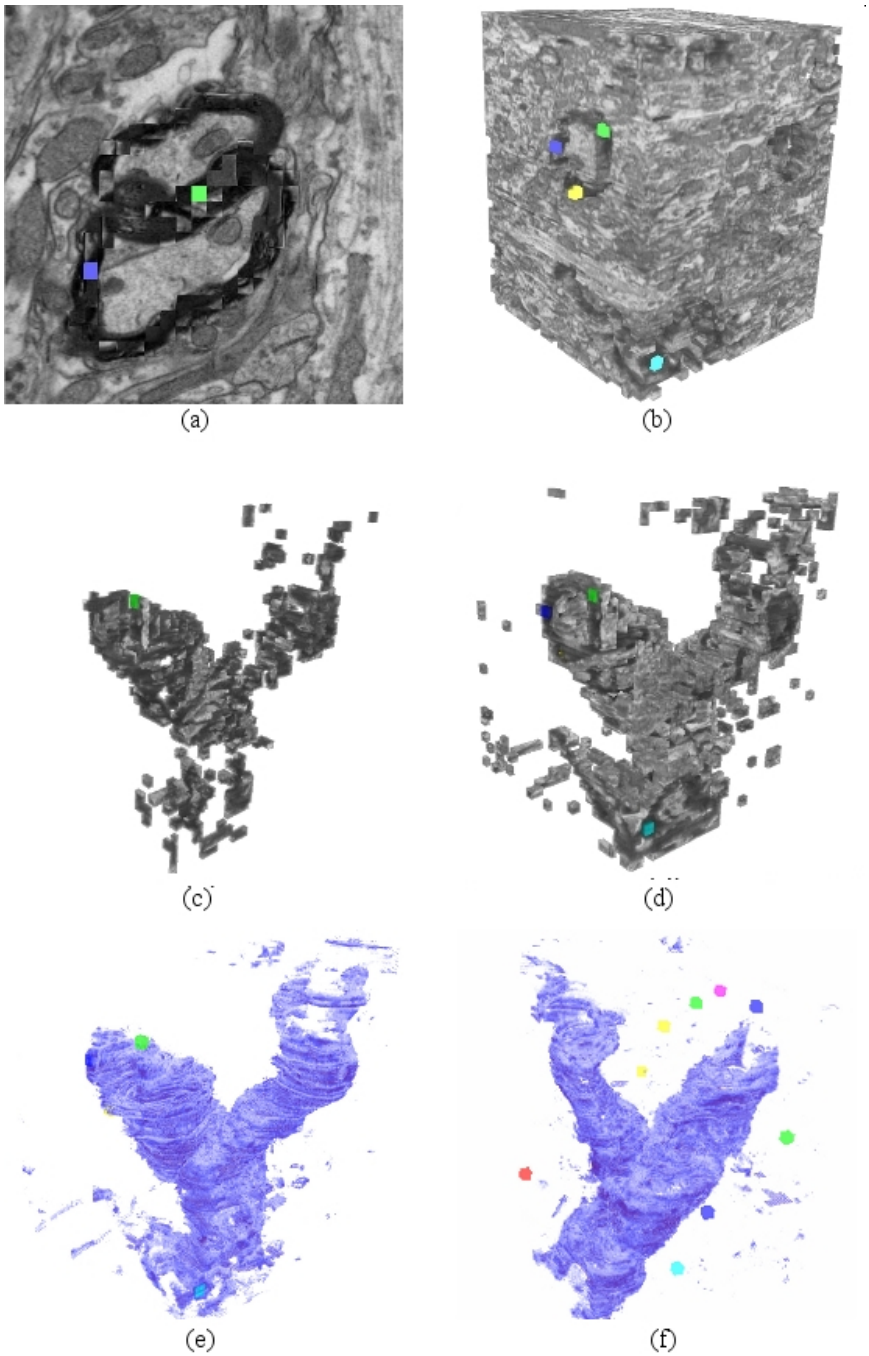


**Figure 6.5:** *Different interaction possibilities with the VE module. (a) Picking bricks using the slice view. (b) Picking bricks in an opaque rendering in DVR. (c) Inverting the selection. (d) Picking additional bricks in inverted selection. (e) Change of transfer function on selection. (f) Picking bricks after application of a transfer function. The colored dots are selected bricks.*

undo this selection pressing this button. He can deselect any picked brick by clicking on it again. After undoing or deselecting a brick, the visibility of all bricks is reset and calculated again.

- **Reset button**: In order to enable the user to explore a volume based on different criteria, the user always can reset the volume to its original state by using this button. Doing so will remove all picked bricks and reset the visibility of all bricks. After that, the whole process of selecting and changing the visibility of bricks can start again.
- **Invert checkbox**: During testing, it has been proven useful to be able to invert the visibility of marked bricks. That means, after checking this checkbox, visible bricks become invisible and vice versa.
- **Show picked bricks checkbox**: The user might as well want to examine visible bricks without having the color overlay of picked bricks occluding them. Therefore, the user can toggle the visibility of the color overlay of picked bricks by checking the „show picked bricks checkbox“.

To provide additional information to the user, the last five multiresolution histograms are displayed in an extra window. Each multiresolution histogram displays all its resolution levels. For each resolution level, the cumulative- and intensity histograms are combined into one visualization. Although the visualization of the cumulative histogram is arranged behind the visualization of the intensity histogram, and thus the first fiew bins of the cumulative histogram are occluded, it still provides insight on the shape of the cumulative histogram. The most important information on the structure of a brick is given by its intensity histogram since it reveals the most prominent intensity distributions. Thus the visualization of the intensity histogram is arranged on top of the visualization of the cumulative histogram. The background color of the histogram visualization matches the color overlay of the picked brick it belongs to. This facilitates the identification of the different histograms that belong to different bricks. Displaying these histograms is necessary to compensate for the lack of 3D information when picking in 2D. The picking process occurs in 2D since the volume is rendered to a 2D canvas. Opaque voxels of a brick occlude voxels behind them and thus only the histograms can reveal the composition of a brick. Figure 6.4 shows the user interface of the VE module. It shows the interaction possibilities with the VE module and the display of the multiresolution histograms. Figure 6.5 shows the different interaction possibilities with a volume. Figure 6.5(a)

shows a slice view of a Connectomics data set where two different bricks have been picked. The picked bricks are colored differently. Figure 6.5(b) shows the picking of bricks on a fully opaque DVR of the same Connectomics data set. Bricks similar to the picked ones have already been set to invisible. The functionality of the „invert“ checkbox is demonstrated in Figure 6.5(c). In this figure, only one brick is picked. To increase the number of matched bricks, additional bricks can be picked. This works as well when the visibility of the bricks is inverted. Increasing the number of matches when inverted visibility is selected is demonstrated in Figure 6.5(d). The application of a 1D transfer function to visible bricks is demonstrated in Figures 6.5(e)-(f). In Figure 6.5(f) we show that bricks can be picked after the application of a transfer function.

# Chapter 7

# Results and Evaluation

*There are no secrets to success. It is the result of preparation, hard work, and learning from failure.*
*-Colin Powell*

This chapter presents the results of our attempt to improve the quality of direct volume rendering of EM data sets by reducing noise and artifacts in these data sets. We show our findings by comparing the results for both approaches we investigated. First, our filtering module is evaluated. The second part of this chapter describes the results that we achieved using our VE module. The performance evaluations for both methods were done on the following system:

- Processor: Intel Core i7 CPU 2.64 GHz
- RAM: 6GB DDR2
- Graphics Processor: NVIDIA GeForce 280GTX

For filtering, we present the execution times and frame rates of the different filters we implemented. The performance of the VE module is evaluated in terms of generation times for the volume pyramid and multiresolution histograms respectively and the achieved frame rates.

| Volume Size | Brick Count | Filter Method | Kernel Size | Execution Time | FPS |
|---|---|---|---|---|---|
| $256 \times 256 \times 159$ | 1594 | Average | $3 \times 1$ | 0.026s | 17 |
| $256 \times 256 \times 159$ | 1594 | Average | $3 \times 3$ | 0.026s | 17 |
| $256 \times 256 \times 159$ | 1594 | Gauss | $3 \times 1$ | 0.097s | 14 |
| $256 \times 256 \times 159$ | 1594 | Gauss | $3 \times 3$ | 0.134s | 14 |
| $256 \times 256 \times 159$ | 1594 | Gauss | $7 \times 1$ | 0.174s | 14 |
| $256 \times 256 \times 159$ | 1594 | Bilateral | $3 \times 1$ | 0.175s | 14 |
| $256 \times 256 \times 159$ | 1594 | Bilateral | $3 \times 3$ | 0.175s | 14 |
| $256 \times 256 \times 159$ | 1594 | Bilateral | $7 \times 1$ | 0.253s | 14 |

**Table 7.1:** *This table shows the execution times of the filters implemented in the filtering module. Note that the increase of time correlates with the increase of the complexity of the filters. The filter setups shown in this table are ordered first by their complexity and second by their filter size.*

## 7.1 Results - Filtering

The results for the filtering module are evaluated due to two criteria: The execution time of each filter and the quality of the produced images. Table 7.1 shows the execution times of the different filters implemented. This results were obtained by filtering a data set that is $256 \times 256 \times 159$ voxels in size. The brick size is set to be $10 \times 10 \times 10$ voxels with 2 border voxels in each dimension. The execution time depends on the number of bricks that need to be filtered and the complexity of the filters. The maximal number of bricks as well as their size are predefined by the brick cache. Since only bricks present in the brick cache are filtered, only these two values influence the execution time.

The visual quality achieved by the filtering module can be observed in Figures 7.1 and 7.2. Figure 7.1 shows filtered slices of an EM data set of neural tissue rendered at full opacity. We used a filter kernel size of $7 \times 7 \times 1$ for all three filters demonstrated in this figure. We set the third dimension in the filter kernels to one to demonstrate that our filtering module is capable of filtering in 2D when displaying 2D slices. Our EM data sets are anisotrop. This means that the slice distance in $z$-direction is greater then the voxel distances in $x$- and $y$-direction. When applying a 3D filter while displaying 2D slices, the voxels of adjacent slices are included in the filter values as well and thus result in an undesired blending of these slices. Filtering only in 2D when displaying 2D slices solves this problem. Figure 7.1 (a) shows the rendering of the unprocessed slice for comparison. Figures 7.1 (b)-(d) show the slice being

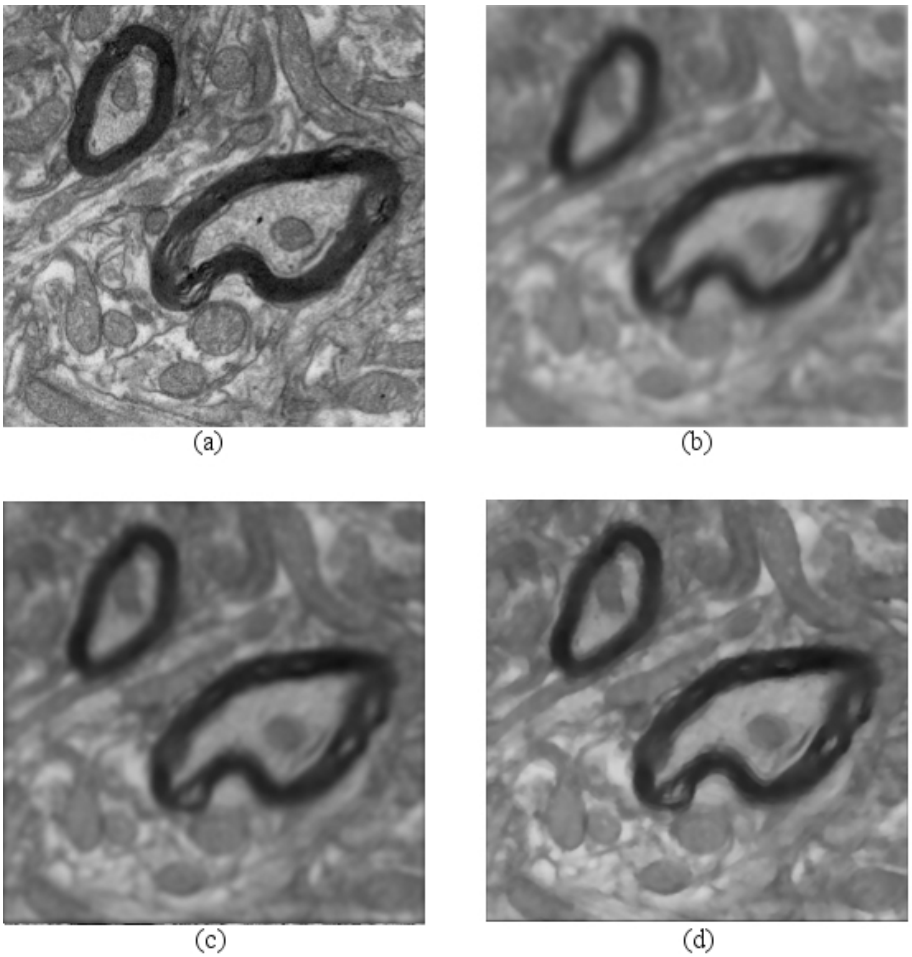


**Figure 7.1:** *2D slice of a filtered* $256 \times 256 \times 159$ *voxel EM data set. (a) Original axis-aligned slice of the data set with a fully opaque transfer function. (b)-(d) show the same slice being rendered by the average (b), Gaussian (c) and bilateral (d) filter. Filter-kernel size for all three filters was* $7 \times 7 \times 1$.

filtered by the average, Gaussian and bilateral filter explained in Chapter 5. The average filter results in the highest degree of blurring. However the image structures are highly blurred as well. Due to using 3D filters and trilinear texture interpolation provided by the graphics hardware, the axis aligned artifacts usually present in average-filtered images (see Section 5.1) are reduced. The Gaussian filter produces uniform blurring of the image and the image structures remain distinguishable. The bilateral filter unifies the blurring of uniform image structures and the preservation of their boundaries. Figure 7.2 shows the DVR of a volume where the noise and artifact reduction is performed with the same filters as in Figure 7.1. Figure 7.2 (a) provides a DVR of the unprocessed EM data set for comparison with the filtered

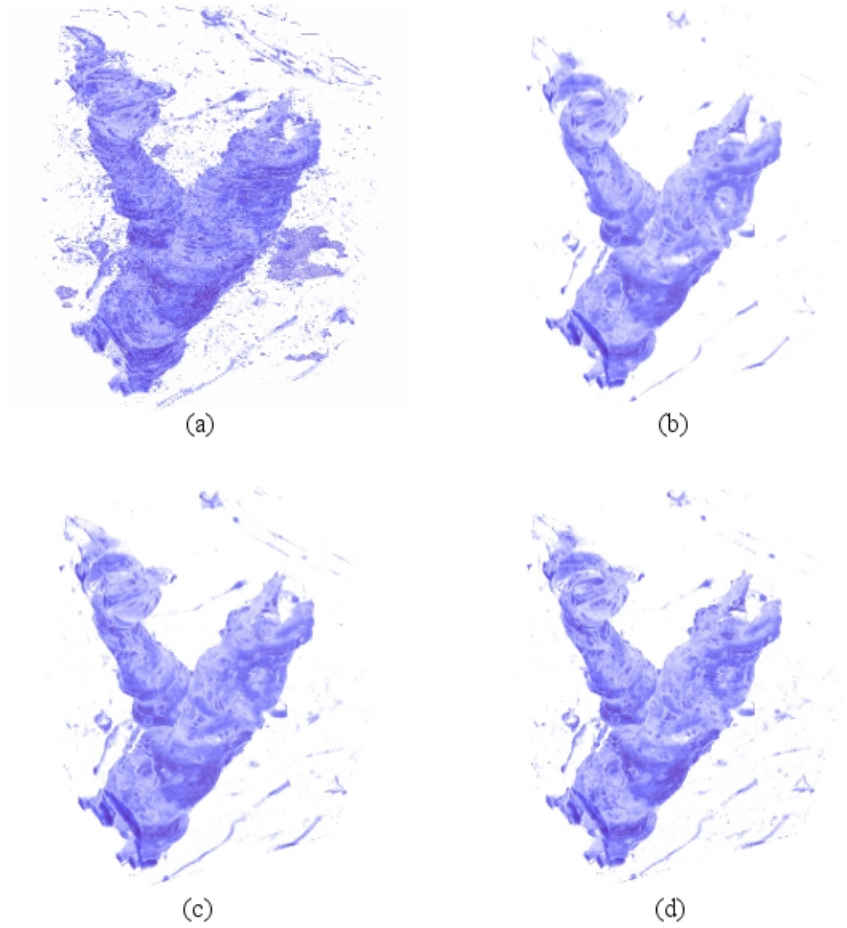


**Figure 7.2:** *3D DVR of a filtered* $256 \times 256 \times 159$ *voxel EM data set. (a) is the DVR of the data set. (b)-(d) DVR of the same EM data set from the same point of view using the same transfer function but different filters. In (b) the volume was filtered by the average filter, in (c) the Gaussian filter was used and (d) was filtered using the bilateral filter. The kernel size for each filter is* $5 \times 5 \times 5$ *voxels.*

ones. Figures 7.2 (b)-(d) show the same EM data set from the same point of view but preprocessed using our filtering module. In (b) the average filter was used for preprocessing, in (c) the Gaussian filter was applied and (d) used the bilateral filter. Every filter kernels had a size of $5 \times 5 \times 5$ voxels. Figure 7.1 and 7.2 show that the desired noise reduction is obtained and the entire volume appears smoother and less grainy.

We further show that our filtering module can be combined with other image-processing techniques. We tested our module in combination with the edge detection and edge highlighting method proposed by Jeong et. al. in [JBH+09]. Figure 7.3

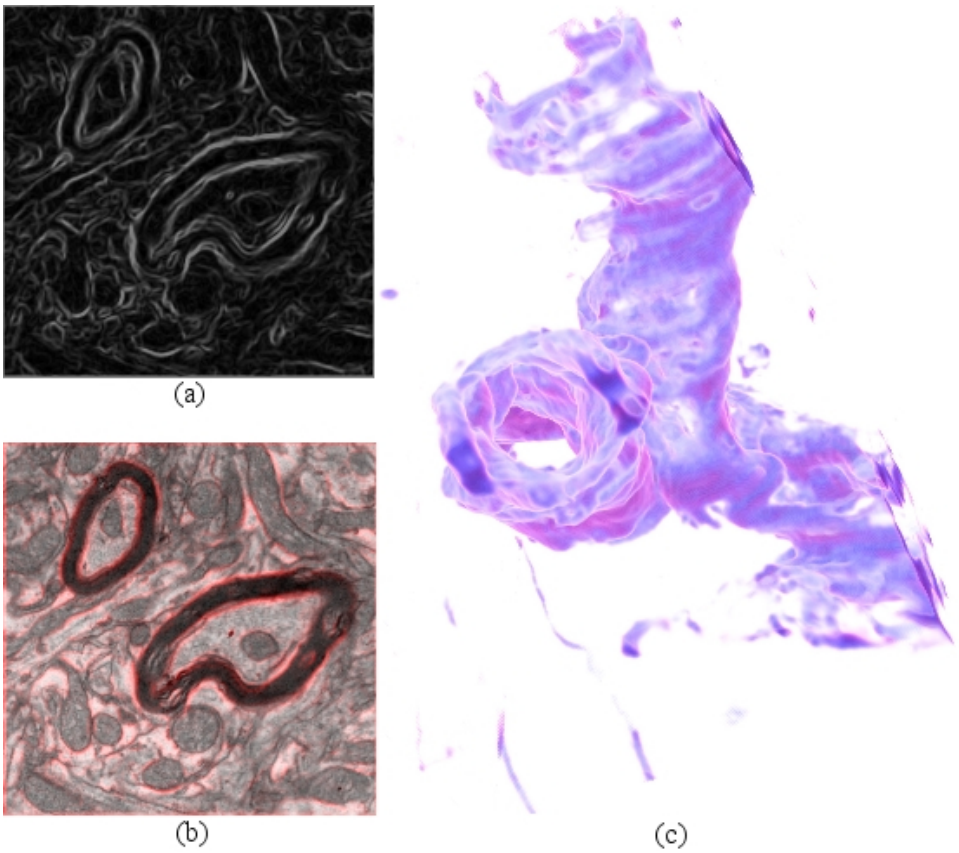


**Figure 7.3:** *Filtering combined with edge detection to highlight boundaries of structures within the volume. (a) Edge detection is applied to a slice of the volume. (b) Enhancement of image structures imposing the edges onto the original slice of the volume. (c) DVR where the edge values are used to enhance the boundaries of whole structures within the volume.*

(a) shows edge detection being applied to a volume slice that is filtered using a $7 \times 7 \times 1$ Gaussian blur. In Figure 7.3 (b) these edges are superimposed on the original volume slice. Superimposing the edges onto the volume slice highlights the most prominent boundaries of the image structures. Figure 7.3(c) shows a DVR of the entire EM data set using a 1D transfer function in combination with the highlighted edges.

## 7.2 Results - Picking

We start this section by evaluating the startup time of the HVR Framework when picking is used. The startup time consists of the time needed to construct the volume

| Dimensions (WxHxD) | Bricks Total | Volume Pyramid Generation | Histogram Processing Time | Total Startup Time | FPS |
|---|---|---|---|---|---|
| $256 \times 256 \times 159$ | 21632 | 1.448s | 62.832s | 64.280s | 30 |
| | 21632 | 1.448s | 0.002s | 1.450s | 30 |
| 1000x750x159 | 240000 | 7.256s | 725.338s | 732.594s | 17 |
| | 240000 | 7.256s | 0.023s | 7.279s | 17 |
| 512x512x164 | 89232 | 6.811s | 257.848s | 264.659s | 21 |
| | 89232 | 6.811s | 0.006s | 6.817s | 21 |

**Table 7.2:** *Total startup time for data sets. Every odd row shows the startup time when the multiresolution histograms have to be calculated. Every even row shows the startup time when they can be loaded from local memory. The columns from left to right represent the dimensions of the test data set showing width, height, depth and size, the total amount of bricks and thus the total amount of multiresolution histograms, the generation time of the volume pyramid, the generation time of all multiresolution histograms, the total startup time and the frame rate.*

pyramid and the time needed to calculate the multiresolution histograms for all the bricks in the volume. Table 7.2 shows the startup times of different volumes at different dimensions, when they are loaded into the HVR Framework for the first time. At this point, the multiresolution histograms are not yet stored locally and have to be computed. Note that the volume generation time as well as the multiresolution histogram generation time depend on the volume size. The average calculation time of a single multiresolution histogram is $3ms$. The greater part of this time is used for copying data between GPU and CPU. This is necessary since the VE module is implemented on the CPU thus needs the multiresolution histograms to be present for the CPU as well. Depending on the number of bricks, $3ms$ calculation time can lead to minutes of startup time. The caching-algorithm mentioned in Subsection 6.2.2.3 reduces the time to setup the multiresolution histograms from several seconds to milliseconds. Table 7.2 also shows the startup times of volumes where the histograms can be loaded from a local cache instead of being calculated. This caching method achieves a speedup of up to 100 times. In the remainder of this section, we present the qualitative results of the multiresolution histogram picking method. We provide screen shots for comparison and address certain issues with this method. The noise and artifact reduction capability of our VE module is demonstrated in Figure 7.4. Figure 7.4 (a) shows an unprocessed EM data set with a size of $1000 \times 750 \times 159$ voxels. Noise and artifacts make it difficult do distinguish between relevant and irrelevant structures of the volume. Especially the artifacts

in the top left corner of this image are easily mistaken for the myelin sheaths of axons. Figure 7.4 (b) demonstrates the noise and artifact removal capability of our VE module. Although the myelin sheaths of the axons are thinned out, these axons are still visible and traceable while noise and artifacts have been removed. It can however not be guaranteed, that only non relevant data have been removed. There were ten bricks picked directly from the DVR of the EM data set to achieve this. Figure 7.5 shows the influence that picking of multiple bricks with a low accuracy-threshold has on the DVR of an EM data set. The lower the accuracy threshold, the lower the probability of considering two bricks a match. Picking more bricks increases this probability while avoiding the incorporation of unwanted structures to the final rendering. The left column in Figure 7.5 shows one picked brick for the entire EM data set. The right column shows the same EM data set but with three picked bricks. Figures 7.5 (a)(b) show the selection of bricks in the fully opaque EM data set. Figures 7.5 (c)(d) show the rendering of this selection when its visibility is inverted. Figures 7.5(e)(f) shows DVRs of Figures 7.5(c)(d) using a transfer function to enhance the axon's boundary to sub-voxel accuracy. When we started investigating multiresolution histogram picking, we hoped that with one click into the EM data set all structures that we considered to be similar will remain visible while all the others disappear. This desired behavior of our VE module is partially achieved when picking prominent structures with large regions (close to the size of a brick) of uniform intensities. As it can be seen in Figures 7.5 and 7.4, multiresolution histogram picking is best suited for finding prominent structures with strong features, like nearly black myelin sheaths of axons in an otherwise grayish volume. When it comes to smaller, less uniform structures like cells with no myelin sheaths, the VE module returns fewer matching bricks. This is not due to the lack of accuracy of the algorithm but the contrary. Multiresolution histograms encode the intensities within the brick. In order to be considered as a matching pair, two bricks have to have a similar composition. Dividing the volume into equally sized bricks introduces the problem that correlating structures of the volume end up in different bricks. However, for knowing which structures correlate, we would need presegmented data. This would be in direct conflict with our goal of providing an out of core solution for our noise reduction problem. Parts of the same structure ending up in different bricks decreases the probability of finding two similar bricks in EM data sets. Increasing the distance-threshold increases the probability of finding bricks but also introduces false matches. As an alternative to increasing the distance threshold, more bricks can be picked with a lower threshold. This results in a greater

probability of finding matching bricks without decreased accuracy. Noise removal, however, can become a tiresome task. Some false matches do not result from the threshold being chosen too high but from wrong user expectations. When picking is used on a fully opaque volume, only parts of a brick are visible. The underlying composition of the bricks are occlude to the user. When, for example, picking a brick that appears to be black, it may as well be black and white. Therefore, when the user picks such a seemingly black brick, he might be surprised when almost white bricks are considered a match.

As a consequence of the limitations our VE module encountered with EM data sets, we investigated its use for less complex data sets like an MR data set of a human head. Figure 7.6 shows the results of this investigation. We used our algorithm to peel away the skull and skin from the head in this MR data set to uncover the brain. We chose this task because brain and skin tissue have similar intensities which makes this operation difficult with standard 1D transfer function design. Although our algorithm performed the desired task, evaluating the usefulness of our results has to be done by domain experts and lies beyond the scope of this thesis. In general, multiresolution histogram picking cuts out prominent structures of a volume. When used together with DVR, the volume appears cleaner and less noisy. Because of wrong matches and wrong user expectations, important structures may be thinned out or get lost. Figure 7.7 shows a comparison between the VE module and the filtering module. Figure 7.7 (a) shows a DVR of an EM data set being preprocessed with a Gaussian filter with filter size $3 \times 3 \times 3$. Figure 7.7 (b) shows a DVR of the same EM data set being preprocessed using our VE module. This image was achieved using 6 picked bricks. The same transfer function was used for both images. The advantage of multiresolution histogram picking lies in the fact that it can provide a significant data-reduction. This is why we consider it to be suitable as a preprocessing step before automated or semi-automated segmentation since less data needs to be processed.

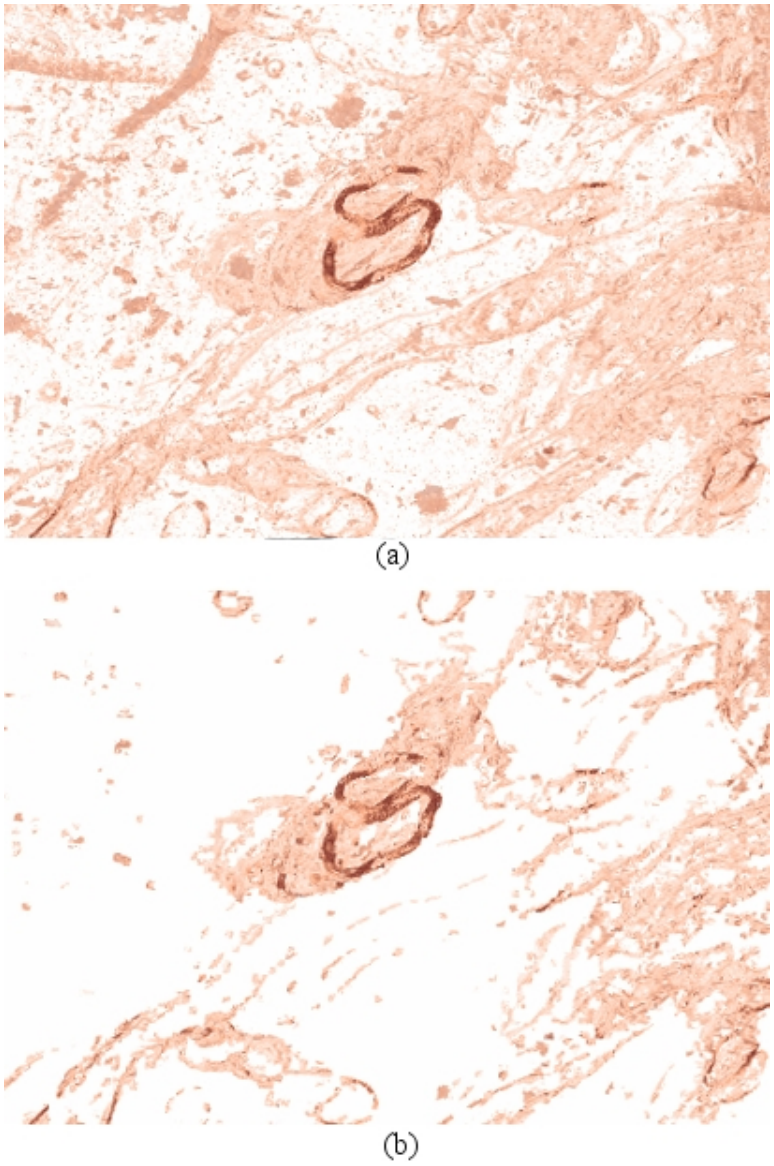


**Figure 7.4:** *Demonstration of noise and artifact reduction capability of picking. Image (a) shows the original unprocessed DVR of an EM data set. (b) demonstrates the result achieved with our picking algorithm. Although the axons in (b) are thinned out, they are still traceable while noise and artifacts have been removed.*

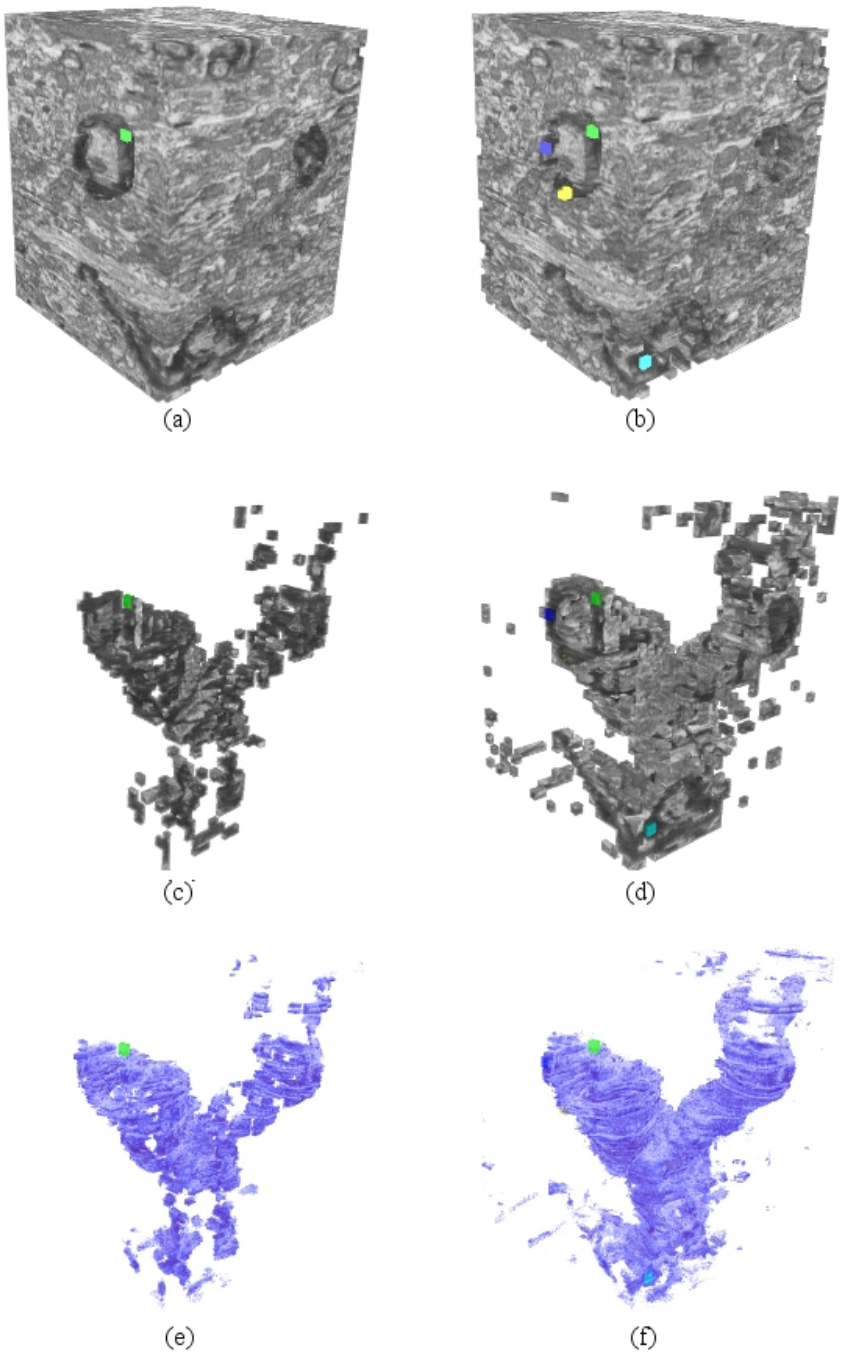


**Figure 7.5:** *Influence of picking multiple bricks with a low accuracy threshold. Left column: One picked brick for the entire EM data set. Right column: Three picked bricks. (a)(b) Selection of the bricks in the fully opaque EM data set. (c)(d) Rendering of the selection by inverting visible and non-visible bricks. (e)(f) DVRs of (c)(d).*

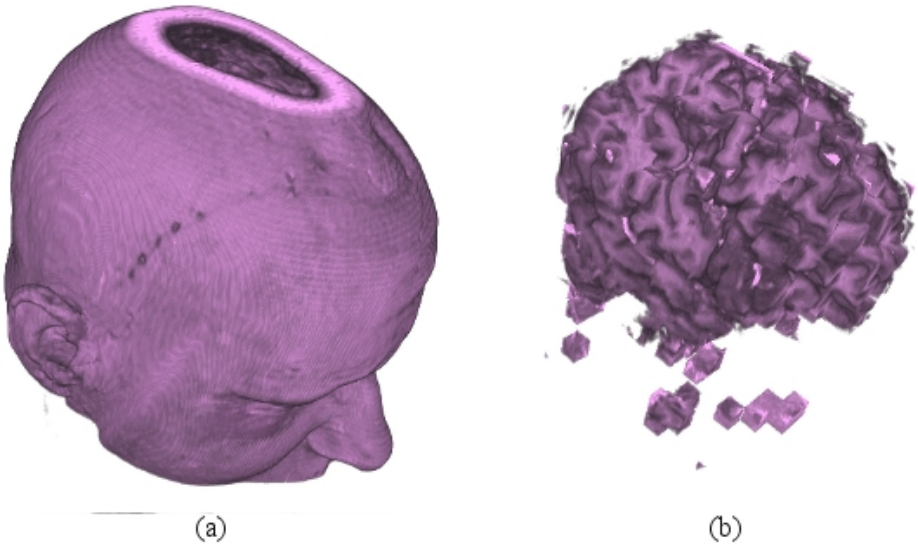


**Figure 7.6:** *Picking algorithm used on an MR data set of a human head to uncover the brain from the surrounding skull and skin. (a) DVR of the unprocessed MR data set. (b) Uncovered brain by using our picking algorithm.*

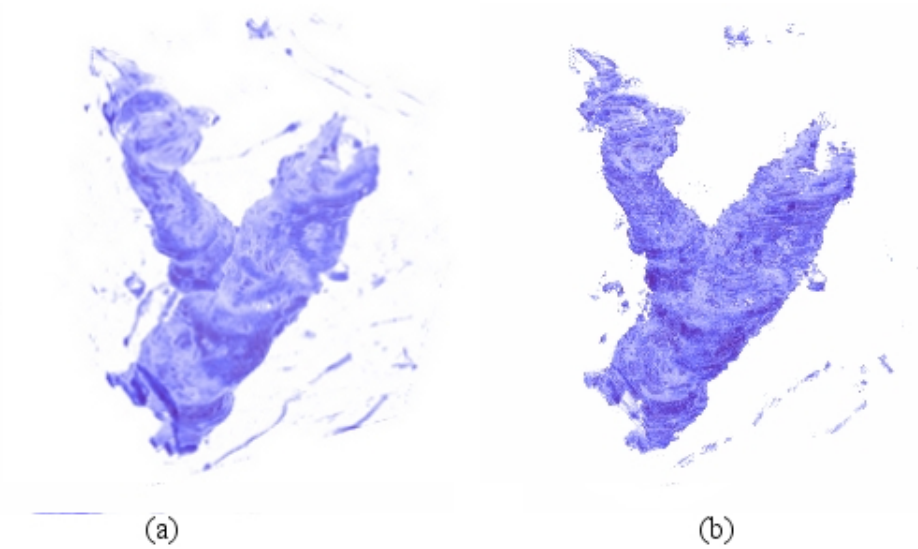


**Figure 7.7:** *Comparison between filtering module (a) and VE module (b). (a) DVR obtained by filtering an EM data set with a* $3 \times 3 \times 3$ *Gaussian filter. (b) Visualization after picking 6 bricks to select the axon. The same transfer function was used for both images.*

# Chapter 8

# Summary

*Try as hard as we may for perfection, the net result of our labors is an amazing variety of imperfectness. We are surprised at our own versatility in being able to fail in so many different ways.*
*-Samuel McChord Crothers*

Connectomics is an emerging area of neuroscience that is concerned with understanding the neural algorithms embedded in the neural circuits of the brain by tracking neurons and studying their connections. The most prominent feature to track in neurons are the myelin sheaths of their axons which possess a very low density to support the processing of electrical signals. Myelin sheaths enclose the axon and are larger and thicker then regular cell membranes. Their size and uniform area of low density makes these myelin sheaths suited for tracking. From all the available scanning technologies only electron microscopy (EM) can provide sufficient scanning resolutions in order to identify neural processes [JBH+09]. EM data sets are of three-dimensional nature and could therefore be considered ideal for DVR [DCH88]. Researchers however rely on 2D slices to track the neural connections [JBH+09]. The reasons not to use DVR for rendering neural connections are the huge data set sizes of EM data sets (up to the terabyte scale), and the fact that EM data sets suffer from bad signal-to-noise ratio and artifacts introduced to the data set during the sectioning and digital reconstruction process of the scanned specimen. The noise and the artifacts present in EM data sets complicate DVR of these data sets because the noise, the artifacts, and the myelin sheaths of the axons

we want to track have the same intensities. DVR in combination with 1D transfer functions is not suited for visualizing these axons because 1D transfer functions can not distinguish noise and artifacts from actual data when they share a common intensity. In the scope of this thesis we investigated two approaches on removing noise and artifacts from EM data sets so that DVR in combination with 1D transfer functions can be used to visualize axons in neural tissue. The main goal of this thesis was to develop extensions to the HVR framework that generally allow noise and artifact reduction on volumetric data sets and which can be used to increase the visual quality of DVRs of EM data sets. To accomplish this goal we investigated two different approaches for noise- and artifact reduction.

The first approach we investigated was to see whether smoothing the EM data set is suitable for achieving noise- and artifact reduction. For this reason we developed an interactive, on-the-fly filtering framework that allows a user to filter even very large volume data set with 3D filter kernels. For comparison, we implemented an average filter, a Gaussian filter and a bilateral filter. The extent of each filter kernel are adjustable by the user. We wanted the users to interactively explore and compare different filter types and filter kernels. Instead of pre-filtering the data set and providing only this pre-filtered data set for exploration, we filter each data set on-the-fly with the filter types and kernel sizes chosen by the user. To perform this task at interactive frame rates, our filtering framework uses NVIDIA's CUDA, a general purpose GPU computing API to perform the filtering operations in parallel on the GPU. Besides choosing filter type and kernel size and adjusting the transfer-function, the user does not have to control the visibility of certain structures in the data set. To allow the user to be more selective when it comes to the visibility of certain structures within neural tissue, the second approach we investigated is a user driven one. The result of this investigation is our Volume Exploration (VE) framework that allows a user to define for himself, which structures in an EM data set he considers to be noise, artifacts or of interest to him. To account for the large number of possible structures within an EM data set, we divide the data set into small equally-sized bricks (e.g. $10 \times 10 \times 10$ voxels), which can be selected and compared to each other. These bricks are selected by clicking on the DVR of the EM data set. Thus, only bricks that are not fully occluded by others can be selected. By selecting a brick, the user indicates that he wants bricks with similar structures to be found in the EM data set. When a brick is selected, all other bricks from the EM data set are compared to the selected one and if considered similar, the

respective bricks are marked as such. This procedure is repeatable. Every time a brick is selected, similar bricks are marked from the EM data set and are added to the final selection. Depending on the users preferences, the marked bricks can either be interpreted to be uninteresting areas of the volume and being blended out, or they can be seen as interesting areas and all other bricks are blended out. We use multiresolution histograms to compare one brick to another. Multiresolution histograms are formed by computing one intensity histogram for each resolution step of the image pyramid of an image. In terms of our bricks, the multiresolution histograms are computed using 5 decreasing resolution steps of a brick and instead of an image-pyramid, we use the volume-pyramid of the respective brick. There is exactly one multiresolution histogram for each brick. We consider multiresolution histograms ideal for finding similar bricks because they do not only encode the statistical intensity distribution in a brick but, due to the inclusion of multiple resolution levels, they encode spatial information as well. The distance between two multiresolution histograms is calculated by the L1 distance between the two feature vectors that are formed by combining cumulative histograms of consecutive intensity histograms. This distance represents the similarity of two multiresolution histograms and by comparing it to a user set similarity threshold, a brick is either marked as similar or dismissed.

## 8.1 Conclusion

Within the scope of this thesis we have come to the conclusion, that both methods we investigated are suitable for removing noise and artifacts in EM data sets. Both techniques achieve interactive frame rates. The multiresolution histogram matching method however needs more startup time than the filtering method when executed the first time on a data set. It also needs more disk space when storing the multiresolution histograms. The filtering method, on the other hand, has a lower frame rate because the EM data set is filtered on-the-fly for each frame. In terms of noise and artifact reduction, the results of smoothing an EM data set using our filtering framework depend on filter type and filter kernel size. The larger the filter kernel, the more noise is removed but relevant details smaller than the filter kernel vanish as well. The larger the filter kernel, the blurrier the borders of the myelin sheaths of the axons get, but we can adjust the kernel size to match the resolution of the data. This results in a perceivable increase in size when observing a blurred axon

in a DVR. This effect can be opposed by changing the filter type or adjusting the transfer-function. Bilateral filters aim at preserving borders while blurring uniform areas of an image. This leads to the observation that average- and Gaussian filter produce a stronger noise reduction in the data set while the bilateral filtering on the other hand reaches a smaller level of noise reduction but keeps the borders of the myelin sheaths intact. The main problem with our on-the-fly filtering approach occurs when a volume is filtered that even with our block-cache barely fits in the memory of the graphics card. Since our filtering approach needs to store the visible and filtered bricks in the memory of the graphics card as well, it can run out of memory. As a consequence to that case, not all visible bricks are filtered.

The multiresolution histogram matching technique is characterized by the high accuracy of the matching algorithm. High accuracy in this case means that, depending on the similarity threshold, only very similar bricks are marked after picking. When picking prominent structures with uniform intensities or strong contrasts within a brick the chance of finding similar bricks is higher than when picking a structure with noise or many different intensities. Finding similar bricks in plain tissue data that does not contain any axons is aggravated by the fact that the EM data set is divided into equally sized bricks which do not consider any structural information of the EM data set when being formed. The EM data set and thus the structures within it are divided randomly which makes finding similar bricks even harder. Another aspect we want to address in this conclusion is the issue that users do not always see all sides of a brick before picking because the bricks may be occluded by neighboring ones. This may lead to a user expecting a result that is not met by the actual result of the matching algorithm. The main problem with our VE module results from the issues stated above. Our matching algorithm finds bricks with similar intensity distributions which can be located anywhere within the volume. It is not guaranteed that the bricks marked as similar by the matching algorithm are the bricks the user had in mind when selecting a certain brick. Despite this issues, the multiresolution histogram matching method is suitable for extracting the myelin sheaths of the axons from noisy EM data sets which increases the visual quality of the DVR of the EM data set.

## 8.2 Future Work

We see the potential for the usage of multiresolution histogram matching going further than selecting prominent and uniform structures in EM data sets. The sensitivity of this method to small intensity and structural changes in any data set makes it capable of matching small and subtle structures. To achieve this, the issues of this technique presented in this thesis have to be addressed. When multiresolution histogram picking is used together with user input to select and find certain structures of a volume, one will have to address the issue of getting less matches then desired or expected. The solution presented in this thesis, namely to increase the similarity threshold, works for prominent and uniform structures in a data set. Increasing this intensity threshold to find small structures or subtle changes in intensity results in false matches. To provide the user with confidence in using our VE framework, it has to be ensured that the matches a user gets correspond to his expectations. This can either be done by increasing the number of positive matches while decreasing false ones, or by eliminating the reasons for wrong expectations. Since wrong user expectations in the context of our VE framework mostly occur due to the lack of visible information on a bricks 3D structure, a way has to be found to show the 3D structure of a brick that goes beyond simply displaying the multiresolution histogram. To increase the number of positive matches while at the same time maintaining a low similarity threshold, we propose to investigate the effects a flexible, user-defined brick-size has on the number of matches. Extending the idea of user defined brick sizes, the location and size of picked bricks should be arbitrary as well. This way, the user can fully enclose the structures he wants to be found. This improvement however requires a possibility to match structures of different brick sizes against each other. To improve the rendering quality and simplify the usage of our ray caster when rendering the results of the multiresolution histogram matching, we want to investigate how our histograms can be used to simplify the transfer function design like it was proposed by Lundström et al. [LLY05, LYL+06].

# Acknowledgments

*A lie is just a great story that someone ruined with the truth!*
*- Barney Stinson*

I dedicate this thesis to my parents who encouraged and unconditionally supported me throughout my educational career. Without this support, finally completing this thesis and my studies would have been a great deal harder. I also would like to thank my brother Daniel for putting up with my continuously brabbeling over my thesis and for providing the occasional kick in the a** ;). Id further like to thank my good friend Ray for proof reading my thesis.

Special thanks to Johanna Beyer and the Team from the VRVIS Vienna for their supervision, ideas and motivation throughout this thesis. I'd further like to thank them for proof reading and for providing final suggestions for the final versions of this thesis.

Further, I thank Eduard Gröller, the Meister, for making this thesis possible and finally the TU Vienna for providing excellent studying conditions throughout the whole master program.

# Bibliography


[AAR04] S. Agarwal, A. Awan, and D. Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, 2004.

[AHP04] T. Ahonen, A. Hadid, and M. Pietikainen. Face recognition with local binary patterns. *Lecture Notes in Computer Science*, pages 469–481, 2004.

[BD06] K.L. Briggman and W. Denk. Towards neural circuit reconstruction with volume electron microscopy techniques. *Current opinion in neurobiology*, 16(5):562–570, 2006.

[BFH+04] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2004 Papers: Los Angeles, California*, pages 777–786, 2004.

[BHMF08] J. Beyer, M. Hadwiger, T. Möller, and L. Fritz. Smooth Mixed-Resolution GPU Volume Rendering. In *IEEE/EG International Symposium on Volume and Point-Based Graphics*, pages 163–170, 2008.

[BIP99] C. Bajaj, I. Ihm, and S. Park. Making 3D textures practical. In *Proceedings of Pacific Graphics 99*, pages 259–268, 1999.

[Bro92] L.G. Brown. A survey of image registration techniques. *ACM computing surveys (CSUR)*, 24(4):376, 1992.

[BS98] V. Braitenberg and A. Schuez. *Cortex: statistics and geometry of neuronal connectivity*. Springer Berlin, 1998.

[Car84] M. Carlotto. Texture classification based on hypothesis testing approach. In *Proc. Intl Japanese Conf. Pattern Recognition*, pages 93–96, 1984.

[CSW+85] M. Chalfie, JE Sulston, JG White, E. Southgate, JN Thomson, and S. Brenner. The neural circuit for touch sensitivity in Caenorhabditis elegans. *Journal of Neuroscience*, 5(4):956–964, 1985.

[DCH88] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 65–74. ACM New York, NY, USA, 1988.

[DD02] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 257–266. ACM, 2002.

[DPVN08] T. Deselaers, R. Paredes, E. Vidal, and H. Ney. Learning Weighted Distances for Relevance Feedback in Image Retrieval. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 2008.

[EEH+00] K. Engel, T. Ertl, P. Hastreiter, B. Tomandl, and K. Eberhardt. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of the IEEE Visualization'00*, pages 449–452. IEEE Computer Society Press Los Alamitos, CA, USA, 2000.

[EKE01] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16. ACM New York, NY, USA, 2001.

[FH99] A. Frangakis and R. Hegerl. Nonlinear anisotropic diffusion in three-dimensional electron microscopy. *Scale-Space Theories in Computer Vision*, pages 386–397, 1999.

[FH01] A.S. Frangakis and R. Hegerl. Noise reduction in electron tomographic reconstructions using nonlinear anisotropic diffusion. *Journal of structural biology*, 135(3):239–250, 2001.

[Fia02] J.C. Fiala. Three-dimensional structure of synapses in the brain and on the web. In *2002 World Congress on Computational Intelligence. May*, pages 12–17, 2002.

[FM08] J. Fung and S. Mann. Using graphics devices in reverse: GPU-based Image Processing and Computer Vision. In *2008 IEEE International Conference on Multimedia and Expo*, pages 9–12, 2008.

[Fuk90] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, 1990.

[FVDFH95] J.D. Foley, A. Van Dam, S.K. Feiner, and J.F. Hughes. *Computer graphics: principles and practice*. Addison-Wesley Professional, 1995.

[GNE+03] J. Goldstein, D.E. Newbury, P. Echlin, C.E. Lyman, D.C. Joy, E. Lifshin, LC Sawyer, and J.R. Michael. *Scanning electron microscopy and X-ray microanalysis*. Plenum Pub Corp, 2003.

[Had04] Markus Hadwiger. *High-Quality Visualization and Filtering of Textures and Segmented Volume Data on Consumer Graphics Hardware*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2004.

[Har05] M. Harris. GPGPU: General-purpose computation on GPUs. In *Presentation at the Game Developers Conference*, 2005.

[HGN01a] E. Hadjidemetriou, M.D. Grossberg, and S.K. Nayar. Histogram preserving image transformations. *International Journal of Computer Vision*, 45(1):5–23, 2001.

[HGN01b] E. Hadjidemetriou, MD Grossberg, and SK Nayar. Spatial information in multiresolution histograms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1. IEEE Computer Society; 1999, 2001.

[HGN04] E. Hadjidemetriou, M.D. Grossberg, and S.K. Nayar. Multiresolution histograms and their use for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 831–847, 2004.

[Hum77] R. Hummel. Image enhancement by histogram transformation. *Computer graphics and image processing*, 6(2):184–195, 1977.

[JBH+09] W.K. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R.T. Whitaker. Scalable and Interactive Segmentation and Visualization of Neural Processes in EM Datasets. volume 15, pages 1505–1514. IEEE Visualization, 2009.

[JHK+99] P.T. Johnson, E.J. Halpern, B.S. Kuszyk, D.G. Heath, R.J. Wechsler, L.N. Nazarian, G.A. Gardiner, D.C. Levin, and E.K. Fishman. Renal Artery Stenosis: CT Angiography Comparison of Real-time Volume-rendering and Maximum Intensity Projection Algorithms1. *Radiology*, 211(2):337, 1999.

[KLK+97] Y.T. Kim, M.H. Lee, H.I. Ko, D.I. Song, W.J. Hwang, B.Y. Ye, S. Kim, Y. Kim, K. Yim, and H. Chung. Contrast enhancement using brightness preserving bi-histogram equalization. *IEEE Transactions on Consumer Electronics*, 43(1):1–8, 1997.

[KVH84] J.T. Kajiya and B.P. Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH Computer Graphics*, 18(3):165–174, 1984.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003*, pages 287–292. IEEE Computer Society, 2003.

[Lev90] M. Levoy. Volume rendering. *IEEE Computer graphics and Applications*, 10(2):33–40, 1990.

[LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994.

[LLY05] C. Lundström, P. Ljung, and A. Ynnerman. Extending and simplifying transfer function design in medical volume rendering using local histograms. *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 263–270, 2005.

[LVB+93] M. Lades, J.C. Vorbrueggen, J.M. Buhmann, J. Lange, C. Malsburg, R.P. Wuertz, and W. Konen. Distortion invariant object recognition

in the dynamic link architecture. *IEEE Transactions on computers*, 42(3):300–311, 1993.

[LW03] X. Liu and D.L. Wang. Texture classification using spectral histograms. *IEEE Transactions on Image Processing*, 12(6):661–670, 2003.

[LYL+06] C. Lundström, A. Ynnerman, P. Ljung, A. Persson, and H. Knutsson. The alpha-histogram: Using spatial coherence to enhance histograms and transfer function design. In *Proceedings Eurographics/IEEE-VGTC Symposium on Visualization*, pages 227–234, 2006.

[Mar02] K.A.C. Martin. Microcircuits in visual cortex. *Current opinion in neurobiology*, 12(4):418–425, 2002.

[Max95] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[MHB+00] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90, 2000.

[MRB+08] A. Maximo, S. Ribeiro, C. Bentes, A. Oliveira, and R. Farias. Memory efficient gpu-based ray casting for unstructured volume rendering. In *IEEE/EG Int. Symp. Volume and Point-Based Graph*, pages 55–62, 2008.

[MS00] S. Michael and J. Sammon. Practical Algorithms for Image Analysis, 2000.

[MV98] J.B.A. Maintz and M.A. Viergever. A survey of medical image registration. *Medical image analysis*, 2(1):1–36, 1998.

[NVI10a] Corporation NVIDIA. Nvidia cuda compute unified device architecture, programming guide 2.0, 2010.

[NVI10b] Corporation NVIDIA. Nvidia cuda technical training, volume 1, 2010.

[OABB85] J.M. Ogden, E.H. Adelson, J.R. Bergen, and P.J. Burt. Pyramid-based computer graphics. *RCA Engineer*, 30(5):4–15, 1985.

[OLG+07] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer Graphics Forum*, volume 26, pages 80–113, 2007.

[OPM02] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971, 2002.

[Opp02] R. Oppermann. User interface design. *Handbook on information technologies for education and training*, pages 233–248, 2002.

[Pit00] I. Pitas. *Digital image processing algorithms and applications*. Wiley-interscience, 2000.

[Qur05] Shehrzad Qureshi. *Embedded Image Processing on the TMS320C6000 DSP: Examples in Code Composer Studio and MATLAB*. Springer, 2005.

[RH99] T. Randen and J.H. Husoy. Filtering for texture classification: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):291–310, 1999.

[Rip08] B.D. Ripley. *Pattern recognition and neural networks*. Cambridge Univ Press, 2008.

[RMB+07] S. Ribeiro, A. Maximo, C. Bentes, A. Oliveira, and R. Farias. Memory-aware and efficient ray-casting algorithm. In *Computer Graphics and Image Processing, 2007. SIBGRAPI 2007. XX Brazilian Symposium on*, pages 147–154, 2007.

[RPSC99] H. Ray, H. Pfister, D. Silver, and T.A. Cook. Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):210–223, 1999.

[Rus06] John C. Russ. *The Image Processing Handbook, Sixth Edition*. CRC Press, 2006.

[RV06] D. Ruijters and A. Vilanova. Optimizing GPU volume rendering. *Journal of WSCG*, 14:9–16, 2006.

[SB91] S. Sarkar and K.L. Boyer. On optimal infinite impulse response edge detection filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(11):1154–1171, 1991.

[Sch05] H. Scharsach. Advanced GPU raycasting. *Proceedings of CESCG*, pages 67–76, 2005.

[SOS00] M. Seul, L. O'Gorman, and M.J. Sammon. *Practical algorithms for image analysis: description, examples, and code*. Cambridge Univ Press, 2000.

[SPHC02] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. *ACM SIGARCH Computer Architecture News*, 30(5):45–57, 2002.

[STR05] O. Sporns, G. Tononi, and K. Rolf. The Human Connectome: A Structural Description of the Human Brain. *PLoS Computational Biology*, 1(4), 2005.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, volume 846, 1998.

[TP86] V. Torre and T. Poggio. On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):147–163, 1986.

[Umb05] Scott E Umbaugh. *Computer Imaging: Digital Image Analysis and Processing*. CRC Press, 2005.

[VVYV98] L.J. Van Vliet, I.T. Young, and P.W. Verbeek. Recursive Gaussian derivative filters. In *International Conference on Pattern Recognition*, volume 14, pages 509–514, 1998.

[WC09] D.B. Williams and C.B. Carter. *Transmission electron microscopy: a textbook for materials science*. Springer Verlag, 2009.

[Wil83] L. Williams. Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11. ACM New York, NY, USA, 1983.

[WM98] F.M. Waltz and J.W.V. Miller. An efficient algorithm for Gaussian blur using finite-state machines. In *SPIE Conf. on Machine Vision Systems for Inspection and Metrology VII*, 1998.

[WVW94] O. Wilson, A. VanGelder, and J. Wilhelms. Direct Volume Rendering via 3D Textures. 1994.

[ZB09] G. Zachmann and D. Bartz. Visual computing for medical diagnosis and treatment. *Computers & Graphics*, 33:554–565, 2009.

[ZvBG01] M. Zwicker, J. van Baar, and M. Gross. EWA volume splatting. In *Proceedings of IEEE Visualization'01*, pages 29–36. IEEE Computer Society Washington, DC, USA, 2001.