Jül-4365

Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)

GPU-accelerated Segmentation of high-resolution Human Brain Images acquired with Polarized Light Imaging

Anna Maria Westhoff

GPU-accelerated Segmentation of high-resolution Human Brain Images acquired with Polarized Light Imaging

Anna Maria Westhoff

Berichte des Forschungszentrums Jülich; 4365
ISSN 0944-2952
Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)
Jül-4365

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL)
unter http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

High-resolution three-dimensional polarized light imaging (PLI) is an approach pursued by the Institute of Neuroscience and Medicine at Forschungszentrum Jülich to map nerve fibers and their pathways in human brains. Sections of cut post-mortem brains are imaged with a microscopic device. A section is moved during the imaging within the microscope so that a mosaic of about 30×30 image tiles is created for a gross histological human brain section. These up to 900 tiles per section and about 1500 sections in total have to be handled in the 3D reconstruction of the brain. A way to accelerate this process is a previous segmentation of the image tiles which leads to black-and-white masks marking the brain and background pixels of the original tiles. Hence, all non-brain parts of the tiles can be ignored during the reconstruction.

A region growing segmentation is developed and implemented for the PLI data. The challenge to adapt this algorithm to the given dataset is to automatize the choice of seeds needed as starting points for the growing process. Therefore, an automated method of seed determination has to be developed. It uses statistics of the whole brain based on the joint intensity histogram. This approach leads to a minimal fixed amount of required manual input which is independent of the number of image tiles to be segmented. The software is parallelized for the GPU cluster JUDGE, i.e. it combines two levels of parallelism, namely a multi-core implementation and the data parallel execution of appropriate subtasks on a GPU. This leads to a well scaling application that achieves the expected segmentation results.

Zusammenfassung

Die Technik des hochauflösenden, dreidimensionalen Polarized Light Imaging (PLI) ist ein am Institut für Neurowissenschaften und Medizin des Forschungszentrums Jülich entwickelter Ansatz, um den Verlauf von Nervenfasern im menschlichen Gehirn zu kartografieren. Schnitte eines Post-Mortem-Hirns werden mit einem Mikroskop aufgenommen. Jeder Schnitt wird für die Aufnahme innerhalb des Mikroskop-Aufbaus bewegt, sodass ein Mosaik aus etwa 30×30 Kacheln für einen histologischen Schnitt des menschlichen Gehirns entsteht. Diese bis zu 900 Kacheln pro Schnitt und 1500 Schnitte pro Gehirn müssen während der 3D-Rekonstruktion des Hirns verarbeitet werden. Um dieses Verfahren zu beschleunigen, wird zunächst eine Segmentierung der Kacheln durchgeführt. Diese erzeugt zu jeder Kachel eine Schwarz-Weiß-Maske, in der für jedes Pixel kenntlich gemacht wird, ob es Hirngewebe oder Hintergrund zeigt. Auf diese Weise können in der Rekonstruktionsphase die Bildbereiche ignoriert werden, die kein Gewebe darstellen.

Für die PLI-Daten wurde eine Region Growing Segmentierung entwickelt und implementiert. Die Herausforderung in der Anpassung des Algorithmus besteht darin, die Wahl der Seeds zu automatisieren, die als Startpunkte für den Growing-Prozess dienen. Die entwickelte, automatisierte Seed-Wahl basiert auf Statistiken des gesamten Gehirns, denen das gemeinsame Intensitäts-Histogramm aller Kacheln zugrunde liegt. Dieser Ansatz führt zu einem manuellen Aufwand, der unabhängig ist von der Anzahl der zu verarbeitenden Bilder. Die Software ist parallelisiert für den GPU-Cluster JUDGE. Es werden zwei Ebenen der Parallelisierung kombiniert, zum einen eine Implementierung für ein Mehrprozessor-System, zum anderen eine Daten-parallele Ausführung geeigneter Teilaufgaben auf einer GPU. Dies resultiert in einer gut skalierenden Anwendung, die die erwarteten Segmentierungs-Ergebnisse erzielt.

Contents

1 Introduction **1**

2 Polarized Light Imaging (PLI) **3**
2.1 Motivation for PLI 3
2.2 Preparation of a Brain 3
2.3 Rotating Polarimeter and Imaging 5
2.4 3D Reconstruction of the Polarimeter Images 7
2.5 Requirements for the Segmentation 9

3 Choice of the Algorithm **13**
3.1 Segmentation Methods 13
3.1.1 Thresholding 13
3.1.2 Region-Based Segmentation 13
3.1.3 Boundary-Driven Techniques 15
3.1.4 Classification Methods 15
3.2 Comparison of Segmentation Tools 16
3.2.1 Images used for the Comparison 16
3.2.2 Process of Testing 17
3.2.3 Results and Evaluation 17
3.2.4 Summarizing Evaluation 35

4 Seeded Region Growing Segmentation **39**
4.1 A basic Algorithm 39
4.2 Adaption of Seeded Region Growing 42
4.2.1 Automated Choice of Seeds 42
4.2.2 Similarity Criterion 54
4.2.3 Choice of the Growing Operator 55
4.3 Processing Pipeline 56
4.4 Review of the Requirements 58
4.5 Optimization of the Runtime Behavior 63
4.5.1 Multi-core Application 64
4.5.2 Data Parallelism and GPUs 65
4.5.3 Evaluation of the Parallelization 69

5 Conclusion and Outlook **77**

Bibliography **83**

A JUDGE Configuration **I**

List of Figures

2.1 Preparation Steps required before the Imaging 4
2.2 Measurement Setup of PLI 5
2.3 Rotating Polarimeter 6
2.4 Reconstruction Pipeline 8
2.5 Smooth Edges in Image Tiles 10
2.6 Image Artifacts: Air Bubbles 11
3.1 Counter-Example: Thresholding 14
3.2 Images used for the Tool Comparison 16
3.3 Result: Thresholding (GIMP) 18
3.4 Seeds for Region Growing with Fiji 19
3.5 Result: Region Growing (Fiji) 20
3.6 Result: Region Growing (ITK Connected Threshold) 22
3.7 Result: Region Growing (ITK Neighborhood Connected) part 1 . . 24
3.8 Result: Region Growing (ITK Neighborhood Connected) part 2 . . 25
3.9 Result: Region Growing (ITK Confidence Connected) part 1 26
3.10 Result: Region Growing (ITK Confidence Connected) part 2 27
3.11 Result: Region Growing (ITK Isolated Connected) 28
3.12 Result: Boundary-Driven (MiaLite) 29
3.13 Seeds for ITK-SNAP 30
3.14 Result: Boundary-Driven (ITK-SNAP) 31
3.15 Result: Boundary-Driven (Fiji Level Sets) part 1 32
3.16 Result: Boundary-Driven (Fiji Level Sets) part 2 33
3.17 Training Data for ilastik 33
3.18 Result: Classification Methods (ilastik) 34
4.1 Structure Chart: Seeded Region Growing by Adams and Bischof . . 40
4.2 Joint Histogram of 10 Sections 43
4.3 Structure Chart: Computation of the Joint Histogram 44
4.4 Joint Histogram of 10 Sections with Local Minima 45
4.5 Average Smoothing Filter 47
4.6 Weighted Average Smoothing Filter 47
4.7 Used Smoothing Filter 48
4.8 Initial Brain Measures 49
4.9 Initial Background Measures 50
4.10 Structure Chart: Final Choice of Seeds 51
4.11 Comparison of Initial and Final Brain Measures 53
4.12 Regions connected at a Corner 55
4.13 Processing Pipeline of Seeded Region Growing 57
4.14 Result: Segmentation Masks 59
4.15 Result: Segmentation Masks - Extracted Edges 60
4.16 Structure Chart: Dilation 61

4.17 Result: Effect of the Dilation 62
4.18 Runtime of the Steps on a CPU 63
4.19 Structure Chart: Parallelized Computation of the Joint Histogram . . 64
4.20 CUDA: Grid of Thread Blocks 66
4.21 Speedup: Histogram 70
4.22 Speedup: Seeded Region Growing (CPU) 70
4.23 Communication Structure used on JUDGE 71
4.24 Optimal Size of Thread Block (per Step) 72
4.25 Optimal Size of Thread Block (summed up) 72
4.26 Comparison: Runtime of the Steps on a CPU and GPU 73
4.27 Comparison: Per-Section Runtimes with and without GPUs 74
4.28 Speedup: Seeded Region Growing (GPU) 75

List of Tables

3.1 Start Parameters: ITK Connected Threshold 21
3.2 Start Parameters: ITK Neighborhood Connected 23
3.3 Start Parameters: ITK Confidence Connected 25
3.4 Start Parameters: ITK Isolated Connected 27
3.5 Comparison of the tested Tools 37

Listings

4.1 CUDA Kernel of the Dilation 67
4.2 Invocation of the Dilation Kernel 68

1 Introduction

Perhaps one of the greatest challenges research faces today is the question how the human brain works. A lot of projects are dedicated to this topic like the Helmholtz Portfolio Theme "Supercomputing and Modeling for the Human Brain"[1], the Human Brain Project[2] or the US-American equivalent BRAIN initiative[3]. To answer this question, the anatomical structure of the human brain on the level of single nerve fibers has to be understood. A group of the Institute for Neuroscience and Medicine (INM-1) at Forschungszentrum Jülich has taken on the task to understand the connectivity of brain regions on the one hand and to study neurodegenerative diseases on the other hand. They develop and improve the technique of Polarized Light Imaging (PLI) that allows to study sections of post-mortem brains with a resolution at sub-millimeter scale.

This comparably high resolution causes the problem that the amount of data to be processed digitally is immense. A lot of processing steps of PLI can be accelerated and their results may be improved if this amount of data can be limited to the parts of interest. Therefore these parts of the images are masked out which do not show the brain tissue. This provides additional advantages for visualization issues.

One way to cut out the background pixels of the original images is to use a segmentation. This task of image processing creates a mask defining which pixels of the original image show background and which show the object, so in this case the brain tissue. The topic of this thesis is the development and implementation of a segmentation for brain images acquired with PLI.

The concept of Polarized Light Imaging is outlined in chapter 2. The preparation of the brain tissue, the imaging process and the 3D reconstruction of the images are briefly presented. Afterwards, the requirements and further constraints on the segmentation induced by the PLI image data are worked out in section 2.5.

Based on these requirements, a segmentation approach has to be chosen since there is a bunch of different ways how to segment images. This process is described in chapter 3. Already existing representatives of the most widespread segmentation methods, which are presented in section 3.1, are applied to a small, representative example data set in order to evaluate the fulfilling of the requirements (see section 3.2). Summarizing the results it becomes apparent that the region growing approaches create the comparably most reasonable masks while fulfilling the requirements best. However, it is important to find a way to automatize this algorithm in so far that the amount of manual input is independent of the number of images to be processed since the tool has to deal with hundreds of thousands of images per

[1]http://www.fz-juelich.de/JuBrain/EN/Helmholtz%20Portfolio.html, accessed at 13/06/2013

[2]The Human Brain Project is a European Flagship Project. http://www.humanbrainproject.eu/, accessed at 13/06/2013

[3]Short for Brain Research through Advancing Innovative Neurotechnologies, http://www.nih.gov/science/brain/, accessed at 13/06/2013

human brain.

It figured out that an application based on [AB94] achieved the best results among all tested region growing tools creating reasonable masks. The adaption of this seeded region growing method to the needs of the given PLI brain images is presented in chapter 4. An important task is here to automatize the choice of seeds so that the amount of required manual input is fixed and thus independent of the number of images to be processed. A joint intensity histogram of all brain images is used for this purpose.

After the image processing pipeline has been worked out (see section 4.3), the formerly established requirements are reviewed to prove that the developed and implemented tool fulfills all of them (see section 4.4). The runtime behavior of the tool is analyzed (see section 4.5) because the processing time is the most important constraint that has to be kept in mind besides the requirements. In order to minimize the runtime consumption, an optimal scaling, two-level parallelization is developed for the segmentation. A porting into a multi-core application is installed and followed by a data parallel execution of some selected steps on a GPU.

2 Polarized Light Imaging (PLI)

2.1 Motivation for PLI

Understanding the anatomical structure of the human brain on the level of single nerve fibers is one of the most challenging tasks nowadays. The mapping of the three-dimensional course of nerve fibers in the human brain is important to understand the function of the brain on the one hand and to treat diseases successfully on the other hand. [Bra13]

There are several techniques to image a brain like magnetic resonance imaging which lack the required resolution to extract information on the level of single nerve fibers. Polarized Light Imaging is a technique applied to histological sections of postmortem brains which allows "*the study of brain regions [...] in unprecedented detail*" [AAG+11] with a resolution at sub-millimeter scale. It is based on an optical property of myelin referred to as birefringence which surrounds the axons of nerve fibers.

To capture the images with PLI, the brain is cut into more than 1500 sections using a freezing microtome. Each section is then imaged with a specific polarimeter developed at the INM-1. Afterward the images of the different sections have to be analyzed and reconstructed to gain three-dimensional information about the course of nerve fibers. These different steps are briefly described in the following paragraphs. [AAG+11, PAG+10]

2.2 Preparation of a Brain

Before the imaging can take place, the postmortem brain has to be prepared in some steps which can be seen in figure 2.1.

After the brain has been removed[1], it is fixed in formalin to prevent it from decomposition. Since PLI captures images from sections of the brain, it has then to be prepared for the sectioning. For that purpose it gets cryoprotected. The frozen brain is cut into slices with a thickness of about $70\mu m$. Cryo-sectioning turned out to be the preferred technique to preserve the birefringent myelin structures.

The brain is deformed non-linearly during the sectioning because the gray and white matter react differently to the cutting and mounting on glass slides. Hence a reference of the non-deformed brain is needed for the three-dimensional reconstruction. Therefore a colored image of the surface of the frozen brain block is taken before each cut. These are the so-called blockface images.

[1]The human postmortem brains are from legal body donor programs of collaborating universities in Germany.

The sections are mounted on glass slides, embedded in Aquatex and finally cover-slipped. During this process the tissue is deformed non-linearly once again and individually for each section because the mounting is done manually. Also artifacts like air bubbles are added unintentionally.

Each mounted section is imaged with the PLI technique at two different resolutions. At the standard resolution with a pixel size of $64\mu m \times 64\mu m$ a whole section is captured in a single image. For the more detailed resolution with a pixel size of $1.6\mu m \times 1.6\mu m$, a polarizing microscope captures a mosaic of about 30×30 images for each section. The mosaic tiles are captured with an overlapping of about 30%. [PAG+10, AGK+11]

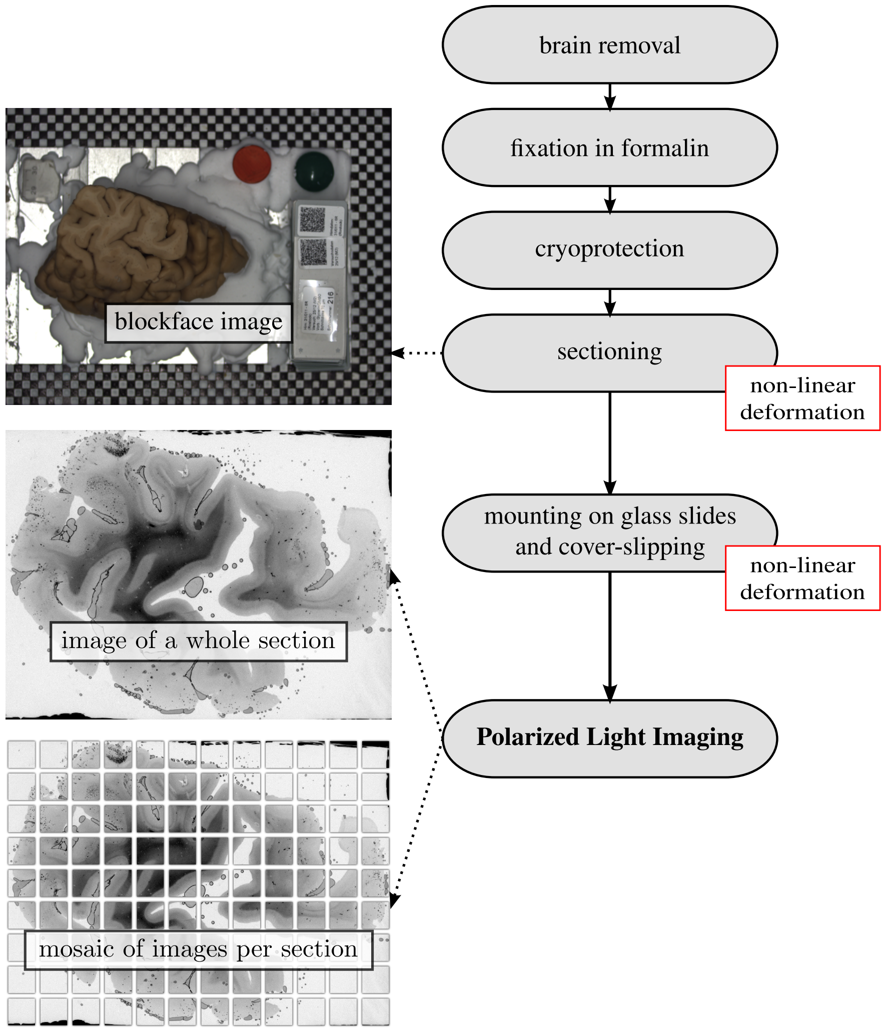


Figure 2.1: Before the brain can be imaged, some preparation steps including the sectioning have to be done first. [PAG+10, AGK+11]

2.3 Rotating Polarimeter and Imaging

The image acquisition of PLI is based on an optical property of the myelin sheaths of nerve fiber axons. The lipid content of the myelin features birefringence whose impact on the transmitted light is measured to extract the course of nerve fibers.

The measurement setup for Polarized Light Imaging is illustrated in figure 2.2. The light emitted from the light source is linearly polarized by a first polarizer before it passes the brain tissue. The axons of the nerve fibers are surrounded by the birefringent myelin. The linearly polarized light interacts with this radially oriented myelin thus inducing a phase shift to the light waves. This way the light becomes elliptically polarized.

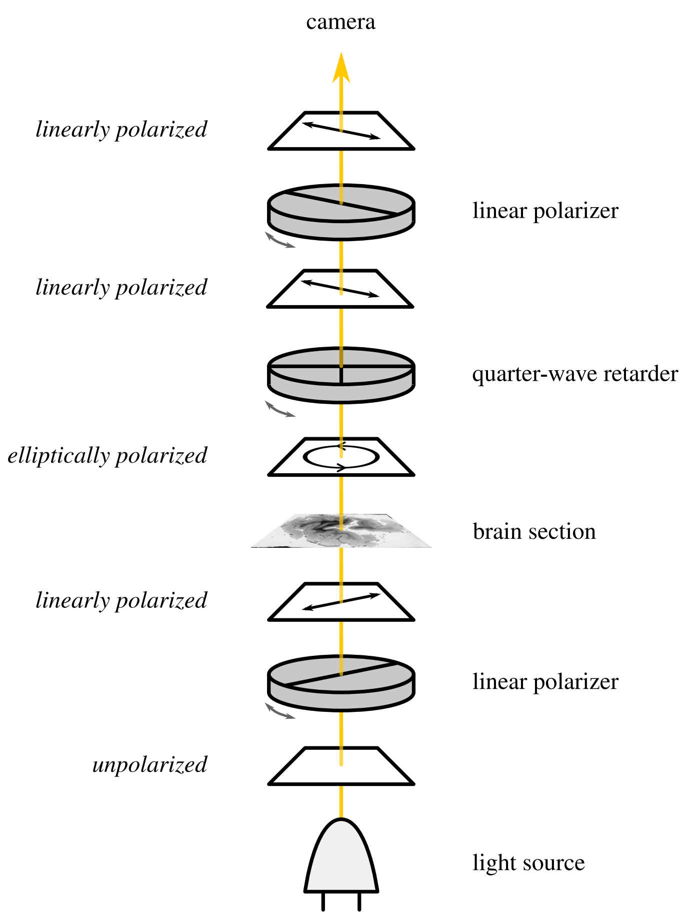


Figure 2.2: To capture the images based on PLI, light is linearly polarized by a first polarizer before it enters the brain section. The birefringent myelin surrounding the nerve fiber axons changes the polarization state into an elliptical one. Afterwards it passes a quarter-wave retarder which changes the polarization state again to linear. The light is then analyzed by a second polarizer whose transmission axis is orthogonal to the one of the first polarizer. The principal axes of the retarder form a 45 degree angle with respect to the transmission axes of the polarizers. [AAG+11, PAG+10]

The light passes a quarter-wave retarder next. Its principal axes form a 45 degree angle with respect to the transmission axis of the polarizer. It gets linearly polarized again leaving the retarder. Before a camera captures the light, it is analyzed by a second polarizer whose transmission axis is perpendicular to the axis of the first one.

The spatial 3D-orientation of the nerve fiber axons can be directly measured from the captured images. It is represented by two angles, the in-plane angle φ describing the component of the fiber orientation within the section and the out-of-section angle α which is the vertical component of the fiber and is also called inclination.

The brain tissue is scanned systematically to gain the principal axes of the nerve fibers and hence the 3D-orientation. Each brain section is imaged 18 times. For each image the polarimeter consisting of the polarizers and the retarder is rotated by ten degrees so that images from 0°, 10°, 20° up to 170° with respect to a reference angle are captured. The images are black and white images containing only intensity information but no colors (see figure 2.3).

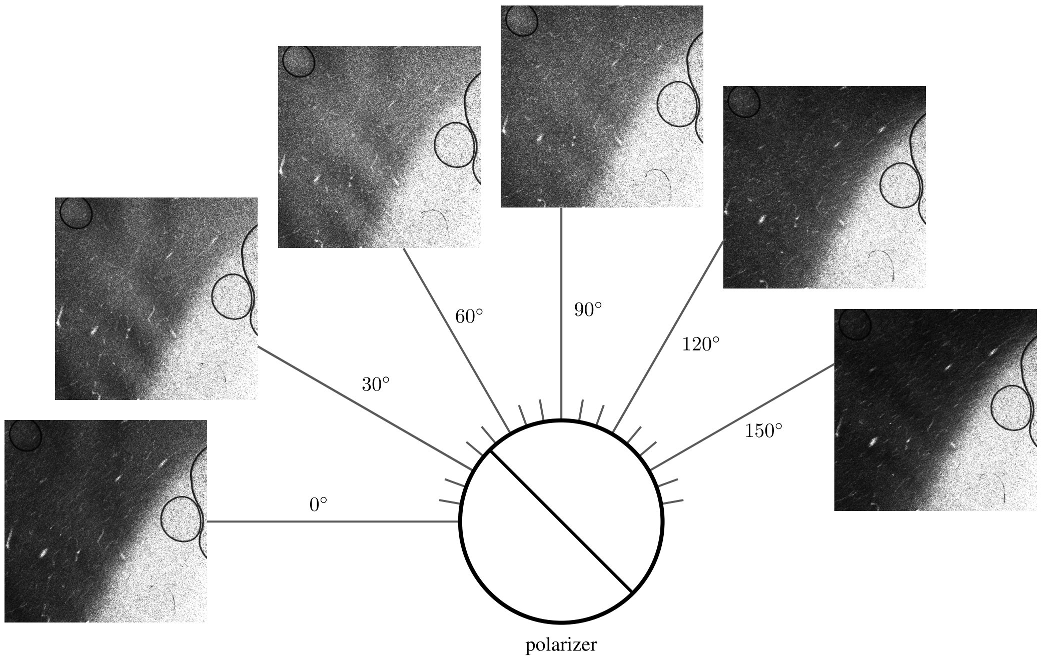


Figure 2.3: The polarizers and the quarter-wave retarder are rotated 18 times for each brain section. The pixels containing parts of nerve fibers show a sine wave pattern, here demonstrated by six of the 18 images. [AAG+11]

The intensity distribution of each pixel showing a nerve fiber within the 18 images is approximately a sine curve. Therefore a sinusoidal curve is fitted through the measured intensities to estimate the fiber directions. Using a discrete harmonic

Fourier transform, the parameters I_0, φ and $|\sin\delta|$ can be extracted:

$$I = \frac{I_0}{2} \cdot [1 + \sin(2\rho - 2\varphi) \cdot \sin(\delta)] \tag{2.1}$$

I is the intensity captured by camera or microscope. I_0 is the so-called transmittance which is "*the intensity of incident light modified by absorption independent of the birefringence of the material*" [PAG+10]. The transmittance image is proportional to the mean of all 18 images. ρ is the rotation of the polarimeter, i.e. the polarizing filters, with respect to a reference angle. φ is the in-plane fiber direction and δ the so-called retardation which is "*the maximum phase shift between the orthogonally polarized components of a light ray passing through*" the brain tissue within the 18 images [PAG+10].

The fiber inclination α, the out-of-section angle, depends on the thickness of the brain section *d*, the birefringence Δn and the wave length of the light λ:

$$\delta = \frac{\pi}{\lambda} \cdot d \cdot \Delta n \cdot \cos^2(\alpha) \tag{2.2}$$

This way the fiber orientation vectors (FOV) can be estimated for the whole-section polarimeter images. The course of the nerve fibers within a section can be extracted out of the FOVs using a streamline algorithm. [AAG+11, PAG+10, TM04]

2.4 3D Reconstruction of the Polarimeter Images

Without further post-processing, it is only possible to extract fiber orientation vectors and thus the course of nerve fibers through single sections but not three-dimensionally through the whole brain. In case of the microscope image tiles, all tiles of a section have to be reassembled, too. The intuitive way for the reconstruction would be to join all mosaic tiles of a section assuming that the offsets of the imaging are well-known. Afterwards, all sections either in low or high resolution can be interpreted as an image stack and hence as 3D data.

This procedure is not possible due to several reasons: The mechanical precision of the mobile microscope stage is much lower than the imaging resolution so that the offset is not known accurately enough. Additionally, as described in section 2.2, the sections are non-linearly deformed twice during the preparation process, namely during the cutting and the mounting on glass slides. These deformations have to be undone in order to track nerve fibers beyond a single section.

Because of these reasons, a 3D reconstruction pipeline has been established to re-deform the images which can be seen in figure 2.4. A so-called registration is used for the equalizing of the non-linear deformations.

A registration is "*used to align two or more images of the same scene*" [GW08, p. 89] or, in case of PLI, neighboring sections or slices of the same object. In most

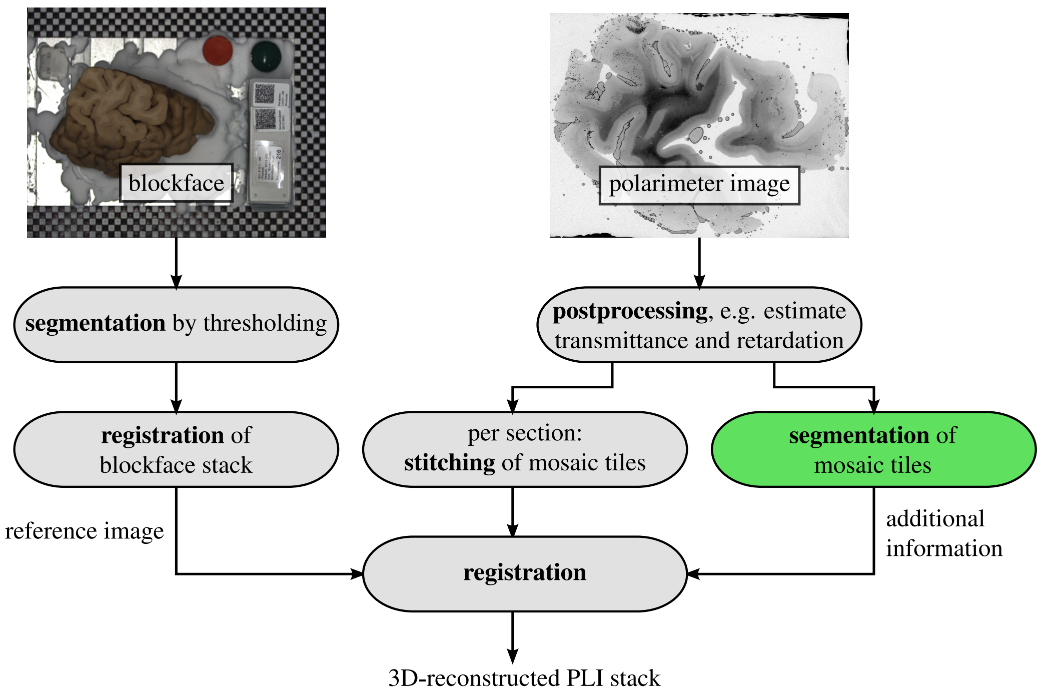

blockface
polarimeter image
segmentation by thresholding
postprocessing, e.g. estimate transmittance and retardation
registration of blockface stack
per section: stitching of mosaic tiles
segmentation of mosaic tiles
reference image
additional information
registration
3D-reconstructed PLI stack

Figure 2.4: This reconstruction pipeline is needed to reverse the non-linear deformations which happened during the preparation of the brain tissue. The polarimeter images are first post-processed, e.g. the estimations of transmittance and retardation are calculated. The mosaic tiles are stitched section-wise. For the registration of the polarimeter images, the already registered blockface images are used as references. A segmentation of the mosaic tiles is needed as additional information to fasten the registration and improve the results. [PAG⁺10]

cases, an image is registered against another one, the reference image, provided that the reference is already well aligned. [GW08, p. 89f]

Since all polarimeter images are uniquely non-linearly deformed, none of them can be reasonably used as a reference for the registration. For that reason, a blockface image has been captured before each cut during the preparation phase "*assuming the imaging setup unchanged during cutting*" [PAG⁺10] (see section 2.2). However, the brain block is slightly moved between two pictures during the cutting. Therefrom the blockface images have to be registered among themselves until they serve as references.

To ease and improve the registration of the blockface images, a check-board pattern is visible in the background of these images. To extract this pattern and other non-brain parts of the images, a distinction between brain and background has to be done first. This differentiation between an object and the background is called segmentation. In case of these color images, a simple thresholding based approach is enough to get an adequate quality of the resulting masks. So the brain can be identified mainly by distinct intervals of intensity and color.

The microscope polarimeter images are stored as mosaic tiles which have to be joined to single images per section before their registration as already mentioned above. The process to align the tiles correctly is called stitching. If the stitched image tiles and the registered blockface images are generated, the registration of the polarimeter images can take place. In this step the mayor challenge is the immense amount of data. Working with the microscope images, there are 1500 to 3000 sections each with about 30×30 tiles. Each image tile has an original resolution of 2048×2048 pixels which results in a file size of about 16MB per tile and approximately 14.4GB per section or 21.6 to 43.2TB per human brain depending on the number of sections. On top of that, there are 18 pictures of each section with different used polarization states which have not been included in this measurement of memory consumption.

Another problem especially with the microscope images are image noise and artifacts of the preparation like air bubbles. Since these effects are individual for each image, the registration has to ignore the affected pixels or regions to not influence the results.

Both problems, so the amount of data and the influence of noise, can be handled better if only those parts of the images are taken into account for the registration which belong to the brain but not to the background. This means that a segmentation of the polarimeter images differentiating between brain and background has to be performed before the registration. In general, a "*segmentation subdivides an image into its constituent regions or objects*" [GW08, p. 689], so in this case into brain and background regions. Another advantage of the availability of segmentation results is, that it can be reused for a 3D visualization of the dataset or extracted information like FOVs to mask out the background. [PAG$^+$10]

2.5 Requirements for the Segmentation

Since segmentations are important for a lot of use cases like motion capture or automatic analysis of images, a bunch of different approaches is available. Hence the requirements for the segmentation induced by the needs of the registration have to be evaluated in order to choose an appropriate algorithm.

Segmenting images bears always the risk to attach a pixel erroneously to the wrong part of the image, so an object pixel is assigned to the background or vice versa. For the further processing of the brain images it is important to avoid cutting away parts of the brain because this would lead to a loss of information. In particular, the calculation of statistics needed for the registration would achieve wrong results if parts of the brain are ignored since they are masked out by the segmentation. This leads to the first requirement for the segmentation:

> *Marking brain pixel as background is worse than marking background pixel as brain.*

Another important aspect of the required segmentation is the large number of brain images to be processed. Many segmentation approaches need manual interaction or input like setting seed points or defining training data. This amount of interaction depends on the number of images to be processed. The approach works for use cases with a manageable number of images but not for the bulk of brain images. That leads to the second requirement:

> *The amount of manual input or interaction has to be independent of the number of images to be segmented which means that an automated segmentation is needed.*

A third requirement is the result of the image quality. Because of the preparation process, the microscope brain images do not show clear edges between brain and background which can be seen in figure 2.5. Even manually it is not possible to define the edges with an accurateness of a pixel. This results in the third requirement:

> *The algorithm has to extract reasonable edges between brain and background although there is a smooth transition between the regions.*

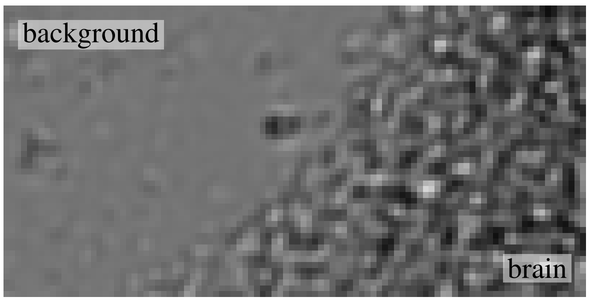


Figure 2.5: This is an enlarged detail of a microscope image with the size of 100×50 pixels. The higher resolution image tiles show smooth transitions between brain and background. A clear edge with a thickness of a single pixel is not optically visible.

Fourthly, the microscope image tiles should be processed independently because the memory consumption of a stitched section is too large to be handled in any image processing. In this case it has to be kept in mind that there are tiles containing parts of the brain and background. But there are also tiles which show only brain or background but not the two regions together:

> *The algorithm has to cope with image tiles containing brain and background pixels as well as tiles consisting of only one of these two.*

The last requirement concerns the handling of artifacts like air bubbles and noise. Air bubbles exist in form of dark rings or even as filled dark objects which can be seen in figure 2.6. Image noise in this case is a generic term for pixels which have an intensity varying a lot from the intensities of adjacent pixels, so for erroneously colored pixels. It may result from various effects like shooting errors occurring in every imaging process, from small dirt particles or from the reaction of the tissue on the polarized light. This last requirement is related to the first one:

> *Artifacts and image noise should be masked in such a way that it does not result in holes within the brain region.*

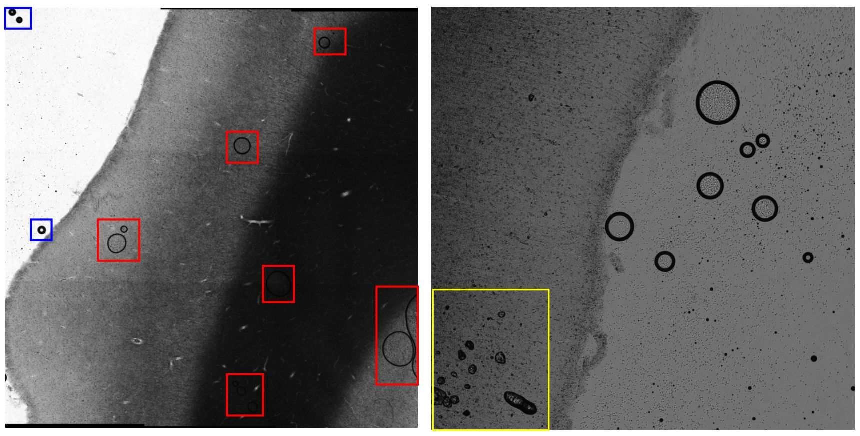

(a) 3 × 3 stitched tiles (b) single tile

Figure 2.6: The images show artifacts in form of air bubbles which are a result of the cover-clipping during the preparation. In the background, the bubbles are dark rings (blue). In the brain, they are either also rings (red) or filled dark regions (yellow).

It means that artifacts within the brain region such as air bubbles should be added to the brain, which is important for the registration. Hence each bubble is not "continued" in the neighboring sections and would complicate the registration. Artifacts in the background region may be assigned to any of the two regions. If it is possible, artifacts should be classified as such.

Beyond these requirements, there are some further constraints regarding the subsequent implementation: The runtime behavior of the application has to be adequate in the sense that also a large number of images can be processed in a reasonable amount of runtime. Possible ways to improve the runtime behavior are a distributed processing of images, so a multi-core application, and/or a transfer of algorithm steps to a GPU[2].

The 18 polarimeter images, the transmittance and retardation are available as input data. The microscope image tiles are provided in the stitched and unstitched version. The application should be able to process a user-defined amount of sections and microscope image tiles in form of a batch job, so a single program run segments all required images.

In summary, there are five obligatory requirements and some further constraints for the segmentation which have to be kept in mind while choosing an algorithm. This choice is described in the following section.

[2]Graphics Processing Units

3 Choice of the Algorithm

Segmentation is one of the standard but challenging tasks of image processing. Hence, a lot of different methods have already been invented and optimized. Which method suits to which images is an important question that has to be answered in advance to achieve the desired results.

In the following section, the most popular segmentation methods are briefly introduced. For a comparison, already existing implementations of the methods are applied to some representative images. Finally, these results are evaluated to choose a method for the present segmentation task.

3.1 Segmentation Methods

3.1.1 Thresholding

Thresholding is one of the simpler but fast methods in order to segment images. Let $f(x,y)$ be a function of the image, e.g. the intensity value of the pixel in row x and column y, and let T be a threshold within the range of f. Then an image g holds the thresholding result and is given as

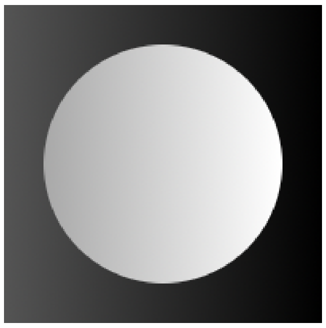
$$g(x,y) = \begin{cases} a & \text{if} f(x,y) > T \\ b & \text{if} f(x,y) \leq T \end{cases} \tag{3.1}$$

The image g is a binary mask with the labels a and b which in general stand for "object" and "background". They are typically illustrated as black and white. With the definition of equation (3.1), the pixels of the original image brighter than T will be marked with a, the others with b, given that f is the intensity function. This operation can be applied to all pixels of the image independently.

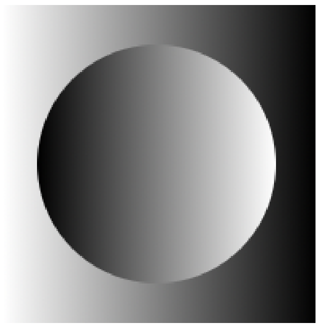
If T is constant for the whole image, the method is called "Global Thresholding", otherwise "Variable Thresholding". This approach only works if such a threshold T exists which separates object and background. So thresholding can be successfully used, if the intensity histogram of the image can be divided into distinct parts and each part belongs either to object or background. Figure 3.1 illustrates two example images, one can and the other cannot be segmented using a global thresholding method. [GW08, p. 738ff]

3.1.2 Region-Based Segmentation

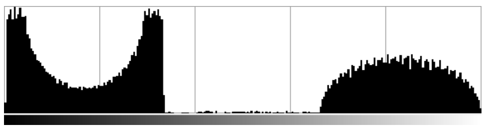
Thresholding attempts to identify various regions of the image, e.g. object and background, by the analysis of image functions like the intensity histogram. Region-based methods try to find these regions directly. There are mainly two different region-based approaches, namely "region growing" and "split and merge".



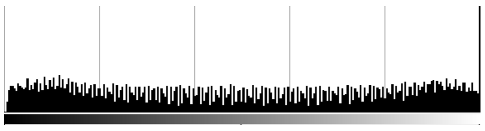
(a) Thresholding possible



(b) Thresholding impossible



(c) Intensity histogram of 3.1(a)



(d) Intensity histogram of 3.1(b)

Figure 3.1: Image 3.1(a) has two clearly separated modes in its intensity histogram 3.1(c). Thus, it can be segmented with any threshold T separating these two intensity intervals. Image 3.1(b) cannot be segmented by thresholding since there are no modes in its intensity histogram 3.1(d).

Region growing is based on the idea to select a single pixel within an object, the so-called seed pixel or seed point, and grow a region starting at this point until the object boundaries are reached. For the other regions of the image, further seeds are needed. All remaining pixels are marked as background in the end.

Thus, region growing mainly consists of two operations: (1) the growing operator defining which pixels are analyzed next, and (2) a similarity criterion defining if a pixel is similar enough to be added to the region or not. The similarity criterion can be "local" like intensity and color or more advanced like the "*likeness of a candidate pixel and the pixels grown so far*" [GW08, p. 764]. This way also images can be processed successfully which could not be segmented by thresholding because of the lack of a valid threshold value since also spatial information are used by the growing operator and optionally also by the similarity criterion.

Split and merge methods are an alternative way to perform a region-based segmentation. An image is first divided into an initial set of disjoint regions. Afterwards the regions are merged and/or split using a similarity criterion. The algorithm results in a set of regions which are unalike to each other. It is implemented recursively most times. For large images, the recursion depth can be very large. This method also has problems to cope with image noise because a noise pixel is unalike compared to the direct neighboring pixels.

In the beginning, the image is divided into four quadrants R_i. The predicate Q defines if a region is uniform ($Q(R_i) = \mathit{true}$) or not. First, each region R_i with $Q(R_i) = \mathit{false}$ is split into four disjoint quadrants until all regions R_i satisfy Q.

Afterwards for each pair of adjacent regions R_i and R_k is checked, if the condition $Q(R_i \cup R_k) = \mathit{true}$ is fulfilled. In this case the two regions are merged. This step is repeated till no regions can be merged anymore.

For both region-based methods, the choice of the similarity criterion has a significant influence on the quality of the resulting mask. It can consist of pixel-wise information like color or intensity, or use also data of the direct neighborhood of a pixel. This value calculated for each pixel can be compared to fix values like thresholds or to the already marked region. [GW08, p. 763ff]

3.1.3 Boundary-Driven Techniques

Boundary-driven methods attempt to move an initial boundary curve until it fits the real object boundaries. A popular sub-class comprises the so-called level set methods.

The level set algorithms start with an initial boundary curve. This curve is then evolved using a speed function F which depends on various information extracted from the image like intensity or the image gradient. The speed F is defined in a way that it is zero, if the curve finally coincides with the object boundaries so that then the evolution stops.

The evolution of the curve is formulated as a partial differential equation. Thus, the computational effort of this approach is comparably high. [Set99, p. 3ff]

3.1.4 Classification Methods

Contextual image classifications segment an image based on a training set which has to be defined manually. This training data contains representative information about the different image regions. Then a feature vector is defined for each pixel holding the information used for the later classification. It is based on information like intensity, texture or edges.

There are a lot of different ways how to gain the final labels out of training data and feature vectors. A popular way is to calculate the contextual classification based on Bayes minimum error classifier. This statistical approach takes into account the error induced by misclassification and tries to minimize this error. Therefor the feature vector of each pixel and the feature vectors and labels of a defined neighborhood around this pixel are taken into account. [SHB07, chapter 8.4]

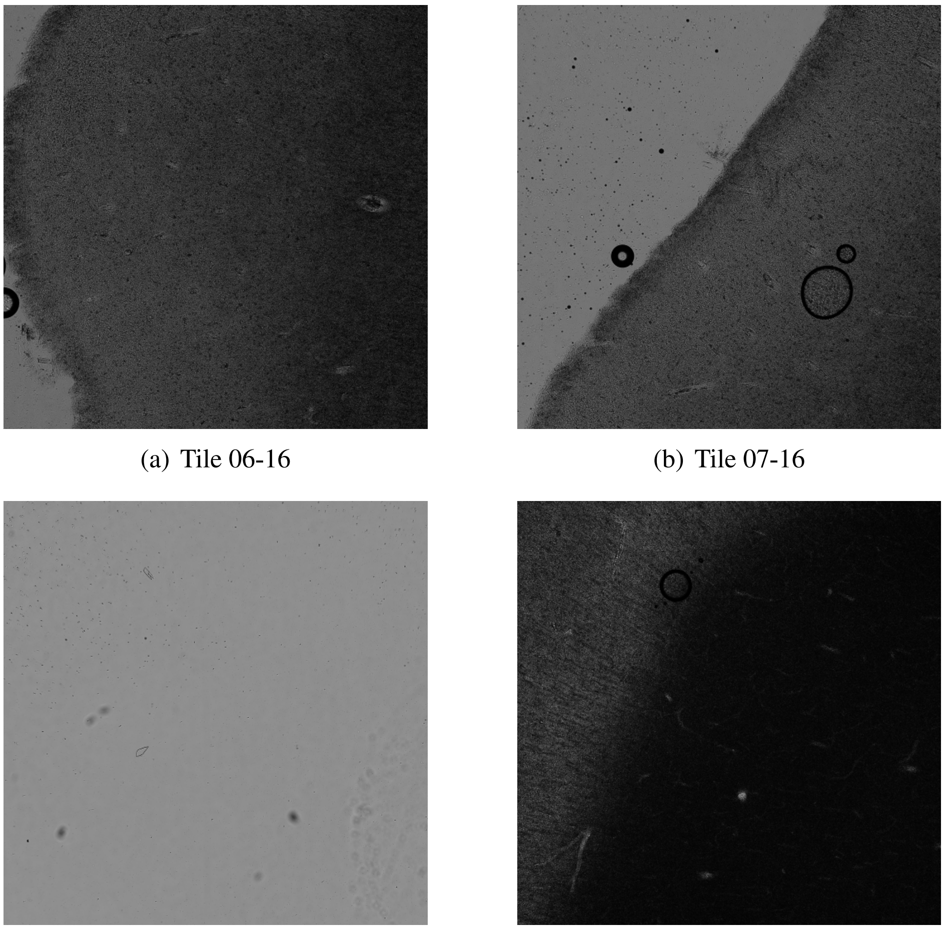
The quality of the result depends a lot on the training data and the choice of the features. The features have to be defined in a way that it can be well differentiated between the image regions based on this information.

3.2 Comparison of Segmentation Tools

Since segmentation is one of the most important tasks of image processing, there are a lot of segmentation tools available. Thus a comparison of different methods can be done only within a selection of existing ones. In the following section some popular tools with implementations of different segmentation methods are compared to find out which works best and with reasonable memory consumption, effort and runtime for the given brain images.

3.2.1 Images used for the Comparison

The given segmentation tools are compared based on four brain image tiles which are representative for the whole data set. They have a color depth of 32 bit and are gray-scale images.


(a) Tile 06-16 (b) Tile 07-16

(c) Tile 00-00 (d) Tile 08-14

Figure 3.2: The images 06-16 and 07-16 show both brain tissue and background. Tile 00-00 contains only background and tile 08-14 only brain.

Image tiles 06-16 and 07-16 (see figures 3.2(a) and 3.2(b)) show both brain tissue and background. The light gray regions in the top left and bottom left corners respectively on image 06-16 belong to the background, the darker parts to the brain. The black rings are air bubbles originating from the cover-slipping step during the preparation as described in section 2.2.

Tile 00-00 (figure 3.2(c)) shows only background with some dirt particles. In contrast, tile 08-14 (figure 3.2(d)) contains only brain. In this tile a small hole is also visible within the brain tissue as a result of the sectioning, which is the light gray speckle in the middle of the lower third of the image. The tools to be tested should also cope with these two images because tiles with only one of the two regions occur frequently.

These images illustrate that in principle the intensity histograms of the brain images can be divided into a light background and a dark brain. Especially the brain features a large variance in intensity. The lighter parts like in tile 07-16 and the top left corner of tile 08-14 show the gray matter of the brain. The darker parts visible in the bottom right corner of tile 08-14 belong to the white matter basically containing the nerve fibers.

3.2.2 Process of Testing

The evaluated tools have been applied to these four images. They all provide a two-colored mask as result. To allow an optical evaluation of these masks, they have been post-processed as follows:

- The edges between the regions are extracted using the edge detection of GIMP [The13] with an edge size of 1.0 pixel.
- The edges are colored red, all other pixels black also using GIMP.
- Fiji [SACF$^+$12] is used to overlay the original tiles with the extracted edges while hiding the black background of the edge images. This is done by the “Image Calculator” using the “Transparent-zero” operator.

The parameters of the given tools have been optimized in an iterative process. The results presented here all belong to the best configurations which have been figured out.

3.2.3 Results and Evaluation

Thresholding

GIMP: Threshold

A thresholding segmentation of GIMP [The13] (version 2.6.11, Windows 64bit) has been tested. This tool interprets the pixel intensities as eight bit values. The threshold value has been chosen as $T = 107$, so the intensity interval $[0, 107]$ belongs to the background and $[108, 255]$ to the brain.

The result (see figure 3.3) clearly shows that thresholding is prone to noise. The background contains also dark pixels, the brain also lighter colored ones. These noise pixels are respectively assigned to the wrong regions of the segmentation mask. Hence, not only the desired edges between brain and background but also edges around all noise pixels are gained. Thus a threshold segmentation does not produce good results.

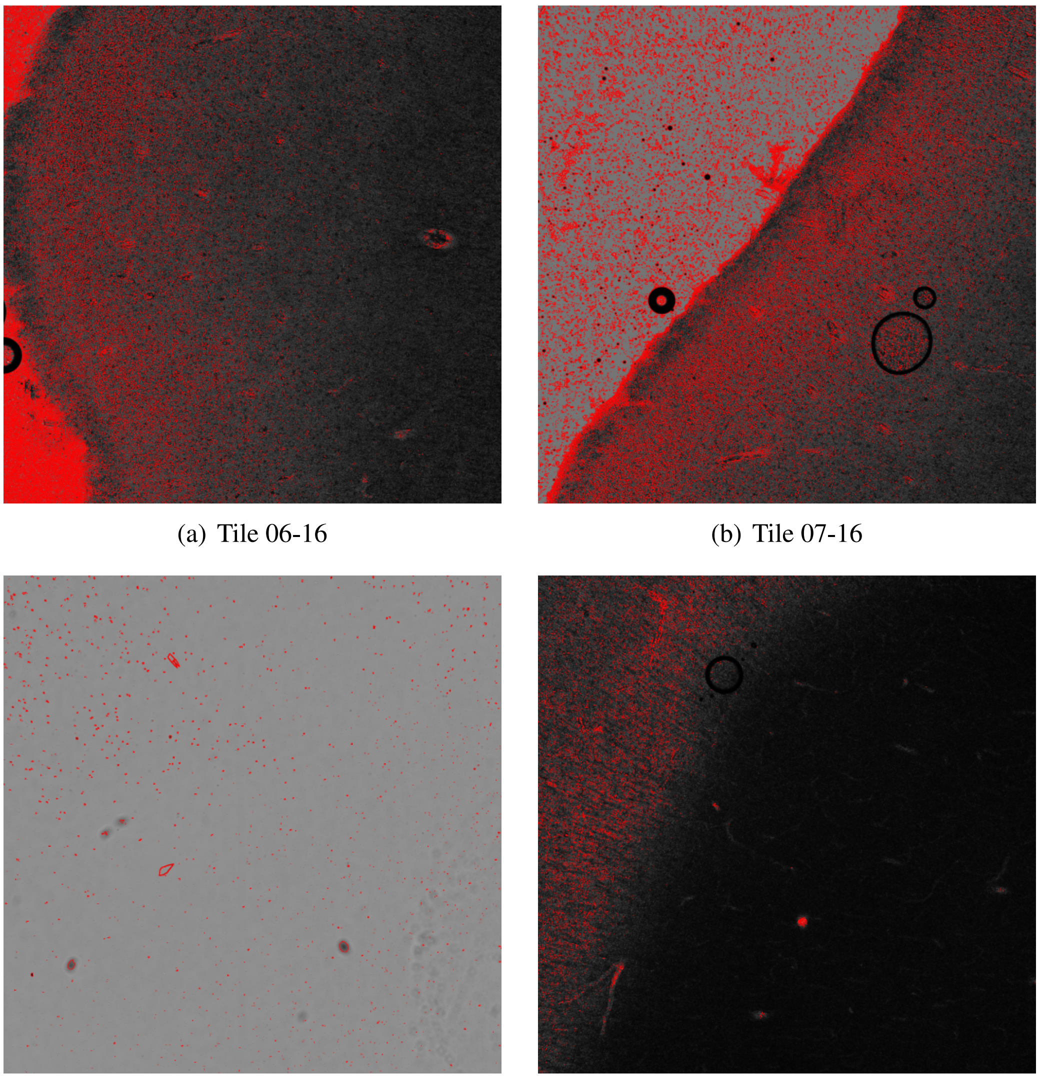

(a) Tile 06-16
(b) Tile 07-16
(c) Tile 00-00
(d) Tile 08-14

Figure 3.3: GIMPs thresholding method is prone to noise effects so that no clear edges between brain tissue and background are gained because the brain images are full of noise pixels.

Region-Based Segmentation

Fiji: Seeded Region Growing Tool

Fiji [SACF$^+$12] (ImageJ version 1.47k, Windows 7 64bit) provides a bunch of different image processing algorithms, inter alia a "Seeded Region Growing Tool". It has been developed by Sasha Jarek [KJ11] based on [AB94], the first description of a region growing algorithm using seeds.

The tool has only been applied to the tiles 06-16 and 07-16, so the tiles showing both brain and background, because this tool requires seeds for at least two different regions.

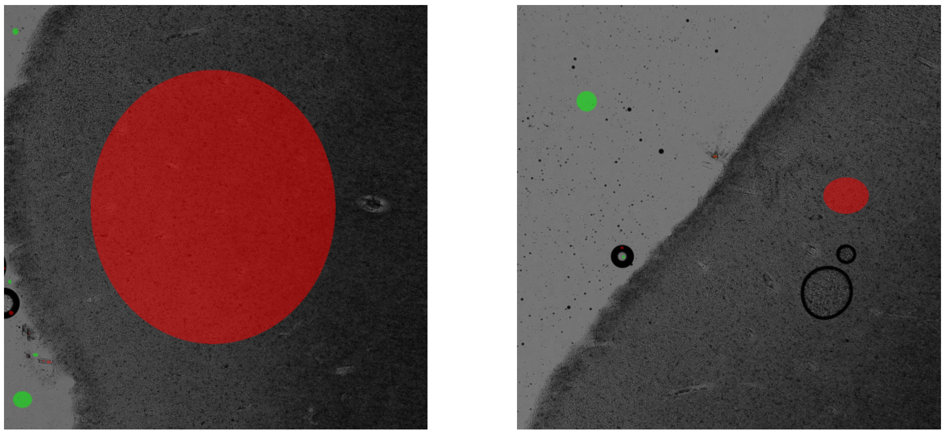
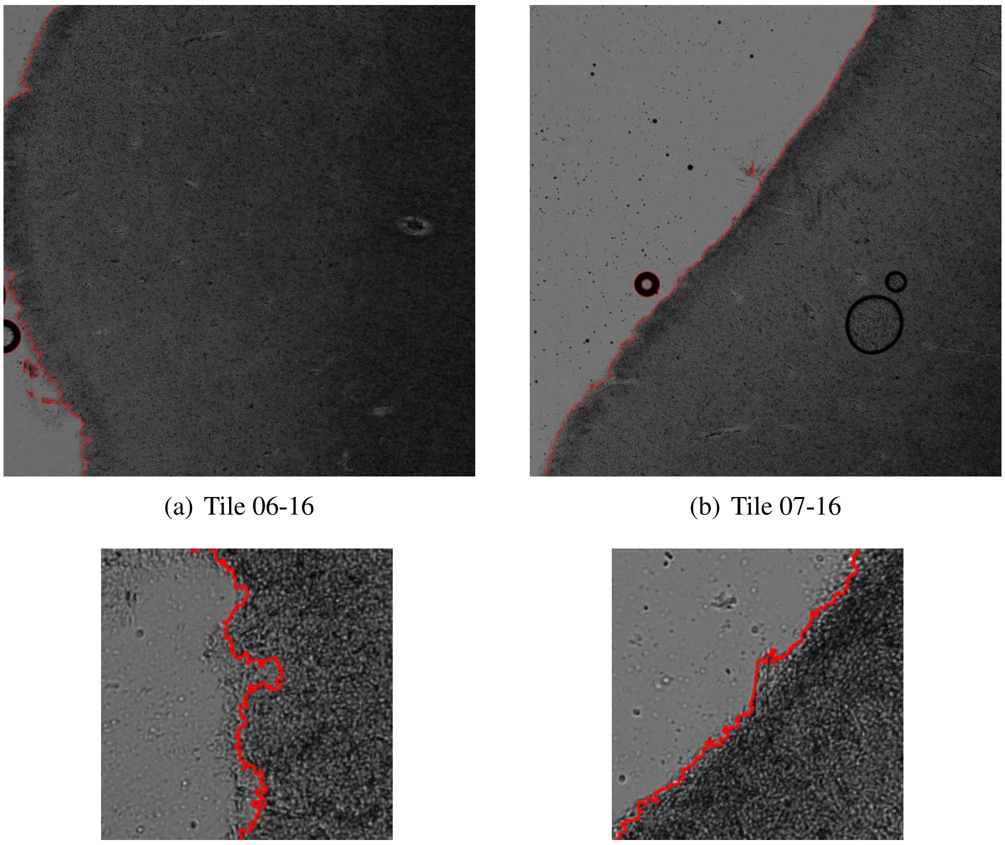

(a) Seeds for tile 06-16 (b) Seeds for tile 07-16

Figure 3.4: The red regions are the seeds for the brain, the green ellipses for the background. It is important to add a seed to each of the not connected parts of an image belonging to the same region. So in 3.4(a) each background region gets an own seed.

The seeds can be applied as arbitrarily shaped forms, in this case some ellipses have been used (see figure 3.4). With each seed, an own region of the image is marked. The seeds for the brain regions are colored red, the ones for the background green. If two seeds have the same color like the green ones in seed image 3.4(a), the tool will join these regions in the resulting mask. Since in this image the background in the top and bottom left corners is not connected, an own seed for each of these regions is needed.

The masks created by this tool show clear edges between brain and background (see figure 3.5). This means that the algorithm can cope with noise in contrast to thresholding. But in the detail images (see figures 3.5(c) and 3.5(d)) it is visible that the extracted edges are moved some pixels away from the correct boundary into the brain region. Anyway, this method seems to work for the brain images with some adaptations to the given images.



(a) Tile 06-16

(b) Tile 07-16

(c) Detail of tile 06-16

(d) Detail of tile 07-16

Figure 3.5: The region growing tool of Fiji creates a mask with clear edges between brain and background. The course of the edge follows the boundary between brain and background but in detail images (3.5(c) and 3.5(d)) it is visible that the extracted edge is moved some pixels into the brain region.

ITK Connected Threshold Region Growing

The Insight Segmentation and Registration Toolkit (ITK, version 4.1.0, Linux 64bit)[1] is a C++ toolkit for image processing. It provides some different region growing methods implemented as executable command line tools. The first one which has been tested is the Connected Threshold Region Growing which is implemented in `ConnectedThresholdImageFilter` [ISNC05, p. 504ff].

This method gets the position of a seed pixel and an intensity interval of the region to be masked as input. After an edge-preserving smoothing of the original image, a single region is then grown starting at the seed point. A neighbor pixel of the already marked region is added to this region if the criterion (3.2) is fulfilled. I is the intensity of a pixel at position (x, y), $[\text{lower}, \text{upper}]$ is the user-defined intensity interval.

$$I(x, y) \in [\text{lower}, \text{upper}] \tag{3.2}$$

[1]http://www.itk.org/

If more than one region should be masked, the tool has to be applied several times, once for each region. This is necessary if the image contains two or more not connected parts of the brain. The resulting masks can be joined with an OR operation. The tool has been started with the following parameters:

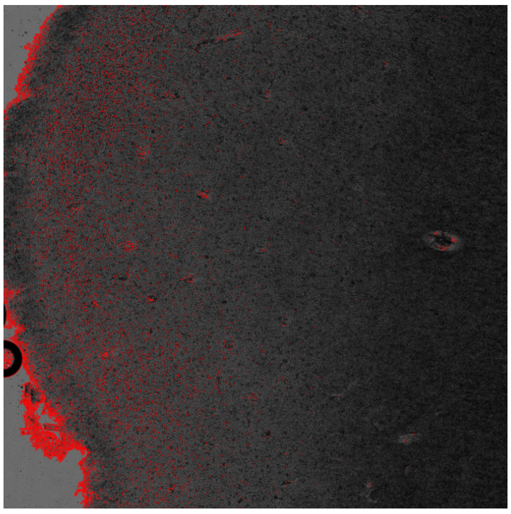
Tile	Seed Position	Intensity Interval	Seed Region
06-16	$(1024, 1024)$	$[0, 3549]$	brain
07-16	$(1215, 1353)$	$[0, 3549]$	brain
00-00	$(1024, 1024)$	$[3550, 65536]$	background
	$(666, 1209)$	$[3550, 65536]$	background
08-14	$(1024, 1024)$	$[0, 3549]$	brain

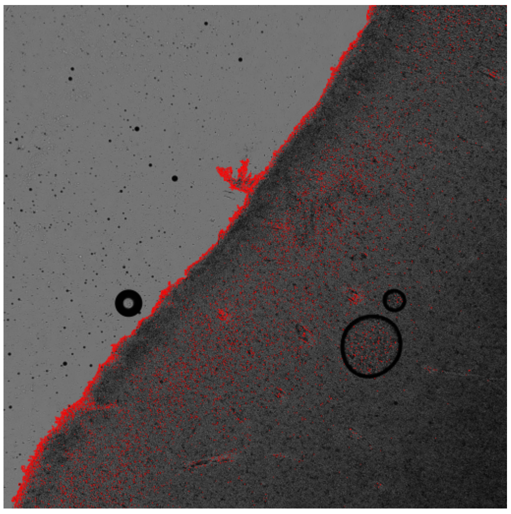
Table 3.1: Start parameters for ITK Connected Threshold Region Growing

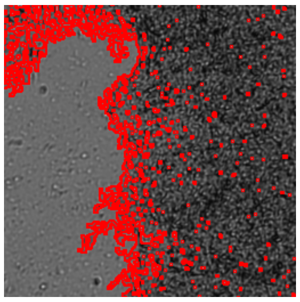
For all images showing brain, the seeds have been placed in the brain region. ITK interpretes the images as 32 bit gray-scale so that the intensity intervals have to be adjusted to this range of possible values. Thus, the intensity interval for the brain has been chosen as $[0, 3549]$.

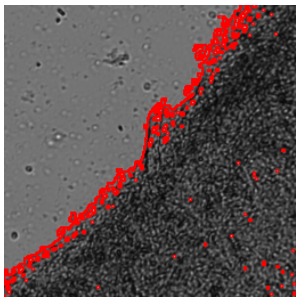
Tile 00-00 shows only background but no brain. Therefore the seed has to be placed in the background together with an intensity interval for the background which is $[3550, 65536]$. This tile has been segmented twice to mark also the inside of an air bubble correctly.

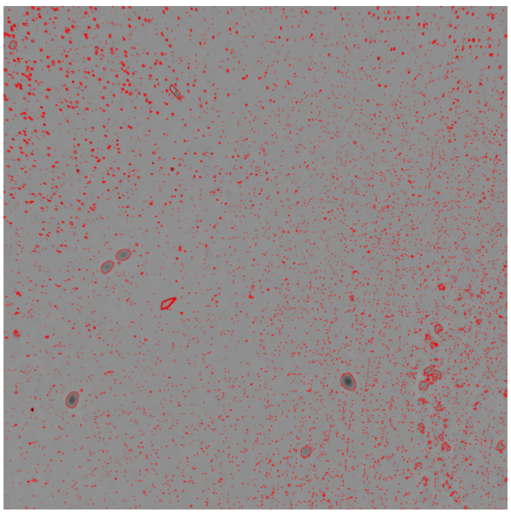
The edges of the resulting masks (see figure 3.6) fit well the actual boundaries between brain and background. However, the method is prone to noise because it is based on thresholding and thus has the same problems to cope with noise as simple thresholding approaches. The effects of noise are only visible in the grown regions but not in the rest of the image because the noise pixels are not added to the growing region. The non-grown parts do not show noise since these parts of the image are connectedly masked after the growing process.


(a) Tile 06-16


(b) Tile 07-16


(c) Detail of tile 06-16


(d) Detail of tile 07-16


(e) Tile 00-00

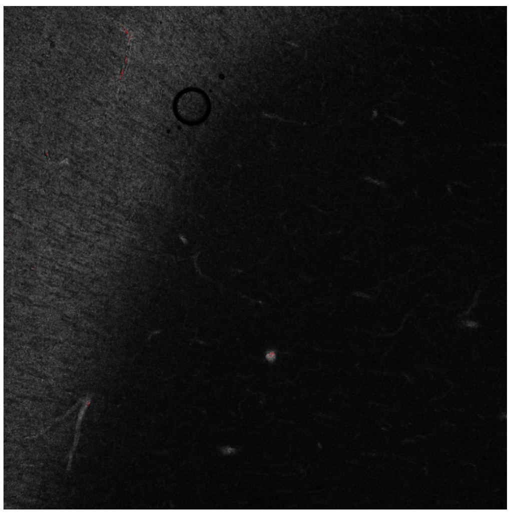

(f) Tile 08-14

Figure 3.6: The ITK Connected Threshold method extracts reasonable boundaries. It is prone to image noise within the grown region. In all tiles except for tile 00-00, the brain region has been grown. In this tile, the background has been grown because this image shows only background.

ITK Neighborhood Connected Region Growing

Another region growing method provided by ITK is the Neighborhood Connected Region Growing implemented in `NeighbourhoodConnectedImageFilter` [ISNC05, p. 510ff]. It is a variant of `ThresholdConnectedImageFilter` which has been tested before.

This algorithm adds a pixel to the region if criterion (3.2) is fulfilled for the examined pixel and if additionally also all neighboring pixel fit in the user-defined intensity interval $[\text{lower}, \text{upper}]$. Therefore, the tool expects the same parameter list like the former one. The following configurations have been used for the execution:

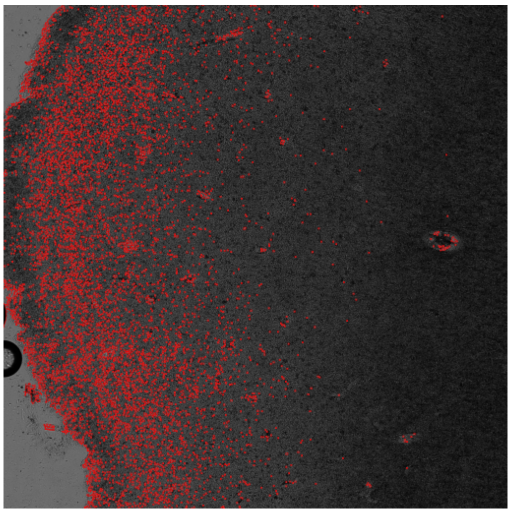
Tile	Seed Position	Intensity Interval	Seed Region
06-16	$(1024, 1024)$	$[0, 3549]$	brain
	$(117, 1584)$	$[0, 3549]$	brain
	$(168, 1716)$	$[0, 3549]$	brain
	$(80, 1870)$	$[3000, 65536]$	background
	$(40, 1310)$	$[3000, 65536]$	background
	$(40, 110)$	$[3000, 65536]$	background
07-16	$(1300, 1300)$	$[0, 3700]$	brain
	$(960, 720)$	$[0, 3700]$	brain
	$(750, 490)$	$[3000, 65536]$	background
	$(500, 1210)$	$[3000, 65536]$	background
00-00	$(1024, 1024)$	$[3000, 65536]$	background
08-14	$(1024, 1024)$	$[0, 3549]$	brain

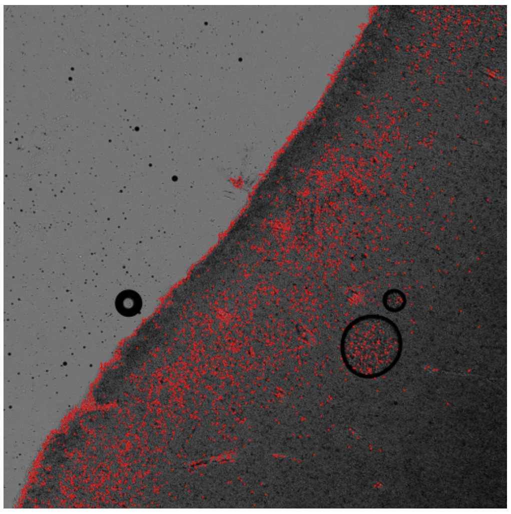
Table 3.2: Start parameters for ITK Neighborhood Connected Region Growing

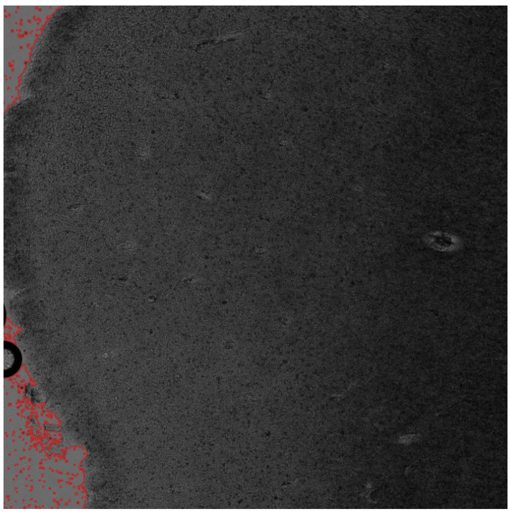
For tiles 06-16 and 07-16, so the tiles showing both brain and background, two different variants have been tried. First, the seeds have been placed in the brain region ("brain seeds"). As an alternative, the seeds have than be placed in the background regions ("background seeds"). It should be noticed that in this case the chosen intervals for brain and background have been chosen as overlapping which produced the best results. The results can be seen in figures 3.7 and 3.8.

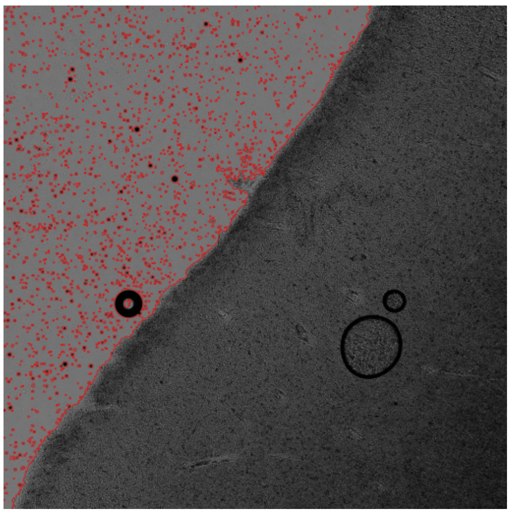
In both variants, the grown regions are influenced by image noise, i.e. they not only contain the edges between brain and background but also a lot of noise isles within the brain respectively background. The found boundaries between the two regions are nearly the expected ones which can be seen in figures 3.7(e) and 3.7(f). So apart from the noise effects, the results are reasonable and even better than the ones from the former algorithm. Also the results for the "single labeled" tiles 00-00 and 08-14 are good.

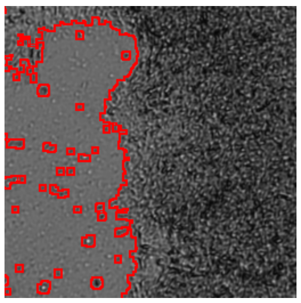
The execution time is slightly larger than for the ITK Connected Threshold method. In exchange, the choice of seeds seems to be more stable, so a small change of the seed position does not affect the result.


(a) Tile 06-16 (brain seeds)


(b) Tile 07-16 (brain seeds)


(c) Tile 06-16 (background seeds)


(d) Tile 07-16 (background seeds)


(e) Detail of 3.7(c)

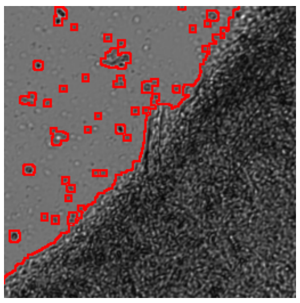

(f) Detail of 3.7(d)

Figure 3.7: The ITK Neighborhood Connected method extracts reasonable boundaries. It is influenced by image noise within the grown region. Two variants have been tested, once with seeds in the brain and once in the background region.

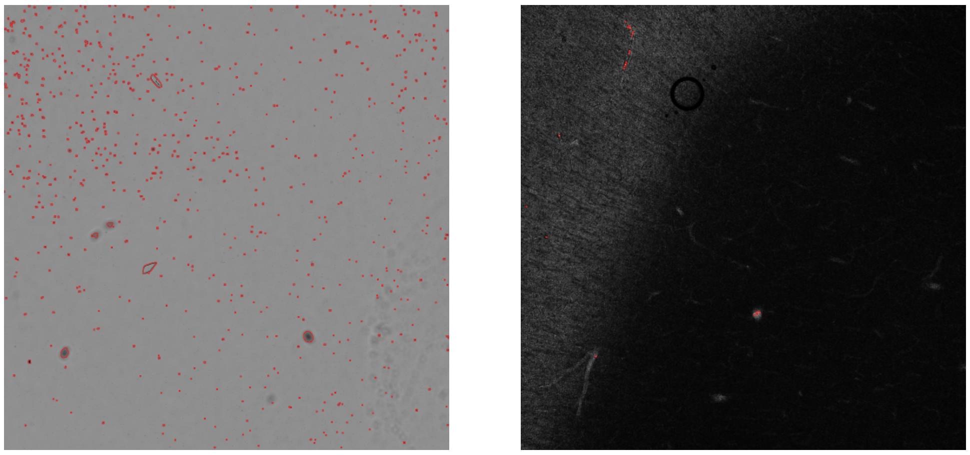

(a) Tile 00-00 (b) Tile 08-14

Figure 3.8: The ITK Neighborhood Connected method copes with tiles containing either brain or background but not both regions.

ITK Confidence Connected Region Growing

A third region growing segmentation of the ITK is the Confidence Connected one implemented in `ConfidenceConnectedImageFilter` [ISNC05, p. 514ff].

This method uses a more advanced similarity criterion compared to the former described ones. After a new pixel has been added to the region, the mean μ and standard deviation σ of the intensity of all pixels belonging to the region are computed respectively updated. Another pixel is added to the region, if criterion (3.3) is fulfilled. $I(x,y)$ is the intensity of a pixel at position (x,y). α is a user-defined factor. In contrast to the last two ITK methods, the user does not define an intensity interval representing the region but only the factor α which has been chosen as $\alpha = 2.5$.

$$I(x,y) \in [\mu - \alpha \cdot \sigma, \mu + \alpha \cdot \sigma] \tag{3.3}$$

This criterion is a real similarity criterion because it allows to add a pixel to the region if the pixel is similar enough to the region instead of similar to a defined intensity interval. The tool has been used with the following parameter configurations:

Tile	Seed Position	Seed Region
06-16	$(960, 720)$	background
07-16	$(750, 490)$	background
	$(500, 1210)$	background
00-00	$(1024, 1024)$	background
08-14	$(1024, 1024)$	brain

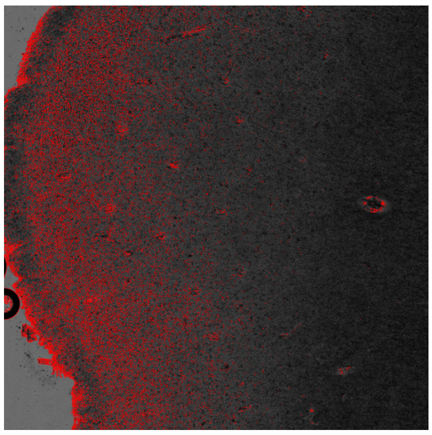
Table 3.3: Start parameters for ITK Confidence Connected Region Growing

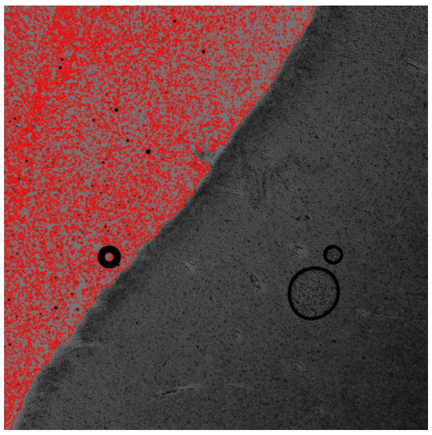
For tiles 06-16 and 07-16 again both variants with seeds in brain or background have been compared. It figured out that for both tiles background seeds produce better results than brain seeds so that only these results can be found in figure 3.9.

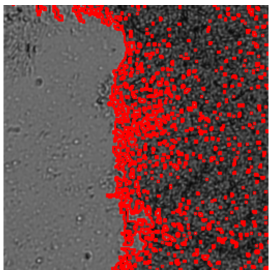
This similarity criterion of this method is prone to image noise especially if the noise pixels are much lighter or darker than the surrounding region. The parameter α can be changed to suppress the noise, but an enlargement of α would result in wrong edges because the intensity boundary between brain and background would be ignored respectively shifted by too large intervals.

The result for tile 08-14 (see figure 3.10) is the worst one. This image shows the large intensity variance of the brain tissue. Even if the seed has got the average of the brain intensities, the tool does not mask the whole brain as brain. The parameter α should be chosen equal for all images because the tiles should be processed batch-like so with the same α for all images. Hence, a manipulation of α is no solution to this problem because it is not known in advance if a tile contains brain or background.

All in all, this method seems to be inappropriate for the given brain images due to the kind of image noise they contain and the problem to handle the intensity variance of brain tissue.


(a) Tile 06-16


(b) Tile 07-16


(c) Detail of tile 06-16

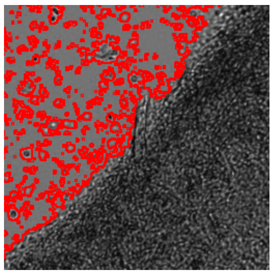

(d) Detail of tile 07-16

Figure 3.9: The ITK Confidence Connected method extracts reasonable boundaries but it is influenced a lot by image noise.

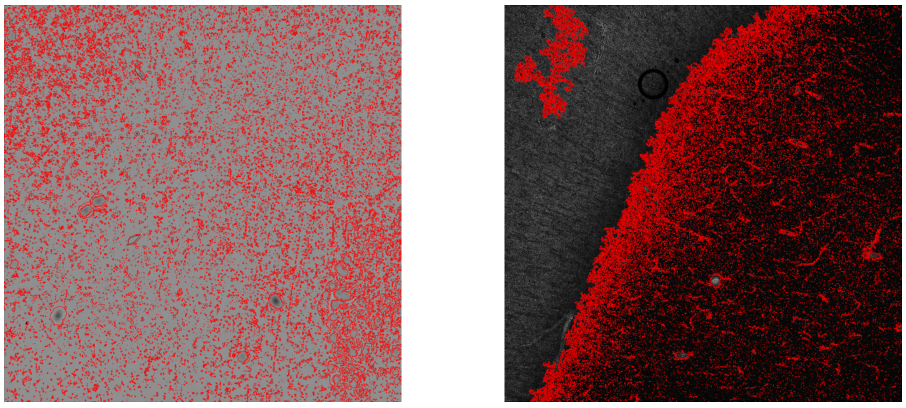

(a) Tile 00-00 (b) Tile 08-14

Figure 3.10: The ITK Confidence Connected method cannot cope with the large intensity variance of brain tissue as 3.10(b) shows.

ITK Isolated Connected Region Growing

The last tested region growing method of ITK is Isolated Connected Region Growing which is implemented as `IsolatedConnectedImageFilter` [ISNC05, p. 5118ff].

This is again a variant of the Connected Threshold method. It requires two seeds, the first one within the region, the second one outside of it. It is assumed that the region is lighter colored than the rest so that in addition a lower threshold for the region has to be provided. The algorithm then determines an intensity value which could be the upper threshold for the region based on this information. The value which separates the two seeds best is found using a binary search.

Since this method needs a seed within and one outside of the region, it cannot be used without modifications for images showing only brain or background. Therefore it has only been tested for tiles 06-16 and 07-16 with the following configuration:

Tile	Seed Position (Region)	Lower Threshold	Seed Position (Outside)
06-16	$(1024, 1024)$	0	$(960, 720)$
07-16	$(1300, 1300)$	0	$(750, 490)$

Table 3.4: Start parameters for ITK Isolated Connected Region Growing

The results of this method can be found in figure 3.11. It is obvious that this method has problems to cope with the large intensity variance of the brain tissue like the former evaluated one. The extracted edges are far away from the correct ones. In the mask of tile 06-16, there are a lot of large holes within the brain region.

In the result of tile 07-16, an overflow of the brain region into the background is visible.

Since a modification of the seed positions does not lead to better results, this approach also seems inappropriate for the present brain images.

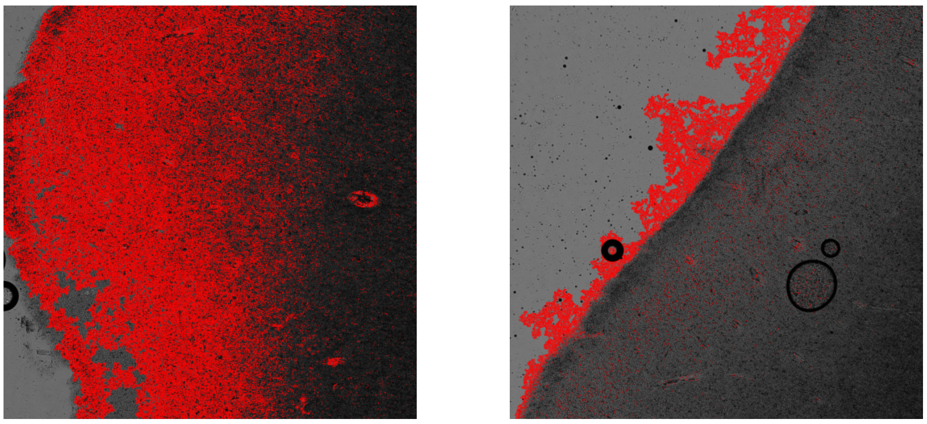

(a) Tile 06-16 (b) Tile 07-16

Figure 3.11: The ITK Isolated Connected method extracts reasonable boundaries but it is influenced a lot by image noise. As figure 3.11(a) illustrates, this algorithm cannot cope with the large intensity variance of brain tissue under polarized light.

Boundary-Driven Techniques

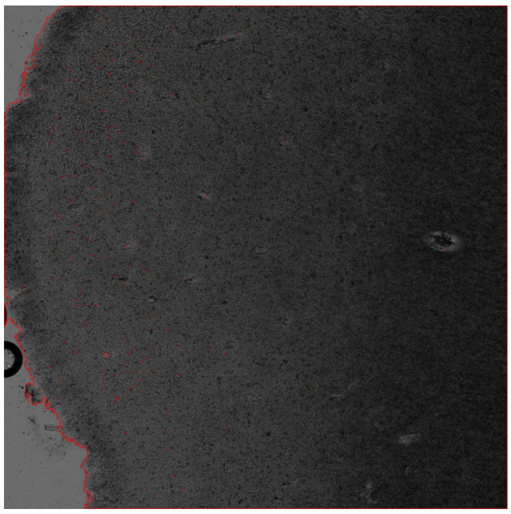
MiaLite

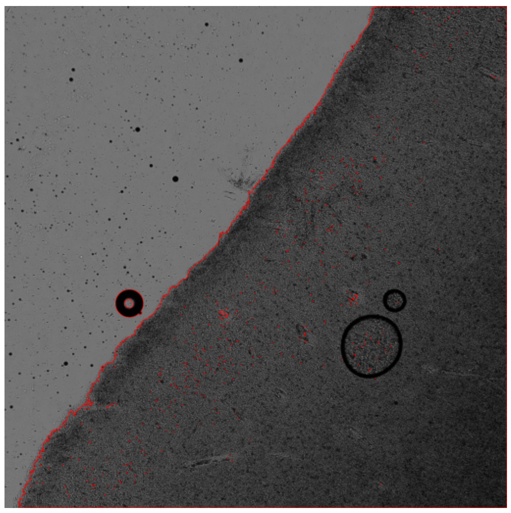
MiaLite (current version from April, 17th 2013 for Window 7 64bit; no official version number provided) is a segmentation tool providing a level set algorithm "*for academic users*" [Mia12]. The user can choose between a threshold based and an edge based variant. Since the brain images do not contain clear edges, the threshold variant achieved better results.

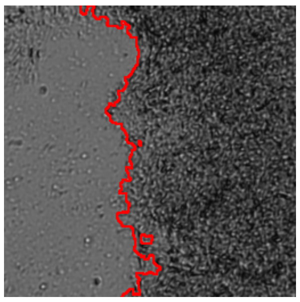
The tool interpretes the images as eight bit gray-scale. Therefore $T = 107$ has been chosen as threshold value again like for the threshold segmentation of GIMP. In addition, a smoothing factor can be defined for a prior image smoothing which has been set to 0.5.

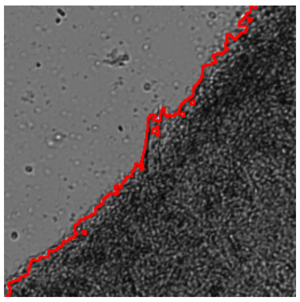
This level set approach works with seed regions which are evolved to the boundaries between object and background. Therefore, a single filled circle has been placed in every brain region. For tile 00-00, so the tile without brain content, the circle has been placed in the background.

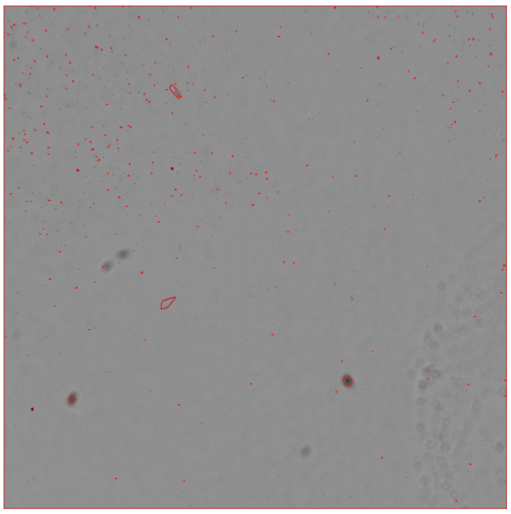
The results of this tool are illustrated in figure 3.12. The gained edges fit well the boundaries between brain and background. The method is not influenced by image noise which might be a result of the smoothing used as preprocessing step. However, this tool has a disadvantage compared with the region growing approaches: The execution time is much larger than for all tested region growing methods.


(a) Tile 06-16


(b) Tile 07-16


(c) Detail of tile 06-16


(d) Detail of tile 07-16


(e) Tile 00-00

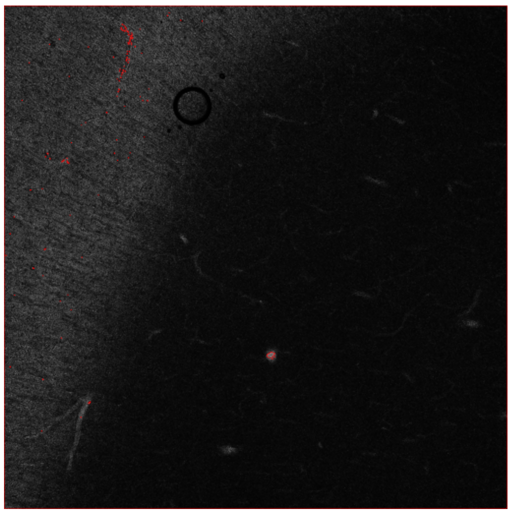
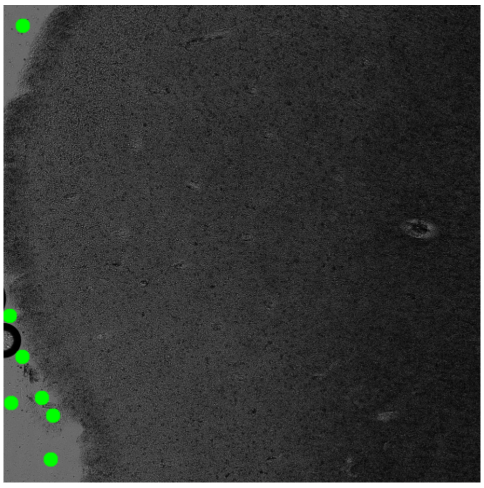

(f) Tile 08-14

Figure 3.12: The resulting edges created by MiaLite fit well the boundaries between brain and background.

ITK-SNAP

ITK provides not only region growing methods but also level set approaches for image segmentation. The tool ITK-SNAP (version 2.4.0, Windows 7 64bit) [ITK11] encapsulates these level set methods and provides a Graphical User Interface for better usage. It is designed to segment 3D images so that the given tiles have to be duplicated to gain image stacks in advance. Furthermore, it only works with 16 bit color depth or less.

This tool also requires a preprocessing based on "intensity regions", so thresholding, which has been done analogue to the other tools. In addition, seed bubbles have to be placed. The edges of the bubbles are than evolved to the object boundaries. Since the execution time for the evolution depends on the size of the region in which they have been placed, the bubbles have been put into the smaller background instead of the larger brain region, what can be seen in figure 3.13. A lot of bubbles are needed to segment the image completely. If less bubbles are placed, parts of the image stay unsegmented. So this tool is not stable concerning the choice of seeds.


(a) Seeds for tile 06-16

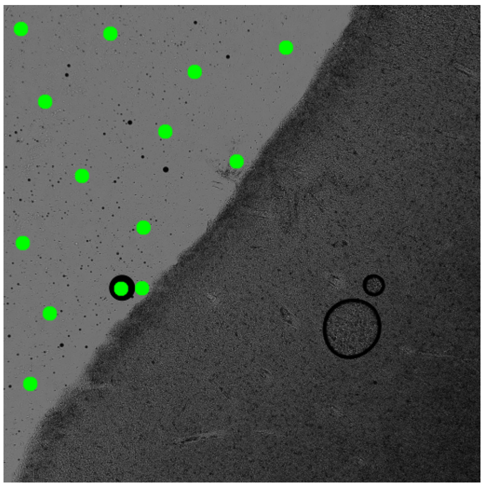
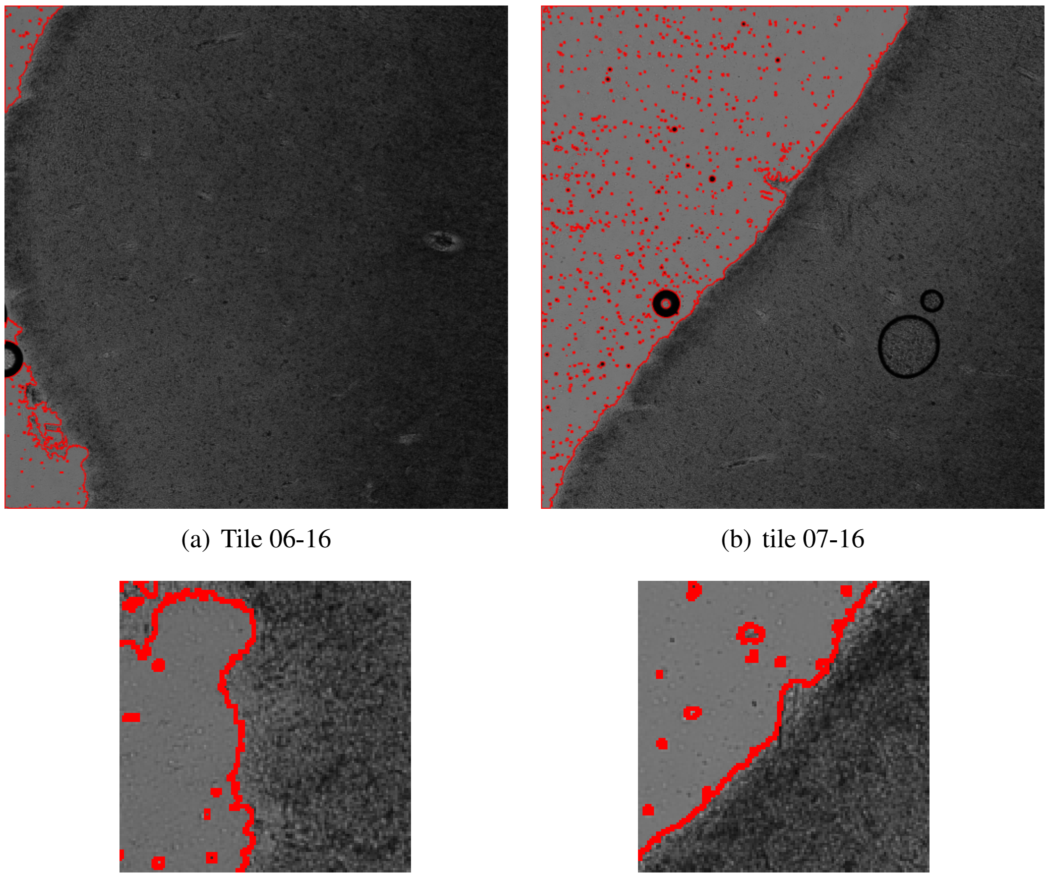

(b) Seeds for tile 07-16

Figure 3.13: The green bubbles have been used as seeds for a level set segmentation with ITK-SNAP. Although most of the seeds are in the same region of the image, the tool does not work correctly with less bubbles.

The results of this segmentation tool can be found in figure 3.14. The edges created by ITK-SNAP are as good as the ones of MiaLite. However, the execution time of this software is even larger than the time needed by MiaLite. To get results in a reasonable amount of runtime, the input images have been scaled to a quarter of the original size, so to 1024×1024 pixels. This effect might be caused by the needed image stack and the consequently larger amount of input data.

The processing of the "single-labeled" images has been stopped due to the immense amount of runtime needed. In principle, the tool segments also these images correctly.

Additionally, the tests figured out that this approach is prone to varying intensities, e.g. because of non-uniform illumination during the imaging process. This results in scraggy edges.


(a) Tile 06-16

(b) tile 07-16

(c) Detail of tile 06-16

(d) Detail of tile 07-16

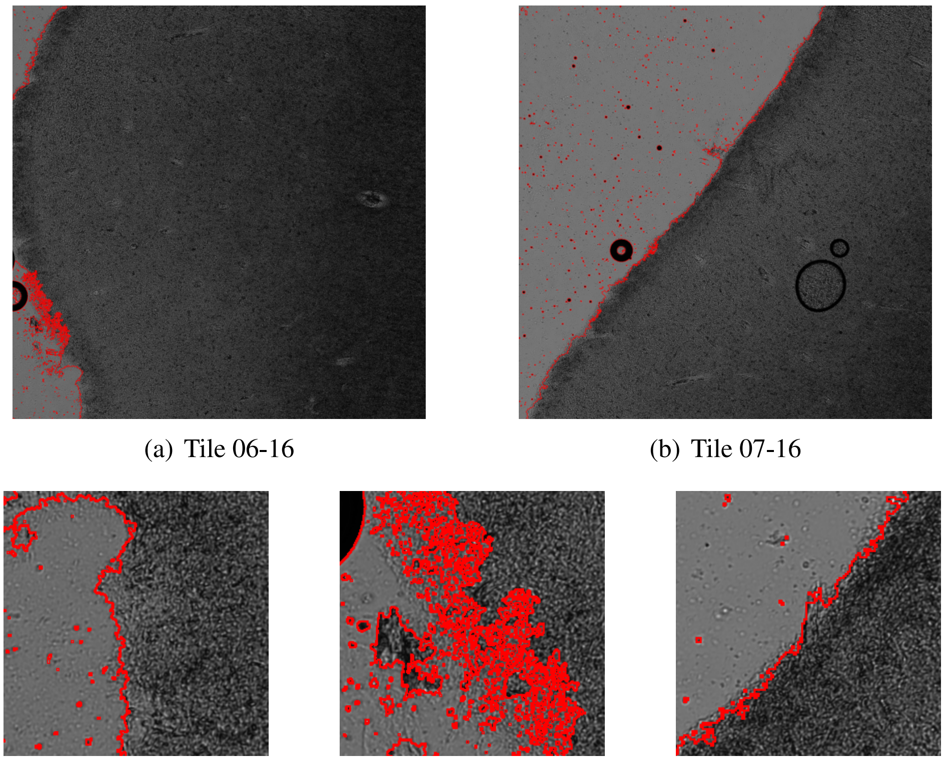
Figure 3.14: ITK-SNAP extracts reasonable edges between brain and background. However, the runtime of this tool is much larger than of all tools tested before.

Fiji Level Sets

The last, tested level set implementation is Fiji Level Sets [SACF+12]. It has the same problems which have been detected while working with ITK-SNAP: This tool also needs a lot of seeds which are in this case seed pixels. All configurable parameters work well with the default values, so a manipulation did not end in better segmentation results.

The runtime again depends on the size of the region in which the seeds are placed. Therefore, for all image tiles it has been tried to set the seeds into the smaller region of the image. For tile 06-16, the background regions are small, so the seeds are placed there and the image has been processed in its original resolution. Tile 07-16 is approximately divided into two halves; it has been scaled down to 1024×1024 pixels. The other two tiles show only brain or background which means that they consist of a single region. So they had to be scaled down to 512×512 pixels to get the results in a reasonable amount of time.

Figures 3.15 and 3.16 illustrate the results of this segmentation tool. The extracted edges are reasonable but it has problems with thin brain tissue originating from rips occurring during the sectioning process (see figure 3.15(d)). It is expressed through visual effects comparable to the reaction to image noise.


(a) Tile 06-16 (b) Tile 07-16

(c) Detail 1 of tile 06-16 (d) Detail 2 of tile 06-16 (e) Detail of tile 07-16

Figure 3.15: Fiji Level Sets finds reasonable edges but has problems with thin brain tissue.

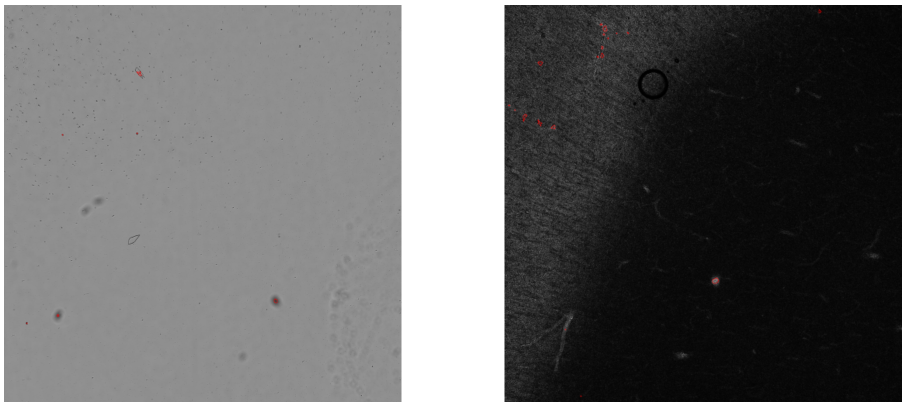

(a) Tile 00-00 (b) Tile 07-16

Figure 3.16: The needed execution time of Fiji Level Sets is comparably large, especially for tiles like 00-00 and 08-14 showing mainly brain or background.

Classification Methods

ilastik

The software ilastik (version 0.5.12, Windows 7 64bit) [SSKH11] has been used to test classification based segmentations. For classifications, first some features have to be chosen. In this case, the color in rings of three, five and seven neighboring pixels has been used. Other features like texture or edges do not seem to work for the brain images as tests have shown.

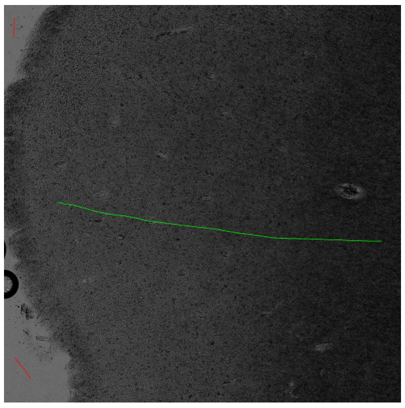
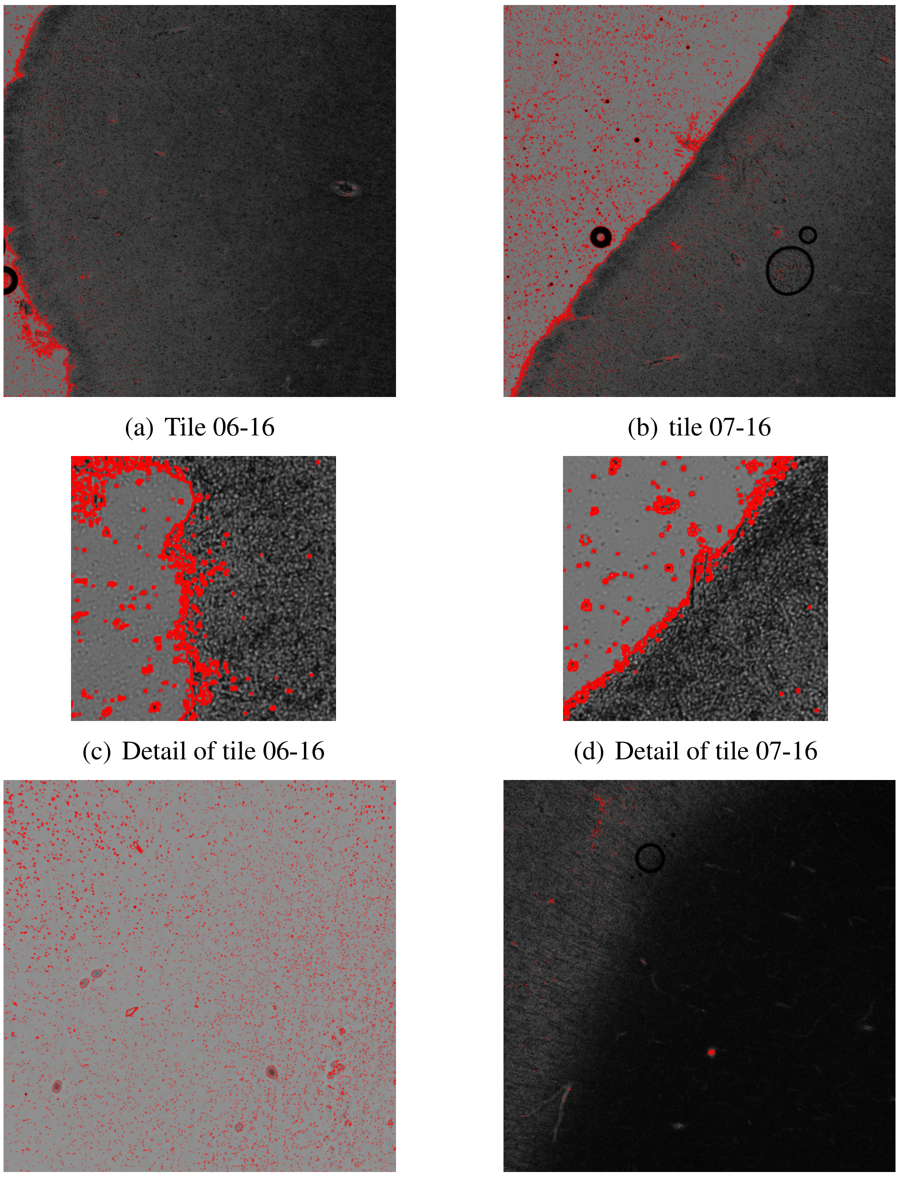


Figure 3.17: This training data has been used for the classification based segmentation of ilastik. The training information for the brain is the green line, for the background there are two red lines.

After the features have been chosen and evaluated for the images, training data has to be defined. These are parts of the image representative for each of the labels to be applied to the image. Ideally, the training data is only masked in some few images which are also representative for all other images. In this case, tile 06-16 has been used for training which can be seen in figure 3.17.

The result of this segmentation can be found in figure 3.18. The edges between brain and background are reasonable but this approach is prone to image noise if color or intensity are used as features. Other features did not work at all for these brain images because they cannot be used to identify the different image regions. The execution time is much smaller than for the level set methods.



(a) Tile 06-16

(b) tile 07-16

(c) Detail of tile 06-16

(d) Detail of tile 07-16

(e) Tile 00-00

(f) tile 08-14

Figure 3.18: ilastik finds reasonable edges but is influenced by image noise.

3.2.4 Summarizing Evaluation

In section 2.5 it has been outlined which requirements exist for the segmentation to be developed. In this section it will be evaluated how they are fulfilled by the tested tools on the one hand and the underlying methods on the other hand.

The first requirement is that brain should never be masked as background. The tests described in the last paragraphs have shown that this requirement is fulfilled by all methods and tools. Only in some cases like the region growing segmentation of Fiji, the extracted edges were moved some pixels into the brain. This problem can be solved easily by applying morphological operators like a dilation to the created mask.

A second reason for erroneously masked pixels is image noise which has become clear especially in case of thresholding based approaches. So it is important that the method copes well with image noise which has been listed separately as the fifth requirement. Therefore, these two requirements can be evaluated together.

Another requirement refers to the needed amount of manual input or interaction. In this case it has to be abstracted from the requirements of the tested tools to the needs of the approach itself. For some methods like thresholding it is obvious that only a fixed amount of input is needed independent of the number of images to be processed, namely the threshold value. Other methods are based on seeds or training data. Here the required amount of input has to be examined in detail.

The third requirement claims to find reasonable edges which is intrinsic working with segmentations. Particularly this aspect has already been pointed out in the evaluation of the resulting masks created by the tested software tools.

Fourthly, the approach has to cope with "single labeled" images, so tiles showing either brain or background but not both. Like with the next-to-last requirement, it has to be abstracted from the tool to the method to rate this aspect in order to figure out if a segmentation approach works well for the present brain images.

Another hint in addition to the requirements is to check the approaches for their parallelization capability in order to facilitate processing the immense amount of images in reasonable execution time. Furthermore, it should be evaluated if the tools are stable regarding variations of the configuration parameters and additional input like seeds. Not least, it has to be kept in mind which effort is needed to re-implement and adapt the algorithms to the given problem.

In the following, the tested software tools are compared and evaluated based on these criteria. As listed in table 3.5, a thresholding segmentation does not work well for the given brain images since this method is prone to image noise, i.e. the images show a lot of noise pixels. If another method copes with noise it depends a lot on the question whether it is based on a threshold or not.

In case of region growing segmentations, some of the tested methods cope with noise like Fiji Region Growing, but others not like ITK Connected Threshold. The difference between these tools is the way how thresholding is included into the algorithm. ITK Connected Threshold compares the intensities pixel-wise with a threshold value. Fiji Region Growing and ITK Neighborhood Connected take into account the already marked region respectively the neighborhood of a pixel, too. This further knowledge softens the effect of noise a lot. So in principle, region growing segmentations can cope with noise if the implementation is being aware of this problem.

The tested boundary-driven methods all cope well with noise. The classification based tool ilastik has the same problems with noise which can be found in some of the region growing segmentations and the thresholding tool. Since image noise is present in the images to be processed, this aspect has to be rated as a criterion for exclusion, so the thresholding and classification based methods will not be discussed further. They are not suitable for the given data.

For the remaining two variants, the needed manual input is of the same dimension. The region based methods as well as the boundary-driven need a distinct manual input for each image. In addition, the amount of input per image may vary. So dealing with this requirement, there is no difference if a region based or a boundary-driven method is used. Also the edges extracted by both approaches are equally good, their quality again depends on the implementation details. Both approaches can be implemented in a way that they can process “single-labeled” images and are stable concerning variations of input or parameter configurations.

The region based methods can be parallelized in the sense that the images are processed independently of each other. The boundary-driven approaches can be parallelized in the same way. Alternatively, all image tiles belonging to the same brain section can be processed distributed to several processors because mathematically this means a distributed calculation of partial differential equations. So again, both methods fulfill this criterion.

Differences between these two methods can be found taking a look at the needed execution time and the effort required to adapt the tool to the given data. The execution time of all tested region based segmentations is, with a span in seconds range, much shorter compared to all used boundary-driven tools which partly need minutes to process an image. Moreover, the needed effort to adapt the approach to the given data, is much higher for boundary-driven than for region based methods.

To sum up, the region based methods are most suitable to be used for a segmentation of the given brain images. They fulfill the requirements as good as others or better. A particular attention has to be paid for the handling of image noise. A comparison of the different region growing tools shows that a simple thresholding based approach is not sufficient but that the algorithm used in Fiji Region Growing delivers good results. Taking all these information into account, the development and implementation of the required segmentation tool should be derived from [AB94] which was basis for Fiji Region Growing. Nevertheless, a solution has to be found to reduce the amount of manual input in form of seeds to a dimension independent of the number of images to be processed. So an automated choice of seeds is required.

Table 3.5: Comparison of the tested software tools

Software Tool	Copes with Noise	Manual Input[1]	Reasonable Edges	Single-labeled Images	Parallelizable	Stable against Variations	Execution Time[4]	Effort for Adaption
				Thresholding				
GIMP	×	1	×	✓	✓[2,3]	○	1	low
				Region Based				
Fiji RG	✓	3	○/✓	○	✓[2]	✓	2	ok
ITK Conn. Thresh.	×/○	3	○/✓	✓	✓[2]	✓	2	ok
ITK Neigh. Conn.	○	3	○/✓	✓	✓[2]	✓	3	ok
ITK Confid. Conn.	×/○	3	○	✓	✓[2]	✓	4	ok
ITK Iso. Conn.	×	3	×	×	✓[2]	✓	5	ok
				Boundary-Driven				
MiaLite	○/✓	3	○/✓	✓	✓[2,3]	✓	7	high
ITK-SNAP	○/✓	3	○/✓	○	✓[2,3]	○	8	high
Fiji Level Sets	○/✓	3	○	✓	✓[2,3]	✓	8	high
				Classification				
ilastik	○	2	○	✓	✓[2,3]	✓	6	high

[1] ①None
②Varying but independent of the number of images
③Varying per image
[2] Independent processing of images
[3] Distributed processing of a brain section
[4] Time scale for execution times: ①($< 1s$) → ⑧(minutes)

4 Seeded Region Growing Segmentation

4.1 A basic Algorithm

The previous comparison of different segmentation approaches figured out that the seeded region growing method of Fiji based on [AB94] produces reasonable results for the present brain images (see section 3). Thence, the algorithm presented in this paper is used as basic algorithm for the segmentation tool to be developed. Certainly it has to be adapted to the needs induced by the given image data.

This region growing segmentation has been the first approach using seeds. Before, region growing methods were mostly based on "split and merge". The number of regions found by these algorithms cannot be influenced directly but only indirectly by modifying parameters. Therefrom, these methods tend to over-segment the images. In contrast, the seeded region growing provides the opportunity to define the number of regions explicitly.

The approach presented by Adams and Bischof is based on the constraint to extract regions "*as homogeneous as possible*" [AB94, p. 641]. It works for images of any shape and any dimension, so also for arbitrarily shaped parts of an image and for three-dimensional data.

The method starts with some seed points which are grouped into n sets $A_1, A_2, \ldots, A_n$, so each set may contain more than one seed pixel. The algorithm then tessellates the image into n connected regions whereat each region belongs to one of the A_i.

Afterward, the algorithm repeats one step until all pixels are labeled. In each iteration, one pixel is assigned to one of the sets A_i. Therefore, the set T of all non-labeled pixels is defined as in equation (4.1). It contains all pixels which have not yet been added to any of the A_i but which are directly adjacent to of at least one A_i. $N(x)$ is the set of all direct neighbors of a pixel at position x.

$$T = \left\{ x \notin \bigcup_{i=1}^{n} A_i \mid N(x) \cap \bigcup_{i=1}^{n} A_i \neq \varnothing \right\} \tag{4.1}$$

If pixel $x \in T$ borders on exactly one of the A_i, let $i(x)$ be the index for which $N(x) \cap A_{i(x)} \neq \varnothing$. Otherwise, if x is neighbor of two or more of the A_i, the index $i(x)$ is defined as the index for which $N(x) \cap A_{i(x)} \neq \varnothing$ and a measure $\delta(x)$ is minimized. This measure $\delta(x)$ is a homogeneity or similarity measure defining "*how different x is from the region it adjoins*" [AB94, p. 642]. Adams and Bischof use the following definition of δ with $g(x)$ being the gray value of pixel x:

$$\delta(x) = \left| g(x) - \underset{y \in A_{i(x)}}{\operatorname{mean}} \left[g(y) \right] \right| \tag{4.2}$$

Alternatively, pixels neighboring two or more of the A_i can be assigned to a new set B containing all boundary pixels.

In the end of each iteration, the pixel $z \in T$ with

$$\delta(z) = \min_{x \in T} \{\delta(x)\} \tag{4.3}$$

is labeled corresponding to $A_{i(z)}$ and appended to this set. The definitions (4.2) and (4.3) assure that the regions A_i are as homogeneous as possible and, by the use of $N(x)$, that each of the regions is connected.

Adams and Bischof suggest to use a sequentially sorted list (SSL) for the implementation of their algorithm. This list contains the positions of all pixels $x \in T$ sorted by $\delta(x)$ so that the first pixel in the list is the one having the largest similarity to one of the A_i. The implementation can be done as follows:

```
group the seeds into $n$ sets $A_1, A_2, \dots, A_n$
label the seeds according to the sets they are assigned to
add the neighbors of the seeds to the $SSL$
$SSL$ is not empty
    take the first pixel $y$ from SSL
    $check \leftarrow$ all neighbors $n_k$ of $y$ have the same label $L_i$ of set $A_i$, no
    label or border label $L_B$?
    test neighbors of $y$: $check$
    true
        set $y$ to label $L_i$
        update the mean of $A_i$
        add all unlabeled $n_k$ sorted by $\delta$ to $SSL$ which
        are not yet in this list
    false
        set $y$ to the boundary label $L_B$
        ∅
```

Figure 4.1: Structure Chart: Seeded Region Growing by Adams and Bischof

It has to be noticed that the entries of the SSL are sorted based on the measure $\delta(x)$ and hence based on the means of the A_i. Although the means are updated in each iteration, the SSL is not re-sorted. The authors of the paper suppose that "*this leads to negligible difference in the results, but greatly enhanced speed*" [AB94, p. 643]. In addition, they place emphasis on the fact that each pixel is processed only once apart from the visiting of neighbor pixels.

Adams and Bischof describe different properties of their seeded region growing segmentation, too. They point out that artifacts are subsumed by the surrounding regions given that no seed is placed within an artifact. This matches well the fact that the present brain images show artifacts like air bubbles which should not be interpreted as own regions.

Furthermore, the authors figure out how critical the choice of seeds is. They come to the conclusion that in general, seeds have to be representative for their region to achieve reasonable results. Single pixels as seeds may result in incorrect segmentation masks if image noise is present and a noise pixel is used as seed, because this pixel is no good estimation for the region's mean. This problem can be solved choosing small regions as seeds. If these regions are sufficiently large, they provide a stable estimation of the regions' means. The definition of $\delta(x)$ has to be more advanced in case that the image noise is not of equal variance in each region.

Adams and Bischof also test the stability of the algorithm with an artificial image containing Gaussian noise. They compare the results of their seeded region growing method to other known segmentation approaches varying the seed position as well. It turned out that the seeded region growing tool achieved the best results among the compared ones. Dealing with a higher level of noise, the results stay good if also the seed regions are enlarged. The exact positions of the seeds have no influence on the segmentation result.

Up to this point, the seeded region growing algorithm can be characterized as a semi-interactive, three-step procedure. First, the seeds have to be chosen. Secondly, the region growing is performed automatically. In a third, optional step the result may be corrected, e.g. by splitting one of the extracted regions with another run of seeded region growing.

Neglecting the third step, the choice of seeds requires the only manual interaction respectively input. The authors of the paper also deal with the question how to automate this step: If there is further, high-level knowledge of an image to be segmented, it can be used to find good seeds. The paper provides the example of an image showing a dark object on a brighter background. So the darkest pixel of the darkest part of the image respectively the brightest pixel of the brightest part can be used as seeds for object and background. It is important to ensure that for all sets of seeds A_i the condition $A_i \cap A_j = \varnothing \ \forall i \neq j$ is fulfilled.

In general, automated seeded region growing can be interpreted as the correction of a previous other segmentation. The resulting mask of this first segmentation can be post-processed, e.g. by morphological operations like a dilation, to gain seeds for the region growing which is performed afterward. [AB94]

4.2 Adaption of Seeded Region Growing

4.2.1 Automated Choice of Seeds

To adapt the seeded region growing approach by Adams and Bischof to the requirements induced by the human brain images on hand (see section 2.5), several aspects of the algorithm have to be considered. The most important task is to automate the choice of seeds in order to minimize the required amount of manual input so that it is independent of the number of image tiles to be processed.

There is already a bunch of different automations to the choice of seeds in literature. To give some examples, Feng et al. [FFJ05] developed a method for semantic video object segmentation. They first perform an initial segmentation with a competitive learning neural network. Pixels in the middle between the boundaries obtained by this prior segmentation are then used as seeds for the actual seeded region growing segmentation. This method is based on the fact that consecutive frames of a video are very similar.

Fan et al. [FYE$^+$01] work with color images. They extract colored edges, mark each connected edge region with an own label and use the centroids between two adjacent edge regions with different labels as seeds. In a later publication (see [FZBH05]), they improve the results of this approach by smoothing and thinning the edges in advance to avoid over-segmentation. To use this method, it is important to find reliable and clear edges. If there are also a lot of sharp edges inside the object, this approach fails and over-segments the images.

Abdelsamea et al. [Abd11] developed a method for 2D biomedical image segmentation. They also use some non-linear image filters to reduce noise. Then an Otsu segmentation is performed which is a threshold-based approach. Pixels with the highest similarity within their neighborhood are then taken as candidate seeds. A K-mean clustering is performed using these candidates to get the best and most reliable seed points. The mask of the Otsu segmentation is used as training data for this algorithm. To use this method, the intensity variances within the different objects have to be in the same order of magnitude.

As shown by these examples, most approaches of automated seeded region growing are based on further, high-level knowledge of the images to be processed like specialties of edges, color and intensity. Moreover, in some cases there is not only one image to be segmented but a list of several images which are interrelated because they are frames of a video or an image series. Consecutive images are mostly very similar. Due to this approach based on specific, high-level knowledge, it is not possible to take on an already existing automated method without adaptations.

Hence, the specialties of the human brain image tiles have to be analyzed in order to develop an automated choice of seeds. Taking a look at the human brain image tiles, it is conspicuous that the background is lighter colored than the brain; the artifacts are even darker. Therefore, it might be possible to use the image histogram and thus a thresholding approach to find reliable seeds. Another indication for this is that in the comparison of different segmentation methods (see section 3) it figured out that in general thresholding works for the present images but that the mask is too much affected by noise. However, it has to be kept in mind that there are tiles showing only brain tissue or background but not both. So the simple calculation of the image tile histogram in order to choose a threshold will not lead to success.

To solve this problem, a joint histogram of all image tiles is computed instead of individual histograms. This has the positive side effect that also variations of illumination are balanced.

Figure 4.2 shows the joint histogram of ten sections containing 5356 tiles in total. The large peek at an intensity of about 3750 belongs to the background because the brightly colored background has a small variance of intensity which results in a large peek at a comparably high intensity. The brain intensities vary a lot. Since the brain is darker than the background, the wide but flat range with the intensities 0 to 3500 corresponds with this object. So the threshold value should be chosen at about $T = 3500$. This is approximately the same threshold which has been used by several of the compared segmentation methods. Based on this joint histogram, the developed automated choice of seeds consists of four steps which will next be discussed in detail.

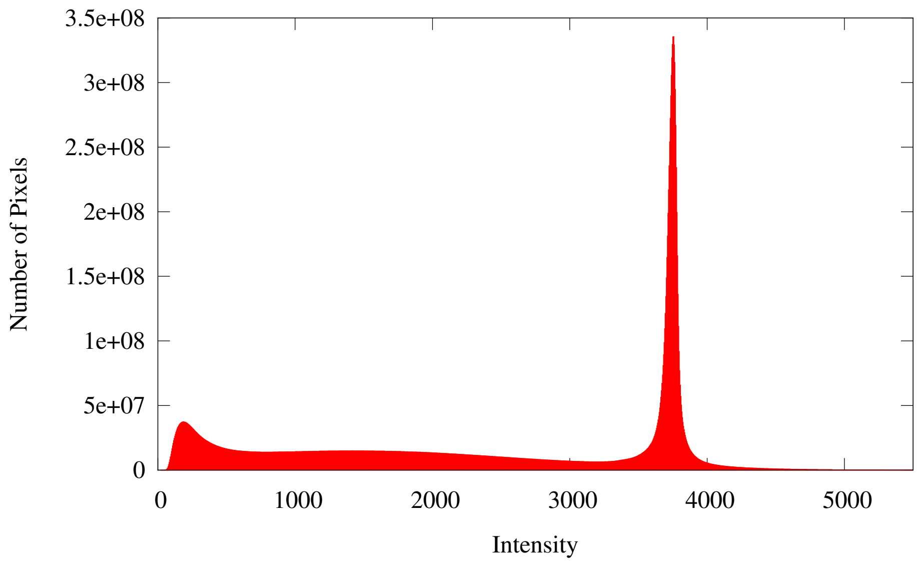


Figure 4.2: This joint histogram combines the intensity information of ten sections containing 5356 image tiles.

Calculation of the Joint Histogram

The first step on the way to automatically chosen seeds is the calculation of the joint intensity histogram of a large amount of image tiles showing the same brain. At best, all available tiles are used to get best results based on reliable statistics.

Structure chart 4.3 outlines the algorithm for the computation of the joint histogram of sections s_{start} to s_{end}. The histogram has got *BINS* bins with a bin width of *BIN_WIDTH*. Each section has got $size_x \times size_y$ tiles. Given that the mosaic is rectangular shaped, the coordinates of the image tiles within the tile mosaic are in $[offset_x, offset_x + size_x - 1] \times [offset_y, offset_y + size_y - 1]$.

```
initialize the array $histogram$ with $BINS$ zeros
for $section = s_{start}, section < s_{end}, section += 1$
    for $x = offset_x, x < offset_x + size_x, x += 1$
        for $y = offset_y, y < offset_y + size_y, y += 1$
            read image tile at position $(x, y)$ of section $section$
            for each $pixel$ position in $tile$
                $histogram[\ tile[pixel] / BIN\_WIDTH\ ] += 1$
```

Figure 4.3: Structure Chart: Computation of the Joint Histogram

Choice of the Threshold Value

If the histogram has been calculated, the threshold value between brain and background intensities has to be found. This step has to be done manually by the user. Tests figured out that the local minimum between the flat wide region and the large peek is a good threshold to start with. Hence, the local minima within the histogram are computed in addition to the histogram itself to assist the user defining the threshold value. At best, it is sufficient if the user removes all other minima from the list which might occur in the histogram. The number of local minima depends on the used histogram bin width and can vary from a double-digit to triple-digit count.

The local minima and the histogram are written to separate files. This allows to reuse the histogram, e.g. in order to fine-tune the segmentation parameters or to modify the threshold value without having to wait for another calculation of the histogram. The choice of the threshold value is the only manual interaction respectively input needed by the developed segmentation tool. This input is independent of the number of image tiles to be processed. Figure 4.4 shows the joint histogram of figure 4.2 together will all local minima as the result of this second step on the way to automatically found seeds.

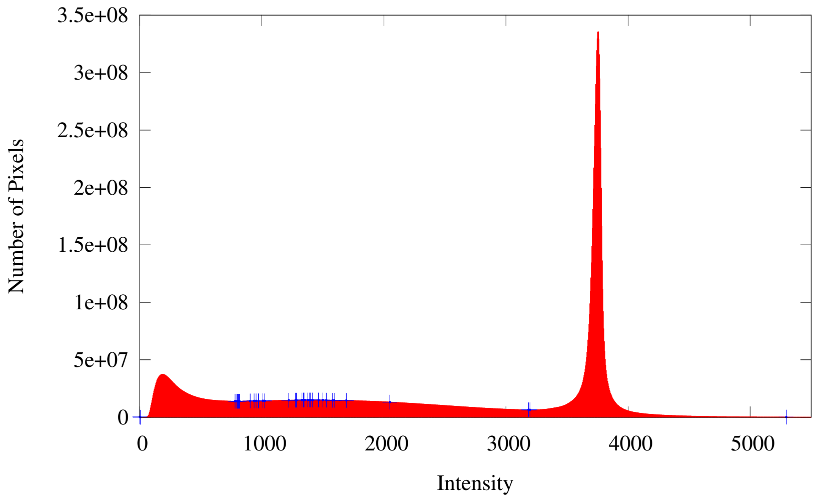


Figure 4.4: This joint histogram combines the intensity information of 10 sections containing 5356 image tiles. Furthermore, all local minima are plotted as blue crosses.

Initial Choice of Seed Candidates

After the choice of a threshold value, initial seeds, also called seed candidates, are chosen. The threshold divides the intensities into two intervals. Tests showed that pixels whose intensity is in the middle of the brain interval with a high probability belong to the brain. The same applies to pixels with the medium background intensity. The only exceptions are noise pixels.

Based on this knowledge, it is reasonable to define pixels as the best seeds for the classes whose intensity is the medium brain respectively background intensity. Pixels with an intensity close to the threshold may belong to any of the regions. The darkest and brightest intensities that can be found in the images belong with a high probability to noise pixels. Summing up these experimental results, a measure (equation (4.4)) has been developed which defines how suitable a pixel is to be a brain or background seed.

$$\begin{aligned} m(x,y) &= \begin{cases} \frac{i(x,y)-q_{0.5}}{q_{0.5}-q_\alpha} & , i(x,y) \leq q_{0.5} \\ \frac{q_{0.5}-i(x,y)}{q_{1-\alpha}-q_{0.5}} & , i(x,y) > q_{0.5} \end{cases} \\ &= \max\left(\frac{i(x,y)-q_{0.5}}{q_{0.5}-q_\alpha}, \frac{q_{0.5}-i(x,y)}{q_{1-\alpha}-q_{0.5}}\right) \end{aligned} \tag{4.4}$$

This measure is computed individually for each pixel and stored in a measure image. It uses the median $q_{0.5}$ of the respective intensity interval within the histogram as well as the α- and $(1-\alpha)$-quantiles with a user-given value α. With these statistics, the measure m is computed for a pixel at position (x,y) with intensity $i(x,y)$.

The measure is the normed distance of the pixel's intensity to the median intensity. Normalization in this case means that the measure is 0 for a pixel with the median intensity and 1 for pixels with an intensity in the quantile bins. For all pixels outside the interval $[q_\alpha, q_{1-\alpha}]$, the measure value is greater than one. With this definition, each pixel with $m(x,y) \leq 1$ is a candidate seed. The candidates with the smallest values are the comparably best seeds. Pixel with intensities similar to the threshold are neither seeds for the brain nor for the background region.

There are two different measure images, one based on the brain and one based on the background intensity interval. If candidate seeds would be chosen based on these measure images without further modifications, also noise pixels were seed candidates. An important characteristic of image noise is that it mostly consists of isolated faulty pixels. So if not only a particular pixel and its intensity were considered to compute the measure value but also the intensities respectively measures of pixels in a defined neighborhood around this pixel, the noise pixels could be filtered out of the candidate set.

To include the neighbor pixels in the calculation of the measure, it is first computed individually for each pixel. Afterward, a linear smoothing operation is applied to this first measure image. The simplest linear smoothing is an averaging of a pixel's and its neighbors' values. Linear smoothing operations are mathematical convolutions (see [GW08, p. 146ff]).

A convolution consists of a convolution kernel or, in terms of image processing, filter $w(x,y)$ and the function or image $f(x,y)$ with (x,y) being the pixel position. Interpreting an image as a discrete function, the convolution $w * f$ is defined as in equation (4.5).

$$w\left(x,y\right) * f\left(x,y\right) = \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} w\left(i,k\right) \cdot f\left(x+i, y+k\right) \tag{4.5}$$

The kernel w has the size $m \times n$ with m and n being odd, positive integers. For the purpose of image smoothing, the kernel is mostly chosen as quadratic, i.e. $m = n$. To give an example, the averaging filter mentioned above with $m = n = 3$ has got the kernel in figure 4.5. A weighted averaging is also possible, e.g. with a kernel as in figure 4.6. To avoid a shading or brighten of the image, the sum of all kernel elements has to be one.

In terms of image processing, a convolution correlates with a linear filter which works as follows. Since the filter has always an odd number of rows and columns, it has got a center value. A filter can lay over the image so that each filter value covers an image pixel. The filter is moved over the image so that its center is located once at every pixel position of the image. At each position, every entry of the filter is multiplied with the covered image value, e.g. the intensity of the pixel. The results of all products for one positioning of the filter are summed up. This sum is then written to another image at the position of the filter center within the original image.

$$\frac{1}{9}\cdot\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Figure 4.5: Filter used for an average smoothing of the image preserving the original brightness of the image.

$$\frac{1}{16}\cdot\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

Figure 4.6: Filter used for a weighted average smoothing of the image preserving the original brightness of the image (Gaussian filter).

The calculation of the initial measure and the following smoothing operation can be combined to a single convolution. This accelerates the whole program because the image has to be traversed only once. In addition, less memory accesses are needed since the image is too large to store it in the cache as a whole. Instead, the measure value is computed multiple times for each pixel.

The seed measure images are computed using convolution equation (4.6). Here not the original intensity values are smoothed but the measure function of the image.

$$\begin{aligned} w(x,y) * f_2(x,y) &= \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} w(i,k)\cdot f_2(x+i,y+k) \\ \text{with} f_2(x,y) &= \max\left(\frac{f(x,y)-q_{0.5}}{q_{0.5}-q_{\alpha}}, \frac{q_{0.5}-f(x,y)}{q_{1-\alpha}-q_{0.5}}\right) \end{aligned} \tag{4.6}$$

It figured out that a weighted averaging filter w works best, if for a central value p all other entries have the same value $\frac{1-p}{m\cdot n-1}$. This is equivalent to an average filter with a higher weighted central value. The central weight has been chosen to $p = 0.1$ and $m = n = 9$ as in 4.7.

One may notice that applying linear filters does not work directly and without modifications for border pixels of the image. In case of the given brain images, all pixels within the $m = n$ pixels wide border of the image can be ignored in the calculation of the seeds. This is possible due to the large overlapping of the adjacent image tiles in the mosaic.

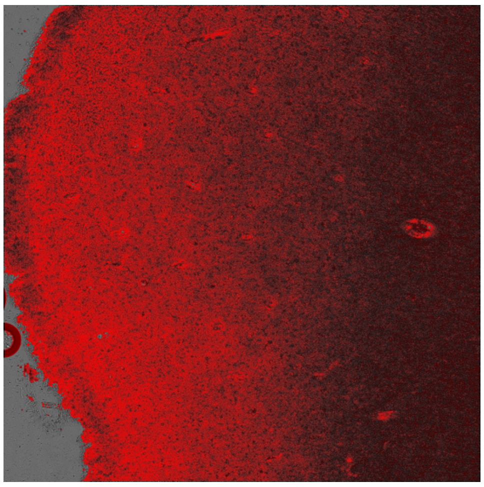
Figures 4.8 and 4.9 illustrate the calculated measures for background and brain seeds of the same example image tiles which have been presented in the previous chapter. The seeds for the background region in figure 4.9 are all located in the actual background of the images. But for example in figure 4.8(b) it is visible that some of the brain seeds can be found in the dirt spots within the background. These erroneously determined brain seeds have to be eliminated in order to achieve best segmentation results. This task will be discussed in the following section.

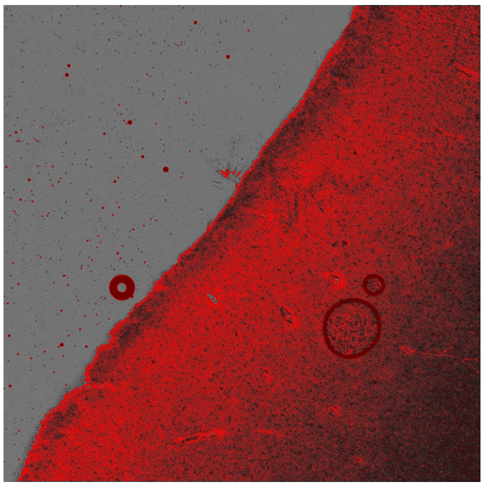
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	**p**	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ
δ	δ	δ	δ	δ	δ	δ	δ	δ

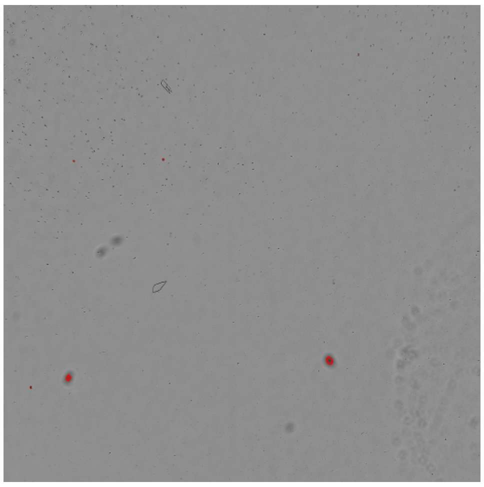
with $p = 0.1, \delta = \frac{0.9}{80} = 0.01125$

Figure 4.7: This filter *w* is used for the smoothing of the measure image.

In addition, it has to be noticed that the large air bubbles always contain brain seeds but never background seeds. This way it is guaranteed that the air bubbles within the brain region are not cut away from the brain and the brain region does not get holey.


(a) Tile 06-16


(b) Tile 07-16


(c) Tile 00-00

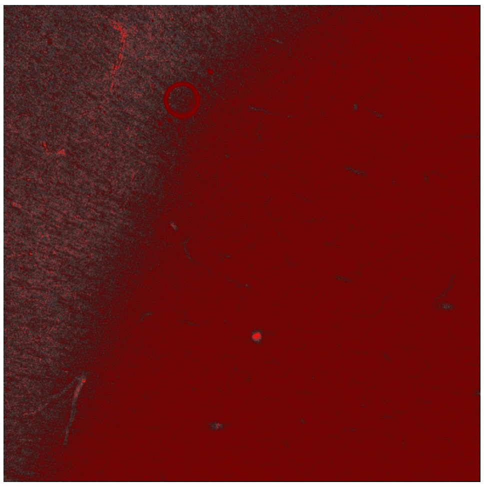
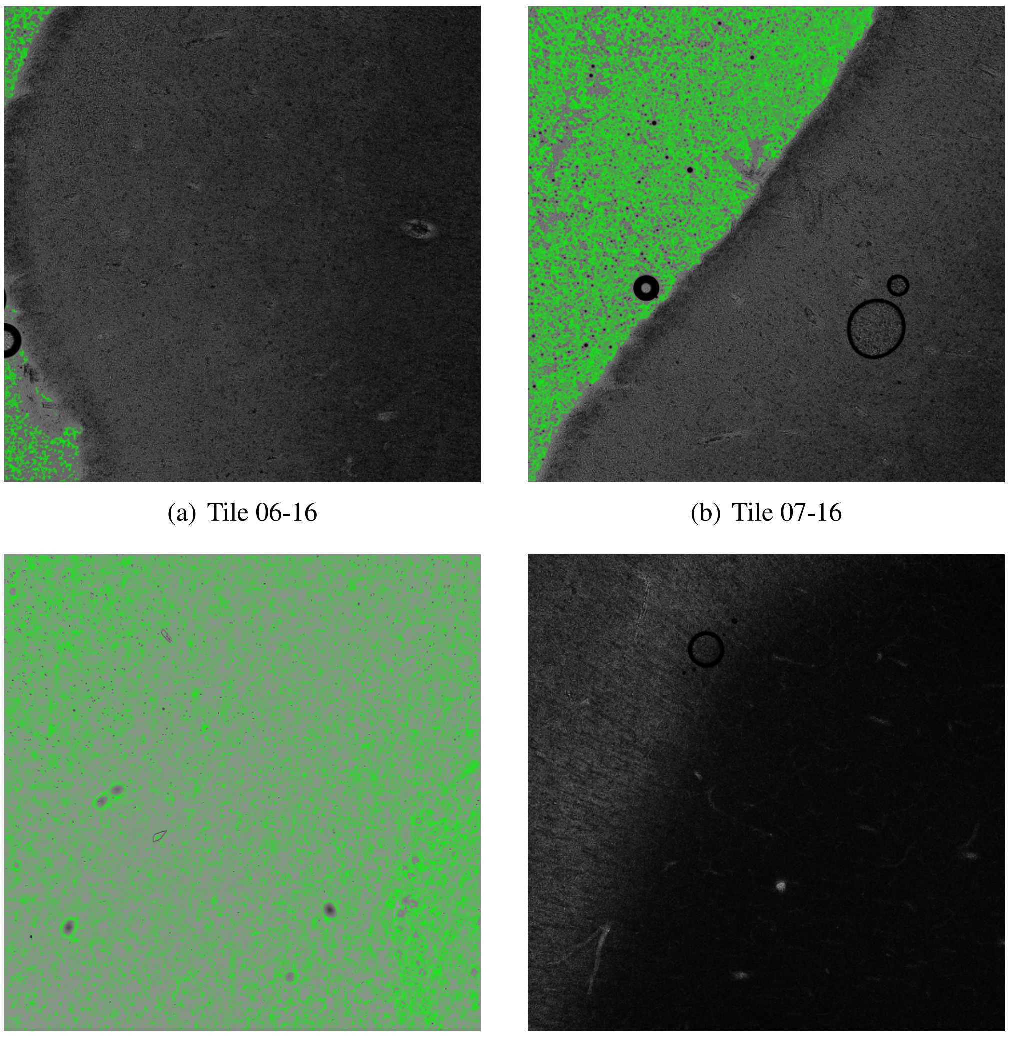

(d) Tile 08-14

Figure 4.8: These measure images are calculated for the choice of **brain seeds**. The original images are visible in the background. The measure values are illustrated as overlays. Red pixels correspond to measure value 0, so good seed pixels. For measure values in $]0,1]$, a gradient from red to transparent is used. All pixels with measure values larger than 1 have a transparent color in the overlay.



(a) Tile 06-16

(b) Tile 07-16

(c) Tile 00-00

(d) Tile 08-14

Figure 4.9: These measure images are calculated for the choice of **background seeds**. The original images are visible in the background. The measure values are illustrated as overlays. Green pixels correspond to measure value 0, so good seed pixels. For measure values in $]0, 1]$, a gradient from green to transparent is used. All pixels with measure values larger than 1 have a transparent color in the overlay.

Final Choice of Seeds

As already mentioned, some of the initial seeds chosen by the use of the measure images have to be sorted out. All these faulty seeds belong to small regions of adjacent seed pixels located in the background but marked as brain seeds. So a way has to be found to eliminate these small "seed isles".

All brain seeds to be erased have in common that there are only few other brain seeds in the respective neighborhood. Based on this knowledge, the following algorithm in structure chart 4.10 has been developed to eliminate the seed isles.

```
$divisor \leftarrow (radius \cdot 2 + 1)^2$
for $x = 0, x < width, x+ = 1$
    for $y = 0, y < height, y+ = 1$
        $newMeasure(x,y) \leftarrow measure(x,y)$
        $measure(x,y) \leq 1.0$
        true
            $sum \leftarrow 0.0$
            $count \leftarrow 0$
            for $a = -radius, a \leq radius, a+ = 1$
                for $b = -radius, b \leq radius, b+ = 1$
                    $measure(x+a, y+b) \leq 1.0$
                    true
                        $count \leftarrow count + 1$
                        $sum \leftarrow sum + (1.0 - measure(x+a, y+b))$
                    f.
                        $\varnothing$
            $count < threshold \cdot divisor$
            true
                $newMeasure(x,y) \leftarrow 1.0 + (1.0 - threshold) \cdot \frac{sum}{divisor}$
            false
                $newMeasure(x,y) \leftarrow 1.0 - \frac{sum}{divisor}$
        f.
            $\varnothing$
```

Figure 4.10: Structure Chart: Final Choice of Seeds

The algorithm computes a new brain measure image (*newMeasure*) using the already existing one (*measure*). If a pixel has a measure value larger than one, so if it is not a seed for the brain, the measure value is taken over from the old image.

For all other pixels, the number of all seed pixels in within a square neighbor region with a size of $(\text{radius} \cdot 2 + 1) \times (\text{radius} \cdot 2 + 1)$ is determined. Furthermore, the sum of their inverted measure values is computed for all these neighboring seeds. So with n seeds in the defined neighborhood, this sum is approximately $n \cdot 1.0$ given that the seeds are good with intensities similar to the median intensity, and $n \cdot 0.0$

if the seed intensities are close the threshold between the brain and background intensity intervals.

If the number of neighboring seeds exceeds a defined threshold, the average measure value of the used neighborhood is chosen as the new measure value for the respective pixel which is

$$1.0 - \frac{\text{sum}}{\text{divisor}}$$

with divisor $= (\text{radius} \cdot 2 + 1)^2$ being the number of pixels within the neighborhood. This way seeds are rated higher if they have other comparably good seeds in their surroundings. Otherwise, the new measure value is set to

$$1.0 + (1.0 - \text{threshold}) \cdot \frac{\text{sum}}{\text{divisor}}.$$

It can be proven that this value is always greater than one, so this pixel will not be used as a seed pixel afterward: As explained above, it applies

$$\text{sum} \in [0, n] \text{ with } n \in [0, \text{divisor}]$$

and thus

$$\frac{\text{sum}}{\text{divisor}} \in [0, 1].$$

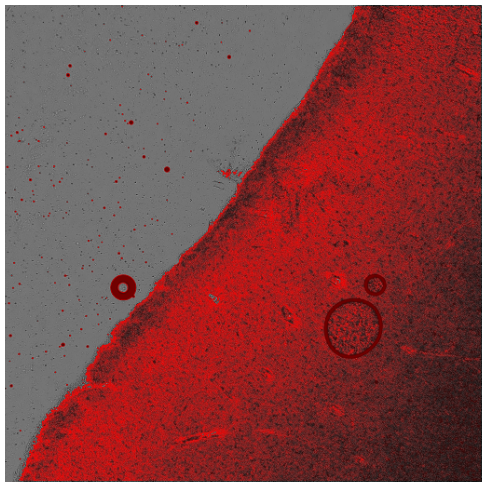
If the threshold is chosen in $[0, 1]$,

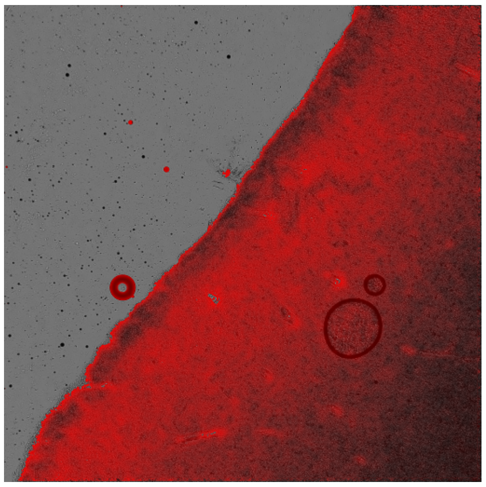
$$\left((1.0 - \text{threshold}) \cdot \frac{\text{sum}}{\text{divisor}}\right) \in [0, 1 - \text{threshold}]$$

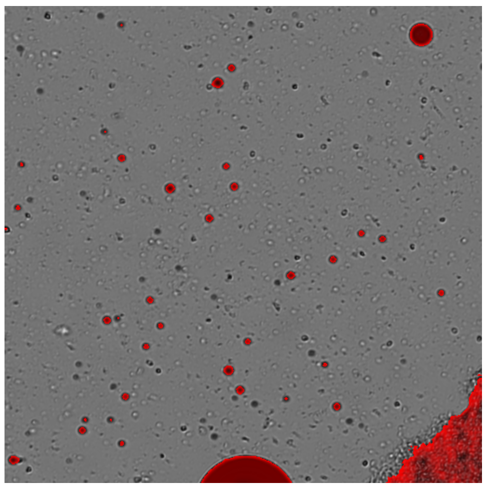
and thus for the whole expression it applies

$$\left(1.0 + (1.0 - \text{threshold}) \cdot \frac{\text{sum}}{\text{divisor}}\right) \in [1, 2 - \text{threshold}].$$

The differences between the initial and the final measure images are illustrated in figure 4.11. The small brain seed isles in the background are erased. Moreover, the new measure image is smoother. This is not the effect of a linear smoothing but of the way to rate seeds as higher reliable if they have a lot of other good seeds in their neighborhood as described above.


(a) Initial Measure


(b) Final Measure


(c) Detail of 4.11(a)

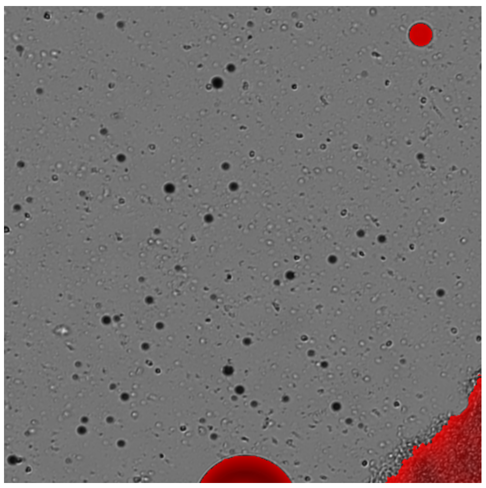

(d) Detail of 4.11(b)

Figure 4.11: On the left side, the initial brain measure of tile 07-16 is displayed. The detail image 4.11(c) demonstrates the small "seed isles" which are a result of dirt and artifacts in the background. The pictures on the right side show the effect of the seed elimination. The seed isles are vanished, the rest of the measure is smoothed.

4.2.2 Similarity Criterion

Region growing algorithms always include a similarity criterion and a growing operator. The definition of these two operations influences the resulting segmentation mask a lot. In simple region growing approaches as described in [GW08], the growing operator is applied first, so the next pixel to be considered is chosen. Afterward, the similarity criterion is used to check whether the pixel is similar enough compared to the region to extend it with this pixel.

Adams and Bischof describe in [AB94] an alternative way how to use a similarity criterion as already described in section 4.1. If a pixel has been added to a region, the growing operator chooses the neighboring pixels. These neighbors are then inserted into a sequentially sorted list. The similarity criterion is used as sorting criterion. The algorithm always takes the first element from the list and decides based on the labels of neighboring pixels with which label the respective pixel should be marked.

This procedure has been adapted to the given needs. Adams and Bischof use a similarity criterion δ to sort the list (see equation (4.3) in section 4.1). This criterion compares the intensity of a pixel with the mean intensity of the already marked region. This definition has the disadvantage that the mean has to be updated with every newly added pixel.

The measures defined above already contain the information of how well a pixel fits to one of the classes “brain” and “background”. Hence, the two measure images are used instead of the function δ as similarity criterion. This has the additional advantage that image noise and dirt particles are already filtered out by these discrete image functions so that the segmentation result is less prone to those undesired effects.

In detail, the measure images are used as follows. The first pixel of the sorted list is extracted. The labels of the eight adjacent pixels in the resulting mask image, the so-called 8-connected neighborhood, are compared. Possible labels are `undefined`, `brain`, `background` and `border`. All pixels are labeled as `undefined` in the beginning. A pixel with `brain` and `background` neighbors is marked as `border`. If only one of these two labels is present in the 8-connected neighborhood, the pixel gets this label. Every pixel, apart from the seeds, has got at least one adjacent pixel with the labels `brain` or `background`.

If a label has been chosen for the considered pixel, it is written to the mask image. If the label is one of `brain` or `background`, the respective corresponding measure image is used to insert the neighboring pixels chosen by the growing operator (see section 4.2.3) into the sorted list, e.g. in case of a `brain` pixel, the brain measure values of the adjacent pixels are used to add them to the list.

This way, a pixel can occur more than once in the sequentially sorted list because a pixel is added by different neighbors. There are two ways how to handle multiple occurrences. Either multiple adding of a pixel is prohibited which results in additional effort because the list has to be traversed to find out if the pixel has already been added. Or a pixel can be inserted into the list more than once, so that it has to be checked for each pixel extracted from the list whether this pixel still has got the `undefined` label. Otherwise the pixel is skipped because it has already been handled. Since the effort for the traversal of the list may be high in case of large images, it has been decided to use the second variant.

4.2.3 Choice of the Growing Operator

Besides the similarity criterion, also the chosen growing operator strongly influences the quality of the segmentation result. Adams and Bischof use an 8-connected neighborhood to choose the next pixels to be inserted into the sequentially sorted list. The same neighborhood definition is used to determine the label of the actually processed pixel. [AB94]

This definition of a growing operator has been inherited for the labeling case. It allows the detection of border pixels already during the growing process. However, using this operator to add further pixels to the list, it may result in two regions which are only connected at the joint corner of two pixels as it is illustrated in figure 4.12. If these two regions result from different seed pixels, the mask is correct. If they originate from the same seed, the illustrated result is worth discussing because it is subjective to define the white regions as connected or unconnected.

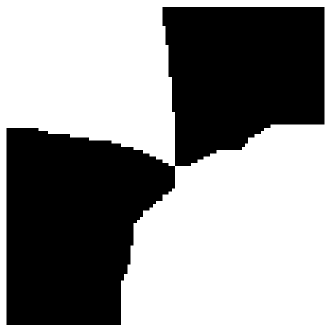


Figure 4.12: The two white regions are only connected to each other at the joint corner of two pixels but not at a joint edge.

Another reason to re-define the growing operator for this use case is the way how multiple occurrences of a pixel within the sequentially sorted list are handled. In the previous section it has been outlined that a multiple adding of a pixel to the list is allowed. If an 8-connected neighborhood is used, a pixel can be found up to eight times in the list.

Due to these reasons, it has been decided to use a smaller neighborhood than Adams and Bischof, namely a 4-connected one. Hence, the pixels above, below, to the left and right of the actually processed one can be added to the sorted list. This way each pixel may occur at most four times in the list. Two regions are connected at least at the joint edge of two pixels.

4.3 Processing Pipeline

The various main steps of the developed seeded region growing algorithm have been outlined in the previous sections. Before one may focus on the review of the requirements (see section 4.4) and on optimizations of the runtime behavior (see section 4.5), it has to be examined how these steps are arranged in a processing pipeline.

Figure 4.13 shows the processing pipeline containing all steps which have been described so far. It consists of two main parts which are actually two different executable programs requiring manual interaction in between.

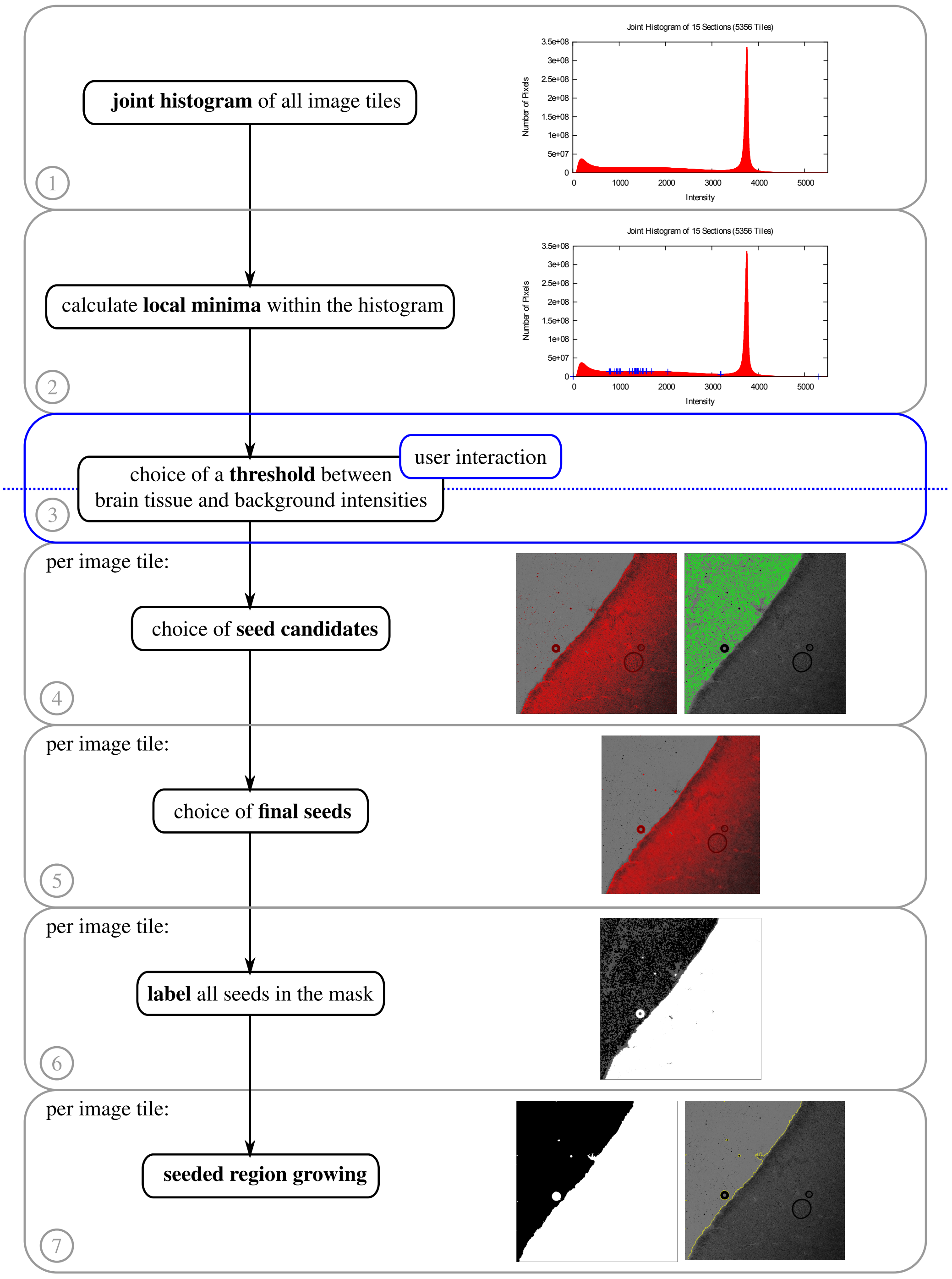
The first part is the calculation of the joint intensity histogram of all image tiles (step 1) and the local minima within these classed dataset (step 2). After this first main part, the user has to define a threshold which differentiates between brain tissue and background intensities (step 3). The local minima serve as a hint for potentially good thresholds. This user interaction is the reason why the software tool is divided into two executables. It is the only step requiring manual interaction.

The second main part is executed independently for each image tile. First, the seed candidates are determined (step 4). To get these seeds, two measure images per tile are computed. The next step is the choice of final seed points out of the seed candidates (step 5) which is only done for the brain similarity measure because the background seeds are already correct.

The intuitive way would be now to choose the best seed out of all, perform the region growing starting at this pixel and repeat this procedure until the whole image is segmented or no further seed is available. However, this approach needs a lot of time for execution. The same result can be reached in a much shorter time if all available seeds are directly masked (step 6). It is valid to do so because all seeds would be masked in any case with the corresponding label.

Dealing with the terms introduced in [AB94], all seeds for the brain tissue are added to a set A_0, all seeds for the background to another set A_1. A pixel is a seed, if the corresponding value in one of the measure images is in $[0, 1]$. Due to the definition of the measures, a pixel can be seed for one of the two classes or for none, but not for both.

After the seeds have been found and labeled in the output mask, the seeded region growing is performed as the final step (step 7). Therefore, all pixels which are in the 4-connected neighborhood of a seed but not marked as such, are added to a sequentially sorted list. Afterward, the growing is processed until all pixels are labeled with another label than `undefined`. Only the edges of the mask image stay marked with the `undefined` label which has already been discussed before.



joint histogram of all image tiles
calculate local minima within the histogram
choice of a threshold between brain tissue and background intensities
user interaction
per image tile:
choice of seed candidates
choice of final seeds
label all seeds in the mask
seeded region growing
Joint Histogram of 15 Sections (5356 Tiles)
Number of Pixels
Intensity
1
2
3
4
5
6
7

Figure 4.13: The processing pipeline of the developed seeded region growing algorithm consists of two main parts. The first one is the calculation of the joint histogram of all tiles (step 1) and of the local minima within the histogram (step 2). Afterward, the user has to choose a threshold between brain tissue and background intensities (step 3) using the minima as a hint for possible thresholds.
The second main part consists of the choice of seed candidates (step 4), the following choice of final seed points (step 5), labeling the seeds in the output mask (step 6) and the growing process (step 7).

4.4 Review of the Requirements

After the processing pipeline of the developed seeded region growing segmentation has been described in detail, it has to be analyzed if this tool fulfills the requirements which have been worked out in section 2.5.

The first requirement to be evaluated is that it has to be rated worse if brain pixels are marked as background rather than the other way round. This requirement is connected to the third one, namely that "*the algorithm has to extract reasonable edges between brain and background although there is a smooth transition between the regions*". To review these two aspects, the created masks have to be checked optically. Figure 4.14 shows the resulting masks of the four example image tiles. In figure 4.15, the edges between the black and white mask regions are painted as an overlay into the original tiles showing both brain tissue and background to ease an optical evaluation.

The edges found by the developed segmentation definitely follow the actual object boundaries. They are comparable to or even more precise than the edges extracted by the formerly tested tools. Thus this requirement is entirely fulfilled. But it is visible in figure 4.15(c) that by trend the edges have to be moved in some tiles a few pixels into the background region in order to ensure that also for thinner cut parts of tissue, no brain tissue is cut away by labeling.

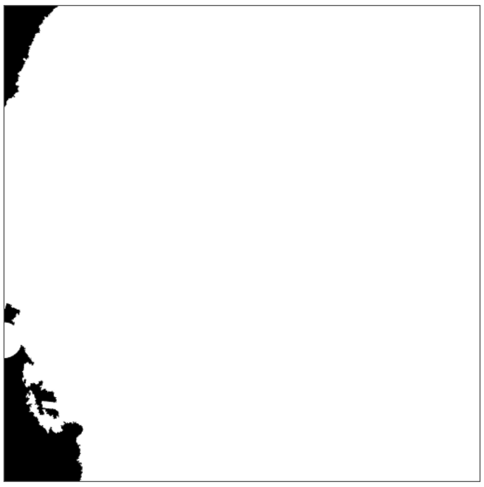
Therefore, an enlargement of the brain region has been added to the processing pipeline as a last optional step. It is implemented in form of a dilation with a user-defined range. A dilation is a morphological operation of image processing defined as

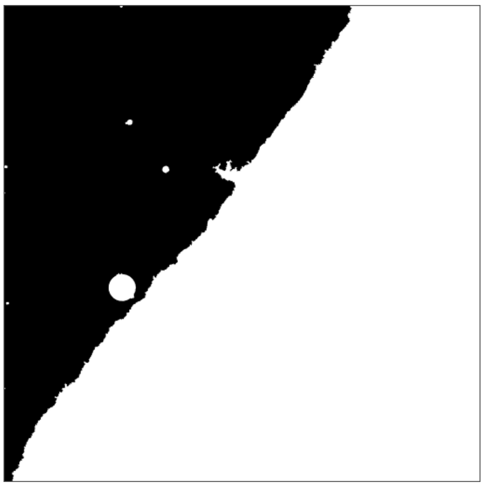
$$[f \oplus b]\,(x,y) = \max_{(i,k)\in b} \{f(x-i, y-k)\} \tag{4.7}$$

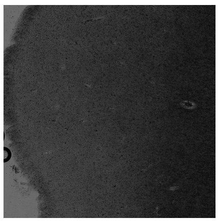
with f being the image to be dilated by a structuring element b. A structuring element is a flat, so two-dimensional shape like a circle or a square. The pixels within the image are addressed with two-dimensional coordinates (x,y).

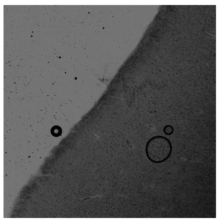
This dilation is applicable to black-and-white masks in order to enlarge the white regions, given that the intensity value of "white" is larger than the one for "black". Each pixel of the original mask which has at least one white pixel within the neighborhood defined by the structuring element b will be colored white in the new mask, too. Only those pixels without any white pixels in their surroundings are colored black. [GW08, p. 666f]

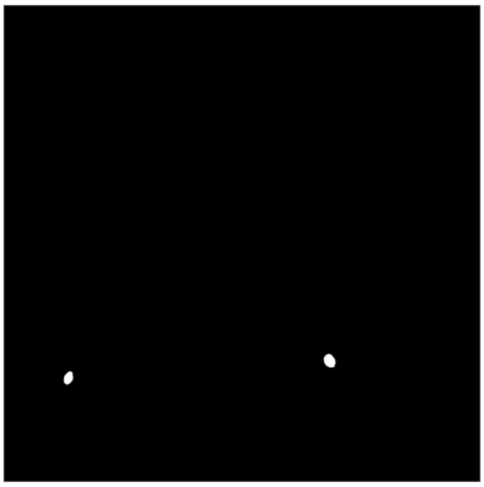
Since the masks created by the given algorithm contain at least the three labels `brain`, `background` and `border`, the dilation operator has to be re-defined for this purpose as it is illustrated in structure chart 4.16. For each pixel in the original mask, it is checked in a first step, which of the labels are present in the neighborhood defined by the structured element. It has been figured out that a circle works best as a structured element. If at least one brain pixel has been detected, the considered pixel is marked as `brain`. Otherwise, it is labeled as `border` if one or more border pixels have been detected. If there are only `background` and

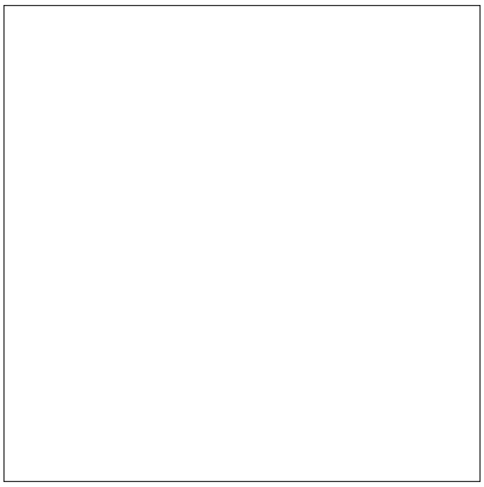

(a) Mask 06-16

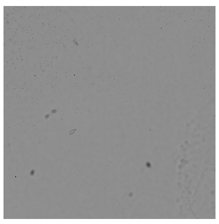

(b) Mask 07-16


(c) Tile 06-16


(d) Tile 07-16


(e) Mask 00-00


(f) Mask 08-14


(g) Tile 00-00

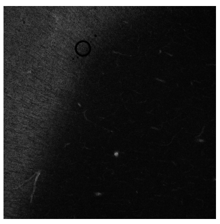
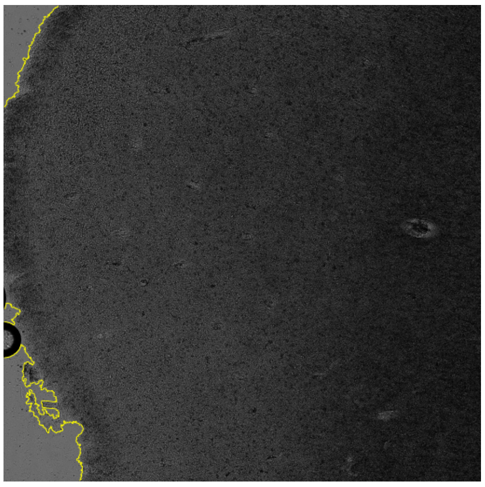
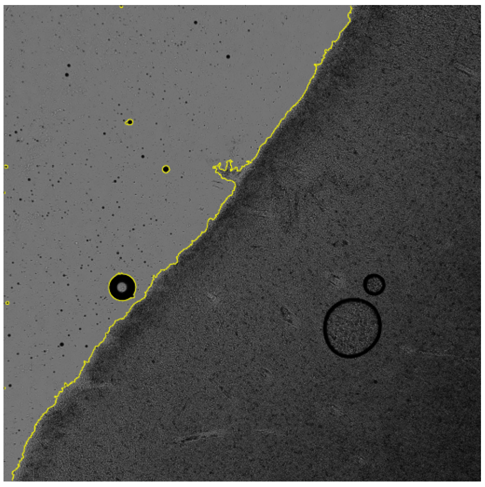
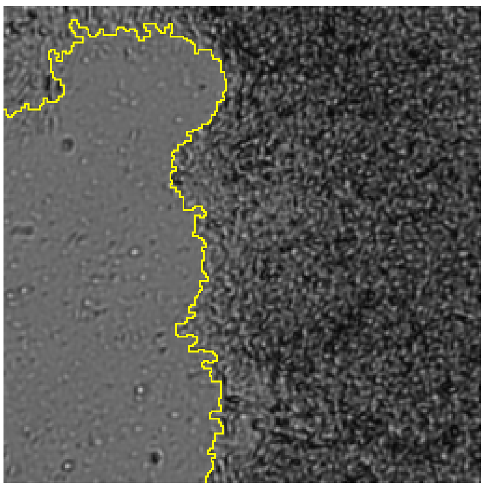

(h) Tile 08-14

Figure 4.14: These masks are the result of the developed seeded region growing. The original tiles are showed as a reference.


(a) Tile 06-16


(b) Tile 07-16


(c) Detail of Tile 06-16

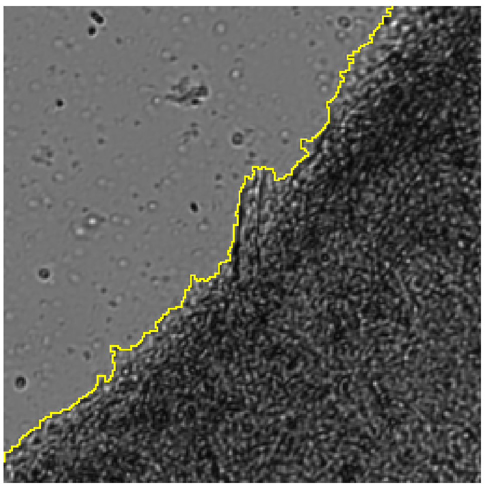

(d) Detail of Tile 07-16

Figure 4.15: The yellow lines illustrate the edges between the white brain tissue and the black background regions in the segmentation masks. They are printed as an overlay covering the original images to allow a better optical evaluation of the segmentation quality.

`undefined` pixels, the current pixel is colored with the `background` label. If all pixels within the neighborhood are masked as `undefined`, the considered pixel gets this label. This can be the case at the edges of the image if a small structured element is used.

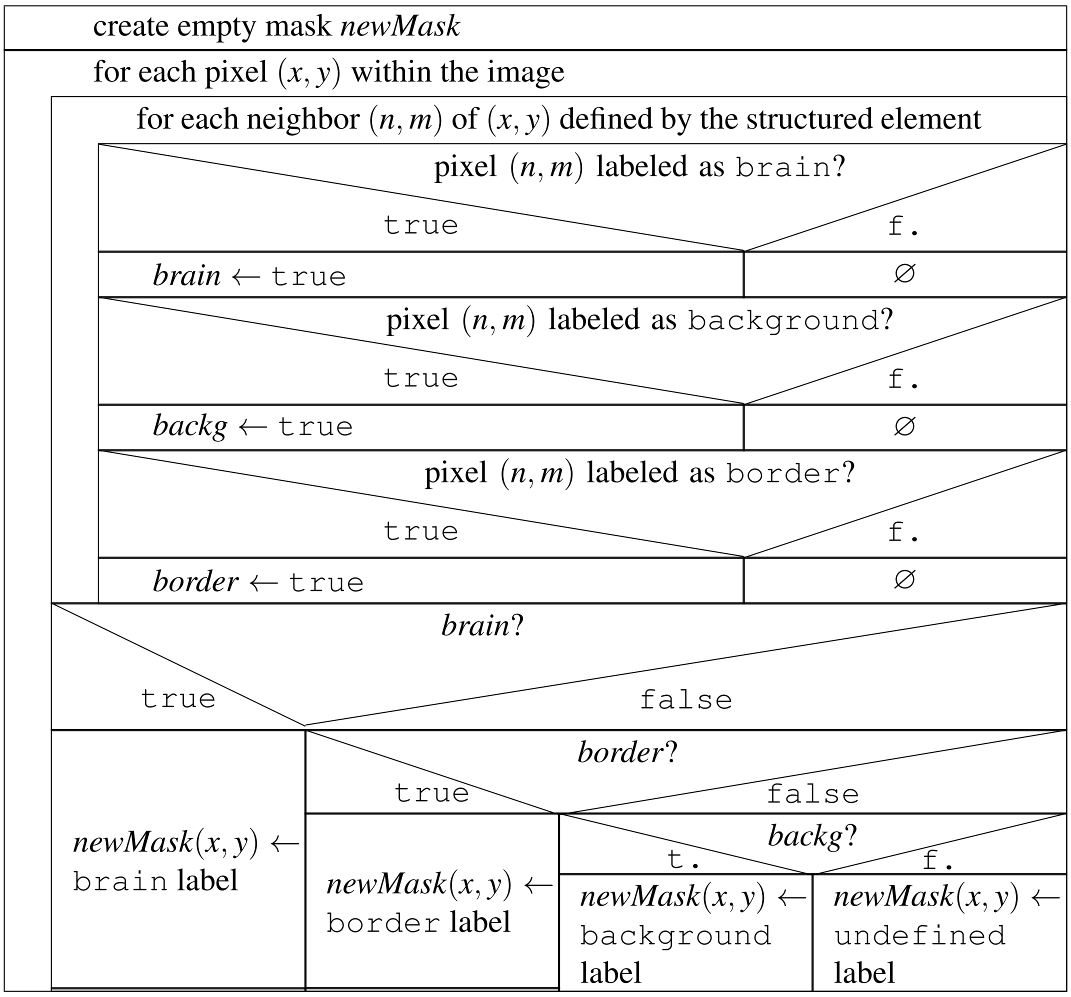
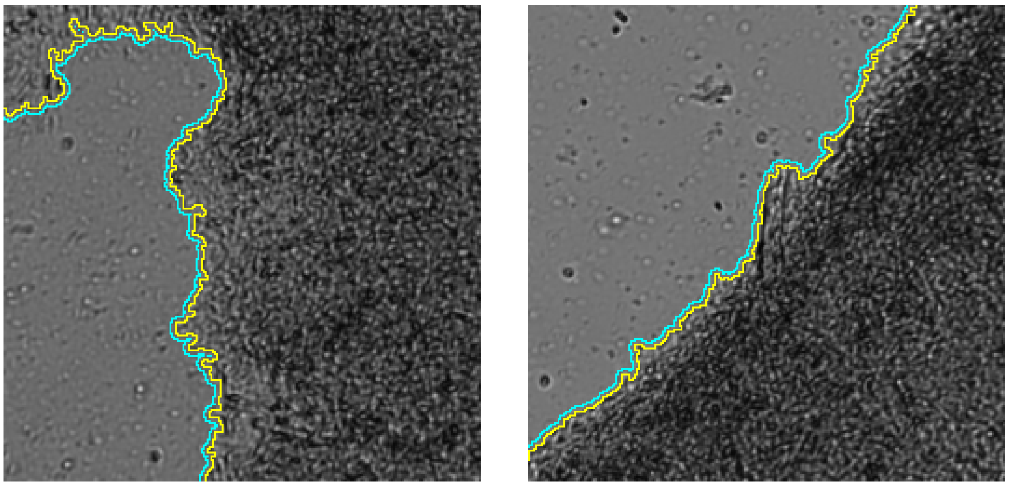

create empty mask *newMask*
for each pixel (x, y) within the image
for each neighbor (n, m) of (x, y) defined by the structured element
pixel (n, m) labeled as `brain`?
`true`
`f.`
brain ← `true`
∅
pixel (n, m) labeled as `background`?
`true`
`f.`
backg ← `true`
∅
pixel (n, m) labeled as `border`?
`true`
`f.`
border ← `true`
∅
brain?
`true`
`false`
$newMask(x, y) \leftarrow$ `brain` label
border?
`true`
`false`
$newMask(x, y) \leftarrow$ `border` label
backg?
`t.`
`f.`
$newMask(x, y) \leftarrow$ `background` label
$newMask(x, y) \leftarrow$ `undefined` label

Figure 4.16: Structure Chart: This dilation has been defined to enlarge the `brain` regions within a mask which also may contain three other labels, namely `background`, `border` and `undefined`.

The effect of the dilation is illustrated in figure 4.17. Here the radius of the structured element has been chosen as three pixels. The newly created, blue colored border ensures that by trend background pixels are marked as brain rather than brain pixels as background. Hence, with the integration of the dilation in the processing pipeline also the first requirement is fulfilled.

Another requirement which has to be checked refers to the amount of manual input or interaction. It has to be independent of the number of images to be processed. The only step requiring user input is the choice of seed points. This process has been successfully automatized so that the user only has to define a threshold



(a) Detail of Tile 06-16 (b) Detail of Tile 07-16

Figure 4.17: The yellow lines show the border between brain tissue and background in the original segmentation mask. The blue lines illustrate the new border created by the dilation operator.

within the joint intensity histogram of all image tiles. The effort of this procedure is completely independent of the number of images so that also this requirement is fulfilled.

The fourth requirement emphasizes that there are image tiles showing both brain tissue and background but also others with only one of these two classes. The segmentation tool has to process both types of images correctly. All parts of the developed algorithm cope with this aspect as it is visible in figure 4.14. The two different measure images are computed for each image tile. If only one of the two classes is visible, one of the measure images does not contain any value smaller than or equal to one. Hence, there are no seeds for the brain tissue or background region. The respectively other region is then grown until it reaches the edges of the image, the resulting mask is single-colored. So this requirement is fulfilled, too.

The last requirement claims the correct handling of artifacts and image noise so that no holes are generated within the brain region. As it is visible in figure 4.14, this requirement is fulfilled. All large air bubbles are marked as brain, whether in brain tissue or background. Only small bubbles located in the background are not labeled as brain which is a result of the smoothing of the measure images.

To conclude, all listed requirements are fulfilled by the developed seeded region growing segmentation. Only to ensure that no brain tissue is cut away, another optional step has to be inserted at the end of the processing pipeline, namely the dilation.

4.5 Optimization of the Runtime Behavior

In the last section, the requirements on the segmentation tool to be developed have been reviewed. It turned out that all of them are fulfilled. In addition to these five main requirements, some further constraints have been listed up in section 2.5. The most important one is that the segmentation should be processable in a reasonable amount of runtime. Hence, the execution time needed to process a section has been measured. It takes about five hours at an average to segment all tiles of a section. This execution time consumption is much too high. Projected to a brain with 1500 sections, it results in 295 days per brain.

Therefore, it has to be analyzed which parts of the algorithm consume the main part of runtime and how these steps can be accelerated. Figure 4.18 illustrates the proportions of runtime required to execute the different main steps of the algorithm. The calculation of the measure images and the choice of final seeds together need 93% of the total execution time. The dilation needs an additional 3.8%. On the contrary, the actual region growing only consumes 3.09% of the total runtime. It has to be noticed, that for these measurements and all following the time needed for input and output operations has not been included due to its dependency on not controllable influences of other processes on the same computer, like a concurrent memory access of different running programs, and on the hardware characteristics.

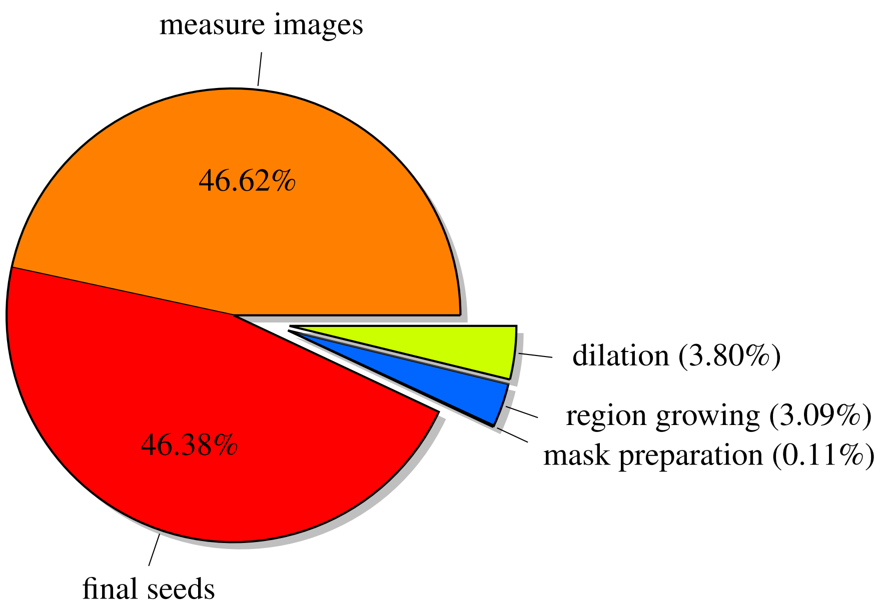


Figure 4.18: This diagram shows the proportions of the different algorithm steps on the total runtime of the tool measured per section. The main part of runtime is consumed by the choice of seeds which is the calculation of the measure images and the choice of final seeds. The actual growing process needs about 3% of the total runtime which is nearly the same amount as for the dilation. The labeling of the seeds in the mask, so the mask preparation, requires only 0.1% of the runtime.

4.5.1 Multi-core Application

To speed up the segmentation, two different levels of parallelization have been established. In a first step, the tool is ported to a multi-core platform. For this purpose, the image tiles of a section are cyclically distributed to all available processes. The same distribution is used for the computation of the joint histogram as well as for the segmentation itself. This parallelization is possible because all tiles can be processed independently of the others. Structure chart 4.19 describes how the distribution of the tiles is done in detail for the calculation of the joint histogram. The parallelization of the region growing is done the same way.

```
for $section \in [s_{start}, s_{end}]$
    for $tile \in [rank, size_x * size_y]$, incremented in steps of $\#proc$
        $x \leftarrow tile\%size_x + offset_x$
        $y \leftarrow tile/size_x + offset_y$
        read image tile at position $(x, y)$ of section $section$
        for each $pixel$ position in $tile$
            $histo\,[\,tile\,[\,pixel\,]\,/\,BIN\_WIDTH\,] += 1$
master process collects all $histo$s
```

Figure 4.19: Structure Chart: Parallelized Computation of the Joint Histogram

The Message Passing Interface (MPI)[1] is used for the inter-process communication. The tiles of a section are traversed with a single loop instead of two nested loops as in the sequential variant in structure chart 4.3 (see page 44). This single loop can be used under the condition that a rectangular, regular grid of image tiles is present. It runs through the numbers from 0 to $size_x * size_y$ which is the number of tiles per section.

Every process handles each $\#proc$-th tile of a section starting at tile number $rank$, the rank of a process within the MPI communicator. Before the image can be read, the actual (x, y) grid coordinates of the tile have to be computed based on the loop index and using the grid dimensions $size_x$ and $size_y$ as well as the grid coordinate offsets $offset_x$ and $offset_y$. Thus, the first tile of the image tile grid has the coordinates $(offset_x, offset_y)$ and is addressed by loop index 0. The last one can be identified as $(offset_x + size_x - 1, offset_y + size_y - 1)$. The calculation of the histogram is performed as formerly described.

This way $\#proc$ distributed parts of the joint histogram are computed, each one covering the whole intensity range but only the information of a disjoint subset of all images. They have to be collected to gain the global joint histogram. A defined master process is responsible to gather the distributed histogram data, to

[1]http://www.mcs.anl.gov/research/projects/mpi/, accessed at 10/06/2013

combine them and to write the global joint histogram to an output file. This final MPI communication is not needed in case of the seeded region growing.

4.5.2 Data Parallelism and GPUs

In addition to the porting to a multi-core platform, another level of parallelism has been introduced into the seeded region growing method. As it is visible in figure 4.18, the steps that consume the major part of runtime are the calculation of the measure images, the choice of final seeds and - with a larger runtime difference - the dilation.

These three operations have in common that they can be computed independently for each pixel, so they can be executed data parallel. Data parallelism in general means "*the simultaneous execution of the same operations across a set of data*" [HQ91, p. 13f].

The only processors which can exploit data parallelism are those who allow the concept of Single Instruction Multiple Data (SIMD) [HQ91, p.14]. These processors compute an instruction on multiple data at the same time. A well-known representative of SIMD processors are Graphics Processing Units (GPUs) which are built in the majority of all offered computers including laptops and workstations. They have thousands of parallel processing units and are originally designed to execute the same instruction on thousands of pixels in the same processor cycle [NVI10].

There are several frameworks available to perform General Purpose Computation on GPUs (GPGPU), so to directly use a GPU for computations, like OpenCL[2] and CUDA[3]. It has been decided to use CUDA version 4.0 due to its comparably simple syntax and easy usage in order to compute the three steps on a GPU that consume the major part of runtime.

CUDA Kernels and Threads

To redefine the three steps of the seeded region growing using CUDA, it is important to understand its concept of kernels and threads first. A portion of code to be executed on a GPU has to be defined as a so-called kernel which is a specially marked method. Each kernel is executed on the GPU by up to thousands of threads. A program using GPGPU has to be started on a CPU, the so-called "host", which then invokes the kernel on the GPU, the "device".

The kernel is executed by an array of threads that has to be defined manually. A grid of thread blocks is launched for the execution of a kernel, whereas a thread block itself is an array of threads (see figure 4.20). The threads in the same block have shared memory and are able to synchronize. Different thread blocks cannot cooperate but all can use the same global memory.

[2]http://www.khronos.org/opencl/, accessed at 10/06/2013

[3]http://www.nvidia.de/object/what_is_cuda_new_de.html, accessed at 10/06/2013

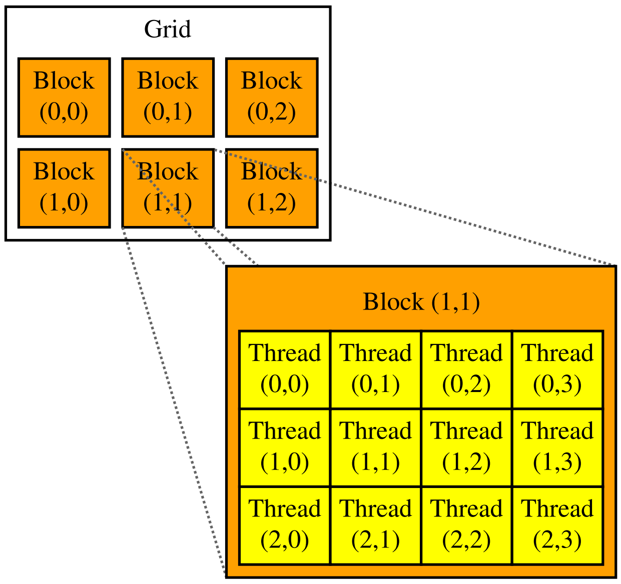


Figure 4.20: The threads running on a GPU are organized in blocks of threads which are arranged as a grid of blocks. In this example, there are twelve threads per block and six thread blocks, so 72 threads in total.

Each thread block is executed on one of the multiprocessors of the GPU. A multiprocessor can execute a defined number of threads respectively thread blocks concurrently. If there are more blocks than units on the multiprocessors, it is up to the GPU to schedule the blocks, i.e. to decide which blocks are directly executed on which processor and which have to wait. Therefore, it is important that the work package of a thread block has to be independent of all others, also by means of the execution order.

Host and device have separate memories. Two steps have to be done before a kernel can be executed on the device. First, the required memory is allocated in the global memory of the GPU. Afterwards, the data is copied from host to device. Then the kernel can be executed. To get the results, they have to be copied from device to host after the execution and the device memory is freed. Since these additional copy operations consume a significant amount of time, the number of instructions to be handled by each thread has to be large enough. Otherwise the runtime gain by the use of the GPU is compensated by the copy operations. [Rue08, NVI08]

Implementation

To describe how the data parallel implementation of the measure images, the choice of final seeds and of the dilation is done using CUDA, it is sufficient to take a glance at one of the three methods since the principle is always the same. The following listing 4.1 contains the CUDA kernel used for the dilation. It is marked as a kernel executed on the device and invoked by the host using the keyword `__global__`.

```
1  __global__ void dilation(mask_pixel* newMask, mask_pixel
     * mask, unsigned int width, unsigned int height) {
2    int i = blockDim.x * blockIdx.x + threadIdx.x;
3    int x = i / width;   //2D-coordinates (row)
4    int y = i % width;   //2D-coordinates (column)
5    int a, b;  //indices to traverse the struct. elem.
6    unsigned int pixel;  //actual pixel to be considered
7    int count[4] = {0, 0, 0, 0};
8    int min = (BACKG<BRAIN) ? BACKG : BRAIN;
9    min     = (BORDER<min) ? BORDER : min;
10   min     = (UNDEF<min) ? UNDEF : min;
11   int offset = -1 * radius;
12   int squareRadius = radius * radius;
13
14   if ( i < width*height ) {
15     pixel = x * width + y;
16     if( x >= radius && y >= radius &&
17         y < width-radius && x < height-radius ) {
18       for(a = offset; a <= offset*-1; ++a) {
19         for(b = offset; b <= offset*-1; ++b) {
20           //structured element: circle
21           if((a*a+b*b)>squareRadius)
22             continue;
23           (count[mask[(x+a) * width + (y+b)] - min])++;
24         }
25       }
26       if(count[BRAIN - min]) {
27         newMask[pixel] = BRAIN;
28       } else {
29         if(count[BORDER - min]) {
30           newMask[pixel] = BORDER;
31         } else {
32           if(count[BACKG - min]) {
33             newMask[pixel] = BACKG;
34           } else {
35             newMask[pixel] = UNDEF;
36           }
37         }
38       }
39     }
40   }
41 }
```

Listing 4.1: CUDA Kernel of the Dilation

In lines two to four, the portion of data to be handled by the invoking thread is determined which is in this case a single pixel of the mask. The three variables `blockDim`, `blockIdx` and `threadIdx` are defined by CUDA. The value of `blockDim.x` is the dimension of the grid of blocks in x-direction, `blockIdx.x` is the index of the actual thread block within this direction. With `threadIdx.x`, the index of the thread within its thread block can be found out. Using this formula, a global index within the threads of all thread blocks is calculated. This index is then used to address the work package of each pixel. Therefore, the thread index is interpreted as a one-dimensional pixel coordinate in $[0, width * height]$. In lines three and four, the two-dimensional pixel coordinates are computed out of the one-dimensional one.

In lines eight to ten, it is searched for the minimum integer value of the four possible mask entries `BACKG`, `BRAIN`, `BORDER` and `UNDEF`. The if condition in line fourteen checks whether there is work to do for the calling thread or not. It is needed in case that the user-defined grid of threads does not fit the dimensions of the mask image so that there are some idle threads without work packages because the thread grid has to be larger or equally dimensioned than the "grid" of work. From line 18 to 25, the labels in the structured element of the actual pixel are traversed as described in the last section so that from line 26 to 38, the considered pixel can be masked in a new mask image.

This kernel is called by the instructions in listing 4.2. It has been decided to use a one-dimensional alignment of threads within a block and also a one-dimensional grid of blocks because it matches the one-dimensional alignment of the pixels in memory. The number of threads per block can be defined by the user since it should be adapted to the dimensions of the images so that it does not result in too many idle threads. It is stored in the variable `BLOCK_SIZE`. In line two, the dimensions of each thread block are defined. The number of blocks required to cover the whole image with threads is computed and stored in the grid dimensions `gridDim` (see line three). In case that the number of threads within a block is not a divider of the image size in pixels, an additional and partly idle block has to be added to the grid. The kernel is called in line four. The dimensions of thread blocks and grid are passed to CUDA with `<<<gridDim, blockDim>>>`.

```
1 unsigned int size = width*height;
2 dim3 blockDim(BLOCK_SIZE);
3 dim3 gridDim((size \% BLOCK_SIZE) ? size/blockDim.x : size/
      blockDim.x+1);
4 dilation<<<gridDim, blockDim>>>(newMask, mask, width, height);
```

Listing 4.2: Invocation of the Dilation Kernel

The two other steps, so the calculation of measure images and the choice of final seeds, are CUDA-parallelized the same way. To sum this process up, the loop traversing all pixels of an image is replaced by the CUDA instructions so that there is a one-to-one assignment between threads and pixels. The rest of the methods is unchanged.

4.5.3 Evaluation of the Parallelization

Juelich Dedicated GPU Environment (JUDGE)

To combine both levels of parallelism, i.e. the multi-core implementation and the data parallel execution of some steps on a GPU, the supercomputer JUDGE has been used, which is one of the supercomputers hosted by the JSC. It consists of 206 compute nodes, each with two Intel Xeon X5650 (Westmere) six-core processors and two NVIDIA Tesla GPUs. There are nodes available with either NVIDIA Tesla M2050 (Fermi) or NVIDIA Tesla M2070 (Fermi) GPUs. These two only differ regarding the available memory. The memory of both GPU types is with 3GB respectively 6GB sufficient for the present application.

All twelve cores of a node have access to both GPUs. It has to be defined how many processes and GPUs per node are used during the job submission. The GPUs are exclusively assigned to a job. Hence, all processes of the same job on a node can use the requested GPUs but not other jobs at the same time since the nodes are not provided exclusively per job. The implementation of the seeded region growing described in this chapter has been done in C++ (for further configurations see appendix A). [JSC13]

Runtime Measurements: Multi-core Implementation

The effect of the porting to a multi-core platform can be seen in figures 4.21 for the histogram and 4.22 for the region growing in form of speedup curves. To gain these speedup measures, the applications have been executed using between one and 64 processes on JUDGE for the same amount of image tiles, so the strong scaling is examined. To make these results comparable to later ones, only two cores per node are used.

In case of the histogram, the speedup curve is linear for small numbers of processes but flattens out as their number grows. This can be explained by the raising influence of MPI communication needed to collect and combine the distributed histograms. It can be seen that the communication effort significantly grows from 16 to 32 processes.

The seeded region growing does not include any MPI communication so that a linear speedup is reached given that the number of image tiles per section can be equally distributed to the processes. An equal distribution in this case means not only that each CPU processes the same number of tiles but also that the effort needed to segment the tiles is comparable. For tiles showing only brain tissue or background but not both, the majority of pixels is already labeled during the mask preparation happening after the choice of final seeds as it is visible in figures 4.8(d) (see page 49) and 4.9(c) (see page 50). Thus, the region growing is finished in a shorter time than for tiles showing both classes.

Due to this reason, the image tiles are rearranged before the runtime measurements have been done to gain the speedup plots. This way it is guaranteed that a load balance is reached and thus the optimal scaling behavior is measured not influenced by effects observable because of the random assignment of image tiles to processes.

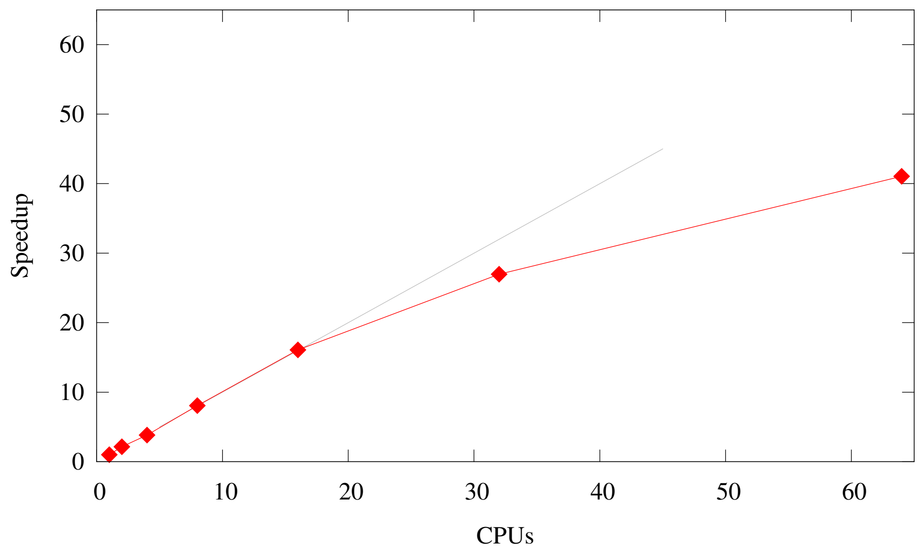


Figure 4.21: The graph shows the speedup per section of the **histogram** calculation reached by porting it to a multi-core computer. The flattening out of this curve can be explained by the growing effort for MPI communication.

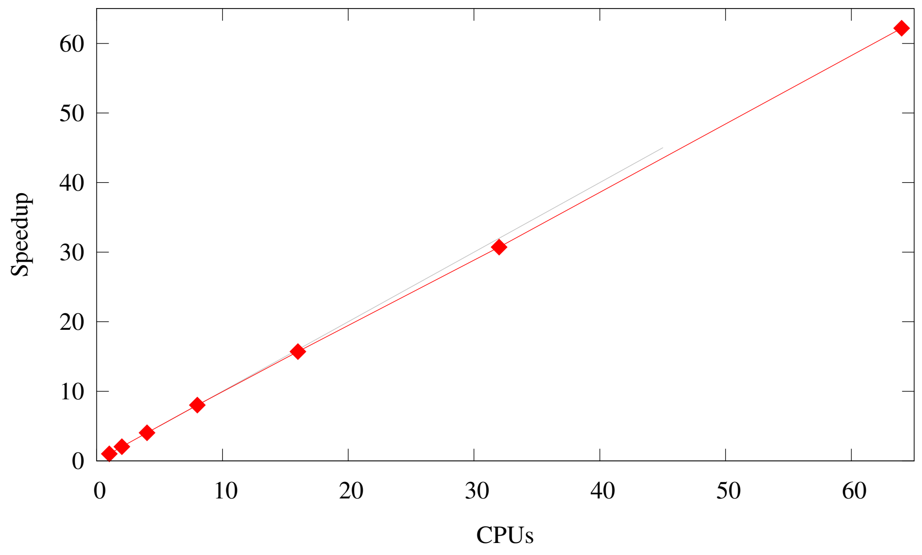


Figure 4.22: The **seeded region growing** does not include any MPI communication so that a linear speedup is reached given that the tiles are equally distributed to the processes.

Runtime Measurements: GPU Parallelization

It is to be expected that the speedup curve gained by the seeded region growing is the same if the calculation of measure images, the choice of final seeds and the dilation are calculated on a GPU. This presumption is assumed due to the fact that no inter-process communication is needed in these steps by porting them to CUDA but only a data exchange between process and GPU. The time needed to copy data to and from the graphics unit is always the same, independent of the content of the image tiles.

To run the seeded region growing on JUDGE, the communication structure illustrated in figure 4.23 is used. Two of the twelve cores and both GPUs are requested on each node which is the reason why in the former speedup measurements also only two cores per node have been used. All cores can directly communicate via MPI as described in the former section even if it is not needed for this tool. They address the GPUs in a one-to-one assignment. Thus, the image tiles of a section are distributed among all requested processes. Each process then sources the three steps out to "its" GPU.

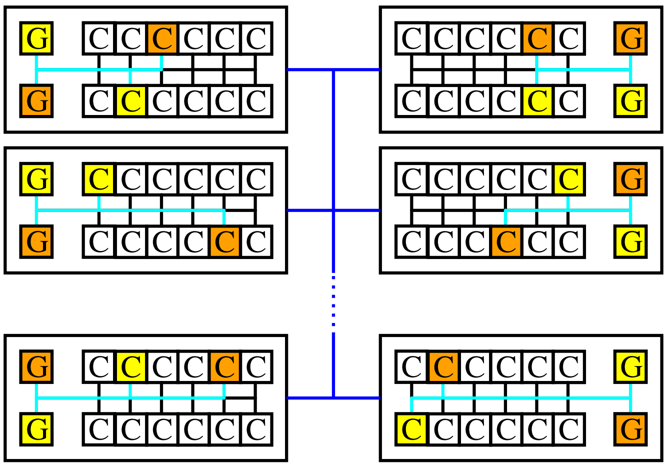


Figure 4.23: On each requested node, two of the twelve cores and both GPUs are used. All cores can directly communicate with each other using MPI. Every allocated core accesses one of the two GPUs so that a one-to-one assignment is reached which is illustrated by the yellow and orange marked compute units and the light blue lines.

To measure only the effect of the GPU parallelization but not of the multi-core approach, the seeded region growing is executed using one process and one GPU. In a first step, the optimal thread configuration has to be figured out. The number of threads per thread block is varied to gain the minimal runtime. The results are illustrated in figures 4.24 and 4.25. It shows the measured runtimes of the three GPU parallelized steps for sizes of thread blocks from 128 to 768 in steps of 64 threads as well as the added runtime of these steps. It is visible that mainly the calculation of the measure images can be accelerated by the optimal parametrization of the thread block size. The other steps reach approximately the same runtime in all configurations. One of the shortest added runtimes is reached at a thread block size of 256.

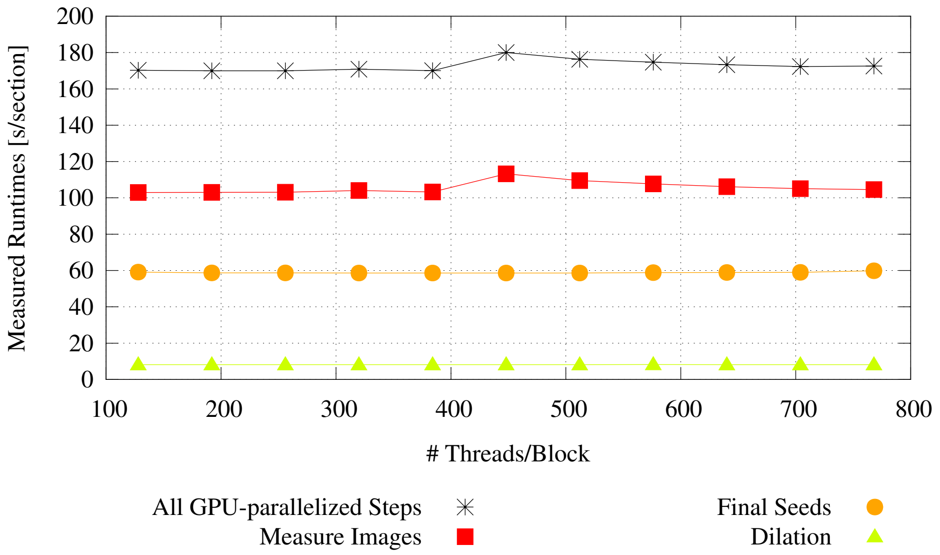


Figure 4.24: These are the runtimes of the GPU-parallelized steps using different sizes of the thread blocks. The major effect is observable for the calculation of the measure images.

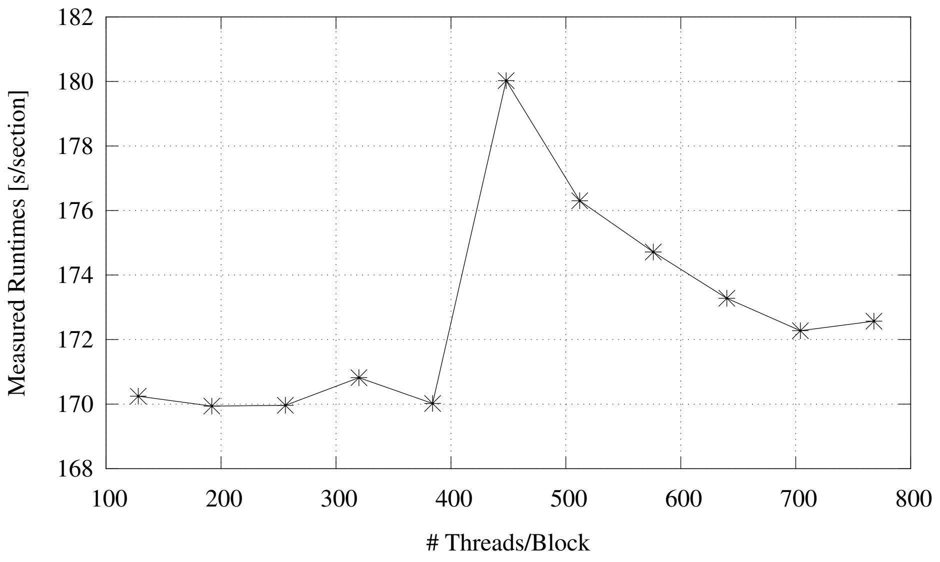
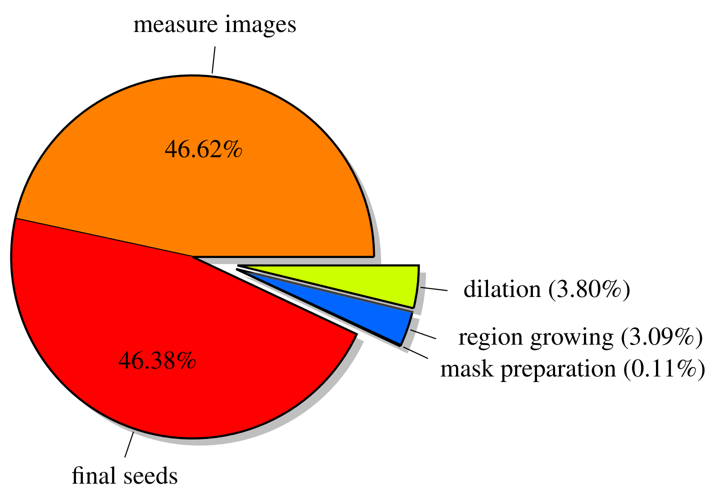
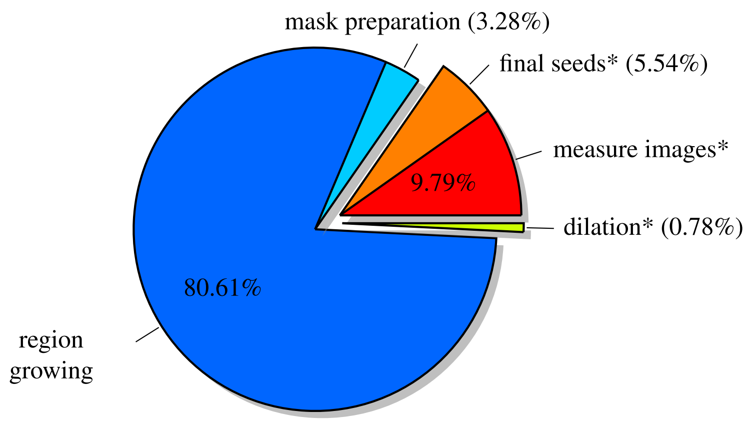


Figure 4.25: This curve shows again the summed up runtime of all three steps which are moved to the GPU to figure out the best configuration of the thread block size. The shortest runtimes are reached with block sizes of 192, 256 and 384 threads.

Using this configuration, the proportions of the different algorithm steps on the total runtime can be compared to the ones gained in the sequential implementation. Figure 4.26 depicts the differences between these proportions executing the seeded region growing with and without the usage of a GPU.



(a) All steps are computed on the CPU.



(b) The methods marked with an astersik (*) are computed on a GPU.

Figure 4.26: The added proportion of runtime of the three steps “measure images”, “final seeds” and “dilation” is reduced from 96.8% to 16.11% if they are computed on a GPU instead of a CPU.

While in the CPU-only variant the two seed steps marked in red and orange consume 93% of the added runtime, their proportion is reduced to 15.33% in the GPU implementation. The proportion of runtime for the dilation (green) is reduced from 3.8% to 0.78%. The execution time of the three steps is reduced from about 4.58h to 140s which is a factor of about 120 by the use of a GPU. Due to the acceleration of these steps, the proportion of the region growing (blue) on the total execution time grows from 3.09% to 80.61% whereby the runtime of this step stays the same because it has not been modified. This means that the total runtime of the seeded region growing decreases if a GPU is used.

Using the CUDA implementations of the three steps instead of the CPU variants, the runtime is mainly influenced by the implementation of the seeded region growing because it consumes the major part of runtime. To compare the scaling behavior of both variants, the tool has been executed again on the rearranged tiles with numbers of processes between one and 64 and the CUDA variants of the three methods. These times are then compared to the ones measured for the CPU-only speedup curve in figure 4.22. The result is illustrated in figure 4.27. It is visible that the use of GPUs does not influence the scaling behavior of the seeded region growing since the curves have the same course. The runtime curve of the CPU-only variant is the same as the GPU curve with an upwards shift.

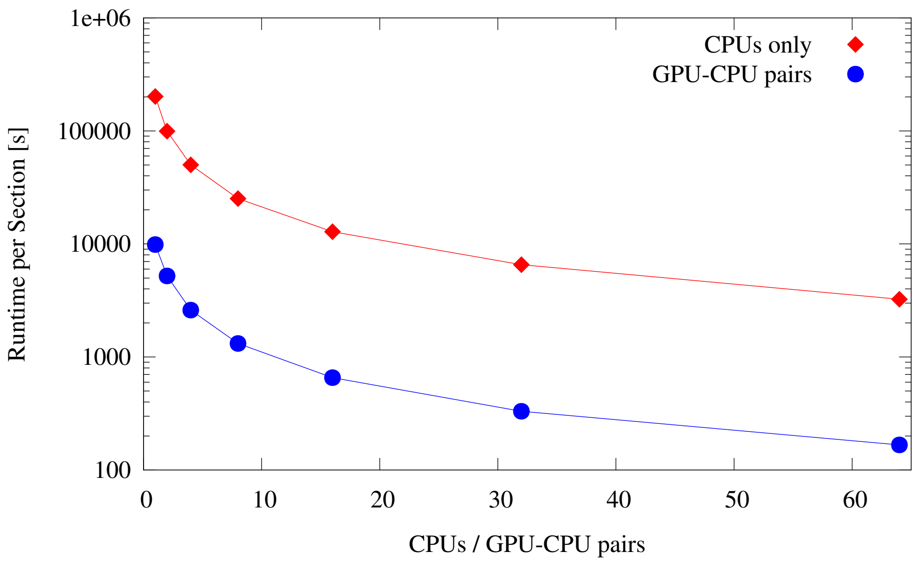


Figure 4.27: The runtime behavior of the seeded region growing using GPUs in addition to the CPUs is the same as for the variant using only CPUs but with a downwards shift.

In the beginning of this section, it has been assumed that the porting of some steps to CUDA does not affect the speedup curve. Figure 4.28 proves this presumption. The speedup of the seeded region growing is linear in the multi-core variant as well as in the multi-core variant with the additional use of GPUs. Nevertheless, it has to be given that an equal balancing of work between the processes can be reached to gain a linear speedup.

To sum up, the calculation of the joint histogram can be implemented as a multi-core application showing a reasonable scaling behavior. In case of the seeded region growing, even a linear speedup is reached because no inter-process communication is needed, given that the image tiles and the effort to process them are equally distributed between the processes. An additional porting of the data parallel steps

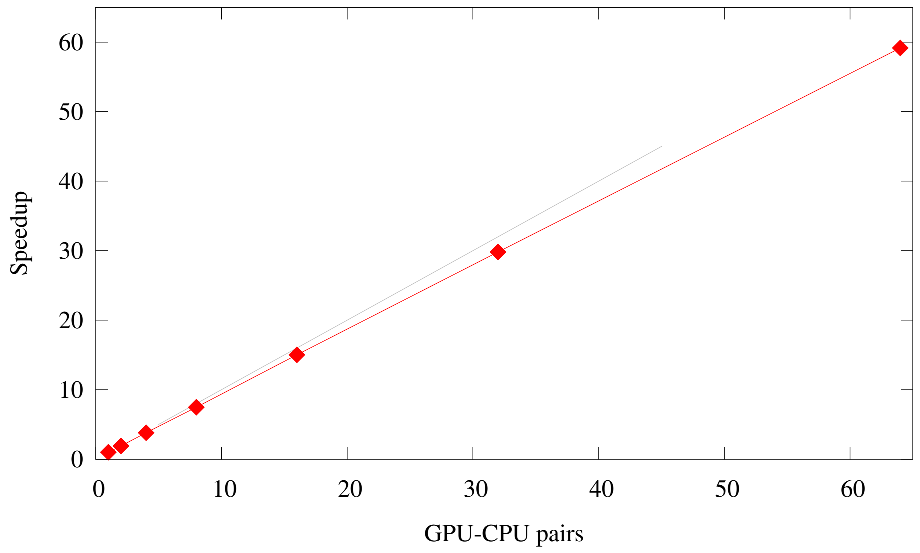


Figure 4.28: A linear speedup is (still) reached even if both levels of parallelism are combined. So the image tiles are distributed among the available processes. Each CPU then sources three steps out to a dedicated GPU.

to the GPU using CUDA reduces the runtime of these steps by a factor of about 120. The linear speedup is still reached in this case. All in all, the tool provides a well scaling, GPU-accelerated segmentation fulfilling all given requirements and constraints.

5 Conclusion and Outlook

This thesis deals with the development and implementation of a segmentation tool for high-resolution human brain images acquired with the technique of Polarized Light Imaging. The segmentation has the aim to create masks differentiating between brain tissue and background in the original images. This step is important as additional knowledge for other steps in the processing pipeline of the PLI data.

To understand the characteristics of these brain images, the technique of PLI has been briefly introduced in a first step. With that in mind, a list of requirements has been worked out which have to be fulfilled by the segmentation tool to be developed. These are, to give some examples, the need for reasonable edges between brain and background which should tend more to extend into the background than into the brain region, and a minimized required amount of manual input independent of the number of images to be processed.

Based on these requirements, the most appropriate segmentation approach has been chosen out of the wide range of existing ones. Therefore a selection of tools has been applied to four example images. It has become clear that a seeded region growing application based on [AB94] achieved the most reasonable masks and fulfills the requirements best among all compared tools.

Nevertheless, this algorithm had to be adopted to the given needs. Above all, the amount of manual effort for the choice of seed points had to be reduced from a dependency to the number of images to a fixed quantity. This has been reached using the joint intensity histogram of all brain images. With this statistic it is possible to select seeds for the brain and background regions in a way that the user has to define a threshold within the histogram as the only manual input. With an additional choice of final seeds out of these seed candidates, reasonable start points for the region growing can be achieved.

In addition to the choice of seeds, also other aspects of the seeded region growing have been modified in comparison to the paper mentioned above, namely the similarity criterion and the growing operator. This results in a processing pipeline which consists of two separate tools. The first one computes the joint histogram of a defined set of images. Afterwards, the threshold is defined manually. This input, i.e. the histogram and the threshold, are reusable for multiple executions of the second tool, the actual seeded region growing.

After the seeded region growing has been developed and implemented, it has to be checked whether the formerly defined requirements are fulfilled. It turns out that the newly developed segmentation achieves results of the same or an even better quality than the previously tested ones. To fulfill also the requirement that no edge extends into the brain region, an additional and optional dilation has been added to the processing pipeline. With this extra step, all five needs are satisfied.

Besides the requirements, some further constrains had been defined. The most important one concerns the runtime behavior of the segmentation. Without further improvements, the processing time for a section would have been too long. Therefore, a two-level parallelization has been applied to the seeded region growing. On the first level, the tool is ported to a multi-core computer, so the image tiles of a section are distributed among the available processes. The region growing achieves a linear speedup with this parallelization if all processes have the same amount of work because no inter-process communication is needed.

The second level of parallelism considers the three steps which consume the major proportion of runtime, namely the calculation of measure images, so the selection of seed candidates, the choice of final seeds and the dilation. Since these three steps can be executed data parallel, they are re-implemented using CUDA. This way they are now executed on a GPU instead of the CPU. This approach leads to an acceleration of these steps by a factor of 120. The linear scaling behavior of the seeded region growing is not influenced by this second level of parallelism.

In future, some aspects can be improved respectively added to the tool. First of all, it is envisaged to provide the option of marking artifacts with an own label given that they can be identified by their intensity. Especially the black colored air bubbles can be characterized by their intensity and shapes. It is also possible to use another two labels, depending on the region the artifact would normally be added to by the region growing. This additional information can be used as a hint for the registration of the sections among each other since the air bubbles are unique for each section.

Another improvement deals with the distribution of image tiles in the multi-core variant of the region growing. Even if all processes have the same number of tiles to be segmented, the required runtime does not need to be the same. Tiles showing only brain tissue or background but not both are segmented much faster than mixed images because the final seeds already cover nearly the whole mask. If some processes get more of these single-colored tiles than others, they are finished earlier and thus idle for the rest of the execution. Therefore, it can be tested to add a load balancing which dynamically distributes the tiles in order to minimize the idle times. However, it has to be kept in mind that the additional communication effort for the load balancing may compensate or even exceed the runtime gain.

The third aspect refers to the future use of the segmentation. If an entire brain has been segmented by the developed tool, stitched and registered, it may be wished to segment the images once more with a higher precision what is possible due to the now three-dimensional information. In this case, the masks created by the developed seeded region growing can be used as a start constellation, e.g. for a level set segmentation which needs an initial mask. For this purpose, the operations applied to the original images always have to be applied to the masks, too.

To sum up, the developed seeded region growing fulfills all requirements induced by the given image data. The resulting masks contain reasonable edges and are processed in an acceptable amount of runtime due to the two ways of parallelization that can be used in combination or separately - depending on the number of images to be segmented and the available hardware. The tool can be used semi-automatic with a fixed amount of manual input independent of the number of image tiles to be processed. There are some ways to improve the results of the developed segmentation and further future use cases. However, the improvements are not mandatory since the developed seeded region growing segmentation already produces good results.

Bibliography

[AAG$^+$11] M. Axer, K. Amunts, D. Gräßel, C. Palm, J. Dammers, H. Axer, U. Pietrzyk, and K. Zilles. A novel approach to the human connectome: ultra-high resolution mapping of fiber tracts in the brain. *NeuroImage*, 54:1091 – 1101, 2011.

[AB94] R. Adams and L. Bischof. Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994.

[Abd11] M. M. Abdelsamea. An Automatic Seeded Region Growing for 2D Biomedical Image Segmentation. In *International Conference of Extrusion and Benchmarking*, volume 21. IACSIT Press, 2011.

[AGK$^+$11] Markus Axer, David Gräßel, Melanie Kleiner, Jürgen Dammers, Timo Dickscheid, Julia Reckfort, Tim Hütz, Björn Eiben, Uwe Pietrzyk, Karl Zilles, and Katrin Amunts. High-resolution fiber tract reconstruction in the human brain by means of three-dimensional polarized light imaging (3D-PLI). *Frontiers in Neuroinformatics*, 5(34), 2011.

[Bra13] Brain Tumor Center Piedmont. Fiber Tract Mapping. `http://www.piedmontbraintumorcenter.org/BrainTumor/FiberTractMapping.aspx`, 2013. Accessed: 05/04/2013.

[FFJ05] Yue Feng, Hui Fang, and Jianmin Jiang. Region Growing with Automatic Seeding for Semantic Video Object Segmentation. In *Proceedings of the Third international conference on Pattern Recognition and Image Analysis - Volume Part II*, ICAPR'05, pages 542–549, Berlin, Heidelberg, 2005. Springer-Verlag.

[FYE$^+$01] Jianping Fan, David K. Y. Yau, Ahmed K. Elmagarmid, Senior Member, and Walid G. Aref. Automatic Image Segmentation by Integrating Color-Edge Extraction And Seeded Region Growing. *IEEE Transactions On Image Processing*, 10:1454–1466, 2001.

[FZBH05] Jianping Fan, Guihua Zeng, Mathurin Body, and Mohand-Said Hacid. Seeded region growing: an extensive and comparative study. *Pattern Recogn. Lett.*, 26(8):1139–1156, June 2005.

[GW08] R.C. González and R.E. Woods. *Digital Image Processing*. Pearson Education. Pearson/Prentice Hall, 2008.

[HQ91] Philip J. Hatcher and Michael J. Quinn. *Data-parallel programming on MIMD computers.* Scientific and engineering computation. MIT Press, 1991.

[ISNC05] Luis Ibáñez, Will Schröder, Lydia Ng, and Josh Cates. The ITK Software Guide. http://www.itk.org/ItkSoftwareGuide.pdf, November 2005. Accessed: 09/04/2013.

[ITK11] ITK-SNAP Team. ITK-SNAP 2.4. http://www.itksnap.org/pmwiki/pmwiki.php?n=Main.HomePage, 2011. Accessed: 11/04/2013.

[JSC13] JSC Webpage. Juelich Dedicated GPU Environment (JuDGE). http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUDGE/JUDGE_node.html, 2013. Accessed: 03/06/2013.

[KJ11] Karl-Heinz Kunzelmann and Sasha Jarek. Seeded Region Growing (ImageJ Plugin). http://www.dent.med.uni-muenchen.de/~kkunzelm/exponent-0.96.3/index.php?section=71, 2011. Accessed: 12/04/2013.

[Mia12] Mia Solution. MiaLite – ultra-fast level set based segmentation in 3D. http://www.mia-solution.com/index.html, 2012. Accessed: 10/04/2013.

[NVI08] NVIDIA Corporation. CUDA Programming Model Overview. http://www.sdsc.edu/us/training/assets/docs/NVIDIA-02-BasicsOfCUDA.pdf, 2008. Accessed: 03/06/2013.

[NVI10] NVIDIA Corporation. Was ist GPU Computing? http://www.nvidia.de/object/gpu-computing-de.html, 2010. Accessed: 03/06/2013.

[PAG$^+$10] C. Palm, M. Axer, D. Gräßel, J. Dammers, J. Lindemeyer, K. Zilles, U. Pietrzyk, and K. Amunts. Towards ultra-high resolution fibre tract mapping of the human brain - registration of polarised light images and reorientation of fibre vectors. *Frontiers in human neuroscience*, 4:1–16, 2010.

[Rue08] Ruetsch, Greg and Oster,Brent. Getting Started with CUDA. http://www.nvidia.com/content/cudazone/download/Getting_Started_w_CUDA_Training_NVISION08.pdf, 2008. Accessed: 03/06/2013.

[SACF$^+$12] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9:676–682, 2012.

[Set99] J.A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1999.

[SHB07] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007.

[SSKH11] Christoph Sommer, Christoph Straehle, Ullrich Koethe, and Fred A. Hamprecht. ilastik: Interactive Learning and Segmentation Toolkit. In *8th IEEE International Symposium on Biomedical Imaging (ISBI 2011)*, 2011.

[The13] The GIMP Team. GNU Image Manipulation Program 2.6. `http://www.gimp.org/`, 2001-2013. Accessed: 11/04/2013.

[TM04] Paul A. Tipler and Gene Mosca. *Physik: für Wissenschaftler und Ingenieure*. Elsevier, Spektrum Akademischer Verlag, München, 2nd edition, 2004.

A JUDGE Configuration

The following configuration of JUDGE[1] has been used:

Hardware

206 Compute Nodes	
Per Node:	2 Intel Xeon X5650 (Westmere) 6-core processors
	2 NVIDIA Tesla GPUs (M2050 or M2070)
	96GB Main Memory
GPU M2050 (Fermi)	1.15GHz, 448 cores, 3GB memory, used on 54 nodes
GPU M2070 (Fermi)	1.15GHz, 448 cores, 6GB memory, used on 152 nodes
Inter-Node Communication	Infiniband

Operating System and Software

Operating System	SUSE Linux Enterprise Server 11 (x86_64)
CUDA	Version 4.0
Graphics Driver	NVIDIA driver version 304.54
MPI Implementation	MPICH2 version 1.2.1p1
C++ Compilers	gcc version 4.3.4
	mpicxx for MPICH2 version 1.2.1p1
CMAKE	Version 2.8.5
MAKE	GNU Make 3.81

[1]http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUDGE/JUDGE_node.html, accessed at 18/06/2013

Jül-4365
September 2013
ISSN 0944-2952