Computational tools for the segmentation and registration of confocal brain images of *Drosophila melanogaster*

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

vorgelegt von

Benjamin Schmid

Würzburg 2010

Eingereicht am: . . . . . .

Mitglieder der Promotionskommission:
Vorsitzender: . . . . . .
Gutachter: . . . . . .
Gutachter: . . . . . .

Tag des Promotionskolloquiums: . . . . . .

Doktorurkunde ausgehändigt am: . . . . . .

Computational tools for the segmentation and registration of confocal brain images of *Drosophila melanogaster*

Benjamin Schmid
bene.schmid@gmail.com
Lehrstuhl für Genetik und Neurobiologie, Biozentrum,
Universität Würzburg,
Am Hubland, 97074 Würzburg, Germany

Abstract

Neuroanatomical data in fly brain research are mostly available as spatial gene expression patterns of genetically distinct fly strains. The *Drosophila* standard brain, which was developed in the past to provide a reference coordinate system, can be used to integrate these data. Working with the standard brain requires advanced image processing methods, including visualisation, segmentation and registration. The previously published VIB Protocol addressed the problem of image registration. Unfortunately, its usage was severely limited by the necessity of manually labelling a predefined set of neuropils in the brain images at hand.

In this work I present novel tools to facilitate the work with the *Drosophila* standard brain. These tools are integrated in a well-known open-source image processing framework which can potentially serve as a common platform for image analysis in the neuroanatomical research community: ImageJ. In particular, a hardware-accelerated 3D visualisation framework was developed for ImageJ which extends its limited 3D visualisation capabilities. It is used for the development of a novel semi-automatic segmentation method, which implements automatic surface growing based on user-provided seed points. Template surfaces, incorporated with a modified variant of an active surface model, complement the segmentation. An automatic non-rigid warping algorithm is applied, based on point correspondences established through the extracted surfaces. Finally, I show how the individual steps can be fully automated, and demonstrate its application for the successful registration of fly brain images.

The new tools are freely available as ImageJ plugins. I compare the results obtained by the introduced methods with the output of the VIB Protocol and conclude that our methods reduce the required effort five to ten fold. Furthermore, reproducibility and accuracy are enhanced using the proposed tools.

Zusammenfassung

Expressionsmuster genetisch manipulierter Fliegenstämme machen den Großteil neuroanatomischer Daten aus, wie sie in der Gehirnforschung der Taufliege *Drosophila melanogaster* entstehen. Das *Drosophila* Standardgehirn wurde u.a. entwickelt, um die Integration dieser Daten in ein einheitliches Referenz-Koordinatensystem zu ermöglichen. Die Arbeit mit dem Standardgehirn erfordert hochentwickelte Bildverarbeitungsmethoden, u.a. zur 3D Visualisierung, Segmentierung und Registrierung. Das bereits publizierte "VIB Protocol" stellte bisher eine Möglichkeit für die Registrierung zur Verfügung, die aber duch die Notwendigkeit manueller Segmentierung bestimmter Neuropile nur eingeschrnkt verwendbar war.

In der vorliegenden Arbeit stelle ich neue Werkzeuge vor, die den Umgang mit dem Standardgehirn erleichtern. Sie sind in ein bekanntes, offenes Bildverarbeitungsprogramm integriert, das potentiell als Standardsoftware in der neuroanatomischen Forschung dienen kann: ImageJ. Im Zuge dieser Arbeit wurde eine hardwarebeschleunigte 3D Visualisierungs-Bibliothek entwickelt, die die Visualisierungsmöglichkeiten von ImageJ ergänzt. Auf Basis dieser Entwicklung wurde anschließend ein neuer halbautomatischer Segmentierungs-Algorithmus erstellt. In diesem Algorithmus werden Neuropil-Oberflächen, ausgehend von ausgewählten Ausgangspunkten, aufgebaut und erweitert. Vorlagen von Neuropil-Oberflächen aus der Segmentierung eines Referenz-Datensatzes, die anhand eines modifizierten "Active Surface" Modells einbezogen werden können, ergänzen die aktuelle Segmentierung. Die so erhaltenen Oberflächen ermöglichen es, korrespondierende Landmarken in den Bildern zu ermitteln, die für eine nichtrigide Registrierung verwendet werden. Schließlich wird dargelegt, wie die einzelnen Schritte voll automatisiert werden können, um die Bilder der Fliegengehirne aufeinander abzubilden.

Die vorgestellten Methoden sind frei als Erweiterungen für ImageJ verfügbar (Plugins). Ein direkter Vergleich mit dem VIB Protokoll zeigt, dass durch die vorgestellten Methoden nicht nur der Benutzeraufwand auf ein Sechstel reduziert, sondern dass gleichzeitig auch die Genauigkeit und Reproduzierbarkeit erhöht wird.

Contents

List of Figures

1 Introduction

The brain has been a fascinating subject of investigation ever since for numerous researchers. Its complexity is enormous, which becomes obvious even in such daily activities like driving a car during rush hour. Not less enormous is the efficiency by which it provides its capabilities. To gain a deeper understanding of how the brain works it is necessary to link its anatomical subunits to the functions they perform, and to understand how these subunits are connected, both physically and functionally. For *Drosophila*, more than for any other organism, genetic and molecular tools exist for investigating these questions. Established techniques such as the Gal4-UAS system allow to specifically express arbitrary genes in arbitrary subregions of the fly brain. Subsequent behavioural assays potentially unveil the effect of this manipulation. Finally, imaging techniques like confocal microscopy provide the means to visualise the manipulations at high resolution as 3D image volumes representing the entire fly brain. As a result of the availability of such appropriate techniques, thousands of genetically different fly lines have been established and characterised by their spatial expression patterns. Eventually, they will be associated with functional phenotypes.

This large amount of available knowledge must be organised. The *Drosophila standard brain* was developed to integrate these data in a functional atlas of the fly brain [1]. In this thesis computational methods are developed that facilitate the work with the standard brain.

1.1 Background about image acquisition

The Gal4-UAS system

The Gal4-UAS system (fig. 1.1), first published in 1993 [2], provides the means to manipulate gene expression in *Drosophila* brains. Two transgenic fly lines are required, the Gal4 driver line and the UAS effector line. The genome of the driver line contains the gene for the yeast transcription factor Gal4 under the control of a specific promoter or enhancer. The genome of the UAS line contains a Gal4 specific binding site, the so-called *Upstream Activation Sequence*

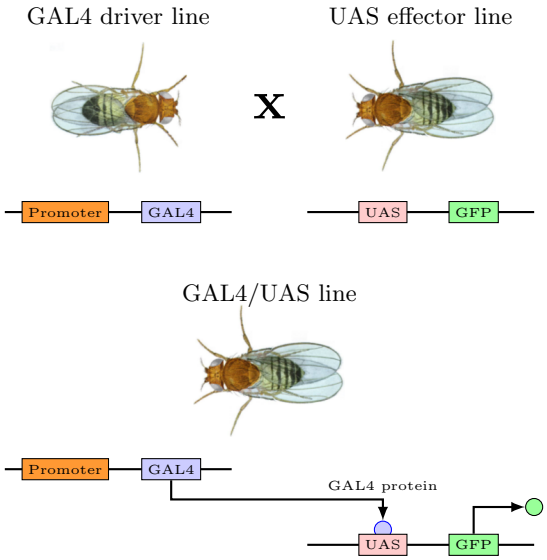

Figure 1.1: UAS-Gal4 system. The Gal4 driver line, the genome of which contains the gene for the transcription factor Gal4 under the control of a specific enhancer or promoter, is crossed with the UAS line, which contains in its genome a so-called effector gene. The genome of the progeny contains both Gal4 and the effector gene. In tissue where the promoter is active, Gal4 is expressed and binds to the UAS sequence, which activates the expression of the effector gene (GFP in the figure).

(UAS), and downstream of it an effector gene, e.g. the gene producing GFP (*Green Fluorescent Protein*). When the two lines are crossed, the genome of the progeny flies contains both the Gal4 gene and the UAS sequence. In tissue where Gal4 is expressed, it binds to the UAS sequence, activating thereby the expression of the effector protein.

The location of the Gal4 gene in the driver line dictates in which tissue the effector gene is expressed, while the UAS line determines the effector gene. The combination of different driver and UAS lines allows to express *in vivo* arbitrary proteins in a spatially restricted manner.

The fine-grained genetic control offered by molecular techniques like the Gal4-UAS system makes *Drosophila* outstanding as a model organism in brain research.

Immunohistochemistry

To visualise fly brains, the Gal4-UAS system can be used to express GFP at the synapses in the fly brain. Expressed GFP is however often too weak for reliable visualisation, so that immunohistochemical techniques are required. For the brain images used throughout this work, the mouse antibody nc82 was used as

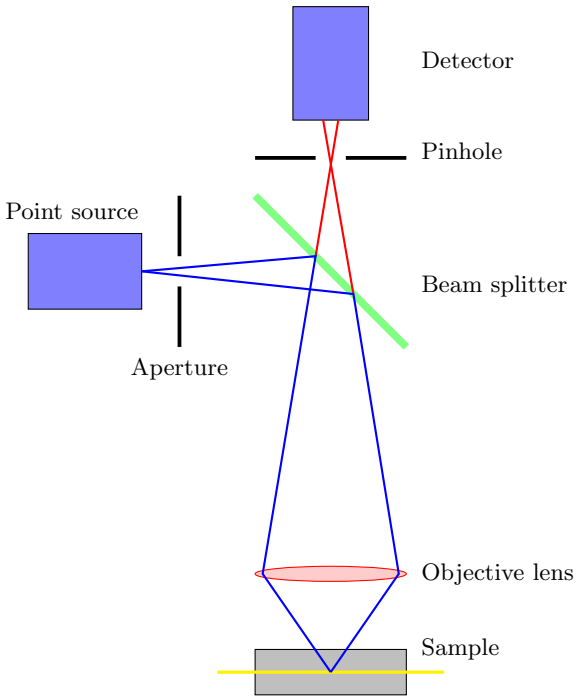

Figure 1.2: Principle of the confocal microscope. Light with a discrete wavelength, originating from a laser, is focused by an objective lens to a small subvolume of the sample. A pinhole restricts the detection to light emitted from the focal plane, and discards light emitted above or below.

a primary antibody which binds to the ubiquitously expressed synapse protein bruchpilot. A Cy3-tagged secondary anti-mouse antibody binds to the primary antibody. The red-fluorescent Cy3 is visible under the confocal microscope. A more detailed description of the staining protocol can be found in [1, 3]. All the brain images used in this thesis were part of the *Drosophila* standard brain project.

Confocal Microscopy

In a confocal microscope (fig. 1.2), a laser beam with discrete wavelengths is focused by an objective lens to excite only a small subvolume within the sample. The emitted light passes a pinhole before reaching the detecting photomultiplier. The position and geometry of the pinhole allows emitted light from the focal plane to pass and reach the detector, while light from above or below the focal plane is blinded out. This accounts for the high effective resolution achieved with confocal microscopy, compared to conventional wide-field microscopy [4]. The whole sample is scanned voxel by voxel. With a twenty-fold objective lens it is possible to record stainings of entire fly brains.

1.2 Research context and goals

The *Drosophila* standard brain

With the above mentioned techniques, studies of the *Drosophila* brain have led to the generation of a large number of genetically distinct Gal4 driver lines. The largest collections were established by the research groups of K. Ito (University of Tokyo) and G. Rubin (Janelia Farm Research Campus, Washington), but a significant number of other laboratories have generated their own fly lines. For most of these lines, plans exist to make them publicly available. Typically, fly lines are characterised by their expression patterns, which are usually stored as confocal image stacks.

In 2002, K. Rein *et al* published the *Drosophila standard brain* [1]. They described a method to align and average confocal images of adult fly brains, and applied this method to 28 brains of each gender. They subsequently selected for each gender the individual brain that correlated most to the calculated average and declared them the *Drosophila* standard brains. Image alignment (or registration) was based on manual segmentation of several brain structures. Overall, their analysis was performed using the commercially available software Amira (Mercury Inc.).

Rein *et al* applied these methods to compare volume and shape differences between male and female fly brains, between different wildtype stocks and between wildtype flies and structural mutants. Additionally, they aligned and standardised Gal4 expression patterns. For this purpose, fly brains were typically double-stained; next to the expression-specific staining, a neuropil marker like bruchpilot was used for a reference staining. Brain images were then aligned to the standard brain, based on the reference staining.

The *Drosophila standard brain* offers a standardised reference fly brain. Line-specific gene expression data can be characterised with respect to the standardised coordinates of the template and integrated into a functional map of the fly brain. Furthermore, aligned expression patterns can be averaged. Averaged images represent the expression pattern of a specific fly line more reliably than single specimens and they directly show their variability across specimens.

Research goals

The goal of this work is to develop computational methods that facilitate the work with the standard brain by improving the tools for visualisation, segmentation and alignment of confocal fly brain images. In particular, it focuses on the automation of image segmentation and registration. Its main purpose is to reduce the amount of work that is required by existing methods, thus allowing to apply the introduced techniques broadly to the growing amount of available data.

1.3 Outline of this thesis

1.3.1 The Virtual Insect Brain Protocol.

Background In 2006, Jenett *et al* published the *Virtual Insect Brain Protocol* (VIB Protocol). The VIB Protocol extended and refined the ideas of Rein *et al* and provided a unified way to register fly brains to the standard brain. It was implemented as a script suite for the commercial imaging software Amira (Mercury Inc.). In chapter 2 the VIB Protocol is revisited. Disadvantages, originating from the dependence on Amira, are revealed: 1) Users need to purchase an expensive Amira license. 2) Amira's numerous runtime instabilities lead to frequent crashes, invalidating the processing before. 3) New Amira versions are in general not backwards-compatible, so that the VIB Protocol is limited to specific versions.

Implementation To address these problems, the VIB protocol is ported to the free image-processing platform ImageJ. ImageJ is not only easily extendable by so-called plugins, but also well-established in the biomedical microscopy community. The VIB Protocol comprises several ordered steps: resampling the images, manual segmentation of a defined set of neuropils, global rigid registration, neuropil-wise local registration, and averaging the individual channels. These steps are implemented here as individual modules, which define mutual dependencies. This modular design allows each module to verify the data it depends on, and will facilitate the future extension to distributed computing.

Results The new version, based on ImageJ, improves the VIB protocol substantially. While delivering equivalent results in terms of alignment quality, the design of the graphical user interface is no longer restricted to the limited possibilities Amira offers. Consequently, usability is improved, and runtime instabilities are avoided. The new version is available as an ImageJ plugin completely for free.

1.3.2 3D visualisation

Background Chapter 3 describes a novel 3D visualisation framework for ImageJ. Numerous software packages, both commercial and open-source, exist for the visualisation of three-dimensional biomedical images. However, none of them offers an appropriate solution, because they are either targeted to end-users and do not allow to access their functionality programmatically, or they are not written in the Java programming language and thus inaccessible to ImageJ. The development of semi-automatic segmentation tools, as presented in later chapters, requires advanced three-dimensional rendering capabilities, such as a combined display of 3D image volumes and segmented surfaces. Additionally, means for interactive user input are needed, which allow an expert to guide the segmentation.

Implementation For this purpose, a hard-ware accelerated 3D visualisation library is developed. It is implemented in the Java programming language, and utilises Java 3D for hardware-accelerated rendering. 3D images can be displayed in several ways, for example as volume renderings (implemented via 2D texture mapping) or as surfaces (using a variant of the marching cubes algorithm for surface extraction). The different rendering modalities of 3D images can be augmented with custom shapes, such as lines, points or triangles. Furthermore, support for user interaction via mouse and keyboard is incorporated. Numerous other features include volume editing, annotation and animation recording and 4D data display.

Results The presented framework offers a 3D visualisation solution both for end-users (as an ImageJ plugin) and for developers. For the latter, it exposes a public application programming interface (API) which allows to access its functionality from third-party software programs. For the work presented in subsequent chapters, it forms the groundwork for the development of semi-automatic methods for segmentation, providing convenient means for interactive user input to guide the algorithms.

1.3.3 Semi-automatic segmentation

Background Image alignment in the VIB Protocol is based on manual segmentations of several brain structures in the confocal images. A user must draw the outlines of a defined set of neuropils slice by slice. This task is not only tedious and time-consuming, it is also highly subjective and thus constrains reproducibility. Chapters 4 and 5 address these problems and together provide an alternative to the VIB Protocol. Chapter 4 thereby proposes a novel semi-automatic method for segmenting the images.

Implementation The new method combines automatic surface growing with template surfaces. From a seed point, a first triangle is constructed which forms an initial surface. Iteratively, this surface is extended by new vertices. An initial guess for the position of a new vertex is obtained assuming the existing surface bends smoothly. After this, its position is optimised along the image gradient. Repeated extension yields a surface which follows salient edges in the image. Typically, the resulting surface (the model surface) lacks major parts. To complete the segmentation, template surfaces are utilised: They are aligned to the model image using a novel shape-preserving variant of an active surface model. The active surface model is initialised by aligning the template surface to the partial model surface, via the Iterative Closest Point (ICP) algorithm.

Results The methods presented in this chapter are implemented as semi-automatic tools: The 3D visualisation library of chapter 3 provides means to visualise the image, model and template surfaces. Via the 3D interface, the user selects seed points and controls the growing surface conveniently via the

scroll wheel. Furthermore, he adjusts the parameters of the active surface, the evolvement of which is continuously updated in the 3D view. The proposed methods allow to quickly and reliably segment the medulla, lobula, lobula plate and central brain of the fly brain images.

1.3.4 Surface-based registration

Background Landmarks in images can be used for model-based non-rigid registration, if correspondences across images are known. Recent research in computer vision has revealed numerous methods for automatically detecting landmark points in images and establishing correspondences between them. Well-known representatives of such algorithms are the scale-invariant feature transform (SIFT) or the multi-scale oriented patches (MOPS).

Implementation In chapter 5 it is demonstrated that these methods are not applicable to the confocal images of fly brains. Instead, a different method for identifying matching landmarks in these images is elaborated, based on the surfaces obtained in chapter 4. For both the model and the reference image, surfaces of the medulla, lobula, lobula plate and the central brain are extracted as described there. These surfaces are subsequently aligned using the Iterative Closest Point algorithm. After the alignment, a point pair (i.e. one point of the template and one of the model surface) is considered corresponding if the two points are mutually nearest neighbours. For the non-elastic registration based on the identified corresponding landmarks, two methods are implemented: a thin-plate spline based algorithm and one based on moving least squares.

Results Typically, about ten thousand point correspondences are identified per image pair. They are filtered, keeping only eight hundred of the best-matching pairs that are evenly spread over the entire template surface. Filtering eliminates false correspondences and thus avoids any warping artefacts. Empirically, the thin-plate spline based warping algorithm is found to be superior to the moving least squares approach. Using the former, the fly brain images can be registered successfully.

1.3.5 Evaluation

Background Semi-automatic segmentation and landmark-based warping, as introduced in chapters 4 and 5, together offer a powerful alternative to the VIB Protocol for the alignment of fly brain images. In chapter 6 both methods are compared regarding the quality of the resulting alignment and the required user interaction.

Implementation For the evaluation, ten fly brain images are selected from the original collection of *Drosophila* brain images that were already part of the *Drosophila standard brain* project. To run the VIB protocol in batch mode, the images are pre-segmented. Hence, for each image a labelfield is obtained, which stores for each pixel the associated neuropil. Registration is then performed using both methods, the VIB Protocol and the novel method, as described earlier. For both methods, the resulting transformations are not only applied to the original intensity images, but also to the corresponding labelfields. From the ten transformed labelfields, the mean overlap is calculated for each neuropil.

Results Using the novel methods, the time required for segmentation and registration is reduced to about twenty percent of the time needed using the VIB Protocol. At the same time, reproducibility is enhanced. The new approach outperforms the VIB Protocol also in terms of alignment quality: All of the sixteen labelled neuropils overlap to a larger extent, except one of the antennal lobes, which shows a slightly reduced overlap.

1.3.6 Towards full automation

Background Two steps of the entire process described in chapters 4 and 5 require user intervention: In the surface growing algorithm, an operator chooses seed points at the shape boundaries of interest and controls the extent to which the surface is expanded. When incorporating template surfaces to complete the segmentation, the user adjusts the parameters of the active surface model. Chapter 7 proposes a possible way to fully automate these steps.

Implementation Concentrating on the fly brain images, a preprocessing pipeline is elaborated to automatically identify seed points for the growing surface algorithm: After downsampling the images and edge-preserving smoothing by a bilateral filter, a three-dimensional variant of a gradient filter is applied. In the resulting images, only salient edges remain. Local maxima are chosen as seed points for the surface growing algorithm, which yields then partial surfaces for the medulla, lobula, lobula plate and central brain. As in chapter 4, template surfaces are aligned using the Iterative Closest Point algorithm and the aforementioned modified active surface model. For the latter, parameter settings are established per neuropil which work reliably across images. The obtained surfaces are subsequently used for landmark-based registration.

Results The proposed approach suggested a possible way for fully automating the semi-automatic tools of previous chapters. Our implementation was successfully tested with a few images from the collection of the original *Drosophila Standard Brain*. In future work, the full set of images will be used for a complete optimisation of the required parameter settings.

1.4 Dedications

Before I go into the details of this work, I would like to express my sincere thanks to all the people who have contributed to this work.

... to Prof. Martin Heisenberg, my principal supervisor. Generally, I am considering myself very lucky with the supervisors I had so far in my life. Nevertheless, the time under your supervision was outstanding. Your door was always open when I was at a loss, and you knew exactly which questions you needed to ask to reveal new directions. And still, I had the most possible freedom for my work, while being confirmed with your full confidence in me. Thank you so much for giving me the opportunity to work in your group.

... to Prof. Frank Puppe. You did not hesitate to spend your valuable time to support this work as a second mentor, although I asked you on rather short notice. Thank you very much for your help!

... to Johannes Schindelin. You have never become tired of suggesting ideas and possibilities. Whenever I needed help, I knew all I have to do is to open skype and I can be sure to find an open ear for any problems. You taught me a lot within the last years, not only workwise. I enjoyed listening to you playing the piano; you won't believe me, I even enjoyed loosing these many chess games against you. Thank your for all of it!

... my friends of the Fiji team, in no particular order: Albert Cardona, Mark Longair, the two Stephans (Preibisch and Saalfeld), Ignacio Arcandas and Pavel Tomancak. Every hackathon with you was extremely funny. I will never forget these times sitting with you over our code until late night, having beer and pizza. Not least it is due to you that I finally got my first paper accepted. Hope to see you all again soon!

... to all the people from our Department of Neurobiology and Genetics here in Würzburg for the great time here.

... to my parents. I could always come home to you, leaving any concerns behind and feel as unworried as only a child can do. Needless to say, without you I would not be here now and writing my thesis. You put me on my way, and you formed me to what I am today. And still, whenever I feel like loosing ground, I know you are there. I cannot thank you enough.

... to my girlfriend, Mandy Böhm. You accompanied me the whole time, and you are the one with whom I share my view about life. You sing for me, and you cycle with me through the near and far environment of Wuerzburg. As if this were not enough, you also were one of the most careful reviewers of this thesis. Thank you!

Thank you all so much!

2 The VIB Protocol for ImageJ

2.1 The VIB Protocol revisited

The VIB Protocol [3] takes a collection of brain images as input and calculates an average image from them, using user-provided segmented regions. For the calculation of the average image, it is necessary to align (register) the individual images to a reference image. The reference image is chosen by the user. The individual steps of the VIB Protocol are described below. The VIB Protocol can align any number of images to a reference image. For clearness, however, only two images are used for the explanations below: the reference (also called template) image and the model image (the one that is aligned to the template image).

The VIB Protocol is capable of standardising multi-channel images, given that one channel contains a reference staining. For the *Drosophila* brains, typically a ubiquitously expressed neuropil marker such as bruchpilot is used for the reference staining. Driver-specific gene expression patterns, present in additional channels, are aligned and standardised along with the reference channel. The example images in the explanations below are confocal recordings of double-stainings. The corresponding flies were obtained by crossing the mushroom body specific mb247 Gal4 driver line to UAS-mcd8GFP. Channel one contains the mb247 expression pattern, channel two the nc82 reference staining (fig. 2.1). The VIB Protocol consists of the following five steps:

Segmentation. The first step is to segment a set of neuropils in each brain image that should be standardised. This is done by manually drawing outlines of the neuropils, slice by slice. Having high-resolution images that consist of up to 200 slices, this step is extremely laborious and time-consuming. The neuropils used for the VIB Protocol are the medulla, lobula, lobula plate, mushroom body and antennal lobe (in both hemispheres). Figure 2.2 shows the segmentations of these neuropils in different slices throughout a stack.

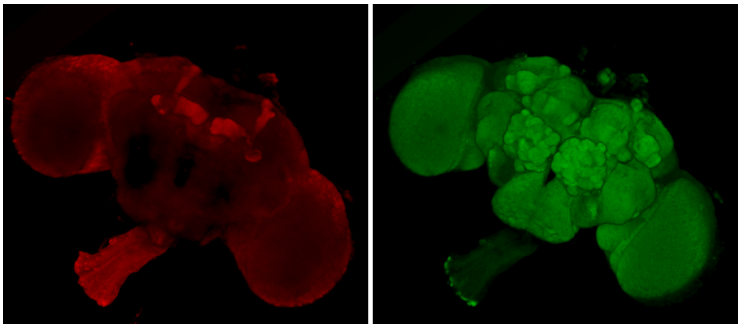

Figure 2.1: The two channels of the confocal image of a double-staining. Both channels are rendered as three-dimensional volume renderings, showing the entire adult fly brain. The green channel contains an nc82 reference staining, the red one the mushroom body specific mb247 expression pattern.

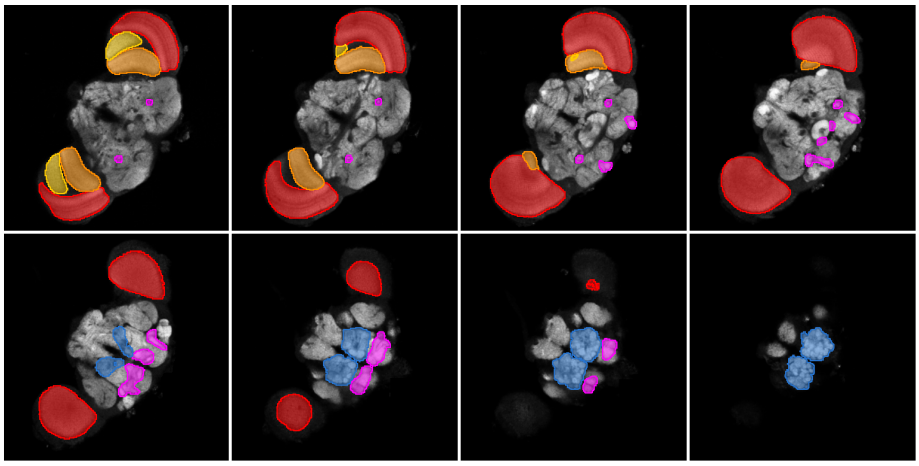

Figure 2.2: Segmentation of the image stacks. For the segmentation, the nc82 reference staining is used. Depicted here are some sample slices of a confocal stack. Segmentation is performed manually by drawing the outlines of the neuropils. Neuropils used in the VIB Protocol are the medulla (red), lobula (orange), lobula plate (yellow), mushroom body (magenta) and antennal lobe (blue).

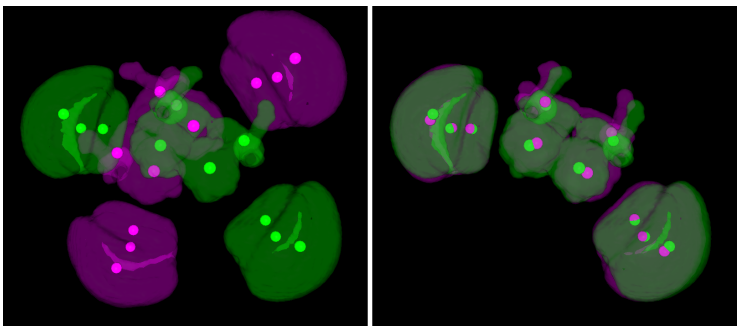

Figure 2.3: Global rigid registration. Based on the centres of gravity of the segmented neuropils, a rigid transformation is estimated that roughly aligns both brains. The model brain is shown in magenta, the template brain in green. The left image shows both brains before, the right one after the alignment.

Global rigid registration. The given neuropil areas from the previous step are subsequently used to calculate the centre of gravity for each segmented neuropil. These points represent landmark correspondences across both images. They are used to estimate a global rigid transformation that aligns both point sets ([5], see figure 2.3). This is a computationally cheap method that brings the brains roughly into the same orientation. Because fly brains of different individuals differ slightly in shape (like the size of the neuropils and their relative distance to each other), the centres of gravity do not match exactly. Further optimisation is necessary.

Local rigid registration. The global rigid registration is used to initialise local rigid alignments: Each neuropil is registered separately, again restricted to rigid transformations. In contrast to the landmark-based method in the previous step, an iterative approach must be used here. A multivariate optimisation algorithm is applied to adjust the parameters of a rigid transformation that maximises the similarity between both images. Conjugate Direction Search is used as an optimisation algorithm [6], image similarity is calculated in terms of shape overlap of the corresponding neuropil. This results in one transformation per neuropil. The optimisation for the medulla is illustrated in figure 2.4.

Diffusion interpolation. The local registrations identify an optimal transformation for each single neuropil. These transformations define a displacement field, i.e. a vector field that points from each pixel location in the model image to the corresponding location in the template image. Displacements are known at the locations of the neuropils, but unknown in between. A diffusion-like iterative algorithm is used to interpolate the vector field. Figure 2.5 illustrates the interpolation of the vector field, for clearness purposes projected to two dimensions. Figure 2.6 shows the result of diffusion interpolation for the two example images.

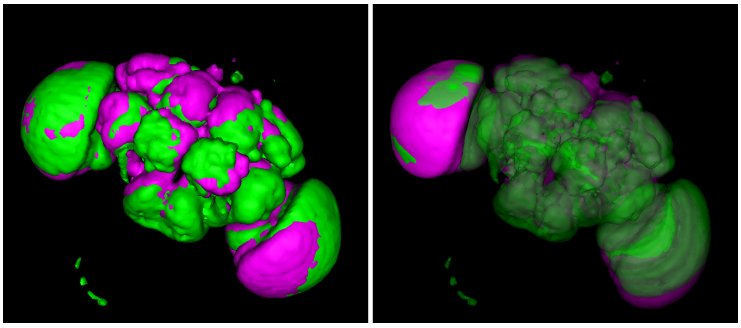

Figure 2.4: Local rigid registration. The global registration is refined for each neuropil individually. Depicted here is the registration of the medulla. An iterative algorithm adjusts the transformation parameters to maximise the overlap of the medulla in the template (green) and model (magenta) image. The left image shows both brains after the global rigid registration, the right one after the local rigid registration of the medulla.

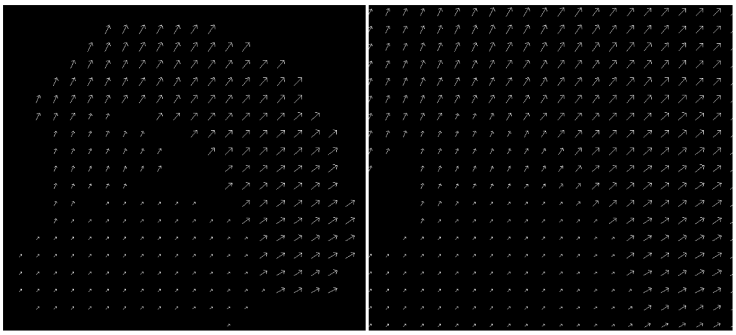

Figure 2.5: Principle of diffusion interpolation. Left the vector field initialised with the displacements obtained from the local rigid registration. Regions between neuropils are interpolated (right). Shown here is a two-dimensional analogy with only an extract of the whole image containing one optical lobe is shown.

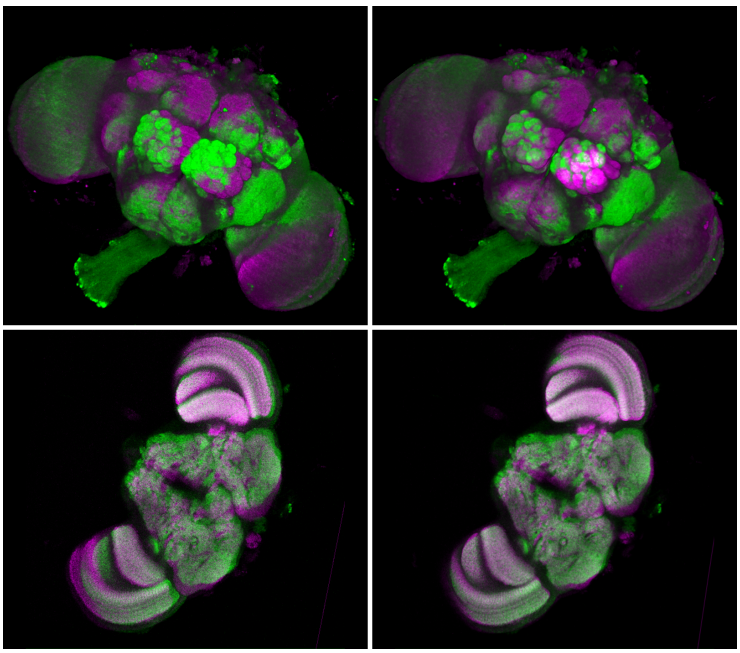

Figure 2.6: Results of diffusion interpolation. Left: the global rigidly registered brains. Right: the result of the diffusion interpolation.

Averaging transformed images. The last step of the VIB Protocol averages the transformed images pixel-wise. In addition to an average greyscale image, an average labelfield is generated. The average labelfield stores for each pixel the percentage of the overlap at that position. A value of 100 therefore states that in all processed images the same neuropil occurs at that pixel position.

2.2 Implementation as a plugin for ImageJ

As mentioned earlier, the VIB Protocol, as published in [3], was implemented as a suite of scripts for the commercial software Amira. The dependency on Amira caused several disadvantages: 1) The Graphical User Interface suffered from the limited possibilities Amira offers. 2) Amira was not freely available. Its highly advanced features, mostly not required for the VIB Protocol, accounted for its high price. 3) Changes in Amira, for example due to a version upgrade, led repeatedly to incompatibilities between the current Amira version and the available script suite. 4) Frequently occurring Amira crashes were annoying for users and forced the authors to develop a storage mechanism that allowed to restore the state before the crash. To circumvent these restrictions, we decided to re-implement the VIB Protocol on a different platform, with the following goals:

- Increased scientific audience: The new platform to be chosen should be freely available to everybody, so that users of the VIB Protocol are no

longer required to buy expensive software. A platform that is already known to the neuroanatomy research community would be desirable.

- Reliability: The platform to be chosen should run robustly, without showing the frequent crashes that occurred using Amira.
- Usability: The new version should provide an intuitive interface, making it fail-proof also for non-experienced users.

These reasons guided our decision to the open-source image processing package ImageJ [7]. ImageJ started as a very small program developed by W. Rasband. It consists of a relatively small core that contains basic image processing functionality, and is easily extensible due to its plugin-based design. This structure links users and developers, and accounts for its high popularity in the scientific community, particularly in the field of imaging and microscopy. Hundreds of plugins and macros exist, mostly developed by users in need of a certain functionality. Most of these extensions are freely available online.

The VIB Protocol can benefit from several features of ImageJ: It is written in the Java programming language, which facilitates development and deployment. It runs on any operating system supported by Java, without requiring any platform-specific testing by the programmer. Many scientists in the field already use ImageJ, which frees them from learning the usage of yet another software. Finally, it provides support for all common file formats of microscope images.

To meet ImageJ's structure, we implement the VIB Protocol as a plugin. The individual steps of the VIB Protocol are designed as separate modules. All modules implement a common interface and specify dependencies on other modules (e.g. the module for calculating the local rigid transformations is dependent on another module for segmentation). Each module checks that previous modules are executed completely and correctly before starting its own work. This ensures the correct functioning of the overall protocol.

2.3 Usage of the ImageJ-based VIB Protocol

Running the VIB Protocol

The VIB Protocol is installed like other ImageJ plugins: After downloading the JAR archive and copying it into ImageJ's plugin folder, ImageJ needs to be restarted. Alternatively, the VIB protocol can be used under Fiji [8]. Fiji is an ImageJ distribution which bundles a large collection of lifescience-oriented plugins. The VIB Protocol is already included in Fiji. After installation, the VIB Protocol can be launched from a new entry in ImageJ's (or Fiji's) "Plugin" menu, called "VIB Protocol". This opens the main input dialog (fig 2.7), which contains various input fields with the following meanings:

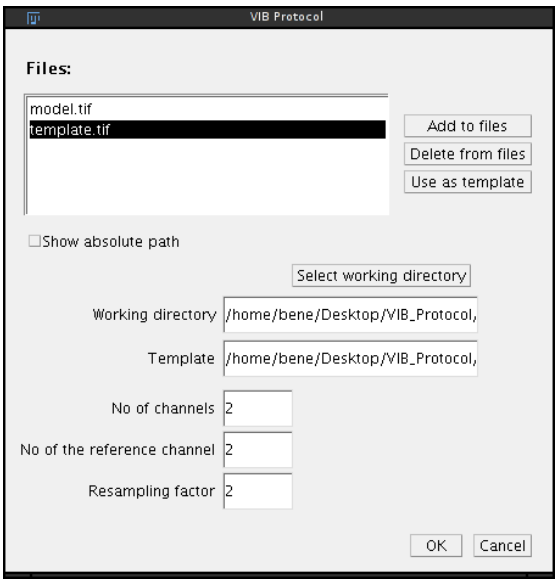

Figure 2.7: Input dialog of the VIB Protocol. This window is exposed to the user after he has started the software. It provides the means to specify the image files for registration and various parameters which are explained in greater detail in the text.

Files The area below "Files" contains the images that should be aligned to the template image. The user can add and remove images using the corresponding buttons next to this area.

Working directory The VIB Protocol stores all results in a dedicated output folder, which must be specified here.

Template One image must be selected as the template image. This can either be one of the images in the "Files" area, or a different image on the hard disk.

No of channels The VIB Protocol can process multi-channel files. Here the user must specify how many channels the files contain.

No of reference channel One channel is used as a reference channel. This is the channel exposed to the user for segmenting the neuropils that are later used for registration. When working with multi-channel images, the user can specify here which channel represents the reference channel.

Resampling factor To accelerate processing of very large images resampling should be considered, which can be specified here. When a resampling factor of "1" is used, the original images are processed, otherwise the images are downsampled by the given factor.

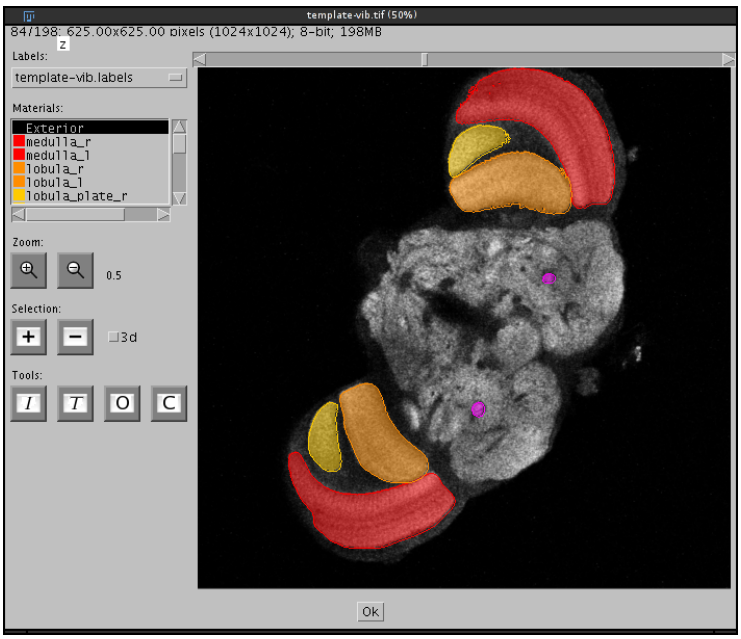

Figure 2.8: Segmentation Editor. The Segmentation editor is implemented as an independent ImageJ plugin which resembles the graphical user interface of Amira's segmentation editor.

The only other step that requires user interaction is the segmentation of images. For this purpose, the segmentation editor is opened automatically for each image. The segmentation editor (fig. 2.8) was implemented to resemble the graphical user interface of Amira's segmentation editor, facilitating for users of the previous Amira-based VIB Protocol to switch to the new version conveniently.

Particularly two features of the segmentation editor are worth mentioning here because they ease this time-consuming step. The first one is local thresholding: The user can select a rough region of interest and then adjust a threshold that is only applied to the selected area. For fly brains, this accelerates particularly the segmentation of the neuropils in the optic lobes. Another feature is the interpolation of segmentations across slices: If the regions to be segmented do not change much between consecutive slices, it is sufficient to segment them in every 3rd, 4th or even 5th slice. Missing segmentations in between are then obtained by shape interpolation.

Structure of the output

As mentioned earlier, the results of the VIB Protocol are saved to a dedicated, user-specified output directory. After running the protocol, the following subfolders are available:

images_1... images_n	n folders containing the result of splitting the original image data into n separate channels.
labels	Contains the labels, resulting from the segmentation step.
resampled_1... resampled_n, resampled_labels,	If the user has specified a resampling factor greater than one, these folders contain the resampled image data of the n channels and the resampled labelfields respectively.
statistics	Contains information about the labelfields, such as centres of gravity, volumes, etc..
warped_1... warped_n warped_labels	n+1 folders containing the n aligned channels and the labelfield of each image.
output_1... output_n output_labels	n+1 folders containing the averaged channels over all images and the probability map of the labelfields.

Batch processing

Apart from the input dialog at the start, the segmentation is the only step that requires user interaction. To run the VIB Protocol in batch mode, one can segment the images in advance, using the segmentation editor as a standalone plugin. The resulting labelfields must be copied to a folder called "labels" in the output directory. The VIB Protocol then detects the labels automatically, it avoids opening the segmentation editor and runs without further interruption.

Facilitating registration and segmentation of *Drosophila* brain images is the focus of this work. After describing the implementation of a 3D visualisation framework in the next chapter, I will return to these problems in chapters 4-7. In these chapters, I will develop new semi- and fully-automatic methods as alternatives to the VIB protocol. Both methods accelerate the registration significantly and yield more reproducible results.

3 3D Visualisation of neuroanatomical data

In the previous chapter I have described the implementation of the VIB Protocol for the non-commercial open-source platform ImageJ. ImageJ however was primarily build for two-dimensional image processing. When new imaging methods like confocal microscopy or electron microscopy emerged, it became important to support three-dimensional data structures. This was built on top of the existing ImageJ, by introducing the concept of image stacks, which represented collections of two-dimensional image slices. Still, all the algorithms worked (and mostly still work) on a 2D basis, but could optionally be applied to all images within a stack. More importantly, the only way to display 3D images was to show them in a so-called stack window, which allowed users to scroll through the slices of a stack. However, a real three-dimensional view was missing.

To establish and improve the usability of ImageJ in the area of neuroanatomy in general, and in particular to offer users of the VIB Protocol advanced visualisation possibilities for their three-dimensional data, we decided to develop a viewer that can display image stacks in a virtual three-dimensional space. What started as a small side-project turned out to become a major part of my work. Not only did it fill a gap in ImageJ, which was a useful extension for many of its users. Over time it also developed from an end-user application to a flexible programming library that allows other applications to easily incorporate 3D visualisation. Hence, it supports the development of new algorithms for 3D image processing by offering means for testing their correct functioning, for debugging them and for interactive user input. Several third-party software products use it by now, some of which will briefly be introduced at the end of this chapter (section 3.8). For my own work, the viewer was of essential value for the development of the semi-automated segmentation procedure that will be described in chapters 4 and 5. This is why I devote a whole chapter to this part of my work.

This chapter is organised as follows: In section 3.1 I briefly review existing 3D visualisation software and discuss why they did not fulfil our requirements. Section 3.2 gives a short introduction to the concepts of Java 3D and Java 3D-based programs. Section 3.3 outlines the internal structure of our framework,

and section 3.4 shows how conventional image stacks can be rendered in a virtual three-dimensional space as surfaces, volume renderings and orthoslices. Section 3.5 then shows how these renderings of image stacks can be augmented by custom geometric shapes, like points, lines, triangles and quads. Usage of the framework, both for end-users and developers, is described in section 3.7, other features of the framework are summarised in section 3.6, and its application in third-party applications is discussed in section 3.8.

3.1 Previous work

We have identified a lack of accessible 3D/4D visualization software libraries for biological image processing. Numerous image processing packages exist, either commercial (Amira, Visage Imaging; MeVisLab, Mevis; Imaris, BitPlane; Volocity, PerkinElmer) or open source (VOXX, [9]; VTK and VTK-based applications such as Slicer3D, BioImageXD, and V3D [10]; VolumeJ [11] and Volume Viewer [12]). These packages offer excellent solutions for the specific problems they were designed to solve. While end-users benefit from well-documented, special-purpose commercial applications, the development of custom analytical tools is better handled by open source packages. The application programming interfaces of existing packages range from the non-existent for most closed commercial solutions, to the very detailed and comprehensive open source VTK environment.

We have created a software library for 3D/4D visualization, with functions for surface extraction, volume rendering and interactive volume editing. Our library removes all the complexity of creating and interacting with image volumes and meshes in a 3D environment. We have designed our library to enrich the core functionality of ImageJ (and its descendant Fiji [8]), an open source image processing application. Via ImageJ, our library has access to hundreds of biological image file formats. Over the years, the scientific community has contributed a very large number of ImageJ extensions, known as plugins, which provide readily accessible implementations of numerous computer vision algorithms. With our library, we empower the ImageJ scientific community to rapidly implement custom analytical tools for 3D/4D data sets, with a minimal investment of time and resources in handling the complex details of a hardware-accelerated 3D environment. This reduction in the difficulty of visualizing 3D information commoditizes the usage of a 3D scene. For example, our library enables software developers to visually assess the correctness of individual algorithmic steps, such as the 3D shape of a mesh deformation. The simplicity of our library is in stark contrast to existing libraries such as VTK, which require detailed knowledge of the underlying data structures.

Java 3D Node	Function
`BranchGroup`	A node capable of being the root of several subtrees.
`TransformGroup`	A node that transforms the subtree below it by a specified transformation
`Shape3D`	A node representing a displayable object. A `Shape3D` consists of a `Geometry` and an `Appearance`
`OrientedGroup`	A node that orders its children in a defined z-order
`Switch`	A node having several child nodes that can be toggled on or off.

Table 3.1: The most prominent Java 3D node classes and their associated function in the scene graph.

3.2 A short primer about Java 3D

3D visualisation is computationally expensive. The power of the CPU alone, even on modern computers, is not sufficient for smooth and interactive 3D rendering. Graphic hardware, equipped with its own processors and memory, is developed and optimised for highly parallel matrix calculations to support the CPU in rendering complex 3D scenes. Two interfaces exist that allow to access its resources programmatically, DirectX and OpenGL. Unfortunately, programs written for DirectX will typically not run under OpenGL and vice versa. Instead of addressing these interfaces directly (through their Java bindings, like Jogl), we use Java 3D. Java 3D is a fully object-oriented 3D visualisation library, which forms an abstraction layer between software applications and the underlying graphic interface. It uses whatever is present in the machine on which the program is running, either DirectX or OpenGL. If no hardware acceleration is available at all, it will fall back to (slow) software rendering instead of failing or crashing.

The most informative Java 3D documentations utilised for the implementation of this framework include Sun's API tutorial [13] and the Java 3D online API [14].

Java 3D uses the concept of a scene graph. The scene graph is an acyclic directed graph that specifies the content and properties of a virtual universe. Each 3D object is defined by a sequence of scene graph nodes characterising its geometry, appearance, location and transformation. The Java 3D rendering engine uses the information in the scene graph to project the given arrangement of the 3D data volumes onto a 2D canvas. Different Java 3D node classes exist which have different functionality. An overview over the most important nodes is displayed in table 3.1.

Each Java 3D based program begins typically by instantiating a `VirtualUniverse`, a class that represents the virtual 3D space in which objects are displayed. The root node of the scene graph (an object of the class `Locale`) can be accessed through the `VirtualUniverse` object and can be used to add

nodes to the scene. A simple example makes the concept of the scene graph clear: To implement two objects that can be transformed both together and individually, a `TransformGroup` needs to be attached to the root node, which would be responsible for the common transformation of the two objects. To the `TransformGroup`, a `BranchGroup` is added, which in turn gets two child `TransformGroup`s, each of which is responsible for the transformation of the object attached to it.

A 3D object itself is represented in Java 3D by the class `Shape3D`, which consists of a `Geometry` object and an `Appearance` object. All possible geometries of objects are composed of lists of so-called primitives, which are either quads, triangles, lines or points. In contrast, the `Appearance` has many parameters like colour, transparency, textures, etc. The objects to be rendered in our software are isosurfaces, volume renderings and orthoslices. Their Java 3D implementation will be described in the following sections.

3.3 Overall program structure

The overall structure of our software is best described by its scene graph and the relationships between the core classes. The scene graph (fig. 3.1) consists of a trunk that splits up into a subtree for each 3D object. The trunk is composed of global transformations that are applied to the whole 3D space: A `TransformGroup` scales the whole scene globally, so that it fits into the field of view. Two `TransformGroup`s globally rotate and translate the scene and are changed upon user interaction. Yet another `TransformGroup` centres the scene in the field of view. To the last `TransformGroup`, a `BranchGroup` is attached which splits the scene graph into several subtrees, each of which belongs to one individual object. Each subtree in turn consists of different `TransformGroup`s, which are responsible for object-individual transformations. At the very end of each subtree, there are the `Shape3D`s that finally represent the 3D objects themselves, which are either volume renderings, isosurfaces or orthoslices. The provided information about transformations and visual properties is used by Java 3D's rendering engine to generate a 2D projection of the three-dimensional objects onto the screen.

A simplified class diagram illustrates the relationships between the core classes in the framework (fig. 3.2). The base package is `ij3d`, which contains the two principal classes `Content` and `Image3DUniverse`. The latter represents the virtual space in which objects are displayed. It extends Java 3D's `VirtualUniverse` class and thus holds the root of the scene graph. When it is instantiated, it composes the above described trunk of the scene graph. Because all 3D objects must finally be attached to the trunk, `Image3DUniverse` provides methods for adding and removing objects. Several other methods are offered by this class for changing view parameters, such as the background colour, and adjusting the global transformations.

Each 3D object is described by an instance of the `Content` class. This class contains a reference to the image stack it displays, and additional attributes like

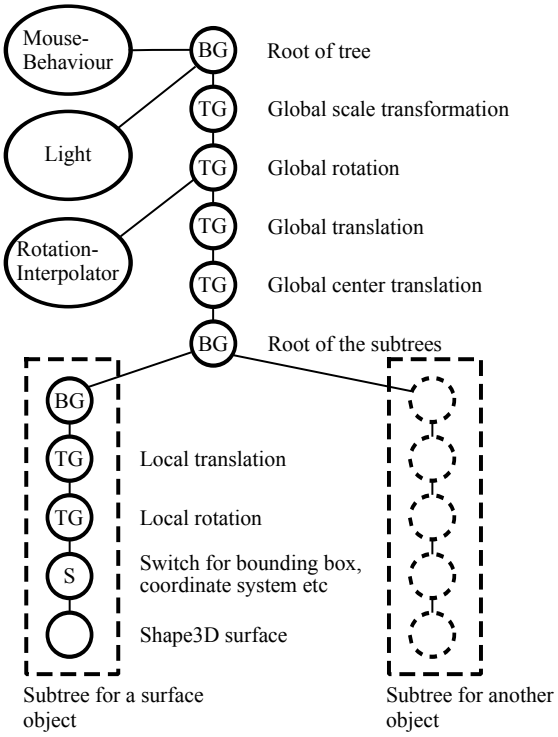

Figure 3.1: Java 3D scene graph of the 3D viewer. Java 3D is based on a scene graph, which determines which and how 3D objects are rendered. The scene graph of our framework consists of global and local `TransformGroup` (TG) objects. Global TGs represent transformations applied to all objects, local TGs represent transformations of individual objects. Other nodes include `BranchGroup` (BG) nodes, which are capable of holding several subtrees, and `Switch` (S) nodes (the child nodes of which can be toggled on and off).

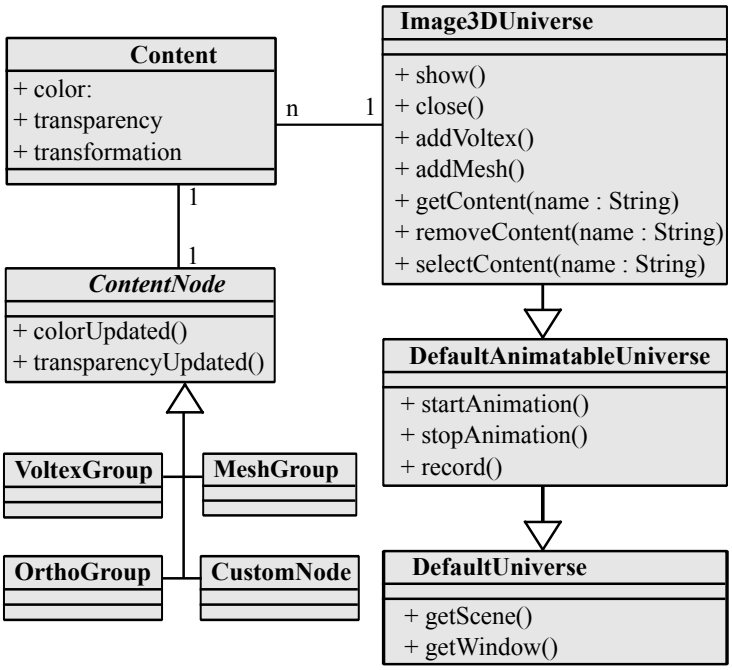

Figure 3.2: Simplified class diagram of the 3D viewer. The most important interactions between the core classes are depicted here. For normal users of the API, two classes are sufficient for accessing most of the framework's functionality: `Image3DUniverse` and `Content`. The former represents the 3D space and provides amongst others functions for adding and removing objects, the latter represents 3D objects with functions for changing attributes like colour, transparency etc.

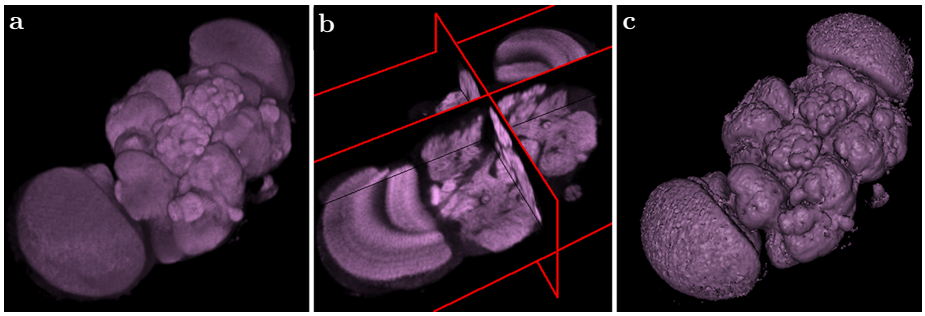

Figure 3.3: Implemented rendering possibilities for volumetric data. Shown here is a confocal stack of a *Drosophila* brain, rendered (a) as a volume rendering, (b) as orthoslices and (c) as an isosurface, using a threshold of 50.

the colour and transparency. It offers methods for changing these attributes as well as the local transformations. Each `Content` object has exactly one reference to a `ContentNode` object. `ContentNode` itself is abstract and extended by the four classes `VoltexGroup`, `MeshGroup`, `OrthoGroup` and `CustomNode`. They represent volume renderings, surface renderings, orthoslices and custom geometries respectively.

3.4 Displaying stacks of 2D images in 3D

This framework aims to provide fast and advanced 3D visualisation possibilities. ImageJ's capabilities are limited to a basic stack window. We implement three additional rendering modalities, which I describe in more detail below: isosurfaces, volume renderings and orthoslices.

3.4.1 Displaying image stacks as volume renderings

Background
In contrast to other rendering methods, volume rendering does not require preprocessing or information extracting prior to the actual rendering step. It preserves as much information of the raw data as possible. Therefore, it is mostly used to visualise the data as it is acquired by various microscopic methods.

Several ways exist to implement volume renderings. The first method investigated in the past is ray-tracing, often also called ray-casting. Ray-tracing models a ray of sight for each pixel on the screen. Each such ray originates from the users virtual eye position and extends beyond the corresponding screen pixel. The 3D image volume is virtually placed behind the screen. Where the extended ray intersects the 3D volume, equidistant samples are taken along the ray, usually using tri-linear interpolation. Thus, the ray can be thought of as the light ray reflected by the 3D object and reaching the observer (in reverse

direction). To each sample point, an opacity value is assigned, using a transfer function. Then, the samples along each ray are integrated to yield a final pixel value.

Ray tracing results in particularly realistic, high-quality renderings, at the cost of a high computational load: Since consumer graphic hardware does not support ray-tracing *per se*, it was restricted to CPU implementations until the advent of programmable graphic hardware, rendering it too slow for many biomedical visualisation applications. Approximation and optimisation methods have been investigated in the past, limiting the number of traversed volume pixels of each ray and allowing rays to terminate early once a certain opacity value is reached [15, 16]. Despite these advances, other hardware-accelerated methods were favoured in the meantime by commercial applications, because of their higher speed. With the advent of programmable graphic hardware, however, ray-tracing re-attracted increased attention: Since each ray is independent, calculations could easily be paralellised. This allowed to transfer large parts of the computations to the graphic hardware, using custom fragment shaders [17, 18].

Another variant of volume rendering implementation is based on texture mapping (see e.g. [19]). Being supported on consumer graphic hardware, this is probably the most applied method in today's commercial applications. 2D and 3D texture mapping are distinguished. 2D texture mapping arranges the data as 2D rectangles one after another in the virtual 3D space, in an arbitrary orientation. To each rectangle, a different image slice through the original volume is mapped as a texture. An opacity value is assigned to each pixel, governed by a transfer function. Depending on the direction of view and the orientation of the object, slices are either taken in the xy-, xz- or yz-plane through the original volume. This accounts directly for the high memory requirement of this method: Three copies of the data need to be cached, each of which is loaded and displayed depending on the view direction. Only 2D interpolation within each texture needs to be performed, which is supported by all common graphic cards. This leads however to rendering artefacts when the displayed set of slices is exchanged due to a change of the orientation. 3D texture mapping avoids both problems. Unfortunately, it is only available on dedicated graphic hardware supporting 3D interpolation. Additionally, the size of 3D textures is very limited.

In this project, we want to implement 3D visualisation for a broad user audience, reducing hardware requirements as much as possible. Our software should not be limited to dedicated workstations, but also run on common laptops, as they are often found in biological research laboratories. Since CPU-based ray-tracing is too slow for rendering the large data sets acquired by biomedical microscopy, and both accelerated ray-tracing and 3D texture mapping require dedicated graphic hardware, we decide to implement 2D texture mapping.

Implementation

The work presented here is highly inspired by Doug Gehringer's online tutorial [20]. For 2D texture mapping, the volume data to be displayed needs to be

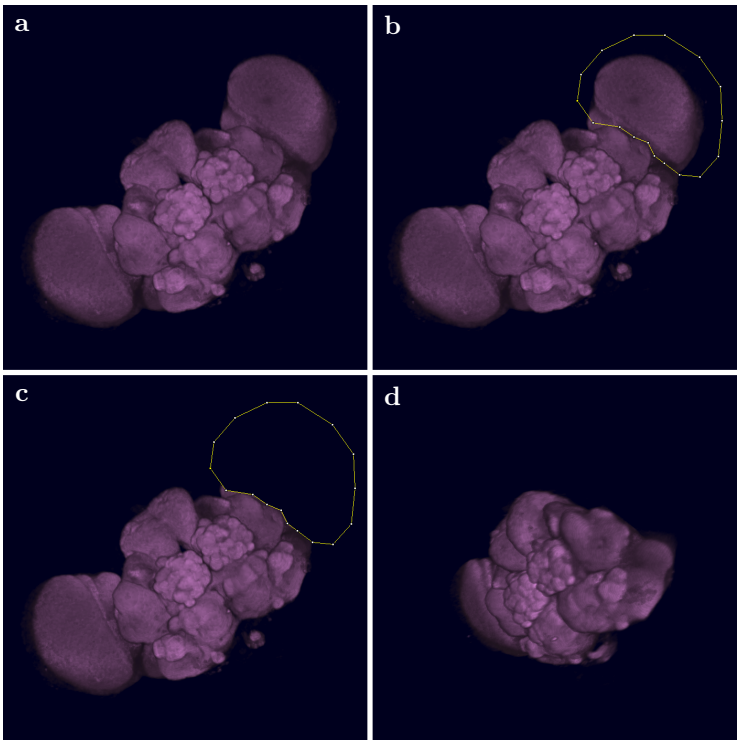

Figure 3.4: Interactive volume editing. Shown here is a volume rendering of a *Drosophila* brain, together with a user-provided 2D ROI. The ROI was projected back onto the image volume, and the intersected region was filled black. Afterwards, the brain was rotated to show the effect.

arranged in sets of 2D slices. This meets ImageJ's stack concept. Here, however, three different stacks are required, resulting from reslicing the volume in x-, y-, and z-direction. Each slice constitutes a `Shape3D`. As mentioned earlier, a `Shape3D` consists of a `Geometry` and an `Appearance`. The `Geometry` of a slice is simply given by a rectangle, the `Appearance` needs to load the appropriate volume slice as a texture and assign a transparency value to each of its pixels. In this way, three sets of `Shape3D`s are obtained, each of which represents one view direction. The slices of each set are attached to an `OrderedGroup` object, which takes care of their correct z-ordering. The slices of the first three sets are cloned, ordered reversely and attached to three additional `OrderedGroup`s, representing the negative view directions. From the resulting six `OrderedGroup`s, only one is displayed a time, depending on the view direction. For this purpose, the six `Group`s are combined in a `Switch` node. `Switch` nodes allow to toggle the visibility of their child nodes.

Results

Figures 3.3a and 3.4 show the volume renderings of a Drosophila brain, obtained by confocal microscopy. Next to pure viewing, we implement methods that allow the user to interactively edit volumes: The user can draw a 2D region of interest

(ROI) on the canvas window. Each pixel of the 3D image volume the projection of which falls within this ROI can then be set to an arbitrary colour. Using black as a fill colour makes the corresponding pixels fully transparent and thus effectively crops the data. This is shown in figure 3.4.

3.4.2 Displaying image stacks as isosurfaces

Background
In contrast to volume renderings, isosurface renderings do not render the volumetric data directly. Rather, geometric information is extracted in form of a surface that is composed of polygons, typically triangles. Thus, isosurface renderings consist of two steps: extracting the surface triangles from the volumetric data and rendering them in a virtual 3D space.

The second step is straightforward, using a low-level 3D library like OpenGL or Java 3D. These libraries perform the necessary calculations for projecting the given list of polygons onto the 2D screen, taking visual effects such as light into account. This paragraph will therefore focus on the surface extraction step.

Isosurfaces assume that the object of interest is separable from its background by adjusting a threshold intensity value (similar to threshold-based segmentation). The probably widest used technique for extracting a surface from a 3D volume is the marching cubes technique [21]. Here, a cube "marches" through the data volume pixel by pixel. At each position, the intensity values at the corners of the cube are compared to the pre-defined threshold value. If both values below and above the threshold occur, the isosurface must intersect the cube. The algorithm then generates, based on a lookup table, the appropriate triangles (faces) which resemble the part of the isosurface intersecting the cube.

For each of the eight corners of a cube, the intensity value may be below or above the isovalue. Thus 256 (2^8) cases can be distinguished. Common implementations use a lookup table that stores the appropriate triangles for each of these cases (also called configurations).

After the original publication of the marching cubes algorithm, much work has been performed on improving shortcomings and performance. It was realised that the original version could not only lead to topological inconsistencies, but also to incorrect isosurfaces. Several researchers worked on resolutions to these issues [22, 23, 24], mostly by developing modified and better suited lookup tables.

Other work has focused on accelerating the surface extraction. A broadly-investigated approach attempted to avoid the traversal of non-contributing empty cubes. Wilhelms *et al* utilise the octree data structure for this purpose. An octree is a hierarchical data structure: At the top level, the whole data volume is represented by one cube. At each consecutive level, the cube of the parent level is split into eight child cubes. Each cube stores its maximum and minimum

intensity values. By comparing the isovalue with both extrema, the marching cubes algorithm can already decide at a high level into which parts of the volume it needs to descend. Empty parts are skipped early. Other interesting methods aiming to avoid empty cubes include span-based techniques (the ISSUE and NOISE algorithms, [25, 26]). They store the minimum and maximum for each single cube in a kd-tree, allowing to efficiently separate empty cubes from non-empty ones.

Yet other approaches try to accelerate isosurface generation by avoiding to produce polygonal surfaces at all. The polygonal representation is typically used to render the results; reported rendering alternatives are based on ray-casting [27, 19] or on a point-based rendering of the resulting isosurface vertices [28].

The marching cubes algorithm is still subject of further improvement and refinement. A comprehensive review was published recently [29], summarising the different development branches.

Implementation

In this work, we implement the surface extraction method called "Marching Cubes 33", which uses an extended lookup table to avoid ambiguities and to yield a topologically correct isosurface [22]. The C++ source code, published together with a follow-up paper [24], is re-implemented in the Java programming language, to be accessible to our framework. Having extracted the set of triangles comprising the entire surface, a Java 3D `Geometry` object is instantiated from them. Together with an `Appearance` object, they compose the final `Shape3D` that represents the isosurface rendering.

Results

Figure 3.3c shows the isosurface rendering of a Drosophila brain. The original volumetric image was obtained by confocal microscopy. After applying Gaussian blurring, the marching cubes algorithm was applied with a threshold value of 50 (with intensity values ranging from 0 to 255).

Using a cube size corresponding to the dimensions of one voxel usually generates an enormous number of triangles, often slowing down the rendering process. The number of resulting triangles can be reduced by increasing the cube size of the algorithm, but this causes an undesired degradation of the surface resolution. Here we follow a different idea: In a post-processing step we reduce the number of triangles, based on the local surface curvature. Plane parts of the surface can be represented with less, larger triangles without loss of detail, while the high number of triangles is preserved in regions with high curvature. We implement a method that iteratively contracts edges in a flat neighbourhood. Our approach is inspired by work published in [30]. A more detailed description is postponed to chapter 4 (p. 46), since edge contraction also plays an important role for the work on segmentation elaborated there.

3.4.3 Displaying image stacks as orthoslices

Orthoslices display volumetric data by showing three orthogonal planes. From the implementation point of view, orthoslices are a special case of volume rendering. In contrast to those, only one slice is shown of each view direction, and all three view directions are shown at the same time. Figure 3.3b shows the orthoslices of the *Drosophila* brain image. In our application, the user can scroll through the slices and toggle the visibility of individual view directions. Orthoslices are particularly useful if the region under study is inside the volume.

3.5 Displaying custom geometries

Background
So far, it has been described how volumetric image data can be visualised in 3D, either as volume renderings, isosurfaces or orthoslices. While this probably suffices the common needs of end-users, application developers who use our framework as a programming library may require additional capabilities. This section shows how the aforementioned rendering modalities can be augmented by custom shapes. Arbitrary geometric objects in the form of point, line, triangle and quad meshes can be combined with the above described visualisation possibilities for image volumes. Various use-cases illustrate the benefits: To visualise a 3D feature detection algorithm, for example, the image data can be displayed as a volume rendering, together with the identified feature points as custom point objects. Similarly, to visualise a 3D segmentation algorithm one can display the 3D image as orthoslices, augmenting it with the segmented surface as a triangle mesh. Particularly this latter case will recur in the following chapters of this thesis that address the problem of (semi-automatic) 3D image segmentation (see for example figure 4.7, p. 45).

Implementation and results
To implement support for arbitrary shapes in our framework, a new package called `customnode` is added. It contains mainly four new classes `CustomTriangleMesh`, `CustomQuadMesh`, `CustomLineMesh` and `CustomPointMesh`, which represent custom triangle, quad, line and point meshes, respectively. All four classes extend a newly introduced abstract class `CustomMesh`. A reference to a `CustomMesh` object is kept in `CustomMeshNode`, which extends the abstract class `ContentNode` (fig 3.5, compare also figure 3.2).

Because `CustomMesh` itself is a `Shape3D`, the four new classes are again composed of a `Geometry` and an `Appearance` object. The `Appearance` is constructed similarly to that of isosurface shapes, while the `Geometry` utilises the specialised Java 3D `Geometry` classes `TriangleArray`, `QuadArray`, `LineArray` and `PointArray`.

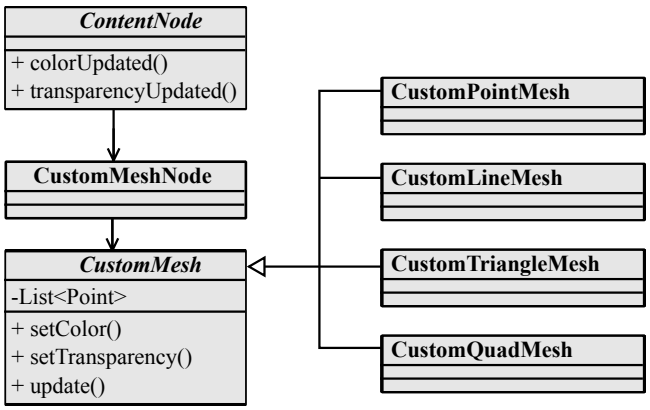

Figure 3.5: Class diagram of the `customnode` package. `CustomMeshNode` extends `ContentNode` (see also figure 3.2) and holds a reference to an object of type `CustomMesh`, an abstract class that is implemented by the four classes `CustomPointMesh`, `CustomLineMesh`, `CustomTriangleMesh` and `CustomQuadMesh`, which represent point, line, triangle and quad meshes respectively.

3.6 Other features

Various other features of our framework that were not mentioned so far are briefly summarised below.

Annotation in 3D space.
The 3D scene can display landmark annotations for each image volume. These are added using the point tool of ImageJ's toolbar. Existing landmarks are listed in a table that allows the manipulation of their properties, such as name and colour. Each image volume hosted in the 3D scene may have an associated set of 3D landmarks of this type. A set of landmarks may be stored in a file for external analysis and reloaded in subsequent annotation sessions.

Landmark-based 3D rigid registration of image volumes.
Two sets of homonymous landmarks positioned over two corresponding image volumes can be used for estimating a rigid transformation model [5] (fig. 3.6). Using this model, one image volume can be aligned onto the other. The "Transform" menu offers options for exporting the transformed image volume as an image stack suitable for further processing with ImageJ.

Animation and recording.
The 3D viewer offers an option to record a 360-degree rotation of any 3D scene. Additionally, a recording mode is available. When this is activated, every manual rotation, translation and zooming of the display or any of its elements is

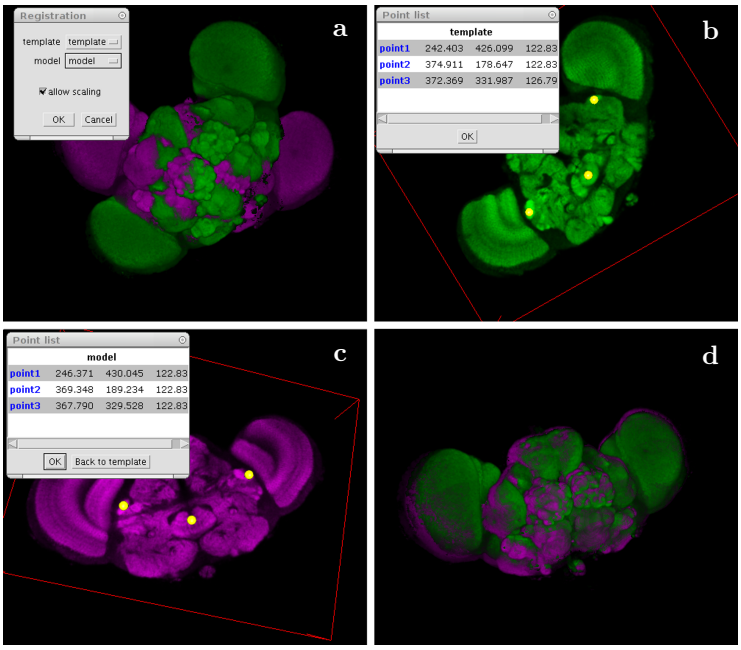

Figure 3.6: Interactive landmark-based registration. (a) Two unaligned *Drosophila* brains. The reference brain is shown in green, the model brain (the one to be registered) is shown in magenta. (b) Landmark selection in the model brain. The display mode is automatically changed to orthoslices. The user can scroll through the image planes and select landmarks, which are displayed in a separate dialog window. Landmarks are shown yellow in the figure. They can be moved, highlighted and renamed. (c) Subsequent landmark selection of the template brain. (d) The landmarks are used to infer a rigid transformation that aligns the model image to the template.

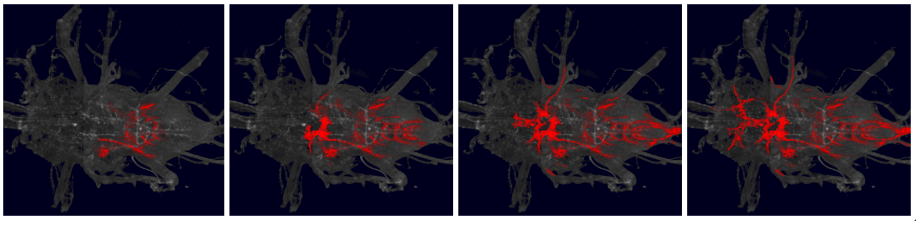

Figure 3.7: Simulation of dendritic growth in the thorax of a *Drosophila*. To simulate the growing of dendrites, the stack was displayed as a grey volume rendering, into which the dendritic parts were painted continuously in red.

recorded; when stopped, the recording is displayed as an ImageJ stack. Recordings may be output as videos via ImageJ, to embed them for example in presentations.

Displaying 4D data
Modern microscopic techniques allow sometimes live imaging over time. Single Plane Illumination Microscopy (SPIM), for example, is used to image and study embryogenesis in *Drosophila* [31, 32]. The resulting time-lapse recordings of 3D data sets can be visualised in our framework. Standard command buttons for play, pause, fast-forward, etc. control the time point displayed in the viewer. Interactive zooming, rotation and panning are enabled as the time sequence progresses. When paused, the visualisation of the current time point may be annotated, interacted with and measured as with any other 3D scene.

Time-dependent display of data can also be achieved by changing individual voxel values in volume renderings. This method is used to simulate the growth of dendrites: Figure 3.7 shows the thorax of a *Drosophila*. It displays the thorax as a grey volume rendering, into which the dendritic parts are painted continuously in red. The source code for this example is shown in appendix C.

3.7 Usage

Usage as a GUI application

Our 3D visualisation library includes a fully functional plugin for ImageJ named "3D Viewer". The plugin is listed automatically in ImageJ's plugin menu. When executed, the plugin initialises a new 3D scene, and automatically offers a dialog for displaying any open image stack. The dialog provides the means to alter the attributes of the image volume, such as its representation type (volume rendering, isosurface or orthoslices), and its colour and transparency settings. The menu of the 3D scene window offers options for inserting further image volumes and editing, annotating and transforming them. Extensive documentation is available online `http://3dviewer.neurofly.de`, along with video tutorials and a "Frequently Asked Questions" section.

Usage as a programming library

Our framework exposes a public API to allow applications to integrate its features. A basic source code example demonstrates the use-case of visualising in 3D an image volume and a mesh. The example illustrates the development of an image segmentation algorithm, which extracts the boundary of the structures of interest as surfaces and represents them as a mesh composed of triangles.

Listing 3.1: A basic source code example.

```
1  Image3DUniverse univ = new Image3DUniverse();
2  univ.show();
3
4  ImagePlus imp = IJ.openImage("flybrain.tif");
5  Content c = univ.addOrthoslice(imp);
6
7  List<Point3f> vertices = createVertices();
8  CustomMesh cm = new CustomTriangleMesh(vertices);
9  univ.addCustomMesh(cm, "triangle mesh");
```

Line 1-3 The first step is to instantiate an object of the class `Image3DUniverse`. Calling its `show()` method opens a new window to show the 3D scene. The scene graph is set up automatically.

Line 4-6 Next, the image volume is loaded. It is displayed as orthoslices in the 3D scene by calling the `addOrthoslice()` method. Alternatively, `addVoltex()` or `addMesh()` could be used to display the image as a volume or isosurface rendering, respectively.

Line 7-9 Assuming that there exists an external method `createVertices()` that returns a list of points describing the vertices of the surface, and that three consecutive vertices define a triangle, these lines show how to make a custom triangle mesh and add it to the scene.

The result looks similar to figure 4.7, which shows a confocal image of a fly brain together with parts of the surface of the medulla and the lobula.

Documentation in the form of source code examples is available online at `http://3dviewer.neurofly.de`, in the *Developer HowTos* category. The documentation demonstrates in a tutorial style the available functionality of our framework.

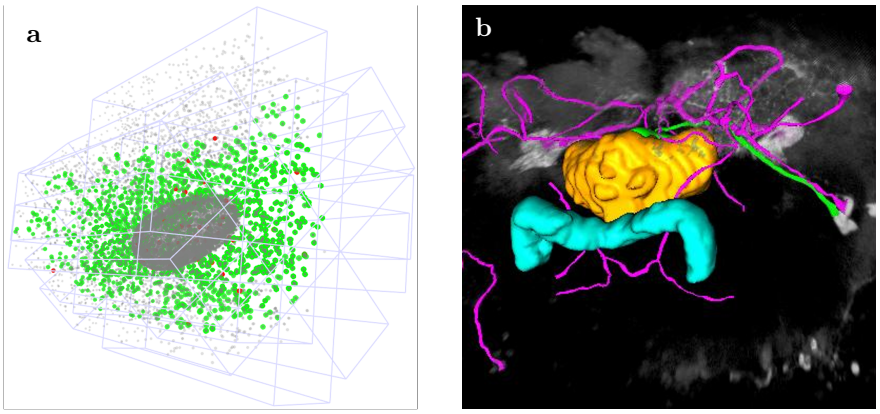

Figure 3.8: Third-party applications using our framework for 3D visualisation. (a) Visualisation of bead-based registration of the different views obtained by SPIM, rendering the beads as point meshes. (b) Snapshot of the Simple Neurite Tracer application, featuring the central compartments of the *Drosophila* brain. The intensity image is displayed as a volume rendering (grey), the protocerebral bridge and the fan-shaped body are shown as surface renderings (cyan & yellow), and the produced traces are displayed as custom meshes (magenta & green).

3.8 Application in third-party software

Numerous ImageJ-based applications currently use our 3D visualisation library. I briefly discuss three key applications below, illustrating the breadth of functionality we provide.

The Simple Neurite Tracer [33, 34] is an ImageJ plugin for semi-automated tracing of neurons in 3D image data. The application provides semiautomatic segmentation of filament-like structures such as neural arborisations and blood vessels. A starting point is chosen and then the filament is auto-traced up to a desired end point. The traced 3D path is visualised using components of our framework (fig. 3.8b). This example demonstrates how an analytical tool for measuring complex 3D structures can be augmented with 3D visualisation capabilities to display those objects.

An algorithm has been developed for registering images of a 3D sample, where each image volume represents a different angle of view obtained by Single Plane Illumination Microscopy [31]. The implementation of this complex algorithm required the 3D visualisation of intermediate and final image registration steps. Our library enabled the algorithm developers to generate the required visualisations with very little effort (fig. 3.8a).

TrakEM2 is an ImageJ plugin for visualisation, analysis, segmentation, reconstruction and registration of very large 3D image data sets obtained by serial section electron microscopy [35, 36]. TrakEM2 makes extensive usage of our framework for interaction with the 3D representation of image volumes and segmented objects of interest. The development of our library empowered

TrakEM2 developers to plan and design for 3D interactive features that would not have been possible otherwise. Reciprocally, the high-performance requirements of TrakEM2 drove implementation of parallel processing strategies for isosurface extraction and mesh composition in the 3D scene.

The interaction of our library with other software packages, each with specific requirements, promotes the development of new features and improves performance. These improvements then propagate back and enhance other ImageJ applications.

Semi-automatic segmentation

In chapter 2 I have introduced the VIB Protocol. As shown there, its alignment strategy is based on the manually obtained segmentations of specific neuropils. Manual segmentation constitutes not only a time-consuming and tedious task that discourages many potential users from applying the VIB Protocol, it is also highly subjective. Since the borders between neuropils are sometimes only weakly defined, it is difficult to obtain reproducible results in repeated segmentation rounds, even if performed by the same operator. To circumvent these shortcomings, segmentation needs to be automated, at least partially.

Automatic image segmentation is one of the oldest problems in image processing, and remains still subject of current research, due to the variety of image modalities. Segmentation methods can rarely be transferred from one image type to another, and the emergence of new imaging methods continuously requires further development and adjustment of existing algorithms. Nevertheless, several approaches for fully-automated image segmentation have proven successful in the past. They can roughly be divided into traditional and model-based methods.

Representatives for the traditional approach to image segmentation are global and local thresholding [37, 38, 39], region growing [40], the Watershed algorithm [41] and edge based methods (such as the Sobel, Canny and Laplacian filters, see for example [42]). Many techniques of this category were applied and tested comprehensively with confocal images of the *Drosophila* brain [43]. In summary, these methods are based on local image information and therefore susceptible to noise and boundary discontinuities; they fail if the boundaries of structures are indistinct or disconnected.

The family of deformable models belongs to the second, model-based category. The most famous representative of this family is the so-called "snake", introduced by Kass *et al* [44]. Their applicability to biological and medical images was soon recognised, after which they were extensively adapted and extended to many segmentation tasks in this area. A full survey of deformable

models is beyond the scope of this thesis, but the fundamental ideas and developments are introduced in section 4.3.1.

In this chapter, I combine the traditional and the model-based approach to develop a new semi-automatic procedure for the segmentation of confocal images of fly brains. In section 4.1 I introduce a new edge-based method for interactive surface extraction. This new technique results in incomplete surfaces. To obtain complete segmentations, template surfaces are utilised, the generation of which I describe in section 4.2. Together with the partial surfaces, they are used in section 4.3 to initialise an active surface model that allows to extract reliably the surfaces of the central brain and the neuropils in the optic lobes. In chapter 5, these surfaces are used to apply a non-rigid registration method.

4.1 Automatic surface extension

4.1.1 Procedure overview

The semi-automatic segmentation approach described below utilises user-provided seed points and image gradients to implement an automatically extending surface. The surface is generated as a triangle mesh and aims to resemble the surface of the anatomical structure to be segmented. The investigated image is loaded into the 3D viewer (using the 3D visualisation methods described in chapter 3) and rendered as a set of orthoslices. This allows the user to conveniently start the segmentation by selecting a point on the shape boundary of interest. The algorithm generates a first triangle that forms the initial mesh. This mesh is then extended automatically, assuming that the boundary of the extracted shape does not bend abruptly. Surface extension is accomplished by repeatedly selecting a point beyond the surface boundary and incorporating it into the existing surface. More precisely, a point outside the surface, but in the tangent plane of one of its previous boundary points, is constructed (fig. 4.1). The new point is moved along the image gradient at the new position, until the absolute value of the gradient is maximised. New triangles are then constructed from the new point and incorporated into the existing surface. Details about the construction of the new point are described below. The user controls conveniently the extent to which the surface grows via the scroll wheel of the mouse.

4.1.2 Implementation

Selection of edges to extend
The existing surface is iteratively extended by connecting a new point to existing border edges. Each iteration starts with selecting the edges that become connected to the new point. This comprises four steps (fig. 4.2): 1) From the existing surface, the border point with the smallest angle enclosed by the two adjacent edges is selected. 2) All edges within a certain distance are collected.

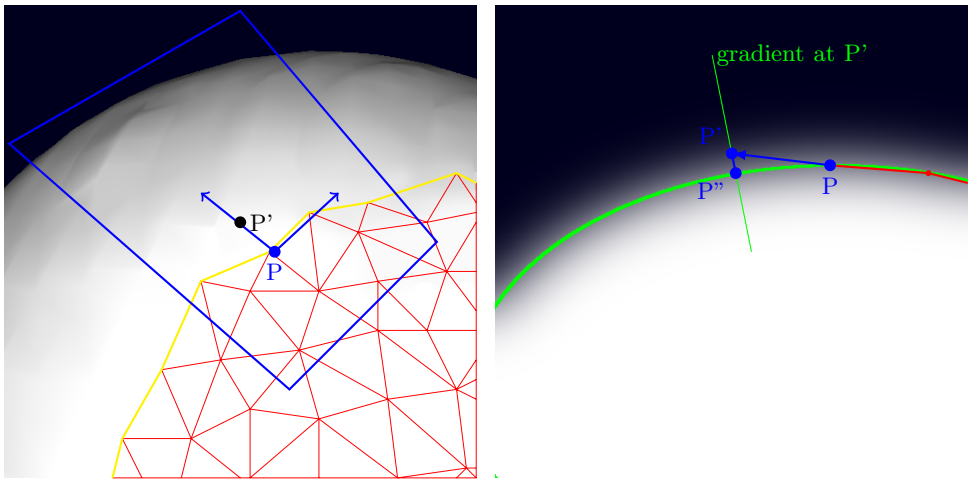

Figure 4.1: Principle of automatic surface extension. The given part of a surface can be extended by adding a new point (P') that is in the tangent plane of a point on the old surface boundary (P) and is only a small distance apart from the surface border. The new point is subsequently moved along the image gradient to the position (P") where the gradient is maximal (indicated by the green contour).

3) From this set of edges, iteratively the furthermost is removed, until all remaining edges form valid triangles with the new point. A triangle is considered valid if there is no obtuse angle. 4) Some stop criteria (to be described later) are evaluated to prevent the surface from growing in wrong directions, before the triangles are finally incorporated into the existing surface.

Construction of a new surface point

After identifying the next edge(s) to extend, the new surface point is calculated as follows (fig. 4.3): First, the third point of the triangle containing the edge is projected onto the plane that is perpendicular to the edge and intersects its midpoint. The projected point is mirrored at the image gradient at the edge midpoint. Finally, this point is moved along the image gradient at the new position, until its maximum is reached.

The image gradient at a position $p = (p_x, p_y, p_z)$ is a vector pointing into the direction where the intensity in the image volume changes most. Its length is proportional to the intensity change. The vector is calculated using finite differences: $grad(p) = (dx\ dy\ dz)^T$, where $dx = (\, I(p_x + 1, p_y, p_z) - I(p_x - 1, p_y, p_z)\,) \,/\, 2$ and $I(p)$ is the intensity at position p. In case of an anisotropically calibrated image, it is important to multiply the denominator with the voxel size.

If there is more than one edge to extend, the above described procedure is repeated for each edge, and the final position of the new point is determined by averaging the positions of the single candidates.

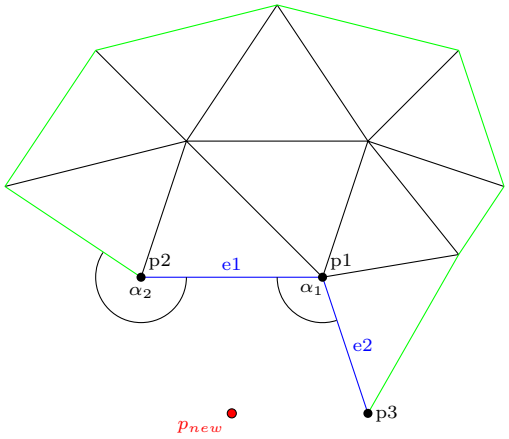

Figure 4.2: Edge selection for automatic surface extension. In each iteration, the border point with the smallest angle (enclosed by the adjacent edges) is selected (p1). From the edges in its neighbourhood, those are chosen (e1 and e2) which could be combined to new triangles by a single new point (p_{new}). Several criteria must be fulfilled before the existing surface is indeed extended by the new triangle candidates.

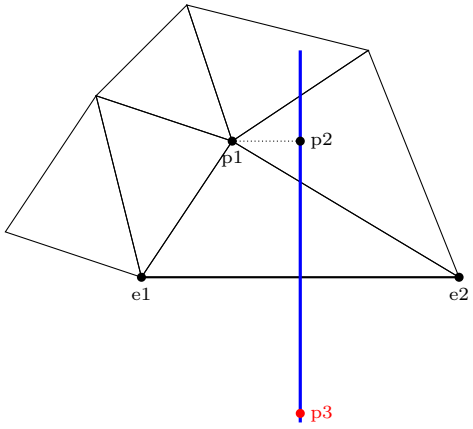

Figure 4.3: Construction of a new surface point. The third point (p1) of the triangle containing the edge is first projected onto a plane (blue) that is perpendicular to the edge and passes through its midpoint (p2). This point is then mirrored at the image normal at the midpoint of the edge, resulting in p3. In a final step, which is not shown in this figure, p3 is moved along the image gradient (at p3) until it reaches the maximum of the gradient.

Stop criteria
To avoid that the surface grows in the wrong direction, several criteria are elaborated that must be fulfilled by the new triangle candidates:

- **Directions of the surface normals**
  The surface normals of the new triangle candidates are compared to the surface normals of the triangles that are extended. If the angle between both exceeds a certain threshold (chosen to be 45°), the triangles are not added, and the edge to be extended is marked as not-extendable.
- **Direction of the image gradient**
  The direction of the image gradient at the new point is compared to the direction of the image gradient at the midpoint of the extended edge. If the angle between both vectors exceeds a certain threshold (chosen to be 45°), the new triangles are also not added, and again, the edge to be extended is marked as not-extendable.
- **Triangle area**
  To avoid the generation of strangely shaped triangles, it is assured that the areas of the new candidates are significantly greater than zero. If there is a triangle for which this is not the case, the edge is again not extended and instead marked as not-extendable.

Different cases of constructing a new triangle
Four different cases can be distinguished for constructing a new triangle from an existing edge and a new point (fig. 4.4).

- The new point is not part of the old surface.
- The new point is part of the old surface, and one of the two edge points is already connected to it.
- The new point is part of the old surface, and both edge points are already connected to it. This is the case when the new triangle closes a hole in the old surface.
- The new point is part of the old surface, but none of the two edge points is connected to it. This last case occurs when different parts of the surface are about to grow together.

Avoiding back-growing
Without precaution, an acute angle at the surface boundary causes the evolving surface to grow back onto itself, as it is demonstrated by the sequence of steps shown in figure 4.5. To avoid this, the edges are chosen in the order of ascending border angle for extension, as mentioned earlier. Additionally, it is checked if the angle in the current iteration is acute; if this is the case, no new point is added, but the neighbour edges are connected directly. The correct order is shown in figure 4.6.

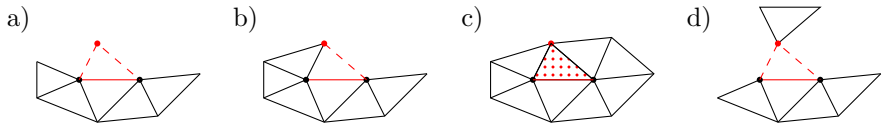

Figure 4.4: The four different cases to construct a new triangle from an existing edge and a new point. The edge and the new point are highlighted in red. a) and b) show the two obvious cases. In c), the new triangle closes a hole in the old surface. Case d) occurs when different parts of the surface are joining.

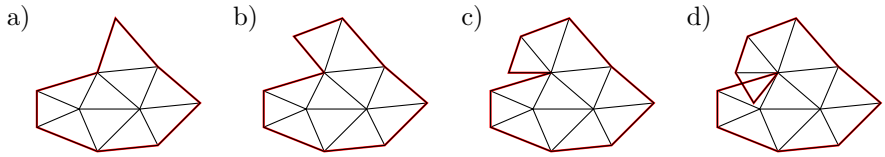

Figure 4.5: Sequence of steps that cause the surface to "grow back" onto itself. This occurs when boundary edges enclose an acute angle (shown in b and c). To avoid it, the two edges adjacent to the acute angle are connected without creating a new point (fig. 4.6).

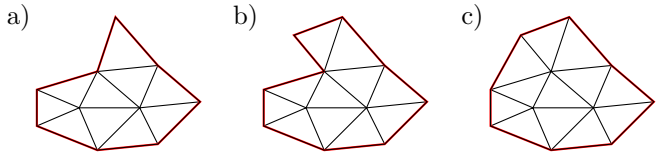

Figure 4.6: Avoiding "back-growing" by not creating a new point in c), where an acute angle occurs at a boundary point, but by connecting the two adjacent edges to form a new triangle.

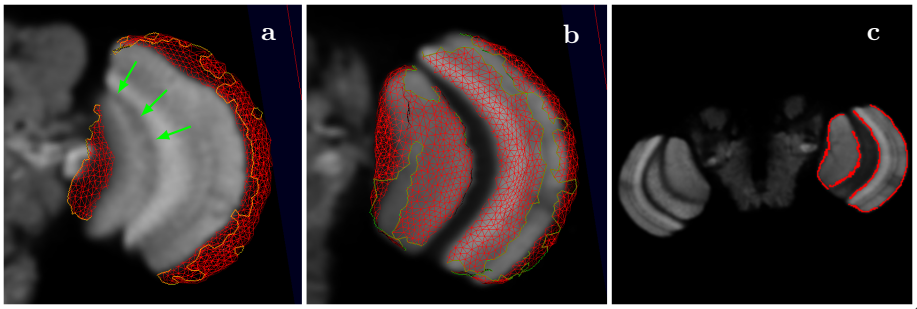

Figure 4.7: Results of the growing surface algorithm, using a confocal stack of a fly brain. (b) and (c): Successful segmentations of large parts of the medulla and lobula. In (b) the three-dimensional triangle mesh is visualised, while in (c), the mesh is drawn into a single slice of the volumetric image. (a) The automatically growing surface may follow wrong edges where the image gradient is diffuse. The desired path for the surface is indicated by the green arrows. To correct this, the user has to stop the segmentation early and select another seed point, for instance at the inner surface of the medulla.

4.1.3 Results

The method described in this section is tested with the confocal images of fly brains. Figure 4.7 shows the segmentation of the right lobula and medulla. Large parts of the neuropils in the optic lobes (medulla, lobula and lobula plate) can be segmented with the growing surface method. It is also possible to segment the central brain as one single structure. More difficult is the segmentation of neuropils within the central brain. Experiments show that it is still possible to segment parts of the antennal lobes and the fan-shaped body. In contrast, segmentation of the mushroom bodies is nearly impossible: Only in some images, parts of the calyx can be reconstructed, in others, the intensity in the corresponding image regions is too low. The peduncle is typically too thin: Surface extension as described above often yields here points that are already far from its surface. At the regions of the mushroom body lobes, finally, the images contain numerous other edges which distract the segmentation.

The tests also reveal that the surface sometimes grows in the wrong direction, despite the precautions described above. An example is shown in figure 4.7a: In the stainings investigated in this work, the border between medulla and lobula often gets diffuse in the anterior part of both neuropils. This causes the evolving surface in the figure to follow the border of the whole optic lobe instead of growing into the cleft between medulla and lobula. To address this problem, the growing surface algorithm is implemented as a semi-automatic method. The user controls to which extent the surface is expanded. He can go back in the segmentation, removing the latest added triangles. Furthermore, he can select a different seed point, upon which the surface starts to grow from the new position. Surface parts resulting from distinct seed points will grow together in the course of surface expansion.

It turns out that the stop criteria explained earlier usually cause the segmentations to remain incomplete. This means that it is not only possible that they contain holes, but also that larger parts of the surface are missing. How these partial surfaces can be completed using template surfaces is the topic of the next sections.

4.2 Generating template surfaces

In the previous section, a method was introduced that allows to segment many anatomical structures by automatically extending a surface from a user-selected seed point. The resulting surfaces however remain incomplete. For a human expert the missing parts of these surfaces are often intuitively imaginable, due to our knowledge about the shapes that are to be segmented. Transferring this idea to a computational method, information about the shapes must be incorporated. This is accomplished by template surfaces. In this section I describe how the template surfaces were generated from existing volumetric labelfields. After this, in section 4.3, I will show how they are incorporated to complete the partial segmentations obtained in the previous section.

The template surfaces were generated using the labelled standard brain. In particular, the segmentation editor (see p. 18) was used to obtain a labelfield for the template brain. Labelfields are images that store at each voxel position the class, here the neuropil, to which the voxel belongs. Typically, they are represented by 8-bit image volumes with an assigned lookup table. The lookup table associates with each index a neuropil, displayed in a different colour.

After obtaining the template labelfield, the surface of each neuropil is extracted separately. For this purpose, a binary (black and white) image is generated for each neuropil. This binary image is subsequently smoothed using Gaussian blurring. The marching cubes algorithm is then used to extract an isosurface, using an isovalue of 127 (which is in the middle of the intensity range, 0 - 255). For more detail on the marching cubes technique, see section 3.4.2 (p. 30). The resulting surface typically consists of an enormous amount of triangles, making it impractical for further analysis. Therefore, we implemented a mesh decimation algorithm that iteratively contracts edges and thus effectively reduces the number of triangles. The implemented method is highly inspired by the work published in [30].

Intuitively, contracting an edge does not introduce much deformation to a surface if the neighbourhood of the edge is flat. The higher the curvature around the edge, the higher is also the introduced deformation. Therefore, a cost is assigned to each edge that quantifies this deformation (fig. 4.8). To calculate the cost for a particular edge, the triangles adjacent to it are identified. For all these triangles the surface normals are calculated. Intuitively, the more the triangle normals are preserved by the contraction, the lower is the cost. Therefore, the edge is virtually contracted and the surface normals are re-calculated. For each triangle, the angle between the normal vectors before and after the contraction is then calculated. The angles are summed up for all triangles, yielding a final

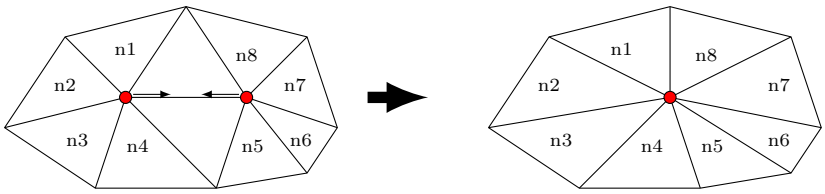

Figure 4.8: Principle of edge contraction. Contracting one edge reduces the number of triangles effectively by two. For each neighbouring triangle (n1...n8), the triangle normal vector before and after the contraction is calculated. The angle between both is summed up for all triangles, yielding a measure for the shape deformation introduced by the contraction.

cost value that measures the error introduced by contracting this particular edge. A priority queue is used to store the edges, the priority being given by the reciprocal cost value. Iteratively, the edge with the lowest cost is then removed from the queue and contracted, upon which the costs of the neighbouring edges must be updated.

After resampling the template image to a resolution of 256x256x57, the marching cubes algorithm yielded a surface with over 50,000 vertices. Using edge contraction, the number of vertices was iteratively reduced to 30,000, 20,000, 10,000 and 5,000 (figure 4.9). The surface with 20,000 was found to provide a good compromise between surface complexity and required level of detail.

4.3 Incorporating template surfaces using a new active surface model

The template surfaces from the previous section are used to complete the partial surfaces obtained from the growing surface method described in section 4.1. This is mainly accomplished by a modified version of an active surface model. Active surfaces are surfaces that are placed with an initial shape in a volumetric image and can then evolve to adapt to salient image features such as edges, following physical rules. Their performance depends mainly on the quality of the initial shape. Below, I first review the most important developments in the history of deformable models in biomedical image processing. Next, I describe how template surfaces and partially segmented surfaces can be combined to provide excellent initial shapes for active surface models. This technique is subsequently applied to the confocal images to obtain robust segmentations of the central brain and the neuropils of the optic lobes.

4.3.1 Active contours in the literature

Active contours, also called snakes, for semi-automatic image segmentation are introduced by Kass [44]. They belong to the family of energy minimisation

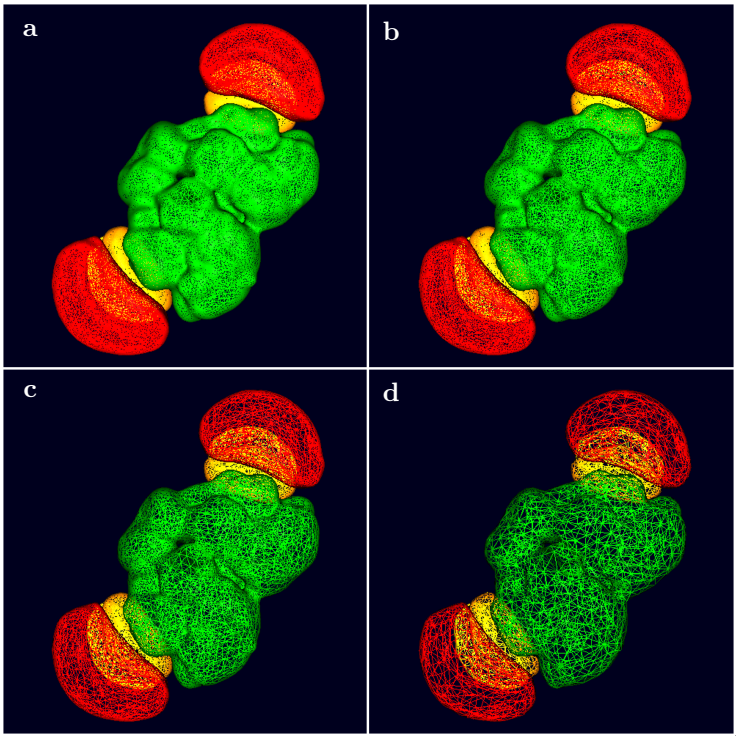

Figure 4.9: Results of iterative edge contraction. The initial surface obtained by the marching cubes algorithm contained over 50,000 vertices. Edge contraction was used to reduce the number of vertices to (a) 30,000, (b) 20,000, (c) 10,000 and (d) 5,000.

methods that aims to find optimal outlines for salient structures in the image. Intuitively, a snake is a contour in an image which develops from its initial shape to its final form by minimising its intrinsic energy. In particular, the energy is given by

$$E_{snake} = \int_0^1 \underbrace{\alpha(s)|v_s(s)|^2 + \beta(s)|v_{ss}(s)|^2}_{internal energy} - \underbrace{|\nabla I(x(s), y(s))|^2}_{external energy} \, ds$$

The integral along the contour consists of both external and internal energy terms. The internal terms reflect the physical properties of the snake: The first-order term, controlled by the parameter α, makes the snake behave like a membrane, while the second-order term (controlled by β) makes it behave like a thin plate. The external energy term is a potential function defined on the image which attracts the snake towards image features, like edges.

Kass uses calculus of variations to derive the Euler-Lagrange equations which are satisfied when the energy is minimal, i.e. the forces are in equilibrium. In this way, the final contour of the snake is obtained.

The advantage of snakes, compared to traditional methods like thresholding, region growing, edge detection etc, is their inherent continuity and smoothness, which can compensate for gaps and other boundary irregularities in noisy images. In the basic problem of 2D image segmentation, the user typically provides an initial contour which roughly resembles the structure to be segmented. The contour is then allowed to evolve, after which it can be fine-tuned manually.

Cohen *et al* realise that the snake in its original formulation shrinks in the absence of image features (based on the first-order smoothing term) and finally vanishes [45]. They tackle this problem by introducing an inflation force which counteracts the shrinkage and makes the contour expand. The initial contour needs then to be chosen inside the structure of interest, and expands, driven by the newly invented force, beyond spurious edges to match the desired image features.

Another problem with the original snake model is the difficulty to attract towards boundary concavities. Xu *et al* demonstrate this by investigating the original external, image-based force field. Based on their findings, they develop a more appropriate force field using gradient vector flow. The resulting snake models are therefore called GVF deformable models [46, 47].

Cohen *et al* extend the active contour concept to $2\frac{1}{2}$D [48]. $2\frac{1}{2}$D emphasises that the algorithm still works two-dimensionally, despite being applied to three-dimensional images or video sequences. In this approach, the user provides an initial contour for the first image slice. The snake is then optimised in this slice. Subsequently the resulting contour is propagated to the next slice, where it serves as a new initial contour. As an obvious advantage, the user only needs to provide the initial contour for one frame.

Miller *et al* introduce the first step towards a real 3D deformable model by suggesting a polygonal sphere as an initial surface [49]. Driven by an expansion

force, it undergoes geometric deformations to conform to the shape of interest. Active contours in three dimensions are often called active surfaces.

More work on active surfaces is published by Cohen *et al* [45, 48] who use the finite element method to parametrise the surface and formulate the update equations. Due to the parametrisation, less surface points need to be used, which accelerates the convergence of the model.

Originating the computer vision research, the applicability of deformable models in medical image processing was discovered soon. Deformable models were extensively modified and applied to innumerable image modalities such as X-ray, computer tomography, magnetic resonance imaging and ultrasound. Next to segmentation, they were also employed for motion tracking, e.g. of the left ventricle of the heart [50]. All in all, they were used for the segmentation of the brain, heart, face, cerebral coronary, kidney, lungs, liver and skull.

In this work, active surfaces are used as part of a pipeline for segmenting confocal stacks of adult *Drosophila* brains. Below I will first show the overall work flow for incorporating template surfaces. This process consists of several steps that gradually improve the position and shape of the template to resemble the neuropil surface of interest. A key role plays a modified variant of an active surface model with an additional regularisation term counteracting non-rigid shape deformation.

4.3.2 Work flow

Incorporating template surfaces to complete the partial segmentations obtained from section 4.1 comprises several steps which integrate four kinds of information: the template surfaces themselves, the template greyscale image, the partially segmented surface and the greyscale image that is currently segmented. The template surfaces are supposed to be segmentations of the template greyscale image. In the case of working with fly brain images, the *Drosophila* standard brain may serve as a greyscale reference image. The three steps to align template surfaces are listed here and described in more detail below:

1. Global rigid alignment of the greyscale images. The resulting transformation is used as an initial rough transformation of the template surface.

2. Alignment of the template surface to the partially segmented surface, using the Iterative Closest Point (ICP) algorithm. The ICP algorithm is initialised using the result of step 1.

3. Free-form refinement of the incorporated template surface using a modified version of an active surface model. The initial shape of the active surface is given by the result of step 2.

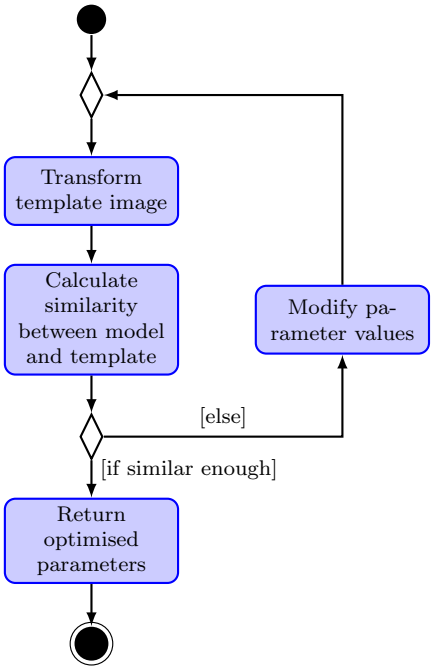

Figure 4.10: Intensity-based rigid registration of the template image to the model image. An iterative optimisation algorithm (Conjugate Direction Search) is used to adjust the parameters of a rigid transformation, to minimise the difference between model and template image. The six parameters of a rigid transformation consist of three rotational and three translational parameters. Image difference is calculated as the Euclidean distance, i.e. the square root of summed squares of pixel differences. The resulting transformation is used to transform the template surface so that it roughly fits the corresponding model structure. The transformed template surface is subsequently subject to further adjustment.

Global rigid registration of the greyscale images

To obtain an initial guess for the position of the template surface within the image to be segmented (in the following called the model image), the template image is registered roughly to the model image. This is done fully automatically, only based on the intensity images. Registration is based on a multivariate optimisation algorithm, in particular Conjugate Direction Search [6]. It iteratively adjusts the six parameters of the rigid transformation (three translational and three rotational parameters) to minimise the difference between model and template image. The difference between both images is calculated using the Euclidean distance (corresponding to the square root of summed squares of voxel differences). The algorithm is illustrated in figure 4.10. The optimised transformation is applied to the template surface, so that it roughly matches the corresponding structure in the model image.

Alignment of the template surface to the partially segmented surface, using the Iterative Closest Point algorithm.
The Iterative Closest Point algorithm (ICP, [51]) is used to refine the alignment between the template surface and the partial model surface. It iteratively identifies for each template surface point a corresponding point on the partial model surface (via nearest neighbour search). The obtained point correspondences are then used to estimate a rigid transformation in terms of minimum least squares [5], which is subsequently applied to the template surface points. The whole procedure is repeated until the template surface does not change any more.

Note: In contrast to the common situation, it is here the template surface that is aligned to the model surface, because the model surface needs to be completed using information from the template.

Free-form refinement using a modified active surface model
Typically, after steps 1 and 2, the template surface fits very well to the corresponding anatomical structure in the model image. However, both methods transform the incorporated surface only rigidly, not altering its shape at all. To account for shape differences, the incorporated surface must be allowed to freely adopt (to a certain extent) to the model image. For this purpose, a new active surface model has been developed which is introduced below.

4.3.3 A modified version of an active surface model

In the original active contour model, prior shape knowledge can only be incorporated by specifying the initial shape of the surface which is then changed in the course of optimisation. However, in biomedical images in general, and in the confocal images of fly brains in particular, the shapes of anatomical structures differ only slightly between specimen. It is therefore desirable to preserve the shape of the evolving surface to an adjustable extent. This limits the search space for the final shape of the surface and therefore reduces the risk for the evolving model of getting attracted to wrong image features. The model I propose here exchanges the second order regularisation term of the original snake model by a rigidity force which counteracts shape deformations of the evolving surface. Figure 4.11 illustrates some of the forces acting on a vertex.

Initialisation of the active surface

For the initialisation of the active surface both the partial segmentation (i.e. the model surface) and the template surface, aligned by the ICP algorithm, are used: The latter provides a topologically correct surface, while the former usually offers better fitting vertex coordinates.

Starting point of a combined surface is therefore the template surface, because it provides a valid connectivity between vertices. For each vertex v_m of

the model surface the nearest neighbour v_t in the template surface is then identified. If not only v_t is the nearest template vertex to v_m, but also v_m is the nearest model vertex to v_t, the corresponding vertex coordinate of the combined surface is set to v_m.

This procedure is particularly successful if the model surface is restricted to a subset of the template vertices, and does not contain surface parts of different anatomical structures.

Calculation of the force terms

The different force terms are calculated as follows:

- The image force F_{img}
  For each vertex v, p is calculated as the position of the maximum gradient along the vertex normal at v. The direction of F_{img} is then given by the difference $p - v$, and its absolute value is given by the absolute value of the image gradient at p.

- The smoothness force F_{smooth}
  For each vertex v, m is calculated as the mean position of the neighbour vertices of v. Then the direction of the smoothness force F_{smooth} is given by the difference $n - v$, and its absolute value is given by a user-adjusted smoothness weight.

- The rigidity force F_{rig}
  To calculate the rigidity force, the original vertex positions V_o are stored before running the model. From the current vertex positions V and the original vertex positions V_o, the best rigid transformation t can be calculated [5]. For each vertex v, the direction of the rigidity force F_{rig} is then given by the difference $(v - t(v_o))$, and its absolute value is given by a user-adjusted rigidity weight.

- The expansion force F_{exp}
  The direction of the expansion force F_{exp} of a vertex v is simply given by the outward pointing vertex normal at v. The vertex normal is calculated by averaging the surface normals of the surrounding triangles of v. The absolute value of F_{exp} is given by a user-adjusted expansion weight.

Iteration

In each iteration, the sum of the four forces is calculated for all vertices and their positions are updated accordingly. Subsequently, the vertex normals are recalculated. Iteration is performed until convergence, or until the user interrupts early.

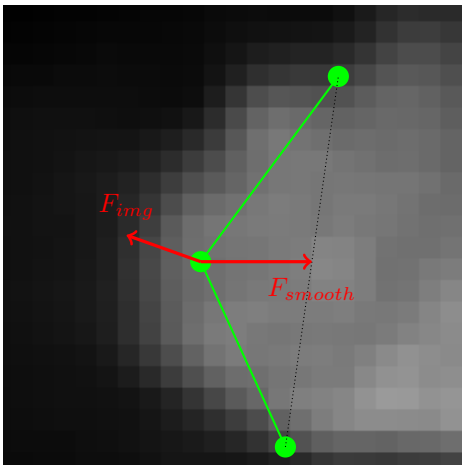

Figure 4.11: Forces of an active surface model, acting on each vertex: The image force F_{img} drags a vertex to local gradient maxima, the smoothness force F_{smooth} drags it to the mean of the neighbour points. The rigidity force F_{rig} (counteracting shape deformation) and the expansion force F_{exp} are not shown here.

The active surface model is implemented as an interactive method: The user can interrupt the iteration at any time, change the parameters, in particular the weights for the different force terms, and resume iteration. The display of the evolving surface is updated continuously, providing visual feedback to the user.

4.3.4 Results

Figure 4.12 shows the result of incorporating the template surface of the medulla. The three rows in the figure correspond to the three steps of the process. Step one, the intensity-based image alignment, yields a rigid transformation which is applied to the template medulla surface. The surface is shown together with the model image in the top row of figure 4.12. The figure demonstrates clearly that this first step allows to estimate the position of the medulla within the model image only very inaccurately. However, it provides a first guess which can be refined in the subsequent steps.

Having identified the rough position of the template surface in the model image, the alignment is refined in step two via the ICP algorithm. The middle row of figure 4.12 shows the template medulla surface, after aligning it to the partial (red) model surface. The template surface resembles the medulla in the model image already quite well. However, only rigid transformations are applied to the template surface so far, which do not account for shape differences across specimen.

For this purpose, the template surface can evolve non-rigidly in the third step, using the active surface model. After convergence, the resulting surface

resembles the medulla in the model image almost perfectly (bottom row in figure 4.12).

The performance of active surface models depends mainly on their initialisation, in this case on the results of the growing surface algorithm. Thus, active surfaces can successfully be applied to neuropils that can at least partially be segmented by the growing surface method. As mentioned in section 4.1, these are the medulla, lobula, lobula plate and the central brain. Additionally, the antennal lobe and the fan-shaped body can be segmented, although with more problems. Both neuropils exhibit partially well-defined borders, which allow to define a well-aligned initial guess for the active surface model. In regions with salient image features the evolving surface gets attracted to them, caused by the predominance of the image force in these regions. In regions with blurred edges, the smoothness and rigidity forces predominate and guide the adjustment of the surface. Segmentation of the mushroom bodies was not possible, since already the growing surface method failed here. Please note that the active surface model may very well be capable of extracting them, but a more appropriate method is needed to initialise it.

Together with the growing surface method of section 4.1, the three steps introduced in this section to incorporate template surfaces allow to quickly and reliably segment the medulla, lobula and lobula plates of confocal stacks of adult fly brains. Also, the central brain as one single structure can easily be segmented via these steps. A segmentation of these seven shapes is shown in figure 4.13. In the next chapter, the obtained surfaces are used for automatic, non-rigid registration of the images.

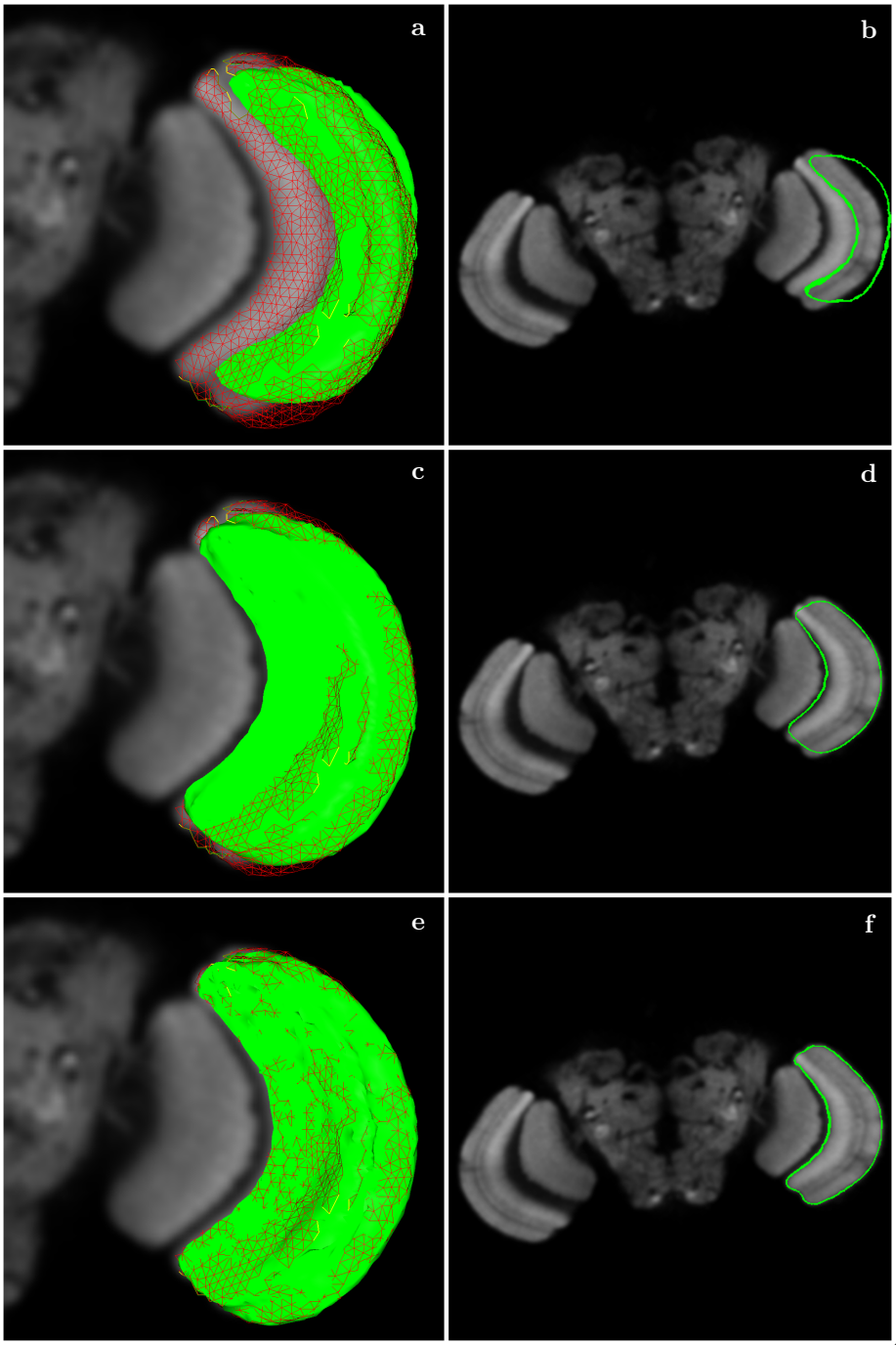

Figure 4.12: Alignment of the template surface of the medulla. The left column shows three-dimensional views, the right one shows the meshes drawn into a single slice of the image stack. The partially segmented model surface is depicted in red, while the aligned template surface is green. (a) and (b): The template surface after transforming it by the rigid transformation that is obtained by rigidly aligning the template intensity image to the model image. (c) and (d): The template surface after aligning it to the available parts of the model surface, using the ICP algorithm. (e) and (f): The template surface after several iterations of the active surface model.

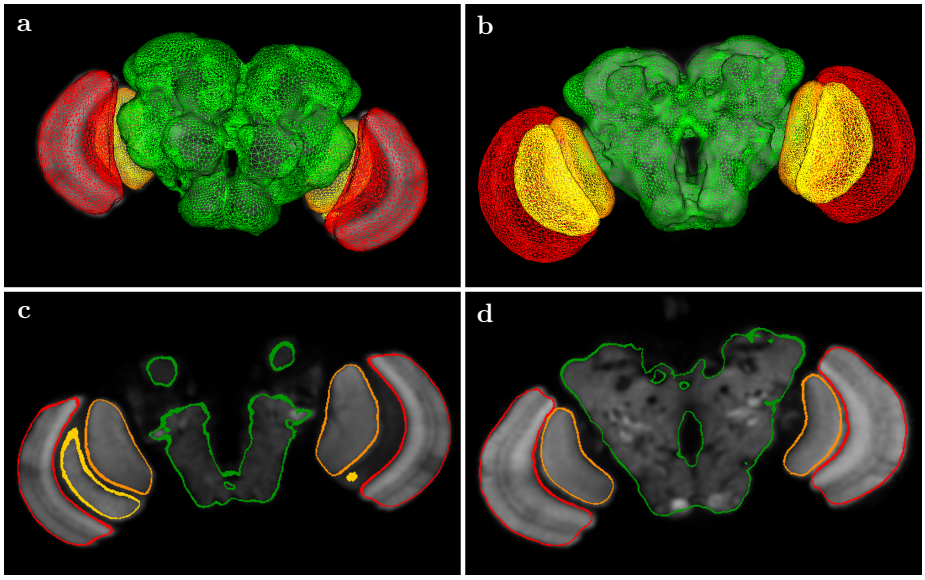

Figure 4.13: Results of incorporating template surfaces. Segmentation of medulla, lobula, lobula plate and central brain can be performed efficiently and reliably, using the growing surface method and aligning template surfaces.

5 Registration based on surface points

5.1 Background

Image registration methods can be subdivided into two major categories. Intensity-driven methods usually define a similarity measure between images based on the intensity values. Commonly used measures are the squared differences of pixel intensities, correlation and mutual information. Depending on the type of transformation (rigid, affine or free-form), the transformation parameters are adjusted iteratively to maximise the image similarity between model and template image, using an optimisation algorithm. In contrast, model-driven methods use corresponding anatomical elements in both model and template images to guide the volumetric transformation. The VIB Protocol is an example for the latter, using manually outlined neuropils to align the brain images. Other methods exist that use point or line correspondences. A comprehensive review about registration in biomedical image analysis can be found in [52]. This chapter focuses on landmark-based registration.

This chapter is organised as follows: First, I introduce some recently developed algorithms for automatically identifying point correspondences in image pairs. I apply these algorithms to the confocal stacks of adult fly brains, and demonstrate that they are not applicable to the problem at hand. In section 5.3 I reveal an alternative way to establish point correspondences across images: This approach uses the vertices of the surfaces that are obtained using the segmentation techniques of the previous chapter. Section 5.4 uses the point correspondences for landmark-based warping. In particular, a 3D variant of the thin-plate spline warping algorithm, introduced by Bookstein [53], is applied. Semi-automated surface extraction and automatic warping together offer a convenient alternative to the VIB Protocol, allowing to efficiently register confocal fly brain images. Both approaches are compared in detail in chapter 6.

5.2 Previous work about automatic feature detection

Recent work in the computer vision field addresses the problem of automatically identifying point correspondences in image pairs. Such correspondences are used for object recognition, object tracking, camera calibration, stereo matching and stitching. In this work, they should be used for landmark-based warping. The most important methods include SIFT (Scale Invariant Feature Transform, [54]), MOPS (Multi-Scale Oriented Patches, [55]), GLOH (Gradient Location and Orientation Histogram, [56]) and SURF (Speeded Up Robust Features, [57]). Although differing in the details, these techniques share the following three steps:

1. Identifying points of interest
The most important property of a point detector is repeatability, i.e. whether a given method finds the same location in corresponding images. Therefore, points are typically identified at corners, blobs and T-junctions in the image, where the curvature is high in each direction. The most famous point detection algorithm is probably the Harris corner detector, which evaluates the eigenvectors of the second-moment matrix at pixel locations [58]. Other methods include those based on the evaluation of the Hessian matrix, like the LoH (Laplacian of Hessian). They are based on the fact that the eigenvalues of the Hessian matrix are large at corner points. Yet another method is the LoG (Laplacian of Gaussian), and its approximation, the DoG (Difference-of-Gaussian).

2. Generating feature vectors
The local neighbourhood is used to calculate a feature vector for each point of interest. The purpose of the feature vector is to describe the local image structure for reliable matching of features across images. Various methods were proposed for calculating such feature vectors: SIFT uses for example a 128-bin histogram of local oriented gradients. MOPS uses image patches, oriented to the dominant local orientation, while the SURF descriptor describes a distribution of Haar-wavelet responses in the local neighbourhood of the point of interest.

3. Matching
Based on a difference measure between the feature vectors, e.g. the Euclidean distance, point correspondences are established across images. Typically, the Random Sample Consensus (RANSAC) algorithm is used to filter out outliers [59]. RANSAC selects iteratively a random subset of the previously established point correspondences as an initial set of inliers. A rigid transformation model is fitted to this initial set of inliers. The remaining point correspondences are added to the set of inliers if they support the model. The whole procedure is repeated a fixed number of times. At the end, the model with the least fitting error (regarding the extended set of inliers) is selected.

Automatic feature detection as described above, in particular the SIFT algorithm, was successfully applied to align and stitch images obtained by Electron Microscopy images [60, 61, 36].

Experiments

For this work a 3D version of a feature detection algorithm was implemented which combines the MOPS and the SIFT approaches; the first step, the location of interest points, follows the SIFT description in [54]: A scale-space pyramid is generated by iteratively blurring the input image volumes with a 3D Gaussian filter. At each pyramid level, a difference image with respect to the next level is calculated. Local extrema in the difference images represent potential interest point candidates, if they are also extrema across the scale dimension. For each candidate, the Hessian matrix is calculated. The eigenvalues of the Hessian are proportional to the principal curvatures at the corresponding position within the image; to avoid points on edges and ridges, candidates are neglected for which the ratio between the largest and the lowest eigenvalue is above a certain threshold (in [54] chosen to be 10). For the local descriptors within the image, we chose the MOPS descriptor, due to the complexity of 3D gradient histograms that would be required by SIFT. We applied our implementation to automatically identify landmark points in the confocal images of adult *Drosophila* brains. Corresponding landmarks across images should subsequently be used for landmark-based warping. Figure 5.1 shows the identified landmark sets of two images. The image patch describing the feature vector by one of the correspondences is depicted by the red box. The figure shows clearly that the point detector failed to identify reliably corresponding structures across brain images.

The first step in the whole process, the interest point detector, is already responsible for the failure, caused by the typical characteristics of the confocal images. The intention of the SIFT algorithm originally was the automatic recognition of known objects in photographs. These images are therefore in stark contrast to the confocal images at hand: The anatomical structures, which represent the common image entities on which registration should be based, do hardly show corners or blobs. However, the evaluation of the Hessian allows only for these features. The upper threshold on the ratio between the largest and the lowest eigenvalue guarantees a similarly high curvature in the direction of each eigenvector, thus by design avoiding the detection of edge-like features. Enlarging the threshold will in general enlarge the number of identified landmarks, but their location will be less robust. Landmark points that are nevertheless identified by our implementation originate mainly from noise in the images, introduced by the optic system of the microscope, or from distortions introduced during the preparation of the brains. The locations of such points are obviously not preserved across images and therefore not applicable for registration.

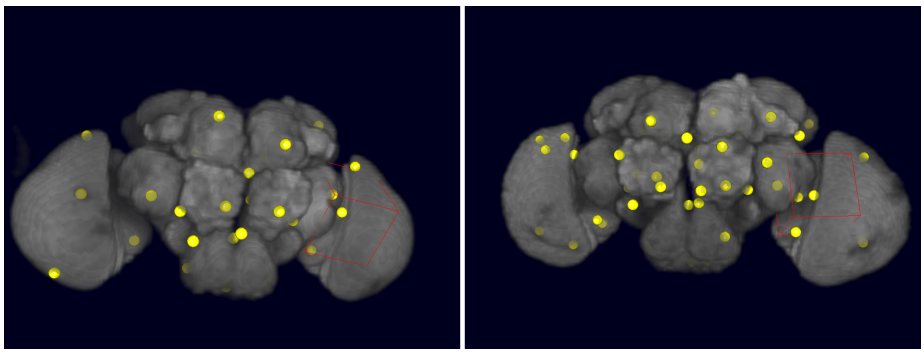

Figure 5.1: MOPS features of two confocal fly brain images. The identified points of interest are rendered as yellow spheres, the image patch describing one of the feature vectors is indicated by the red box.

5.3 Surface-based landmark points

As shown above, existing feature detection algorithms are not applicable for the images at hand. In contrast, a much simpler approach proves more successful: In the previous chapter, characteristic points in the images were identified as surface points at the boundary of the medulla, lobula, lobula plate and the central brain. These points can be used as landmarks, if correspondences can be established across images.

To infer such correspondences, the Iterative Closest Point (ICP) algorithm ([51], see also paragraph 4.3.2, p. 52) is applied to align the points of both images. After the alignment, points are considered to correspond to each other if their distance is minimal. Let for example *Set A* denote the set of points in image one and *Set B* the set of points in image two. If P_b is the closest point of *Set B* to a point P_a in *Set A*, and if P_a is also the closest point of *Set A* to P_b, then P_a and P_b are considered a corresponding point match.

Although the surfaces describe corresponding structures in both images, it is possible that the vertices have different positions on the surfaces. However, since the surfaces are represented by dense meshes, the introduced error is negligible, as will be shown later.

To obtain more robust results, the ICP algorithm is not applied to the entire surface, but to the surface of each of the described anatomical shapes (medulla, lobula, lobula plate and central brain) individually. This improves the resulting alignments significantly, and consequently allows to establish more reliable correspondences across images. This finding is explained by the loose connection between the optic lobes and the central brain in the fly brain preparations, which was already mentioned in previous chapters.

Typically, about ten thousand point correspondences are found for an image pair in this way. Filtering is necessary to reject wrong point matches and to speed up the subsequent warping. The remaining points should be spread evenly

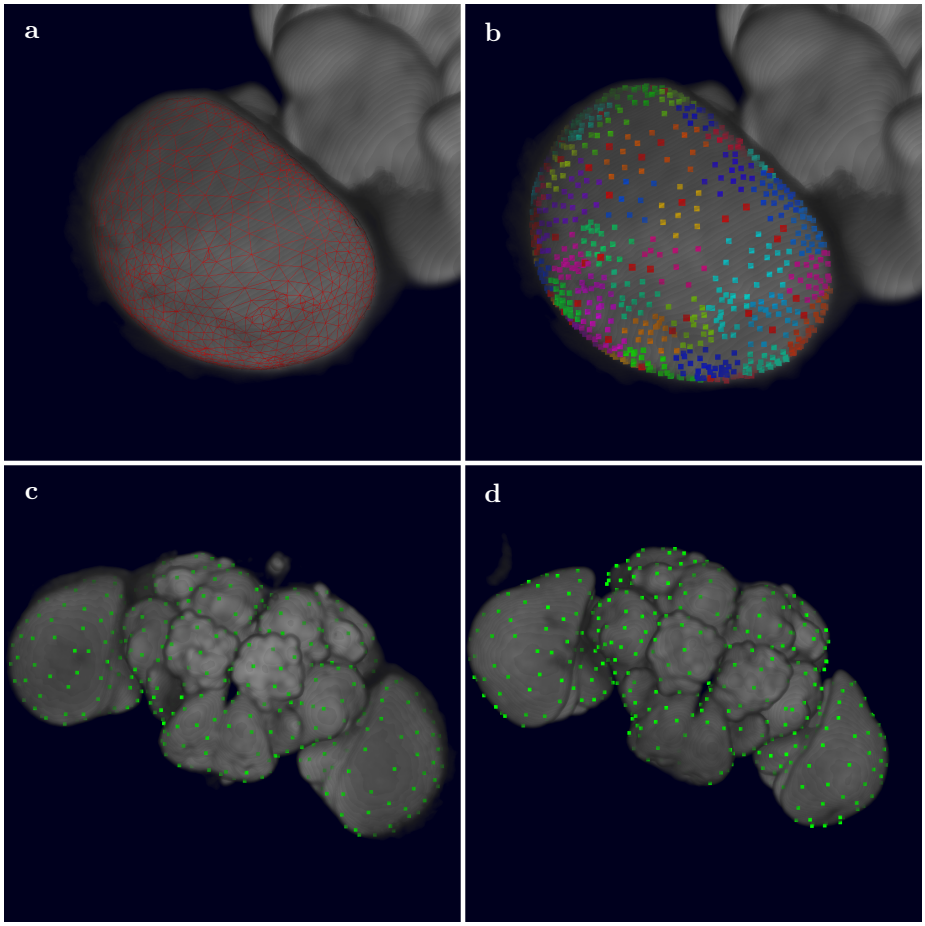

Figure 5.2: Establishing landmark correspondences using surface points. a) The surface of the medulla, as obtained by the segmentation techniques of the previous chapter. b) The entire set of medulla surface points are partitioned into 100 clusters. From each cluster, the best-matching point is selected for the subsequent warping. c) and d) Remaining point correspondences of two distinct brain images.

across the whole surfaces. To achieve this, k-means clustering is applied to the surface points. Like the ICP algorithm, this is done separately for each structure. K-means requires to specify the number of desired output clusters. For each structure, this number is adjusted according to the number of surface points of that structure, compared to number of surface points of the entire surface. The total number of clusters per image is set to 800. The top row of figure 5.2 shows the clustering of the surface points of the left medulla. The medulla contains about one-eighth of all the surface points. The number of medulla clusters is therefore set to 100 (= 800/8). In each cluster, one representative point correspondence is chosen for the subsequent warping: the one where both points in source and target image have the shortest distance. Remaining point matches after filtering are shown in the bottom row of figure 5.2 for two different brain images.

5.4 Warping

Landmark correspondences in two images can be used to elastically register a model image to a reference image. The most often used landmark-driven method for non-rigid image alignment is based on thin-plate splines [53]. A more recent approach is based on rigid moving least squares [62]. Both methods are briefly described below.

Landmark-based warping using thin-plate splines

In this approach, potentially irregularly spaced control point correspondences specify an interpolating deformation function. The interpolation function passes the control points exactly and interpolates between them such that the bending energy is minimised.

For this purpose, radial basis functions are defined at each control point. The basis functions take the form $U(r) = r^2\ log\ r^2$, where r is the distance to the corresponding control point. At each pixel location, the displacement is then given by a weighted sum of these basis functions:

$$f(x) = \sum_{i=0}^{n} w_i\, U(|x - c_i|)$$

where n is the number of control points, c_i denotes the ith control point and w_i is the associated weight. Given a set of corresponding control points, the weights can be determined using a least-squares approach. For this problem, a closed-form solution exists which is described in [53]. The name “thin plate spline” refers to a physical analogy involving the bending of a thin sheet of metal.

Landmark-based warping using moving least squares

In chapter 2, Horn's method [5] is used to infer a global rigid alignment across point correspondences. The optimal transformation is calculated by minimising a least-squares problem. The moving least squares approach extends this method: A different rigid transformation is calculated at each pixel location, by stating a similar least-squares problem, but weighting the control points, depending on their squared distance to the pixel: At pixel v, the transformation r_v is calculated that minimises

$$\sum_{i=0}^{n} w_i |r_v(p_i) - q_i|^2$$

where n is again the number of control points, p_i is the ith control point in the model image, q_i is the corresponding point in the reference image and w_i is the associated weight of the ith control point. The weights are proportional to the inverse square of the distance of the pixel v to the control point:

$$w_i = \frac{1}{|p_i - v|^2}$$

Effectively, each pixel gets transformed by a different rigid transformation. The weights change smoothly between neighbouring pixels, resulting in an equally smooth transition between neighbouring transformations. The restriction to rigid transformation accounts for particularly realistic warpings.

For 2D, an implementation is outlined in [62], which can easily be extended to 3D using Horn's algorithm with weighted control points. This is described in detail in [5].

5.5 Results

Both warping methods are implemented and applied to the fly brain images. Landmark point correspondences are established and filtered as described earlier. Experiments show that 800 landmarks in both images provide the best trade-off between required degree of freedom for the non-rigid transformation and undesired model complexity. Both methods are significantly accelerated by calculating the displacements only at certain grid points in the image and interpolate linearly in between. We use here a dense grid with a grid size of two pixels in each direction. Empirically, Bookstein's method using thin-plate splines is found to yield better results than the moving least squares approach.

The results are depicted in figure 5.3. The top row shows the original, untransformed brain images. The second row shows the transformed model image, after rigid registration (left) and after the above described warping procedure (right). The bottom row shows an overlay between the template image and the transformed model images. It is not surprising that the newly applied warping

procedure aligns the images better than the purely rigid registration. Nevertheless, the figure demonstrates clearly the accuracy of the alignment, and the smoothness of the warping; no artefacts are encountered.

A more comprehensive quality assessment is elaborated in the next chapter, which compares the results yielded by the methods of this and the previous chapter with those obtained by the VIB Protocol (chapter 2).

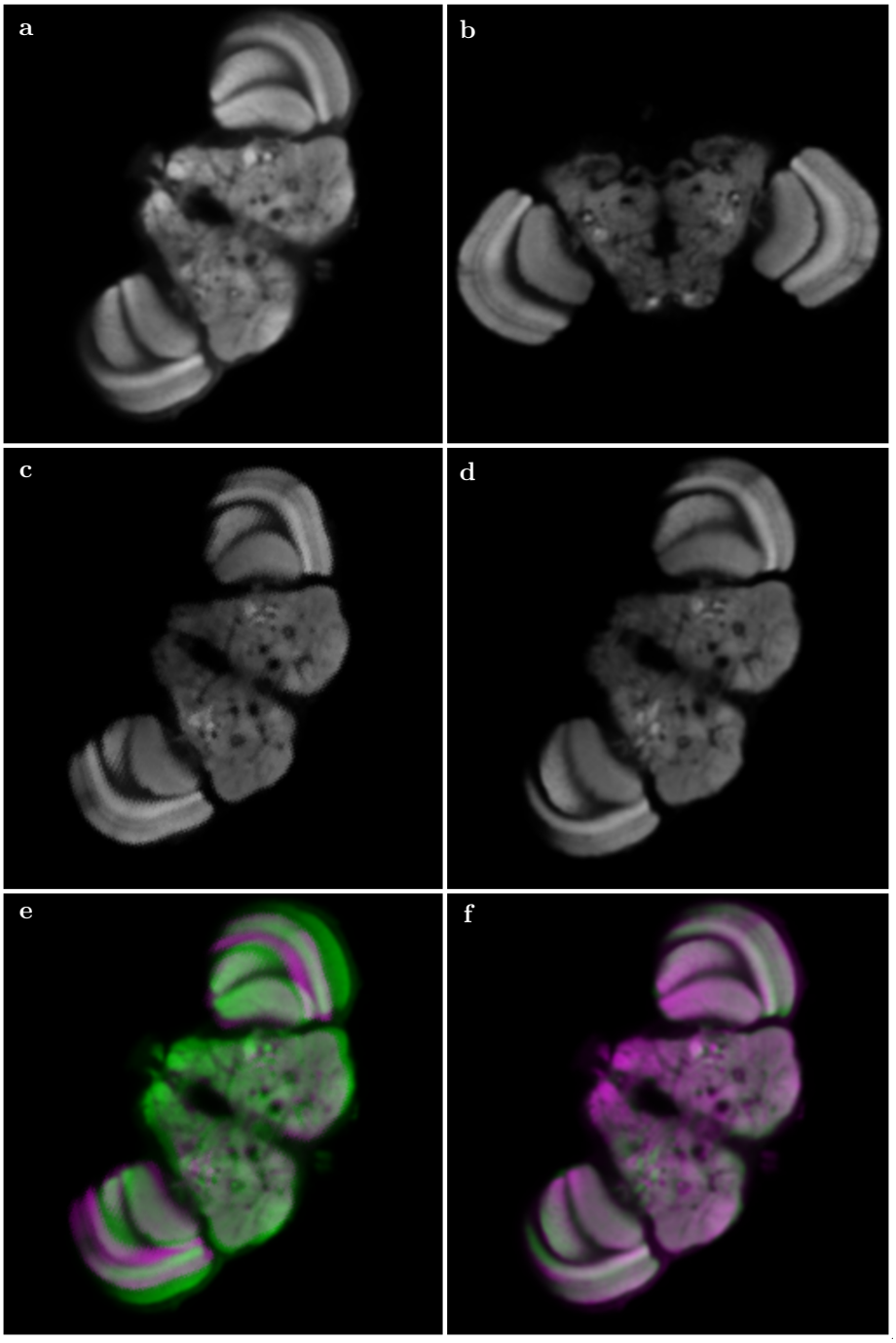

Figure 5.3: Comparison of landmark-based warping and rigid registration. a) The template image. b) The model image. c) The model image after rigid registration. d) The model image after warping. e) Merge of template and rigidly registered model image. f) Merge of template and warped model image. In e) and f), the template is depicted in green and the model in magenta.

6 Evaluation

As pointed out in chapter 2, working with the VIB Protocol is inconvenient due to the need of manually segmenting a set of neuropils. The methods described in chapters 4 and 5 together provide a powerful alternative for aligning fly brain images. In this chapter, both methods are compared with respect to the required user interaction and the quality of the resulting alignment. Additional tests are performed to investigate the reproducibility of the new approach.

6.1 Evaluation setup

For the evaluation of both methods, ten confocal images of adult fly brains are selected. All of them belong to the original data set that was used to calculate the *Drosophila standard brain* [1]. All ten images are segmented manually in advance, allowing to run the VIB Protocol in batch mode (see section 2.3). Next to the neuropils required for the VIB Protocol (medullas, lobulas, lobula plates, antennal lobes and mushroom bodies) six additional neuropils are segmented, which are later used to assess the quality of the alignment. Segmentations are stored as labelfields.

Registration is then performed for both methods, the VIB Protocol and the new approach, as described in previous chapters. For the VIB Protocol, only the aforementioned VIB-specific neuropils are used for registration. For both methods, the resulting transformations are applied to the original greyscale images and to the labelfields. The transformed labelfields are used to calculate the mean relative overlap per neuropil. This is described in detail below.

All given runtime measurements below indicate the mean time needed for processing one image. Images are downsampled to a resolution of 512x512x~100 before processing. The experiments are performed using a laptop with a 2GHz dual core processor.

6.2 Required time for running and user interaction

VIB Protocol
Three steps take up most of the time spent for the execution of the VIB Protocol: segmentation, local rigid registration and the interpolation of the displacement field (see chapter 2.1). The time spent for global rigid registration and image averaging are negligible. Depending on the experience of the user, segmenting an entire brain image requires about two hours. Aggravatingly, this time must be spent actively by the user, who is performing the segmentation manually. Under the mentioned conditions, local registration needs about 6 minutes per image, diffusion interpolation about 15 minutes. Both steps are less decisive since the VIB Protocol can be run in batch mode, allowing to execute these steps automatically, for example over night. The average time needed for one image sums up to about 2.5 hours.

New approach
The new approach is divided into two parts, segmentation and warping. Segmenting requires between ten and fifteen minutes. Warping is usually performed in two minutes, using the approach described in chapter 5. Hence, the overall process runs in about fifteen to twenty minutes, during which it requires active user interaction.

6.3 Quality of the alignment

As described in the evaluation setup, the resulting transformations of both approaches are applied to the greyscale images and the labelfields. From the transformed labelfields, the relative overlap of each neuropil can be calculated, providing a measure for the quality of the obtained alignments. It is derived for each neuropil separately. For this purpose, all ten aligned labelfields are transformed to binary images, having a value of 1 at pixel positions where the corresponding neuropil occurs and a value of 0 elsewhere. The pixel locations are extracted that are covered by the corresponding neuropil in any of the ten labelfields. A frequency histogram is then obtained over these pixels after summing up all of the ten binary images. The mean overlap is now calculated as the mean of this histogram.

Some slices of the sum of the binary images, combined for all neuropils, are depicted in figure 6.1. A colour lookup table is used to display the relative overlaps, ranging from blue (zero percent overlap) to red (one hundred percent overlap). The figure contrasts the result of the VIB Protocol (left column) with the result of the new approach (right column). Individual slices of the stack are displayed to emphasise the differences in individual neuropils.

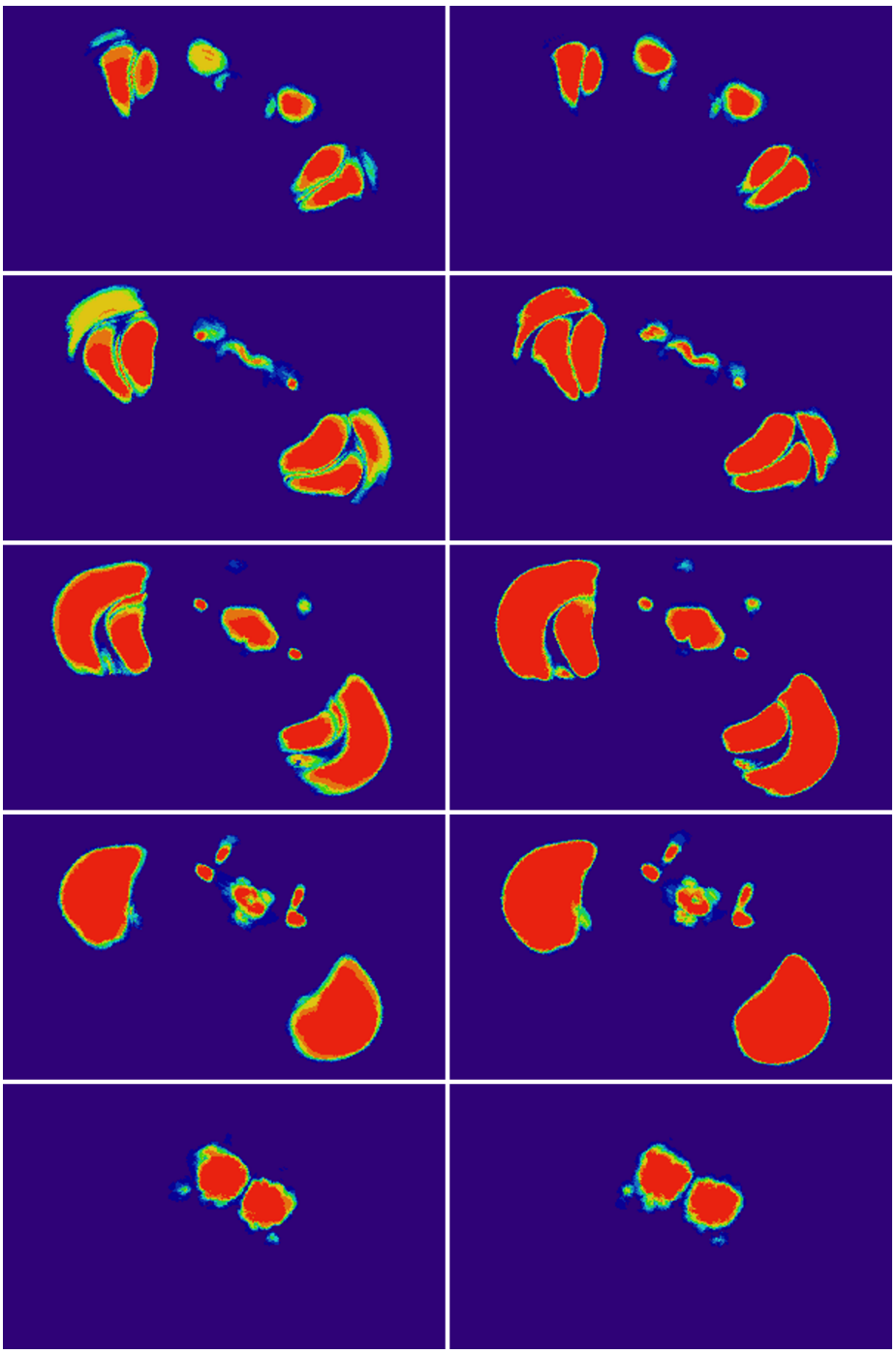

Figure 6.1: Comparison with the VIB Protocol. The left column shows the results of the VIB Protocol, the right one those of the new approach. The overlap of the individual neuropils is encoded by the colour, ranging from blue (0% overlap) to red (100% overlap). Special attention is paid for (from top to bottom): the mushroom body calyces, the protocerebral bridge, the fan-shaped body, the ellipsoid body and the antennal lobes.

	mean overlap [%]	
neuropil	using the VIB Protocol	using the new approach
right medulla	74.8	86.0
left medulla	73.4	85.6
right lobula	69.4	80.9
left lobula	71.9	81.8
right lobula plate	61.9	75.0
left lobula plate	61.3	76.3
right mushroom body	54.2	55.1
left mushroom body	54.7	58.1
ellipsoid body	50.4	51.3
noduli	35.5	45.8
fan-shaped body	56.7	60.4
protocerebral bridge	29.8	38.2
right antennal nerve	36.4	40.3
left antennal nerve	32.2	39.1
right antennal lobe	70.6	63.9
left antennal lobe	62.3	62.4

Table 6.1: Overlap of the transformed neuropils, expressed in percentages. The new approach yields better results for all neuropils except for the right antennal lobe.

The numeric values for the relative overlaps of the labelled neuropils are summarised in table 6.1. They show that the new approach outperforms the VIB protocol in all neuropils, except for the right antennal lobe, where the VIB Protocol yields a slightly better value.

6.4 Reproducibility

Additionally, the reproducibility of the new approach is investigated. For this purpose, a similar setup is chosen as described above for the comparison with the VIB Protocol. Instead of applying the new approach to ten different images, however, the process is repeated ten times using the same image. As a reference image, an external sample image is selected. Pre-segmented labelfields are again transformed together with the greyscale image. Aligned labelfields are utilised to calculate the mean overlap as described above. The result, shown in figure 6.2, indicates relative overlaps close to or above ninety percent.

neuropil	mean overlap [%]
right medulla	94.3
left medulla	94.6
right lobula	90.8
left lobula	92.3
right lobula plate	82.7
left lobula plate	85.1
right mushroom body	86.7
left mushroom body	88.5
ellipsoid body	88.8
noduli	91.3
fan-shaped body	91.8
protocerebral bridge	84.8
right antennal nerve	91.4
left antennal nerve	85.6
right antennal lobe	92.8
left antennal lobe	90.8

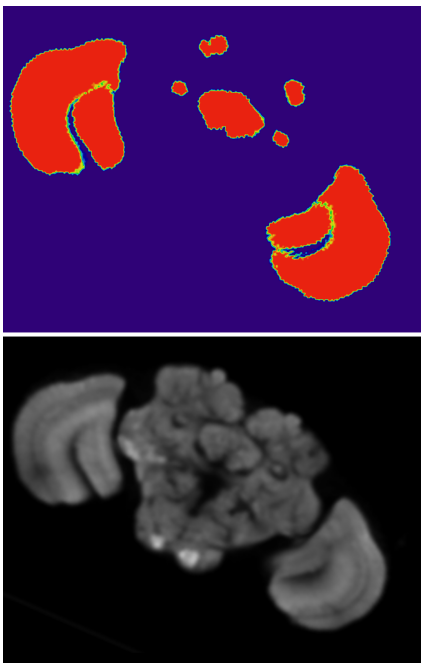

Figure 6.2: Relative overlaps for aligning the same image ten times independently to a reference image. As reference image, the same (external) image was used for all ten repeats.

6.5 Summary

Using the novel tools proposed in chapters 4 and 5, the time required for segmentation and registration of the confocal fly brain images can be reduced to one sixth of the time that is needed using the VIB Protocol. At the same time, the quality of the alignments and the reproducibility are enhanced.

7 Towards fully-automated registration of fly brain images

In chapter 4, a new tool for semi-automatic 3D image segmentation was introduced, which was then used in chapter 5 for automatically warping the images onto a template. While warping was performed automatically, segmentation required user interaction: The user had to choose seed points for the growing surface and guide the incorporation of template surfaces, in particular the parameters of the active surface model.

While testing the implementations of these methods, it is observed that a user tends to adopt his interactions with the software, the more images he processes. This holds true for the selection of initial seed points for the growing surface, where the user "learns" that certain points on the neuropil surfaces are better suited than others, and is observed again later when applying the active surface model: The very same parameters are applicable for a specific neuropil, even across images, while other neuropils require different settings. The longer the software is used, the more repeatable become the individual steps. This apparent consistency suggests to automate the entire process.

While the semi-automatic tool of chapters 4 and 5 can be used for many kinds of (3D) images, the automation of these steps is now targeted specifically to confocal images of *Drosophila* wholemount stainings and will probably not work with other image modalities.

This chapter is divided into two parts, the first one describing the automation of the surface growing, the second one describing the automation of incorporating template surfaces.

7.1 Automatic Surface Growing

In this section, a number of preprocessing steps are elaborated that allow to reliably identify seed points for the growing surface algorithm (described in section 4.1). Conforming to the previous chapters, the anatomical structures that are segmented are the medulla, lobula, lobula plate and the central brain. Therefore, points must be found that are located at the boundaries of these structures. Once such points are extracted, they are used for the growing surface algorithm. While the semi-automatic segmentation in chapter 4 allows the user to control the growing surface, e.g. via the mouse, it is applied in this chapter in an automatic manner: For each seed point, the surface is extended until the algorithm stops by itself.

Bilateral filtering

In the confocal images, the structures of interest (medulla, lobula, etc) are well-defined. By applying a 3D variant of a gradient filter, a 3D edge image is obtained in which the boundaries of the aforementioned structures appear as most prominent (i.e. brightest). However, many other contours remain there, too. To suppress them, Gaussian blurring could precede the gradient filter. A Gaussian filter, however, smooths desired edges together with darker, undesired ones. A more appropriate alternative is the bilateral filter [63]. Both filters, the Gaussian and the bilateral, replace each pixel with a weighted average of its neighbours. While for the Gaussian kernel, the weights depend only on the spatial distance to the kernel centre, the bilateral filter uses a distance measure in both spatial and intensity space. This effectively smooths regions with similar intensity values more than those with different ones. Consequently, salient contours are preserved, while less prominent ones are blurred. The result of the bilateral filter depends on the balance between the spatial and the intensity-based standard deviation of the Gaussian kernel. Figure 7.1 depicts the results for different intensity-based standard deviations, while keeping the spatial standard deviation fixed: With increasing intensity-based standard deviation, more and more salient edges are smoothed. For the fly images, reliable parameter settings for the purpose here are $\sigma_{spatial} = 5$ and $\sigma_{intensity} = 150$, applied after resampling the images to a size of 256 x 256 x ~90.

Identification of seed points

After resampling and bilateral filtering, a convolution with a 3D gradient filter is applied to identify edges in the volumetric image. To suppress weak edges, pixels in the edge image with an intensity value below 50 are set to zero. From the remaining pixels, local maxima are collected that can then be used for the surface growing algorithm. Figure 7.2 shows the results of the individual steps. Despite smoothing and thresholding, the resulting surface may still contain undesired surface parts. This does not matter as long as the subsequent incorporation of template surfaces can be performed reliably. Automation of aligning template surfaces is covered in the next section.

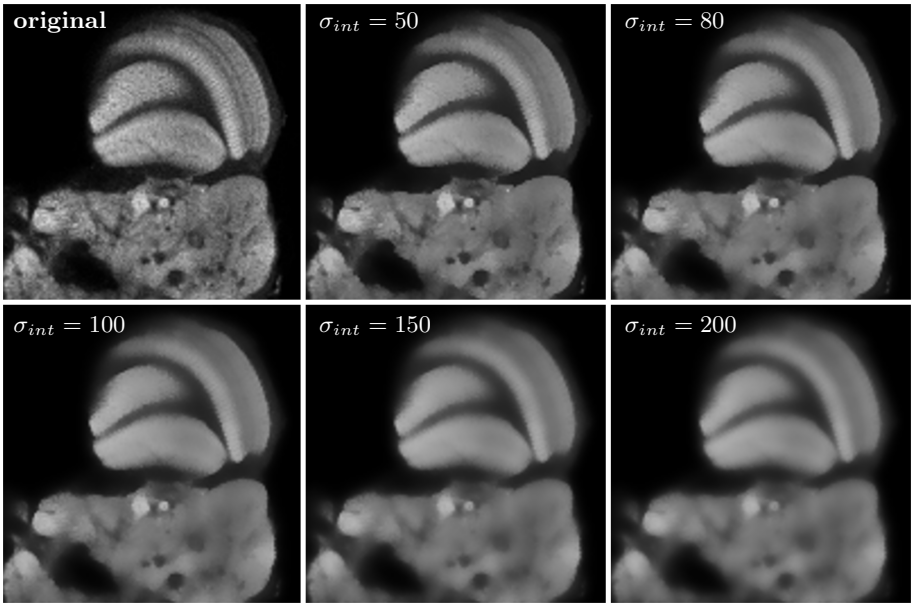

Figure 7.1: Bilateral filter with increasing intensity-based standard deviation. The original image is shown top left. The remaining images show the effect of the bilateral filter, applied to the original image, with increasing intensity-based standard deviation, ranging from $\sigma_{int} = 50$ to $\sigma_{int} = 200$. The spatial standard deviation of the Gaussian kernel is fixed in all images to $\sigma_{spatial} = 5$.

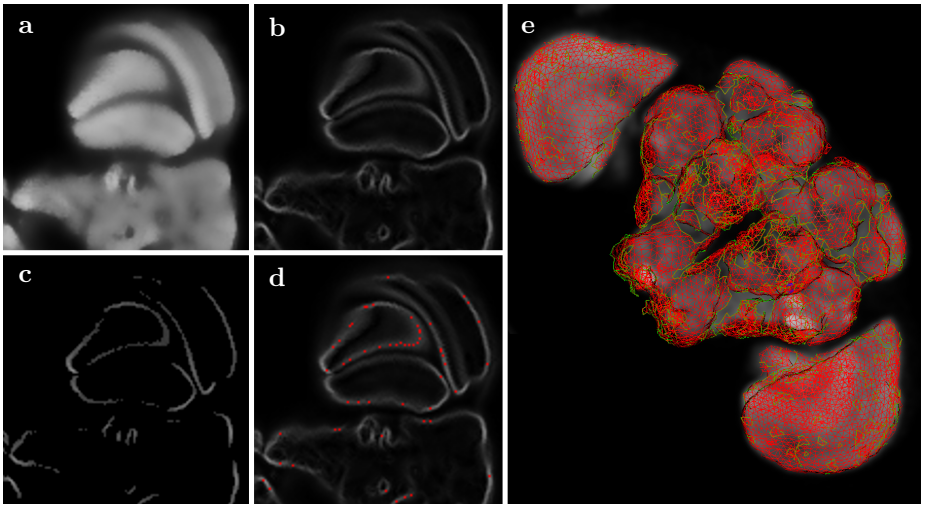

Figure 7.2: The four steps of fully-automatic surface extension. The original image is first downsampled and subsequently filtered with the bilateral filter (a). Then, a 3D gradient filter is applied (b). After suppressing pixels below a certain threshold (t=50, (c)), local maxima are identified (red, (d)). These maxima are used as seed points for the growing surface algorithm (e), which reconstructs the most salient surfaces in the images that are used to align the template surfaces.

7.2 Automatic incorporation of template surfaces

As described in section 4.3 (p. 47 ff), the incorporation of template surfaces comprises three steps: 1) a rough intensity-based rigid registration of the template to the model intensity image, providing an initial transformation for the template surface, 2) a subsequent rigid alignment of the corresponding template and model surfaces via the ICP algorithm and 3) a final refinement of the registration via an active surface model. Since each of the three steps relies on the results of the previous one, failure of one of them invalidates the result.

In the semi-automatic approach, each neuropil is segmented separately. The user controls the growing surface: When the template surface is aligned, the partial surface thus consists only of parts that correspond to the neuropil boundary which is currently segmented (see the red mesh in figure 7.3b). In contrast, in the full-automatic setup the growing surface algorithm is applied to the entire image before the individual template surfaces are aligned. Thus, the model surface consists no longer only of the surface parts corresponding to the template surface, but covers the entire brain structures. This impairs the alignment via the ICP algorithm severely, often resulting in misled convergence by wrong surface parts.

Figure 7.3 illustrates the problem: In a), the position of the right lobula plate is shown after step 1, the rough rigid registration. This is the basis for the ICP algorithm in the semi-automatic as well as in the fully-automatic setup. Figure 7.3b shows the convergence of the ICP, when the partial model surface is restricted to the shape under study: This is the case in the semi-automatic setup: The convergence is successful. In contrast, figure 7.3c depicts the situation for the fully-automatic setup, where the partial model surface is extracted for the entire image before incorporating individual template surfaces. The model surface comprises all parts of the brain, the convergence of the ICP is misled.

It was verified empirically that the problem described above occurs with the alignment of the lobulas and the lobula plates only; the central brain and both medullas do not show these difficulties and can be aligned without problems. Consequently, for both the lobulas and the lobula plates the ICP algorithm requires a more precise initialisation than the rough rigid registration. These observations lead to the following adopted procedure: First, the central brain and both medullas are aligned, using the previously described sequence of steps. The ICP algorithm yields rigid transformation matrices for the left and right medullas. These matrices are now used to initialise the ICP for the lobulas and lobula plates, instead of the matrix obtained by rough rigid registration. They provide a closer initialisation for the ICP algorithm, resulting in a successful convergence. This is depicted in figure 7.3d.

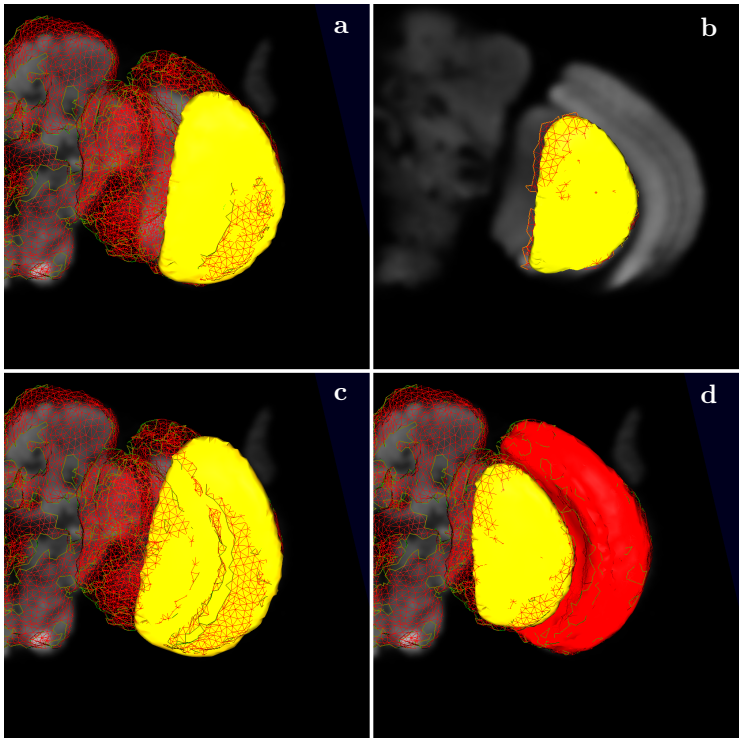

Figure 7.3: Automatic surface alignment for the lobula plate. (a) Position of the template surface of the lobula plate (yellow) after rough initial registration of the template and model image. This is the basis for the ICP algorithm, both in the full-automatic and the semi-automatic setup. (b) Result of the ICP algorithm in the semi-automatic setup: Because the model surface (red) is restricted to the boundary of the lobula plate, the ICP converges correctly. (c) Result of the ICP algorithm in the full-automatic setup: The model surface (red) resembles the boundaries of all the brain structures, distracting the convergence of the ICP algorithm to wrong surface parts. (d) Correct convergence in the full-automatic setup has proven reliable for both medullas. Their position can be used to initialise the ICP algorithm for the lobula and lobula plate more accurately, so that its convergence succeeds.

neuropil	F_{smooth}	F_{rig}	F_{exp}	neighbourhood	iterations
central brain	15.0	1.0	15.0	3	2000
right medulla	5.0	5.0	1.0	3	2000
left medulla	5.0	5.0	1.0	3	2000
right lobula	5.0	10.0	0.0	1	1000
	3.0	5.0	0.0	1	1000
left lobula	5.0	10.0	0.0	1	1000
	3.0	5.0	0.0	1	1000
right lobula plate	5.0	10.0	0.0	1	1000
	3.0	5.0	0.0	1	1000
left lobula plate	5.0	10.0	0.0	1	1000
	3.0	5.0	0.0	1	1000

Table 7.1: Optimal parameter settings for the active surface models for each neuropil. Two rows for a single neuropil indicate that two active surface models are applied consecutively with different parameter settings.

Automation of the active surface model

After a particular template surface is rigidly aligned to the model surface via the ICP algorithm, an active surface model is applied which adjusts the template surface to features of the model image. As described in section 4.3, the active surface model has a number of parameters that determine the physical properties of the evolving surface. In particular, these settings include the weights for the different force terms and the extent of the search space. It was found that the optimal settings for these parameters differ from structure to structure, but are consistent across images. For example, the template surface for the central brain converges faster for a relatively large expansion force, while the same value for the expansion force leads to a wrong convergence for the lobula plate. Optimal parameters were identified empirically. They are summarised in table 7.1.

7.3 Results

The repeatability of the individual steps for semi-automatically segmenting confocal images of adult fly brains allows to implement a fully-automatic procedure. This chapter describes the necessary steps and possible pitfalls. Full-automation is obtained by using the same methods that are applied in the semi-automatic setup. The latter provides the means for identifying necessary preprocessing steps and reliable parameters to automate both the surface growing and the incorporation of template surfaces. Together with the warping procedure of chapter 5, the work of this chapter provides a fully-automated method for registering the fly brain images. Tests were successfully performed with the images

that were already used to test the semi-automatic method in chapter 5. The obtained results were almost identical to the results shown there (figure 6.1, p. 71). Before fully establishing the automatic method, however, further adjustment of the parameters will be performed on a larger data set to yield maximum robustness.

8 Discussion

In this work, I have developed a flexible 3D visualisation proramming library. It provided the means for intuitive user interaction with three-dimensionally displayed images, based on which I then implemented a novel semi-automatic 3D segmentation method. This new tool was used to segment several anatomical structures in the confocal images of *Drosophila* brains, yielding surfaces that were subsequently used for the non-rigid alignment of the images. The proposed developments allow for quick and reliable image registration, and reduce required user interaction and time drastically compared to the previously employed VIB Protocol.

3D visualisation

In chapter 3, a 3D visualisation framework for the visualisation of biomedical images was presented. Numerous software tools were available previously, so one might ask if it was indeed worthwhile to implement yet another framework for this purpose. The following arguments reveal the contributions of this project.

High-level 3D programming library

This thesis aimed to ease segmentation and registration of confocal images of the *Drosophila* brain. The work presented in chapters 4 and 5 addresses these challenges and proposes a method for semi-automating both tasks. In general, semi-automatic methods require user interaction. For 3D image processing, it is essential to provide means for this via an interactive 3D interface. Existing 3D software is however mostly targeted to end-users, and provides limited to no access to its features programmatically. The development of new semi-automatic methods is not possible using these tools. In contrast, our framework offers a public Application Programming Interface (API), allowing custom applications to integrate 3D visualisation with a minimum of required effort.

Integration into ImageJ

The integration into ImageJ represents probably the biggest advantage our framework offers: ImageJ itself is a small program for image processing, which is easily extendable by virtue of its plugin-based structure and its powerful macro language. This property accounts for its broad acceptance in many fields, mainly however in biology and microscopy. Thousands of users have developed their own extensions, addressing their specific problems. Most of these extensions are available publicly. All these plugins can now benefit from easy-to-integrate 3D visualisation, and reciprocally, the functionality of our 3D viewer can be extended by accessing the features of available plugins. Hence, a whole collection of new possibilities arises by this combination, allowing for integrated solutions that exceed the functional scope of existing software. Recently, for example, an ImageJ extension was published for volume reconstruction of electron tomography data, called TomoJ [64]. Depending on 3D visualisation for displaying its results, it was in need of an external software. The authors recommended Chimera [65] to their users. With our proposed framework, they can now offer an integrated solution which is not provided so far in other software packages. To integrate into ImageJ, a usable 3D visualisation library needs to be written in the Java programming language, since ImageJ itself is written in Java. As of writing this thesis, there existed no other accelerated 3D visualisation library for Java.

Extension to 4D data

The advent of high-throughput microscopy has increased the number and size of biological image data sets in need of analysis. The acquisition of 4D data, such as from laser-scanning fluorescent microscopy of cells moving through space, has become commonplace. Interactive data analysis of 4D data sets for object motion tracking is in increasing demand. From the beginning, our software contained internally all necessary components for 4D rendering (i.e. 3D data over time). Recently, we succeeded to extend it to a real 4D visualisation library. While it was rudimentarily possible to display 4D data before (as described in chapter 3), the 4D design is now built-in inherently. 3D display is of course still possible, but in the present version it is just a special case for the general case of 4D rendering. Existing commercial packages like Imaris (BitPlane) provide 4D visualisation, too. However, the functionality they offer with respect to 4D analysis and segmentation is limited. As mentioned earlier, our framework is unrestricted in this regard, as it has access to numerous available plugins. With the recent development of a new n-dimensional data structure for ImageJ [66], 4D analysis will drastically improve even further. This new image container framework supports arbitrary image dimensions, but abstracts the dimension from algorithm development. For this purpose, it uses extensively new features of the Java programming language, so-called generics. With this development it will be possible to implement algorithms in a generic way and apply them to images of arbitrary types and dimensions.

Digital brain atlases

The segmentation and registration tools of chapters 4 and 5 were developed with the *Drosophila Standard Brain* in mind. However, brain atlases have already existed before for various species. In human medical research, they were established quite some time ago to compare brain anatomy across age, gender, time, health and disease and populations. Additionally, techniques were developed to integrate data of different imaging modalities, such as Computer Tomography (CT), Positron Emission Tomography (PET), Magnetic Resonance Imaging (MRI) and cryosection images in a common atlas. These techniques were applied to data obtained for both humans and primates. Comprehensive reviews about human brain atlases can be found in [67, 68].

In recent years, brain atlases for other species, particular for insects, have attracted increased attention. Here, they were first used for studies on structural differences. Rein et al have for example investigated dimorphism in the optic lobes of *Drosophila melanogaster*, using a standard brain atlas [69]. They found the size, shape and position of the optic lobes to be highly conserved between animals, but also revealed that the optic lobes of female flies were in average 6% larger than those of males. Dreyer *et al* provided a standard brain atlas for the red flour beetle *Tribolium castaneum* for studies on antennal lobe glomeruli, metamorphic development and sexual dimorphism [70]. El Jundi *et al* developed a brain atlas for the sphinx moth *Manduca sexta* to show that the conspicuous sexual dimorphism in the olfactory system of this species is only reflected in the shape of the antennal lobe glomeruli, and not in other brain neuropils involved more downstream in the olfactory pathway [71].

In addition to the investigations of structural dimorphism, brain atlases were used to study metamorphic development. Huetteroth *et al* combined 3D reconstruction and 4D visualisation techniques with a brain atlas of the moth *Manduca sexta* [72]. They monitored the development of selected neuropils by recording images of different animals during defined stages of pupal development. A very impressive video accompanies the mentioned publication as a supplement.

A third application of digital atlases of insect brains is the investigation of circuitry and connectivity. In the desert locust *Schistocerca gregaria*, a standard brain was developed and employed for anatomical studies on polarisation pathways in the central complex [73, 74]. In this study, the authors reveal a potential connection between a CPU1 neuron (a polarisation-sensitive columnar neuron) and a GFS neuron (a tangential neuron activated during flight), by registering single cell stainings of the two neurons to a standard brain and visualising them subsequently. Kvello *et al* developed a standard brain atlas for the moth *Heliothis virescens* [75]. For the investigation of the neural networks involved in chemosensory coding and learning, they recorded and stained single cells, in particular gustatory and olfactory interneurons and the axonal projections of gustatory receptor neurons. The individual images were registered and combined in the atlas and visualised, revealing their projections to the corresponding brain neuropils. In *Drosophila*, Jefferis *et al* combine single-cell

labelling with image registration techniques to obtain high-resolution, quantitative maps of the mushroom body and lateral horn for input projection and lateral horn neurons [76]. Other studies were performed also in honeybees (see for example [77]).

In each of the above mentioned projects, the number of images which needed to be registered to the atlas was well below one hundred. New connectivity studies in *Drosophila* however work on a larger scale. Their ambition is to obtain a full physical wiring diagram showing the information flow from sensory input to behaviour output of a complete *Drosophila* brain [78]. For thousands of neurons, single cell stainings must be recorded and registered to a connectivity atlas. For this huge number of individual images, efficient and accurate techniques are required for registration. In this work, I have presented novel tools for registration and segmentation, which address these demands. Both the semi-automatic and the fully automatic methods offer time-saving, robust and precise methods with the potential to process the accruing data. With the development of such an aforementioned single cell atlas, it will finally be possible to address questions like: How much is the position of cell bodies and axons preserved across individual animals, across different genetic strains, and during development? How do single neurons connect, and to which extent are these connections preserved, again with respect to developmental stages and genetic differences. With appropriate tools at hand, one can then establish a statistical map depicting likelihoods for the positions and shapes of single neurons, similar to the map for neuropils shown earlier in figure 6.1 (p. 71).

Applicability

Applicability to other insects

As mentioned above, brain atlases were already established for a number of different insects. The difficulties they face are identical to those which occur with the *Drosophila* atlas. Although not tested, we strongly believe that the methods introduced in this thesis can successfully be transferred to other insects. This assumption is based on the similarity across confocal brain images of different insects, in particular if the same antibody is used in the staining protocol. Figure 8.1 (adapted from the indicated publications) compares manually labelled neuropils from the honeybee [77], the desert locust [73], the red flour beetle [70] and the fruit fly and shows the similarity of individual neuropils in the four species.

Applicability to other species and imaging modalities

While the proposed steps for full automation (chapter 7) are tailored to the confocal images of fly brains, the semi-automatic setup is not restricted to this application. Generally, it can be applied to any three-dimensional data independent of the acquisition method, such as electron microscopy or magnetic resonance images. There are however several factors which either facilitate or

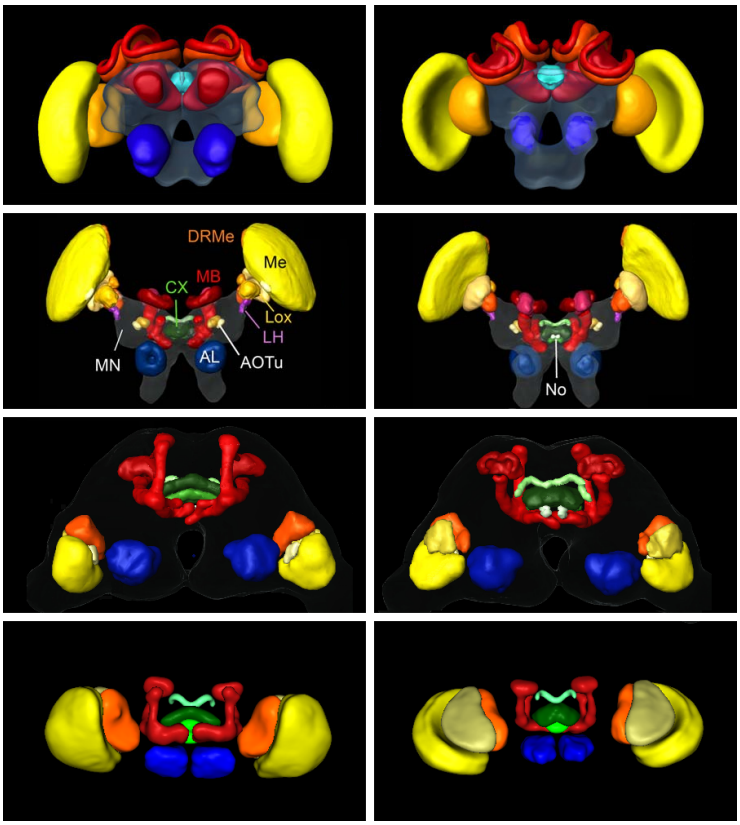

Figure 8.1: Comparison of the shape of manually labelled neuropils of the honeybee [77], the desert locust [73], the red flour beetle [70] and the fruit fly (from top to bottom). The similarity of the structures across different insect species accounts for the applicability of the proposed segmentation techniques to other insect brain images.

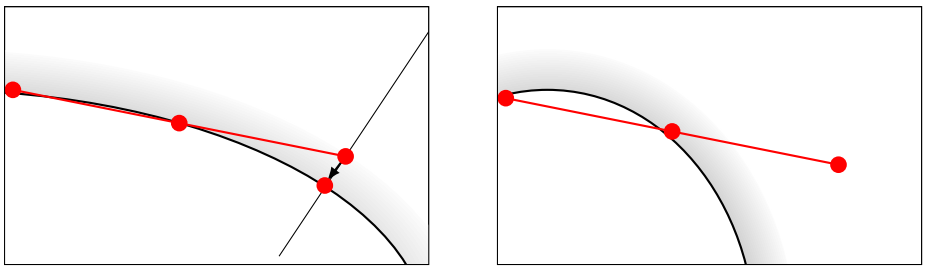

Figure 8.2: If the curvature of the shape boundary is small, the growing surface method succeeds (*left*). If it is high, the calculated surface point may be too far from the true shape boundary, so that subsequent optimisation along the image gradient fails (*right*). In this case, performance could be improved by decreasing the distance between the existing surface and the next surface point candidate.

impair its application. These restrictions are mainly imposed by the surface growing method (chapter 4).

Based on the assumption that the contours to be segmented bend smoothly, the surface growing method iteratively extends the existing surface with new points. As described in chapter 4, a new point candidate is first chosen a fixed distance beyond the existing surface boundary, simply following the tangent plane of the surface. The position of the new point is then optimised along the image gradient. There is no problem if the surface does not bend much. The more it bends, the more important becomes the distance between the existing surface and the point. If this distance is too long, the new point candidate is too far from the true shape surface, and subsequent optimisation along the image gradient fails. This is depicted in figure 8.2, for clearness shown in a 2D analogy.

For a successful segmentation, this distance must therefore be chosen as short as possible. It is however limited by the image resolution and cannot be below the size of a pixel. If the images at hand do not meet this requirement, a possible solution is consequently to scale the image up.

Even if the shapes bend abruptly, our method can still be applied. The growing surface method is not required to yield a full segmentation. In this case, template surfaces must be available. They are then aligned to the partial surface and subsequently adjusted to the image via the active surface approach described in this work.

Relation to existing software

Segmentation possibilities in existing software packages

Existing commercial software packages offer a variety of tools for segmentation, which are however often restricted to traditional approaches to image segmentation and rarely include model-based algorithms. Amira's segmentation tools

(Mercury Inc.) range from purely manual to fully automatic. They include a brush tool for manual painting, a lasso tool for semi-automatic contouring and a magic wand for region growing. Furthermore, they provide thresholding, intelligent scissors and active contours for contour fitting [79]. However, these tools are applicable only for individual 2D slices of an image volume, and not in higher dimensions. There exists a freely-available extension that implements an active surface model, which is unfortunately designed specifically for the segmentation of plant cells [80]. Imaris users (BitPlane) are provided with global and local thresholding, region growing and semi-automatic surface generation methods, the latter being based on a locally applied marching cubes algorithm with an interactively selectable iso-value. MeVisLab (MeVis) supports image segmentation via automatic region growing and a semi-automatic livewire tool. The applicability and shortcomings of all of these methods with respect to the fly brain images were investigated in [43]. The most comprehensive software package regarding segmentation and registration functionality is the open source library ITK (Insight Toolkit, [81]). Its numerous options include region growing, watershed segmentation and, as a powerful model-based approach, levelset segmentation. Levelsets for the segmentation of *Drosophila* brain images were also described in [43].

Novelty of the proposed methods

Although rarely included in today's available commercial packages, levelsets and active surface models have been studied intensively in the past. Our particular implementation is unique due to the combination of the proposed modified variant of an active surface model with the growing-surface method for its initialisation. Active surfaces have undergone an expansive development in the computer vision and image analysis research. They have proven particularly successful for the segmentation of anatomical images. Common problem is the extraction of a shape which is generally preserved, but slightly deformed across images. This is usually the case for anatomical structures in images originating from different individuals. Key feature of active models is their inherent smoothness which makes them insusceptible against noisy data. For a successful application, active surfaces require a close initialisation, i.e. an initial shape which is close to the true shape in the image under study. In practise, they are often embedded into hybrid methods, where they are combined with traditional segmentation techniques such as edge detection.

The approach proposed in this thesis can be seen as such a hybrid method. The active surface model is applied to align template surfaces to a model image, but to initialise the active surface model, another technique is utilised, here based on image gradients. This is the described surface growing algorithm. Together with the subsequent surface alignment via the ICP algorithm, the surface growing algorithm offers an excellent way to obtain a surface which is close to the true shape one wants to segment. Only this accuracy of the initial shape makes a successful convergence of the active surface model possible.

Consequently, the active surface model fails where the growing surface algorithm does not yield satisfying results. In the investigated fly brain images,

this is particularly the case for several neuropils within the central brain, for example the mushroom bodies. Chapter 4 explains in detail the reasons for this failure. Importantly, this does not mean that the active surface model is not capable of extracting the corresponding shapes in general. However, the surface growing method must be extended or replaced by a more appropriate initialisation method. Elaborating powerful alternatives is left to future research.

Software for image registration

The most considerable software package with regard to image registration was published recently under an open-source license: The Computational Morphometry Toolkit (CMTK) [82]. It offers fully automatic registration based on an algorithm published in [83, 84]. This algorithm models motion between images by a global affine transformation, followed by a free-form refinement. The latter is based on B-Splines, which specify the displacement at sparse control points and guarantee smooth warping. CMTK does not provide a graphical user interface, it is invoked from the command line and accepts various command line options for adjusting the parameters of the registration algorithm, including the grid size and regularisation options. The toolkit was successfully applied to *Drosophila* confocal brain images [76].

In contrast to manual segmentation, fully automatic techniques offer the greatest possible convenience because they require no operator interaction at all. Reciprocally however, there remains also no possibility for the user to intervene and thus to prevent erroneous interpretations of individual images for which the fully automatic procedure fails. This is not only a crucial aspect for critical applications which therefore often prefer semi-automatic methods. It plays also an important role for the images investigated in this work, considering how they were obtained: Before acquiring them via the confocal microscope, the scanned fly brains were dissected. The head is separated from the thorax and the cuticula is removed, using forceps. Obviously, these mechanical interventions easily deform, sometimes even damage the prepared brains, decreasing the quality of the resulting scans. Furthermore, the lack of visual feedback in fully automatic methods hampers the adjustment of algorithm parameters, rendering a successful application often difficult or impossible.

Semi-automatic methods provide a trade-off between the convenience of fully automatic setups and the fine-grained control of manual methods. While they drastically decrease labour intensiveness and enhance reproducibility, they still allow for interactive guidance by an expert. We have proposed a novel semi-automatic method in chapters 4 and 5. Appendix A demonstrates in detail the convenience with which segmentation and registration can be performed, while still retaining full control over the process. Additionally, however, we have elaborated an approach to fully automate the entire process in chapter 7. In particular, we have described a preprocessing pipeline which allowed to automatically identify locations in the images which served as seed points for the growing surface method. We have also elaborated a set of parameters for the active surface model which was used to fit template surfaces to the images. Preliminary results have turned out promising and suggest to follow up this

approach by optimising the proposed method further. We therefore provide two methods: Users can benefit from the convenience of full automation, but for individual images where this method delivers unusable results, our semi-automatic tool offers a fall-back to gain valuable results from these images, too.

Although designed for it, our software is not limited to image registration. In many other biomedical image processing tasks, segmentation is the required first step of a subsequent quantitative analysis, such as volume measurements of 3D structures or distance measurements of tracked cells. In the general case, template surfaces will not be available to support the segmentation. Rather, holes in the incomplete surfaces, as obtained by the proposed growing surface method, can be closed by connecting adjacent edges directly to yield a topologically correct surface. This surface can then be subjected to the active surface model for further improvement of the segmentation.

Future objectives

By now it is possible to segment the medulla, lobula, lobula plate and the central brain with the proposed methods. Segmentation of the neuropils within the central brain does not work reliably. For these neuropils, already the surface growing method does not yield satisfying results which could be used for an appropriate initialisation of the active surface model. Based on the surfaces of the extractable neuropils, however, it is possible to align the images to the template.

Instead of aligning the model image to the template, as it was described before, it is also possible to do the opposite: to transform the template image so that it fits the model. Together with the template image, the template surfaces can be warped. The warped template surfaces will be very close to their counterparts in the model image, close enough to initialise another active surface model. In this way, it should be possible to obtain a more complete segmentation of the neuropils within the central brain.

A Manual for semi-automatic registration

A.1 Starting the software

Typically, the users start with opening the image which they want to segment (again called "model" image for the rest of this chapter). After that, the plugin is launched from ImageJ's plugin menu. This opens a new 3D window, in which the model image is rendered in the orthoslice view.

The 3D view allows to rotate and shift the image, and users can scroll through the perpendicular slices, all of which is controlled conveniently by the mouse. Next to the 3D Viewer, a dialog opens and asks the user for a template image and the template surfaces (fig. A.1). This information is required for the alignment of template surfaces, as described earlier. The accepted format for surface files the OBJ file format (Wavefront Technologies). In case a user does not plan to align template surfaces, the dialog can be cancelled. The startup process will then continue normally, but template surfaces will not be available later.

A.2 Segmenting a structure

The growing surface method requires user-provided seed points. The orthoslices view offers an optimal way to select such points: In contrast to other rendering modalities, which show only the surface of the brains, it provides means to select arbitrary points, also inside the brains. A user just needs to click at an appropriate location while holding down the 'g' key. Initial points should be selected on the surface of the structure which one wants to extract. Once a new seed point is chosen, a first triangle is added which can subsequently be expanded. Expansion is controlled either by using the keyboard or the scroll wheel, and can be performed in different step sizes: Holding down the '0' and

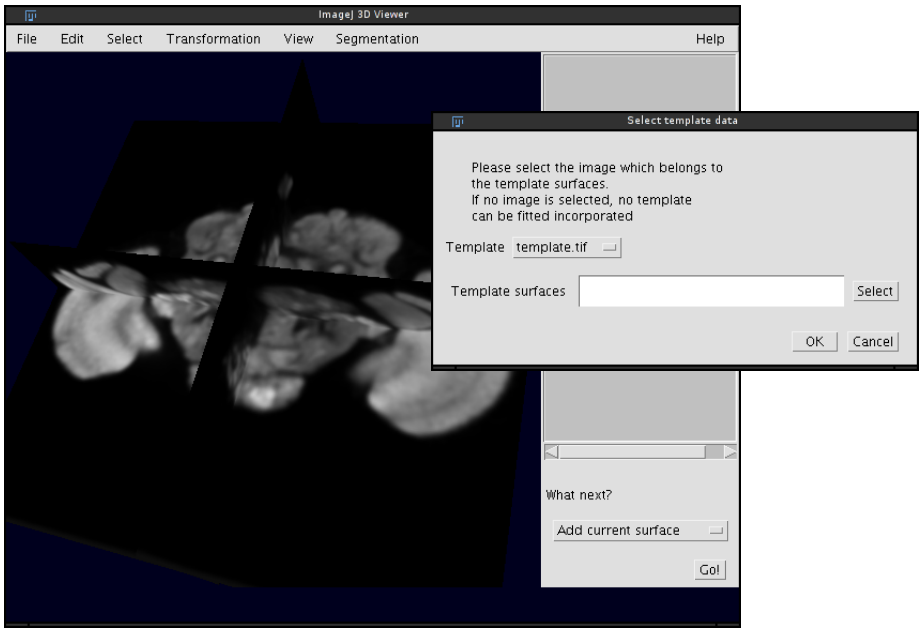

Figure A.1: Start of the semiautomatic segmentation plugin. The model image is displayed in orthoslice view in a new 3D window, and a dialog window asks the user for a template greyscale image and corresponding template surfaces. If the user cancels here, the startup process will continue normally, but template surfaces will not be available later.

either pressing additionally the right arrow key or scrolling down one unit will extend the surface by one new point. Using '1', '2' or '3' instead of '0' will extend the surface by 10, 100 or 1000 additional surface points, respectively.

From time to time, the expanding surface happens to follow wrong contours. It is then necessary to go back a few steps, i.e. to remove the latest added triangles. This can be performed similarly, by pressing the left arrow key (instead of the right one) or scrolling up (instead of down), again holding down either '0', '1', '2' or '3' key. Especially the scroll wheel allows so to quickly go forth and back to complete the segmentation to the desired degree.

In case the surface follows wrong edges, a new seed point can be selected, again by clicking at the desired location while holding down the 'g' key. Further expansion of the surface now starts from the newly selected point. If the segmentation turns out to be unsuccessful, "reset" is possible by pressing the 'Backspace' key.

At any time, the expanding surface is displayed in the 3D view as a triangle mesh, overlaying the orthoslice view of the segmented image. The triangles themselves are rendered red, extendable border edges are displayed yellow, while green border edges indicate those edges which will not be extended by the algorithm (due to various possible conditions, such as too high curvature. For details see section 4.1.2). Figure A.2 shows an ongoing segmentation.

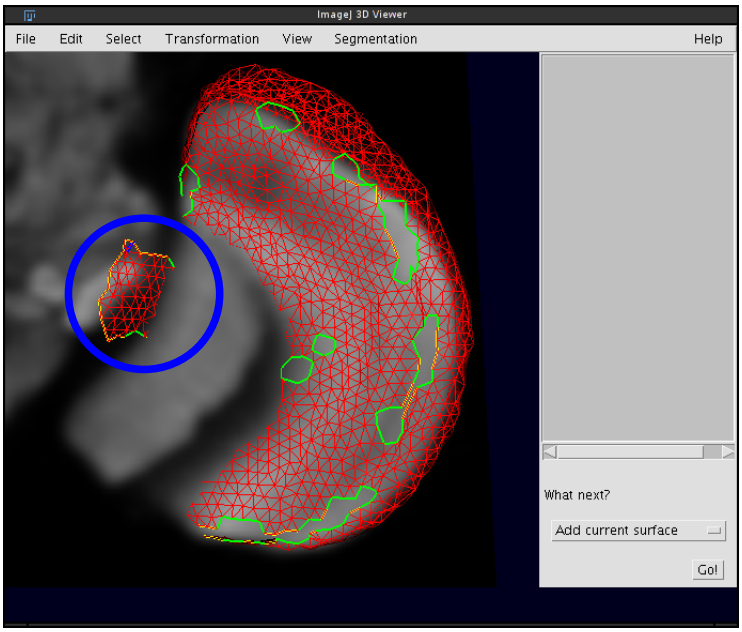

Figure A.2: Ongoing segmentation via the growing surface method. Various aspects are depicted here: The segmentation is displayed in the 3D view as a red triangle mesh, overlaying the model image. Normal border edges are rendered yellow, while those regarded as non-extendable are rendered green. The blue circle indicates a part of the surface which follows undesired edges. User can control such a situation by going back and forth in the segmentation and selecting different seed points.

A.3 Surface completion

As mentioned in previous chapters, the meshes resulting from the surface growing algorithm may be incomplete and contain holes (see also figure A.2). The user has several options about how to proceed, all of which are activated via the "Segmentation" menu. Smaller holes can be completed using "Close holes". Users are asked to enter the maximum number of edges a hole may contain in order to be closed. "Close holes" simply connects the individual border points of the holes. Larger holes can be completed with either "Close surface" or "Align template surface". The latter is described below.

As explained before, template surfaces are incorporated in three steps. First they are roughly aligned using intensity-based image registration, then a more accurate rigid alignment is achieved via the ICP algorithm, and finally, the active surface model is applied, allowing free-form deformations. All three steps are invoked automatically once a user clicks on "Align template surface". After each step the alignment is shown as a green surface, and users can decide if they want to cancel the process or continue (see also figure 4.12, p. 56).

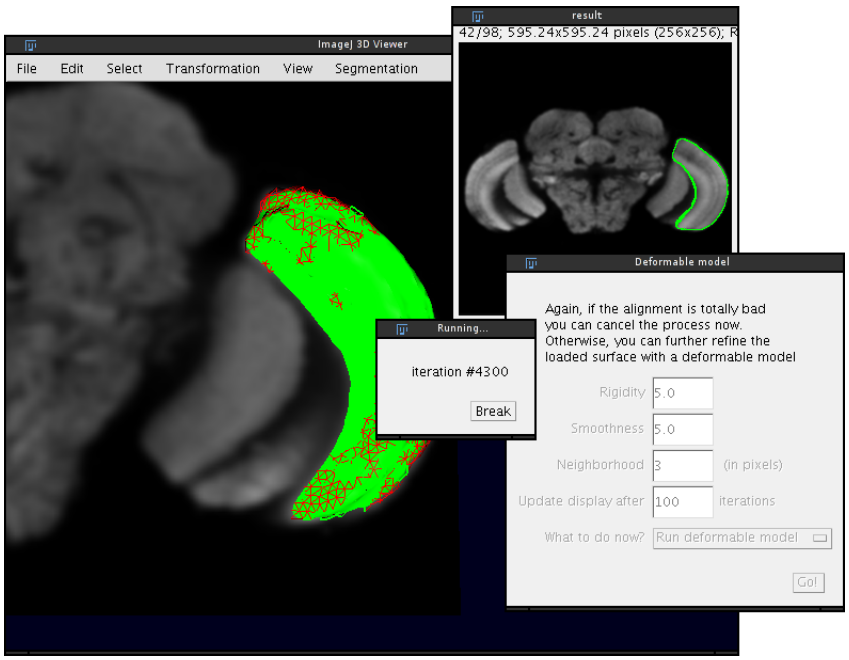

Figure A.3: Aligning template surfaces. Shown here is the final step, the active surface model. While the model is evolving, the 3D display is updated continuously, so that users can observe the process. He can interact with it at any time by interrupting the iterative process and changing the model parameters.

While the rigid registration and the ICP algorithm run automatically without user interaction, the active surface model is executed interactively, i.e. users can at any time cancel the iterations, change the model parameters, continue, reset etc. The adapting surface is shown in the 3D view and simultaneously projected onto the stack window (fig. A.3).

A.4 Managing completed surfaces

Typically, after segmenting one structure, a user wants to store the result somehow and pass on to the next structure. In this plugin, the stored surfaces are displayed in a panel at the right side of the window (fig. A.4): On the top, a list is displayed with all the segmentations finished so far. Every entry in this list can optionally be shown in the 3D Viewer or hided. This is done via the right-mouse-click context menu. At the bottom of the panel a choice box allows users to choose among actions:

Add current surface
: Transfers the current segmentation (as visible in the 3D Viewer) to the list of finished segmentations. It assigns automatically a name and a colour, both of which can be altered by the user.

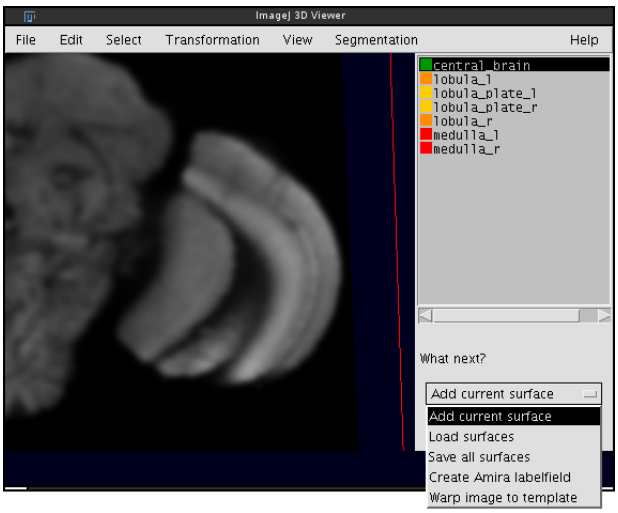

Figure A.4: Completed segmentations are managed in the right panel. A list shows the segmented surfaces, which can optionally be displayed or hided in the 3D view. A choice box at the bottom provides access to several utility functions.

Save all surfaces
: Saves all the surfaces displayed in the list in a single file. The generated file is again in the OBJ file format.

Load surfaces
: Loads previously saved surfaces. Saving and loading also allows to interrupt the work and continue at a later time point.

Create Amira labelfield
: Produces a new image stack which can be saved in the Amira labelfield format. This is a convenient way to store segmentations in volumetric form.

Warp image to template
: Uses the segmentations in the list to align the model image to the template image with the techniques described in chapter 5.

B Manual for fully-automatic registration

B.1 The configuration file

The implementation of the fully-automatic approach is still work in progress. The current state focuses on automation and adjustment capabilities. It does not offer a sophisticated GUI, as it was implemented for the semi-automatic plugin. All settings must be provided by the user via a configuration file, which is briefly described below.

Basically, the configuration file stores all settings as key-value pairs. There is one pair per line. Key and value are separated by an '=' character. Lines beginning with '#' are ignored. The configuration file is divided into several sections. The beginning of each new section is indicated by a line which encloses the section's name in square brackets. An example is shown below:

```
# Comment which is ignored
[Name of new section]
    property1 = value1
    property2 = value2

[Name of another section]
    property1 = value1
...
```

B.2 The *Preprocessing* section

The program expects as the first section to be called "Preprocessing". As described in chapter 7, preprocessing comprises several steps: First, the image under study is downsampled. Next, a bilateral filter is applied for smoothing

and a 3D gradient filter for edge detection. Weak edges are suppressed by setting pixel values below a certain threshold to zero. From the resulting edge image, maxima are identified which serve as seed points for the surface growing algorithm. In the "Preprocessing" sections, the following keys are recognised by the software and influence the work flow:

key	**header**
resx	Resampling factor in x-direction.
resy	Resampling factor in y-direction.
resz	Resampling factor in z-direction.
sigma_spatial	Spatial standard deviation of the bilateral filter.
sigma_intensity	Intensity-based standard deviation of the bilateral filter.
threshold	Threshold for neglecting weak edges.

B.3 Sections for individual structures

After the *Preprocessing* section follows one section for each anatomical structure which one wants to segment. In case of the confocal images, these are the the medulla, lobula, lobula plates (for both hemispheres) and the central brain. The name of each section is the name of the corresponding structure.

As described in chapter 4, the alignment of template surfaces comprises three steps: 1) A rough rigid registration based on intensity data of the model and the template images, 2) Shape alignment via the ICP algorithm and 3) Free-form refinement via an active surface model. As elaborated in chapter 7, rough rigid registration is often not precise enough to initialise the ICP algorithm. This is in particular the case for the lobula and lobula plates. The ICP of these structures must can be initialised more accurately using the results from aligning the medulla (see chapter 7, p. 78 for more details). The only parameters which influence the template surface alignments are those for the active surface model: the weights of the rigidity, smoothness and expansion forces, the searched neighbourhood and the number of iterations.

To encode this information in the configuration file, two keys are recognised and evaluated in each section: The first key is *init_from*. The associated value must be the name of the structure from which the ICP algorithm is initialised. In the section of the right lobula ([lobula_r]), the corresponding line would read *init_from = medulla_r*. If the *init_from* key is absent, the ICP is initialised by the intensity-based rigid registration.

The second key is *activesurface*. The value is a character string storing all the parameters of an active surface model, enclosed in squared brackets. For example, this line may look like: *activesurface = [1.0 15.0 15.0 3 2000]*. In this case, an active surface model would be applied with a rigidity force weight of

1.0, a smoothness and expansion force weight of 15, the search space would have an extent of 3 pixels (radius), and the model would be run for 2000 iterations. If more than one *activesurface* line is present in a section, more models are applied to the template surface consecutively.

The description of the configuration file may sound more complicated than it actually is. For the case of the fly brain images, it is relatively short and shown below. It also documents the empirically derived parameter settings.

FullyAutomatic.conf:

```
# Example configuration file for automatically segmenting
# confocal images of adult Drosophila brains.

[Preprocessing]
      resx = 1
      resy = 1
      resz = 1
      threshold = 50
      sigma_spatial = 5
      sigma_intensity = 150

[central_brain]
      activesurface = [1.0 15.0 15.0 3 2000]

# right hemisphere
[medulla_r]
      activesurface = [5.0 5.0 1 3 2000]

[lobula_plate_r]
      init_from = medulla_r
      activesurface = [10.0 5.0 0 1 1000]
      activesurface = [5.0 3.0 0 1 1000]

[lobula_r]
      init_from = medulla_r
      activesurface = [10.0 15.0 0 1 1000]
      activesurface = [5.0 5.0 0 1 1000]

# left hemisphere
[medulla_l]
      activesurface = [5.0 5.0 1 3 2000]

[lobula_plate_l]
      init_from = medulla_l
      activesurface = [10.0 5.0 0 1 1000]
      activesurface = [5.0 3.0 0 1 1000]

[lobula_l]
      init_from = medulla_l
      activesurface = [10.0 15.0 0 1 1000]
      activesurface = [5.0 5.0 0 1 1000]
```

API example for the 3D visualisation library

We designed our 3D visualisation framework with a dual purpose, on the one hand to provide an interactive 3D scene for end-users, on the other hand to provide an easy programming interface to instantiate a 3D scene and manipulate its internals. Programmers can trivially augment applications with 3D visualisation capabilities. This section demonstrates the latter aspect. A working example is listed. Similar examples and tutorials, together with source code, are found online at `http://3dviewer.neurofly.de`.

Listing C.1 shows how an instance of `Image3DUniverse` is created. An image is opened and added to the universe as a volume rendering via `addVoltex()`. This method returns the new `Content`, which is then used to make it partially transparent.

After adjusting the view, the `VoltexVolume` of the volume rendering is retrieved. The `VoltexVolume` allows to change voxel values in volume renderings directly. The example shows how the values are read from a file input stream, which are interpreted as tuples of 4 consecutive integer values, specifying the x, y and z coordinate and the new voxel value.

The example uses data obtained via a segmentation algorithm. The red channel shows an antibody staining, visualising the expression pattern of a *Drosophila* GAL4 driver line. This channel was used to simulate axon growth.

Listing C.1: Source code for opening an image, rendering it as a volume rendering in a virtual universe and finally changing individual voxel values, as read from a stream.

```
1  import ij3d.Image3DUniverse;
2  import ij3d.Content;
3
4  import voltex.VoltexVolume;
5  import voltex.VoltexGroup;
6
7  import ij.IJ;
8  import ij.ImageJ;
9  import ij.ImagePlus;
10
11 import java.io.DataInputStream;
12 import java.io.FileInputStream;
13 import java.io.EOFException;
14
15 public class Editable_Volume {
16
17     public static void main(String[] args) throws Exception {
18         new ImageJ();
19
20         // Create a 3D scene and show it
21         Image3DUniverse univ = new Image3DUniverse(600, 480);
22         univ.show();
23
24         // Open the image and add it as a volume
25         ImagePlus image = IJ.openImage("thorax.tif");
26         Content c = univ.addVoltex(image);
27         c.setTransparency(0.7f);
28
29         // Adjust the view
30         univ.rotateToNegativeXY();
31         univ.waitForNextFrame();
32         univ.getViewPlatformTransformer().zoom(40);
33
34         // Retrieve the VoltexVolume of this volume rendering
35         VoltexVolume volume = ((VoltexGroup)c.getContent()).
36                     getRenderer().getVolume();
37
38         // Read in data from file
39         DataInputStream is = new DataInputStream(
40                     new FileInputStream("thorax.stream"));
41
42         while (true) {
43             try {
44                 int x = is.readInt();
45                 int y = is.readInt();
46                 int z = is.readInt();
47                 int v = is.readInt();
48                 volume.set(x, y, z, v);
49             } catch(EOFException ex) {
50                 break;
51             }
52         }
53         is.close();
54     }
55 }
```

Curriculum Vitae

PERSÖNLICHE DATEN:

Adresse:	Benjamin Schmid Hintere Kirchgasse 6a 97078 Würzburg
Geburtsdatum:	5. März 1981

AUSBILDUNG:

2006 - 2010:	Promotion an der Maximilians Universität Würzburg
2001 - 2006:	Studium an der Fachhochschule Weihenstephan, Diplom-Studiengang Bioinformatik
2000 - 2001:	Zivildienst beim ARV Tirschenreuth
1991 - 2000:	Stiftland-Gymnasium Tirschrenreuth

PROJEKTE UND FORSCHUNG:

Seit Juli 2006	**Universität Würzburg, Lehrstuhl für Neurobiologie und Genetik** *Promotion bei Prof. Dr. Martin Heisenberg* Computational tools for the segmentation and registration of confocal brain images of *Drosophila melanogaster*
Sept. - Juni 2006	**University of Cambridge, Dep. of Plant Sc.** *Diplomarbeit Prof. Dr. Alison Smith* Mathematical modelling of the regulation network of tetrapyrrole biosynthesis in *Arabidopsis thaliana*.
Sept. - März 2005	**Molecular Networks GmbH, Erlangen** *Praktikum bei Prof. Dr. Johann Gasteiger* A linear model for predicting the solubility coefficient for organic molecules.
April - Sept. 2003	**GSF Neuherberg, Institut für Bioinformatik** *Praktikum bei Dr. Axel Facius* PRIME, a graphical interface for integrating genomic and proteomic databases.

Bibliography

[1] Rein K, Zöckler M, Mader MT, Grübel C, Heisenberg M: **The Drosophila Standard Brain**. *Current Biology* 2002, **12**(3):227 – 231.

[2] Brand AH, Perrimon N: **Targeted gene expression as a means of altering cell fates and generating dominant phenotypes**. *Development* 1993, **118**(2):401–415.

[3] Jenett A, Schindelin J, Heisenberg M: **The Virtual Insect Brain protocol: creating and comparing standardized neuroanatomy**. *BMC Bioinformatics* 2006, **7**:544.

[4] Pawley JB: *Handbook of Biological Confocal Microscopy 2nd ed.* New York: Plenum Press 1995.

[5] Horn BK: **Closed-form solution of absolute orientation using unit quaternions**. *Journal of the Optical Society of America* 1987, **4**:624–642.

[6] Brent RP: **Algorithms for finding zeros and extrema of functions without calculating derivatives**. *PhD thesis*, Stanford University, Stanford, CA, USA 1971.

[7] Rasband W: **ImageJ: Image Processing and Analysis in Java [version 1.43k]**. Bethesda, Maryland, USA 1997–2009, [http://rsb.info.nih.gov/ij/].

[8] Schindelin J: **Fiji is just ImageJ – Batteries included**. In *Proceedings of the ImageJ User and Developer Conference*, Luxembourg 2008.

[9] Clendenon JL, Phillips CL, Sandoval RM, Fang S, , Dunn KW: **Voxx: a PC-based, near real-time volume rendering system for biological microscopy**. *American Journal of Physiology - Cell Physiology* 2002, **282**:C213–C218.

[10] Peng H: **V3D** 2009, [http://penglab.janelia.org/proj/v3d/v3d2.html].

[11] Abràmoff M: **VolumeJ** 2003, [http://bij.isi.uu.nl/index.htm].

[12] Barthel KU: **Volume Viewer, a volume rendering plugin for ImageJ** 2005-2007, [http://rsb.info.nih.gov/ij/plugis/volume-viewer.html].

[13] Sun Microsystems Java 3D Engineering Team: **Java 3D API Tutorial** 2000, [http://java.sun.com/developer/onlineTraining/java3d].

[14] **Online Java 3D Application Programming Interface**[http://download.java.net/media/java3d/javadoc/1.5.2/index.html].

[15] Levoy M: **Display of Surfaces from Volume Data**. *IEEE Computer Graphics and Applications* 1988, **8**(3):29–37.

[16] Drebin RA, Carpenter L, Hanrahan P: **Volume rendering**. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM 1988:65–74.

[17] Krueger J, Westermann R: **Acceleration Techniques for GPU-based Volume Rendering**. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003*, Washington, DC, USA: IEEE Computer Society 2003:38.

[18] Roettger S, Guthe S, Weiskopf D, Ertl T, Strasser W: **Smart hardware-accelerated volume rendering**. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association 2003:231–238.

[19] Engel K, Hadwiger M, Kniss JM, Lefohn AE, Salama CR, Weiskopf D: **Real-time volume graphics**. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, New York, NY, USA: ACM 2004:29.

[20] Gehringer D: **Tutorial on Volume Rendering in Java 3D** 2006, [http://java3d.j3d.org/tutorials/quick_fix/volume.html].

[21] Lorenson WE, Harvey CE: **Marching cubes: A high resolution 3D surface construction algorithm**. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, Volume 21*, New York, NY, USA: ACM Press 1987:163–169.

[22] Chernyaev EV: **Marching Cubes 33: Construction of Topologically Correct Isosurfaces**. Tech. rep., CERN 1995.

[23] Nielson GM: **On Marching Cubes**. *IEEE Transactions on Visualization and Computer Graphics* 2003, **9**:283–297.

[24] Lewiner T, Lopes H, Vieira AW, Tavares G: **Efficient implementation of Marching Cubes' cases with topological guarantees**. *Journal of Graphics Tools* 2003, **8**:2003.

[25] Shen HW, Hansen CD, Livnat Y, Johnson CR: **Isosurfacing in Span Space with Utmost Efficiency (ISSUE)**. In *In IEEE Visualization 96* 1996:287–294.

[26] Livnat Y, Shen HW, Johnson CR: **A Near Optimal Isosurface Extraction Algorithm Using the Span Space**. *IEEE Transactions on Visualization and Computer Graphics* 1996, **2**:73–84.

[27] Parker S, Shirley P, Livnat Y, Hansen C, pike Sloan P: **Interactive Ray Tracing for Isosurface Rendering**. In *In IEEE Visualization 98* 1998:233–238.

[28] Livnat Y, Tricoche X: **Interactive point-based isosurface extraction**. In *IEEE Visualization* 2004:457–464.

[29] Newman TS, Yi H: **A survey of the marching cubes algorithm**. *Computers & Graphics* 2006, **30**(5):854–879.

[30] Schroeder WJ, Zarge JA, Lorensen WE: **Decimation of triangle meshes**. In *SIGGRAPH '92: Proceedings in Computer Graphics* 1992:65–70.

[31] Preibisch S, Saalfeld S, Rohlfing T, Tomancak P: **Bead-based mosaicing of single plane illumination microscopy images using geometric local descriptor matching**. In *Medical Imaging 2009: Image Processing, Volume 7259*. Edited by Pluim JPW, Dawant BM, SPIE 2009:72592S.

[32] Huisken J, Swoger J, Del Bene F, Wittbrodt J, Stelzer EHK: **Optical Sectioning Deep Inside Live Embryos by Selective Plane Illumination Microscopy**. *Science* 2004, **305**(5686):1007–1009.

[33] Longair M: **Computational Neuroanatomy of the Central Complex of Drosophila melanogaster**. *PhD thesis*, University of Edinburgh, School of Informatics 2009.

[34] Longair M: **Simple Neurite Tracer, an ImageJ plugin for tracing and reconstructing neurites.** 2006-2009, [http://homepages.inf.ed.ac.uk/s9808248/imagej/tracer].

[35] Cardona A, Saalfeld S, Tomančák P, Hartenstein V: ***Drosophila* brain development: closing the gap between a macroarchitectural and a microarchitectural approach**. *Cold Spring Harbor Symposia on Quantitative Biology* 2009.

[36] Cardona A: **TrakEM2: an ImageJ-based program for morphological data mining and 3d modeling**. In *Proceedings of the ImageJ User and Developer Conference*, Luxembourg 2006.

[37] Pal NR, Pal SK: **A review on image segmentation techniques**. *Pattern Recognition* 1993, **26**(9):1277 – 1294.

[38] Ballard DH, Brown CM: *Computer Visison*. Prentice-Hall 1982.

[39] Kundu A: **Local segmentation of biomedical images**. *Computerized Medical Imaging and Graphics* 1990, **14**(3):173 – 183.

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

[41] Lotufo RA, Falcão AX, Zampirolli FA: **IFT-Watershed from Gray-Scale Marker**. *Brazilian Symposium on Computer Graphics and Image Processing,* 2002, **0**:146.

[42] Burger W, Burge MJ: *Digital Image Processing: An Algorithmic Introduction using Java*. Springer 2007.

[43] Schindelin J: **The standard brain of Drosophila melanogaster and its automatic segmentation**. *PhD thesis*, University of Wuerzburg 2005.

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

[45] Cohen LD: **On active contour models and balloons**. *CVGIP: Image Understanding* 1991, **53**(2):211–218.

[46] Xu C, Prince JL: **Gradient vector flow: a new external force for snakes**. In *IEEE Proceedings in Computer Vision and Pattern Recognition* 1997.

[47] Xu C, Prince JL: **Snakes, shapes, and gradient vector flow**. *IEEE Transactions on Image Processing* 1998, **7**(3):359–369.

[48] Cohen LD, Cohen I: **Finite-Element Methods for Active Contour Models and Balloons for 2D and 3D Images**. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1993, **15**(11):1131–1147.

[49] Miller JV, Breen DE, Lorensen WE, O'Bara RM, Wozny MJ: **Geometrically deformed models: a method for extracting closed geometric models form volume data**. *SIGGRAPH Compututer Graphics* 1991, **25**(4):217–226.

[50] McInerney T, Terzopoulos D: **A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4D image analysis**. *Computerized Medical Imaging and Graphics* 1995, **19**:69 – 83.

[51] Besl PJ, McKay ND: **A Method for Registration of 3D Shapes**. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1992, **14**(2):239–256.

[52] Maintz JBA, Viergever MA: **A survey of medical image registration**. *Medical Image Analysis* 1998, **2**:1–36.

[53] Bookstein FL: **Principal Warps: Thin-Plate Splines and the Decomposition of Deformations**. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1989, **11**(6):567–585.

[54] Lowe DG: **Distinctive Image Features from Scale-Invariant Keypoints**. *International Journal of Computer Vision* 2004, **60**(2):91–110.

[55] Brown M, Szeliski R, Winder S: **Multi-Image Matching Using Multi-Scale Oriented Patches**. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA: IEEE Computer Society 2005:510–517.

[56] Mikolajczyk K, Schmid C: **A Performance Evaluation of Local Descriptors**. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2005, **27**:1615–1630.

[57] Bay H, Tuytelaars T, Van Gool L: **SURF: Speeded-Up Robust Features**. In *9th European Conference on Computer Vision*, Graz, Austria 2006:404–417.

[58] Harris C, Stephens M: **A Combined Corner and Edge Detection**. In *Proceedings of The Fourth Alvey Vision Conference* 1988:147–151.

[59] Fischler MA, Bolles RC: **Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography**. *Communications of the ACM* 1981, **24**(6):381–395.

[60] Carreras IA, Sorzano C, Marabini R, Carazo J, De Solorzano CO, Kybic J: **Consistent and Elastic Registration of Histological Sections Using Vector-Spline Regularization**. In *Computer Vision Approaches to Medical Image Analysis* 2006:85–95.

[61] Saalfeld S: **Automatic inter-slice registration of tiled Transmission Electron Microscopy (TEM) images**. *Diploma thesis* 2008.

[62] Schaefer S, McPhail T, Warren J: **Image deformation using moving least squares**. *ACM Transactions on Graphics* 2006, **25**(3):533–540.

[63] Tomasi C, Manduchi R: **Bilateral Filtering for Gray and Color Images**. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, Washington, DC, USA: IEEE Computer Society 1998:839.

[64] MessaoudiI C, Boudier T, Sorzano C, Marco S: **TomoJ: tomography software for three-dimensional reconstruction in transmission electron microscopy**. *BMC Bioinformatics* 2007, **8**:288.

[65] Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, Meng EC, Ferrin TE: **UCSF Chimera–a visualization system for exploratory research and analysis.** *Journal of computational chemistry* 2004, **25**(13):1605–1612.

[66] Preibisch S, Saalfeld S: **Imglib: An n-dimensional image container** 2009, [http://pacific.mpi-cbg.de/wiki/index.php/Imglib].

[67] Toga AW, Thomson PM: **Maps of the brain**. *The Anatomical Record* 2001, **265**(2):37–53.

[68] Toga AW, Thompson PM, Mori S, Amunts K, Zilles K: **Towards multimodal atlases of the human brain**. *Nature reviews Neuroscience* 2006, **7**(12):952–966.

[69] Rein K, Zöckler M, Heisenberg M: **A quantitative three-dimensional model of the Drosophila optic lobes**. *Current Biology* 1999, **9**(2):92–96.

[70] Dreyer D, Vitt H, Dippel S, Goetz B, el Jundi B, Kollmann M, Huetteroth W, Schachtner J: **3D standard brain of the red flour beetle Tribolium castaneum: a tool to study metamorphic development and adult plasticity**. *Frontiers in Systems Neuroscience* 2010, **4**.

[71] el Jundi B, Huetteroth W, Kurylas AE, Schachtner J: **Anisometric brain dimorphism revisited: Implementation of a volumetric 3D standard brain in Manduca sexta**. *The Journal of Comparative Neurology* 2009, **517**(2):210–225.

[72] Huetteroth W, el Jundi B, el Jundi S, Schachtner J: **3D-reconstructions and virtual 4D-visualization to study metamorphic brain development in the sphinx moth Manduca sexta**. *Frontiers in Systems Neuroscience* 2010, **4**(7).

[73] Kurylas AE, Rohlfing T, Krofczik S, Jenett A, Homberg U: **Standardized atlas of the brain of the desert locust, Schistocerca gregaria**. *Cell and Tissue Research* 2008, **333**:125–45.

[74] el Jundi B, Heinze S, Lenschow C, Kurylas AE, Rohlfing T, Homberg U: **The locust standard brain: A 3D standard of the central complex as a platform for neural network analysis**. *Frontiers in Systems Neuroscience* 2009, **3**.

[75] Kvello P, Løfaldli BB, Rybak J, Menzel R, Mustaparta H: **Digital, three-dimensional average shaped atlas of the heliothis virescens brain with integrated gustatory and olfactory neurons**. *Frontiers in Systems Neuroscience* 2009, **3**(14).

[76] Jefferis GS, Potter CJ, Chan AM, Marin EC, Rohlfing T, Maurer CR Jr, Luo L: **Comprehensive Maps of Drosophila Higher Olfactory Centers: Spatially Segregated Fruit and Pheromone Representation**. *Cell* 2007, **128**(6):1187–1203.

[77] Brandt R, Rohlfing T, Rybak J, Krofczik S, Maye A, Westerhoff M, Hege HC, Menzel R: **Three-dimensional average-shape atlas of the honeybee brain and its applications.** *Journal of Comparative Neurology* 2005, **492**:1–19.

[78] Chiang AS: **Fly Circuit Database** 2009, [http://www.flycircuit.tw].

[79] Mercury Inc: **Amira 5.2 User's Guide** 2009, [http://www.amira.com/documentation/manuals-and-release-notes.html].

[80] Rudge T: **Amira extension for the segmentation of plant cells** 2003, [http://www.plantsci.cam.ac.uk/Haseloff/TimRudge].

[81] Ibanez L, Schroeder W, Ng L, Cates J: *The ITK Software Guide*. Second, Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf 2005.

[82] Rohlfing T: **The Computational Morphometry Toolkit (CMTK)** 2009, [http://www.nitrc.org/projects/cmtk/].

[83] Rueckert D, Sonoda LI, Hayes C, Hill DLG, Leach MO, Hawkes DJ: **Nonrigid registration using free-form deformations: Application to breast MR images**. *IEEE Transactions on Medical Imaging* 1999, **18**:712–721.

[84] Rohlfing T, Maurer CR Jr: **Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees.** *IEEE Transactions on Information Technology in Biomedicine* 2003, **7**:16–25.

Erklärung

Erklärung gemäß §4 Absatz 3 der Promotionsordnung der Fakultät für Biologie der Bayerischen Julius-Maximilians-Universität zu Würzburg vom 15. März 1999.

Hiermit erkläre ich, die vorgelegte Dissertation selbständig angefertigt zu haben und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet zu haben. Alle aus der Literatur genommenen Stellen sind als solche kenntlich gemacht. Des Weiteren erläre ich, dass die vorliegende Arbeit weder in gleicher noch in abgeänderter Form bereits in einem anderen Prüfungsverfahren vorgelegen hat. Ausser dem Diplom in Bioinformatik habe ich bisher weder einen akademischen Grad erworben, noch einen solchen zu erwerben versucht.

Würzburg, den

_____

Benjamin Schmid