FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Deformation Based Manual Segmentation in Three and Four Dimensions

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Tobias Fechter
Matrikelnummer 0325253

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Math. Dr.techn. Katja Bühler

Wien, TT.MM.JJJJ

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Deformation Based Manual Segmentation in Three and Four Dimensions

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Medical Informatics

by

Tobias Fechter
Registration Number 0325253

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Math. Dr.techn. Katja Bühler

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Tobias Fechter
Nappersdorf 7, 2023 Nappersdorf

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

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First of all I would like to thank Eduard Gröller for supervising and reviewing my work on this master thesis.

I thank Katja Bühler for providing me with the topic and for supporting and advising me during the entire work. Additionally I would like to thank the whole MedVis Group at the VRVis for providing me with all necessary facilities and a lot of beneficial ideas.

Special thanks to my parents for enabling me my study. Furthermore I would like to convey my deepest gratefulness to my parents and my sisters for supporting and encouraging me during my studies and my thesis writing. Without you this work would not have been possible!

Finally, I would like to express my appreciation to all my friends who were always ready for me and provided me with the sometimes necessary distraction.

Abstract

Segmentation of medical image data has grown into one of the most important parts in medicine during the past years. The main fields of segmentation in the area of medicine are surgical-planning, diagnosis, therapy-planning and simulation. The focus was in the last years mainly on automatic and semi-automatic segmentation methods for 3D and 4D datasets. Most of them are highly specialized as they need a lot of prior knowledge and often the results, especially in presence of pathologies or other abnormalities, have to be corrected manually. An alternative to automatic and semi-automatic methods is to perform the segmentation manually. The main drawbacks of this approach are that it is very tedious and time consuming, the user knowledge has a very high impact on the results and it is very hard to reproduce specific results.

This work presents a tool that enables the user to enhance results of automatic and semi-automatic algorithms and to do fast manual segmentation of shapes of arbitrary topology from scratch. The tool can deal with three and four dimensional image datasets captured by different modalities. The segmentation is mesh based and performed with a 2D cut approach. With the use of this approach the user aligns a 2D cut through the mesh to the shape to segment but in the background the 3D mesh gets deformed. To achieve a better alignment of the edges to a specific shape the edge class based Sticky Edges algorithm is introduced. Furthermore, well known mesh optimization algorithms like subdivision, smoothing and decimation were implemented to accomplish better results. To achieve a faster segmentation of four dimensional datasets two methods are presented. With the first one the user can record its interactions on one volume in the 4D dataset and apply them automatically to the other volumes. The other one enables the user set an already segmented mesh as start position for the segmentation of other volumes.

The approach presented in this work is up to 25 times faster than the Livewire approach [41] that was used to evaluate this tool. Moreover, the mesh quality regarding smoothness, curvature and triangle quality are at eye level with the evaluation meshes. The geometric distance to the ground truth meshes is on average 2 mm and the normal deviation is between 0.3 and 0.4 degree. To sum up, this master thesis introduces a tool for fast manual image segmentation that provides proper mesh quality.

Kurzfassung

Die Segmentierung von medizinischen 3D und 4D Bilddaten etablierte sich in den letzten Jahren zu einem der wichtigsten Teilgebiete der Medizin. Die Hauptaufgabengebiete der Segmentierung im medizinischen Bereich liegen in der Diagnose, der Simulation und der Planung von Therapien und Operationen. Der Forschungsfokus lag in den letzten Jahren hauptsächlich in der Entwicklung von automatischen und halb-automatischen Segmentierungsalgorithmen. Der Nachteil dieser Algorithmen ist, dass sie auf bestimmte Problemstellungen spezialisiert sind, weil viel Vorwissen für die automatische und halb-automatische Segmentierung notwendig ist und dass die Segmentierungsergebnisse oft nachkorrigiert werden müssen, wenn Pathologien oder andere Abnormalitäten existieren. Eine Alternative zu automatischen und halb-automatischen Methoden bietet die manuelle Segmentierung. Die Nachteile dieses Ansatzes liegen darin, dass er sehr zeitaufwändig und ermüdend ist, dass das Ergebnis sehr stark vom Wissen des Arztes abhängt und dass es sehr schwer ist, Ergebnisse zu reproduzieren.

Diese Arbeit präsentiert einen Ansatz, der es ermöglicht, die Ergebnisse von automatischen und halb-automatischen Segmentierungsalgorithmen zu verbessern, sowie eine schnelle manuelle Segmentierung von Objekten willkürlicher Form von Grund auf durchzuführen. Mit Hilfe des Programms, dass im Zuge dieser Arbeit entwickelt wurde, können 3D und 4D Bilddatensätze aller gängigen Bildgebungsverfahren segmentiert werden. Die Segmentierung, basierend auf Dreiecksnetzen, wird anhand eines zweidimensionalen Schnittes durch das Gitternetz durchgeführt, der durch den Benutzer an die zu segmentierenden Form angepasst wird. Im Hintergrund wird dabei allerdings das Gitternetz in allen drei Dimensionen verändert. Um das Gitternetz besser an Kanten anzupassen, wird der Sticky Edges-Algorithmus vorgestellt, der den Anwender in dieser Aufgabe unterstützt. Um eine schnellere Segmentierung von 4D Datensätzen zu ermöglichen, stehen dem Benutzer zwei Methoden zur Verfügung. Mit Ersterer können die Segmentierungsschritte, die an einem Datensatz durchgeführt werden, aufgezeichnet und danach automatisch auf andere Datensätze übertragen werden. Die zweite Methode ermöglich es, ein bereits segmentiertes Gitternetz eines Datensatzes als Ausgangspunkt für die Segmentierung eines anderen Datensatzes zu verwenden. Der Ansatz dieser Arbeit ist bis zu 25 mal schneller, als die Segmentierung mit der Livewire-Methode [41], die zur Evaluierung herangezogen wurde. Bezüglich Glätte, Krümmung und Dreiecks-Qualität der Gitternetze sind die generierten Ergebnisse auf Augenhöhe mit denen der Evaluierungssoftware. Der durchschnittliche geometrische Abstand zwischen den Gitternetzen der Evaluierungssoftware und dieses Ansatzes beträgt 2 mm. Die Abweichung der Normalen beträgt zwischen 0.3 und 0.4 Grad. Zusammenfassend wird in dieser Arbeit eine Methode präsentiert, die eine schnelle manuelle Segmentierung ermöglicht und Gitternetze von guter Qualität liefert.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

According to Sherrow [48] more than a quarter billion imaging procedures were performed in the USA in the year 2006. These image procedures create datasets with 2-, 3- or 4-dimensional data. This vast amount of data has to be processed and interpreted to gain information that can be used for patients' treatment. As these procedures have to be done as fast as possible without the loss of quality, technical routines were developed to assist radiologists in their work. Bankman [4] classifies these routines into:

Enhancement: reduces image noise and improves contrast in regions of interest

Segmentation: delineates regions of interest from background

Quantification: extracts essential information from segmented shapes

Registration: finds correspondences between two or more images

Visualization: presents data in a convenient way

Compression, Storage, and Communication: deals with efficient and lossless compression and storage, plus the communication of data

With the exception of some preprocessing steps that are related to the class of enhancement, this thesis deals mainly with the field of segmentation 1.1 is concerned with imaging methods relevant to the area of segmentation. Section 1.2 describes the basics of segmentation and gives a short introduction to this field. The sections 1.3 and 1.4 will give an understanding of the terms *deformation* and *four dimensional data* that are crucial for the further reading of this thesis. 1.5 and 1.6 explain the problem this master thesis tries to solve and summarize the aim of this work. 1.7 outlines shortly the remaining chapters of this thesis.

1.1 Imaging Methods

Note that this section gives only a brief overview over the imaging methods relevant for segmentation. For a more detailed view please consult Yoo [57]. In the remainder of this section two terms for resolution will occur. On the one hand "spatial resolution" is the area covered by a voxel and on the other hand "sampling resolution" is the number of voxels in each dimension of a slice.

1.1.1 Computed Tomography

With the help of X-rays, Computed Tomography (CT) generates a 3D volume out of a series of transaxial taken 2D images. The sampling resolution of CT is either 256 x 256 or 512 x 512 voxels. The spatial resolution lies usually in the interval between 0.3 and 2 mm. The value of each voxel represents the ionizing radiation attenuation characteristic of the tissue it depicts. The unit of measure is called Hounsfield unit (HU). The zero point of the Hounsfield scale is the attenuation coefficient of distilled water at standard temperature and pressure.

1.1.2 Nuclear Medicine

In nuclear-medicine imaging methods, a pharmaceutical tagged with a radioactive agent is injected into the patient. The distribution of this pharmaceutical is measured by detecting the emitted radiation. The detection of emitted radiation works similarly to CT. The main difference of this method to CT is that the radiation source is inside the human body. Nuclear-medicine methods are often used to depict physiological activities. The main drawbacks of this kind of methods are the low resolution and the high amount of noise. This is due to the limited doses of the radioactive agent. A higher dose would cause higher radiation which yields harmful consequences. The most popular representatives are Single Photon Emission Computed Tomography (SPECT) and Photon Emission Tomography (PET).

1.1.3 Magnetic resonance imaging

In contrast to CT and nuclear medicine, magnetic resonance imaging (MRI) uses magnetism and radio waves to generate images. In this work T1 weighted and T2 weighted MR datasets are used. In T1 weighted images fat tissue is depicted lighter than water. In T2 weighted images the tissue behaviour is vice versa. A special MRI technique is the magnetic resonance angiography (MRA) which is used to make pictures of blood vessels. For a more detailed view of the technical features of MRI please refer to Faulkner and Seeram [21].

One advantage of MRI over CT is that it can produce slices oriented in any plane. CT is only capable of producing transaxial slices. Another difference to the CT is that the output of MRI is not calibrated on any scale. The sampling resolution of MRI is either 256 x 256 or 512 x 512 voxels. The spatial resolution lies between 1 and 2 mm. Although MRI has a worse spatial resolution than CT it is more suitable to distinguish between very similar tissues. One huge drawback, especially in the area of segmentation, is that a MRI scanner does not produce constant values over space and time due to inhomogeneities in the magnetic fields.

1.1.4 Ultrasound

Ultrasonography makes use of the different reflections of high frequency sound waves by diverse tissues of the human body. In contrast to CT and the methods of nuclear medicine it is a non-invasive method. Ultrasound images are captured in real time. So it is easy to visualize the movement of human parts. The big disadvantage of ultrasound images is that they typically contain a lot of noise and therefore the identification of objects of interest is often quite hard.

1.2 Segmentation

Yoo [57] stated that segmentation can be simply defined as the partitioning of a dataset into contiguous regions (or sub-volumes) whose member elements (e.g., pixels or voxels) have common cohesive properties.

Segmentation has grown into one of the most important parts in medicine during the past years. The main fields of segmentation in the area of medicine are surgical-planning, diagnosis, therapy-planning and simulation. There are three possibilities how to utilize segmentation:

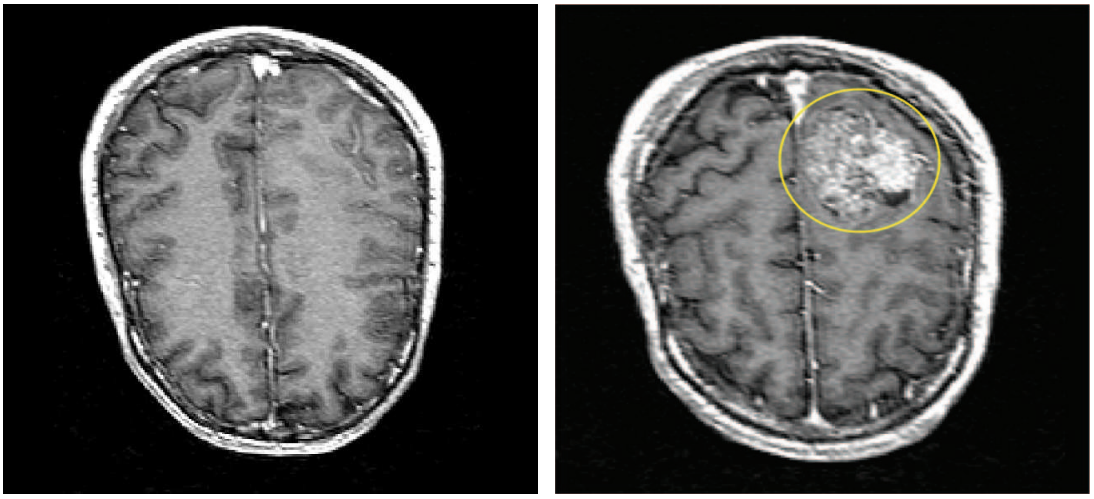
- automatic segmentation
- semi-automatic segmentation
- manual segmentation

Automatic segmentation methods have to accomplish the segmentation task without user interaction. Semi-automatic segmentation allows only a minimal input by the user. For example, at the MICCAI-Challenge [14] one seed point per vessel was allowed as user input. All segmentation methods that require more user interaction belong to the area of manual segmentation.

According to Bankman [4] typical segmentation algorithms either identify all pixels or voxels that belong to an object or detect all pixels or voxels that form the boundary of an object. As both techniques are classification problems many methods of this area are applicable to segmentation. A lot of techniques are available to segment medical images and they can be combined in many ways. Which techniques are used, depends on many factors like image modality, image quality, segmentation task, high or low user interaction. For example, to segment the cranial bone in Figure 1.1a a threshold that divides the image intensities into a class “bone” and a class “no bone” would provide an acceptable solution. On the other hand to delineate the tumour in Figure 1.1b more sophisticated methods or the combination of multiple methods would be necessary to acquire a proper result.

1.3 Deformation

The deformation approach used in this thesis tries to enable the user to deform objects without any constraints as far as it is technically feasible. One approach to deform objects is by deforming the space in which the objects are embedded. This approach was presented by Sederberg



(a) Brain without special characteristics.

(b) Brain with a big tumour (marked by the yellow circle)

Figure 1.1: Two MRI brain scans.

and Parry [46]. This method, based on Bernstein polynomials, enables object deformation by deforming the space the objects are embedded in by changing the position of control points in the coordinate system and then calculating the object shape according to these changes. Space deformation is independent from the object representation and failures in the object's topology (e.g., missing vertices), so it can be used for all kind of objects. The downside is after Botsch et al. [9] that for fine or detailed deformations a large number of control points is necessary which leads to high computational costs.

Another approach is surface deformation which corresponds more to what people expect a deformation to look like. In this case the object deformation is done by directly moving parts of the object (i.e., vertices, triangles, squares ...) within a constant space. Taking Botsch et al. [8] into account a disadvantage of this approach is that the computational effort and numerical robustness increase with the complexity and quality of the surface tessellation. This is because for every deformation a deformation field has to be calculated and the higher the tessellation is the higher is the computational effort to calculate this field. Another disadvantage is that artefacts in the tessellation lead to several problems in the calculation of the deformation. The advantages of surface based deformation compared to space based deformation are that it is easier to do fine and detailed deformations and that it is possible to use topological information in the deformation process. On the ground of these advantages a surface deformation driven approach was chosen for the mesh deformation operations in this thesis.

1.4 Four Dimensional Datasets

In this thesis four dimensional datasets will be mentioned frequently. In the particular case of this paper these datasets consist of at least two three dimensional datasets. These three dimensional

datasets, also called *volumes*, can be either records of the same shape with the same image modality at different moments or records of the same shape with different image modalities at different moments. In the first case the depicted shape mostly changes its form from a 3D dataset to a 3D dataset. In the second case the form most often shows only small variations but depth, colour and reflectance values change due to the different modalities. In the remainder of this thesis the shortcut 3D+T stands for the first case and 3D+M stands for the second one. If four dimensional data are mentioned without further specification, this means that it is not crucial to know which particular type of dataset is used. For both kinds of four dimensional datasets their particular three dimensional datasets need to be registered as no special registration tool was implemented. Registration is the spatial alignment of the volumes.

1.5 Problem Statement

Most automatic and semi-automatic segmentation methods are highly specialized in one segmentation task and need a lot of prior-knowledge on the data. The segmentation results often have to be corrected manually as the algorithms either over- or under-segment the data. Especially in the presence of pathologies a correction of the results has to be done retroactively. The main drawbacks of manual segmentation are that it is very tedious and time consuming, that user knowledge has a very high impact on the results and that it is very hard to reproduce specific results.

Through the increasing usage of 4D Ultrasound Scanners and 4D CT- or MR-scans of the human heart the demand for segmentation tools that are capable of segmenting 4D datasets is growing. A lot of related work segmenting 4D human heart datasets is available but in the area of 4D ultrasound data or when segmenting other organs than the heart, not much research has been done up to now.

Most medical image segmentation programs run on special machines or are hardware accelerated to process the vast amount of data in appropriate time. As radiologists usually work on customary computers they are not able to use these programs at their workstations. An enhancement or solution of this points would save a lot of time in the clinical everyday life and would increase the quality of treatment.

1.6 Aim of the Thesis

The aim of this thesis is to develop a segmentation tool that runs on every customary computer with no hardware acceleration. The tool should fulfil the following requirements:

- Fast manual segmentation of different shapes (organs, tumours etc.) captured by diverse modalities from scratch.
- Manual improvement of 3D and 4D models, that are the output of an automatic or semi-automatic segmentation algorithm and do not fulfil the clinical requirements.

This requires:

- Simple and easy placement and deformation of a mesh.
- Fast parameter adaptation to enable segmentation of new modalities.
- Intuitive user interface of the application.
- User support in the segmentation task with a number of algorithms that run in the background.

1.7 Thesis Overview

Chapter 2 gives an overview of related work. In Chapter 3 the approach of this thesis is explained. Chapter 4 illustrates the methods used to develop a segmentation framework that can deal with multi- and mono-modal three and four dimensional data. The Graphical User Interface is described in chapter 5. Furthermore, chapter 5 describes how the methods of chapter 4 were combined to a complete program and states which values were assigned to the different method variables. The results of some selected segmentation tasks can be seen in chapter 6. Finally chapter 7 provides a conclusion of the work and presents possible future work.

CHAPTER 2

Related Work

2.1 Overview

Section 2.2 gives an overview of state of the art manual segmentation in three and four dimensions and the deformation methods used. As manual segmentation is a comprehensive field of study, from very basic methods to really sophisticated approaches, which interacts with many other areas, it is not possible to cover the whole field within the scope of this section. Therefore only the works that are closely related to this thesis are mentioned and shortly explained in section 2.2.

2.2 Related Work

Not much research has been done on manual segmentation of medical images over the last years. The focus of research is in the area of fully automatic and semi-automatic segmentation. Therefore only a few approaches exist that present complete systems for manually segmenting shapes from scratch. The trend is towards systems for refinement and correction of automatic segmentation results. This section tries to give an overview of state of the art methods and systems in the area of medical image segmentation closely related to this work.

Bornik et al. [7] present a Virtual Reality system for interactive segmentation refinement of the outcome of fully automatic liver segmentation algorithms in 2D and 3D. The system enables the user to deform models with either a free-handed 6 degrees of freedom input device (called the *Eye of Ra*) in combination with a steroscopic display or a tablet PC. The work flow of the program is as follows:

Inspection: The user employs the Eye of Ra to locate errors in the outcome of the automatic segmentation.

Error marking: If errors are found, the user marks them with a specific color. The color indicates how well the model fits the shape of the liver.

Error correction: The marked regions are fixed by the use of special correction tools.

The real time deformation of the model can be done with the use of four correction tools:

Sphere tool: The user can place a sphere inside the dataset and then deform the model with the sphere. This is done in a way that every part of the model that lies inside the sphere is moved out on the shortest possible path.

Plane tool: Works similar to the sphere deformation tool. It can be used to flatten the surface.

Point tool: Single surface points can be moved directly to a desired position. The movement of one vertex exerts a force on the vicinity of the vertex to simulate a natural behaviour.

Template shape tool: This tool can be used to deform larger surface regions. The user marks a region to be deformed. Then a number of contour polylines are drawn by the user. These polylines are used to generate a template shape. Afterwards the tool approximates the model to the template shape.

For the model Bornik et al. [7] use simplex meshes. The tool offers different model rendering modes to simplify the segmentation. The model can be rendered as coloured or textured surface, as wire frame or as outline on a cutting plane. The rendering process makes use of the GPU to achieve real time interactivity.

Dornheim et al. [18] present a semi-automatic method to segment the thyroid cartilage based on *stable mass-spring models (MSM)*. The idea of stable mass-spring models is presented by Dornheim et al. [19]. Based on a manual segmentation of an average larynx a volumetric MSM is built. The user places the initial model in the dataset to be segmented by specifying the position of some distinctive model landmarks in the dataset. Positions and numbers of the landmarks can be specified by the user. After the landmark positioning the rest of the model is adapted automatically by the internal spring and torsion forces of the model. As soon as the user is satisfied with the position of the model the adaptation step can be stopped manually. Subsequently the rest lengths and rest directions of all springs are set to their current value. After this initialisation step the segmentation step is started. In this step the mass points are moved towards gradients and specific intensity values. If the user is content with the segmentation result before the algorithm stops automatically the process can be aborted with a click. During the segmentation step the user can always drag mass points to other positions if they do not coincide with the shape to be segmented.

A user guided segmentation approach for arbitrary organs is proposed by Erdt et al. [20]. The initial mesh is built by using the *Marching Cube Algorithm* on a binary segmentation mask (for a detailed view of the Marching Cube Algorithm please study the work of Lorensen and Cline [35]). This mask can be either the output of a manual segmentation or taken from an organ atlas. The initial mesh is then smoothed and decimated. The user assigns a stiffness constraint to every vertex connection. The stiffness constraint indicates the flexibility of a vertex connection and ranges from 0 (soft connection) to 100 (stiff connection). Placing the mesh in the dataset to be segmented is done by rotating, scaling and translating the mesh until it roughly approximates the shape to be segmented. In the next step the mesh is deformed by aligning the

boundaries of the mesh towards the real boundaries. This is done by manually dragging mesh vertices in a 2D view. The 2D view shows the outline of the mesh on the currently viewed image slice. The force applied on a vertex is propagated to its neighbourhood attenuated by a Gaussian function. During the deformation performed by the user the mesh gets permanently optimized by minimizing an energy function.

A model adaptation method with the use of *Markov Random Fields (MRF)* is presented by Kainmueller et al. [28]. More information about MRF is available in the work of Kindermann and Laurie [31]. The algorithm starts by placing a sphere around every mesh vertex. The sphere, with the centre at the current vertex position, contains a discrete set of uniformly distributed points that represent possible positions where the vertex can be moved to. Then for every possible position the likelihood whether this position lies on a real boundary or not is calculated. Furthermore the distance between all possible deformations is calculated. This distance should serve as a penalty for deformations that may cause intersections. By minimizing a function that rests on the likelihood and the distance mentioned above a displacement field is calculated that is then applied to the mesh vertices.

A hybrid method based on *region growing* and *deformable models* is proposed by del Fresno et al. [16]. In this approach the user places some seed points in the area to be segmented. Emanating from every seed point the region growing algorithm examines all adjacent voxels. If a voxel complies with a similarity criterion based on voxel intensities it is added to the region. The algorithm continues with all neighbours of the added voxel that are not already part of the region until no more voxels directly adjacent to the region fulfil the similarity criterion. In the second step the region growing algorithm is used once again but this time it starts from the frontier voxels of the region generated in the first step and uses a gradient based similarity criterion to better fit the real borders of the image. In the third step a triangle mesh is generated based on the frontier voxels of the region produced by the second region growing step. The last step moves every mesh point in the direction of high gradients and intensities similar to the intensity of the region generated in the first region growing step but with respect to the mesh shape to avoid a distortion of the mesh.

Jackowski and Goshtasby [26] introduce a system for the correction of segmentation errors. Compared with the previous methods this approach uses a *Rational Gaussian (RaG)* surface instead of a mesh to present the shape to be segmented. Further information about RaG can be found in the work of Goshtasby [24]. The RaG surface has a set of control points. If an error in the shape is found a click close to the error initializes a sphere with a specified radius and the centre at the clicked position. All control points within this sphere are selected. By changing the radius of the sphere the user can select more or fewer control points. After all desired control points have been selected the user is able to drag the centre of the sphere. Every control point inside the sphere is moved towards the centre of the sphere scaled by the cosine of the angle between the control point-centre direction and the direction of the mouse motion. Only control points with positive cosines are moved.

Poon et al. [42] present a *3D Livewire* approach to semi-automatically segment objects of arbitrary shape. As the explanation of the Livewire functionality would go beyond the scope of this section the interested reader can find further information in the paper of Mortensen and Barrett [41]. To segment an image the user first performs 2D Livewire segmentations in a couple

of slices in two orthogonal directions. This is done by defining some seed points along the shape to be delineated in the slices to be segmented. This first step produces 2D contour lines. These contours are used to find seed points for the Livewire segmentation in the third direction. The new seed points are sorted according to an order which simulates a sequence that a user would choose to set the seed points. After ordering, these new seed points are used to segment the image automatically in the third orthogonal direction. The result of this algorithm is a set of contour lines in all three orthogonal directions outlining the shape to be segmented. If the user is not satisfied with the result, the procedure that determines the contours in the third direction can be run again with additional manually set contours to refine the result. Three tools for the interactive correction of segmentation results are presented by Kang et al. [29]. The tools are:

Hole filling: This tool allows the user to close holes due to erroneous segmentation by marking the region with the hole with a sphere and then applying either a *mathematical morphology* operator or a volume growing method. The morphological operator closes the hole by using morphological closing (explained by Fisher et al. [22]) with different structuring elements. The volume growing method marks every voxel in the sphere either as inside the hole or outside the hole. This is done by sending rays from every voxel into different directions. If more than 65 % are "reflected" the point is considered to be inside the hole and therefore to be part of the segmentation region.

Point bridging: If some voxels are falsely classified as not to belong to the segmented shape, the user is able to manually classify the voxels and then connect them to the segmentation region with the use of morphological closing.

Surface dragging: The user can drag the surface of an already segmented region by placing a sphere on the surface in such a way that the sphere contains the surface region to be deformed. The centre of the sphere is the control point. By changing the position of the control point the user deforms the surface inside the sphere. This can be done in two ways. One option is to move every surface point parallel to the direction of the control point motion. The strength of the movement depends on an attenuation function. The second option is to use *cubic B-splines*. During the movement of the control point the B-spline is deformed. The deformed surface then takes on the shape of the deformed B-spline. For more information about B-splines please refer to the work of Salomon [44].

Most of the research in the area of 4D medical image segmentation has been done to measure the motion of the human heart, especially the left ventricle. Most approaches make use of a priori knowledge of the motion of the human heart. As examples serve the papers of Gerard et al. [23] and Chandrashekara et al. [13] among many others. A more generic way for segmenting 3D+T images is presented by Montagnat and Delingette [40]. This simplex mesh model based approach looks along every vertex normal of the mesh for edges within a specific distance. If an edge is found, a force tries to pull the vertex towards this edge. To avoid distortions of the mesh shape and to keep a natural motion over time this force is attenuated by spatial and temporal regularization constraints.

A simple approach to segment 4D datasets is to segment the shape in one sub-dataset and to use the point positions as an approximation for the segmentation in the other sub-datasets.

A short overview on this method is given by McInerney and Terzopoulos [37]. A software for manual 3D multi modality image segmentation is presented by Yang et al. [56]. The name of the Software is *"MIASYS"* (Multi Modality Image Analysis System). After having imported the images to be segmented, they need to be registered in the first step. This can be done manually by the user or by an automatic rigid registration algorithm. Before the segmentation the user can optimize the images using different preprocessing tools. Amongst others, MIASYS offers the following preprocessing functionalities: cropping, smoothing and image intensity correction. In using the MIASYS tool, which offers manual and semi-automatic segmentation methods, every image has to be segmented separately. The manual segmentation can be done by delineating the contour or the area of the shape to be segmented with a painting tool. If the user marks the contours only on a few slices, the contours for the slices between can be calculated by interpolation of the existing outlines. Furthermore the tool can display two to three images simultaneously in different color channels to help the user distinguish different structures. The following semi-automatic algorithms are supported by MIASYS: an image intensity threshold method, C-means clustering by Bezdek [6], K-means clustering by Anderberg [3], geodesic active contours by Caselles et al. [11], level sets by Sethian [47] and active contours without edges presented by Chan and Vese [12]. If the user is not satisfied with the segmentation result segmented regions can be smoothed, deleted or combined.

Summary

All these methods and approaches presented above are not capable to solve the problem stated in section 1.5. The approach of Bornik et al. [7] is only able to deal with three dimensional data and needs special hardware that is rarely available. The MSM driven method of Dornheim et al. [18] is very computation intensive and highly specialized as MSM have to be generated for every shape to be segmented. The segmentation approaches proposed by Erdt et al. [20], Kainmueller et al. [28], del Fresno et al. [16], Jackowski and Goshtasby [26], Poon et al. [42] and Kang et al. [29] can be applied only to two and three dimensional datasets. The four dimensional approach presented by Montagnat and Delingette [40] needs a generic model of the shape to be segmented and is not able to segment multi modal datasets. The way McInerney and Terzopoulos [37] tries to segment 4D datasets is too time consuming. This is because no algorithms were implemented that try to align the propagated mesh to the shape to be segmented automatically and so the user still has to align every mesh manually. With the use of the MIASYS tool presented by Yang et al. [56] it is only possible to segment 3D+T datasets. Furthermore most of the segmentation algorithms used in the MIASYS tool need a lot of input parameters that are very hard to understand if the user is not familiar with the specific algorithm. Due to all these reasons, mentioned in the last paragraph, it was necessary to develop the approach presented in this thesis.

CHAPTER 3

Approach of this Thesis

This work presents a tool that enables the user to enhance results of automatic and semi-automatic algorithms and to do fast manual segmentation of shapes of arbitrary topology from scratch. The tool can deal with three and four dimensional image datasets captured by different modalities. The fourth dimension can be either time or result from images of the same shape captured with a different imaging technique. The main emphasis of this thesis is on generality in a way that the methods mentioned below can be used for shapes of arbitrary topology and for all kinds of image modalities. Moreover, the implementation of the program is designed in a way that it can be easily integrated into other applications. The work flow of the program developed in this thesis is as follows: The user loads a dataset that contains the shape to be segmented. Then an arbitrary triangle mesh is placed in this dataset and with simple deformation operations roughly approximated to the desired form. After this initialization the mesh can be fitted to the desired shape with two deformation tools which are similar to the one presented by Erdt et al. [20]. The deformation is done in a 2D view to give the user a better view of the outlines of the mesh. The user clicks on or near to the outlined part that has to be deformed. The point on the outlined part that is the closest to the click is marked as deformation centre. Around the deformation centre a deformation area is highlighted that contains all points that are going to be affected by the deformation. The size of the deformation area can be interactively changed by the user. With the mouse the user drags the deformation centre to a desired position. The other points in the deformation area follow this motion weakened by an attenuated function. After the user has dropped the deformation centre, an algorithm searches for an edge within an area around the deformation centre. If an edge is found, the algorithm looks in 3D for edges in the vicinity of every moved mesh point and moves these points to this edge if it is within a specific region and fulfils some similarity criteria. This algorithm is called Sticky Edges algorithm. After every deformation step the mesh is smoothed to warrant a good mesh quality. If a 4D dataset has been loaded in the first step the deformations performed in one sub-dataset can be propagated to the other sub-datasets as a starting point for their segmentation. These propagation can be done in two ways. One approach records the user interactions and propagates these interactions. The second approach sets an already deformed mesh into other sub-datasets as starting point for

their segmentation. These 3D and 4D deformation steps can be repeated until the user thinks that the mesh fits the underlying shape well. The program is designed in a way that it runs on a common workstation without the need of hardware acceleration. Except for the propagation of the deformation to the fourth dimension all working steps are performed in real time so that the user has no interruption of his work flow.

CHAPTER 4

Implemented Methods

4.1 Overview

This section describes all basic methods implemented in this thesis. This embraces the methods that are used in the final implementation as well as the methods that were tried out but later on discarded because it turned out that they were not suitable to solve the problem stated in section 1.5. Finally section 4.15 shows how the methods were put together to the algorithm used in the chapters 5 and 6. For this purpose, a pseudo code snippet is given after every method that illustrates shortly the functionality of the method. These snippets are called *modules* and are put together in section 4.15 to give a good overview over the program flow. A typical module looks as follows:

1 **methodName**[inputParameters x]
  1: ...*code*
  2: **return** returnValue

Before going into detail a few variable definitions have to be given. Table 4.1 gives an overview of the variables used in this section and their meaning.

4.2 Surface Representation

Surface representation methods can be roughly divided into parametric and implicit surface representations. Implicit surfaces are described by a function F. For every point $\mathbf{x}$ on the surface $F(\mathbf{x}) = 0$ is true. Implicit surfaces are well suited for querying if a point lies on, inside or outside a surface, ray tracing applications and the detection respectively avoidance of self intersections. Parametric surfaces can be expressed by a function F that maps two dimensional parameters to a three dimensional surface. A point $\mathbf{x}$ on the surface can be expressed by $\mathbf{x} = F(u, v)$ where u and v are surface parameters. They have the advantage that several problems on the

Variable	Meaning
θ:	an angle
u, x, y:	three auxiliary variables; their meaning can be seen from the content or is explained separately
u, x, y:	three auxiliary vectors; their meaning can be seen from the content or is explained separately
$R_{\theta u}$:	rotation matrix for angle θ around a given axis u
∇:	a gradient
σ:	standard deviation
$\|\mathbf{u}\|$:	Euclidean norm of an arbitrary vector u
ω:	a scalar weight to scale a tuple
v:	an arbitrary vertex
v_i:	a vertex with index i
V:	a set of vertex indices
$\mathbf{v}_i$:	position of a vertex with index i
$val(v_i)$:	valence of a vertex with index i
$\mathbf{n}_{v_i}$:	normal of a vertex with index i
$d(v_i, v_i)$:	Euclidean distance between the vertex with index i and the vertex with index j
K_i:	set of the indices of the vertices adjacent to the vertex with index i (one ring neighbourhood)
e_{ij}:	an edge from a vertex with index i to a vertex with index j
E:	a set of edges
t_i:	a triangle with index i
$a(t_i)$:	area of a triangle with index i
$\mathbf{n}_{t_i}$:	normal of a triangle with index i
T_{v_i}:	a set of triangles incident on v_i
$T_{e_{ij}}$:	a set of triangles incident on e_{ij}
c:	deformation centre
$\mathbf{n}_c$:	normal of the deformation centre
$\mathbf{d}_i$:	deformation vector for a vertex with index i
k:	kernel size
$size(S)$:	number of entries in a set S

Table 4.1: Variables used in this work with their meaning.

surface can be reduced to the lower dimensional parameter space Botsch et al. [8]. A discrete representation of parametric surfaces are triangle meshes. A triangle mesh is a set of triangles where each triangle defines a part of the whole surface. The advantage of triangle meshes is that shapes of arbitrary form and detail can be modelled. The shorter the edges of the triangles are the more details can be expressed. The disadvantage is that the more triangles a mesh has, the more memory is needed and the longer is the computation time for algorithms. Another advantage is that algorithms are very convenient to implement and geometric mesh properties like normals

can be computed easily. Due to this advantages the triangle mesh structure was chosen to be the proper surface representation for the subject-matter of this thesis.

4.3 Bounding Box Calculation

If algorithms with very high memory consumption have to be executed or too little memory is available, it is necessary to run the algorithm in a bounding box. The bounding box includes only relevant parts of a dataset which saves computation time. In the program that was developed in the course of this work the bounding box method is applied in combination with the edge detection algorithm as only the detection of edges in a dataset in the vicinity of the mesh is relevant. The bounding box serves the purpose of saving computation time because it would be too time consuming to calculate the edges for the whole dataset. The length of the box is defined as 1.5 times the maximal extent of the mesh in x-direction, the height is 1.5 times the maximal mesh extent in y-direction, the depth is 1.5 times the maximal extent in z-direction and its centre is at the centre of the mesh. The *maximal extent* in a particular direction is calculated by using the difference between the direction coordinate of the vertex with the lowest value and the vertex with the highest value in that direction. The factor 1.5 was chosen as it turned out during implementation and testing that a deformation of the mesh more than 1.5 times its own size is barely necessary and so the recalculation of the edges due to a resizing of the bounding box is only rarely required.

The module for the bounding box calculation is defined in algorithm 4.1.

1 **calculateBoundingBox**[mesh m]
1: $setBoundingBoxCentre(m.getCentre())$
2: $setBoundingBoxXLenght(m.getExtentX())$
3: $setBoundingBoxYLenght(m.getExtentY())$
4: $setBoundingBoxZLenght(m.getExtentZ())$
5: **return** boundingBox

Algorithm 4.1: Bounding box calculation module

4.4 Edge Detection

Edges are one of the most important properties in an image, as they determine where one area ends and another begins. This makes edge detection one of the most crucial fields in relation to image segmentation. Edge detection algorithms find edges by searching for sharp changes in the image brightness. According to Barrow and Tenenbaum [5] these changes in brightness are due to depth-, colour-, orientation- or reflectance changes in the real world. The perfect outcome of an edge detection algorithm would be an image with closed lines that indicate all relevant edges. In medical image processing this is not possible for several reasons. Some of these reasons are:

Noise: Small artefacts in the image lead to small edges that do not exist in the real world. These edges are called *false edges*. Especially in ultrasound images noise is a big problem.

Low resolution: A low image resolution contributes to poor edge perceptibility.

Bad contrast: Also bad contrast (i.e., due to poor illumination) makes it harder to differentiate between edges and areas.

As this thesis does not specialize on one particular image type, two basic edge detection algorithms for grey scale images were implemented. These are the *Canny* and the *Marr-Hildreth* operator.

4.4.1 Canny Edge Detection

The Canny edge detection operator was developed by Canny [10] in the year 1986. According to Canny an edge detection algorithm has to fulfil the following points:

- Detect as many real edges as possible.
- The distance between the real edge and the pixels marked as edge should be minimal.
- The algorithm should only respond to an edge once.

The Canny operator consists of five successively performed steps. The first step tries to filter out any noise by convolving the image with a Gaussian filter kernel. The higher the standard deviation of the kernel is chosen the lower is the sensitivity of the filter to image noise. A deeper look into convolution is given by Fisher et al. [22].

In the second step the gradients of the image are measured with the use of the *sobel* operator. The sobel operator convolves the image with two 3x3 convolution masks. One mask (figure 4.1a) estimates the first derivative in the vertical direction and the other mask (figure 4.1b) estimates the first derivative in horizontal direction. The total gradient is the summation of the first derivatives in horizontal and vertical direction. Step three calculates the edge direction with the formula:

$$\theta = \arctan(\frac{\nabla_h}{\nabla_v}) \tag{4.1}$$

where θ is the direction of the edge, ∇_h the first derivative in horizontal direction and ∇_v the first derivative in vertical direction. If one of the two derivatives is zero, the value of the zero derivative is set to 1 in the above equation. The Canny operator uses only four edge directions in an image (see figure 4.2). Therefore θ is assigned to one of these directions (either 0, 45, 90 or 135 degrees). Step four is called *nonmaximum suppression*. In this step every pixel value, where the value is lower than the value of one of its directly adjacent neighbours in gradient direction is set to zero. For the fifth and last step the Canny operator needs two thresholds defined by the user. A higher one $T1$ and a lower one $T2$. Every pixel in an image that has a value greater than $T1$ is marked as edge point. Then, every directly adjacent neighbour of an edge point that has a value greater than $T2$ is marked as edge. This is repeated until there are no edge points left with neighbours with a value greater than $T2$. The purpose of the second threshold $T2$ is to overcome noise caused by lower edge intensities in particular areas. The Canny edge detection operator needs four input parameters:

- the size of the Gaussian kernel,
- the standard deviation of the Gaussian kernel,
- threshold $T1$,
- and threshold $T2$.

The module for the Canny edge detection can be seen in algorithm 4.2.

1 **calculateEdgesForImgWithCanny**[kernelSize k, standardDeviation σ, threshold T1 and T2]

1: $arrayWithEdgeIndices = canny(k, \sigma, T1, T2)$

2: **return** arrayWithEdgeIndices

Algorithm 4.2: Canny edge detection module

-1	-2	-1
0	0	0
1	2	1

(a) vertical operator

-1	0	1
-2	0	2
-1	0	1

(b) horizontal operator

Figure 4.1: Sobel operator masks.

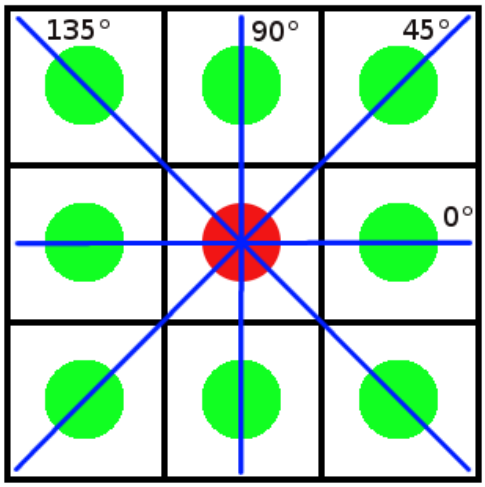


Figure 4.2: The green dots mark the pixels that surround the pixel marked with a red dot. The blue lines determine the four possible directions an edge can have originating at the centre pixel.

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 4.3: LoG filter kernel with a standard deviation of 1.4.

4.4.2 Marr-Hildreth Edge Detection

The Marr-Hildreth operator, also know as *LoG* (Laplacian of Gaussian) was presented in 1980 by Marr and Hildreth [36] and makes use of the second spatial derivation of the image intensities to localize edges. The second spatial derivation is calculated by convolving the image with a Laplacian filter kernel. Calculating the edges only by convolution of the image with this kernel is very noise sensitive. Therefore the image is smoothed with a Gaussian filter kernel before. To save computation time first the Gaussian kernel is convolved with the Laplacian kernel and then the resulting kernel is convolved with the image. This can be done because convolution is associative. The continuous 2D version of the LoG is

$$LoG_{(x,y)} = -\frac{1}{\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{x^2+y^2}{2\sigma^2}\right) \tag{4.2}$$

with the Cartesian coordinates x and y. A discrete approximation with kernel size 9 and standard deviation 1.4 can be seen in figure 4.3.

Zero values in an image that was convolved with a LoG filter kernel correspond to constant intensity regions in the original image. A positive value with an adjacent negative value corresponds to an intensity change from bright to dark and a negative value followed by a positive matches with an intensity change from dark to bright. A zero crossing detector finds all zero crossings in an image that correspond to an edge. Zero crossings that belong to an edge in the real image need a short distance between the positive and the negative peak as long distances indicate a smooth transition from dark to bright or vice versa and the magnitude of the peak and the valley must be over respectively under a given threshold because small fluctuations around zero can be due to noise. The LoG edge detection operator needs three input parameters:

- the size of the Gaussian kernel,
- the standard deviation of the Gaussian kernel,
- and a zero crossing threshold.

The module for the LoG edge detection can be seen in algorithm 4.3.

1 calculateEdgesForImgWithLoG[kernelSize k, standardDeviation σ, threshold zc]

1: $arrayWithEdgeIndices = log(k, \sigma, zc)$
2: **return** arrayWithEdgeIndices

Algorithm 4.3: LoG edge detection module

4.4.3 Conclusion

The Canny edge detector has the advantages that it produces edges with very few gaps because of the two thresholds, it does not create many isolated line segments and it is very noise resistant. The disadvantage of the Canny edge detector is that it needs more computation time and a higher memory consumption than the LoG operator. The benefit of the LoG edge detector is that it is fast to compute and that it finds more edges than the Canny edge detector. Unfortunately the detection of more edges yields also more false edges and isolated edge segments. Another disadvantage of the LoG filter is that it is more noise sensitive than the Canny operator. The output of both operators depends extremely on parameter tuning.

As the Canny edge detector performs better on noisy images a noise estimation method, proposed by Zambal [59], was implemented to decide, depending on its outcome, whether the Canny edge detector or the LoG edge detector should be used. The noise estimator measures the absolute grey value difference of a pixel to its next neighbours in vertical respectively horizontal direction. The difference is measured 10 times in horizontal and vertical direction in every image slice. Subsequently, the measured differences are summed and normalized. Although the noise estimator provides good results, the LoG edge detector was chosen to be the standard edge detection method. This was done because the LoG algorithm needs less computation time, consumes much less memory and performs good on images without a high noise level. Additionally the computation time of the noise estimator is saved. If the LoG algorithm does not produce a satisfying result the user still can either tune the algorithm parameters or choose the Canny edge detector manually.

4.5 Edge Classification

The edge classification algorithm is based on the gradient direction of the edges and how well the edges are connected. For every pixel of the image it determines whether the pixel belongs to an edge or not. If the pixel is an edge pixel a 3D structure element is centred at the pixel position. The structure element is a cube with 5x5x5 pixels with a hole at its centre. The specific element size was chosen because it is a good trade off between computation time and the bridging of edge gaps. If an already classified pixel with a gradient direction difference lower than or equal to an angle of $45°$ lies within the structure element, the edge pixel is assigned to the same class, otherwise the edge pixel is assigned to a new class. If there are two or more pixels with a similar gradient but different classes, these classes are merged. Figure 4.4 shows an image with classified edges. The module for the edge classification is defined in algorithm 4.4.

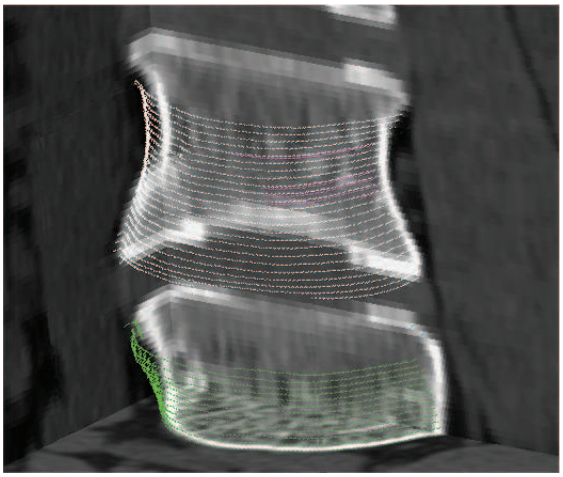


Figure 4.4: The classified edges within a bounding box of two vertebral bodies. The colours indicate the edge class. The pink lines that can be seen through the white ones belong to an edge class inside the vertebral body.

classifyEdges[image img, edgeIndexArray eIdx]

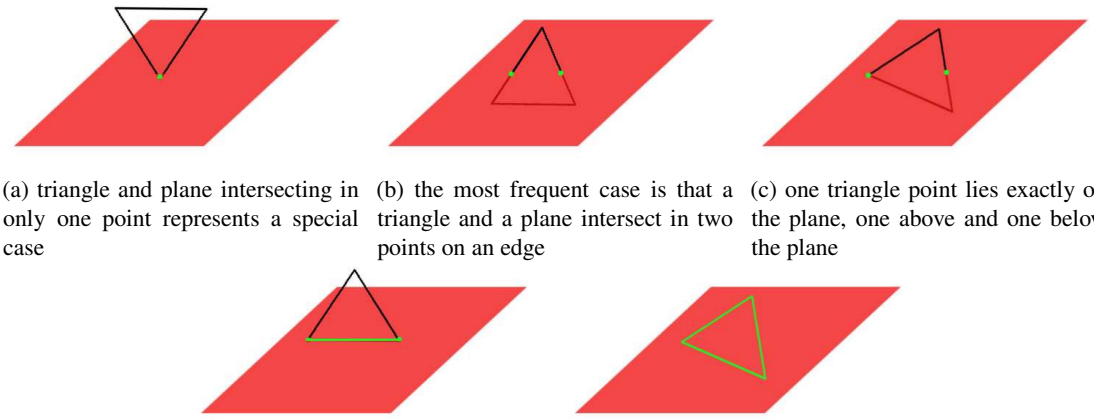
1: $gradients = calculateGradients(img, eIdx)$
2: $edgeIndexToEdgeClassMap = classify(eIdx, gradients)$
3: **return** edgeIndexToEdgeClassMap

Algorithm 4.4: Edge classifier module

4.6 2D Cut

To depict the outline of a mesh in a specific image slice a "cut" of that image slice through the mesh is calculated. The points where the mesh intersects the image slice, are connected with lines to approximate the outline. This is done by checking for every triangle of the mesh if its edges intersect the mesh. Figure 4.5 shows five ways a triangle could intersect a plane. The case shown in figure 4.5a is not considered by this algorithm because neighbouring triangles share the same intersection point but have an additional second intersection point. Also 4.5e is discarded because it is easier to work with the neighbouring triangles that conform to the case depicted in figure 4.5d. So only the cases pictured in the figures 4.5b, 4.5c and 4.5d have to be considered. If two edges of a triangle intersect the image slice the intersection point coordinates and the edge indices are stored in a *doubly linked list*. In the case shown in figure 4.5d the end points of the line that lies exactly on the plane are stored as intersection points. The corresponding edge indices depend on the iteration order (first come, first serve) but one edge index can only be stored with exactly one intersection point.

A doubly linked list is a special list type that stores the successor and ancestor for every list entry. In this case one edge is the successor respectively the ancestor of another edge and vice versa. To find the ancestor respectively the successor is very easy because one intersection point is at least part of two triangles that intersect the slice, as every edge has two adjacent triangles.



(a) triangle and plane intersecting in only one point represents a special case

(b) the most frequent case is that a triangle and a plane intersect in two points on an edge

(c) one triangle point lies exactly on the plane, one above and one below the plane

(d) two triangle points lie exactly on the plane

(e) all triangle points lie exactly on the plane.

Figure 4.5: The green points respectively the green lines mark the intersection points at which the black triangle crosses the red plane.

Furthermore the edge indices and the intersection point coordinates are indexed in the list, so an element can be accessed by the use of an edge index or an intersection point coordinate. This type of linked list can be traversed extremely fast and single entries can be found very easily. Currently the algorithm handles only closed surfaces, but it is no problem to extend it to be able to handle also open surfaces. The module for the calculation of the 2D Cut can be seen in 4.5.

1 **calculate2DCut**[mesh m, cuttingPlane plane]

1: $intersectionPoints = getAllEdgeIntersectionPoints(m, plane)$
2: $doublyLinkedList = connectIntersectionPoints(intersectionPoints)$
3: **return** doublyLinkedList

Algorithm 4.5: 2D cut calculation module.

4.7 Manual Mesh Transformation and Deformation

To adjust a mesh in a way that it approximates the shape to be segmented as good as possible the user can utilize three transformation methods and two deformation methods. The transformation methods are translation, scaling and rotation. These operations affect the whole mesh. It is not possible to translate, deform or scale only single parts of the mesh. To transform a mesh every mesh vertex position tuple is multiplied with a transformation matrix or a combination of them. All used transformation matrices are depicted in figure 4.6. The module for the transformation of a mesh is defined in algorithm 4.6.

As the transformation methods are only used to align the mesh to the shape to be segmented, the module is kept very simple. To deform a mesh a *deformation vector* **d** is applied to a deformation centre **c**. The deformation vector points in the direction where the deformation centre

1 **transform**[]

1: $alignedMesh = alignMeshRoughlyToShapeToSegment()$

2: **return** alignedMesh

Algorithm 4.6: Transformation module

1	0	0	t_x
0	1	0	t_y
0	0	1	t_z
0	0	0	1

(a) translation matrix

s	0	0	0
0	s	0	0
0	0	s	0
0	0	0	1

(b) scaling matrix

1	0	0	0
0	$\cos\theta$	$-\sin\theta$	0
0	$\sin\theta$	$\cos\theta$	0
0	0	0	1

(c) rotation matrix for rotation around x-axis

$\cos\theta$	0	$\sin\theta$	0
0	1	0	0
$-\sin\theta$	0	$\cos\theta$	0
0	0	0	1

(d) rotation matrix for rotation around y-axis

$\cos\theta$	$-\sin\theta$	0	0
$\sin\theta$	$\cos\theta$	0	0
0	0	1	0
0	0	0	1

(e) rotation matrix for rotation around z-axis

Figure 4.6: Transformation matrices for translation, scaling and rotation. t_x, t_y and t_z give the translation of the mesh in x-, y- and z-direction. s is the scale factor. θ is the rotation angle.

is pulled and the length of the deformation vector indicates how far the centre is moved. The deformation centre can be any arbitrary point on the surface of the mesh. With the use of a given *kernel size* a deformation area is calculated. The deformation area contains all mesh vertices that have a distance value lower than the kernel size to the deformation centre.

When the deformation centre is moved to its new position with the use of the deformation vector all vertices within the deformation area also change their positions. How the vertices change their positions depends on which deformation method and which attenuation function is chosen. The two selectable deformation methods are described subsequently.

4.7.1 Simple Deformation

The first deformation method is a very simple one. If the angle between the normal of the deformation centre and a the normal of a vertex within the deformation area in their initial position is below 90° the direction vector is scaled by an attenuation function and applied to the vertex, otherwise the vertex stays unchanged. Checking of the angle is necessary to avoid that, if a part of a mesh with a very low inner diameter is deformed, the whole part is moved in the direction of the deformation centre because the kernel size is big enough to include the vertices of the whole part. For an illustration see figure 4.7. As can be seen, an enlargement of the process diameter is only possible with the angle constraint. In contrast to the method presented in the next subsection this method does not check for possible self-intersections during the deformation. Self-intersections occur very rarely and are, except of some extreme cases, caused by the user. The module for the simple deformation is shown in algorithm 4.7.

1 **deformSimple**[deformationVector dv, deformationCentre dc, kernelSize k]

1: $deformationArea = calculateDeformationArea(dc, k)$
2: $deformDeformationCentre(dv, dc)$
3: **for** $j = 0 \rightarrow deformationArea.size()$ **do**
4: $vertex = getVertx(j)$
5: $weight = getGaussWeight(dc, vertex, k)$
6: **if** $normalAngle(dc, vertex) < 90°$ **then**
7: $deformVertex(vertex, dv, weight)$
8: **end if**
9: **end for**
10: **return** deformedMeshVertices

Algorithm 4.7: Simple deformation module

4.7.2 Deformation Using the Normals

The second deformation method calculates the angle θ between the normal of the deformation centre and the deformation vector of the undeformed mesh. The new position $\mathbf{v}_i'$ of vertex with the index i is calculated by:

$$\mathbf{v}_i' = \mathbf{v}_i + \|\mathbf{d}\|\omega_{id}\omega_{in}R_{\theta u}\mathbf{n}_{v_i} \tag{4.3}$$

where

$$\omega_{in} = \mathbf{n}_{v_i} \cdot \mathbf{n}_c. \tag{4.4}$$

and ω_{id} denotes the weight of an attenuation function which will be discussed below. $R_{\theta u}$ is the rotation matrix for axis u for angle θ. In the program implemented in the course of this thesis u is the axis orthogonal to the slice with that the 2D Cut has been calculated. This deformation method moves vertices along their normal, which keeps the shape of a region during the deformation.

Note that equation 4.3 can only be applied to a convex surface region. In the case of a concave surface this deformation would cause self-intersections. For an illustration see figure 4.8. Therefore the algorithm checks the shape of the surface before the deformation starts. This is done by checking the normal of the deformation centre and the normals of all vertices in the deformation area. If the normal of the deformation centre and the normal of a vertex in the deformation area point into directions that cross each other the vertex would be moved with the first deformation method otherwise equation 4.3 would be applied. The movement of vertices with a normal that points in a direction that differs from the deformation centre normal by more than $90°$ is also suppressed in this method.

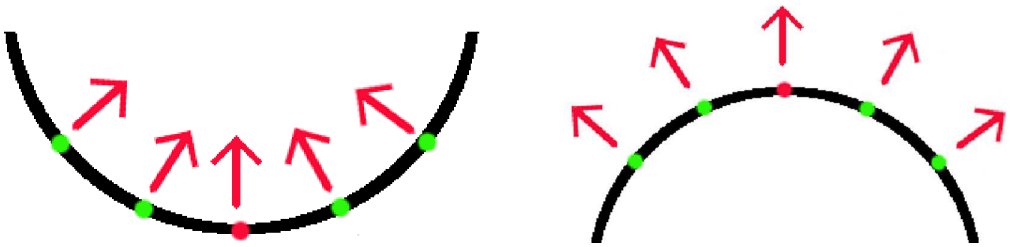
The second deformation method has the advantage that the shape of the region to be deformed is approximately kept as all vertices in the region move along their normals. The deformation of the mesh with the first method on the other hand correlates more with a deformation that is expected by the user when a point is moved and is also much faster to computate. The final module for the deformation using normals is shown in algorithm 4.8.

(a) outline of a vertebra unchanged

(b) outline changed without angle constraint

(c) outline changed with angle constraint

Figure 4.7: (a) the unchanged outline part of the vertebra can bee seen. (b) shows the outline after a deformation without constraint. (c) shows the outline after a deformation of the right process with the constraint that the normals of a vertex and the deformation centre must not differ by more than $90°$. The yellow part of the outline marks the deformation area. The red arrows indicate the direction of the movement of the yellow outline parts.



(a) in the case of a concave shape this deformation method causes self-intersections

(b) as all points move in the direction of their normals the shape is preserved over deformation

Figure 4.8: The red point marks the deformation centre. The green points are the vertices in the deformation area. The red arrows indicate the movement of the points. This illustrates the direction of the movement of the points. The attenuation is not depicted.

1 **deformWithNormals**[deformationVector dv, deformationCentre dc, kernelSize k]

1: $deformationArea = calculateDeformationArea(dc, k)$
2: $deformDeformationCentre(dv, dc)$
3: **for** $j = 0 \rightarrow deformationArea.size()$ **do**
4: $vertex = getVertx(j)$
5: $weight = getGaussWeight(dc, vertex, k)$
6: **if** $normalAngle(dc, vertex) < 90°$ **then**
7: **if** $isConvex(dc, vertex)$ **then**
8: $rotateDV(dv)$
9: $deformVertex(vertex, dv, weight)$
10: **else**
11: $deformSimple(dv, dc, k)$
12: **end if**
13: **end if**
14: **end for**
15: **return** deformedMeshVertices

Algorithm 4.8: Deformation using the normals module

Attenuation Functions

Two attenuation functions were implemented and tested in the course of this work. Both of them are distance dependent. The first one uses a Gaussian function with kernel size k. The weight by which the deformation vector is scaled for a vertex i is defined as:

$$\omega_{id} = \frac{gauss\left(\|\mathbf{c} - \mathbf{v}_i\|\right)}{gauss\left(0\right)} \tag{4.5}$$

where

$$gauss\left(x\right) = \frac{1}{\frac{k}{3}\sqrt{2\pi}} e^{\frac{-x^2}{2\left(\frac{k}{3}\right)^2}}. \tag{4.6}$$

With this parametrization the deformation vectors applied to the vertices at the border of the region to be deformed is downscaled to nearly zero. The second attenuation function is a cubic one. The weight is calculated by:

$$\omega_{id} = cubic\left(\frac{\|\mathbf{c} - \mathbf{v}_i\|}{k}\right) \tag{4.7}$$

where

$$cubic(x) = 2x^3 - 3x^2 + 1 \tag{4.8}$$

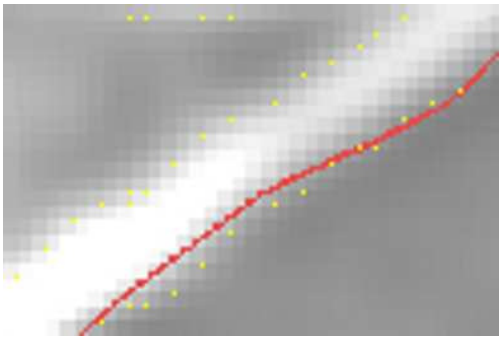
Figure 4.9: The green line describes the Gaussian function in equation 4.5 with a kernel size 1. The red line depicts the cubic attenuation function in equation 4.8. Only the parts of the attenuation functions that are equal to or lie between 0 and 1 on the x-axis are used for attenuation, so strange behaviour due to negative values is excluded.

Both attenuation functions are depicted in Figure 4.9. Only the parts of the attenuation functions that are equal to or lie between 0 and 1 on the x-axis are used for attenuation. As can be seen, the used part of the cubic function is similar to a linear attenuation and barely dampens the point movement in the outer regions of the deformation area compared to the Gaussian function. During the development and evaluation of this work it turned out that the Gaussian attenuation function corresponds more to the expectations of the user how the surface is going to be deformed. Therefore the Gaussian function was chosen to be the standard attenuation function.

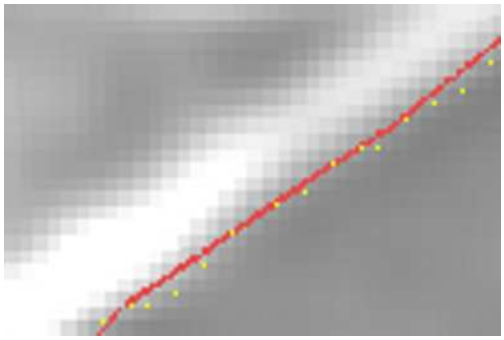
4.8 Sticky Edges

Often the deformation centre is repositioned precisely on an edge or in the immediate neighbourhood of an edge by the deformation vector but the mesh vertices in the vicinity of the deformation centre are not deformed in such a way that they also lie on that edge. This is because the shape of the attenuation function does not fit the edge shape. The Sticky Edges algorithm was implemented to avoid that every vertex has to be moved on the edge separately. A prerequisite for this algorithm is that the edges have been classified with the method explained in section 4.5. The Sticky Edges algorithm is called after the deformation centre has been placed at a specific position and the vertices in the deformation region have been moved to their new positions. If the deformation centre was placed directly on an edge, the class of this edge is memorized. Otherwise the path from the new position of the deformation centre to its old position is examined. For the first edge found at a position e a distance variable δ is calculated with the use of equation 4.9. If δ is bigger than the length of the vector from e to the new deformation centre, the edge class of the edge is memorized and the algorithm proceeds with the following step. If no edge is found the algorithm terminates.

After the edge class of the deformation centre has been determined the algorithm calculates, starting from the new vertex positions of the deformation area, whether there is an edge along the direction of the deformation vector of a vertex or its opposite direction or not. If an edge is



(a) deformation with the Sticky Edges algorithm deactivated



(b) deformation with the Sticky Edges algorithm activated

Figure 4.10: (a) result of a deformation without the use of the Sticky Edges algorithm. (b) result of the same deformation with the use of the Sticky Edges algorithm. The yellow dots indicate the edge of the shape to be segmented. The red line is a part of the outline of the mesh at the currently viewed slice. As can be seen the line in (b) fits the edge much better than the line in (a).

found within the length of the deformation vector of a vertex that has the same edge class as the edge the deformation centre lies on, the simple deformation method explained in 4.7.1 is used to move the vertex to the position of the edge. A kernel size appropriate for this small movement is double the distance between the edge position and the vertex position. The distance variable δ is calculated with the following formula:

$$\delta = \frac{l(M_x)\frac{d(\mathbf{c}_x, e_x)}{d_{sum}} + l(M_y)\frac{d(\mathbf{c}_y, e_y)}{d_{sum}} + l(M_z)\frac{d(\mathbf{c}_z, e_z)}{d_{sum}}}{u} \tag{4.9}$$

where

$$d_{sum} = d(\mathbf{c}_x, e_x) + d(\mathbf{c}_y, e_y) + d(\mathbf{c}_z, e_z). \tag{4.10}$$

$l(M_x)$, $l(M_y)$ and $l(M_z)$ are the extent of the mesh in x-, y- and z-direction. $d(\mathbf{c}_x, e_x)$, $d(\mathbf{c}_y, e_y)$ and $d(\mathbf{c}_z, e_z)$ are the differences between the x-, y- and z-coordinates of the new deformation centre position and the position e. With the use of u the maximal feasible distance of an edge to the deformation centre can be controlled. The higher the value of u is chosen the lower δ is. During the implementation, experience has shown that a value of 5 for u produces good results. Figure 4.10 shows the outcome of a deformation with and without the use of the Sticky Edges algorithm. The module for the Sticky Edges algorithm is outlined in algorithm 4.9.

4.9 Interpolation between Slices

In some datasets the distance between different image slices is very big and the kernel size chosen for a deformation is often not large enough to reach from one slice to the next one. These

1 **stickyEdges**[deformationArea da, deformationCentre dc]

```
1:  if dc.isOnEdge then
2:      edgeClass = dc.getEdgeClass
3:  else
4:      edgeClass = dc.searchForEdgeClass()
5:      if edgClass.isNotWithinRange() then
6:          break
7:      end if
8:  end if
9:  for vertexv : da do
10:     edge = v.searchForEdge()
11:     if edge.isWithinRange()ANDedge.edgeClass = edgeClass then
12:         k = 2 * (edge.position − v.position)
13:         dv = edge.position − v.position
14:         deformSimple(dv, v.positon, k)
15:     end if
16: end for
17: return deformedMesh
```

Algorithm 4.9: Sticky Edges module

extreme cases, with slice distances of more than 1 centimetre, can cause "valleys" between adjacent slices. This phenomenon can be seen in figure 4.11a. In this particular case the user tried to segment the shape in two adjacent slices but the kernel size was not big enough to reach the next slice.

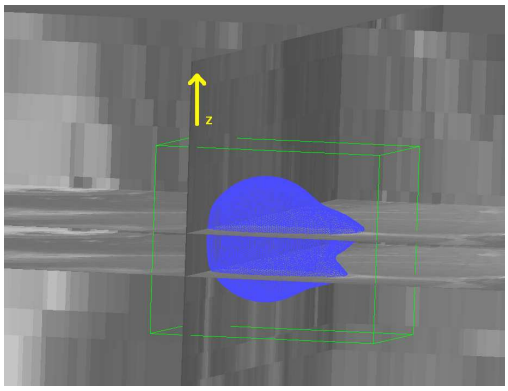
To eliminate these "valleys" by raising the bottom them, an interpolation method was implemented. The method needs at least three vertices as input. One vertex whose position needs to be interpolated, and at least two points as sampling vertices. The sampling vertices are used to determine the new position of the vertex to be interpolated. The deformation vector for the vertex to interpolate with index i is calculated by:

$$\mathbf{d}_i = \left(\sum_{\forall j \in V} \mathbf{v}_j * \omega_j \right) - \mathbf{v}_i \tag{4.11}$$

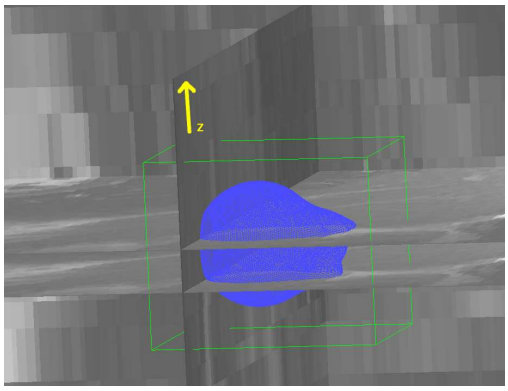
where

$$\omega_j = \frac{\left(\sum_{\forall s \in V} d(\mathbf{v}_s, \mathbf{v}_i)^2 \right) - d(\mathbf{v}_j, \mathbf{v}_i)^2}{\left(\sum_{\forall s \in V} d(\mathbf{v}_s, \mathbf{v}_i)^2 \right) (n-1)}. \tag{4.12}$$

V contains the indices of all sampling vertices and n is the number of sampling vertices. The weights ω_j are normalized. The benefit of the weight function in equation 4.12 is, that points with shorter distance to the vertex to be interpolated have a higher impact on the deformation vector. If the distance of all sampling vertices to the vertex, whose position needs to be interpolated, is equal, the position of the vertex whose position needs to be interpolated is the average



(a) a sphere with a "valley" between two adjacent slices



(b) the "valley" nearly vanished after interpolation

Figure 4.11: The slice distance in z-direction of the dataset depicted is 1.12 centimetre. Therefore a "valley" in the mesh occurred during the segmentation process. (a) shows the mesh before, (b) shows the mesh after the vertex position interpolation.

of all sampling vertices positions. After the deformation vector has been calculated it is applied to the vertex to be interpolated with the use of the deformation method mentioned in subsection 4.7.1. The kernel size for the deformation is calculated by:

$$k = \max\left(d\left(\mathbf{v}_i - \mathbf{v}_y\right)\right) \tag{4.13}$$

with $y \in V$. The maximum of the absolute distances between the vertex to be interpolated and the sampling vertices is taken as kernel size. The module for the interpolation can be seen in algorithm 4.10.

1 **interpolate**[vertexToInterpolate v, samplingVerticesArray sv]

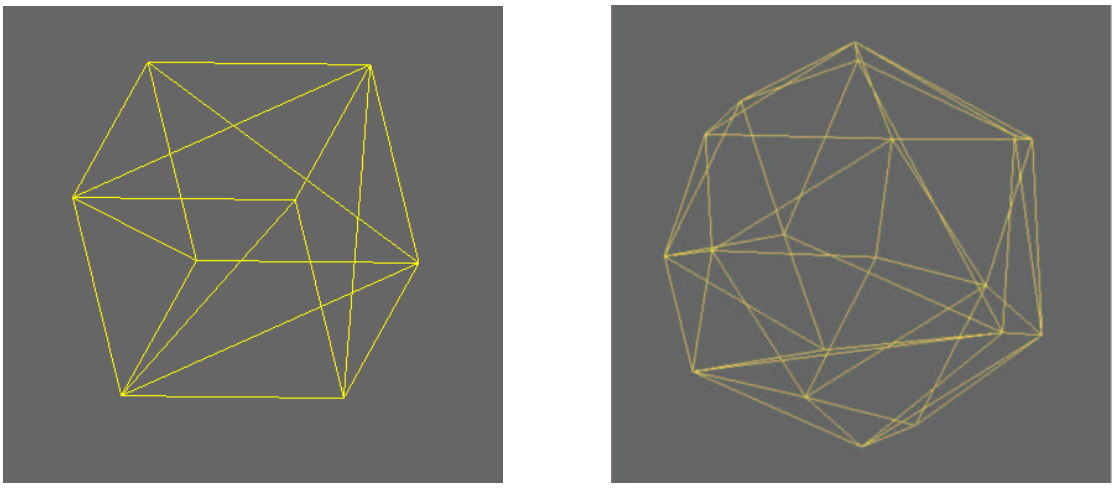
```
1:  for j = 0 → sv.size() do
2:      vertex = getVertx(j)
3:      distance = getDistance(v, vertex)
4:      distanceMap.put(distance)
5:      totalDistance+ = distance
6:  end for
7:  for entry : distanceMap do
8:      weight = (totalDistance − entry)/(totalDistance ∗ (dinstanceMap.size() − 1)
9:      weightSum+ = weight
10:     pos = entry.getVertexPos.scale(weight)
11:     vertexToInterpolatePosNew+ = pos
12: end for
13: deformVector = vertexToInterpolatePosNew − v.getPosition()
14: deformSimple(deformVector, v, max(distanceMap))
15: return deformedMeshVertices
```

Algorithm 4.10: Interpolation module

4.10 Subdivision

Subdivision is used as a method to refine meshes. It is needed if a mesh is too coarse to align it smoothly to the shape to be segmented. It splits the polygons the mesh is composed of into smaller ones and adjusts the old and new vertex positions according to a smoothing criterion. This step is repeated until the desired smoothness level is reached and a smooth alignment is possible If only a part of the mesh is subdivided this is called adaptive subdivision. Figure 4.12 shows a simple triangle mesh before and after a $\sqrt{3}$-subdivision. As subdivision algorithm the $\sqrt{3}$-subdivision presented by Kobbelt [32] was chosen due to the following reasons:

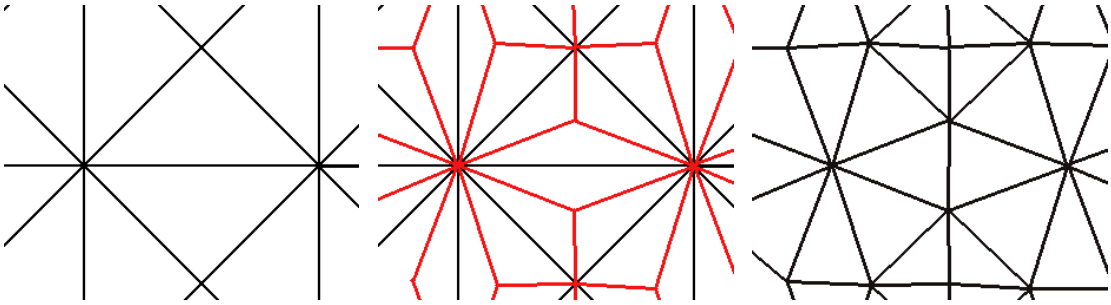
- the number of triangles increases slower than with the use of other subdivision algorithms
- simple extension to adaptive subdivision
- all meshes to subdivide are triangle meshes
- lower shrinkage effect (explained in 4.12) than other subdivision methods
- if a mesh is propagated to datasets to be segmented and subdivided in only one of these datasets, the $\sqrt{3}$-subdivision algorithm preserves the vertex correspondences



(a) original mesh

(b) mesh after $\sqrt{3}$-subdivision

Figure 4.12: Mesh before and after $\sqrt{3}$-subdivision



(a) $\sqrt{3}$-subdivision step 0. (b) $\sqrt{3}$-subdivision step 1. (c) $\sqrt{3}$-subdivision step 2.

Figure 4.13: This figure illustrates the $\sqrt{3}$-subdivision steps. Figure 4.13a shows the initial triangles. The newly inserted vertices and edges are shown in red in figure 4.13b. The resulting triangulation after flipping the old edges is depicted in figure 4.13c.

4.10.1 $\sqrt{3}$-subdivision

Given a mesh M_0 a 1-3 split is performed for every triangle by inserting a vertex at its centre. 1-3 split means that one triangle is split into three triangles. After that every new vertex is connected to the surrounding vertices of the triangle in which centre it lies. The next step is to rebalance the valence of the mesh vertices. This is done by flipping every old edge to the centre points of its two adjacent triangles. If the 1-3 split is applied twice, every original triangle is split into nine subtriangles. The application of the 1-3 split twice is also called a tri-adic split. Therefore Kobbelt regarded one subdivision step as the square root of the tri-adic split. Figure 4.13 illustrates the steps of this subdivision method.

After the new vertices have been inserted and the edge flipping is done the positions of the old and new vertices have to be adjusted to retain the smoothness of the mesh. The position $\mathbf{v}_i$ of an inserted vertex v_i with index i is calculated as

$$\mathbf{v}_i = \frac{1}{3}\left(\mathbf{v}_j + \mathbf{v}_k + \mathbf{v}_l\right) \tag{4.14}$$

where j, k and l are the indices of the vertices of the original triangle into which v_i was inserted. The new position $\mathbf{v}'_i$ of an already existing vertex v_i is calculated by:

$$\mathbf{v}'_i = (1-\alpha)\,\mathbf{v}_i + \alpha \frac{1}{val(v_i)} \sum_{\forall k \in K_i} \mathbf{v}_k \tag{4.15}$$

where α is defined as

$$\alpha = \frac{4 - 2\cos\left(\frac{2\pi}{val(v_i)}\right)}{9}. \tag{4.16}$$

The module for the $\sqrt{3}$-subdivision is outlined in algorithm 4.11.

1 **subdivide**[mesh m]

1: $insertVertexAtEveryTriangleCentre(m)$
2: $connectNewVertices()$
3: $flipOldEdges()$
4: $calculateNewVertexPositions()$
5: **return** subdividedMesh

Algorithm 4.11: $\sqrt{3}$-subdivision module

4.10.2 Adaptive $\sqrt{3}$-subdivision

Although in the case of $\sqrt{3}$-subdivision the number of triangles increases not as fast as in the case of other subdivision methods (e.g. Loop [34]), after a few subdivision steps performed on the whole mesh the number of vertices goes beyond the computational power of an ordinary computer. Therefore it is useful to subdivide the mesh only in regions where a higher number of triangles is necessary. This is done by adaptive subdivision.

The algorithm implemented and developed in the course of this work is designed to preserve *border and feature edges*. A feature edge describes an edge for which the angle between the normals of the two adjacent triangles is beyond a specific threshold θ. To ensure the preservation of border and feature edges a counter c_i has to be determined for every edge. The i denotes for the edge index. The counter c_i counts the times a specific border or feature edge was not flipped in the subdivision process. The algorithm needs a set of edges E as input that determines the area of the mesh to be subdivided.

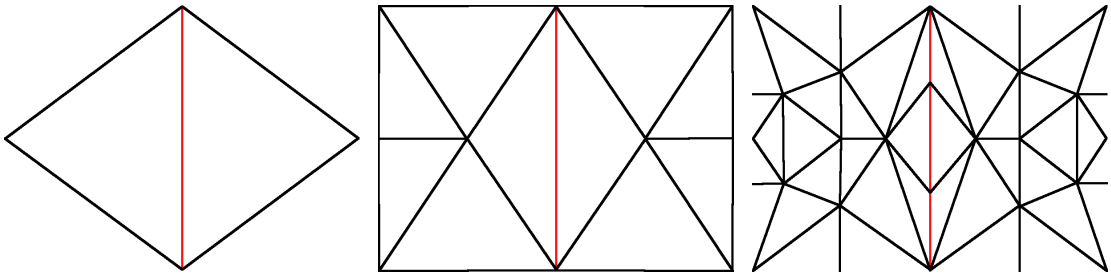
The algorithm works as follows: At the beginning a vertex is inserted at the centre of every triangle adjacent to any of the edges. Subsequently every new vertex is connected to the surrounding vertices of the triangle in whose centre it lies. The edge flip is done only for edges that are not marked as feature edges. If an edge is a feature or border edge and no edge flip is done and c for edge e_{ij} is odd, c is increased by one. If no edge flip is done and the above introduced counter c for edge e_{ij} is even, two vertices v_l and v_k are inserted. One at the centre of the first half of the edge and one at the centre of the second half of the edge. These two new vertices are connected to the points previously inserted at the centre of the adjacent triangles. As shown in Figure 4.14 it is possible to preserve feature edges throughout the subdivision steps. The position of $\mathbf{v}_l$ and $\mathbf{v}_k$ are calculated by

$$\mathbf{v}_l = \frac{1}{3}\mathbf{v}_i + \frac{2}{3}\mathbf{v}_j \tag{4.17}$$

and

$$\mathbf{v}_k = \frac{2}{3}\mathbf{v}_i + \frac{1}{3}\mathbf{v}_j \tag{4.18}$$

If an edge is a feature edge its vertices remain unchanged. The positions of all other points are calculated as described in equation 4.15. To give a better understanding the pseudo code in algorithm 4.12 exemplifies the module with the approach of the algorithm. This adaptive $\sqrt{3}$-subdivision increases the number of triangles more than other known subdivision algorithms (like Yu et al. [58]), but has the advantage that it can be programmed in a way that only one



(a) original mesh (b) one subdivision (c) two subdivisions

Figure 4.14: The red edge in the three images is the feature edge. (a) shows the original triangulation. (b) shows the triangle after the first subdivision step. No vertices were inserted on the feature edge as its counter is odd after the first step. (c) shows the mesh after the second subdivision step. Now two vertices are inserted in the feature edge and connected to the vertices inserted at the centre of the adjacent triangles of the feature edge.

iteration over the mesh part to be subdivided is necessary. Yu et al. [58] in contrast needs, due to the edge flip, at least two iterations. Because of this advantage in computational effort this way to subdivide parts of the mesh was chosen.

4.11 Decimation

Decimation algorithms simplify the topology of meshes in areas where a high tessellation is not necessary. In the remainder of this section the decimation approach of Hussain et al. [25] is presented. This method was chosen because of its easy implementation and its fast computation.

Hussain et al. [25] decimate meshes by collapsing edges. The idea of the edge collapse used is depicted in figure 4.15. To delete an edge from a vertex v_1 to a vertex v_2, all occurrences of v_1 are replaced by v_2 or vice versa. Additionally, the two triangles that share the edge from v_1 to v_2 are deleted. The decision which vertex is replaced depends on which vertex replacement has the lower costs and which vertices have the lower visual importance. The whole decimation process for one mesh is, according to Hussain et al. [25] defined as follows:

1. Calculate the visual importance for every v of the original mesh M_0.
2. Compute the cost for every possible vertex replacement.
3. Assign every vertex of M_0 the smallest cost of its possible replacements and scale it with the visual importance. If the visual importance is below a threshold put the obtained value in a priority queue.
4. Take the smallest value out of the priority queue and replace the vertex corresponding with the value by its assigned vertex.
5. Replace M_0 with the new resulting mesh and go to 1. except when there are no more values in the priority queue.

```
1 adaptiveSqrt3Subdivision[EdgeSet E]
 1: edgeSetSize = E.size()
 2: for i = 0 → edgeSetSize do
 3:    edge = E.getEdge(i)
 4:    triangle0 = edge.getTriangle0()
 5:    triangle1 = edge.getTriangle1()
 6:    insertPointAtTriangleCentre(triangle0)
 7:    insertPointAtTriangleCentre(triangle1)
 8:    centreVertex0 = triangle0.getCentreVertex()
 9:    centreVertex1 = triangle1.getCentreVertex()
10:    for j = 1 → 3 do
11:       vertexTr0 = triangle0.getVertex(j)
12:       vertexTr1 = triangle1.getVertex(j)
13:       connectVerticesWithEdge(centreVertex0, vertexTr0)
14:       connectVerticesWithEdge(centreVertex1, vertexTr1)
15:    end for
16:    if isFeatureOrBorderEdge(edge) then
17:       c = edge.getCounter()
18:       if isOdd(c) then
19:          c ← c + 1
20:       else
21:          firstHalfVertex = inserVertexAtFirstEdgeHalf(edge)
22:          secondHalfVertex = inserVertexAtSecondEdgeHalf(edge)
23:          triangle0.connectVertexWithCentre(firstHalfVertex)
24:          triangle0.connectVertexWithCentre(secondHalfVertex)
25:          triangle1.connectVertexWithCentre(firstHalfVertex)
26:          triangle1.connectVertexWithCentre(secondHalfVertex)
27:          c ← c + 1
28:       end if
29:    else
30:       flipEdge(edge)
31:    end if
32: end for
33: return subdividedMesh
```

Algorithm 4.12: Adaptive $\sqrt{3}$-subdivision module

To fasten up the algorithm the calculation of the visual importance of every vertex and the costs of the vertex replacements after every vertex replacement can be discarded. This is only necessary if an edge is collapsed and an edge of the one ring neighbourhoods of the deleted vertex and its substituting vertex also needs to get collapsed. This is because the collapse of an edge only exerts influence on the visual importance or the costs in this region. The new algorithm flow changes slightly and now looks as follows:

1. Calculate the visual importance for every v of M.
2. Compute the cost for every possible vertex replacement.
3. Assign every v the smallest cost of its possible replacements and scale them with the visual importance. If the visual importance lies below a threshold put the obtained value in a priority queue.
4. Replace all vertices, starting from the smallest value until there are no possible replacements left that share no common vertex with the one ring neighbourhoods of a replaced vertex and its substituting vertex of one of the already replaced vertices.
5. Go to 1. except there are no more values in the priority queue.

This change of the algorithm can produce a different result from the original one as it replaces vertices with high priority later then vertices with low priority when the vertices with high priority and their substituting vertices share a common vertex in their one ring neighbourhoods. This loss of an optimal vertex replacement order has been chosen as this method is more than 30 times faster and the decrease of the quality of a mesh is minimal. The faster computation is because the calculation of the vertex weights and the visual importance of the mesh vertices after a vertex replacement has to be done only a few times instead of after every vertex replacement. The decimation of a vertebra mesh with 4710 vertices and 14130 edges needs about 30 seconds with the original algorithm and less than 1 second with the adapted one on machine 1 described in section 6.3.

The remainder of this section shows the calculation of the costs and the visual importance as well as the module for the decimation algorithm. Before the costs can be calculated, the visual importance for every vertex is needed. The visual importance for a vertex v_i is defined as

$$\omega_{v_i} = 1 - \alpha_{v_i} \tag{4.19}$$

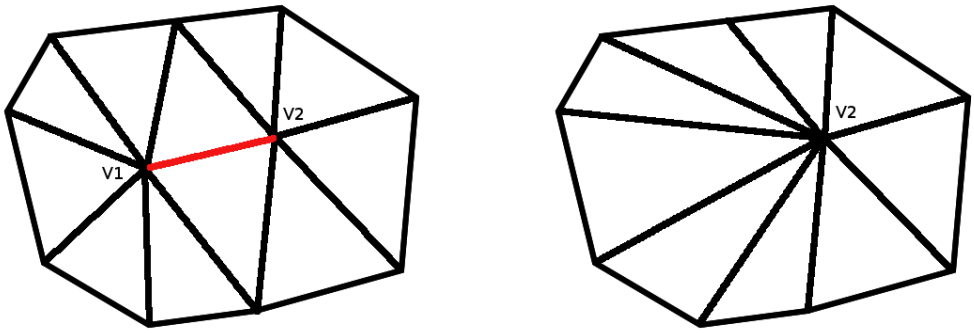
where

$$\alpha_{v_i} = \| \frac{\sum_{\forall t \in T_{v_i}} a(t)\mathbf{n}_t}{\sum_{\forall t \in T_{v_i}} a(t)} \|. \tag{4.20}$$

ω_{v_i} is zero if vertex v_i has a flat one ring neighbourhood. The costs for replacing a vertex v_i by a vertex v_j are calculated as follows:

$$Cost\left(v_i, v_j\right) = \sum_{t \in T_{v_i} - T_{e_{ij}}} \left(0.5\left(a(t) + a(t')\right)\theta_{tt'}\right). \tag{4.21}$$

t' is the triangle t after the vertex replacement. $\theta_{tt'}$ is the angle between the normals of t and t'. For a faster computation, $\theta_{tt'}$ can be approximated, according to Hussain et al. [25], by $1 - \mathbf{n}_t\mathbf{n}_{t'}$. The module for the decimation process is shown in algorithm 4.13.



(a) before vertex replacement (b) after vertex replacement

Figure 4.15: The edge collapse is done by replacing all occurrences of v_1 by v_2 and deleting the two triangles that share the edge between these two vertices.

decimate[mesh m]

while $true$ **do**
  for vertex : m **do**
    $calculateVisualImportance(v)$
  end for
  $computeCostsForHalfEdgeCollapse()$
  $scaleCostsWithVisualImportance()$
  $priorityQueue = getPriorityQueue()$
  $collapseHalfEdges(priorityQueue)$
  if !$priorityQueue.isEmpty$ **then**
    $updateMesh()$
  else
    $break$
  end if
end while
return decimatedMesh

Algorithm 4.13: Decimation module

4.12 Smoothing Methods

Another method to increase the quality of meshes is to smooth them. The advantage over subdivision and decimation is that smoothing methods alter only the position of vertices, the topology

remains unchanged. According to Botsch et al. [8] smoothing methods can be divided into *denoising* and *fairing* methods. Denoising has the aim of eliminating high frequency artefacts like small changes in the vertex positions that do not belong to the mesh structure. Denoising algorithms can be regarded as low pass filters. Fairing a mesh is similar to denoising but it also takes aesthetic aspects into account. Sapidis [45] states that aesthetic shapes are free of unnecessary features and have a simple design. Sapidis [45] summarizes this as *the principle of simplest design*. Energy minimization functions are used to fair meshes.

Three smoothing algorithms were implemented in this work. A simple Laplacian smoothing algorithm by Desbrun et al. [17], a recursive Laplacian method by Kobbelt et al. [33] and a barycentre approach developed by Kainmueller et al. [27]. They are all denoising methods as for medical structures, like tumours, aesthetic aspects need not to be taken into account. The main drawback of this kind of algorithms is the *shrinkage effect*. As they all pull the vertices to smooth towards the barycentre of their one ring neighbourhood the volume of a model gets smaller with every smoothing step. Section 4.12.5 will discuss the differences of these three methods and will explain why the algorithm by Kobbelt et al. [33] was chosen as the best suited algorithm to support solving the problem stated in section 1.5.

4.12.1 Laplacian Mesh Smoothing

One common mesh smoothing method is Laplace smoothing. It is defined as the *divergence* of the *gradient* of a function f or in other words the sum of all second partial derivatives of f in x_i:

$$\Delta f = \nabla^2 f = \sum_{i=1}^{u} \frac{\partial^2 f}{\partial x_i^2}. \tag{4.22}$$

x_i denotes for the Cartesian coordinates, u is the number of dimensions of the space. There are different discrete approximations of the Laplace operator. One of them is

$$L(\mathbf{v}_i) = \sum_{\forall x \in K_{v_i}} \omega_{ij} \left(\mathbf{v}_x - \mathbf{v}_i\right) \tag{4.23}$$

which is defined by Desbrun et al. [17]. ω_{ij} was, due to the need of fast computation, chosen as $1/n$, n is the number of vertices in the one ring neighbourhood of v_i. With this weight, equation 4.23 is also know as *umbrella operator*. The Laplacian operator for the entire mesh can be represented by an mxm matrix A:

$$A_{ij} = \begin{cases} -1 & i = j \\ w_{ij} & (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

where m is the number of vertices of the mesh and E is a set that contains all edges of the mesh. This operator pulls a vertex towards the barycentre of its one ring neighbourhood. The smoothed position $\mathbf{v}_i'$ for vertex v_i is calculated with

$$\mathbf{v}_i' = \mathbf{v}_i + L(\mathbf{v}_i). \tag{4.25}$$

4.12.2 Recursive Laplacian Mesh Smoothing

An extension to the approach of section 4.12.1 is the recursive umbrella operator presented by Kobbelt et al. [33]. The recursive umbrella operator

$$L(L(\mathbf{v}_i)) = \sum_{j \in K_i} \omega_{ij} \left(L(\mathbf{v}_j) - L(\mathbf{v}_i) \right) \tag{4.26}$$

is a discrete approximation of $\Delta^2 f$. The smoothed position for a vertex $\mathbf{v}_i$ can be calculated by

$$\mathbf{v}_i' = \mathbf{v}_i - \frac{1}{d} L(L(\mathbf{v}_i)) \tag{4.27}$$

with a so called "diagonal element"

$$d = 1 + \frac{1}{val(v_i)} \sum_{\forall y \in K_{v_i}} \frac{1}{val(\mathbf{v}_y)}. \tag{4.28}$$

The module for the smoothing method with the recursive Laplacian operator is defined in algorithm 4.14.

```
1 smoothMesh[vertexArray v]
1: for vertex : v do
2:     l = computeLaplacian(vertex)
3:     laplaceMap.put(l)
4: end for
5: for l : laplaceMap do
6:     l2 = computeLaplace(l)
7:     vertexPosNew = vertexPosOld − (l2/diagonalElement)
8: end for
9: return smoothMesh
```

Algorithm 4.14: Recursive Laplacian operator module

4.12.3 Barycentre Mesh Smoothing

A fully automatic liver segmentation is presented by Kainmueller et al. [27]. Their approach is based on statistical shape models and a constrained free form deformation step. This free form deformation step contains a smoothing algorithm similar to the approach in section 4.12.1 that can be applied to simple mesh smoothing. Given a set of vertices V, every vertex v of V is, like in section 4.12.1, pulled towards its barycentre. The smoothed vertex position $\mathbf{v}_i'$ of a vertex v_i is calculated by

$$\mathbf{v}_i' = \mathbf{v}_i - \mathbf{u} \tag{4.29}$$

where

$$\mathbf{u} = \omega \left(\mathbf{v}_i - \frac{\sum_{\forall y \in K_{v_i}} \mathbf{v}_y}{val(v_i)} \right) \tag{4.30}$$

ω is set to 0.15 according to Kainmueller et al. [27]. The difference to the technique in section 4.12.1 is that this approach attenuates the force that pulls a vertex towards the barycentre with ω. This lowers the shrinkage effect.

4.12.4 Laplacian Eigenvalue Smoothing

A mesh can be seen as a discrete time signal. The removal of all high frequencies would smooth the mesh. According to Taubin [52] the eigenvectors of the Laplace matrix A (definition 4.24) correspond to the frequencies of the mesh. Eigenvectors with high eigenvalues correspond to high frequencies and vice versa. The eigenvectors form the Laplacian eigenbasis (LBS). The mesh can be represented in the LBS as

$$x = \sum_{i=1}^{n} u_i^T x u_i \tag{4.31}$$

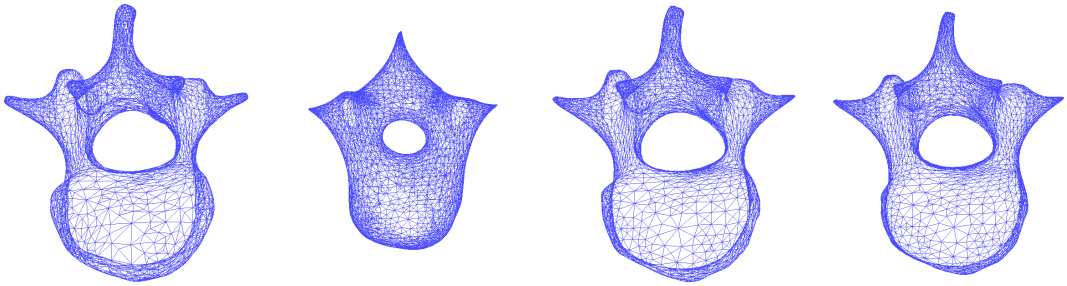
where x stands for the Cartesian coordinates and u for the eigenvectors. By truncating the eigenvectors with high eigenvalues the mesh gets smoothed. Sorkine and Nealen [50] presented a smoothing method that makes use of the eigenvectors and eigenvalues of the Laplace matrix. This method was also implemented in this thesis but discarded later because the computation of the eigenvectors and eigenvalues took too much time and is therefore not applicable for a real time smoothing.

4.12.5 Conclusion

The approach of 4.12.2 was chosen to be the best suited algorithm for the program developed in this thesis because it presents, according to Desbrun et al. [17] the best compromise between computational cost and shrinkage effect. Furthermore, the computational costs of the algorithm can be minimised by avoiding the double computation of the umbrealla operator. Figure 4.16c shows that the recursive Laplacian best preserves the original shape of the vertebra. With the use of the simple Laplacian the processes of the vertebra vanish (figure 4.16b). The barycentre approach (figure 4.16d) provides, due to the weighting factor, a much better result than the simple Laplacian but has a higher shrinkage effect than the recursive Laplacian.

4.13 Curvature Measurement

As smoothing meshes consumes a lot of time and interrupts the user interaction flow, an algorithm was implemented that determines whether a mesh should be smoothed or remain unchanged. To smooth only the areas of a mesh where it is really necessary saves a lot of computational costs. The here presented algorithm evaluates the curvature at every vertex position and assigns the vertex a value that decides if the area around the vertex should be smoothed. The method implemented to solve this task was proposed by Desbrun et al. [17] and calculates a *curvature normal* $\overline{\kappa}n$ for every vertex. $\overline{\kappa}n_i$ is defined as:



(a) original vertebra mesh (b) mesh after Laplacian smoothing (c) mesh after recursive Laplacian smoothing (d) mesh after barycentre smoothing

Figure 4.16: (a) shows the original mesh. (b), (c) and (d) show the mesh after 100 iterations of the respective smoothing algorithm. It can be seen clearly that the recursive Laplacian operator seen in (c) shrinks the mesh least of all operators.

$$\overline{\kappa}n_i = -\frac{1}{4\sum_{\forall t \in T_{v_i}} a(t)} \sum_{\forall x \in K_{v_i}} (\cot \alpha_{v_x} + \cot \beta_{v_x}) (\mathbf{v}_x - \mathbf{v}_i) \tag{4.32}$$

α_{v_x} and β_{v_x} are the two angles opposite to the edge from vertex v_i to v_x. If the length of $\overline{\kappa}n_i$ is over a specific threshold the area around the vertex is going to be smoothed. The advantage of using the value of this operator as a decision threshold is, that this operator results in zero if the area is flat. The Laplace operator is only zero if the evaluated vertex lies exactly at the barycentre of its one ring neighbourhood. The module for the curvature measurement is specified in algorithm 4.15.

measureCurvature[vertexArray v]

1: **for** $vertex : v$ **do**
2: $curvatureNormal = calcCurvatureNormal(vertex)$
3: $curvatureNormalArray.put(curvatureNormal)$
4: **end for**
5: **return** curvatureNormalArray

Algorithm 4.15: Curvature measurement module

4.14 4D Deformation and Transformation Propagation

To manually segment a four dimensional dataset, 3D dataset by 3D dataset, can be very tedious and time consuming. Therefore some tools were implemented that ease and speedup the 4D segmentation process. With the use of the first tool the mesh of one 3D dataset can be set as starting mesh for the other datasets. This can be useful if the shape is already segmented in one 3D dataset, so that this segmentation can be used as initialization for the segmentation of the other datasets. This method is very similar to the one presented by McInerney and Terzopoulos

[37]. The difference is that with the introduced method the user can choose the 3D datasets where the current mesh model should be inserted. The advantage over a global setting of the model is that this method can be used more than once in one segmentation task as the global setting of the model overrides always all already segmented shapes. The module for this method is defined in algorithm 4.16.

1 setMeshToDatasets[Mesh m, DatasetArray ds]

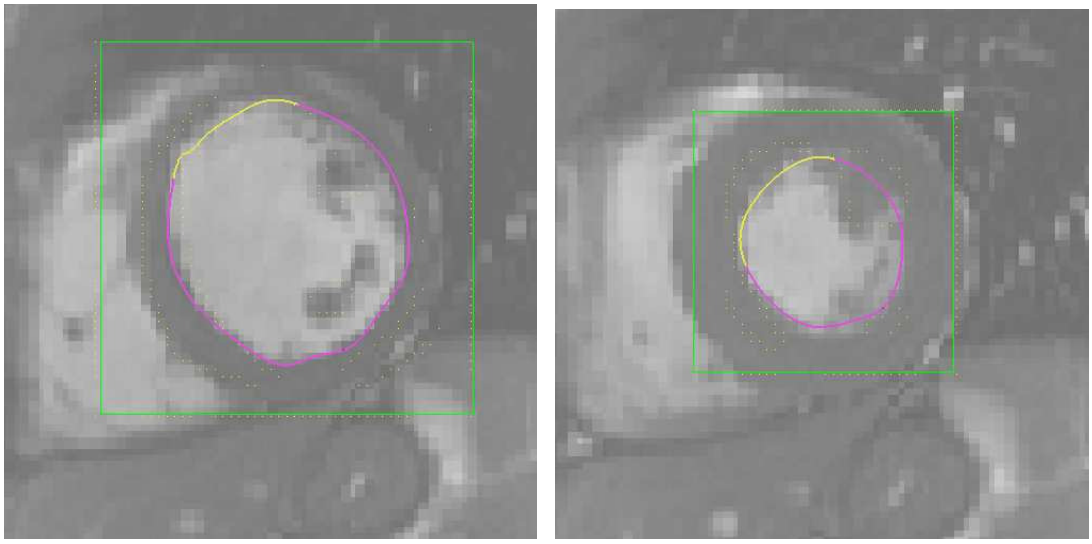
1: **for** $dataset : ds$ **do**
2: $\quad dataset.setMesh(m)$
3: **end for**
4: **return** datasetsWithNewMesh

Algorithm 4.16: Set mesh in dataset module

The second tool is capable of recording deformation and transformation steps that are carried out on a 3D dataset. Afterwards all recorded steps can be propagated to the other datasets. For the different deformation respectively transformation methods different variables have to be recorded. Using the information of these variables it is possible to simulate a complete deformation respectively transformation step carried out by the user in the different datasets. By simulating deformation respectively transformation steps and the use of the Sticky Edges algorithm varying edge positions from dataset to dataset can be handled. Furthermore, it is possible to scale the deformations and transformations along the fourth dimension. The scaling is done with the Gaussian function mentioned in equation 4.5 with the adjustment that the Gaussian function in the numerator gets as argument the number of datasets that lie between the dataset where the operations were recorded and the dataset where the operations are currently applied. With the adjustment of σ the number of slices on which the deformations and transformations have an impact on can be defined. In the case of a transformation the translation variable, the scale factor and the rotation angle are weighted with the Gaussian function. In the case of deformations the deformation vector and the kernel size are downscaled.

With the use of scaling, motion can be segmented easier as the shape of motion in nature is often similar to the shape of a Gaussian function. Furthermore, scaling simplifies the deformation of meshes of different sizes. For example, if systole and diastole of the heart motion are to be segmented, the mesh in the systole dataset is smaller than the mesh in the diastole dataset. To deform the systole mesh in the same way as the diastole mesh would not be meaningful as the diastole mesh needs more precise deformations as the heart is in the systole state smaller than in the diastole state. Furthermore, the kernel size used in the diastole dataset is too big for the mesh in the systole dataset as the use of the same kernel size in the systole dataset would encompass a larger anatomical region than in the diastole dataset. This can be seen in figure 4.17 and can be avoided by downscaling the deformations on the diastole dataset before they are applied to the systole dataset.

The big advantage of simulating the deformation and transformation steps in the different datasets is that it does not make a difference which topology the meshes in the datasets have. This facilitates the usage of subdivision or decimation if a more precise or coarser tesselation is necessary in different datasets. The module for the second tool is outlined in algorithm 4.17.



(a) diastole dataset (b) systole dataset

Figure 4.17: In both figures the yellow line marks the outline the deformation area with a kernel size of 20. The pink line marks the outline of the part of the mesh that does not belong to the deformation area. The green square denotes the bounding box around the mesh and the yellow dots indicate edge pixels in the dataset. As can be seen in the systole dataset the kernel embraces a much bigger part of the mesh as in the diastole dataset.

1 **propagateOperations**[OperationArray op, boolean scale]

```
1: for dataset : allDatasets do
2:     for operation : op do
3:         if scale then
4:             scaleOperation(op, dataset)
5:         end if
6:         dataset.applyOperation(op)
7:     end for
8: end for
9: return datasetsWithNewMesh
```

Algorithm 4.17: Propagate operations module

4.15 Putting the Methods Together to an Segmentation Algorithm

This section gives information about the architecture of the algorithm. It shows, how the previously mentioned methods were put together to an algorithm that is able to segment shapes captured by diverse modalities in 3D and 4D. The algorithm can be divided into a deformation, a 4D propagation and an optimization module.

4.15.1 Deformation

The first step is to load a dataset. In the next step the mesh respectively, in the case of a 4D dataset, the meshes are loaded and placed in the dataset. If a 4D dataset is loaded but the mesh file contains only one 3D mesh, the same mesh is set in every volume of the 4D dataset. If the mesh file contains stored edge information for the dataset this information is also loaded, otherwise the edge positions are calculated with the Canny (section 4.4.1) or the LoG (section 4.4.2) edge detection algorithm in a bounding box (section 4.3) around the mesh. After the edges are determined they are also getting classified with the algorithm mentioned in section 4.5.

Before the deformation starts, the mesh is roughly aligned to the shape to be segmented with the use of the three transformation functions (section 4.7). To deform a mesh the algorithm needs the deformation centre, the deformation vector, the kernel size and which deformation method shall be chosen as input variables. With the use of the kernel size, the deformation area is calculated. Every vertex of the mesh that lies within the specified kernel size is added to the deformation area. The distance to the deformation centre is the shortest edge path from the deformation centre to a specific vertex. For the deformation the normal of the deformation centre and the normals of the mesh vertices in the deformation area are needed. The normals of the vertices are calculated by taking the mean of the normals of the triangles the vertex is part of. The normal of the deformation centre is calculated by taking the mean of the normals of the vertices that are part of the edge that lies closest to the deformation centre. After the determination of the region to deform, the mesh is, according to the corresponding input variable, deformed with the simple deformation (section 4.7.1) or the deformation with the use of the normals (section 4.7.2). After the deformation the Sticky Edges method (4.8) is executed for a better alignment of the deformation area to the shape to be segmented.

As after a free deformation step and the Sticky Edges step the mesh surface may have some unwanted sharp features, the curvature of the vertices in the deformation area is determined with the use of the algorithm explained in section 4.13. If the curvature of one vertex lies above a given threshold, the deformation area is smoothed with the recursive Laplacian smoothing operator mentioned in subsection 4.12.2. This smoothing step is repeated 10 times or there is no vertex with a curvature higher than the given threshold left left. The iteration threshold of 10 was chosen because sometimes vertices with a curvature higher than the given threshold lie at the border of the deformation area. The recursive Laplacian smoothing operator is not able to smooth this vertex in a way that its curvature value falls below the threshold because for that, also the vicinity of this vertex would need to be smoothed. As the vicinity is not part of the deformation area the algorithm is not able to smooth the area around the vertex without the curvature value going beyond given threshold. Such a case would, without the iteration threshold, cause an infinite loop. Experience showed that coarse areas caused by deformation steps are

completely smoothed out after 10 smoothing steps. After every deformation and transformation step the program examines if the bounding box around the area of the calculated edges is still big enough. For this purpose two criteria are taken into account. If one of them is fulfilled the edges are recalculated and classified. These two criteria are:

Mesh ratio: If the maximal extent in x-, y- or z-direction of the mesh after deformation respectively transformation divided by the maximal extent before is higher than a given threshold, the edges are recalculated.

Centre change: If the change of the mesh centre in x-, y- or z-direction divided by the maximal extent of the unchanged mesh in the particular direction is higher than a given threshold the edges are recalculated.

This is done to avoid that vertices are dragged out of the bounding box where the Sticky Edges algorithm can not be applied. The module for the whole deformation process is specified in algorithm 4.18.

4.15.2 4D Propagation

To propagate the deformation to other meshes in a 4D dataset, the deformed mesh can be either included in the other 3D volumes or the deformation and transformation operations can be recorded and then applied to the other meshes. To use the second method, specific variables have to be recorded. The following listing gives an overview on the recorded variables:

transformation: the position from where the user is currently viewing the mesh (camera position), transformation mode (translation, rotation or scaling), the positions where the user pressed and released the mouse

deformation: kernel size, attenuation function, deformation mode, camera position, the indices of the currently viewed slices, the positions where the user pressed and released the mouse, if available the gradient direction of the edge where the user dropped the deformation centre

These variables are necessary to simulate a deformation or transformation carried out by the user on the other datasets. Together with these variables the method needs a boolean value that indicates whether the recorded operations shall be weighted or not. The variable σ for the Gaussian weighting of the operations was chosen to be one third of the number of 3D datasets contained in a 4D dataset. This value was chosen because with this adjustment it does not matter which dataset is segmented. The propagation always affects every other dataset. The 4D propagation module is specified in algorithm 4.19.

4.15.3 Mesh Optimization

To optimize a mesh after deformation four methods can be used. Some of the optimization methods are able to handle input variables (i.e., a set of triangles or edges). With the use of this

1 **deform**[edgeDetection ed, deformMethod dm, deformationCentre dc, deformationVector dv, kernelSize k, standardDeviation σ, threshold zc, threshold T1 and T2, curvatureThreshold curTr, meshRatioThreshold mrTr, centreChangeThreshold cCTr]

```
1:  loadDataset()
2:  meshes = loadMesh()
3:  calculateBoundingBox(m)
4:  if dataset has no edge informaiton then
5:     if ed = LoG then
6:        eIdx = calculateEdgesForImgWithLoG(k, σ, zc)
7:     else
8:        eIdx = calculateEdgesForImgWithCanny(k, σ, T1, T2)
9:     end if
10:    classifyEdges(mesh.dataset, eIdx)
11: end if
12: transform()
13: calcDeformationArea(dc, k)
14: if dm = simple then
15:    deformSimple(dv, dc, k)
16: else
17:    deformWithNormals(dv, dc, k)
18: end if
19: stickyEdges(deformationArea, dc)
20: curvatureNormals = measureCurvature(mesh.vertices)
21: while curvatureNormals.notEmpty() OR i <= 10 do
22:    for curvatureNormal : curvatureNormals do
23:       if curvatureNormal > curTr then
24:          vertex v = getVertexForCurvature(curvatureNormal)
25:          smoothingArray.add(v)
26:       end if
27:    end for
28:    smoothMesh(smoothingArray)
29:    curvatureNormals = measureCurvature(mesh.vertices)
30:    i++
31: end while
32: mc = getChangeOfMeshCentre() / getMaxMeshExtentOld()
33: msp = getMeshExtentNew() / getMeshExtentOld()
34: if cCTr > 0.1 OR msp > mrTr then
35:    calculateBoundingBox(m)
36: end if
37: return deformedMesh
```

Algorithm 4.18: Deformation module

1 **propagate**[propagationType t, datasetsToSetMesh datasets, recordedOperations op, boolean weightOperations]

1: **if** $t = setMeshes$ **then**
2: $m = getCurrentMesh()$
3: $setMeshToDatasets(m, datasets)$
4: **else**
5: $propagateOperations(op, weightOperations)$
6: **end if**
7: **return** propagatedMeshes

Algorithm 4.19: 4D propagation module

variables it is possible to specify the area of a mesh on which the algorithm should be applied. The first one is the adaptive $\sqrt{3}$-subdivision algorithm described in subsection 4.10.2. If the method is applied and the input edge set is empty the whole mesh is evaluated whether it contains edges to subdivide or not. An edge needs to get subdivided if the angle between its two adjacent triangles is greater than $40°$ or its subdivision counter is even. Feature edges, also mentioned in subsection 4.10.2, are not used in this step as the aim of this step is a fine mesh without sharp edges. If an edge fulfils the subdivision criterion it is added to a set that contains all edges to subdivide. The edges in the set are then processed by the algorithm and the evaluation step starts again. This is repeated until no more edges are contained in the set or this loop has iterated five times. The threshold of five is used to avoid a mesh with too many triangles. Because in the case that subdivision criterion is not fulfilled after fiver iterations of the algorithm, the use of the smoothing algorithm would make more sense. If the input edge set is not empty it is processed by the algorithm. In this case no evaluation of the mesh is done before or after subdivision and the algorithm terminates after the edges in the set have been subdivided once. The second one is the the decimation method explained in section 4.11. In contrast to subdivision this method is not able to handle input. The whole mesh is always evaluated. For the evaluation, the method needs a threshold for the visual importance, which is used in step three of the algorithm as input parameter. The recursive umbrella operator discussed in section 4.12.2 is used to smooth the mesh. The algorithm iterates only once over the mesh. As the deformation area gets scanned after every deformation step to determine if smoothing is necessary and gets smoothed if the scanning algorithm returns true, a higher iteration number is not required as a good base level of smoothness is thereby guaranteed. The last method is the interpolation operator shown in section 4.9. The method needs the vertex whose position has to be interpolated and an array of sampling vertices as input.

These methods are currently not part of the core algorithm and have to be called separately. This was done as machine 1 (section 6.4) where the program was developed in the course of this work is too slow to guarantee a fast computation of the program when the optimization methods are implemented in a post processing step of the deformation module. The module that combines all optimization methods is specified in algorithm 4.20.

```
1 opimize[optimizationMethod op, edgesToSubdivide sdEdges, vertexToInterpolate vToInt,
  samplingVerticesArray samplV]
 1: if op = subdivide then
 2:   if sdEdges.isEmpty() then
 3:     meshEdges = mesh.getEdges()
 4:     while i <= 5 do
 5:       for edge : meshEdges do
 6:         if edge.getSubdivisionCounter.isEven() then
 7:           edgesToSubdivde.add(edge)
 8:         else
 9:           angle = getAngleBetweenAdjacentTriangles(edge)
10:           if angle > 40° then
11:             edgesToSubdivde.add(edge)
12:           end if
13:         end if
14:       end for
15:       if sdEdges.isNotEmpty() then
16:         subdivide(mesh)
17:       else
18:         break
19:       end if
20:       i++
21:     end while
22:   else
23:     subdivide(mesh)
24:   end if
25: else if op = decimate then
26:   decimate(mesh)
27: else if op = smooth then
28:   smoothMesh(mesh.getVertices())
29: else if op = interpolate then
30:   interpolate(vToInt, samplV)
31: end if
32: return optimizedMesh
```

Algorithm 4.20: Optimization module

CHAPTER 5

Implementation

5.1 Overview

This chapter shows how the algorithm mentioned in section 4.15 was implemented to build a program that fulfils the needs listed in section 1.6. The program was developed as a plug-in for a prototyping framework called *Model Building Studio* developed at the VRVis. The plug-in is called *Mesh Plugin*. Section 5.2 shows and describes the graphical user interface (GUI), section 5.3 explains the data structures used and section 5.4 illustrates the work flow and how the algorithm was implemented to enable the user to achieve the best segmentation results.

5.2 GUI

The GUI seen by the user when the Model Building Studio starts is depicted in figure 5.1. The tab seen is called the navigation tab. With the use of the sliders in the area marked with a red *1* the currently visible slices can be set. This can also be achieved by holding the ctrl-key and moving the mouse wheel. The buttons in area *1* hide respectively show a particular plane. With the use of the radio buttons in area *2* the regions under and above a particular plane can be cut off. The thickness of these regions can be adjusted with a slider. An example can be seen in figure 5.2. Area *3* shown in figure 5.1 gives information about the distances between the slices. With the use of the sliders in area *4* the range of the shown intensity values can be adjusted. In case of a four dimensional dataset the intensity range can be adapted for every 3D sub-dataset separately. With the slider in area *5* the currently shown volume of a 4D dataset can be chosen. This can also be done with the left and right arrow keys of the keyboard. Area *6* contains the 3D area that shows the dataset and the mesh the user can work on. The drop down menus in the upper left corner of the GUI, seen in figure 5.3, offer functionalities to open and export datasets, load plug-ins and to change the program's properties. A click on *open volume* or *open DICOM* in the *file* drop down menu opens a file chooser dialogue which is shown in figure 5.4.

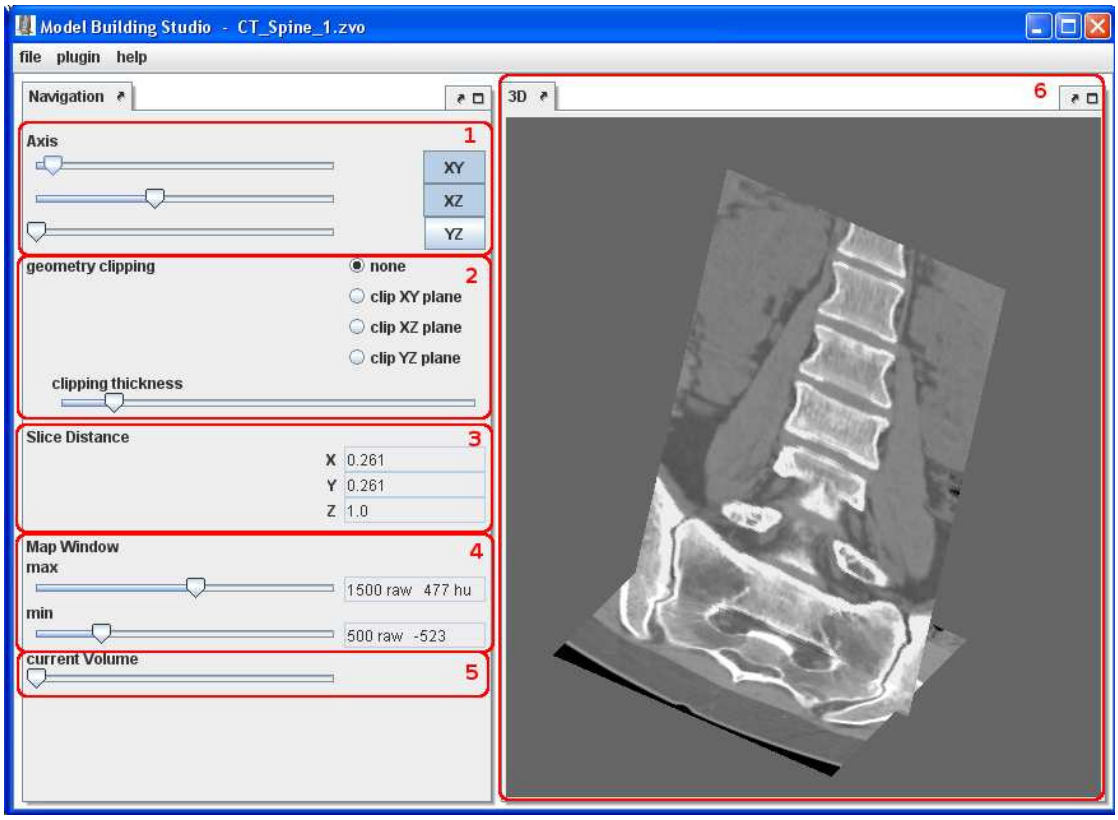


Figure 5.1: The navigation tab is seen by the user when the application starts. Its input elements can be used to traverse a dataset and to change the view of a dataset.

To segment a dataset the Mesh Plugin has to be loaded. This step can be seen in figure 5.5. The Mesh Plugin is depicted in figure 5.6. With the use of the buttons of area *1*, seen in this figure, a mesh model can be loaded respectively saved. Area *2* can be used to deform or transform a mesh and is called the *Transform/Deform area*. With the use of the slider, the kernel size can be determined. The radio buttons are used to choose the desired deformation respectively transformation method. The three buttons and the check box at the bottom of this area provide functionalities to support the user in the segmentation of four dimensional datasets. A click on the *Set Mesh ...* button opens the dialogue shown in figure 5.7. This dialogue contains a check box for every 3D dataset loaded. Area *3* (the *optimization area*) contains four buttons to optimize the mesh shape or mesh topology. The mesh can be smoothed, decimated or subdivided with the use of the particular button. The *Interpolate* button is needed to interpolate vertex positions. With the use of the elements in area *4* an edge detection algorithm can be chosen and its parameters adjusted. This area is called the *edge detection area*. With the two radio buttons at the bottom of the box the edge detection algorithm can be selected. *Canny* stands for the Canny edge detection operator described in subsection 4.4.1 and *LoG* for the Marr-Hildreth operator explained in subsection 4.4.2. The *Kernel Size* field contains the size

Figure 5.2: A vertebra model clipped at the xy-plane.

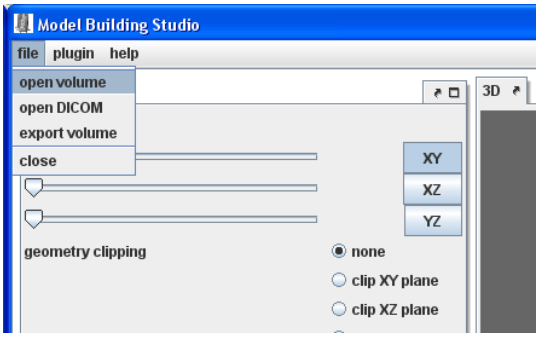


Figure 5.3: Click on *open volume* or *open DICOM* to open a dataset.

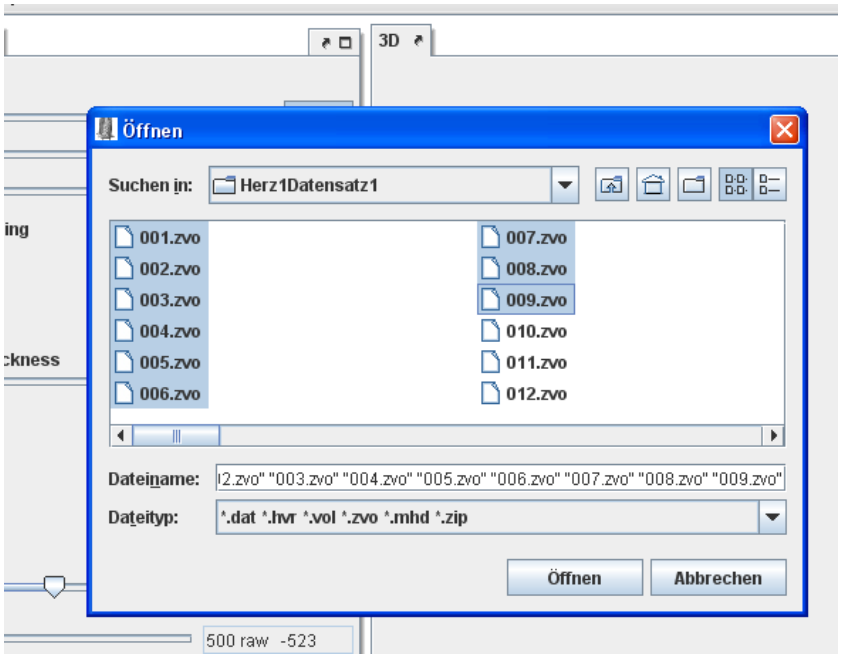


Figure 5.4: The *open volume*- dialogue to open datasets.

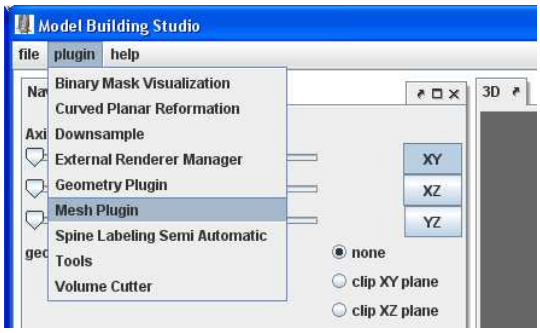


Figure 5.5: Open the Mesh Plugin by clicking on it in the plugin menu.

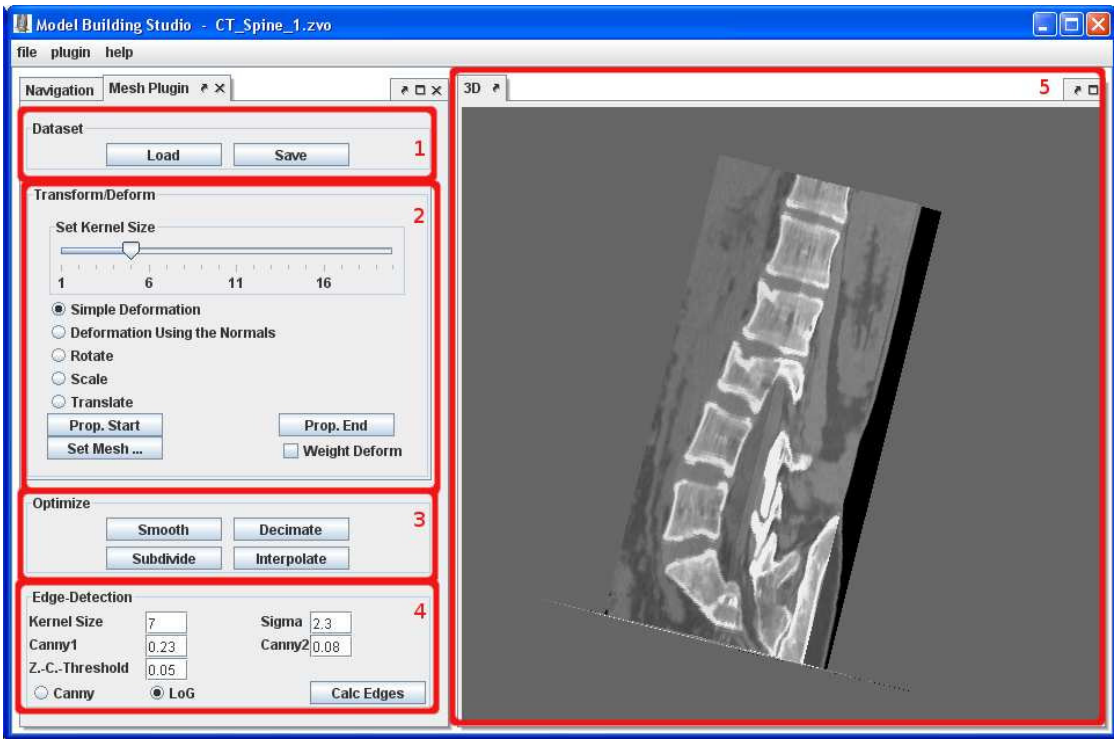


Figure 5.6: The Mesh Plugin offers the user the necessary tools to segment a dataset.

of the convolution kernel and *Sigma* the standard deviation σ. The fields *Canny1* and *Canny2* hold the values for the thresholds $T1$ and $T2$ which are specific for the Canny edge detector. *Z.-C.-Threshold* holds the zero crossing threshold used by the Marr-Hildreth operator. Area *5* contains the 3D area already mentioned above. In this chapter only the GUI parts relevant for this thesis were described. An explanation of the whole GUI would go beyond the scope of this work.

5.3 Data Structures

The used 3D datasets can be stored in different file types that contain the intensity values of the three dimensional image. These files can be zipped with other files that may contain additional information like the slice distance. In the case of 4D datasets it is possible that one file contains all corresponding 3D datasets or that they have to be loaded separately. As this work is not about the storage of medical image datasets, only a concise explanation is given. A short illustration of the used *zvo* file format used in this thesis can be found in appendix A.1.

A Java implementation of the data structure proposed by Tobler and Maierhofer [53] was used to represent the meshes. Figure 5.8 shows the shows the structure of the stored topological information. References from vertices to edges, from edges to faces and from faces to vertices exist. This data structure enables a fast mesh building process and fast access to diverse mesh

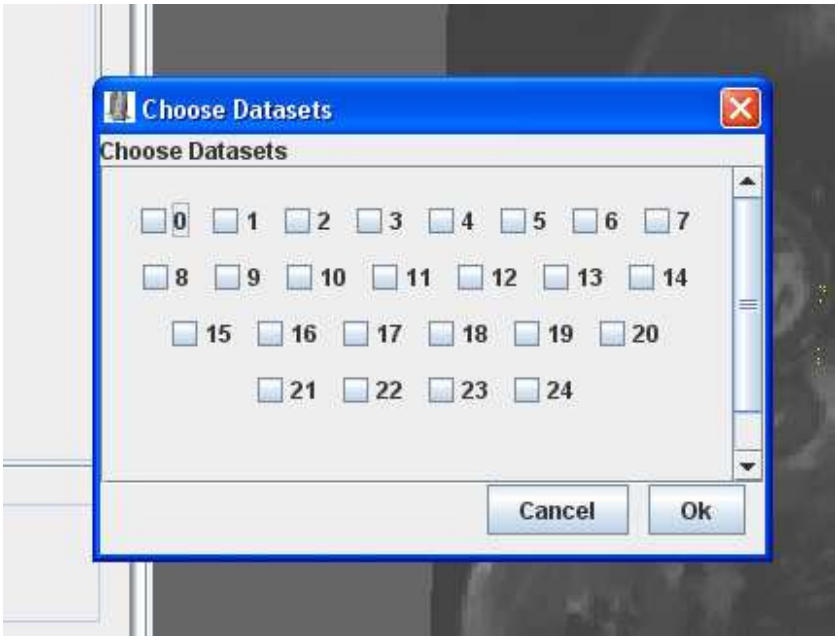


Figure 5.7: With the use of this dialogue the datasets for which the currently active mesh is set as starting mesh for further segmentations can be chosen.

elements like vertices, edges or faces. This is necessary as the whole application is interactive, therefore no performance bottlenecks due to mesh building or mesh accessing are acceptable. Adding vertices to an existing mesh can also be done very fast. A disadvantage is that the deletion of vertices or the substitution of a vertex by another, which is necessary in the case of the half edge collapse, is very costly. After vertices are deleted or substituted the array that stores the vertex positions needs to get re-indexed which is very time consuming. As vertex deletion or substitution is only needed when the decimation algorithm is applied to a mesh and this occurs considerably less often than the access of mesh elements or the need for the mesh building method, this represents an acceptable drawback. When a mesh is stored only an array that contains the vertex positions and an array that indicates which vertex is part of which triangle together with possible attributes like vertex or triangle colours need to be written to a file. To save computation time when the mesh is loaded multiple times, additionally the edges and gradient directions are saved with the mesh by storing the index positions of the edges and pointers to the positions of the corresponding gradient directions in an array.

5.4 Workflow and Program Description

The following subsections explain the work flow as it presents itself to the user who wants to segment a dataset but also the routines and method interactions within the program, that run in the background. Furthermore the values chosen for the specific method parameters are listed.

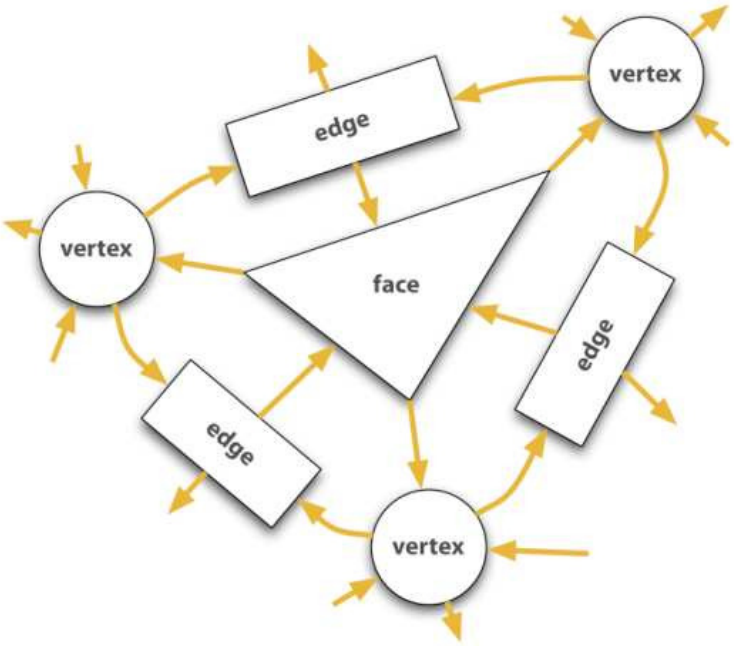


Figure 5.8: Structure of the topological information in the data structure used for mesh representation [53].

5.4.1 Initialization

The first step in a segmentation process is to load a dataset. This can be done by clicking on *file* in the upper left corner and choosing either *open volume* or *open DICOM* in the opened drop down menu. In the appearing file chooser dialogue one or more datasets can be loaded by marking the corresponding file or files and clicking on *Öffnen*. The selection order determines the order of the datasets in the program. A reordering afterwards is currently not possible. The next step is to load a mesh. This is done via the *Load* button of the Mesh Plugin. If the mesh file contains no stored edge information the edges are calculated due to the user settings contained by the fields seen in area *4* in figure 5.6.

5.4.2 Transformation

After the mesh has been loaded the user roughly aligns the mesh to the shape to be segmented with the use of the transformation functions: rotation, scaling and translation. This can be achieved by selecting one of these three functions in the Transform/Deform area in figure 5.6. The transformation is done by clicking in the 3D area with the right mouse button and then dragging the mouse in the desired direction. The following listing gives an overview how the values needed for the transformation methods are calculated.

Rotation: The rotation angle is the angle between the vector that points from the mesh centre to the clicked position and the vector that points from the mesh centre to the current mouse position.

Scaling: To gain the scale factor the difference between the clicked position and the mouse position is calculated. Then the coordinate with the highest difference is taken and divided by the maximal extent of the mesh in this direction. The received value plus one is the scale factor.

Translation: The translation value is simply calculated by taking the difference between the clicked position and the mouse position.

The respective values are used with the particular transformation matrix which is multiplied with every mesh vertex position to calculate the transformation. The mesh transforms simultaneously with the mouse motion. By releasing the mouse button the transformation step is stopped.

5.4.3 Deformation

The deformation methods can be used to change the shape of a mesh and align it to the shape to be segmented in the underlying image. The deformation is done in a view mode that displays only two dimensional cuts through the mesh. This is done to prevent the occlusion of fine details in the dataset by parts of the mesh. Although during the deformation the user sees only a change in the shape of the outline of the cut through the mesh, the whole mesh is deformed.

Initialization

To deform the mesh, the user needs to select one of the two deformation methods shown in area *2* in figure 5.6. By selecting *Simple Deformation* the deformation method explained in subsection 4.7.1 is chosen. By selecting *Deformation Using the Normals* the deformation method described in subsection 4.7.2 will be used by the program. After one of the two possibilities has been chosen the program automatically switches to a view mode in which the outline of the cut through the mesh with the currently viewed image slice is displayed. This border is represented by the 2D cut explained in section 4.6. To give the user a feeling for the 3D changes of the mesh, the 2D cuts two slices above and below the currently viewed slice are also shown. This can be seen in figure 5.9. To mark a region to be deformed, the user clicks on or near the 2D cut. The closest point of the 2D cut to the clicked position is set as deformation centre. Then the deformation area is calculated with the use of the value of the kernel size slider (seen in Figure 5.6, area 2). The kernel size can be changed either with the use of the slider depicted in area *2* in figure 5.6 or by holding the shift button and simultaneously moving the mouse wheel. Which region of the mesh lies within the kernel size is shown by colouring the 2D cut as seen in figure 5.10.

Deformation step

To deform the mesh the user has to click in the 3D area with the right mouse button and drag the mouse to a desired position. The deformation vector is the vector that points from the deformation centre to the actual mouse position. Concurrently with the dragging, all points in

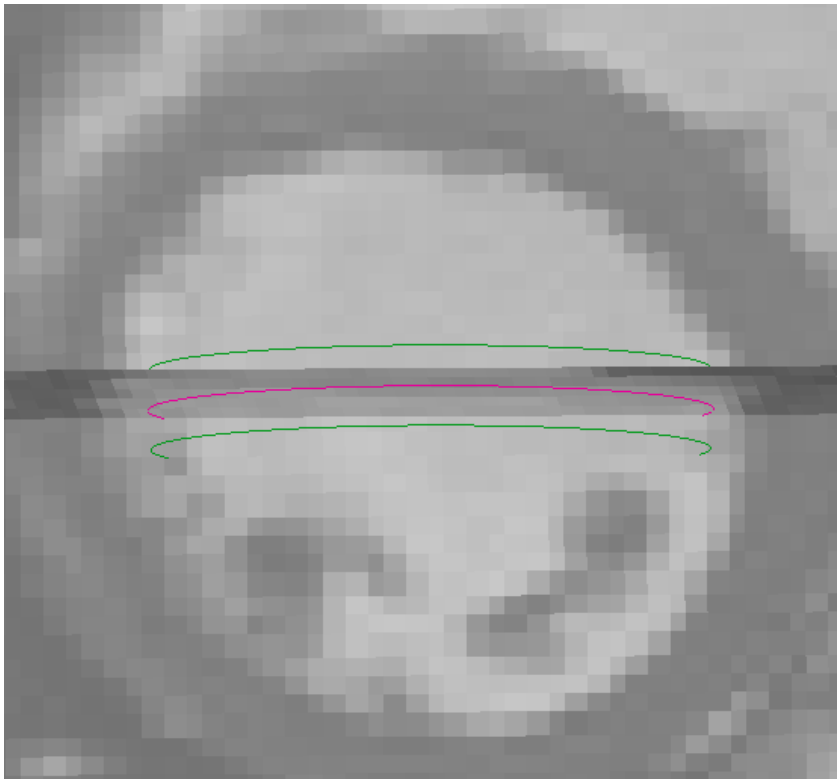


Figure 5.9: The user always sees three cuts through the mesh during deformation. The pink line is the cut at the currently viewed slice. The green lines are two slices above respectively below. This gives the user a feeling of how the mesh changes in 3D when the 2D cut is changed.

the deformation area change their positions according to the Gaussian attenuation function depicted in equation 4.5. The 2D cut is updated concurrently with the mouse movement during the dragging process. The user sees the deformations in real time. The period between the start of dragging and dropping is considered as one deformation step. The kernel size can be changed after every deformation step.

In an approach, that was however discarded, the user was allowed to set an initial kernel size and then increase it automatically the farther the deformation centre was dragged. This strategy was discarded as it would have needed, compared to the final approach, more computational costs to increase the kernel size concurrently with the deformation process and it turned out that this approach was not an intuitive one and confused the user. The confusion arises because the user does not expect the deformation area to increase and often the use of a large deformation area for big deformations is not required. Another drawback can be seen in the deformation of small meshes. For example if the user wants to segment a tumour with a very long but thin excrescence. In this case a small region of the mesh has to be dragged very far. With the use of a concurrently increasing kernel size the whole mesh would very quickly be part of the deformation area and would get deformed. Also a mixed solution was discussed, giving the user

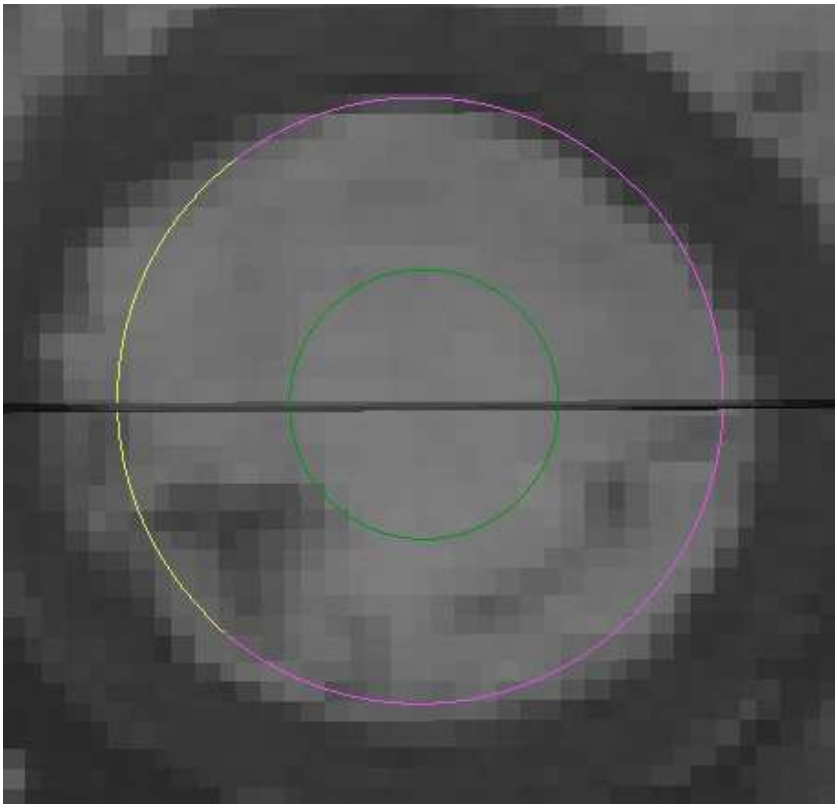


Figure 5.10: The yellow part of the 2D cut marks the deformation area.

the opportunity to choose between a variable and a fixed kernel size. Also the mixed solution was rejected as good solutions can be achieved by just using the fixed kernel size and the user interface should be kept simple and limited to the necessary features.

5.4.4 4D Deformation and Transformation

The program offers two options (already explained in section 4.14) to support the user in a 4D segmentation task. With the first one the user segments a shape in one dataset and then uses the shape as segmentation initialization in the other 3D datasets. The second one works by recording the deformation and transformation operations that are performed to segment one 3D dataset and then propagating the operations to the other datasets.

The first option is done by navigating to the dataset that contains the mesh which should serve as initialization for other datasets either by using the left and right keyboard arrow key or by using the slider seen in area *5* in figure 5.1. After the desired dataset has been reached the dataset is segmented. If the user is satisfied with the result the *Set Mesh ...* button in area *2* of figure 5.6 has to be pressed. The opened dialogue shows a check box for every dataset. By setting the check boxes of datasets active with a check mark and clicking the *Ok* button, the mesh of the current dataset is inserted in the datasets checked in the dialogue.

The second method to support the user in segmenting a 4D dataset is to record all deformation and transformation operations performed by the user and then to apply them to the other datasets. This is done by navigating to the dataset to be segmented in the same way as mentioned above and then clicking the *Prop. Start* button in area *2* of the Mesh Plugin seen in figure 5.6. From this moment all deformations and transformations the user applies on the dataset are recorded.

To stop the recording of the operations and to start the propagation to the other datasets, the *Prop. End* button has to be clicked. This click starts the application of all recorded operations on the other 3D datasets in the same order as they were carried out by the user. For every dataset a single thread is started to run the propagation for different datasets in parallel processes. This is done to speed up the whole process as often a lot of transformation and deformation steps are necessary to segment a shape and the propagation in one single thread would take too much time.

With the use of the check box *Weight Deform* seen in area *2* in the Mesh Plugin in figure 5.6 it is possible to weight the recorded deformation and transformation methods as explained in section 4.14. If the check box is activated the recorded operations are getting down scaled, if not the methods are propagated without any scaling. In figure 5.11 it can be seen how well the natural motion of the heart can be approximated with this method.

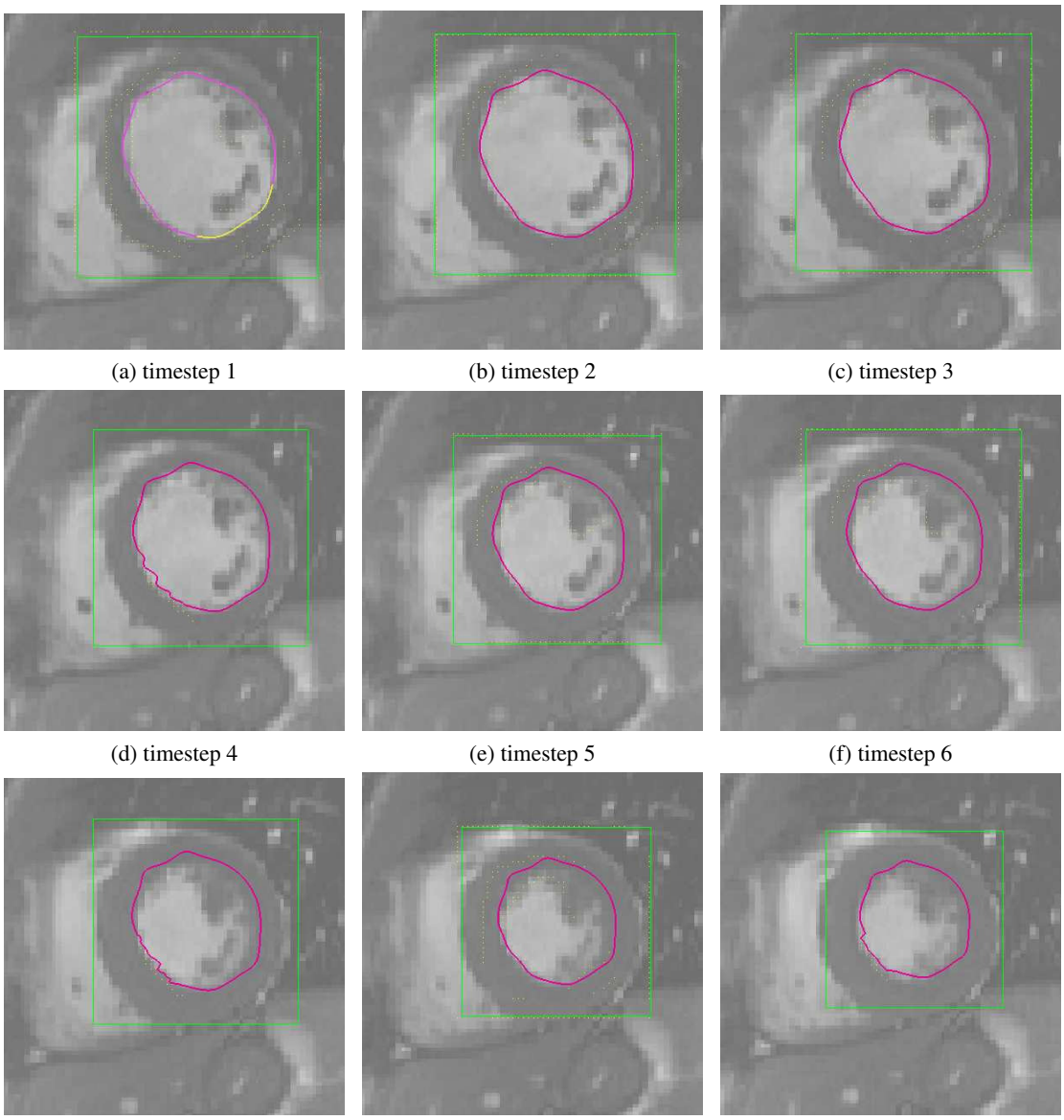
5.4.5 Mesh Optimization

Often, after a lot of deformation and transformation steps, the appearance of a mesh does not meet the requirements of the user any more. For this case four tools were implemented to optimize the mesh topology and mesh shape. With the use of this tools the mesh can be subdivided, decimated and smoothed. Also vertex positions can be interpolated to eliminate so called "valleys" between two slices.

Before commenting in detail on the optimization methods, an explanation of how triangles and vertices of a mesh can be marked by the user has to be given. To mark a single triangle the user just has to click on the mesh triangle. To mark multiple triangles the user has to hold down the *Shift* button of the keyboard, before or after the first triangle has been marked, and then click on the other triangles to mark those. Marked triangles are highlighted to give the user a visual feedback which triangles are already marked. If a triangle is clicked a second time the mark is removed. To mark a vertex whose position is to be interpolated the user has to hold down the *i* button of the keyboard and simultaneously click on the wanted vertex. Figure 5.12 shows highlighted triangles and a vertex that has been marked to interpolate its position.

To subdivide a mesh the user can either mark triangles and then press the *Subdivide* button or simply press the *Subdivide* button. The *Subdivide* button is placed in area *3* of the Mesh Plugin seen in figure 5.6. For the subdivision process the adaptive $\sqrt{3}$-subdivision algorithm described in subsection 4.10.2 of the previous chapter is used.

A mesh can be decimated by simply clicking on the *Decimate* button in area *3* of the Mesh Plugin. The implemented mesh decimation is at this point not capable of handling user input like the subdivision method. As decimation algorithm the method proposed in section 4.11 is used.



(a) timestep 1 (b) timestep 2 (c) timestep 3

(d) timestep 4 (e) timestep 5 (f) timestep 6

(g) timestep 7 (h) timestep 8 (i) timestep 9

Figure 5.11: This time sequence shows how well the motion of the left ventricle can be approximated by weighting the deformation and transformation methods with a Gaussian kernel. (a) dataset where the ventricle was segmented. (b) to (i) datasets where the transformation and deformation operations were propagated. The irregularities in the outline in some images result from the Sticky Edges algorithm.

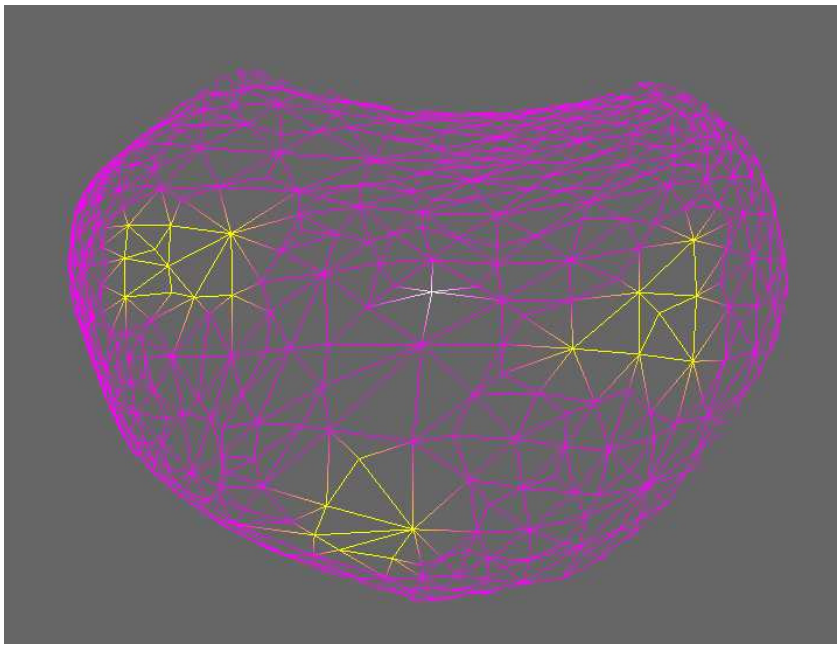
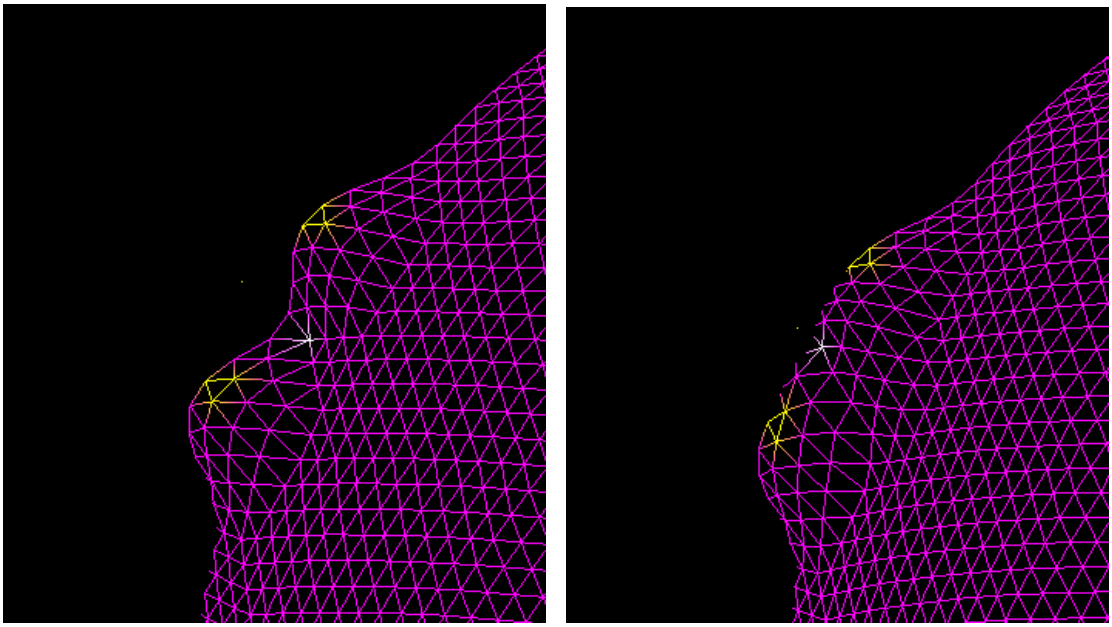


Figure 5.12: The yellow highlighted triangles have been marked by the user. The white vertex in the centre has been marked to interpolate its position.

With the use of the *Smooth* button positioned in area *3* of the Mesh Plugin depicted in figure 5.6 the mesh can be smoothed with the recursive umbrella operator discussed in 4.12.2. In contrast to the method triggered by the *Subdivide* button the user is not able to choose which region is going to be smoothed. The algorithm initiated by the *Smooth* button always treats the whole mesh.

The fourth mesh optimization tool is the interpolation tool. This tool makes use of the method described in the previous chapter in section 4.9 to remove so called "valleys". To remove such a "valley" the user has to mark the vertex whose position has to be interpolated with the use of the *i* keyboard button. The sampling vertices are determined by marking triangles with the use of the *Shift* keyboard button. Every vertex of a marked triangle is used as sampling vertex. If the user is satisfied with his/her choice a click on the *Interpolate* button starts the algorithm. An example of a mesh before and after interpolation can be seen in figure 5.13.



(a) mesh with a “valley” before interpolation

(b) mesh with a “valley” after interpolation

Figure 5.13: The vertices of the yellow marked triangles are the sampling vertices. The position of the white coloured vertex at the “valley” bottom is going to be interpolated. (a) mesh before and (b) mesh after one interpolation step. As can be seen the “valley” has nearly vanished.

CHAPTER 6

Results

6.1 Overview

This chapter shows how the developed program performs and how exact segmentation results can be achieved compared to a segmentation framework developed with MeVisLab. MeVisLab was developed by the MeVis Medical Solutions AG [1] and Frauenhofer MEVIS [38]. MeVisLab is a powerful tool for medical image processing, segmentation, registration and visualization. It is module based. A module offers a specific functionality and can be combined with other modules to resolve complex tasks. For further information about MeVisLab please refer to the program homepage [2].

As no segmentation ground truth exists for the anatomies to be segmented in the available medical image datasets they were manually segmented with the use of MeVisLab at first. The MeVisLab segmentation was done with a Livewire approach. The idea of Livewires is presented in the work of Mortensen and Barrett [41]. The module framework used for segmentation is explained in appendix A.2. The MeVisLab segmentation results were used as ground truth and compared with the results of the program developed in the course of this work.

The comparison was done by taking into account different *figures of merit (fom)* and the time involved. The fom were calculated with the use of PolyMeCo. PolyMeCo is a tool for the analysis and comparison of the characteristics of polygonal meshes. Further information about the tool is provided by Silva et al. [49] and de Santiago [15]. The different fom and the amount of time needed for segmentation provide a good insight how the Mesh Plugin performs in comparison to an approved and well known medical image segmentation approach. All segmentation tasks were performed by the author. The skills of the author in segmenting datasets with the use of the Mesh Plugin as well as with the MeVisLab Livewire approach are comparable.

6.2 Use Cases

6.2.1 Data

Four different datasets were used to validate the Mesh Plugin. Table 6.1 gives an overview on the data characteristics. The following listing provides a more detailed explanation of the datasets.

Dataset 1: This dataset contains a T1 weighted three dimensional MR volume of a human brain. On the upper left part of the parietal lobe is a big tumour located. This tumour is the shape to be segmented. This dataset was chosen as test dataset as the tumour is a very diffuse structure with no clear borders and therefore it is hard to segment. One slice of this dataset can be seen in figure 6.1.

Dataset 2: Images of different modalities are contained in dataset two. The first two scans in the four dimensional dataset are 3D CT scans with different contrast settings. The third one is a three dimensional PET dataset. The three scans depict a human heart. The structure to be segmented is the left ventricle. This dataset was chosen to test how the Mesh Plugin can deal with multi modal datasets. One slice of every 3D volume contained in dataset 2 can be seen in figure 6.2.

Dataset 3: The third dataset is a 3D+T MR scan of the human heart. Unfortunately the MR type is not known. The shape to be segmented is the left ventricle. This dataset contains 25 3D MR scans. The scans were taken every 0.4 seconds, so this dataset can be watched like a short movie of the human heart motion. This dataset was chosen to evaluate how well motion can be segmented with the Mesh Plugin. For a better understanding of the dataset, one slice each of volume 1, volume 10 and volume 25 is depicted in figure 6.3.

Dataset 4: This is the second multi modal dataset. It contains three 3D MR scans of a human brain. A T1 weighted MR scan, a T2 weighted MR scan and a MRA. As this brain shows no obvious pathological structure, the cavum septi pellucidi was chosen as the part to be segmented. This dataset was chosen to test how the Mesh Plugin performs on multi modal datasets. A slice of every scan can be seen in figure 6.4.

The datasets contain different images of different organs and were taken by different modalities so the results presented later on give an overview on how the Mesh Plugin performs in the different areas of medical imaging. Unfortunately it was not possible to get an ultrasound dataset for this paper, which would have further added to the findings in relation to the well known image modalities explained in section 1.1. All four dimensional datasets were registered with the use of the MeVisLab module called *Reformat*. As registration basis always the first volume in a dataset was used.

6.2.2 Compared Systems

The MeVisLab segmentation, which served as ground-truth for the evaluation, was performed with the use of the MeVisLab *LiveWireMacro* module with the predefined parameters for bone images. The segmentation was started by loading a dataset and scrolling through the xy-slices

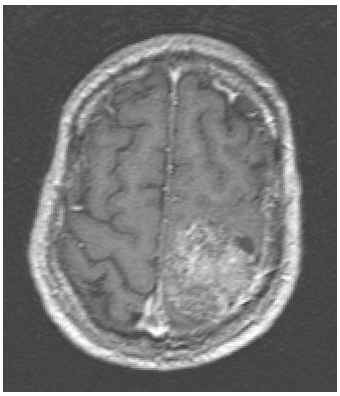
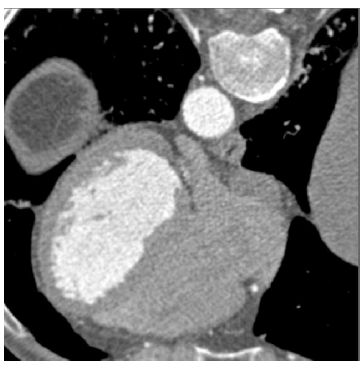
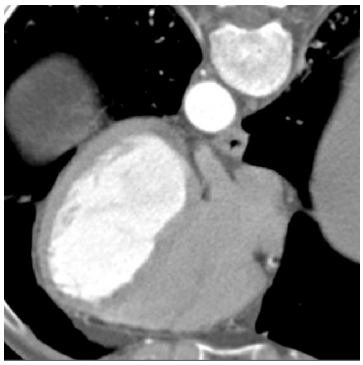


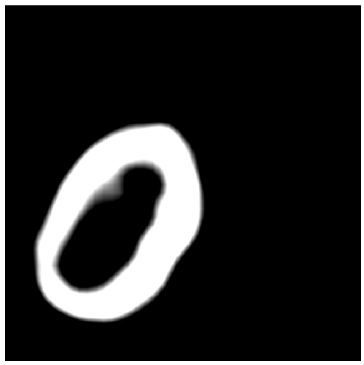
Figure 6.1: A slice of dataset 1. The brighter structure at the bottom of the right hemisphere is the tumour to be segmented.



(a) one slice of the first CT scan of dataset 2



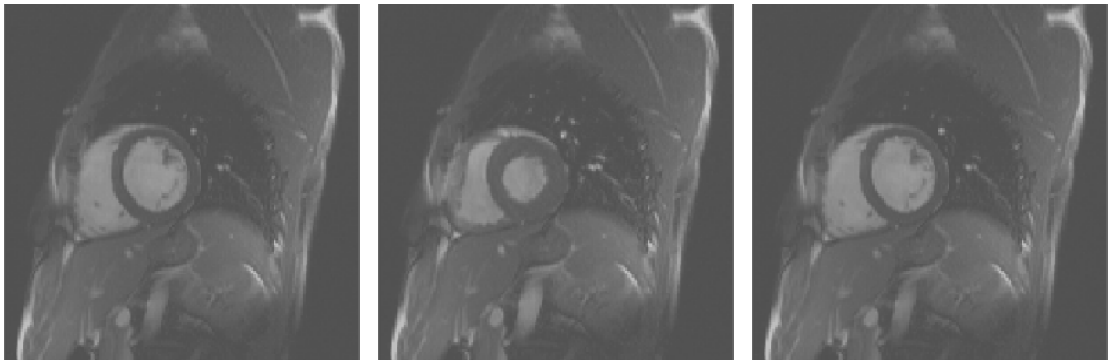
(b) one slice of the second CT scan of dataset 2



(c) a slice of the PET volume of dataset 2

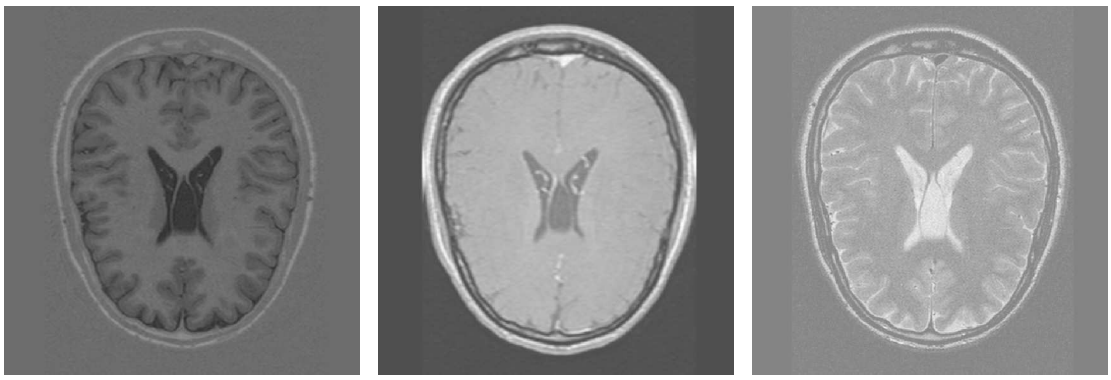
Figure 6.2: Dataset two contains three volumes that give different views of the left ventricle. (a) and (b) show the cavity whereas (c) shows the heart muscle.

until the shape to be segmented was found. Starting from one end of the shape in z-direction the outline of the shape was delineated with the Livewire algorithm in every slice, in every second slice, in every third slice or in every fourth slice. How many slices can be skipped depends on the shape of the structure to be segmented. The bigger the changes in the structure, the fewer slices can be skipped. The outlines of the skipped slices were calculated by interpolating the outlines of the processed slices. Unfortunately the MeVisLab module used for the Livewire algorithm is not able to interpolate over more than four slices. The possibility of skipping more than four slices could have saved a lot of time, especially in the case of dataset 2 with more than 400 slices. With the outlines marked, a binary image mask was generated. With the use of this mask the final mesh was created with the left module branch seen in appendix A.2. Finally the mesh was saved in a Wavefront .obj file. More details about the Wavefront file format can be found in appendix A.3. The Wavefront file format was chosen as PolyMeCo can only deal with this type of format.



(a) volume 1 (b) volume 10 (c) volume 25

Figure 6.3: The three figures show slices of volume 1, volume 10 and volume 25 of dataset 3. In all three figures the muscle of the left ventricle can be seen very well. In (a) the diastole of the heart at the beginning of image recording can be seen. (b) shows a systole of the heart in the middle of the recording and (c) shows the diastole at the end of the recording.



(a) T1 weighted MR scan (b) T2 weighted MR scan (c) MRA acquisition

Figure 6.4: This figure shows a slice of each MR volume in dataset 4. The order of the images correlates with the order of the volumes in the dataset. In each figure the cavum septi pellucidi can be seen. As it is filled with liquor it is dark in image (a) and brighter in image (b) and (c).

Dataset	Organ	Shape to Segment	Resolution (Voxel)	Voxel Size (Millimeter)	Dim	Image Modality
1	brain	tumour	256x256x85x1	0.89x0.89x1.99	3	MR
2	heart	left ventricle	512x512x414x3	0.35x0.35x0.40	4	CT, PET
3	heart	left ventricle	156x192x10x25	1.97x1.97x11.19	4	MR
4	brain	cavum septi pellucidi	512x512x65x3	0.44x0.44x1.19	4	MR

Table 6.1: The table shows the shapes to be segmented in the test datasets, the image resolution, the voxel size, the dimension and the image modality. Voxel size and image resolution are good indicators for the image quality. The higher the resolution and the lower the voxel size are, the higher is the image quality.

The segmentation with the Mesh Plugin was performed by using the LoG (section 4.4.2) edge detection algorithm. The parameters used for the edge detection can be seen in table 6.2. To start the segmentation the specific dataset and a sphere with 20480 triangles was loaded. Then the sphere was dragged to the centre of the shape to be segmented and scaled to a size that serves as good starting point for the exact delineation. After that the sphere was aligned to the borders of the segmentation shape with the use of the available tools. In the case of a four dimensional dataset the mesh produced in one dataset was then propagated to the other datasets either by inserting the mesh or by applying the recorded deformation steps. The last segmentation step was to adjust the propagated meshes in a way that they fit the structure to be segmented. In some test cases, when bigger deformations where necessary, parts of the mesh were subdivided to enable an exact alignment to fine structures. The parameters used for the mesh deformation and the mesh optimization are shown in table 6.3. In the end the generated mesh was saved in a Wavefront .obj file.

Variable	Value
Kernel size:	7
Standard deviation:	2.3
Zero crossing threshold:	0.05

Table 6.2: The parameters used for the LoG algorithm that was used to generate the Mesh Plugin results.

6.2.3 Evaluation Methods

As already mentioned, the comparison of the Mesh Plugin and the MeVisLab meshes was done by comparing the amount of time needed for the segmentations and different figures of merit

Variable	Value
curvature threshold:	0.3
decimation threshold:	0.0006
mesh ratio threshold:	1.2
centre change threshold:	0.1

Table 6.3: The parameters used for the mesh deformation and mesh optimization.

calculated with the use of PolyMeCo. To analyse the generated meshes the intrinsic properties listed below were taken into account.

Smoothness: This measure indicates how smooth the surface of a mesh is. For every vertex the centre of its one ring neighbourhood is calculated and compared with the position of the vertex. The smaller the difference is the smoother is the surface.

Gaussian curvature: The Gaussian curvature is calculated by using methods proposed by Meyer et al. [39].

Mean curvature: The mean curvature is calculated by using methods proposed by Meyer et al. [39].

Triangle quality: For every triangle its minimum angle is calculated. When the minimum angle is close to 60° the triangle is considered to be equilateral which constitutes a good triangle quality.

To compare the meshes produced with MeVisLab and the Mesh Plugin the following fom were used.

Geometric deviation: The geometric distance between the two mesh surface vertices is measured. If the two meshes to compare do not have the same number of vertices the surfaces are sampled to get common points to compare. How the geometric deviation and the vertex correspondences are determined is explained by Roy et al. [43].

Normal deviation: This methods compares how the normals of the vertices change from one mesh to the other one. This method is also based on the work of Roy et al. [43].

Gaussian curvature deviation This fom gives information about the Gaussian curvature deviation. PolyMeCo calculates the Gaussian curvature deviation with methods proposed by Karni and Gotsman [30] and Sorkine et al. [51].

Mean curvature deviation: The mean curvature deviation compares how large the difference in the mean curvature is between two meshes. For this comparison PolyMeCo uses methods discussed by Karni and Gotsman [30] and Sorkine et al. [51].

The temporal requirements to be compared were measured as follows.

MeVisLab: The recording of the time needed to segment a dataset was started when the navigation to the first slice to be segmented was carried out and the segmentation with the Livewire approach was started. The time recording was stopped immediately after the mesh had been generated with the use of the binary mask.

Mesh Plugin: After the mesh of the sphere had been dragged to its initial position (in all cases the centre of the shape to be segmented) the timing of the segmentation started. As soon as the segmentation and the enhancement of the mesh quality by smoothing, decimation and subdivision finished, timing was stopped.

Starting the timing after the navigation to the first slice to be segmented respectively the placing of the sphere at the centre of the shape to be segmented, helped to exclude lag time due to not finding the shape to be segmented in the set of slices and to compare only the time needed for segmentation. If some unexpected errors occurred, like the breakdown of the MeVisLab slice interpolation algorithm because too many slices were skipped or holes occurred in the generated mesh, the segmentation process was aborted and started again. After the time recording had been stopped no further changes were made to the meshes. For both techniques, the first result generated was used for the comparison. This was done to exclude good results due to routine. Before starting the segmentation all datasets were inspected to get a clue of the shape to be segmented.

6.3 Used Hardware

Three different PCs were used to segment the datasets mentioned in the previous section. Table 6.4 shows the details of the three different machines.

Machine	Operating System	CPU	RAM
1	Windows XP, 32-bit	Intel Pentium 4, 3.2 GHz	2 GB
2	Windwos 7, 64-bit	Intel Core 2 Quad Q6600, 2,4 GHz per Core	4 GB
3	Windwos 7, 64-bit	Intel Core i7, 2,67 GHz per Core	12 GB

Table 6.4: This table shows the three different PCs used for testing.

The segmentation of the datasets 1, 2 and 4 with MeVisLab and the Mesh Plugin was performed on machine 1. Due to the huge amount of data in dataset 2 it was not possible to segment it on machine 1 as there was not enough memory available. Therefore the MeVisLab segmentation of dataset 2 was done on machine 2 and the Mesh Plugin segmentation was carried out on machine 3. Machine 3 was needed for the Mesh Plugin in the case of dataset 2 because a big amount of RAM was necessary to store the edge information. In the selection of the machine used for segmentation the main points to be observed where that the application runs seamlessly and smoothly and that the user has no lag time during the segmentation due to computation latency of the machine. Only usability and quality of the segmentation algorithms ought to determine the outcome.

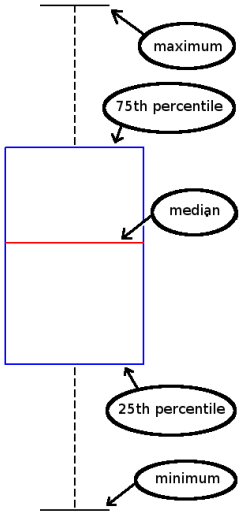


Figure 6.5: This figure explains what kind of information the box-plots used in this paper provide about the underlying data.

6.4 Results

All fom are visualized with box-plots. Figure 6.5 explains shortly how to interpret box-plots. In some plots the dataset numbers have decimal places. This decimal places indicate the number of the volume within a 4D dataset. In some plots the y-axis is scaled logarithmically to give a good view of the data even if there are a lot of outliers. Table 6.5 shows the number of vertices of the different meshes as in some cases it is necessary to know them to understand the result.

6.4.1 Time Needed

The time that was needed to segment the particular datasets either with MeVisLab or the Mesh Plugin is depicted in Figure 6.6. It can be seen that on all four datasets the segmentation with the Mesh Plugin was much faster than with MeVisLab. In the case of the three dimensional dataset 1 the segmentation performed with the Mesh Plugin was 5 times faster than the segmentation with MeVisLab. In the case of the four dimensional datasets the segmentation with the Mesh Plugin was done 7 to 25 times faster than with the LiveWire approach of MeVisLab. The increase of segmentation speed is due to the fact that with MeVisLab nearly every slice has to be segmented separately and with the Mesh Plugin it is possible to deform regions that spread over many slices with an appropriate kernel size. Also the Sticky Edges algorithm helps to align the mesh to the desired edges faster. These advantages of the Mesh Plugin can be seen very clearly from the results of dataset 2 that contains a huge amount of slices. The lower time requirements of the Mesh Plugin in case of four dimensional datasets arises from the possibilities to propagate deformation steps over the fourth dimension and to set an already deformed mesh as start mesh in another volume. This saves a lot of work compared to the MeVisLab LiveWire approach where every volume has to be segmented from scratch.

	Vertices			Vertices	
Mesh	**MeVisLab**	**Mesh Plugin**	**Mesh**	**MeVisLab**	**Mesh Plugin**
1	237	10243	2.13	64	10242
2.1	6323	10243	2.14	71	10242
2.2	2403	10242	2.15	75	10242
2.3	5136	10242	2.16	74	10242
3.1	80	10242	2.17	74	10242
3.2	82	10242	2.18	79	10242
3.3	82	10242	2.19	77	10242
3.4	69	10242	2.20	77	10242
3.5	62	10242	2.21	78	10242
3.6	54	10242	2.22	78	10242
3.7	45	10242	2.23	77	10242
3.8	42	10242	2.24	85	10242
3.9	42	10242	2.25	86	10242
3.10	40	10242	4.1	1007	10294
3.11	46	10242	4.2	6938	10294
3.12	55	10242	4.3	895	10294

Table 6.5: The number of vertices of the meshes produced with MeVisLab and the Mesh Plugin.

6.4.2 Smoothness Analysis

The figures 6.7, 6.8 and 6.9 show the results of the smoothness analysis performed with PolyMeCo. The lower the value, the better is the result. As can be seen the smoothness values of the Mesh Plugin meshes are lower than the values of the meshes produced with MeVisLab. This is because the Mesh Plugin meshes have much more vertices but the same size as the MeVisLab meshes which implies that the distance of a vertex to the centre of its one ring neighbourhood is smaller. The smoothness analysis results for the meshes of the Mesh Plugin as well as the meshes of MeVisLab are acceptable as the median smoothness is only a fraction of the mesh size. For example the median smoothness of the mesh produced with MeVisLab with dataset 1 (the smallest of all meshes) is 0.98 mm. To to enclose the whole mesh a bounding box with the side lengths of 45x50x30 mm would be necessary.

6.4.3 Gaussian Curvature

The results of the Gaussian curvature analysis of the meshes are depicted in figure 6.10 for the datasets 1, 2 and 4, in figure 6.11 for the first part of dataset 3 and in figure 6.12 for the second part of dataset 3. The figure 6.13 shows the Gaussian curvature deviation of the Mesh Plugin and MeVisLab meshes for the datasets 1, 2 and 4. Figure 6.14 shows this fom for dataset 3. The metric unit for the y-axis in all figures is *dioptre (dpt)*. Dioptre is the reciprocal of the focal length measured in metres. The figures show that the Gaussian curvature of all meshes is very low and that the Gaussian curvatures of the MeVisLab and the Mesh Plugin meshes are very

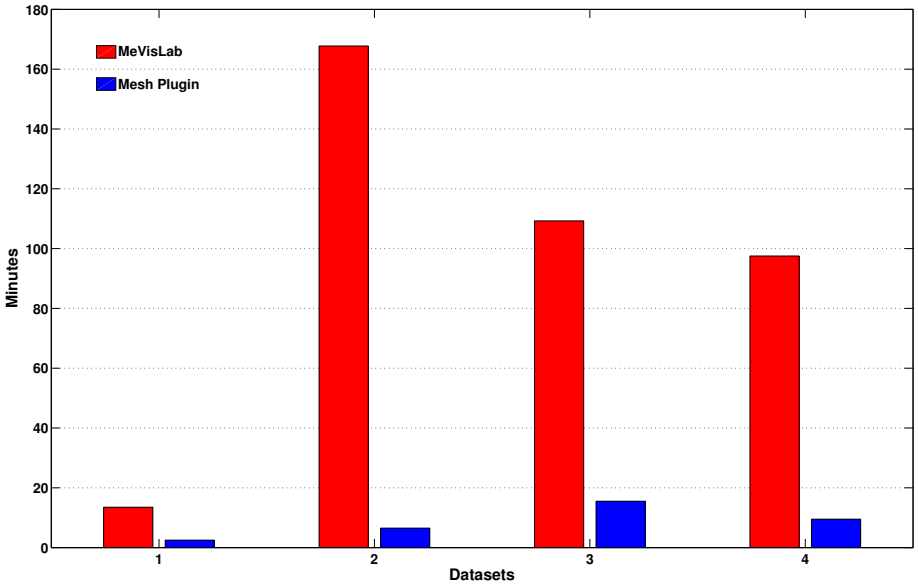


Figure 6.6: Bars indicate the time in minutes that was needed to segment the dataset.

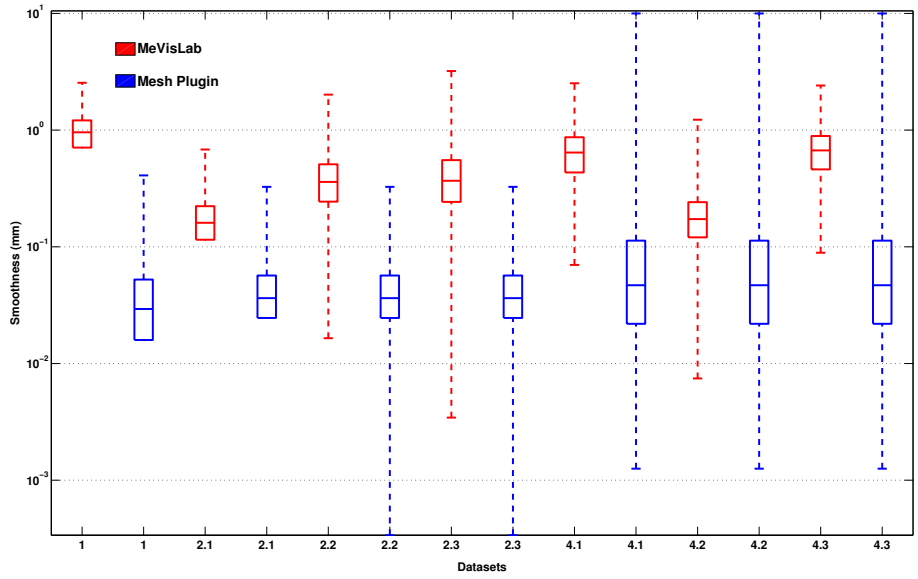


Figure 6.7: The results of the PolyMeCo smoothness analysis algorithm applied on the meshes produced with MeVisLab and the Mesh Plugin with the datasets 1, 2 and 4.

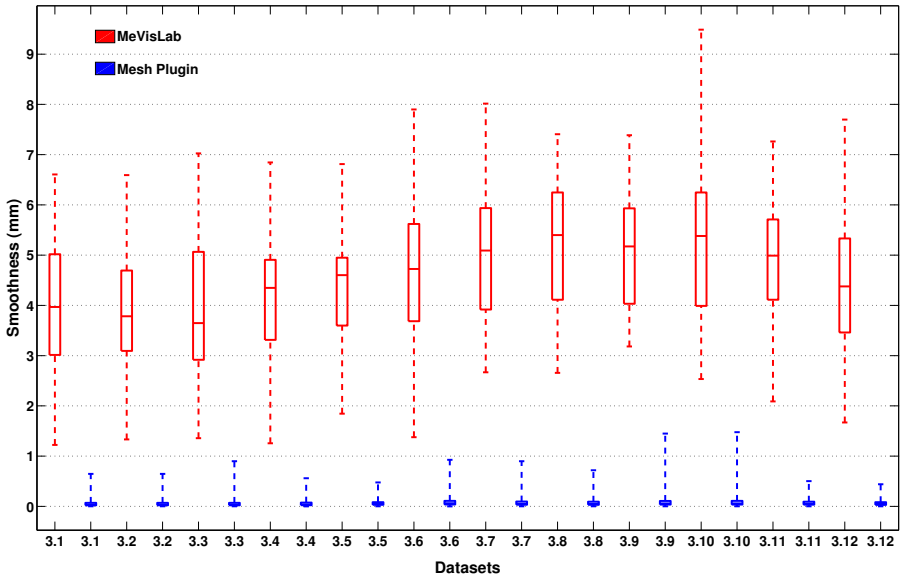


Figure 6.8: The results of the PolyMeCo smoothness analysis algorithm applied on the meshes produced with MeVisLab and the Mesh Plugin with the first 12 volumes of dataset 3.

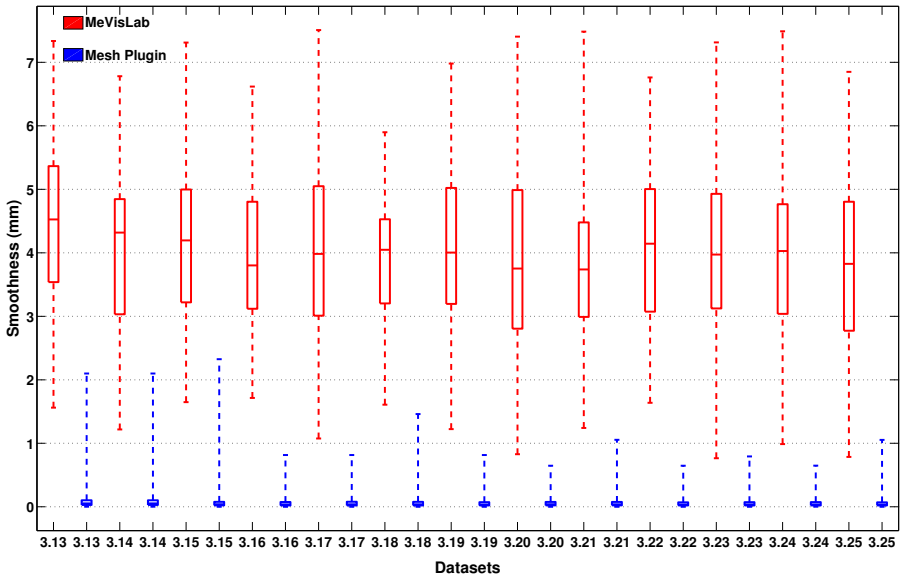


Figure 6.9: The results of the PolyMeCo smoothness analysis algorithm applied on the meshes produced with MeVisLab and the Mesh Plugin with the last 13 volumes of dataset 3.

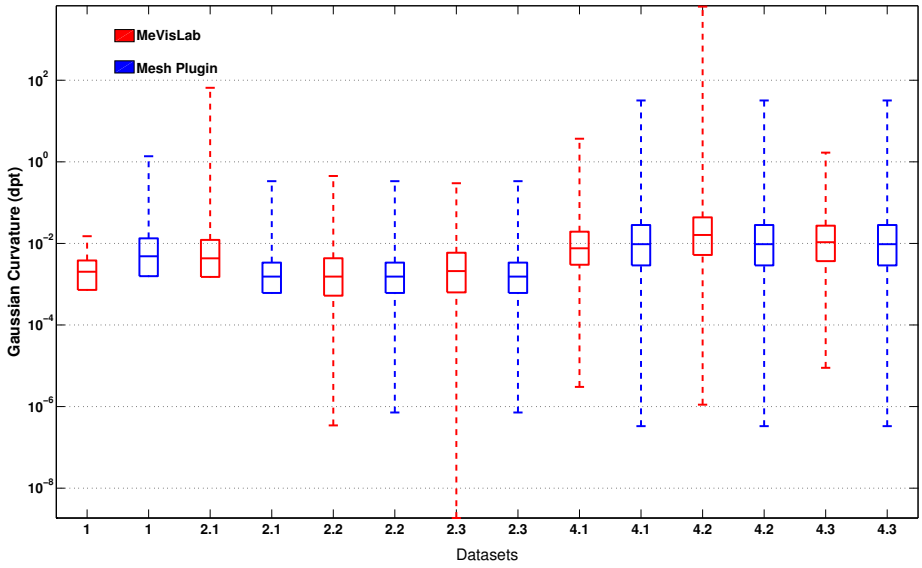


Figure 6.10: The Gaussian curvature analysis results of the meshes produced with MeVisLab and the Mesh Plugin with the datasets 1, 2 and 4.

similar except for some outliers of the Mesh Plugin meshes in dataset 3. These outliers result from an indentation which causes a sharp peak and a narrow valley on the surface and therefore an increase of the Gaussian curvature in that part of the mesh. Figure 6.15 shows the coloured Gaussian curvature analysis of the Mesh Plugin mesh with its segmented indentation and the MeVisLab mesh generated with dataset 3.10.

6.4.4 Mean Curvature

The mean curvature analysis results for the meshes generated with MeVisLab and the Mesh Plugin are presented in figure 6.16 for the datasets 1, 2 and 4, in figure 6.17 for the first part of dataset 3 and in figure 6.18 for the second part of dataset 3. The mean curvature deviation of the MeVisLab and Mesh Plugin meshes is depicted in figure 6.19 for the datasets 1, 2 and 4 and in figure 6.20 for dataset 3. The metric unit for the y-axis in all figures is dioptre (dpt). The results are similar to the results of the Gaussian curvature analysis. The mean curvature of all meshes is very low and the mean curvatures of the meshes generated with the Mesh Plugin and the meshes generated with MeVisLab are similar except for outliers of the Mesh Plugin meshes in datset 3. These outliers are, as already mentioned in the Gaussian curvature analysis 6.4.3, the result of a segmented indentation. This indentation is represented with a small peak and a narrow valley that causes an increase of the mean curvature in that region.

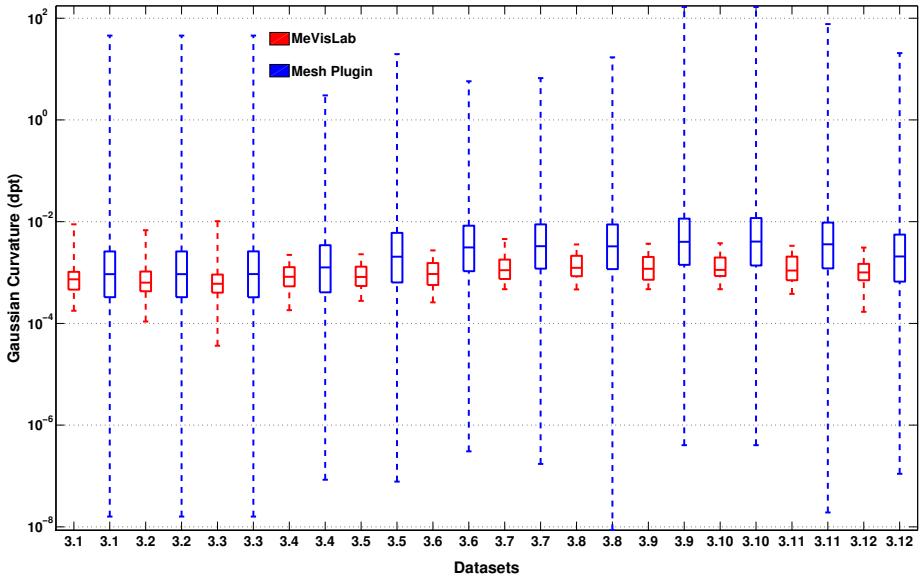


Figure 6.11: The Gaussian curvature analysis results of the meshes produced with MeVisLab and the Mesh Plugin with the first 12 volumes of dataset 3.

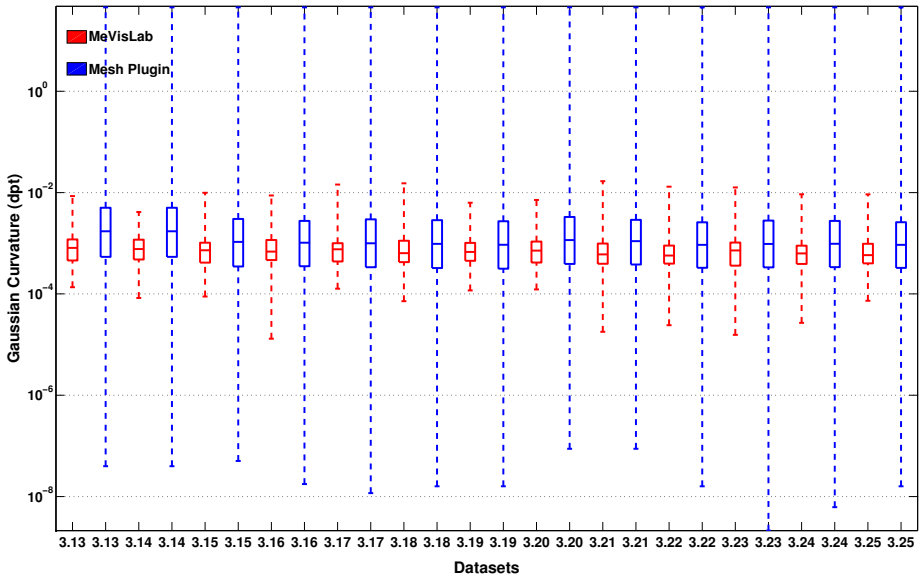


Figure 6.12: The Gaussian curvature analysis results of the meshes produced with MeVisLab and the Mesh Plugin with the last 13 volumes of dataset 3.

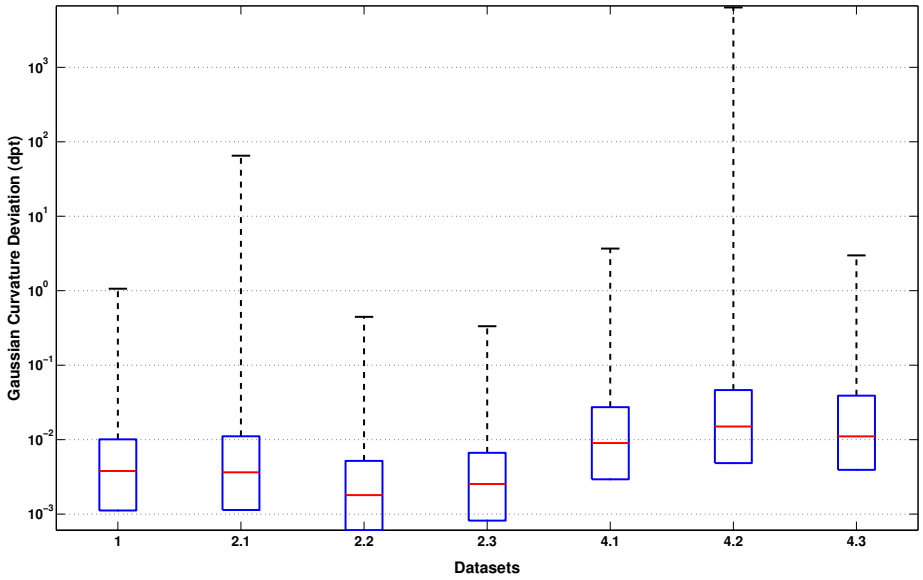


Figure 6.13: The Gaussian curvature deviation between the meshes of the datasets 1, 2 and 4 produced with MeVisLab and the Mesh Plugin.

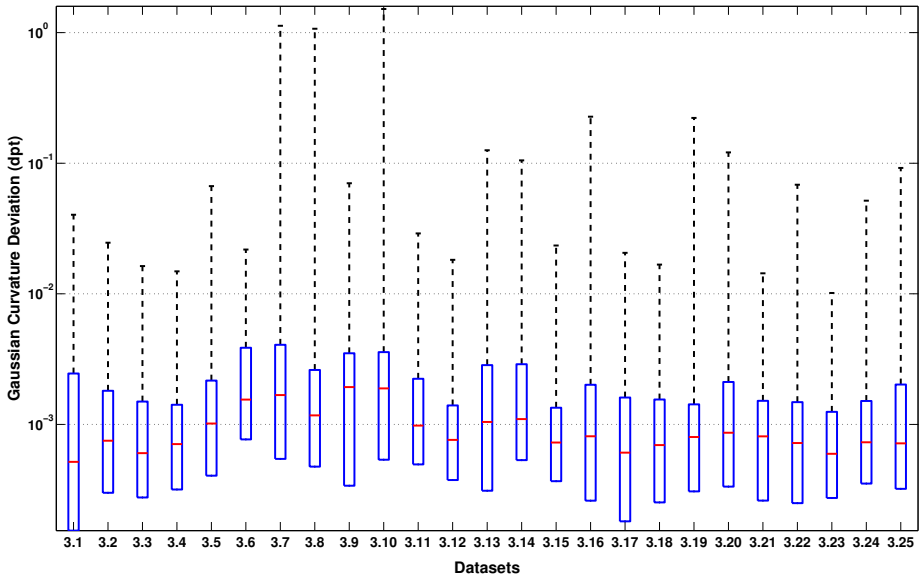
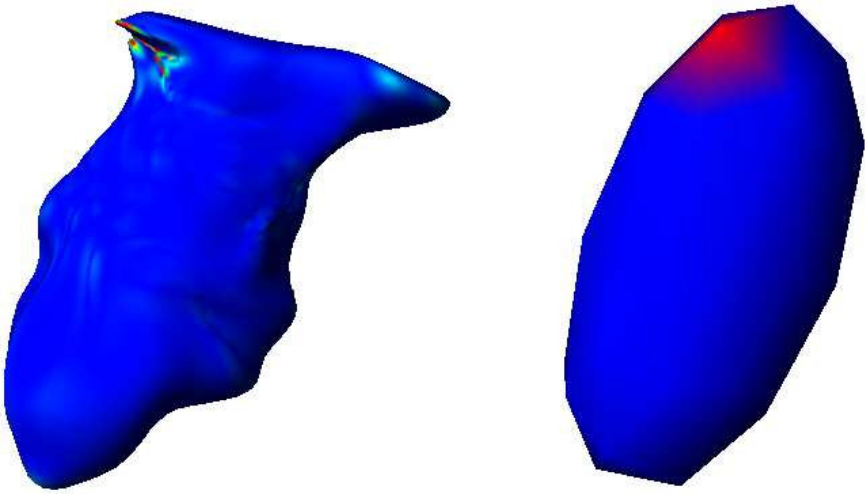


Figure 6.14: The Gaussian curvature deviation between the meshes of dataset 3 produced with MeVisLab and the Mesh Plugin.



(a) Gaussian curvature analysis of the Mesh Plugin mesh of dataset 3.10.

(b) Gaussian curvature analysis of the MeVisLab mesh of dataset 3.10.

Figure 6.15: The Gaussian curvature of the meshes generated with the Mesh Plugin ((a)) and MeVisLab ((b)) with dataset 3.10. The red parts indicate a bad Gaussian curvature value, which can be seen very clearly in the area of the small segmented indentation on the top of the Mesh Plugin mesh. Blue regions indicate a good Gaussian curvature value.

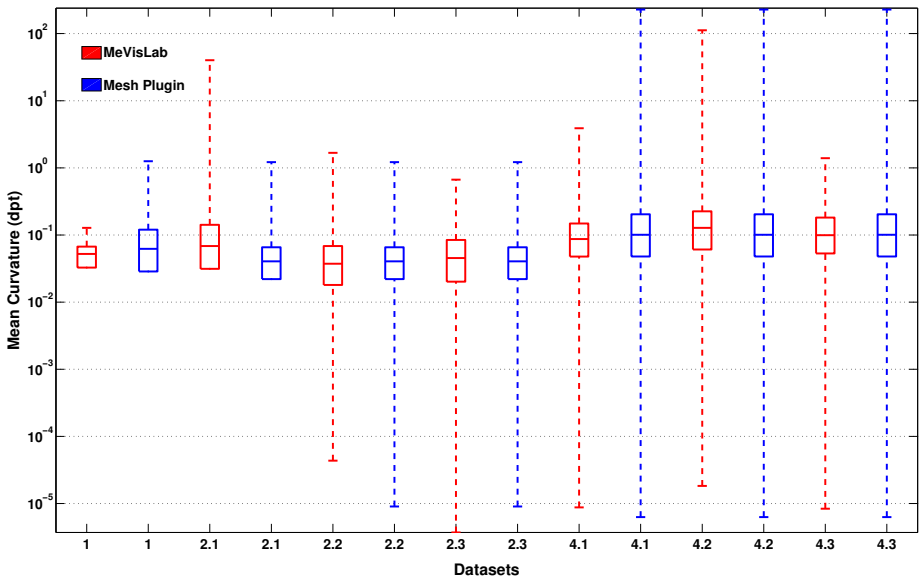


Figure 6.16: The mean curvature of the meshes of the datasets 1, 2 and 4.

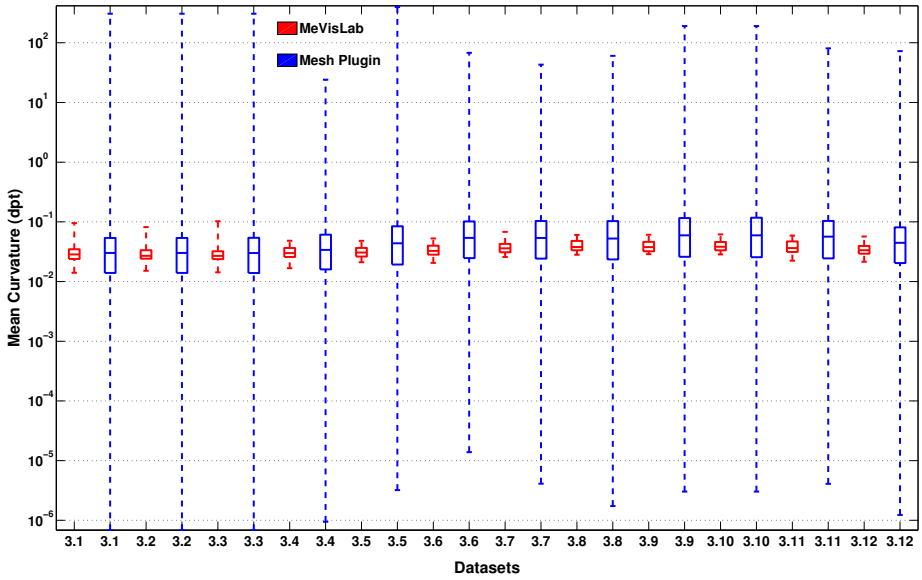


Figure 6.17: The mean curvature of the meshes of the first 12 volumes of dataset 3.

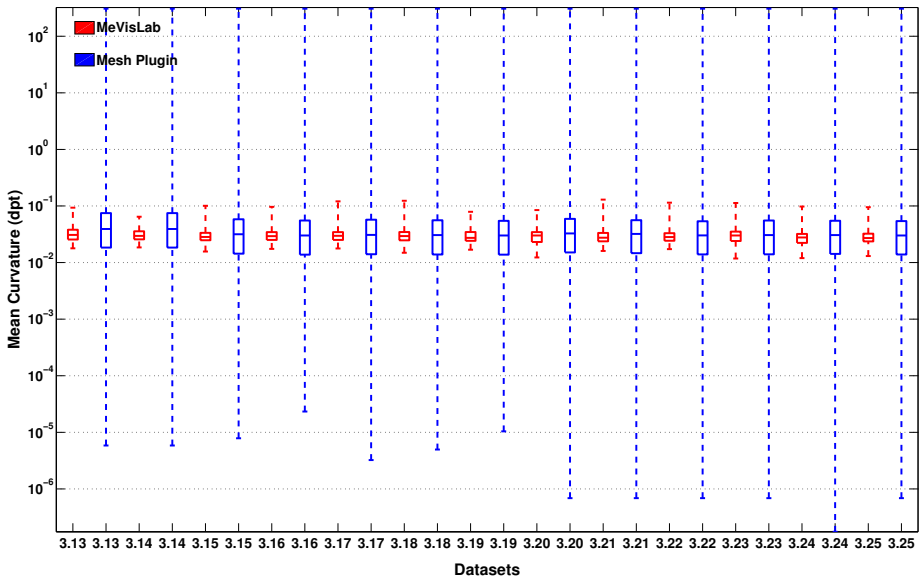


Figure 6.18: The mean curvature of the meshes of the last 13 volumes of dataset 3.

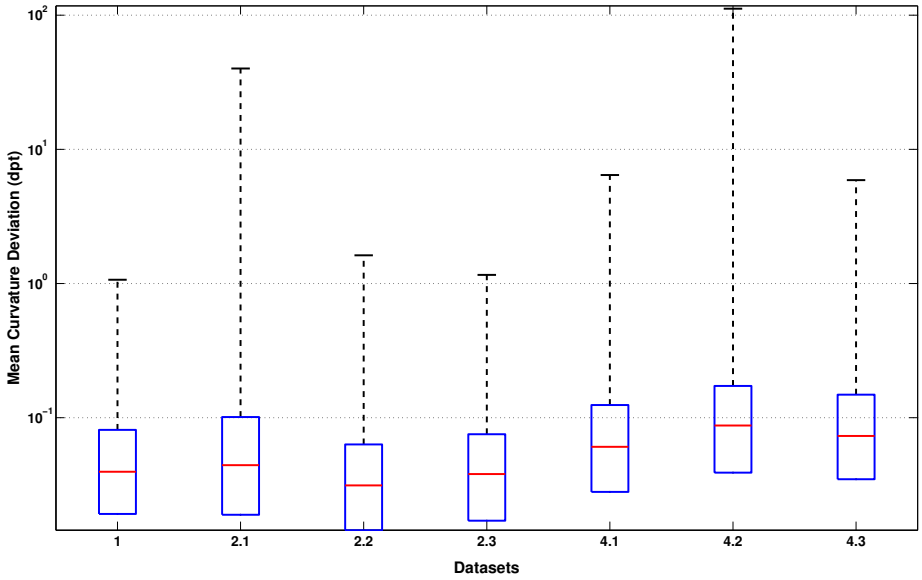


Figure 6.19: The mean curvature deviation between the meshes of the datasets 1, 2 and 4 produced with MeVisLab and the Mesh Plugin.

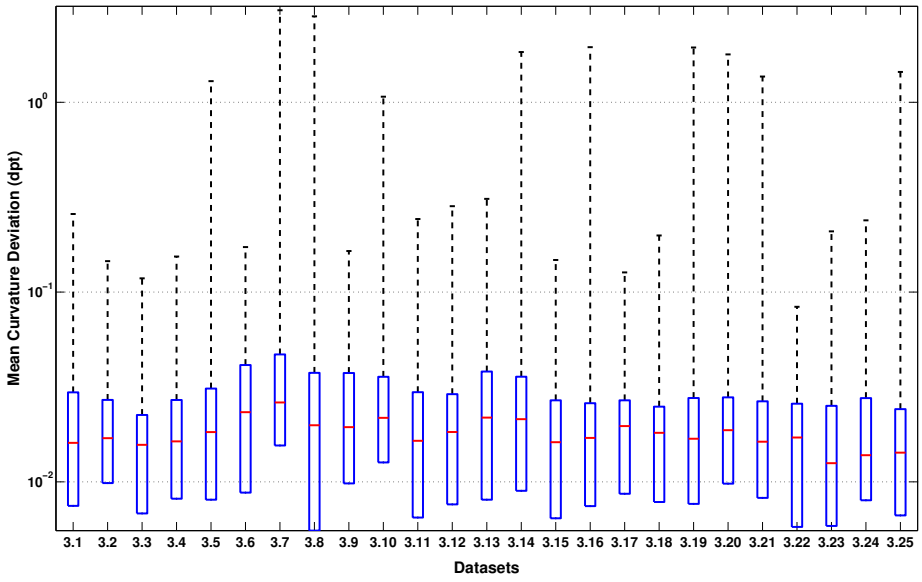


Figure 6.20: The mean curvature deviation between the meshes of dataset 3 produced with MeVisLab and the Mesh Plugin.

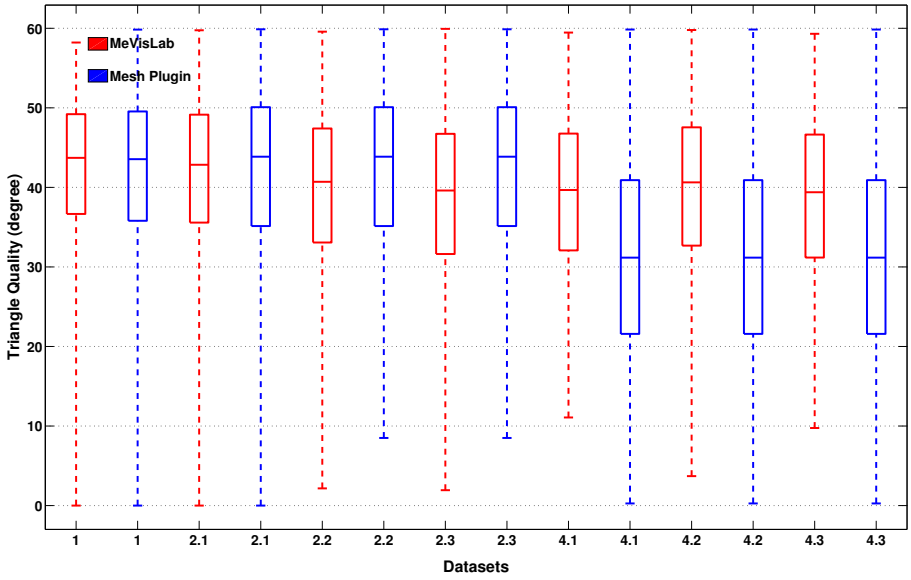


Figure 6.21: The triangle quality of the meshes of the datasets 1, 2 and 4.

6.4.5 Triangle Quality

An overview of the triangle quality analysis results can be seen in figure 6.21 for the datasets 1, 2 and 4 and in the figures 6.22 and 6.23 for dataset 3. The triangle quality of the meshes produced with MeVisLab and the meshes generated with the Mesh Plugin is similar. Both programs provide meshes with a median minimum triangle angle between 30 and 45 degree. The worst triangle quality is present in the meshes produced with the Mesh Plugin for dataset 4. This is due to the fact that the cavum septi pelludici is a very long form and therefore the sphere has to be lengthened which results in elongated triangles with a small minimum angle. Figure 6.24 shows the coloured triangle quality of the meshes produced with the Mesh Plugin and MeVisLab on dataset 4.1. The best results were achieved on the datasets 1 and 3. This is because the shape to be segmented in these datasets is more sphere-like so the triangles do not have to be elongated. As can be seen in figures 6.22 and 6.23 the meshes created with the Mesh Plugin with dataset 3 show some really badly shaped triangles with a minimum triangle angle close to zero. These badly shaped triangles are due to an indentation. In the segmentation of this indentation changes on the mesh with a small kernel size had to be made which elongated the involved triangles.

6.4.6 Geometric Distance

The outcomes of the PolyMeCo analysis regarding the geometric distance between the MeVisLab and the Mesh Plugin meshes can be seen in figure 6.25 for the datasets 1, 2 and 4 and in figure 6.26 for dataset 3. Notice the the big geometric distance in the third volume of dataset 2

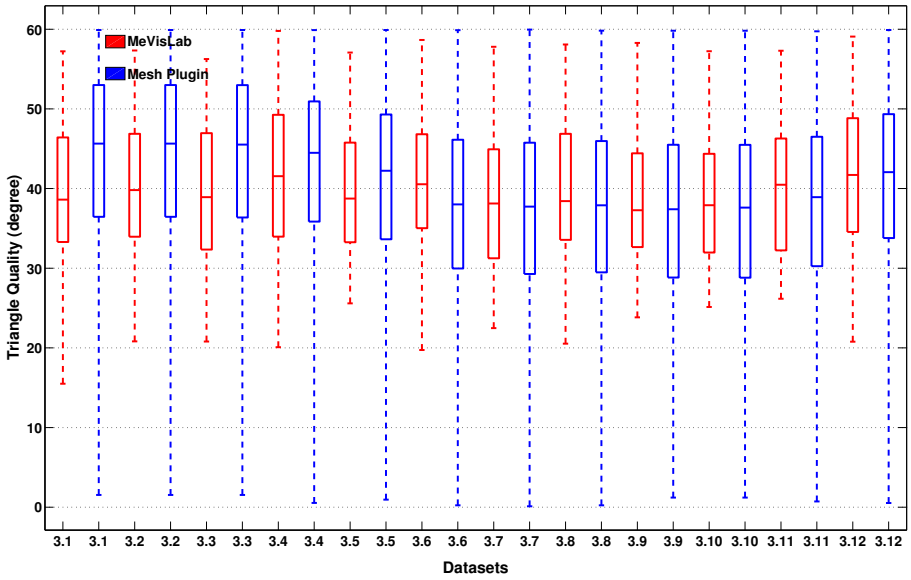


Figure 6.22: The triangle quality of the meshes of the first 12 volumes of dataset 3.

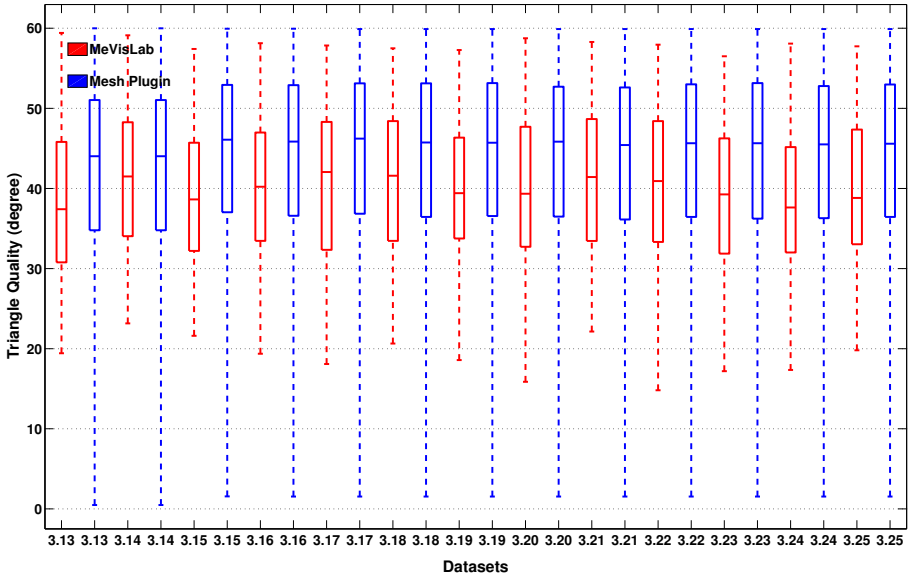
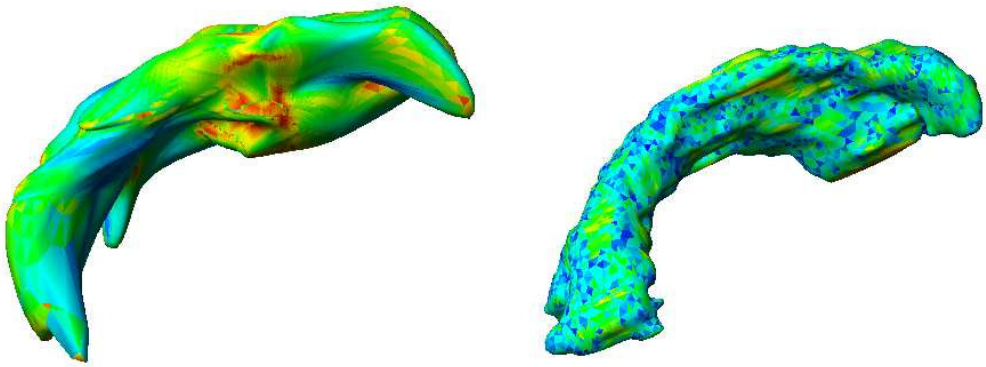


Figure 6.23: The triangle quality of the meshes of the last 13 volumes of dataset 3.



(a) coloured triangle quality of the Mesh Plugin mesh of dataset 4.1

(b) coloured triangle quality of the MeVisLab mesh of dataset 4.1

Figure 6.24: The triangle quality of the Mesh Plugin mesh and the MeVisLab mesh produced with dataset 4.1. Blue triangles indicate good triangle quality. Red triangles indicate a bad triangle quality.

(the PET dataset). This is because in this dataset the heart muscle around the left ventricle is very easy to recognize and with the use of MeVisLab this muscle was segmented whereas with the Mesh Plugin only the ventricle was segmented. This was done because the work flow of the two programs is different. With MeVisLab the segmentation of a volume has to be done from scratch and so the only visible structure in the third volume of dataset 3 was segmented, which was the muscle around the left ventricle. In the case of the segmentation with the Mesh Plugin the left ventricle had already been segmented in the other two volumes. This segmentation was then used as start mesh for volume 3 therefore the basic form was already known and the shape of the mesh only had to get aligned to the inner edges of the muscle seen in volume 3.

6.4.7 Normal Deviation

The normal deviation between the Mesh Plugin and the MeVisLab meshes of the datasets 1, 2 and 4 is depicted in figure 6.27. Figure 6.28 shows the results of the normal deviation analysis for dataset 3. The third volume of dataset 2 shows a very strong deviation of the meshes compared to the other datasets. As already mentioned in 6.4.6, this is because with the use of MeVisLab the muscle around the ventricle was segmented and with the use of the Mesh Plugin the cavity of the left ventricle was segmented. As the muscle is a quasi-hollow pattern and the left ventricle was segmented as closed form the normals of the surface in the inner part of the muscle point to the opposite direction compared to the normals of the ventricle surface. There is no explanation as to why this deviation has a maximum of just 2 degrees as there is no discussion available how PolyMeCo calculates the normal deviation.

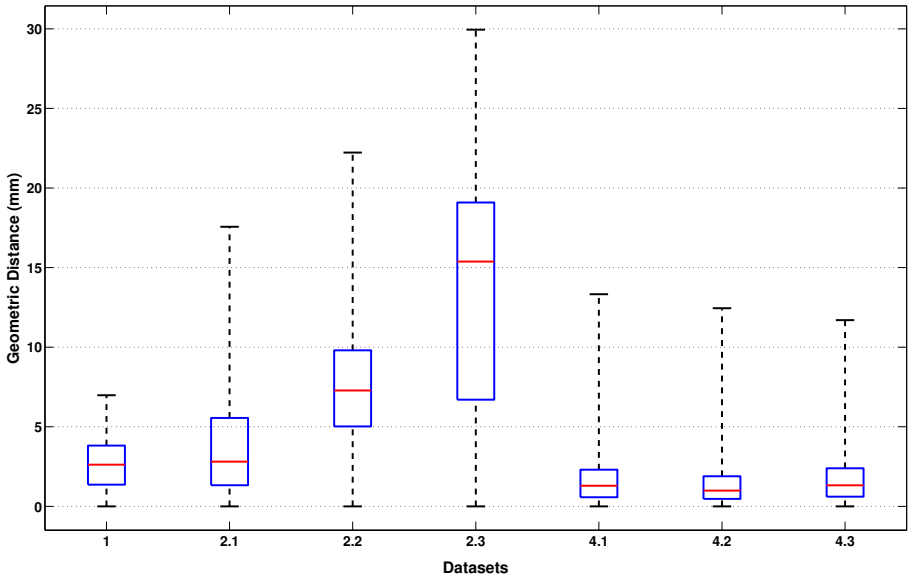


Figure 6.25: The geometric distance between the meshes of the datasets 1, 2 and 4 produced with MeVisLab and the Mesh Plugin.

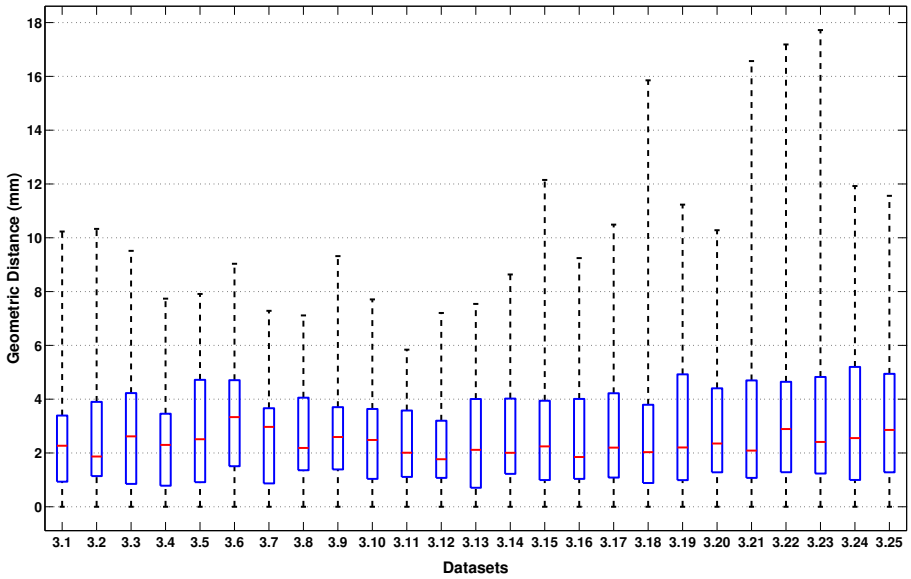


Figure 6.26: The geometric distance between the meshes of dataset 3 produced with MeVisLab and the Mesh Plugin.

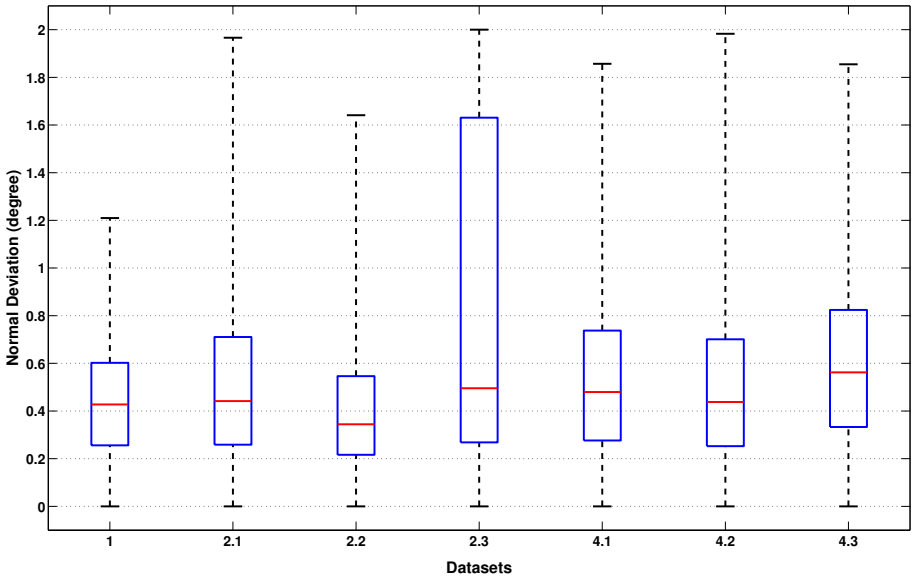


Figure 6.27: The normal deviation between the meshes of the datasets 1, 2 and 4 produced with MeVisLab and the Mesh Plugin.

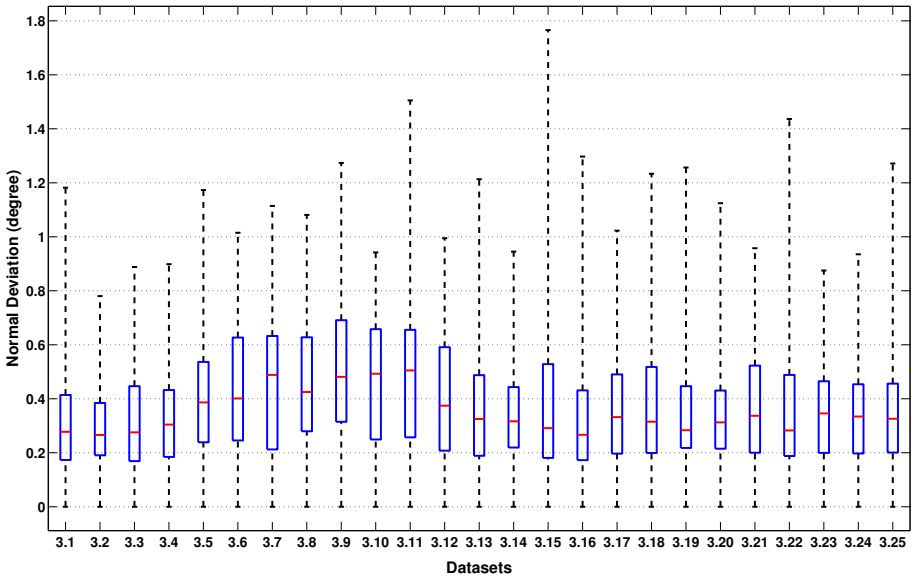
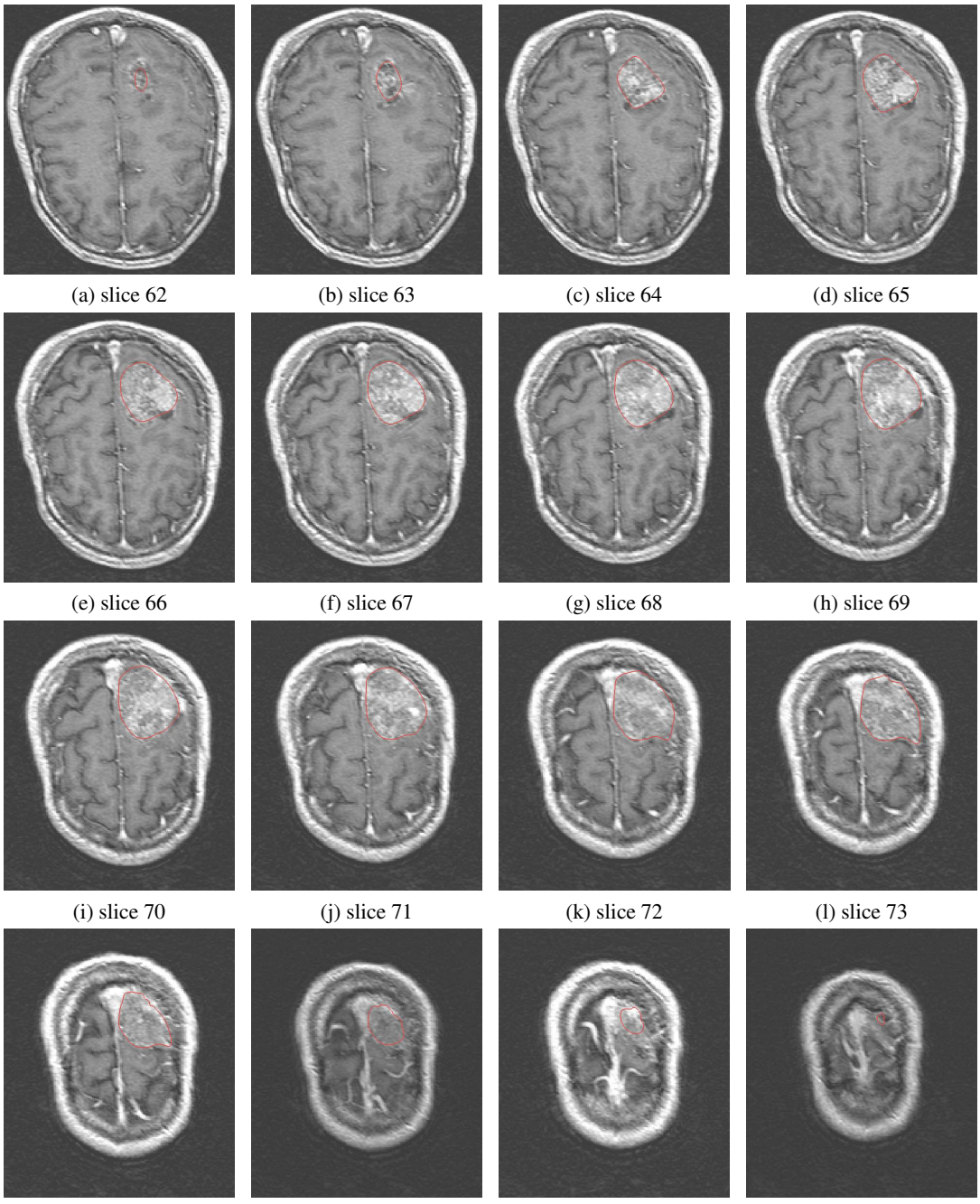


Figure 6.28: The normal deviation between the meshes of dataset 3 produced with MeVisLab and the Mesh Plugin.

6.4.8 Discussion

Summarizing, it can be said that the Mesh Plugin outperformed the Livewire approach realized with MeVisLab regarding the time needed for segmentation. Furthermore the Mesh Plugin produced meshes of good quality. Only in cases with huge deformations or in the segmentation of very sharp structures, bad mesh quality can be established. The results for the segmentation time and the mesh quality can still be improved by using other initial meshes which are more similar to the shape to be segmented. For example an appropriate model would be a model of the organ to be segmented. This would reduce the deformation steps and increase the mesh quality a lot, as the model only has to be adjusted to the anatomical peculiarities of the actual dataset and no big deformations would be necessary. Finally, to complete this chapter and to give one entire segmentation result of the Mesh Plugin, figure 6.29 shows all slices of dataset 1 in which the tumour to be segmented and the outlines of the mesh produced with the Mesh Plugin on dataset 1 are visible.



(a) slice 62 (b) slice 63 (c) slice 64 (d) slice 65

(e) slice 66 (f) slice 67 (g) slice 68 (h) slice 69

(i) slice 70 (j) slice 71 (k) slice 72 (l) slice 73

(m) slice 74 (n) slice 75 (o) slice 76 (p) slice 77

Figure 6.29: Images (a) to (p) show all slices of dataset 1 in which the tumour to be segmented is visible combined with the outlines of the mesh generated with the Mesh Plugin on dataset 1.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

In this thesis a tool is presented that enables the user to manually segment structures of arbitrary shape from scratch and to improve or correct segmentation results of other segmentation methods. The tool is capable of dealing with three and four dimensional datasets of different image modalities. Solely the parameters of the edge detection algorithm have to be adjusted if a dataset of another image modality is loaded.

To segment a 3D dataset an initial mesh is loaded and aligned to the shape to be segmented with simple transformation methods and a novel deformation approach. With the use of this approach the user sees and deforms only the 2D outline of the mesh at a specific slice but in the background the whole mesh is affected by the actions performed. During the deformation the user is supported by the tool to align the outlines of the mesh to the shape to be segmented faster and more exact. Support is provided by the newly developed Sticky Edges algorithm that pushes the mesh vertices in the vicinity of the deformation centre towards an edge if the user has placed the deformation centre on or near this edge. This algorithm uses edge classes to push the vertices only to edges that have the same edge class as the edge on which the user placed the deformation centre. The edges are classified by the use of the edge gradient and their spatial vicinity. In the case of a 4D datasets segmentation, steps performed in one 3D sub-volume can be propagated along the fourth dimension to facilitate the segmentation of the other sub-volumes. The propagation is mesh topology independent. So meshes with different vertex counts can be used in the sub-volumes. To ensure a good mesh quality subdivision, decimation and smoothing methods were implemented that are triggered in the background but can also be applied by the user manually. The tool runs on every customary computer. Just for very big datasets more internal memory is needed because of the large amount of data that arises from the edge detection algorithm. The tool was evaluated by comparing the segmentation results of four different datasets with the segmentation results of a Livewire approach realized with MeVisLab. The comparison was done by matching the amount of time needed for segmentation and different

figures of merit generated with PolyMeCo. The evaluation showed that segmentation with the Mesh Plugin is much faster than the Livewire approach and produces meshes of good quality.

7.2 Future Work

Future continuation of this work, which would aim at preventing errors during the segmentation process, is the implementation of self intersection detection and prevention. Sometimes the outline of one part of a mesh is dragged over the outline of another part of the mesh by the user. Such a self intersection of the mesh does not exist in nature and is therefore not necessary. Also, it is very hard to undo in the current version of the program. A detection of self intersections and the subsequent prevention of such deformations by the tool would prevent these errors.

A decrease in computation time can be achieved by implementing a hierarchical data structure to store the vertex positions. Such a data structure would make the detection of the closest point to an arbitrary point in the dataset, for example the clicked position of the user, easier and faster.

As a lot of the methods presented in this work are based on the outcome of the edge detection algorithm, an edge preserving smoothing algorithm that enhances the outcome of the edge detection algorithm would increase the quality of the results and decrease the time needed for segmentation. This enhancements would be due to the fact, that the Sticky Edges algorithm could align the vertices around the deformation centre to an edge the more efficiently the better the outcome of the edge detection algorithm is. A possible approach can be found in the work of Wang [54].

To improve the 4D propagation a method would be useful that finds for every vertex in one mesh the closest related vertex of another mesh, independent of the vertex count of both meshes. The closeness of two points should be determined by their position and their curvature so that for example the tip of the nose can be found even if the head is turned to the side in one volume. These relationships would have to be calculated after loading the meshes and after every change in topology but would avoid the calculation of the deformation area for every propagated deformation step for every mesh.

APPENDIX A

Appendices

A.1 ZVO File Format

The zvo file format or *zipped volume object* is used in this work to store three and four dimensional image datasets. To store a dataset the image data need to be present in a *dat* or *raw* file format. The dat file format is in this case also a raw file with additional information about the volume size. In the case of four dimensional datasets for every time step respectively modality one 3D dataset is needed.

In addition to the files that contain the image data either a *hvr* or *vol* file is created during the saving process. These two files provide information about the image dataset so that the dataset is displayed correctly. Both file types are text files that contain a list of all 3D volumes. In the case of a 3D dataset the list has only one entry, in a 4D case the list order determines the order of the volumes in the Mesh Plugin. Also the voxel size for the datasets is stored in the file. Only one slice distance is stored. So it is assumed that all datasets have the same voxel size. The image data files and the information file are zipped to the zvo file in the ZIP format.

A.2 MeVisLab Module Framework

The MeVisLab modules used to segment the medical image datasets and to create the ground truth meshes can be seen in figure A.1. With the modules outgoing from the *ImgLoad* module, the binary masks that indicate the segmentation were generated with the use of the Livewire approach. The modules origin from the *LoadMask* module was used to generate meshes utilizing the binary masks generated in the first step. More information about the modules is provided by the documentation on the MeVisLab site [2].

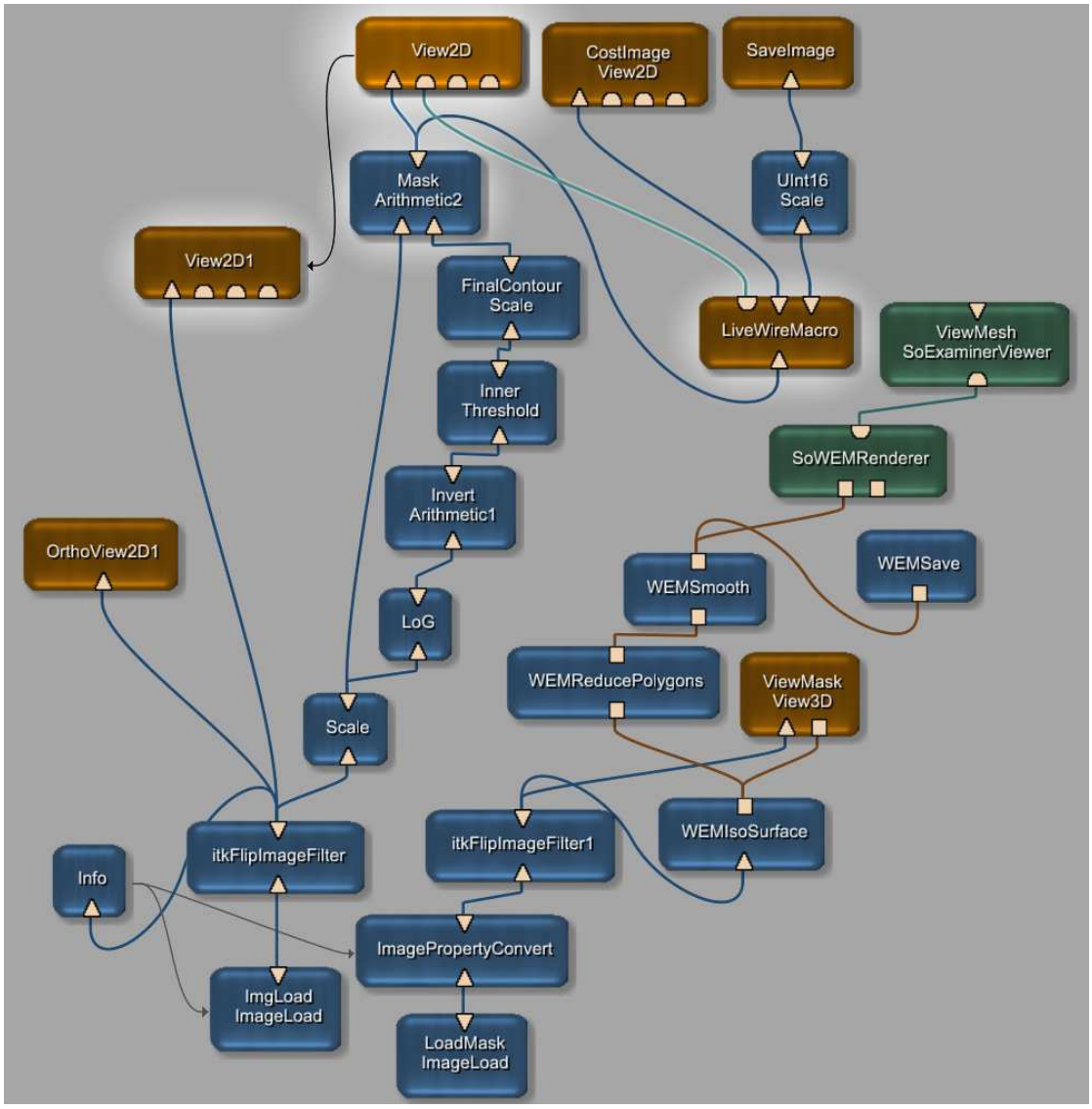


Figure A.1: The MeVisLab modules and their combination used to generate the ground truth meshes that were used to evaluate the Mesh Plugin.

A.3 Wavefront file format

The Wavefront file format is a format developed by Wavefront Technologies [55]. This section gives only a short overview on the parts of the file format used for this work. In this work it was only necessary to store triangles and the vertices with their positions that are part of the triangles. This is done by generating a text file and initially writing every vertex with its position in one line. To store three vertices, one with position 2/4/5, one with position 2/3/3 and one with position 4/3/5 the file looks like the following:

```
v  2  4  5
v  2  3  3
v  4  3  5
```

where v denotes a vertex, the first digit is the x-coordinate, the second one the y-coordinate and the last one the z-coordinate. To store a triangle that is composed of the above mentioned vertices the file has to be extended by one line. This would look like:

```
v  2  4  5
v  2  3  3
v  4  3  5
f  2  1  3
```

where the f denotes a face and the following numbers show the vertex indices. The indices in the Wavefront format start at 1. The vertices are stored in a counter clockwise order.

Bibliography

[1] MeVis Medical Solutions AG. MeVis Medical Solutions, http://www.mevis.de/mms/index.html. Accessed: 2011-05-17.

[2] MeVis Medical Solutions AG. MeVisLab, medical image processing and visualization, http://www.mevislab.de/. Accessed: 2011-05-17.

[3] M.R. Anderberg. *Cluster analysis for applications*. Academic Press, New York, 1973.

[4] I.N. Bankman. *Handbook of medical imaging: processing and analysis*. Academic Press Series in Biomedical Engineering. Academic Press, 2000.

[5] H.G. Barrow and J.M. Tenenbaum. Interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence*, 17(1-3):75 – 116, 1981.

[6] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[7] A. Bornik, R. Beichel, and D. Schmalstieg. Interactive editing of segmented volumetric datasets in a hybrid 2D/3D virtual environment. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '06, pages 197–206, New York, NY, USA, 2006. ACM.

[8] M. Botsch, M. Pauly, L. Kobbelt, P. Alliez, B. Lévy, S. Bischoff, and C. Rössl. Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07. ACM, 2007.

[9] M. Botsch, M. Pauly, M. Wicke, and M. Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, 2007.

[10] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:679–698, November 1986.

[11] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *Int. J. Comput. Vision*, 22:61–79, February 1997.

[12] T.F. Chan and L.A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266 –277, February 2001.

[13] R. Chandrashekara, A. Rao, G. Sanchez-Ortiz, R. Mohiaddin, and D. Rueckert. Construction of a statistical model for cardiac motion analysis using nonrigid image registration. In *Information Processing in Medical Imaging*, volume 2732 of *Lecture Notes in Computer Science*, pages 599–610. Springer Berlin / Heidelberg, 2003.

[14] C.T. Metz, M. Schaap, T. van Walsum, A.G. van der Giessen, A.C. Weustink, N.R. Mollet, G. Krestin and W.J. Niessen. 3D segmentation in the clinic: A grand challenge II - Coronary artery tracking. *Insight Journal*, 2008.

[15] Campus Universitário de Santiago. PolyMeCo - Polygonal Mesh Analysis and Comparison Tool, http://www.ieeta.pt/polymeco/. Accessed: 2011-05-17.

[16] M. del Fresno, M. Vénere, and A. Clausse. A combined region growing and deformable model method for extraction of closed surfaces in 3D CT and MRI scans. *Computerized Medical Imaging and Graphics*, 33(5):369 – 376, 2009.

[17] M. Desbrun, M. Meyer, P. Schröder, and A.H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[18] J. Dornheim, L. Dornheim, B. Preim, I. Hertel, and G. Strauß. Generation and initialization of stable 3D mass-spring models for the segmentation of the thyroid cartilage. In *DAGM-Symposium*, pages 162–171, 2006.

[19] L. Dornheim, K.D. Tönnies, and J. Dornheim. Stable dynamic 3D shape models. In *IEEE International Conference on Image Processing*, volume 3, pages 1276–9, 2005.

[20] M. Erdt, M. Kirschner, and S. Wesarg. Smart manual landmarking of organs. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7623, March 2010.

[21] W. Faulkner and E. Seeram. *Rad Tech's Guide to MRI: Basic Physics, Instrumentation, and Quality Control*. Wiley-Blackwell, 2001.

[22] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. *HIPR - The Hypermedia Image Processing Reference*. John Wiley & Sons Ltd, 1996.

[23] O. Gerard, A.C. Billon, J.-M. Rouet, M. Jacob, M. Fradkin, and C. Allouche. Efficient model-based quantification of left ventricular function in 3-D echocardiography. *IEEE Transactions on Medical Imaging*, 21(9):1059 –1068, September 2002.

[24] A. Goshtasby. Design and recovery of 2-D and 3-D shapes using rational Gaussian curves and surfaces. *International Journal of Computer Vision*, 10:233–256, 1993.

[25] M. Hussain, Y. Okada, and K. Niijima. Efficient and feature-preserving triangular mesh decimation. In *Proceedings of WSCG'04*, pages 167–174, 2004.

[26] M. Jackowski and A. Goshtasby. A computer-aided design system for revision of segmentation errors. In *Proceedings of the 8th International Conference on Medical image computing and computer-assisted intervention - Volume Part II*, MICCAI'05, pages 717–724. Springer-Verlag, 2005.

[27] D. Kainmueller, T. Lange, and H. Lamecker. Shape constrained automatic segmentation of the liver based on a heuristic intensity model. In *Proceedings of the MICCAI Workshop 3D Segmentation in the Clinic: A Grand Challenge*, pages 109 – 116, 2007.

[28] D. Kainmueller, H. Lamecker, H. Seim, S. Zachow, and H.-C. Hege. Improving deformable surface meshes through omni- directional displacements and mrfs. In *Proceedings of the 13th International Conference on Medical Image Computing and Computer-Assisted Intervention: Part I*, MICCAI'10, pages 227–234. Springer-Verlag, 2010.

[29] Y. Kang, K. Engelke, and W.A. Kalender. Interactive 3D editing tools for image segmentation. *Medical Image Analysis*, 8(1):35 – 46, 2004.

[30] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 279–286. ACM Press/Addison-Wesley Publishing Co., 2000.

[31] R. Kindermann and J.S. Laurie. *Markov Random Fields and Their Applications (Contemporary Mathematics ; V. 1)*. Amer Mathematical Society, 1980.

[32] L. Kobbelt. $\sqrt{3}$-subdivision. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 103–112. ACM Press/Addison-Wesley Publishing Co., 2000.

[33] L. Kobbelt, S. Campagna, J. Vorsatz, and H.P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 105–114. ACM, 1998.

[34] C.T. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, U.S.A., August 1987.

[35] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[36] D. Marr and E. Hildreth. Theory of Edge Detection. *Royal Society of London Proceedings Series B*, 207:187–217, February 1980.

[37] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91 – 108, 1996.

[38] Fraunhofer MEVIS. Fraunhofer MEVIS, http://www.mevis.fraunhofer.de/. Accessed: 2011-05-17.

[39] M. Meyer, M. Desbrun, P. Schröder, and A.H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *Building*, 3(7):1–26, 2002.

[40] J. Montagnat and H. Delingette. 4D deformable models with temporal constraints: application to 4D cardiac image segmentation. *Medical Image Analysis*, 9(1):87 – 100, 2005.

[41] E.N. Mortensen and W.A. Barrett. Intelligent scissors for image composition. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 191–198. ACM, 1995.

[42] M. Poon, G. Hamarneh, and R. Abugharbieh. Efficient interactive 3d livewire segmentation of complex objects with arbitrary topology. *Computerized Medical Imaging and Graphics*, 32(8):639 – 650, 2008.

[43] M. Roy, S. Foufou, and F. Truchetet. Mesh comparison using attribute deviation metric. *International Journal of Image and Graphics*, 4(1):127–140, January .

[44] D. Salomon. *Curves and Surfaces for Computer Graphics*. Springer New York, 2006.

[45] N.S. Sapidis. *Designing fair curves and surfaces: shape quality in geometric modeling and computer-aided design*. Geometric Design Publications. Society for Industrial and Applied Mathematics, 1994.

[46] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 151–160. ACM, 1986.

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

[48] V. Sherrow. *Medical Imaging (Great inventions)*. Marshall Cavendish Benchmark, 2006.

[49] S. Silva, J. Madeira, and B.S. Santos. POLYMECO - A Polygonal Mesh Comparison Tool. In *Proceedings of the Ninth International Conference on Information Visualisation*, pages 842–847, Washington, DC, USA, 2005. IEEE Computer Society.

[50] O. Sorkine and A. Nealen. A note on laplacian mesh smoothing. Submitted for publication, 2006, http://rutgers.academia.edu/AndrewNealen/Papers/236778/A_Note_on_Laplacian_Mesh_Smoothing, Accessed: 2011-08-02.

[51] O. Sorkine, D. Cohen-Or, and S. Toledo. High-pass quantization for mesh encoding. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 42–51, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[52] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 351–358. ACM, 1995.

[53] R.F. Tobler and S. Maierhofer. A mesh data structure for rendering and subdivision. In *Proceedings of WSCG (International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision)*, pages 157–162, 2006.

[54] X. Wang. On the gradient inverse weighted filter [image processing]. *Signal Processing, IEEE Transactions on*, 40(2):482 –484, February 1992.

[55] Wikipedia, the free encyclopedia. Wavefront Technologies, http://en.wikipedia.org/wiki/Wavefront_Technologies. Accessed: 2011-06-01.

[56] D. Yang, J. Zheng, A. Nofal, D. Joseph, and I.M. El Naqa. Techniques and software tool for 3D multimodality medical image segmentation. *Journal of Radiation Oncology Informatics*, 1(1):1–21, 2009.

[57] T.S. Yoo. *Insight Into Images: Principles and Practice for Segmentation, Registration, and Image Analysis*. Ak Peters Series. A K Peters, 2004.

[58] R. Yu, Z. Liu, and Y. Liu. A local $\sqrt{3}$-subdivision algorithm preserving sharp features. In *Control, Automation, Robotics and Vision, 2006. ICARCV '06. 9th International Conference on*, pages 1 –6, dec 2006.

[59] S. Zambal. *Anatomical Modeling for Image Analysis in Cardiology*. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Austria, March 2009.