Segmentation of ray and shark tesserae

David Knötel
Matrikelnummer: 4220582
david.knoetel@fu-berlin.de

Masterarbeit in Informatik am Fachbereich Mathematik und Informatik
der Freien Universität Berlin (FUB) angefertigt am Konrad-Zuse-Zentrum
für Informationstechnik Berlin (ZIB) in Kooperation mit dem
Max-Planck-Institut für Kolloid- und Grenzflächenforschung (MPIKG)

Gutachter:
Prof. Dr. Helmut Alt (FUB)

Betreuer:
Dr. Daniel Baum (ZIB)

Kooperationspartner:
Dr. Mason Dean (MPIKG),
Ronald Seidel (MPIKG)

Berlin, den 29.09.2014

Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit mit dem Titel *Segmentation of ray and shark tesserae* selbstständig und nur unter Zuhilfenahme der angegebenen Quellen erstellt habe.

Berlin, den 29.09.2014
David Knötel

Abstract

Rays and sharks are cartilaginous fishes. Most of the cartilaginous skeleton is covered with calcified tiles to improve the stability of the skeleton. These tiles are called tesserae and enclose areas of uncalcified cartilage. Because of the special properties of the tesserae, biologists are interested to understand shape and structure of tessellated cartilage.

This thesis presents a segmentation pipeline for the separation of tesserae on the cartilaginous skeleton of rays and sharks. The segmentation pipeline consists of an automatic initial segmentation step followed by manual error corrections by the user. The initial segmentation is based on the contour tree data structure that tracks the evolution of level sets in a dataset during iso-value changes. The presented segmentation concepts are not limited to the segmentation of tesserae but also viable for similar kinds of tiled structures. The input datasets are given as micro-CT scans.

The contribution of this thesis is the development of a segmentation pipeline. The pipeline uses a newly developed fast version of the contour-tree-based segmentation algorithm that, after a preprocessing step, does not need to iterate over all voxels in the dataset. Visualizations and computations are done with the software system ZIBAmira. Used algorithms are either implemented as new ZIBAmira modules or they extend already existing ZIBAmira modules.

Zusammenfassung

Rochen und Haie sind Knorpelfische. Der Großteil ihres Knorpelsekeletts ist von einer Schicht mineralisierter Plättchen mit hohem Kalziumgehalt überzogen, um die Stabilität des Skeletts zu erhöhen. Diese Plättchen heißen Tesserae und umschließen Bereiche bestehend aus unmineralisiertem Knorpel. Aufgrund der speziellen Eigenschaften von Tesserae sind Biologen daran interessiert, ihre Form and Struktur zu verstehen.

Die vorliegende Arbeit präsentiert eine Segmentierungspipeline, die einzelne Tesserae auf dem Knorpelskelett von Rochen und Haien voneinander trennt. Die Segmentierungspipeline besteht aus einer automatischen Erstsegmentierung gefolgt von manueller Fehlerkorrektur durch den Benutzer. Die Erstsegmentierung basiert auf dem Contour Tree, einer Datenstruktur, die die Evolution von Niveaumengen während der Änderung von Isowerten verfolgt. Die zu segmentierenden Datensätze sind als Mikro-CT Aufnahmen gegeben.

Der Hauptbeitrag dieser Arbeit ist die Entwicklung der Segmentierungspipeline. Diese Pipeline benutzt eine neuentwickelte schnelle Version des Contour-Tree-basierten Segmentierungsalgorithmus, der, nach einem initialen Vorbereitungsschritt, nicht über alle Voxel des Datensatzes iterieren muss. Visualisierungen and Berechnungen werden mit Einsatz des Softwaresystems ZIBAmira erledigt. Dabei werden verwendete Algorithmen entweder als neue Module implementiert oder bestehende Module werden erweitert.

Acknowledgments

First of all I want to thank my instructor at the Zuse Institute Berlin, Daniel Baum, for his support and his help during text revision. I also want to thank my supervisor Helmut Alt. Further thanks go to our collaboration partners Mason Dean and Ronald Seidel at the Max Planck Institute of Colloids and Interfaces. This project is funded by the Human Frontier Science Program (HFSP) that also enabled the presentation of this work at the poster session of the Tomography for Scientific Advancement symposium (ToScA) in London.

Contents

Chapter 1

Introduction

Image segmentation is the process that partitions an input dataset into segments that share some kind of similar property. This thesis presents an image segmentation pipeline based on the contour tree data structure to solve a specific segmentation problem in a biological context.

1.1 Motivation

Over the last decade, better access to micro-CT scanners has given researchers working in biological fields new possibilities to explore their data. More and more facilities like natural history museums scan their collections. The exchange of digitized datasets between different researchers is much easier than the exchange of their physical counterparts. Furthermore, the existence of micro-CT scans enables new ways to analyze the data. For example, virtual histologies made with the help of slices through micro-CT scans do not destroy the object. So the amount of available digital data that can be accessed and analyzed with visualization software, is increasing all the time and gives image processing algorithms like image segmentation interesting applications.

Studying the properties of natural material can lead to new important insights. If it can be achieved to understand structure and growth process, one can try to imitate natural materials for the human benefit.

An example for such interesting natural materials is the skeleton of rays and sharks. Rays and sharks are cartilaginous fishes, so their skeleton does not consist of bones but of cartilage. This has advantages and disadvantages. Cartilage is more flexible but the jaw of sharks needs also a high stability. Locations, where strength is very important, are fortified with little tiles of calcified cartilage, called tesserae (singular is tessera). Typically, such a

(a) (b)

Figure 1.1: Photographs: (a) angelshark (Squatina) braincase; (b) close-up of tesseral structures. The pictures are taken from a presentation of Mason Dean, our collaboration partner at the Max Planck Institute of Colloids and Interfaces.

tessera is around 200-400 micro-meter wide and around 150-300 micro-meter deep. Figure 1.1 shows photographs where this tiling structure is visible.

Figure 1.2 shows the structure of tesserae in multiple resolutions including cross-sections. The layer of tesselated cartilage encloses uncalcified cartilage as shown in Figure 1.2 (right top image) resulting in a material that is flexible and stable. Small parts connecting two neighbored tesserae are called intertesseral joints, see Figure 1.2 (right bottom image). Biologists study shape and structure of tesserae [1] and are interested in how properties like the size or the number of tesserae are changing with the age of the animal.

Image segmentation is a very powerful concept and an important part of many applications in medicine or biology. A segmentation to separate individual tesserae is a first step for further algorithmic-driven analyses of tesseral structures. This is the goal of this thesis: to provide algorithms to segment individual tesserae in micro-CT scans. Of course, such segmentation algorithms are not limited to only this purpose but can also be applied to other tiled structures.

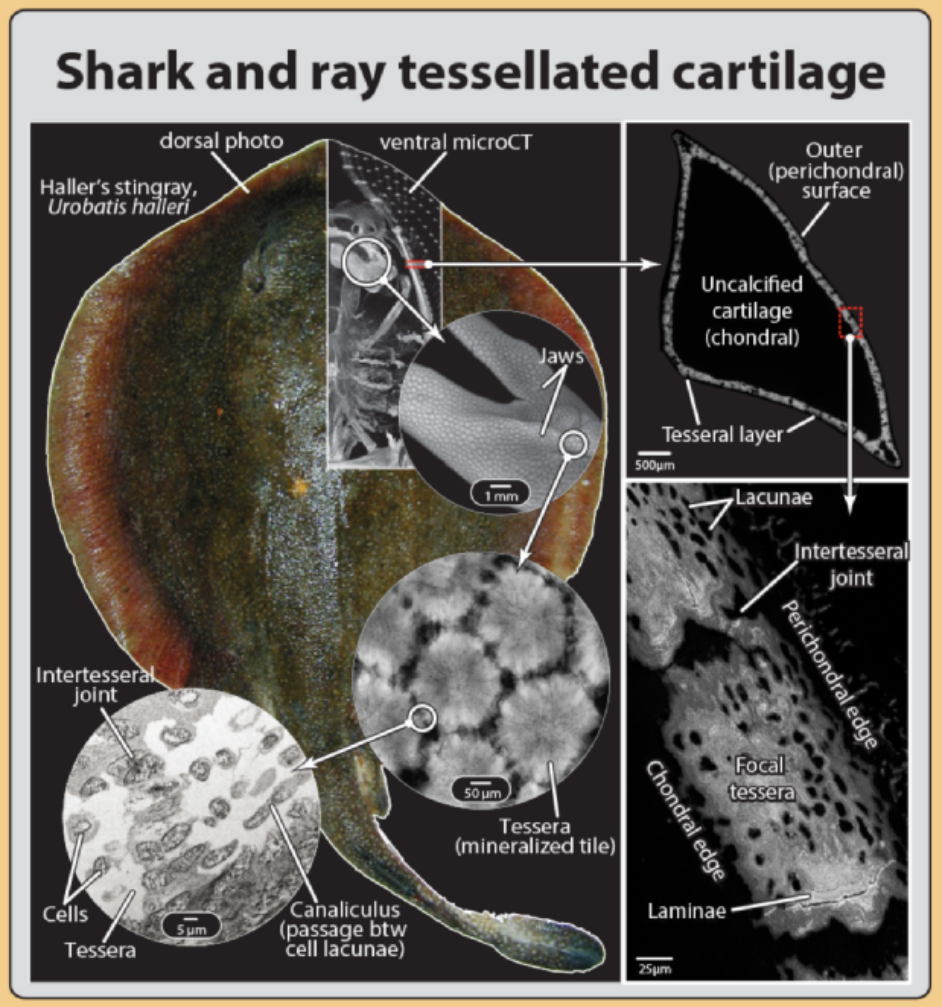


Figure 1.2: Schematic representation of the tesseral structure of a ray: (Left) tesseral structure on multiple resolutions; (Right top) the tesseral layer encloses uncalcified cartilage; (Right bottom) a thin tesseral joint connects two tesserae. The picture is a modification of a picture for a paper in review by Mason Dean, our collaboration partner at the Max Planck Institute of Colloids and Interfaces.

1.2 Contributions

During this thesis, I present a newly developed segmentation pipeline consisting of an automatic initial segmentation step that exploits the geometrical shape of the tesserae and uses the contour tree data structure. The contour tree tracks the evolution of level sets in a dataset during isovalue changes. The initial segmentation is followed by a manual correction step. The algorithms were implemented into the visualization software ZIBAmira as new or extended computation modules.

Furthermore, I developed a graph based abstraction of the segmentation to support further analysis and to enhance user interaction.

The contour tree based segmentation algorithm is sped up by a new approach that I call fast contour tree segmentation.

1.3 Overview

This thesis is structured in the following way. The first chapters describe concepts and algorithms that are later used to solve the segmentation task. This includes Chapter 2 which summarizes basic knowledge about segmentation algorithms and volume rendering. Chapter 3 introduces the contour tree data structure and explains the usage of the contour tree for segmentation purposes. Chapter 4 presents the visualization software ZIBAmira which is used throughout this thesis.

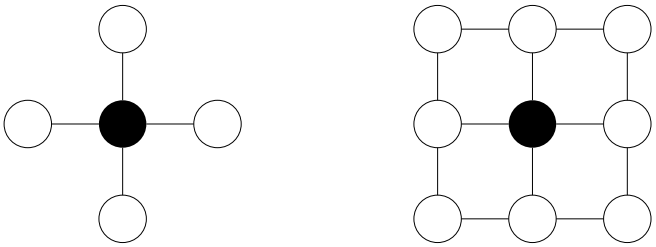
The contour tree segmentation is part of the segmentation pipeline presented in Chapter 5. The quality of these segmentation results is discussed in Chapter 6. Chapter 7 describes how the segmentation results can be used for further analysis of tesseral structures. The final chapters give an outlook to possible future work and summarize the presented work.

Chapter 2

Basics

The grayscale images in this thesis are functions defined on two or three-dimensional meshes. A two-dimensional image I is defined as an $m \times n$ matrix of natural or real numbers. An individual element of this matrix is called pixel. The elements of the three-dimensional equivalent are called voxels.

Algorithms working on neighbors of pixels or voxels are affected by the choice of the neighborhood relation. For example, if a local maximum is defined as a voxel with higher value than any of its neighbors, then there are more local maxima if a voxel has fewer neighbors. The best choice of the neighborhood relation is often not obvious. Figure 2.1 shows two basic neighborhood definitions in two-dimensional space with 4 or 8 neighbors. The equivalent definitions in three dimensions have 6, 18 or 26 neighbors.



(a) (b)

Figure 2.1: Neighborhood relations in 2D: (a) shows 4 neighbors of the central black voxel; (b) shows 8 neighbors.

2.1 Segmentation

Image segmentation is a huge field with a vast number of concepts and algorithms created over the last decades. This chapter gives an overview of segmentation basics. The algorithms that are used for the segmentation of tesseral structures are presented more accurately during the next sections. The usage of the contour tree for segmentation purposes is described in Section 3.3. This chapter focuses on the segmentation of three-dimensional images, hence the following text deals with voxels instead of pixels.

Segmentation algorithms partition the input image into multiple segments. The segmentation assigns the same natural number, called label, to all voxels of one segment. Voxels with the same label share certain characteristics, one possibility is for example that they have similar grayscale values. The input datasets can vary from simple two-dimensional pictures to complex datasets from medical or biological applications. The result of the segmentation process is a dataset consisting of labels, often called label field. An oversegmentation refers to a segmentation result with too many labels.

There are multiple possibilities to classify segmentation algorithms, for example whether the algorithm just separates the foreground from the background, whether the algorithm is fully-automatic or needs additional user interaction, whether the algorithm works on voxels, edges or treats the image like a graph, and so forth. For more information see the review papers from Wirjadi [2] and Pal et al. [3]. The following paragraphs summarize some of the most common segmentation approaches.

Thresholding might be the easiest of the segmentation algorithms but it is sufficient for a lot of applications where the grayscale values distinguish the regions of interest. In a binary thresholding, all voxels smaller than the threshold value t get assigned label 0, and the remaining voxels get assigned label 1. A separation in more than two segments is possible with multiple threshold values. There are several variations of thresholding algorithms, for example local thresholding [2] with different threshold values for different parts of the image.

Edge-based segmentation algorithms are suitable if the segments are separated by edges. Then the segmentation can be done by identifying these edges with edge detection algorithms. An example for this kind of segmentation algorithm is intelligent scissors [4].

Region growing algorithms start with seed regions. Neighbor voxels are iteratively added to the growing region if they fulfill some region homogeneity criterion. Examples are the magic wand, usable in image processing softwares like GIMP, or the grow cut algorithm [5].

Graph-based segmentation algorithms treat the input image as a graph and solve the segmentation task by performing operations on that graph. In most cases, there is one vertex for each voxel and there are edges between vertices if they share common properties, for example if they are neighbored due to a useful neighborhood criterion. Often edge weights are included to measure the similarity between two vertices. Example algorithms are graph cut [6], normalized cuts [7], random walker [8] and spectral clustering [9]. The last two algorithms are presented more accurately in the following sections.

The idea of watershed segmentation [10] is to interpret the grayscale values as altitude information. Then the dataset is flooded with water. The water basins start to grow at each local minimum and build barriers where different basins meet. The segments of the segmentation are the individual water basins that have been built after the whole dataset has been flooded. This segmentation approach usually leads to an oversegmentation, so there exist different versions of hierarchical watershed algorithms.

Surface or shape-based segmentation algorithms solve the segmentation problem by identifying specific objects in the image. This is for example possible with the creation of an active shape model [11] of the searched object.

Energy-based segmentation algorithms create a function defining a segmentation and try to maximize an energy functional. Examples are the Mumford-Shah [12] functional or the Chan-Vese [13] functional.

It is important to notice that there is not one perfect segmentation algorithm which outclasses all others. One needs to understand a given segmentation problem and has to choose the algorithms that are most appropriate for this specific problem. Also the importance of properties like the running time differs between applications.

2.1.1 Spectral clustering

Spectral clustering refers to a family of clustering algorithms that use the spectrum of matrices belonging to the input data. See the paper of von Luxburg [9] for a comprehensive introduction to spectral clustering.

First, the problem to segment the image into k segments is reformulated as a graph clustering problem. The vertices of the similarity graph $G = (V, E)$ correspond to voxels in the image, the weight $w_{i,j}$ of the edge $(i, j) \in E$ measures the similarity between the vertices i and j. Now, the solution of a clustering of G into k clusters refers to a segmentation of the input image.

Assume that a similarity graph $G = (V, E)$ is given.

Definition 1 (Laplacian matrix)**.**
Let $G = (V, E)$ be a weighted undirected graph with vertices V, edges $E \subseteq$

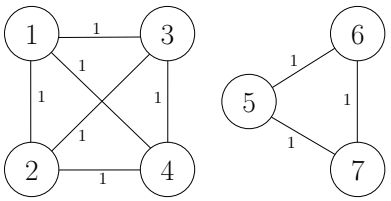
$V \times V$, $|V| = n$ and edge weights $w_{i,j}$. Let $D = (d_{i,j})$ be the degree matrix with $d_{i,i} = \sum_{j=0}^{n} w_{i,j}$ and $d_{i,j} = 0$ for $i \neq j$, and let A be the adjacency matrix using the edge weights. Then the Laplacian matrix $L = (l_{i,j})$ is defined as

$$L = D - A$$

It follows that $l_{i,j} = \begin{cases} d_{i,j} & i = j \\ -w_{i,j} & i \neq j \end{cases}$.

L is symmetric, positive semi-definite and has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. The constant vector $\mathbf{1}$ is an eigenvector with eigenvalue 0. The proof is shown in [9].

Now, the eigenvalues and eigenvectors of L are used to perform the graph clustering. The following example demonstrates the functionality.



(a)

$$L = \begin{pmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

(b)

Figure 2.2: (a) similarity graph with seven vertices and two connected components; (b) corresponding Laplacian matrix.

Figure 2.2 shows a simple example of a similarity graph with edge weights 1 and the corresponding Laplacian matrix. The graph consists of two connected components $C_1, C_2 \subseteq V$. It is easy to see that the indicator vectors $\mathbf{1}_{C_1}$ and $\mathbf{1}_{C_2}$ are eigenvectors with eigenvalue 0. Furthermore, for a similarity graph with m connected components $C_1, C_2, \ldots, C_m$, the eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbf{1}_{C_1}, \mathbf{1}_{C_2}, \ldots, \mathbf{1}_{C_m}$. The proof can be found in [9]. So the eigenvectors with eigenvalue 0 relate to the connected components in the similarity graph.

Now assume that there is an edge between vertex 4 and vertex 5. Then only the constant vector $\mathbf{1}$ spans the eigenspace with eigenvalue 0 but the eigenvector with the second-smallest eigenvalue enables the separation between the first four vertices and the last three vertices, see the first two eigenvectors from a Matlab calculation in the following table.

$$\text{Eigenvector 1} = \begin{bmatrix} -0.37796 \\ -0.37796 \\ -0.37796 \\ -0.37796 \\ -0.37796 \\ -0.37796 \\ -0.37796 \end{bmatrix}, \text{Eigenvector 2} = \begin{bmatrix} 0.35604 \\ 0.35604 \\ 0.35604 \\ 0.21422 \\ -0.29656 \\ -0.49289 \\ -0.49289 \end{bmatrix}$$

So spectral clustering calculates the first k eigenvectors, the i-th row belongs to the i-th vertex of the similarity graph, and uses each row as a vector for a k-means clustering into k clusters on n datapoints with dimension k. The result is the final clustering of the similarity graph. The following pseudo code summarizes the algorithm.

Algorithm 1 Spectral clustering pseudo code for a clustering in k clusters

1: Create the similarity graph
2: Compute the Laplacian matrix L
3: Compute the first k eigenvectors
4: Do k-means clustering on the rows of the eigenvectors

A similarity graph with n vertices leads to an $n \times n$ Laplacian matrix. So it is necessary to compute the k smallest eigenvalues and the corresponding eigenvectors of an $n \times n$ matrix. The k-means algorithm clusters n datapoints in k-dimensional space into k clusters.

2.1.2 Random walker segmentation

Again, the input image is treated as a weighted graph with a vertex for each voxel and edges between two vertices if their corresponding voxels are neighbored. Random walker segmentation [8] requires starting seeds as additional input. If there are k different seeds, the dataset will be segmented into k segments.

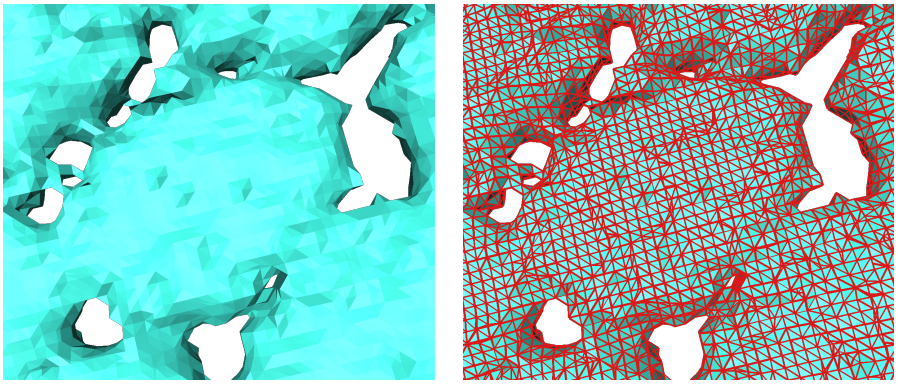
Imagine that each unlabeled voxel releases a random walker. The algorithm computes the probability that a random walker reaches a specific seed first. For each voxel, k probabilities are computed, the probability that the random walker reaches the first seed first, the probability that the random walker reaches the second seed first and so forth. Finally, the voxel is assigned to the seed with the highest probability. The random walk is done on the weighted graph, so the probability to reach the neighbored voxel j from

a voxel i is determined by the edge weight. The most common edge weight is the Gaussian weighting function $w_{i,j} = exp(-\beta(g_i - g_j)^2)$ where g_i is the image intensity value at voxel i and β is a free parameter.

For each seed, it is necessary to solve a system of linear equations, so in total $k-1$ linear systems have to be solved. The last linear system is unnecessary because the sum of all probabilities at one vertex equals 1. The size of the linear systems is $u \times u$, where u is the number of unseeded vertices. Computational details can be found in the paper of Grady [8].

2.2 Surface

In the context of this work, a surface is a discrete mesh embedded in three-dimensional space consisting of points and edges which create a triangular structure. Triangular surface meshes can be created in several ways, for example by scanning the surface of a real object with a laser. A common approach is the creation of a surface belonging to a segmentation. The creation of surfaces resulting from label fields with more than two labels can be done using the generalized marching cubes (GMC) algorithm [14]. Thereby, the created mesh triangles are placed where different labels of the segmentation meet each other. Figure 2.3 shows the surface and the corresponding mesh of a tessera, and its surrounding neighbors.



(a) (b)

Figure 2.3: Surface mesh example: the surface is generated from a foreground-background segmentation. The surface triangles in (b) separate the foreground from the background.

2.3 Volume rendering

Volume rendering is a standard technique to visualize volumetric scalar data defined on a three-dimensional mesh. Volume rendering using ray casting shoots rays through all pixels of the image plane and follows them on their way through the data field. The individual rays are sampled and the scalar values at the sample points are accumulated to the final pixel value on the two-dimensional image plane using the volume rendering equation. The volume rendering equation describes the interaction between the ray and the volume and deals with emission, absorption and scattering of light rays. The color of an image voxel depends on the choice of the transfer function.

The visualization algorithm used for segmentation results needs to be fast. Computation and visualization of the corresponding surface does not reach the required running time. The solution is to use volume rendering on the label field of a segmentation. The transfer function assigns a random color to each label. The evaluation of the sample points uses an interpolation algorithm. Linear interpolation is not useful for label fields because the field only contains natural numbers, linear interpolation between two labels reduces the visualization quality. Instead nearest neighbor interpolation should be used for volume rendering of label fields.

Chapter 3

Contour tree

The contour tree is a data structure that tracks certain changes of level sets during isovalue changes. The contour tree was introduced by Boyell and Ruston [15] for a two-dimensional application. This section follows the paper and definitions by Carr et al. [16].

3.1 Definition

Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^d$ with different scalar values $\{h_1, h_2, \ldots, h_n\}$, the considered mesh M is a simplicial mesh with vertex set P. Simplicial meshes consist of simplices, so M is a triangular mesh in two-dimensional space and a tetrahedral mesh in three-dimensional space. Now, f is a function on M with $f(p_i) = h_i$ and f is linearly interpolated inside the simplices.

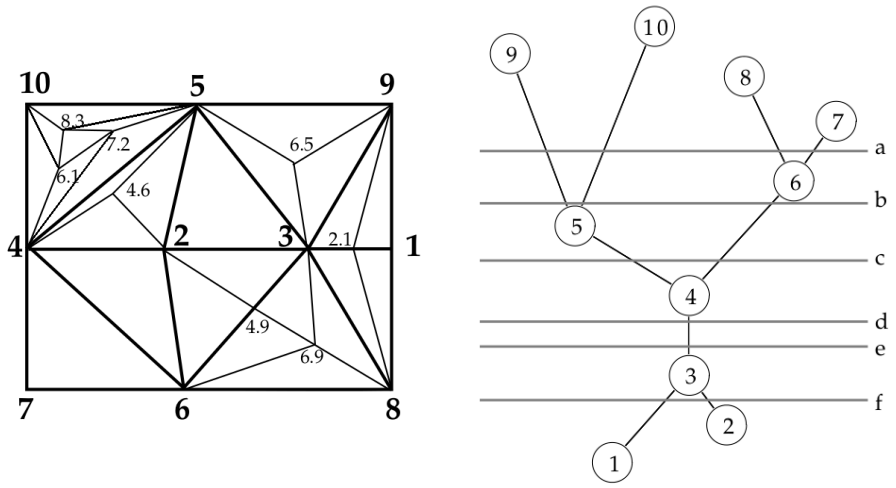
Definition 2 (Level set)**.**
The level set of f for isovalue h is defined as $\{x \in \mathbb{R}^d | f(x) = h\}$. In two dimensions, the level set is called isoline, in three dimensions isosurface. One level set can consist of multiple connected components, in the following called contours.

The contour tree tracks the evolution of these contours of f.

Definition 3 (Critical point)**.**
Critical points are points at which the topology of level sets change.

Definition 4 (Morse function)**.**
Critical points are isolated if they occur at distinct points and values. If all critical points of a function are isolated, the function is a Morse function.

The function f defined on M is a Morse function and all critical points of f are at mesh vertices. See [16] for the proof. The contour tree nodes are



(a) (b)

Figure 3.1: (a) simplicial mesh; (b) contour tree corresponding to the simplicial mesh. The pictures are taken from the paper of Carr et al. [16].

a subset of the critical points. The contour tree consists of local maxima, local minima and nodes where the number of contours change. One edge corresponds to one contour and tracks the evolution of this contour while the isovalue is changing. The augmented contour tree shown in Figure 3.2a contains nodes for all mesh vertices, not only for critical points.

Figure 3.1 shows a function defined on a two-dimensional simplicial mesh and the corresponding contour tree. Nodes 9, 10, 8 and 7 are local maxima, nodes 1 and 2 are local minima and nodes 3, 4, 5 and 6 are nodes where the number of contours change. Node 3 is called split node, nodes 4, 5 and 6 are called join nodes. The number of intersection points between the contour tree and a horizontal line created for an isovalue h shows the number of contours in the level set of isovalue h. The level set of the isovalue belonging to line a in Figure 3.1 consists of 4 contours. One encloses the local maximum 9, the other ones enclose 10, 8 and 7. The contours starting at 8 and 7 grow with decreasing isovalue and meet at isovalue 6. There, the contours merge into one contour and the number of contours at isovalue b is decreases to 3.

3.2 Computation

Let N be the number of simplices in M and n be the number of mesh vertices. Van Kreveld et al. [17] developed an algorithm that created the contour tree in $O(N \log N)$ in two dimensions, and in $O(N^2)$ in higher dimensions. Tarasov and Vyalyi [18] presented an $O(N \log N)$ algorithm in three dimensions based on sweeps through the mesh.

The algorithm by Carr et al. [16] computes the contour tree in $O(n \log n + N\alpha(N))$, where α is the inverse Ackermann function, and uses three sweeps. First, the algorithm iterates over all mesh vertices in a descending manner and constructs the join tree consisting of all local maxima and all join nodes. The join tree belonging to the example in Figure 3.1 can be seen in Figure 3.2b. Note that the join tree contains a node for each mesh vertex, not only for critical points, but there exist no split nodes.

The following pseudocode describes the computation of the join tree. The algorithm iterates over the descending mesh vertices. For each mesh vertex, a join tree node is created (line 3). If there is exactly one neighboring contour, an edge between the lowest node of this contour and the current node is created (line 10). If there are multiple connecting contours, the current node gets connected to all these contours and becomes a join node. The tracking and merging of the contours is done using a union-find data structure, called Component in the pseudo code (initialization in line 2, merge in line 9).

Algorithm 2 Computation of join tree, input mesh vertices $p_1, \ldots, p_n$ are sorted: $h_1 < h_2 < \ldots < h_n$

```
 1: for i = n → 1 do
 2:    Component[i] = i
 3:    Create new node on position Position[i]
 4:    LowestVertex[i] = Position[i]
 5:    for each neighbor p_j of p_i do
 6:       if j<i or Component[i] = Component[j] then
 7:          continue
 8:       end if
 9:       Merge(Component[i], Component[j])
10:       AddEdgeToJoinTree(Position[i], LowestVertex[Component[j]]
11:       LowestVertex[Component[j]] = Position[i]
12:    end for
13: end for
```

The split tree is calculated in the same way as the join tree, just in reverse order from the lowest to the highest value, and contains all local minima and split nodes. Split and join trees are shown in Figure 3.2.

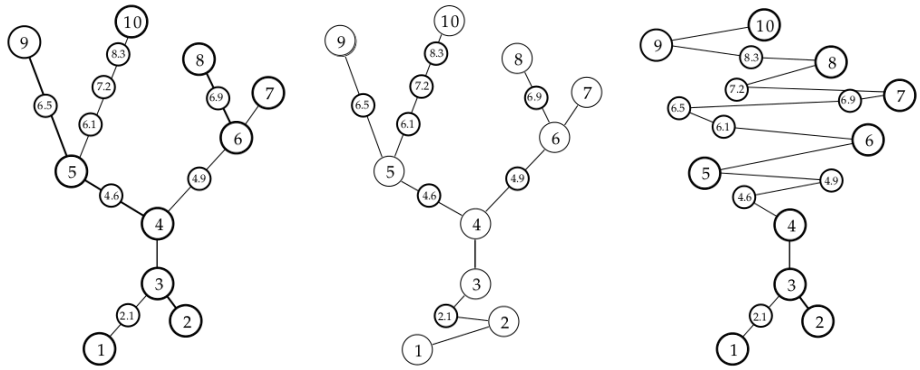


Figure 3.2: (a) augmented contour tree containing all mesh vertices; (b) join tree; (c) split tree. The pictures are taken from the paper of Carr et al. [16].

The augmented contour tree is a combination of join and split tree. This combination algorithm starts with an empty contour tree. It iteratively adds nodes to the contour tree and removes them from the join and split trees, until both the join and split trees are empty. A node can be added if it has no parent (edge to a node with higher value) in the join tree and one child (edge to a node with lower value) in the split tree, or it has no child in split tree and one parent in the join tree.

Detailed descriptions of the whole algorithm, including proofs and a series of example images for the combination algorithm, can be found in [16].

3.3 Contour tree segmentation

The most common applications of contour trees are fast isosurface computation [19] and transfer function selection for volume rendering [20]. Marching cubes is the standard algorithm for isosurface computation and has to iterate over all voxels to find starting seeds of the isosurface. Then the contour is followed until the starting seed is reached again. The contour tree contains information about the number of connected components belonging to an isovalue and enables immediate seed generation. Regarding volume rendering, the contour tree enables the application of separate transfer functions to individual connected components. The contour tree can also be used to perform image segmentation. In the following, three different segmentation algorithms based on the contour tree and on the join tree are presented.

The easiest approach is to assign one segment to each edge of the contour tree. Imagine a simple two-dimensional relief with two mountains and a

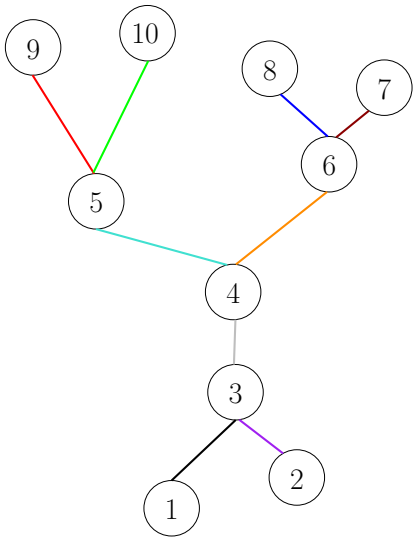


Figure 3.3: First segmentation approach: contour tree with edges that are colored according to their segment.

mountain pass. Then the algorithm creates two segments for the tops of the mountains. The segments start at the local maxima and spread out until they reach the mountain pass. At the mountain pass a third segment starts. Figure 3.3 shows a contour tree with edges colored according to the created segment. In total, 9 different segments are created.

The second approach uses the join tree and is based on the idea that each node v in the join tree stands for all data points $D(v)$ lying on an edge above v. So $D(v)$ contains all mesh points that can be reached by following the contour corresponding to v with increasing isovalue up to the reachable local maxima. For example, a local maximum just stands for itself, there are no edges above. Figure 3.4a shows a node where the described edges and nodes are colored in red. The creation of one segment corresponds to the choice of one node v, then the segment contains all voxels in $D(v)$. The choice of multiple nodes creates multiple segments. Assume $v_1, v_2, \ldots$ are the selected vertices. To achieve a unique segmentation, it must hold that the sets $D(v_1), D(v_2), \ldots$ have no common elements. Figure 3.4b shows the choice of a valid segmentation. Note that there are data points which remain unlabeled. For example, the selection of a local minimum would not allow the selection of another node, so voxels with low values will most probably stay unlabeled. This segmentation approach was implemented for a diploma thesis [21].

The third approach uses the join tree and starts with one segment for each local maximum. At join nodes, multiple segments meet. Now, segments meeting at a join node can either be merged into one segment or the seg-

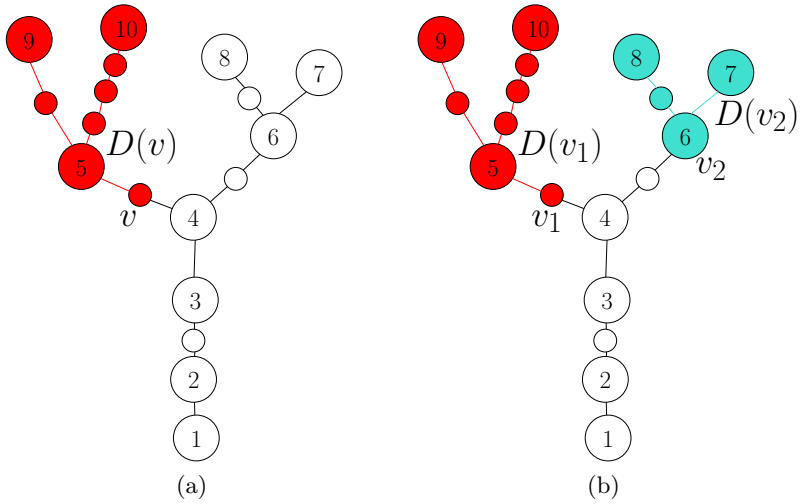


Figure 3.4: Second segmentation approach: (a) red regions belong to v; (b) segmentation into two segments with two selected vertices v_1 and v_2.

ments can stay separated depending on a persistence criterion introduced by Edelsbrunner et al. [22]. In the context of this algorithm and in the context of the following parts of the thesis, these join nodes are called merge nodes or split nodes. See Figure 3.5 for an example. Throughout this thesis, the term contour tree segmentation refers to this third segmentation approach using a persistence criterion.

To be more precise, the segmentation algorithm iterates over the data points in the order of decreasing scalar values. If there is a new local maximum, a new segment containing this voxel is created. Each voxel that is neighbored to exactly one already labeled segment gets the same label as its neighbor. If a new voxel is neighbored to more than one segment, the segments merge if the difference between one of the maxima and the value at the current voxel is less than a user-defined persistence value (merge node). Otherwise the segments stay separated (split node). The final segmentation is defined by the values of the union-find data structure, again called component in the pseudo code in algorithm 3. If the persistence value is so small that no merges occur, the number of segments in the final segmentation equals the number of local maxima. On the other hand, if the persistence value is so large that merges occur at each join node, all segments merge into one segment.

Assume that a new voxel v has two segments as neighbors but the segments cannot be merged because the persistence value is too small. Now the segments stay separated but v has to be assigned to one of these adjacent segments. Two possibilities are to either assign v to the segment of the

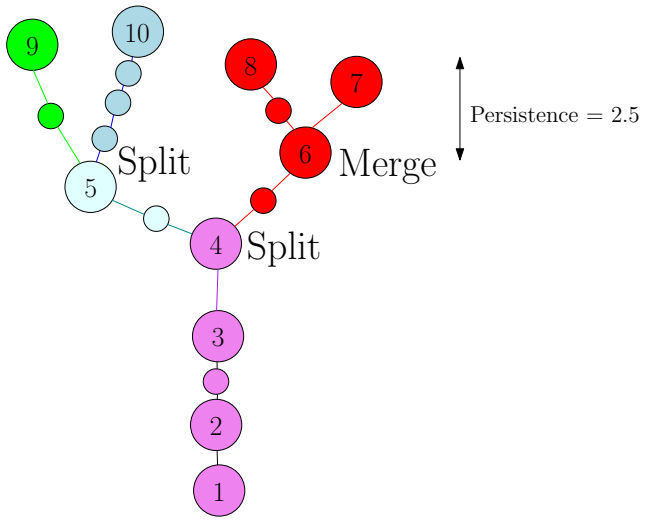


Figure 3.5: Third segmentation approach: The segments/contours starting at 7 and 8 meet each other at 6. $8-6=2$ and $7-6=1$ is smaller than the persistence value 2.5, hence the segments get merged. Nodes 5 and 4 are split nodes because the difference to the maxima values is larger than 2.5. The node between 5 and 4 can belong to the blue or to the green segment. If blue is added to green one gets cyan. This is the reason why the node between 5 and 4 has the color light-cyan.

Algorithm 3 Contour tree segmentation, input mesh vertices $p_1, \ldots, p_n$ are sorted: $h_1 < h_2 < \ldots < h_n$

```
for i = n → 1 do
    Component[i] = i
    MaxValueOfComponent[i] = h_i
    for each neighbor p_j of p_i do
        if j<i or Component[i] = Component[j] then
            continue
        end if
        if Persistence criterion then
            Merge(Component[i], Component[j])
            Update MaxValueOfComponent[i] or MaxValueOfComponent[j]
        end if
    end for
end for
```

neighbor with the largest value or to the segment where most of its neighbors belong to.

As long as there are no split nodes (all segments get merged), the voxels with multiple neighbor segments correspond exactly to the join nodes. Imagine a node where segments S_1 and S_2 meet and stay separated. Then it is possible that there are following voxels neighbored to S_1 and S_2 and the persistence value decides again whether the segments should be merged or not. So the decision between merge or split was made for a voxel not corresponding to a join node. But as shown later, the segments meeting at such nodes will never merge.

Contour tree segmentation is useful if voxels between the wanted segments have lower values than voxels inside the segments. Then it can be possible to find a persistence value that creates split nodes between the segments. So hopefully, different maxima inside the segments get merged but the segments are split at the borders.

The contour tree segmentation has similarities with watershed segmentation. Watershed starts from the local minima and floods the water basins while the contour tree segmentation starts from the maxima. Both algorithms have to deal with the problem of oversegmentation. For watershed exist hierarchical algorithms, the contour tree segmentation handles oversegmention with the persistence criterion.

3.3.1 Persistence criteria

The described persistence criterion is not the only possibility. Another example is a persistence criterion that includes the size of the segments. So very small segments with high values can be merged with their neighbor segment.

The one-dimensional example in Figure 3.6 shows a very thin segment with high maximum value S_1 and a large segment S_2. Using the first persistence criterion, the segments stay separated because the difference between the maxima and the value at the current voxel is not less than the persistence threshold. A persistence criterion including the size could identify S_1 as noise and include it into S_2. The following parts of the thesis use the standard persistence criterion without regarding the segment size.

3.3.2 Fast contour tree segmentation

The contour tree segmentation algorithm has to sort the whole dataset and has to iterate over all voxels in order to merge segments according to the persistence criterion. The idea of our fast contour tree segmentation algo-

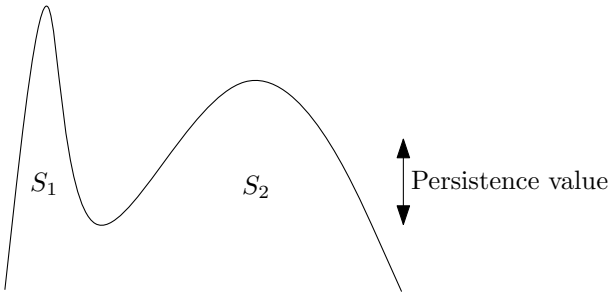


Figure 3.6: Persistence criterion example: A persistence criterion depending on the size of segments can merge S_1 into S_2.

rithm is to iterate only over the nodes of the join tree. The join tree is calculated in one initial computation and afterwards it is only necessary to iterate over the join nodes for different persistence values. In general, there are significantly less join tree nodes compared to the number of voxels in total, so the running time decreases dramatically.

The idea is to precompute the join tree and the oversegmentation where all segments stay separated. The oversegmentation is given as the corresponding union-find data structure. Additionally, the maximum values of all components in the union-find data structure are saved. Now, for a specific persistence value, the algorithm only iterates through the join nodes of the join tree and tests whether it is a merge node or a split node. If it is a merge node, the components of the union-find data structure get merged. The final segmentation is again defined by the union-find result. The pseudo code in algorithm 4 illustrates the functionality of the algorithm.

Algorithm 4 Fast contour tree segmentation, inputs are join tree: J, union-find of oversegmentation: Component, maximum values of components: MaxValueOfComponent

```
1: for each join tree node in order of decreasing value do
2:    p_i is mesh vertex corresponding to join tree node
3:    for each neighbor p_j of p_i do
4:       if j<i or Component[i] = Component[j] then
5:          continue
6:       end if
7:       if Persistence criterion then
8:          Merge(Component[i], Component[j])
9:          Update MaxValueOfComponent[i] or MaxValueOfComponent[j]
10:      end if
11:   end for
12: end for
```

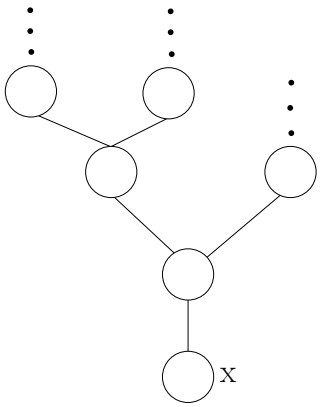


Figure 3.7: Proof sketch.

The fast contour tree segmentation result equals the standard contour tree segmentation result under certain conditions, which are described in the remainder of this chapter. But first, it must be shown that all merges during the application of the standard contour tree segmentation algorithm happen on a join tree node and not somewhere in between. In the following the segments are called components because of the used union-find data structure.

Theorem 1. *During the application of the standard contour tree segmentation algorithm, merges only occur on join nodes.*

Proof. Observation 1: If two segments (components) meet and get not merged, they will never merge again. This is because a later voxel, where both segments meet again, has an even lower value than the one before. So the difference between maxima and voxel value increases and cannot get lower than the persistence value.

Observation 2: If a component C meets another component and is not merged, it can only be merged later to a component that fulfills the persistence criterion. The difference between all following voxel values and the maximum of C is larger than the persistence value.

Now assume that two components C_1, C_2 merge in x but x is not a join node and has just one parent (see Figure 3.7) . This is only possible, if C_1 and C_2 were both parts of at least one prior split node. Hence for C_1 and C_2 holds observation 2 and they cannot be merged. This contradicts the assumption.

□

Now imagine that the fast contour tree segmentation algorithm reaches a join node where the components C_1 and C_2 merge to C. After the merge,

voxels that belonged to C_1 or C_2 are now in C, what is exactly the same result as in the standard algorithm. The question is if there are voxels with lower value than the join node which are not in C_1 or C_2 but would have been in C using the standard algorithm. This is possible what is illustrated with the following example.

Assume a voxel v has C_1, C_2 and S as neighbor components and these three components cannot merge at v. Assume that v shares more neighbors with S than with C_1 or C_2 but C shares more neighbors with v than S. Then v belongs to S in the oversegmentation where all contours stay separated and hence v belongs not to C after the merge of C_1 and C_2. But this is a mistake because in the standard algorithm C_1 and C_2 were already merged before v, so v gets assigned to C.

The problem is that the component choice for v depends on the number of neighbors which is affected by merges. So the result of the component choice depends on previous merges. This is not the case if the choice depends on the largest neighbor. The neighbor with the largest value does not change during merges. So the fast contour tree segmentation equals the standard contour tree segmentation if this choice depends on value size instead of neighborhood.

Chapter 4

Amira

Amira, Avizo and ZIBAmira [23] are data visualization, processing and analysis softwares developed by FEI Visualization Sciences Group in Bordeaux and the Zuse Institute Berlin (ZIB). Amira and Avizo are commercially distributed by FEI Visualization Sciences Group where Amira focuses on life sciences and Avizo focuses on material sciences. ZIBAmira is developed at the Zuse Institute Berlin for research partners and contains, for example, also experimental or special purpose modules.

Amira is able to handle a broad range of input file formats such as TIFF stacks or DICOM data. Once loaded, the data can be stored using the native file format AmiraMesh. The user can visualize or process scalar fields, vector fields, label fields, triangular surface meshes and many more kinds of data.

A high flexibility is obtained due to Amira's modular design. Loaded datasets occur in the object pool (part (a) in figure 4.1) and can be visualized with rendering modules or processed with computation modules. Multiple computation modules can be used to create a working pipeline to solve a specific task. The latter module gets the result of the previous module as input. Such computation modules include, for example, Gaussian smoothing, gradient computation of a scalar field, random walker segmentation or surface generation from a given segmentation. Three-dimensional visualization modules are, for example, modules for isosurface visualization, volume rendering or surface visualization. Figure 4.1 shows the Amira interface with two datasets and two modules in the object pool and a volume rendering as visualization example.

The segmentation editor of Amira allows manual and semi-automatic segmentation. The user can iterate through the slices and select the label of each voxel manually, or use algorithms like thresholding or region growing. The segmentation editor creates a label field as segmentation result. In ZIBAmira, such label fields behave like scalar fields optimized for segmen-

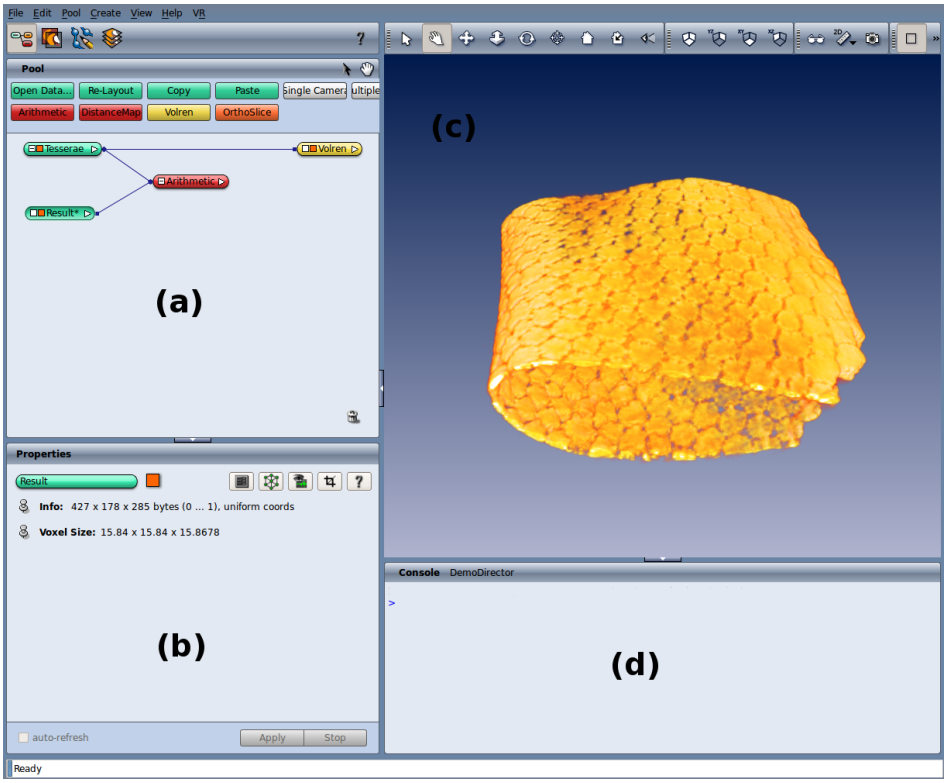


Figure 4.1: ZIBAmira user interface: (a) the object pool contains all datasets and modules (green: datasets, yellow: 3D visualization modules, red: computation modules, orange: 2D visualization modules); (b) the properties area shows properties of selected datasets and allows to adjust the settings of computation modules; (c) the 3D viewer shows visualization results, here a volume rendering of a tesseral structure; (d) the console window is used for text outputs and TCL commands.

tation results and can only store up to 256 labels. Scalar fields containing natural numbers could handle the results of larger segmentations but they cannot be processed by the segmentation editor.

The user can automatize tasks with the help of the script language TCL. TCL commands can be used in the console window (part (d) in figure 4.1) or in TCL script files. These script files can be written by every Amira user.

This thesis was created in the Visualization and Data Analysis department at the Zuse Institute Berlin, so all visualizations and computations in the next chapters were done with ZIBAmira. Hence it was possible to use all existing ZIBAmira functionalities and to write own modules in C++ which were added to the program. So there were three possibilities: either it was possible to simply use an existing ZIBAmira module, or an existing module was extended, or a completely new module was created.

Chapter 5

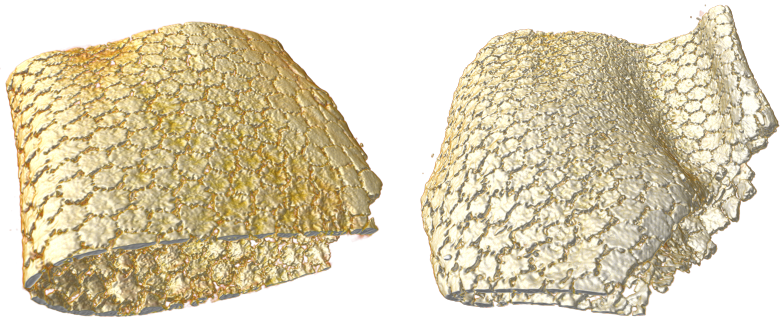
Segmentation of tesserae

This chapter describes how the algorithms of the previous chapters can be used to perform a segmentation of tesseral structures. The chapter contains the following parts. In the first part, the input micro-CT datasets are presented. Then, a segmentation of a dataset with few tesserae is calculated. The main part is the presentation of a segmentation pipeline to segment datasets consisting of hundreds of tesserae.

5.1 Introduction of datasets

The input datasets are micro-CT scans of tesserae. Figure 5.1 gives a first impression of the data and shows volume renderings of ray skeletons.

Humans can identify individual tesserae by just looking at the volume rendering. This is possible because of multiple reasons. First, the intensity values of tesserae are higher than the values of all other background struc-



(a) (b)

Figure 5.1: Volume renderings of two ray datasets with visible tesserae.

tures like uncalcified cartilage. Second, there are holes between the tesserae, that means, there are small areas not consisting of calcified cartilage. So the tesseral joints are small in comparison to the main parts of the tesserae. These observations can be seen in the slice in Figure 5.2b.

The Figures 5.2b and 5.2d focus on the intensity distribution over tesserae voxels. High values are mainly found in the boundary regions and on the tesseral joints. There are no borders between individual tesserae to use border detection algorithms. So the micro-CT values of the tesserae do not give enough information to perform the tesserae separation. Instead, it is the geometry of the tesserae with small tesseral joins that should be exploited by the following segmentation algorithms.

5.2 Segmentation of small areas

The segmentation of only a few tesserae, that means around ten to thirty instead of hundreds, is an easier task than the segmentation of a large dataset. It is possible to use segmentation algorithms that are too slow for the whole dataset but which perform well on a small amount of tesserae or voxels. It is also possible to use seed-based segmentation algorithms because the needed amount of user time and the possibility of user mistakes is lower.

As explained in the previous chapter, the geometry of the tesserae with small tesseral joints should be exploited for the segmentation. Because of these small joints, a random walker will most probably stay inside the tessera where he was released. Using a graph only consisting of vertices corresponding to tesserae voxels, spectral clustering could split the graph on the tesseral joints.

A random walker module already exists in ZIBAmira. It gets the micro-CT scan and user-defined seeds for each tessera plus one seed for the background as input and returns a segmentation of the whole dataset. The seeds can be constructed with the ZIBAmira segmentation editor. It is important that there is also a seed for the background label, the random walker module does not distinguish between foreground labels and the background label.

There is one linear system for each seed and each linear system has n variables, where n is the number of unlabeled voxels. These linear systems are solved by an iterative solver. Significant improvement of running time can be reached by maximizing the background seed, that means by performing a background segmentation first. The size of the linear systems depends on the number of unlabeled voxels, so the running time decreases if the seeds get larger.

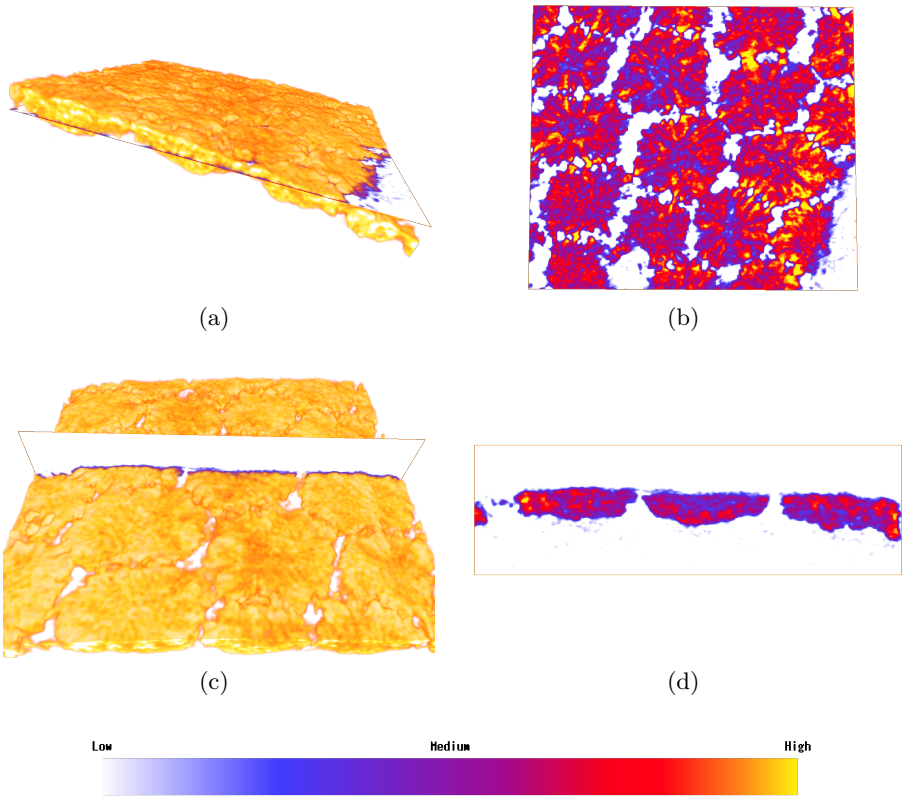
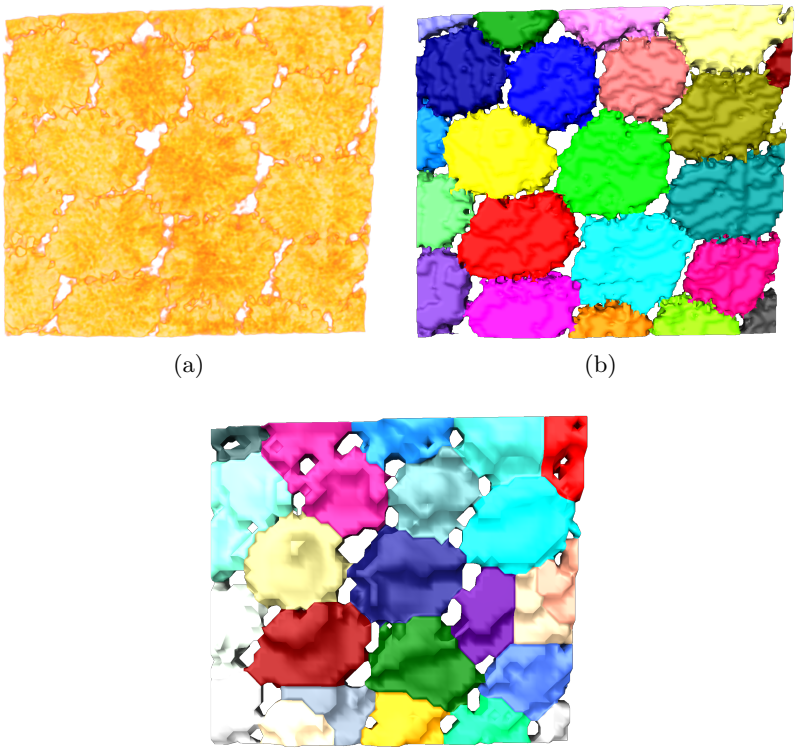


Figure 5.2: Two dimensional slices through the micro-CT dataset: (a) and (c) show the slice positions as white planes in the three-dimensional dataset; (b) and (d) show the slices; (e) colormap which is used for the slices, not for the volume renderings. These are just two-dimensional slices, so tesserae that are neighbored but not connected in these images are most likely connected in three dimensions.



(a) (b)

(c)

Figure 5.3: Segmentation results: (a) volume rendering of micro-CT scan; (b) segmentation result obtained by random walker segmentation; (c) segmentation result obtained by spectral clustering.

As expected, random walker segmentation leads to good segmentation results as shown in Figure 5.3b.

Spectral clustering is implemented as a new ZIBAmira module. It gets the foreground tesserae voxels and the wanted number of segments k as inputs. Then the Laplacian matrix of the similarity graph is constructed with one vertex for each foreground voxel and an edge between two vertices if they are neighbored according to the neighborhood relation with 26 neighbors. All edge weights are set to one.

The Laplacian matrix L has a size of $m \times m$ where m is number of foreground voxels. Matlab is used to do the eigenvector computation and to do the clustering with k-means as shown in the following Matlab code. Matlab can be directly included into ZIBAmira. Currently L is stored as a scalar field in the pool area of ZIBAmira and then loaded into the Matlab module. This is a very slow process and it does not allow the usage of sparse matrices.

So the dataset is resampled to a very low resolution and this might be the reason why the result in Figure 5.3c contains errors.

Algorithm 5 Spectral clustering matlab code, L is Laplace matrix and k the number of clusters

```
1: L = sparse(L);
2: [V,D] = eigs(L, k, 0);
3: resultClustering = kmeans(V, k);
```

5.3 Segmentation pipeline

The algorithms of the previous chapter are not capable to realize the segmentation of much larger datasets. For example, random walker segmentation of a dataset with 500 tesserae needs the user to set 500 seeds and afterwards the algorithm has to solve 500 large linear systems. So there is a need for a new segmentation algorithm.

The idea is to use a flexible pipeline consisting of exchangeable parts that starts with a fully-automatic initial segmentation followed by manual corrections of segmentation errors. Manual corrections are necessary because it is often not possible to get a perfect segmentation with only one automatic step.

Figure 5.4 shows a schematic representation of the suggested pipeline. First, the tesserae are separated from the background. Then, the main idea is to create a function with low values on the tesseral joints. This function enables a contour tree segmentation with split nodes on the tesseral joints to separate the tesserae. Such a function is created in step two by a distance map. In the end, this first automatic segmentation is improved by the user to get a final segmentation of the tesserae.

Detailed explanations of the individual steps follow in the next sections. Example images are taken from one dataset that is used throughout all pipeline steps.

5.3.1 Separation of tesserae and background

In the micro-CT scan, tesserae have a larger density than uncalcified cartilage or soft tissue. Hence it is possible to separate tesserae from the background with the help of simple thresholding. Figure 5.5 shows the surface belonging to the binary segmentation with threshold value 50.

Simple thresholding is prone to background noise with high values on small amounts of voxels which are not connected to the tesseral structure. Later, a

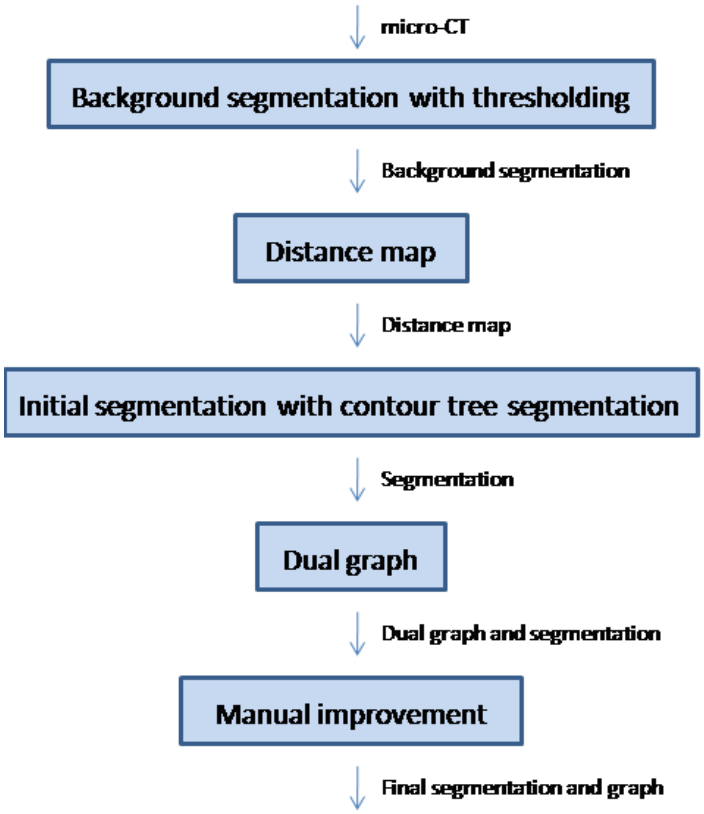
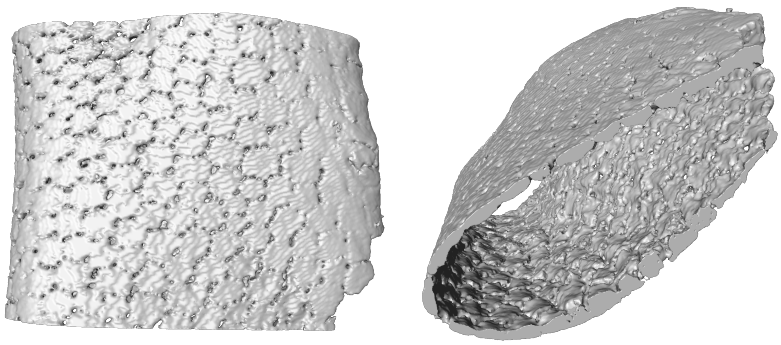


Figure 5.4: Segmentation pipeline.



(a) (b)

Figure 5.5: Background surface from two perspectives.

label is assigned to each of these voxels. It is possible to correct these errors after the initial segmentation by removing all tiny segments. Nevertheless, it is more sensible to exclude these areas already during the thresholding.

This can be achieved by using the magic wand tool of the segmentation editor. This tool performs a region growing, that is, the user selects a voxel and a gray value range, and the algorithm selects the largest connected area including the selected voxel and with values inside the given range. The result is a threshold segmentation where all selected voxels are connected.

Another problem is the possibility of changes in the gray value intensity over a large dataset. As a result, no suitable global threshold can be found, because the selection is too small in one area and in another area too large. This problem can be solved using local thresholding algorithms or prior smoothing of the dataset.

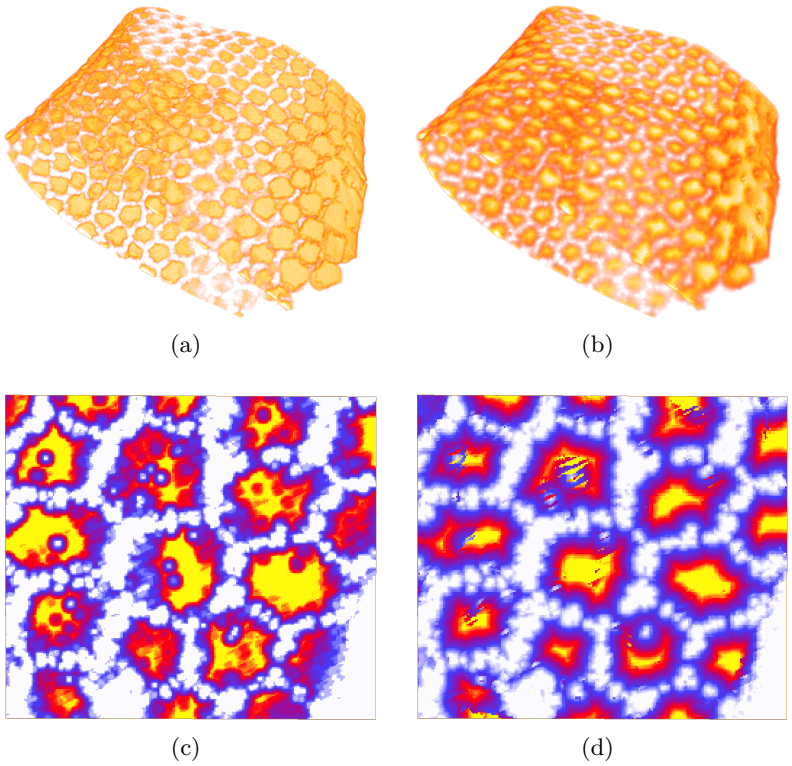
5.3.2 Distance map

Humans can identify individual tesserae because of regions consisting of uncalcified cartilage (holes) between the tesserae. So the geometry enables the separation and not the gray value structure. The idea of the distance map step is to create a function with high values at the tesserae centers and low values on the tesseral joints. Then, the topological properties of this function are exploited to do an initial segmentation with the help of a contour tree segmentation.

The described function is created by calculating the distance map, that means each foreground voxel gets assigned the shortest distance to the background. Because of the tesserae geometry, this distance is lower on the tesseral joints than in the centers. One tessera can contain multiple maxima.

The distance map in Figure 5.6a and in Figure 5.6c is computed using an existing ZIBAmira module that implements a three-dimensional distance map. The algorithm uses a region growing approach starting at the background voxels.

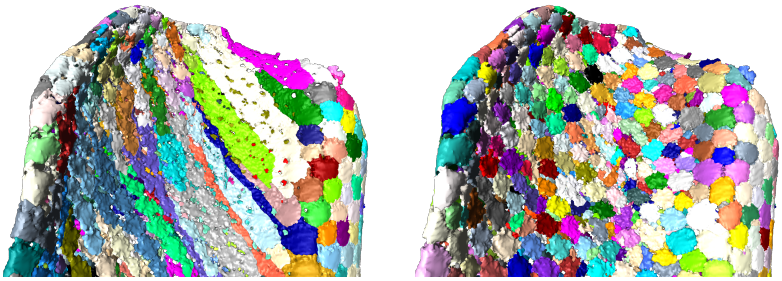
The three-dimensional distance map works for the example dataset used for the description of this pipeline. But the distance map calculation works in three dimensions, so there are voxels where the distance map value measures the distance to the top or bottom of the tessera because the height of tesserae is smaller than the diameter. This is a problem if the width of the tesseral joint is too large. In such cases, voxels on the tesseral joints and voxels inside the tesserae get a height value as distance map value. So the difference between center values and values on the joints is too low, and the contour tree based segmentation algorithm in Section 5.3.3 can not separate the



(a) (b)

(c) (d)

Figure 5.6: Volume rendering of distance maps: (a) 3D distance map; (b) 2D distance map; (c) close-up of 3D distance map; (d) close-up of 2D distance map. Compare the close-up images with Figure 5.2b.



(a) (b)

Figure 5.7: Failure of 3D distance map: (a) initial segmentation of 3D distance map with long segments that can not be separated; (b) initial segmentation of 2D distance map with separated tesserae. These are the only images in this section that are showing another dataset.

tesserae. Figure 5.7a shows how the initial segmentation fails and how long, not separatable segments are created.

Actually, the distance to the next hole or to the borders on the side of the tesserae is the important one, so our idea is to use two-dimensional distances instead of three-dimensional. Then, the distances are only calculated inside the plane orthogonal to the height dimension (see plane in Figure 5.8b).

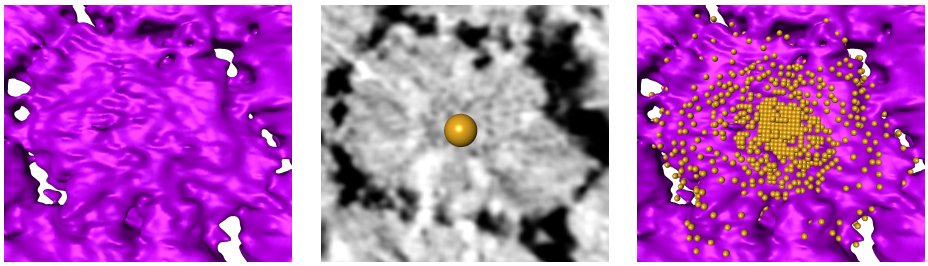
There are two steps for each voxel v: first the plane is calculated and then the shortest distance to the background inside this plane.

For the plane calculation, the first idea is to use a three-dimensional principle component analysis including all foreground voxels inside a cube around v. For plate-like structures as the tesserae, the first two eigenvectors of the larger eigenvalues are spanning the searched plane and the third eigenvector is orthogonal to the plane. To save the plane it is sufficient to store the orthogonal vector.

The second idea is to shoot rays from v in all directions (three-dimensional) and to compute the first intersection point of such a ray with the background. The intersection points with the top of the tesserae can be seen in Figure 5.8c. The searched plane is the plane with the minimal sum of squared distances to all intersection points.

Both algorithms are implemented in a new ZIBAmira module HxPlaneDistanceMap. Currently the second idea is used with 1214 rays because it is much faster than the variant using the principle component analysis and gives similar results.

Once the plane is found, a user defined number of rays are shot inside this plane and the distance along each ray to the first background voxel is com-



(a) (b) (c)

Figure 5.8: Two-dimensional distance map calculation: (a) surface of a tessera; (b) slice through the tessera with voxel v, the slice corresponds to the searched plane; (c) background intersection points on the top of the tessera after shooting 1214 rays.

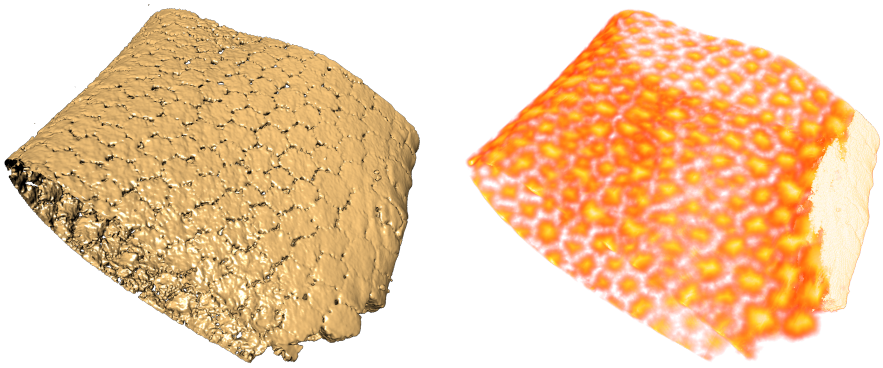
puted. The iteration over the voxels of a ray is done using the Bresenham algorithm [24]. The final distance is the smallest of the computed distance values. A two-dimensional distance map can be seen in Figure 5.6.

Two-dimensional distance map approaches solve the problem in Figure 5.7a, but there are problems with very large tesserae at regions with high curvature as the right highlighted region in Figure 5.9b. These tesserae do not have a plate-like form, so it is not clear how to place the plane.

So there are cases where the three-dimensional distance map fails and other cases where the two-dimensional version has problems. To solve this problem, the idea is to combine both versions for complicated datasets, that means both distance maps are computed and the three-dimensional values are used in areas where it was not possible to find a useful plane. These regions are identified with the help of the expected value and the variance of the squared distances between intersection points and calculated plane. The right highlighted area in Figure 5.9b has high variance values and is the area where it is not possible to find a useful plane approximation.

5.3.3 Initial segmentation

The automatic initial segmentation is done with the contour tree segmentation algorithm on the distance map. The theoretical background of the algorithm is described in Section 3. The standard contour tree segmentation algorithm already exists as a module in ZIBAmira. The module uses the standard persistence criterion without using the size of the segments and has now been extended with the fast contour tree segmentation algorithm described in Section 3.3.2. Figure 5.10 shows the final module interface. The



(a) (b)

Figure 5.9: Identification of regions where the plane calculation is not successful: (a) isosurface of whole dataset; (b) two-dimensional distance map with highlighted region where it is not possible to find a suitable plane for each point, highlighted region contains voxels with high variance values regarding the squared distances between intersection points and plane.

user can choose whether the fast segmentation algorithm should be used or not.

In addition to the description in Section 3, the module also uses a user-defined threshold value. All voxels with smaller values than the threshold are ignored during the segmentation and get automatically assigned to the background label. The input separation between tesserae and background is maintained if the threshold value is larger than zero and smaller than the smallest distance map value not equal to zero.

The maximum number of created segments equals the number of maxima in the distance field. The larger the persistence value the more merges will happen during the segmentation process, that means the number of segments decreases. The lowest possible number of segments equals the number of connected components over all voxels with larger value than the threshold value.

To obtain a good final segmentation, it is better to start with a slight over-segmentation because it is much easier to merge two segments in the manual correction step than to split a segment that is too large. Hence the persistence value should better be too small than too large.

Immediate visualization of segmentation results is required to quickly find a suitable persistence value. ZIBAmira already includes a volume rendering module for label fields but the module also needs segment colors as input. The module has been extended to generate random colors if no input colors

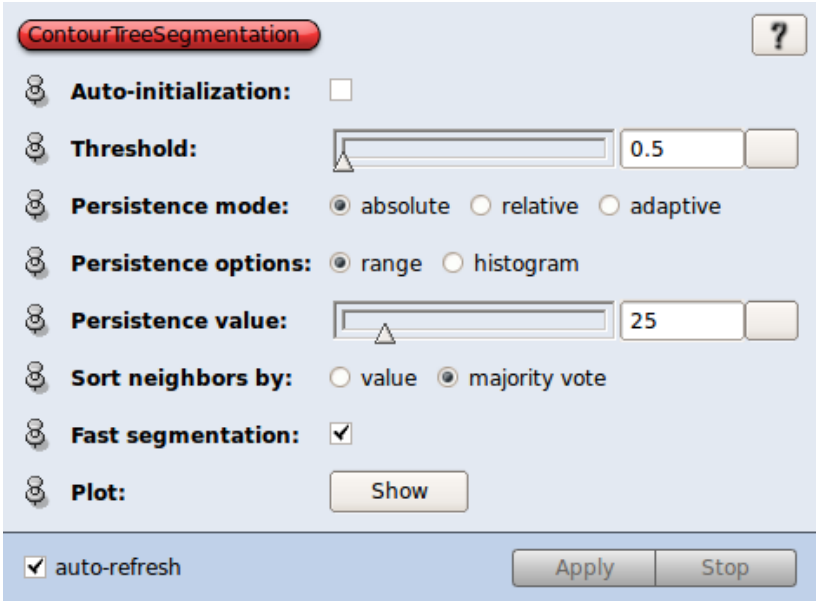


Figure 5.10: Interface of contour tree segmentation module.

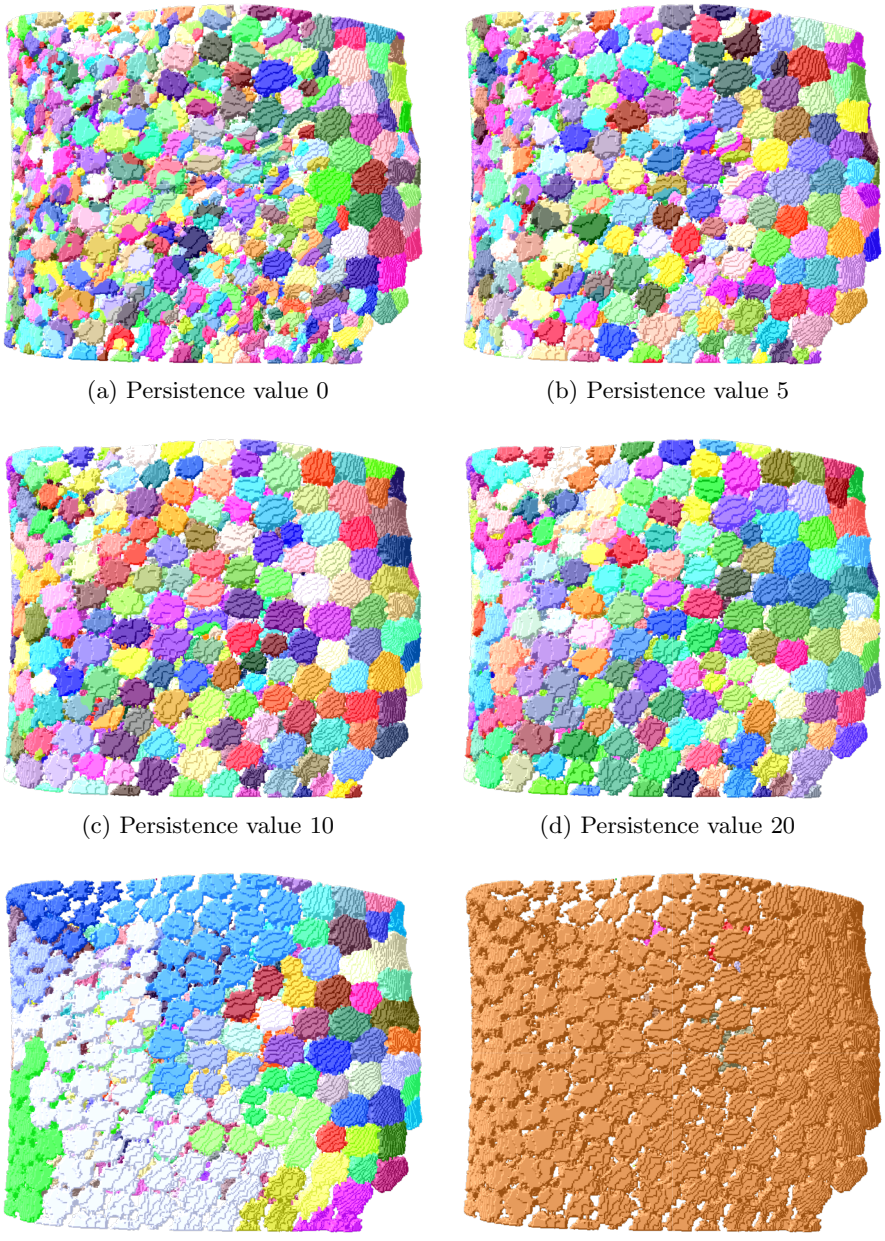
are available. The images in Figure 5.11 visualize segmentation results for increasing persistence values. All pictures are generated by volume rendering.

To summarize the workflow: the user simply connects the distance map to the contour tree segmentation module. He selects the wished threshold and persistence value and visualizes the segmentation result with volume rendering. For a fixed threshold value, the fast contour tree segmentation algorithm has to do the precalculations only for the first segmentation. For the following persistence values, the algorithm only iterates over join nodes of the join tree and enables the fast choice of a suitable persistence value.

5.3.4 Dual graph of segmentation

The dual graph of a given segmentation is a graph (V, E) embedded in $\mathbb{R}^3$, that means each vertex $v \in V$ has three-dimensional coordinates. There is one vertex for each segment and there is an edge between two vertices if the corresponding segments are neighbors. Figure 5.12 shows an example of the whole dataset.

The dual graph has two main advantages. First, the graph is an abstraction of the segmentation. Even without annotations it stores all information about the neighborhood relations. With annotations it can store properties on the edges or on the vertices like the size of regions and curvature information. More information about these properties are shown in Chapter 7.



(a) Persistence value 0

(b) Persistence value 5

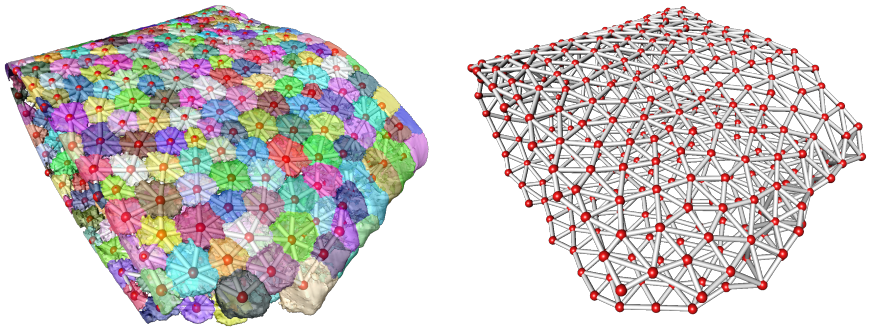
(c) Persistence value 10

(d) Persistence value 20

(e) Persistence value 50

(f) Persistence value 200

Figure 5.11: Initial segmentation results with increasing persistence value. The threshold value is always 0.5. All images are results of volume rendering on label fields.



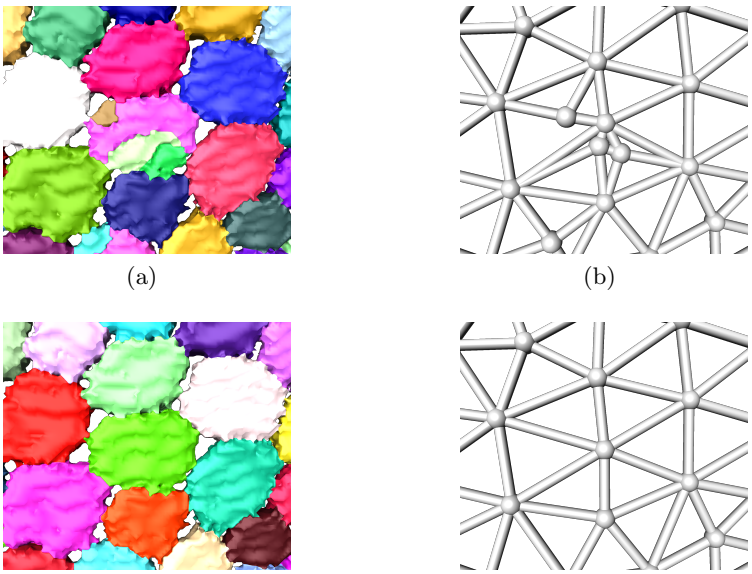
(a) (b)

Figure 5.12: Whole dataset: (a) transparent surface with dual graph; (b) dual graph.

Furthermore, the dual graph helps to identify problematic areas. In the case of a perfect segmentation, the dual graph has a regular structure. Otherwise, for example if there are multiple vertices in a very small area as shown in Figure 5.13b, there is most probable a segmentation error. Second, the dual graph is a useful tool to handle user interaction. Manual interaction with a segmentation requires the selection of individual segments. The selection of a vertex is comfortable and identifies a segment uniquely.

The graph is computed with the new module HxCreateSurfaceGraph. The current implementation constructs a graph edge between two vertices if the corresponding segments are connected in a given surface. Thus it is necessary to construct the surface belonging to the initial segmentation from the previous section. Of course it is also possible to create the edges purely based on the label field but it is easier to use the triangular surface mesh. Figure 5.14a shows an example where two neighbored segments share no common boundary so the dual graph contains no edge between them. To solve this problem it is possible to enlarge the segments before the surface creation (shown in Figure 5.14b). All unlabeled voxels which are next to a segment and which have higher values than a given threshold value get the label of that segment. This process is repeated until there are no such voxels left. The existing ZIBAmira module HxPropagateContours implements this algorithm.

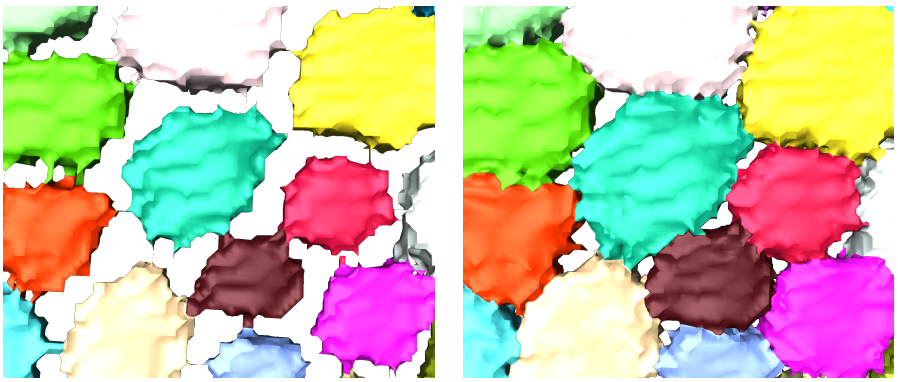
The last step is to find suitable vertex positions. The straight forward solution is to place them in the centers of gravity of the segments. This approach needs the segmentation as additional input of the graph computation module. It is easy to implement but has problems with non-convex segments. Furthermore, the vertices are inside the label field, that means the surface and the vertices are not visible at the same time. The best solution would



(a) (b)

(c) (d)

Figure 5.13: (a) and (b) show a segmentation with errors and the corresponding irregular graph structure; (c) and (d) show the corrected segmentation with the corresponding regular graph structure.



(a) (b)

Figure 5.14: Propagation of label fields: (a) surface that contains neighbored regions with no common boundary; (b) the corresponding label field has been propagated, now neighbored regions share a common boundary.

be to create the vertices on the surface but this is much more difficult and not used in the current implementation.

5.3.5 Manual improvement of segmentation

It seems unlikely to obtain a perfect segmentation result using only the fully-automatic segmentation step, although this would clearly be the best solution for the given segmentation problem. Therefore it is necessary to provide the user with tools to improve a segmentation. Manual user corrections are useful because a human can detect the searched tesserae in a volume rendering and is therefore able to detect errors in a given segmentation by simply comparing it with the volume rendering.

The usability of the manual improvement tool is important in order to minimize the needed amount of time. Of course this is only possible if the quality of the initial segmentation is high enough. The correction of a poor initial segmentation takes either a long time or it is even impossible. This underlines the importance of the initial segmentation step.

The existing segmentation editor in Amira is not capable to solve this task. Labelfields in ZIBAmira can handle a maximum of 256 labels what is not sufficient for the given problem. Additionally, there are no methods to comfortably correct the segmentation errors.

The newly developed module HxAnalyzeSegmentation should fulfill three main tasks. First, it is able to identify problematic segments automatically. The user can then decide whether corrections are needed or not. Second, it is able to merge segments in case of an oversegmentation. Third, it is able to split a segment that covers several tesserae. The interface of the module is shown in Figure 5.15. The module needs the segmentation and the corresponding dual graph as inputs.

The user must be able to select individual segments. Such user interaction is done with the help of the dual graph. There is a one-to-one-correspondence between the graph nodes and the segments in the segmentation. So the selection of a node identifies a unique segment. This one-to-one-correspondence must be preserved during the whole improvement process. Hence it is necessary to update the dual graph after each merge or split.

Identification of problematic segments

The automatic identification of problematic segments has two main purposes. First, it simply helps the user to find the segmentation errors faster. Then the user can do manual merges and splits to solve the problem. Second, it enables automatic operations executed on all critical segments. For

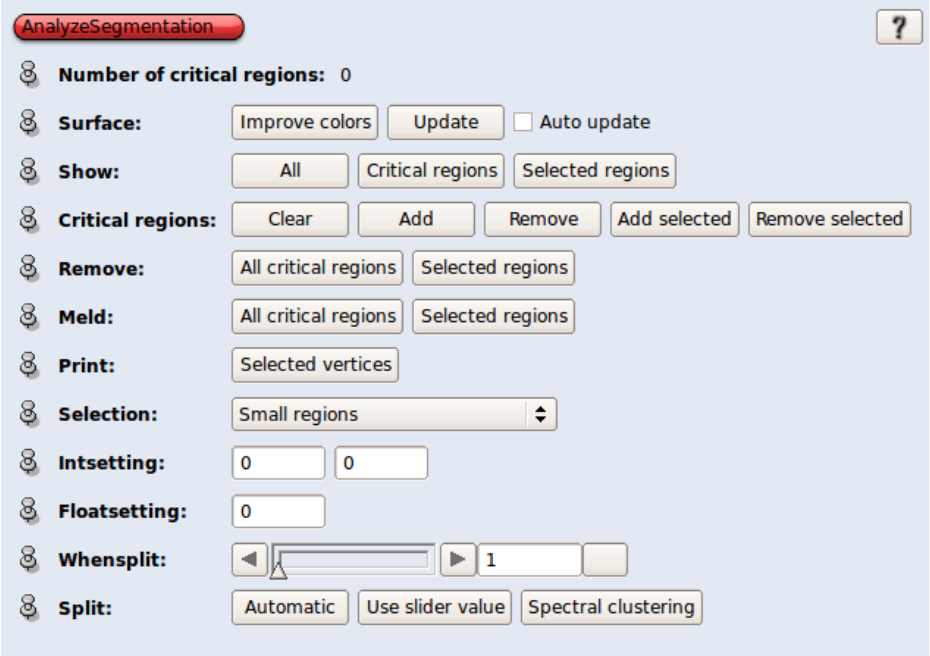


Figure 5.15: Interface of new module for manual improvement of segmentation.

example, it enables the deletion of all critical segments by merging these segments to the background. So one can remove all small artifact segments in one operation. Another example is the removal of a second structure in the background that forms an own connected component that is not connected with the main graph. It is also possible to merge all critical segments with the most probable neighbor, that is the neighbor with the most number of connecting triangles in the surface. Such operations must be used carefully, manual merges are safer because the user chooses the goal and not an algorithm.

The following list contains all implemented criteria to find critical segments.

- size: too small or too large segments
- number of neighbors: segments with too few or too many neighbors
- isolated segments: only connected to background
- enclosed segments: no connection to background
- small connected component: segments that form a connected component with too few nodes
- distance: segments with too small distance to a neighbor

(a) (b) (c)

Figure 5.16: Critical segments: (a) whole surface; (b) surface segments with less than 5 neighbors; (c) surface segments with less than 500 voxels.

Figure 5.16a shows an example of the whole surface followed by surface parts that belong to different critical segments.

Visualization

The visualization of the surface can be controlled by the HxAnalyzeSegmentation module. It is possible to show only surface parts belonging to selected vertices or belonging to critical segments as in Figure 5.16b and in Figure 5.16c. The module is also capable to adjust the colors with the goal that two neighbored segments do not get too similar colors. The original color choice is random, so it is likely that there are two neighbored, different regions that look like one in the surface visualization.

Merge

The merge of two or more segments simply means that all of them get the same label. It is an easy operation but effective if there is a slight oversegmentation and one tessera consists of several segments. The user has to select the corresponding graph vertices and after the merge all these vertices merge into one new vertex. A single merge can not deal with a segment that is part of multiple tesserae. During the merge operation all internal structures are updated, that means for example that the vertex position is recomputed, the unnecessary vertices are deleted, new edges are created and so forth. An example is shown in the first two pictures of Figure 5.18.

Split

To split one segment R consisting of r voxels into two or more tesserae equals the primary segmentation task but needs to be performed only on a

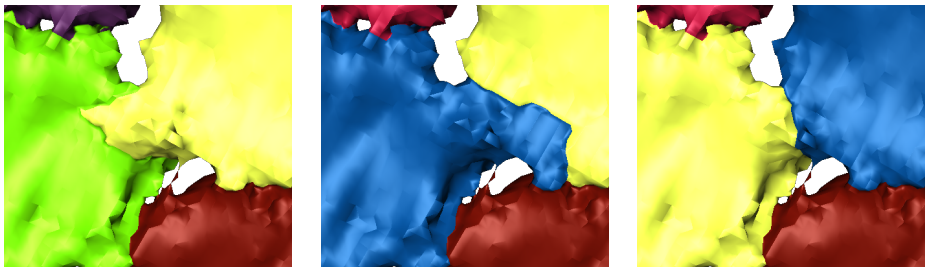
small part of the dataset. Therefore it is possible to adopt the algorithms from Section 5.2 and include them into the HxAnalyzeSegmentation module. Three splitting algorithms will be discussed: random walker, spectral clustering and a contour-tree-based algorithm.

Spectral clustering works similar to the previous approach in Section 5.2, only that the Laplacian matrix L contains voxels of one segment and not of the whole foreground. An injective mapping $M : \{0, 1, \dots r-1\} \rightarrowtail \{0, 1, \dots\}$ between the index of a voxel in R and the index in the whole dataset is needed. The first row of L stands for the voxel $M(0)$ and so forth. The Laplacian matrix is written to a scalar field, the matlab module gets the scalar field as input, does the necessary computation and the result is written back to the original segmentation label field. The advantage of this approach is that there is not much user interaction necessary. Selection of the vertex and setting the number of wished segments is enough. But if the resulting segmentation is not correct there is no possibility to use spectral clustering on R with the wished result.

Using random walker segmentation requires the user to create seeds. The algorithm only needs to work on voxels belonging to R. Because of the seeds, the user has much influence on the split. If R should be divided into two segments the user can set the seeds close to the boundary. Then it is not possible that one segment spreads itself across the boundary into the other one. Obviously this advantage has also the drawback that the seeds have to be created. If the segmentation is not successful the process can be redone with new seed positions.

The third approach uses the contour tree and is built to split a segment into exactly two parts. The idea is to iterate only over voxels in R and to select exactly one join tree node where the split should happen. All other join nodes function as merge nodes. The first implementation allows the user to choose this split node interactively and view the result immediately. It takes too much time to find a suitable node so the second implementation tries each possible node as split node and takes the best one. Under the assumption that both new segments should be nearly of the same size, the algorithm calculates the number of voxels and minimizes the absolute value of their difference. It should be possible to use the fast contour tree segmentation approach from Section 3.3.2 but currently the algorithm iterates over all voxels in R.

Currently, only the contour tree split and the spectral clustering split are implemented. Figure 5.17 shows an example where the contour-tree-based split is not capable to separate the tesserae, but the spectral clustering split succeeds.



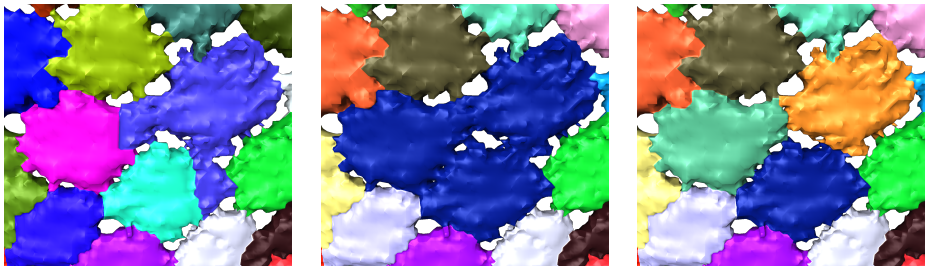
(a) (b) (c)

Figure 5.17: Different split results: (a) the yellow tessera has parts inside the green tessera, the tesserae get merged and then split; (b) contour-tree-based split was not able to correct the error ; (c) spectral-clustering-based split separated the green tessera and the yellow tessera successfully.

Combination of merge and split

The combination of merge and split operations enables the correction of more complicated segmentation errors.

For example, if one segment is part of multiple tesserae as shown in Figure 5.18a, then it is possible to merge these segments into one segment and to do a split operation on this new segment. If the number of merged segments is greater than two, then it is just possible to perform the spectral clustering split, the implemented version of the contour-tree-based split can only handle two regions.



(a) (b) (c)

Figure 5.18: Combination of merge and split: (a) three tesserae with errors; (b) three tesserae merged into one; (c) result of a successful spectral clustering split into three tesserae.

There exist segmentation errors that are not solvable with the presented tools. In such a case it is necessary to use the ZIBAmira segmentation editor. If the number of segments is larger than 256, one needs to extract

a smaller part with less than 256 segments and solve the problem with the segmentation editor. Afterwards the extracted area must be reinserted into the original segmentation. This process needs a lot of time but is the only chance to correct the remaining errors. This also underlines the importance of the presented corrections tools which work much faster.

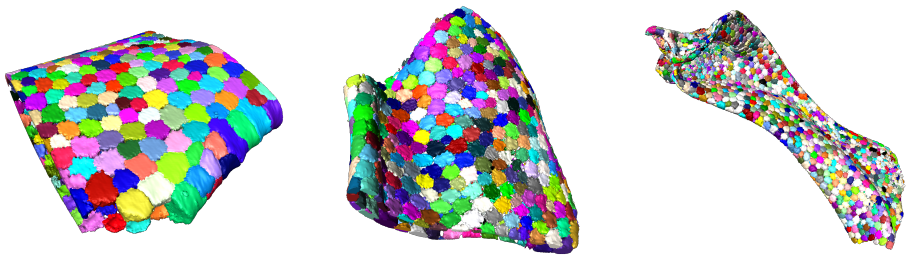
Chapter 6

Evaluation

Quantitative evaluation is difficult because there exist no ground truth segmentation to compare with. Real quantitative analysis would need a manual segmentation of a domain expert. So the focus of this chapter lies on qualitative evaluation.

6.1 Datasets

The segmentation pipeline is tested on three ray datasets. The technique used during the micro-CT scans of datasets 2 and 3 differs from the first one. The datasets 2 and 3 were freeze-dried and as a result the skeleton was squashed together. That means there is just a little space between the two sides of the dataset. This effect can be seen in the Figures 6.1b and 6.1c. More information can be found in the following table.



(a) (b) (c)

Figure 6.1: Segmentations of the three datasets.

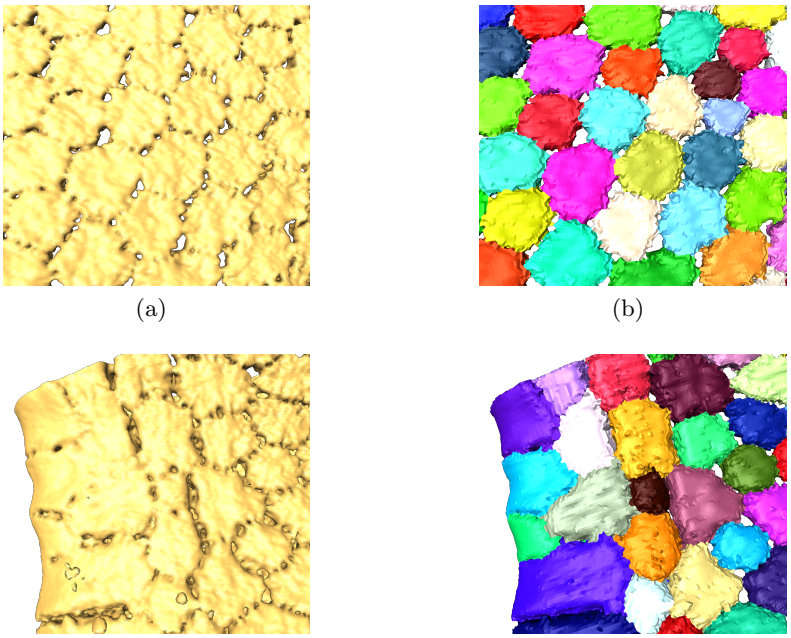
	Dataset 1	Dataset 2	Dataset 3
Size	1708×712×1142	1424×648×1249	620×548×1311
Resolution (in µm)	3.96×3.96×3.96	5.42×5.42×5.42	5.28×5.28×5.28
File size	1.4 GB	1.2 GB	445 MB
Size after resampling	427×178×285	356×162×312	310×274×655
Resolution after resampling (in µm)	15.84×15.84×15.84	21.68×21.68×21.68	10.57×10.57×10.57
File size after resampling	21.7 MB	18 MB	55.6 MB
Grayscale value range	0 .. 255	0 .. 255	0 .. 255

The datasets were resampled because the quality was still sufficient and the fast contour tree segmentation finished immediately instead of taking a few seconds.

6.2 Qualitative evaluation

As mentioned before, the user is in most cases able to identify individual tesserae. So qualitative evaluation can be done by comparing the final segmentation result with an isosurface.

The second and third datasets use the two-dimensional distance map approach because the standard three-dimensional version leads to the errors shown in Figure 5.7. The used settings are shown in the following table.

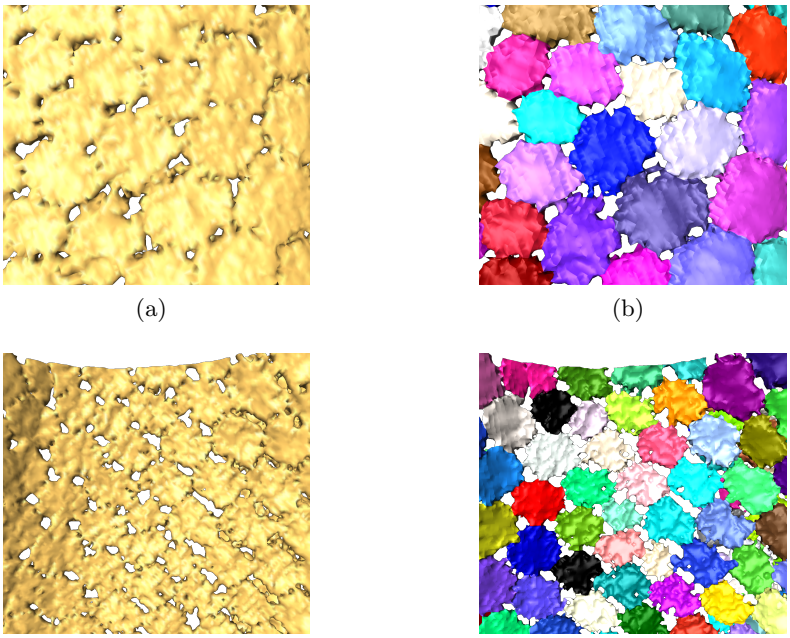


(a) (b) (c) (d)

Figure 6.2: Excellent segmentation results in dataset 1. The images on the right side show the segmentation results corresponding to the isosurfaces on the left side.

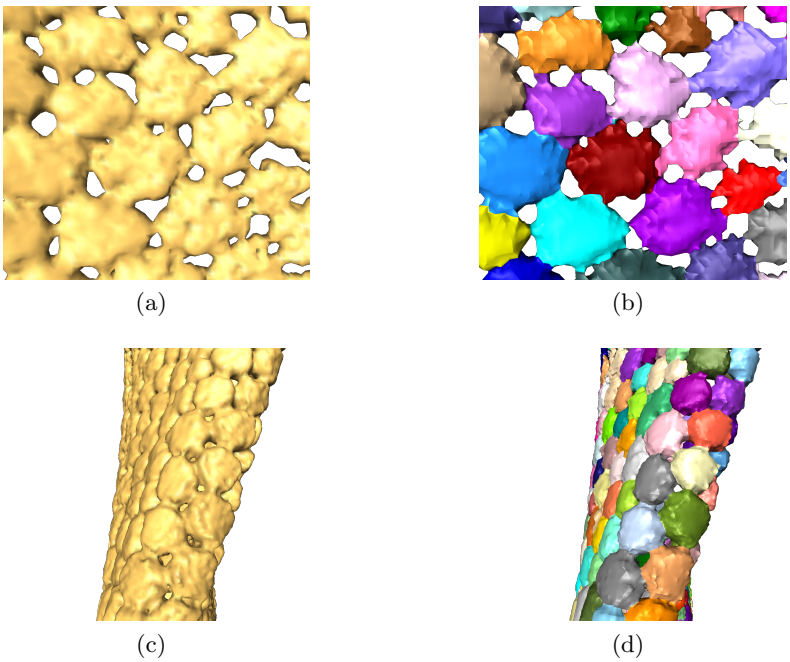
	Dataset 1	Dataset 2	Dataset 3
Threshold value	50	65	65
Type of distance map	3D	2D	2D
Persistence value	9	20	10

The Figures 6.2, 6.3 and 6.4 show areas with excellent segmentation results for all three datasets. These areas allowed such good segmentation results because there are large enough background areas (holes) separating the tesserae. The quality of the distance map result depends on the existence of such holes, otherwise the distance to the border regions cannot be measured correctly, because they do not exist as background in the foreground-background segmentation.



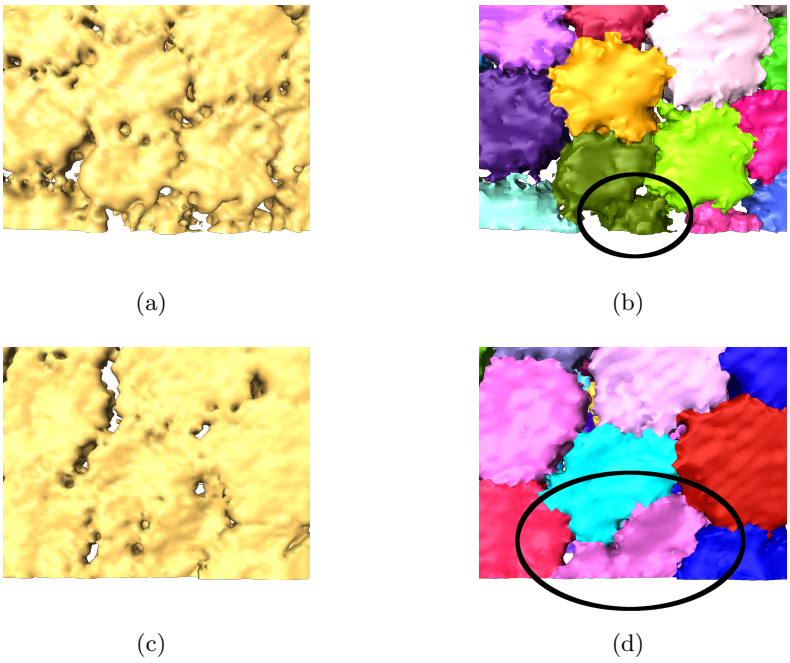
(a) (b) (c) (d)

Figure 6.3: Excellent segmentation results in dataset 2.



(a) (b) (c) (d)

Figure 6.4: Excellent segmentation results in dataset 3.



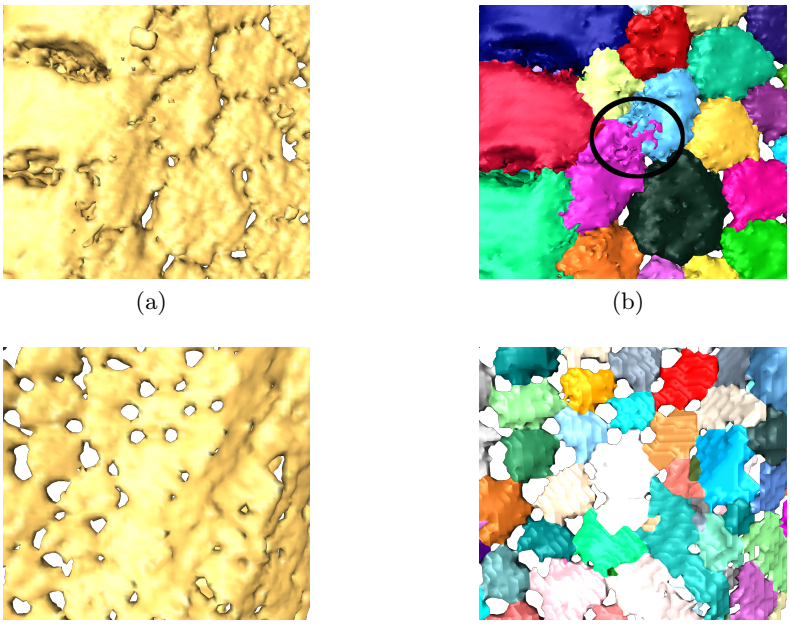
(a) (b)

(c) (d)

Figure 6.5: Images are taken from dataset 1. The border of the micro-CT scan is in the lower part of the images. (a,b) initial segmentation could not separate the green segment in 2 tesserae, contour tree based split fails, spectral clustering based split succeeds; (c,d) even the isosurface allows no clear separation, contour tree based split and spectral clustering based split fail.

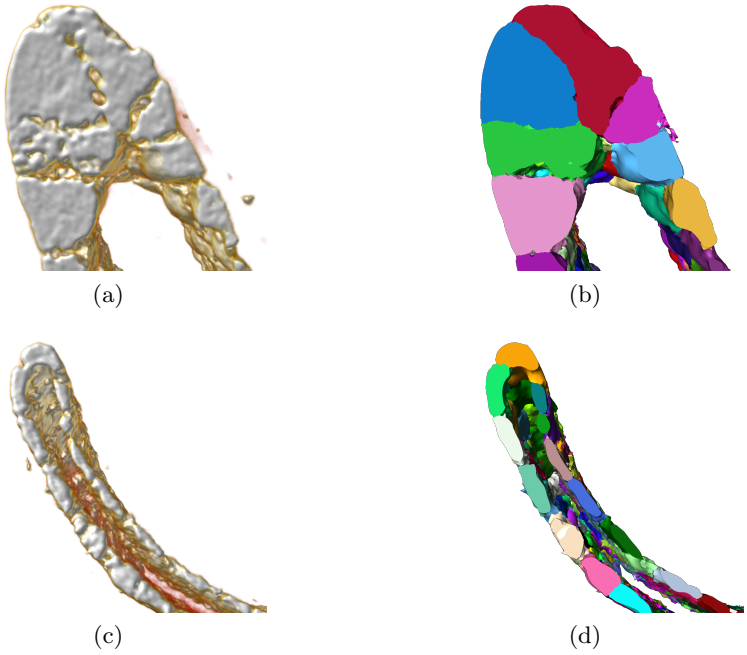
More problematic are areas at the border of the micro-CT scan. There, the real dataset is cut which leads to very small parts of tesserae whose main parts lie outside of the micro-CT scan. This problem can be solved by removing the border regions after the segmentation process. Figure 6.5 shows examples where these small parts are not separated correctly from a neighbored tessera.

The largest problem is the existence of areas where only few background voxels are separating two tesserae. In such cases the distance map values between two tesserae are not small enough, so one segment will have voxels in both tesserae. This segment can only be corrected with a split. As described in Section 5.3.5, the results of the currently existing contour-tree-based split and spectral-clustering-based split cannot be controlled by the user. So if these methods fail, there is currently no possibility to split the segment without manually using the ZIBAmira segmentation editor. The random-walker-based split described in Section 5.3.5 will most probable solve this problem. Figure 6.6 shows examples of such problematic parts.



(a) (b) (c) (d)

Figure 6.6: (a,b) images belong to dataset 1, the purple segment is too large and has voxels inside the tessera belonging to the blue segment; (c,d) images belong to dataset 3, an example for a really difficult area, there are nearly no holes in the right part of the isosurface, this leads to a wrong segmentation that is not correctable with the current splits.



(a) (b)

(c) (d)

Figure 6.7: Segmentation results of inner parts of dataset 1 (a and b) and dataset 2 (c and d).

The previous images showed only results on the surface but the performed segmentation is three-dimensional. So it is also necessary to look at segmentation results in the inner parts. This is done with help of slices through the dataset. Figure 6.7 shows segmentation results in the inner parts of the tesserae.

To summarize the results that were presented with the above example images: the segmentation quality crucially depends on the existence of holes between the tesserae. Figure 6.6c shows the isosurface for an isovalue where this property is not fulfilled. The increasing of the threshold value during the background segmentation will create these holes, but such a threshold value is not suitable for the rest of the dataset. To solve this problem, it is either necessary to improve the background segmentation or to merge these segments in the manual correction step and correct them with a random walker split.

The reached segmentation quality of the first two datasets is very good, there are only very few segments that could not be corrected with the currently available tools. The segmentation quality of dataset 3 is poor, there are successfully segmented parts as shown in Figure 6.4 but there are also multiple regions like the one shown in Figure 6.6c. The segmentation of

dataset 1 was also very fast. It seems, that the freeze-drying of datasets 2 and 3 complicates the segmentation process.

6.3 Running time

Regarding the running time, there are two discussable parts. First, the running time of the automatic algorithms, and second, the time the user needs using the presented pipeline.

All computations were carried out on a Linux computer with 4 Intel Xeon 3.0 GHz processors and 16 GB of main memory. The running time of background segmentation, dual graph generation and performing individual steps of the manual correction phase is negligible compared to standard visualization operations like volume rendering or isosurface computation. Only the computation of the two-dimensional distance map is expensive. Here, expensive means 15 minutes for the distance map computation on the resampled first dataset.

The most time-consuming step for the user of the pipeline is obviously the manual correction step. The needed time depends on the quality of the initial segmentation and how much merge and split steps are necessary for a good segmentation result. Dataset 1 could be handled fast, that means in around half an hour. Dataset 2 was more difficult and needed around three hours with standard settings.

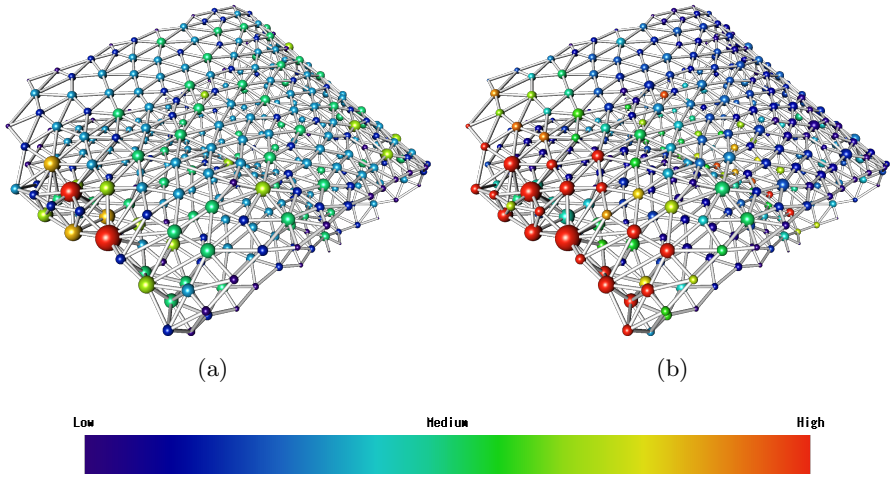
Chapter 7

Analysis of tesseral structures

A successful segmentation is the first step to realize further analysis of tesseral structures. Biologists are interested to understand relations between the shape of tesserae and properties like the number of neighbors or the curvature. The segmentation and the dual graph enable automatic calculation and comparison of these properties. For example, it is not necessary to count the neighbors of each tessera manually, what would be a common but time-consuming approach if there is no algorithmic solution available. This chapter gives only an overview of these properties and how they can be visualized, the real evaluation of these information in a biological context would be part of another work.

7.1 Size and number of neighbors

Size and neighborhood information are attached to the vertices of the dual graph and are updated during all user-dependent steps of Section 5.3.5. Thereby size only refers to the number of voxels inside a segment, not to the real volume. So the absolute values change between different image resolutions but the size values of one resolution are comparable. Figure 7.1 shows examples of annotated graphs.



(a) (b)

(c)

Figure 7.1: Graph annotations: (a) number of neighbors; (b) size of segments; both properties are encoded as the radius of the vertices (larger radius means larger value) and as the vertex colors (see colormap (c)).

Using the dual graph, it is easy to see that there is a variation in the number of neighbors. Even in very flat regions exist smaller tesserae with more neighbors than their neighboring tesserae. There are also large size differences, mainly between flat regions and regions with high curvature.

7.2 Curvature

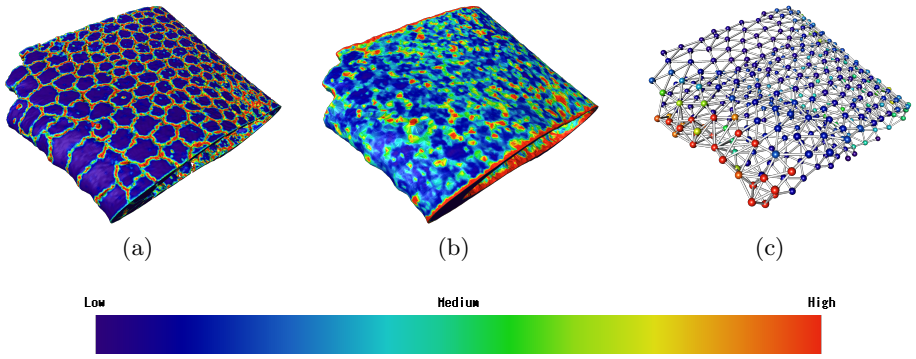
Definition 5 (Curvature)**.**
Given a three-dimensional parametric surface $S : \mathbb{R}^2 \to \mathbb{R}^3$*, for each arbitrary surface point* $P \in \mathbb{R}^3$ *exist a tangent plane. The intersection between the surface and a plane orthogonal to the tangent plane and containing* P *is a curve on the surface. Different planes lead to different curves that can have different curvatures. The minimum and maximum curvature values are called principal curvatures. The Gauss curvature is the product of the principal curvatures.*

In the actual problem, the surface is a triangular mesh and not given as a parametric equation. There are several approaches [25] how to adopt the above definitions to a discrete context. The most common idea is to use normals of triangles in a neighborhood of the point. Within a small neighborhood, the calculated curvature is a very local property. This is similar to

the continuous definition but also prone to noise. Larger neighborhoods lead to a more robust calculation but differ more from the original definition.

There is an Amira module to calculate different types of curvature for triangular surfaces. The first problem is the existence of holes between the tesserae. These are the regions with the highest maximum curvature as seen in Figure 7.2a but these are not the regions of interest. Second, even if these holes are filled as in Figure 7.2b, the surface is still uneven and the tesserae on the border of the whole structure have no larger values than parts in flat regions.

A solution is to calculate curvature values only for the vertices of the dual graph as shown in Figure 7.2c. All other properties are also stored on the graph vertices and the graph mesh is fine enough to deliver meaningful curvature information. The new algorithm first calculates normals for each vertex by averaging the normals of the adjacent graph faces. Then it calculates the curvature for a vertex v by comparing it with the normals of adjacent vertices. For Figure 7.2c, the angles between the normal of v and all of its neighbor normals are calculated and v gets simply the maximum of these angles. Of course it would also be possible to use other variants, for example to average over all these angles.



(a) (b) (c)

(d)

Figure 7.2: Curvature information: (a) maximum curvature on original surface; (b) maximum curvature on smoothed surface; (c) curvature on graph vertices; (d) colormap.

The largest curvature values in Figure 7.2c are in the large, strongly curved regions. These are the searched regions, the surface computation methods in 7.2a and 7.2b were not able to identify them.

Chapter 8

Future work

Chapter 5 described a segmentation pipeline for separating individual tesserae in micro-CT scans. There are several opportunities how the presented work can be used or improved in the future. The work on the segmentation of tesserae is not finished with the end of this master thesis. The project will continue, so hopefully some of the following ideas will be implemented soon.

8.1 Improvement of the presented pipeline

Further improvement of the presented tools and algorithms is necessary. This improvement includes work on the usability of the segmentation pipeline. The manual correction step contains no possibility to undo the last action. A split can be undone by a simple merge, the opposite direction is more difficult because most probably the split will not separate the segment in exactly the same parts as before.

Another interesting feature would be that the manual correction module automatically zooms to an incorrect part of the dataset, the user corrects the errors and restarts the process. The identification of problematic areas can be done with algorithms from Section 5.3.5.

The number of tested datasets in the evaluation chapter is currently too small. It is important to use the pipeline on other ray and shark datasets for further evaluation and improvement of the process. A dataset with a given ground truth segmentation would enable a real quantitative evaluation.

The spectral clustering implementation uses the ZIBAmira Matlab module. So the input Laplacian matrix is written to a scalar field that lies in the object pool. This is quite inefficient and it would be a great improvement

to store the Laplacian matrix only internally as a sparse matrix and use Matlab directly.

The initial segmentation can lead to a lot of small segments. Automatic removal with algorithms described in Section 5.3.5 is prone to errors. On the other hand, manual corrections take a lot of time. It might be possible to remove this problem by using a persistence criterion including the size of the segments as described in Section 3.3.1.

8.2 Application of new algorithms

The two main drawbacks of the presented spectral clustering method are the running time and that the number of segments must be known before the computation. But the number of tesserae is not known, except if the user counts them manually. The running time might be reduced by resampling the dataset and by using a multi-resolution approach. Also concepts like super-voxels as a combination of multiple individual voxels could lower the running time. Robust Perron Cluster Analysis [26] was originally developed to find almost invariant subsets of states in Markov chains but the method can also be applied as a clustering algorithm [27] using eigenvectors of a row stochastic matrix. This matrix contains transition probabilities. The algorithm has much similarities with the spectral clustering algorithm presented in Section 2.1.1 but has one significant advantage: Robust Perron Cluster Analysis does not need the number of clusters as input. Applications of Robust Perron Cluster Analysis as a clustering method can be found in [27]. It might be possible to use a spectral clustering algorithm based on Robust Perron Cluster Analysis for the segmentation of tesserae in a large dataset.

The Morse-Smale complex segments the domain into regions of uniform gradient flow behavior. A separation into segments where the gradient flow has the same target leads to segments belonging to a local maximum. There exist algorithms to simplify Morse-Smale complexes to remove irrelevant segments. This simplification is also based on a persistence criterion. It could be possible to use a Morse-Smale complex segmentation on the distance map that merges segments belonging to different maxima in one tessera but preserves the intertesseral joints. An example for Morse-Smale complex segmentation can be found in the paper of Laney et al. [28].

An active shape model [11] of a tessera could be used to do a surface-based segmentation approach. Problems are that the shape of tesserae can vary widely so it is not clear if one model would be enough. Furthermore, the creation of the model is not trivial.

The initial segmentation fails if the distance map is not capable to detect the intertesseral joints. This was shown in Figure 6.6c. The quality of the dis-

tance map depends on the foreground-background segmentation. Right now, background segmentation is done with region growing and thresholding. It might be possible that other algorithms like Chan-Vese segmentation [13] or local thresholding algorithms [2] improve the result.

In the initial segmentation step described in Section 5.3.3, the algorithm should perform a slight oversegmentation because manual merges are easier than splits. Another approach would be to use an algorithm for automatic merges. The initial segmentation starts with an extreme oversegmentation and an algorithm for automatic merges creates the individual tesserae. Such an approach is used in the paper by Haris et al. [29], where the initial segmentation is performed by a watershed segmentation.

8.3 Biological applications of segmentation

Segmentations of tesserae help biologists to analyze the shape of tesserae, how the shape changes with the curvature and so forth. With successful segmentations of a series of micro-CT scans of the same species but with different ages, it would be possible to study how shape and tiling patterns change with the aging of the species.

Another opportunity would be to carry out finite element simulations to study stress relaxation properties of the tessellated cartilage. It would also be possible to use a 3D printer to obtain a real three-dimensional model of the tessellated structure.

Of course it is also possible to study other materials as long as these materials consist of some kind of tiles.

Chapter 9

Conclusion

This thesis described a segmentation pipeline to separate tesselated structures in micro-CT scans of rays and sharks. The pipeline consists of two main parts. First, the dataset is segmented with an initial automatic segmentation step based on a newly developed fast version of a contour-tree-based segmentation algorithm. Second, the automatic segmentation is corrected by the user. Furthermore, the dual graph of the segmentation is created as an abstraction of the segmentation and to support user interaction. Additionally, algorithms for the segmentation of datasets with few tesserae were suggested: spectral clustering and random walker. These algorithms are also useful during the manual correction step.

The suggested pipeline was implemented into the visualization software system ZIBAmira and used for the segmentation of ray datasets with hundreds of tesserae. The quality of the segmentation result depends on whether there exist parts of uncalcified cartilage between the tesserae.

In this thesis it was shown that the suggested segmentation pipeline is capable to solve the given segmentation task. Some problems remain, mainly concerning the distance map which is the basis for the following initial segmentation step. The initial segmentation becomes error-prone if the distance map is not able to identify the boundaries between two tesserae. The project in which the thesis was prepared will continue and the suggestions from the previous chapter will hopefully solve the remaining problems.

Bibliography

[1] Dean, Mason N. ; Summers, Adam P.: Mineralized cartilage in the skeleton of chondrichthyan fishes. In: *Zoology* 109 (2006), Nr. 2, S. 164–168

[2] Wirjadi, Oliver: *Survey of 3d image segmentation methods*. ITWM, 2007

[3] Pal, Nikhil R. ; Pal, Sankar K.: A review on image segmentation techniques. In: *Pattern Recognition* 26 (1993), September, Nr. 9, 1277–1294. http://dx.doi.org/10.1016/0031-3203(93)90135-j. – DOI 10.1016/0031–3203(93)90135–j. – ISSN 00313203

[4] Mortensen, Eric N. ; Barrett, William A.: Intelligent Scissors for Image Composition. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1995 (SIGGRAPH '95). – ISBN 0–89791–701–4, 191–198

[5] Vezhnevets, Vladimir ; l. eta: *"GrowCut" – Interactive Multi-Label N-D Image Segmentation By Cellular Automata*. 2005

[6] Boykov, Y.Y. ; Jolly, M.-P.: Interactive graph cuts for optimal boundary amp; region segmentation of objects in N-D images. In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* Bd. 1, 2001, S. 105–112 vol.1

[7] Shi, Jianbo ; Malik, Jitendra: Normalized Cuts and Image Segmentation. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), August, Nr. 8, 888–905. http://dx.doi.org/10.1109/34.868688. – DOI 10.1109/34.868688. – ISSN 0162–8828

[8] Grady, L.: Random Walks for Image Segmentation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 28 (2006), Nov, Nr. 11, S. 1768–1783. http://dx.doi.org/10.1109/TPAMI.2006.233. – DOI 10.1109/TPAMI.2006.233. – ISSN 0162–8828

[9] Luxburg, Ulrike: A Tutorial on Spectral Clustering. In: *Statistics and Computing* 17 (2007), Dezember, Nr. 4, 395–416. http://dx.doi.org/10.1007/s11222-007-9033-z. – DOI 10.1007/s11222–007–9033–z. – ISSN 0960–3174

[10] Beucher, Serge ; Lantuéjoul, Christian: Use of watersheds in contour detection. (1979)

[11] Cootes, T. F. ; Taylor, C. J. ; Cooper, D. H. ; Graham, J.: Active Shape Models&Mdash;Their Training and Application. In: *Comput. Vis. Image Underst.* 61 (1995), Januar, Nr. 1, 38–59. http://dx.doi.org/10.1006/cviu.1995.1004. – DOI 10.1006/cviu.1995.1004. – ISSN 1077–3142

[12] Mumford, D. ; Shah, J.: Boundary detection by minimizing functionals. In: *IEEE Conference on Computer Vision and Pattern Recognition*, 1985

[13] Chan, T.F. ; Vese, L.A: Active contours without edges. In: *Image Processing, IEEE Transactions on* 10 (2001), Feb, Nr. 2, S. 266–277. http://dx.doi.org/10.1109/83.902291. – DOI 10.1109/83.902291. – ISSN 1057–7149

[14] Hege, Hans-Christian ; STALLING, DETLEV ; SEEBASS, MARTIN ; Zockler, Malte: A Generalized Marching Cubes Algorithm Based On Non-Binary. (1997)

[15] Boyell, Roger L. ; Ruston, Henry: Hybrid Techniques for Real-time Radar Simulation. In: *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference*. New York, NY, USA : ACM, 1963 (AFIPS '63 (Fall)), 445–458

[16] Carr, Hamish ; Snoeyink, Jack ; Axen, Ulrike: Computing Contour Trees in All Dimensions. In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2000 (SODA '00). – ISBN 0–89871–453–2, 918–926

[17] Kreveld, Marc van ; Oostrum, René van ; Bajaj, Chandrajit ; Pascucci, Valerio ; Schikore, Dan: Contour Trees and Small Seed Sets for Isosurface Traversal. In: *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*. New York, NY, USA : ACM, 1997 (SCG '97). – ISBN 0–89791–878–9, 212–220

[18] Tarasov, Sergey P. ; Vyalyi, Michael N.: Construction of Contour Trees in 3D in O(N Log N) Steps. In: *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*. New York, NY, USA : ACM, 1998 (SCG '98). – ISBN 0–89791–973–4, 68–75

[19] Carr, Hamish ; Snoeyink, Jack: Path Seeds and Flexible Isosurfaces Using Topology for Exploratory Visualization. In: *Proceedings of the Symposium on Data Visualisation 2003*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2003 (VISSYM '03). – ISBN 1–58113–698–6, 49–58

[20] Weber, Gunther H. ; Dillard, Scott E. ; Carr, Hamish ; Pascucci, Valerio ; Hamann, Bernd: Topology-Controlled Volume Rendering. In: *IEEE Transactions on Visualization and Computer Graphics* 13 (2007), März, Nr. 2, 330–341. http://dx.doi.org/10.1109/TVCG.2007.47. – DOI 10.1109/TVCG.2007.47. – ISSN 1077–2626

[21] Rosanwo, Olufemi: *Interactive Multi-Object Segmentation with Max-Tree Filtering*, Diplomarbeit

[22] Edelsbrunner, Herbert ; Harer, John: *Persistent Homology – a Survey*

[23] Stalling, Detlev ; Westerhoff, Malte ; Hege, Hans-Christian: *Amira: a highly interactive system for visual data analysis*. 2005

[24] Bresenham, J. E.: Algorithm for Computer Control of a Digital Plotter. In: *IBM Syst. J.* 4 (1965), März, Nr. 1, 25–30. http://dx.doi.org/10.1147/sj.41.0025. – DOI 10.1147/sj.41.0025. – ISSN 0018–8670

[25] Rusinkiewicz, Szymon: Estimating Curvatures and Their Derivatives on Triangle Meshes. In: *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium*. Washington, DC, USA : IEEE Computer Society, 2004 (3DPVT '04). – ISBN 0–7695–2223–8, 486–493

[26] Deuflhard, P. ; Weber, M.: Robust Perron Cluster Analysis in Conformation Dynamics. In: *Lin. Alg. Appl. – Special Issue on Matrices and Mathematical Biology* Bd. 398. 2005, S. 161 – 184

[27] Weber, Marcus ; Kube, Susanna: Robust Perron Cluster Analysis for Various Applications in Computational Life Science. In: *Proceedings of the First International Conference on Computational Life Sciences*. Berlin, Heidelberg : Springer-Verlag, 2005 (CompLife'05). – ISBN 3–540–29104–0, 978–3–540–29104–6, 57–66

[28] Laney, D. ; Bremer, P. T. ; Mascarenhas, A. ; Miller, P. ; Pascucci, V.: Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities. In: *IEEE Transactions on Visualization and Computer Graphics* 12 (2006), September, Nr. 5, 1053–1060. http://dx.doi.org/10.1109/TVCG.2006.186. – DOI 10.1109/TVCG.2006.186. – ISSN 1077–2626

[29] Haris, K. ; Efstratiadis, S.N. ; Maglaveras, N. ; Katsaggelos, AK.: Hybrid image segmentation using watersheds and fast region merging. In: *Image Processing, IEEE Transactions on* 7 (1998), Dec, Nr. 12, S. 1684–1699. http://dx.doi.org/10.1109/83.730380. – DOI 10.1109/83.730380. – ISSN 1057–7149