MESH GENERATION FROM IMAGING DATA

Marcelo Siqueira

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2006

Jean Gallier
Supervisor of Dissertation

Rajeev Alur
Graduate Group Chairperson

To my wife Poliana,

my daughter Sofia, and

my parents Roberto and Dilma.

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Jean Gallier, for his guidance and advice throughout the entire execution of this thesis work. Jean also helped me become a lecturer at the CIS department during my last year of my PhD. This job allowed me to stay in the United States for one more year and conclude my PhD studies.

I also would like to thank the committee members for their insightful comments and suggestions: Camillo J. Taylor, George Biros, Noel J. Walkington, and Sudipto Guha.

In particular, I feel honored and privileged for having had the opportunity of discussing issues related to the meshing problem with Noel, whose work on the meshing problem has been influential and inspirational to me and the entire meshing research community.

My heartfelt thanks go to my closest collaborators in this thesis: Jim Gee, Longin Jan Latecki, and Suneeta Ramaswami.

Jim introduced me to the particular case of the meshing problem addressed in this thesis, and he was my thesis advisor during my first year as a PhD student in the CIS department.

Longin provided a characterization of 3D well-composed images, which turned out to be extremely useful for my thesis. I am very glad to have teamed up with him to build upon his previous and elegant work.

Suneeta guided me through the meshing literature and invited me to work with

her on our quadrilateral meshing algorithm. She sponsored my participation in meshing conferences, and studied and discussed several research papers with me.

Many other people have influenced my thesis work, either by direct collaboration and/or through helpful discussions. I am very grateful for having collaborated with Tessa Sundaram and Peer Stelldinger, and for the opportunity of exchanging ideas with Hui (Gary) Zhang, Dianna Xu, Paul Yushkevich, Nick Tustison, Brian Avants, André Souza, Bruno Carvalho, Carlos Prolo, Rodrigo Carceroni, Milan Djabirov, Paulo Pagliosa, and Fábio Henrique.

I also would like to thank the administrative staff of the CIS department, in particular, Amy Calhoun, Gail Shannon, Janean Williams, Jennifer Finley, Kamila Mauro, Mark West, Mike Felker, and Rita Powell. Mike Felker, the CIS graduate coordinator, has been a guardian angel for all graduate students and has helped me deal with several administrative issues since I became a PhD student in the CIS department.

I would like to acknowledge several people who made my time in the United States very enjoyable: my Brazilian friends Ronaldo Ferreira, Everaldo (Pacato) Venancio, Fabrício Vargas, Kátia and Eduardo Lima, Juciene and André Souza, Bruno Carvalho and Simone Nunes, Hellen Pacheco and Rodrigo Carceroni, Emília e Cássio Turra, Rucélia e Fernando Moreira, Antônio Gidi and Ana Cláudia Costa, Luís Araújo, Francisco Brito, Tânia Calovi, Siome Goldenstein, and Carlos Prolo; my school friends Hui (Gary) Zhang, Marc Corliss, Jean Griffin, Pat Palmer, David Matuszek, Milan Djabirov, Stanislav Angelov, Sid Suri, Harb Boulos, Axel Bernal, Brian Avants, Susana Santos, and Mark Lee; and my colleagues from the Orange Crush soccer team: Preston, Catherine, Oleg, Alice, Jason, and Steve.

I also received constant support of many of my colleagues from the DCT (Department of Computing and Statistics) at UFMS (Federal University of Mato Grosso do Sul), where I have an appointment as an assistant professor. I would like to mention a few of them: Fábio Henrique, Nalvo Almeida, José Craveiro, Paulo Pagliosa, Marcelo

Henriques de Carvalho, Edson Cáceres, Nahri Moreano, and Henrique Mongelli. I cannot forget to express my gratitude to my mentors during my undergraduate and Master programs back in Brazil: Jackson William Marques de Carvalho, Antônio Castelo Filho, and Maria Cristina Ferreira de Oliveira.

Pursuing the doctorate abroad would not be possible without the financial support from CNPq (Brazilian Council for Scientific and Technological Development), UFMS, and the CIS department at UPenn. In particular, I am extremely grateful to Fernando Pereira, head of the CIS department, David Matuszek, director of the MCIT (Master of Computer and Information Technology), and Norm Badler, director of the DMD (Digital Media Design) undergraduate program, for giving me the opportunity to become a lecturer and fund myself in my last year as a PhD student at CIS.

Finally, I would like to thank my parents, Roberto and Dilma, my brother Beto, and my sister Raquel for their love and unconditional support. My parents made me understand the value of education very early in life. For that, I will be forever grateful to them. I would not have succeeded in this journey without the love, patience, care, kindness, and full support of my wife Poliana. We had a lovely and unforgettable time in the United States, and most importantly of all, we were both blessed with the birth of our first child Sofia.

ABSTRACT

MESH GENERATION FROM IMAGING DATA

Marcelo Siqueira

Jean Gallier

Digital images from computerized tomography (CT) and magnetic resonance (MR) scanners can be used to create computer models of anatomical shapes. These models are typically used in biomedical applications for the purpose of physical simulation and visualization. In this thesis, we describe new algorithms for creating models from 2D and 3D binary digital images. Our models are meshes of 2D shapes represented by 2D binary digital images, and meshes of the surface of 3D shapes represented by 3D binary digital images. More specifically, this thesis contains the following two contributions:

First, we give an algorithm for converting constrained triangular meshes of polygonal domains into constrained and strictly convex quadrilateral meshes of the same domain. Our algorithm has linear time in the number of triangles of the input triangular mesh, produces a bounded number of quadrilaterals, offers better bounds than similar algorithms that also produce strictly convex quadrilateral meshes of bounded size, and tends to preserve the grading of the input triangular mesh. We also present examples to demonstrate that our algorithm can be successfully used to create quadrilateral meshes from 2D binary digital images of anatomical shapes, such as the human brain.

Second, we provide a new algorithm for generating simplicial surface meshes that approximate the boundary of 3D shapes represented by 3D binary digital images. Our algorithm is based on a simplified version of one of two known algorithms for generating provably good quality simplicial approximations of smooth, implicit surfaces. Provably good quality simplicial surfaces are very desirable for visualization purposes, and the main advantage of our algorithm is to offer this provable quality guarantee.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A 2D (resp. 3D) digital image is a map that assigns a real value, called color, to each point of a finite and discrete grid of points of the Euclidean plane (resp. space). In biomedical applications, digital images are produced by imaging devices, such as computerized tomography (CT) and magnetic resonance (MR) scanners. These devices measure some physical quantity at points of a continuous domain (e.g., a human organ) represented by the points of the image grid. Image points located in regions of the continuous domain with similar physical properties are assigned similar colors.

Digital images from CT and MR scanners can be used to create computer models of human anatomy for the purpose of physical simulation and visualization. Here, we are particularly interested in creating computer models from 2D digital images for the purpose of physical simulation, and computer models from 3D digital images for the purpose of visualization. In both cases, our models are discrete approximations (i.e., *meshes*) of the continuous domain (or of its boundary) represented by the digital image.

A *mesh* is a discretization of a continuous domain into simple shapes, such as triangles and quadrilaterals if the domain is a subset of the Euclidean plane or a surface in the Euclidean space, and tetrahedra and hexahedra if the domain is a

subset of the Euclidean space. In physical simulations, the accuracy of a problem's solution, and the efficiency with which it is obtained, are highly dependent on a variety of mesh parameters, such as number of elements of the mesh and their shape, size, and direction [10, 133, 47, 122]. Furthermore, the "optimal" value for each mesh parameter also depends on the problem, and it may largely vary from problem to problem [122].

Triangular meshes have been extensively investigated by the meshing community, and their theoretical properties are now well understood [8]. Besides, algorithms for generating provably good triangular meshes have been proposed and successfully implemented [32, 116, 121, 95]. In contrast, the generation of provably good quadrilateral meshes is not as well understood. While a few algorithms exist to generate quadrilateral meshes of bounded size [44, 9, 101], bounded largest angle [9], and with directionality control [123, 137], there is no known algorithm for generating quadrilateral meshes that is provably guaranteed to simultaneously optimize several mesh parameters.

Since there are applications in which quadrilateral meshes may be more desirable than triangular ones [3, 78], provably good algorithms for producing quadrilateral meshes are of great interest. Here, we provide a new algorithm for generating quadrilateral meshes of polygonal regions of the Euclidean plane. Our algorithm is based on an indirect approach that converts a triangular mesh into a quadrilateral mesh of the input domain. This approach relies on the premise that a provably good quality quadrilateral mesh may be more easily generated from an existing good quality triangular mesh, which can be obtained by one of the triangular meshing algorithms in [32, 116, 121, 95].

Our quadrilateral meshing algorithm is provably guaranteed to produce a bounded number of quadrilaterals (in terms of the number of the triangles of the input triangular mesh), and this this bound is better than the ones provided by previous algorithms that enjoy the same guarantee [44]. Another feature of our algorithm is

that it lends itself nicely for generating quadrilateral meshes from 2D binary digital images, which are images in which each point is assigned one of two colors only. We used our algorithm for obtaining meshes from images of the human brain, and we evaluated the quality of the output meshes with respect to an image registration application [125, 113].

We also provide a new solution for the problem of generating surface meshes from 3D binary digital images. Such surface meshes are triangulated surfaces that approximate the boundaries of the continuous objects represented by the image. They are very appropriate for visualization purposes, as there are several algorithms (both in software and hardware) for fast rendering of triangles. These surface meshes should also possess a number of important properties, which can save storage and affect the performance and accuracy of algorithms for visualization, editing, and animation.

For instance, the surface meshes should be adaptive, adjusting triangle size to local curvature in order to faithfully represent a shape with a minimum number of triangles. Keeping the number of triangles around a minimum helps to interactively view, edit, and animate large surfaces. Surface meshes should also capture the topology of the boundary of the object represented by the image, and its triangles should have a good aspect ratio, i.e., they should be as close as possible to an equilateral triangle.

Although several algorithms for generating surface meshes from 3D binary digital images have been developed in the past years [87, 117, 139, 52, 73, 77], none of them are provably guaranteed to generate surface meshes with *all* desirable properties mentioned above. More recently, two algorithms have been developed for generating simplicial approximations of implicitly defined surfaces [15, 26]. Both algorithms are guaranteed to produce triangulated surfaces that enjoy the desirable properties mentioned above.

Both algorithms in [15, 26] can also be used for generating surface meshes from

3D binary digital images. All we have to do is to first define a smooth implicit surface that approximates the boundary of the object represented by the image, and then use one of the two algorithms for creating a simplicial approximation for the implicit surface. This is precisely what our solution for the aforementioned surface meshing problem does.

Our solution has three steps. First, we "perturb" the input image, so that it becomes a *well-composed image*, i.e., a 3D binary digital image for which we can unambiguously define a surface $S \subset \mathbb{R}^3$ from the image map, which separates the image points assigned to a color from the others. Second, we define an implicit surface that has the same topology as S. Finally, we used a simplified version of the algorithm in [26] to generate a surface mesh from the implicit surface, which also has the same topology as S.

The main advantage of our solution is the fact that it enjoys the same properties as the algorithm in [26]: the triangles of the output surface are guaranteed to have good aspect ratio, the size of the triangles are adapted to the local curvature of the surface, and the number of triangles is a constant factor of the number of triangles of any mesh of the input surface that satisfies a certain size criterion[1] (see Section 7.6.6). The main contribution of our solution is the simplification of the algorithm in [26], which was made possible by its first and second steps. These steps in turn can be seen as independent contributions, as they have applications to other (un)related problems.

This thesis is organized as follows: Chapter 2 details the problem of generating a mesh from a 2D binary digital image. Chapter 3 describes our quadrilateral meshing algorithm for the problem presented in Chapter 2. Chapter 4 provides details of the problem of generating a surface mesh from a 3D binary digital image. Chapter 5 and Chapter 6 describe the first and second steps of our proposed solution for the

[1]This criterion does not imply that the number of triangles of the mesh is minimum or a constant factor of the minimum.

problem in Chapter 4, respectively. Chapter 7 introduces the algorithm in [26], and Chapter 8 describes details of our simplification of the algorithm in [26] (i.e., the third step of our solution). Finally, Chapter 9 summarizes our contributions and discusses future work. We also provide two appendices. Appendix A contains some basic concepts from point-set topology, combinatorial topology and discrete geometry, which are often used in several chapters of this thesis. Appendix B presents some proofs concerning the results in Chapter 3.

Chapter 2

Meshes from 2D Images

This chapter introduces the problem of generating meshes from two-dimensional imaging data. It also gives an overview of our solution to this problem and its main contributions. Most of the material in this chapter are based on a book on digital geometry by Herman [65], a survey paper by Bern and Eppstein [8], and the books by Frey and George [47] and Edelsbrunner [39] on mesh generation and Delaunay triangulations.

2.1 Two-Dimensional Digital Images

For any positive real number δ, we define

$$\delta\mathbb{Z}^2 = \{(\delta m_1, \delta m_2) \in \mathbb{R}^2 \mid (m_1, m_2) \in \mathbb{Z} \times \mathbb{Z}\}.$$

Definition 2.1.1. A *two-dimensional (gray-scale) digital image* $\mathcal{I} : D \to V$ is a function from a set $D \subset \mathbb{R}^2$ to a set $V \subseteq \mathbb{R}$ such that D is a translation of the set $\delta\mathbb{Z}^2$; that is, we can express D by the pair (O, δ),

$$D = \{p \in \mathbb{R}^2 \mid p = O + \delta \cdot g,\ g \in \mathbb{Z} \times \mathbb{Z}\},$$

where $O \in \mathbb{R}^2$. We refer to D as the *image domain* or *image grid*, to the elements in V as *colors*, to δ as the *grid spacing*, and to the point O as the *grid origin*.

A digital image $\mathcal{I} : D \to V$ can be viewed as a discrete sampling of a continuous function $f : U \subset \mathbb{R}^2 \to \mathbb{R}$ at the points of D, i.e., $\mathcal{I}(p) = f(p)$ for all $p \in D$, and $D \subset U$. In medical applications, digital images are produced by imaging devices, such as computerized tomography (CT) and magnetic resonance (MR) scanners. These devices measure some physical quantity represented by the function f at a discrete set of points D of a continuous object, which is represented by the domain U of f. So, the values of $\mathcal{I}$ in regions of the object with similar physical properties are nearly the same.

Definition 2.1.2. A *two-dimensional binary digital image* is a two-dimensional digital image,

$$\mathcal{I} : D \to V,$$

in which the set V of colors is the binary set $\{0, 1\}$. We shall denote a binary image $\mathcal{I}$ by the pair (D, X), where D is the grid of $\mathcal{I}$ and X is the set

$$X = \{p \in D \mid \mathcal{I}(p) = 1\}.$$

The sets X and X^c, where

$$X^c = D - X = \{p \in D \mid \mathcal{I}(p) = 0\}$$

is the complement set of X with respect to D, are commonly referred to as the *foreground* and the *background* of (D, X), respectively. Since every background point is assigned the color 0 and every foreground point is assigned the color 1, we sometimes denote X by X_1 and X^c by X_0.

A binary digital image (D, X) is in general obtained from a gray-scale digital image $\mathcal{I}' : D \to V$ through *segmentation* or *thresholding* [57]. Both operations classify each point p of D as belonging to either X_0 or X_1 based on the value of $\mathcal{I}'$ at p. A binary image can be viewed as a way of highlighting certain regions of a gray-scale image, which are in general represented by the foreground. An important

feature of digital images is that the geometry of regions of the continuous object represented by image points with similar colors can be approximated from the colors of the points, certain adjacency relations on grid points, and the notion of *continuous analog*.

Definition 2.1.3. Let $CA : D \to \mathcal{P}(\mathbb{R}^2)$ be the function that identifies each point $p = (p_1, p_2)$ of D with the square

$$\left[p_1 - \frac{\delta}{2}, p_1 + \frac{\delta}{2}\right] \times \left[p_2 - \frac{\delta}{2}, p_2 + \frac{\delta}{2}\right],$$

where $\mathcal{P}(\mathbb{R}^2)$ denotes the power set of $\mathbb{R}^2$, and δ is the grid spacing of D. We refer to $CA(p)$ as the *continuous analog* of p, or simply as the *pixel* of p.

We can extend the definition of continuous analog of a point to a set of points as follows:

Definition 2.1.4. Let X be any subset of D, and let $CA : \mathcal{P}(D) \to \mathcal{P}(\mathbb{R}^2)$ be a function such that

$$CA(X) = \bigcup_{p \in X} CA(p).$$

We refer to $CA(X)$ as the *continuous analog of the set* X.

Figure 2.1 illustrates the continuous analog $CA(X)$ of a set $X = \{a, b, c\}$ consisting of three points a, b, and c of a grid. We now define two important binary relations on D:

Definition 2.1.5. Two distinct points p and q of D are said to be *edge-adjacent* if $CA(p)$ and $CA(q)$ share an edge, or equivalently, if one of the coordinates of p and q is the same and the other one differs by δ, where δ is the grid spacing of D. Two distinct points p and q of D are said to be *corner-adjacent* if $CA(p)$ and $CA(q)$ share a vertex but not an edge, or equivalently, if the same coordinates of p and q differ by δ.

For instance, points a and b in Figure 2.1 are edge-adjacent, while points b and c are corner-adjacent but not edge-adjacent. The edge-adjacency relation is also known as 4-*adjacency*. Another useful binary relation on D is the 8-adjacency binary relation:

Definition 2.1.6. Two points p and q of D are said to be 8-*adjacent* if they are edge- or corner-adjacent. For instance, in Figure 2.1, a and b are 4- and 8-adjacent, and b and c are 8-adjacent but not 4-adjacent.

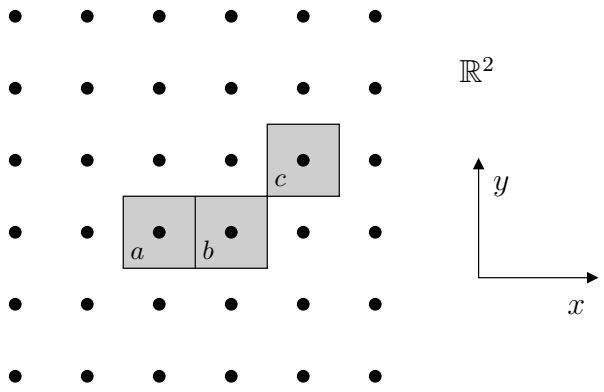


Figure 2.1: Continuous analog of three grid points.

Definition 2.1.7. Let X be any subset of D. For any p and q in X, the sequence $\langle x^{(0)}, \ldots, x^{(k)} \rangle$ of points of X is said to be a *ρ-path in X connecting p to q*, where $\rho \in \{4, 8\}$, if $x^{(0)} = p$, $x^{(k)} = q$, and $x^{(i-1)}$ is ρ-adjacent to $x^{(i)}$, where $1 \leq i \leq k$. In particular, there are ρ-paths of length zero, e.g., $\langle x \rangle$. We refer to them as *trivial paths*.

Definition 2.1.8. If there is a ρ-path in X connecting p to q then we say that *p is ρ-connected in X to q*.

For instance, if X is the set of points in Figure 2.1, then a is 4- and 8-connected in X to b and c, respectively.

Definition 2.1.9. A subset X of D is said to be a *ρ-connected component* if, for any p and q in X, the element p is ρ-connected in X to q. A ρ-connected component X of D is said to be *maximal* if X is not a proper subset of any other ρ-connected component of D.

By using the aforementioned binary relations on D and the notion of connectedness, we can extend the definition of continuous analog to pair of points and to sets of pairs of points, and characterize the continuous analog of the foreground of a binary image and its boundary:

Definition 2.1.10. Let $CA : D \times D \to \mathcal{P}(\mathbb{R}^2)$ be the function that identifies a pair (p, q) of points of D with the intersection set $CA(p) \cap CA(q)$. Note that if p and q are edge-adjacent then $CA((p, q))$ is precisely the edge (a line segment) shared by $CA(p)$ and $CA(q)$. Likewise, we let $CA : \mathcal{P}(D \times D) \to \mathcal{P}(\mathbb{R}^2)$ be the function such that, for any subset Y of $D \times D$,

$$CA(Y) = \bigcup_{(p,q)\in Y} CA((p, q)).$$

Definition 2.1.11. Let X be a nonempty subset of D. Then, we define the *digital boundary $bd(X)$ between X and X^c* as

$$bd(X) = \{(p, q) \in D \times D \mid p \in X,\ q \in X^c,\ \text{and } p \text{ and } q \text{ are edge-adjacent}\}.$$

Note that the continuous analog $CA(bd(X))$ of the digital boundary $bd(X)$ between X and X^c is precisely the (topological) boundary of $CA(X)$ in $\mathbb{R}^2$. From now on, we denote $CA(bd(X))$ by $bdCA(X)$.

Assume that the foreground X of the binary image (D, X) is a finite set, and let $\mathcal{E}$ be the set of edges corresponding to the pairs of points (p, q) of $bd(X)$. We can show that the edges of $\mathcal{E}$, along with their vertices, form a pure one-dimensional polytopal complex with empty boundary [76]. Furthermore, the underlying space of this complex, i.e., $bdCA(X)$, separates the continuous analog of the maximal

4-connected components of X and X^c, i.e., any curve connecting a point in the interior of the continuous analog of such a maximal component to its exterior must cross $bdCA(X)$ [65]. Figure 2.2 shows the continuous analog of the finite foreground X of a binary image (D, X), and the one-dimensional polytopal complex whose underlying space is $bdCA(X)$.

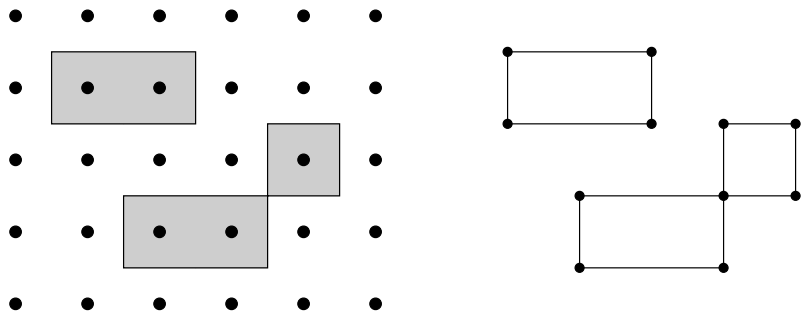


Figure 2.2: (a) Continuous analog of the (finite) foreground X of a binary image (D, X). (b) One-dimensional polytopal complex whose underlying space is $bdCA(X)$.

2.2 Mesh and Mesh Generation

Let Ω be a *polygonal region*, i.e., a closed, bounded, and connected subset of $\mathbb{R}^2$, whose boundary $bd(\Omega)$ consists of one polygon or the union of a finite number of disjoint polygons. Figure 2.3(a) shows a polyhedral region.

Definition 2.2.1. A (two-dimensional) *mesh* $\mathcal{M}$ of Ω is a polytopal complex such that $|\mathcal{M}| = \Omega$, where $|\mathcal{M}|$ is the underlying space of $\mathcal{M}$ [136]. Typically, the 2-faces of $\mathcal{M}$ are either triangles or quadrilaterals, or a combination of both. We say that $\mathcal{M}$ is a *conforming mesh* of Ω if $\mathcal{M}$ conforms to $bd(\Omega)$, i.e., if each vertex of $bd(\Omega)$ is a vertex of $\mathcal{M}$, and if each edge of $bd(\Omega)$ is the union of one or more edges of $\mathcal{M}$.

Figure 2.3(b) shows a mesh of the polygonal region in Figure 2.3(a). Now, we can formally state the *two-dimensional mesh generation problem*: given a polygonal region Ω, find a conforming mesh $\mathcal{M}$ of Ω. Henceforth, we assume that Ω is given in terms of the vertices and edges defining the boundary of Ω, $bd(\Omega)$.

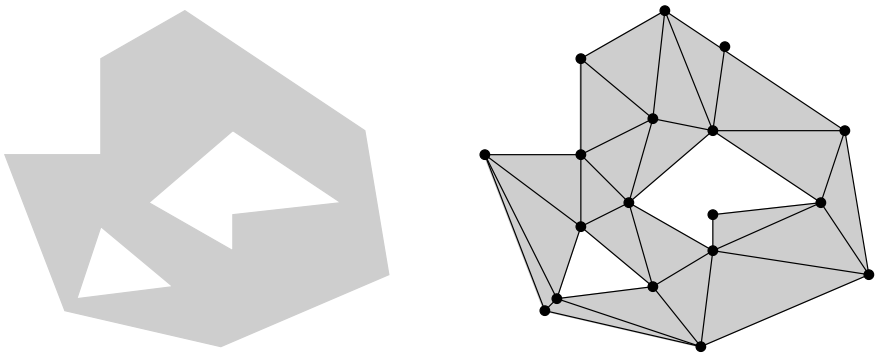


Figure 2.3: (a) A polygonal region. (b) A mesh of the polygonal region in (a).

In some applications, the input for the mesh generation problem may also include an additional set of interior vertices and edges:

Definition 2.2.2. An *interior vertex* is a vertex located in the interior of Ω. An *interior edge* is an edge whose endpoints are either two interior vertices or an interior vertex and a vertex of $bd(\Omega)$.

Whenever interior vertices and edges are present in the input problem, a conforming mesh is a mesh that conforms to $bd(\Omega)$ and to the interior vertices and edges as well. Note that the vertices and edges of $bd(\Omega)$, along with the interior vertices and edges, form a one-dimensional polytopal complex whose underlying space is a subset of Ω. Such complex is sometimes described by a planar-straight linear graph (PSLG) [8], as shown by Figure 2.4.

2.2.1 Mesh Constraints

In fact, the mesh generation problem as stated before is hardly of any practical interest. The reason is that a mesh is usually built with some application in mind (e.g., finite element analysis), and applications often require the mesh to satisfy some *mesh constraints*. These constraints are specified in terms of *mesh parameters*, such as size, shape, orientation, and number of *mesh elements*, or equivalently, of 0-, 1-, or 2-faces of the mesh. However, optimal values for mesh parameters are highly dependent on the application, and they may largely vary from one application to another [122].

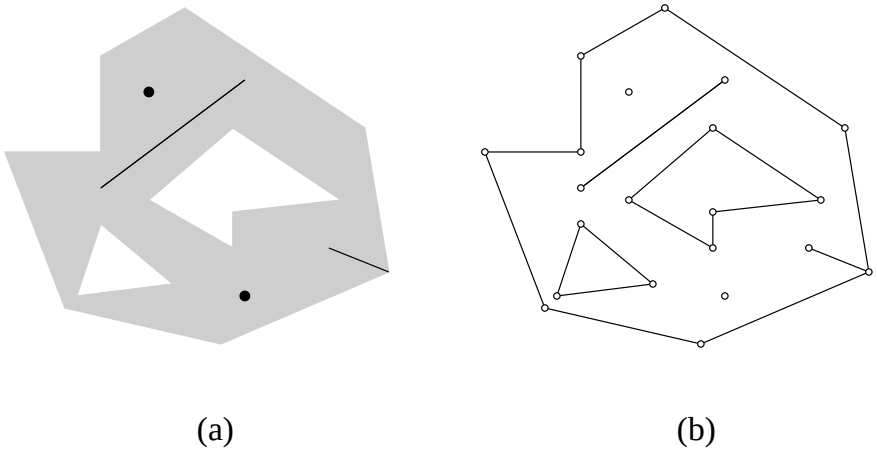


Figure 2.4: (a) A polygonal region with interior vertices and edges. (b) A PSLG describing the boundary of the polygonal region and the interior vertices and edges.

We can think of the optimal size of a mesh element as being specified by a *spacing function* $h : \Omega \to \mathbb{R}_+$, where $h(p)$ is a measure of the largest admissible size of any element that contains a point $p \in \Omega$. The function h depends on the geometry of Ω and on the numerical conditions of the application using the mesh. In practice, h can be given as an input parameter for the meshing algorithm or it can be indirectly estimated by a process that combines error estimation and size refinement of mesh elements [47].

Optimal shape of a mesh element is usually defined as the largest (or sometimes the smallest) value of some *shape metric*. Some meshing algorithms provide an input parameter which is a lower bound on a shape metric. Then, they try to generate a mesh in which all elements have "shape value" at least as large as the given lower bound. Ideally, given a sizing function and a shape metric lower bound, a meshing algorithm should try to generate a smallest possible mesh satisfying the size and shape constraints. By smallest, we mean a mesh with the smallest number of mesh elements.

2.3 Meshes from Images

Let (D, X) be a two-dimensional binary image. From now on, we assume that X is a nonempty and finite subset of D, and that the image domain D is also a *finite* orthogonal grid, i.e.,

$$D = \{p = (p_1, p_2) \in \mathbb{R}^2 \mid p = O + \delta \cdot g\},$$

where $O = (o_1, o_2) \in \mathbb{R}^2$, $\delta \in \mathbb{R}$, and $g = (g_1, g_2) \in \{0, n_1\} \times \{0, n_2\}$, for some positive integers n_1 and n_2. We also assume that if $p = (p_1, p_2) \in X$, then $p_i \neq o_i$ and $p_i \neq o_i + \delta \cdot n_i$, for all $i \in \{1, 2\}$. The latter assumption ensures that $bdCA(X)$ is still well-defined; that is, the grid D contains all points in the pairs of points of $bd(X)$.

Let Ω be the convex hull $conv(D)$ of D. From our previous assumptions, we know that Ω is a rectangle and that $bdCA(X)$ is in the interior of Ω. Now, we can state *the problem of generating a mesh from a two-dimensional binary image* as follows: Given a two-dimensional binary image (D, X) (satisfying our previous assumptions), obtain a mesh $\mathcal{M}$ of $\Omega = conv(D)$ such that $\mathcal{M}$ conforms to $bd(\Omega)$ and to the common edges (along with their vertices) of the pixels $CA(p)$ and $CA(q)$, for every (p, q) in $bd(X)$.

As we have seen before, the common edges (along with their vertices) of the pixels $CA(p)$ and $CA(q)$, for every (p, q) in $bd(X)$, form a pure one-dimensional polytopal complex $\mathcal{C}$ with empty boundary, whose underlying space is precisely $bdCA(X)$. The edges and vertices of $\mathcal{C}$ can be viewed as interior edges and vertices in the statement of the meshing problem in Section 2.2. If we augment $\mathcal{C}$ to include the vertices and edges of the rectangle Ω, we can represent the input of our problem by only one pure one-dimensional polytopal complex with empty boundary, as shown in the example in Figure 2.5.

Here, when discussing the problem of generating a mesh from a two-dimensional binary image, we use the sentence "a mesh that conforms to $bdCA(X)$" to mean a mesh that conforms to the complex $\mathcal{C}$ described above whose underlying space is $bdCA(X)$. Also, we are often interested in a constrained version of the problem, which takes into account mesh constraints such as element shape and size, and number of elements.

2.4 Our Solution and Its Contributions

There is a trivial solution for the problem presented in the previous section: Let the mesh $\mathcal{M}$ be the set of pixels of all points of D along with their edges and vertices. This trivial solution has two serious drawbacks. First, the number of quadrilaterals (i.e., pixels) is in general unnecessarily large. The reason is that the pixels have the same size. This problem can be solved by using a mesh in which elements may differ in size.

Second, the mesh $\mathcal{M}$ conforms to $bdCA(X)$. Although this is a requirement of the problem, it has been observed that meshes that conform to $bdCA(X)$ lead to finite element solutions that are less accurate than those obtained from meshes that conform to "smoother" approximations of $bdCA(X)$ [69, 70]. This is mainly due to the fact that the geometry of $bdCA(X)$ is characterized by abrupt transitions and

right angles. So, it is often the case that the input for the meshing algorithm is a modified complex, which is defined from an approximation to $bdCA(X)$, rather than $bdCA(X)$ itself.

Given a polytopal complex $\mathcal{C}$ describing the boundary of an arbitrary polygonal region Ω and an additional set of interior vertices and edges, there are several known algorithms that can generate a triangular mesh from $\mathcal{C}$ [47]. Some of them are also guaranteed to generate meshes in which all triangles are well-shaped [32, 116, 121, 95], i.e., triangles with no small angle, except for triangles containing small input angles and possibly triangles close to small input angles. An *input angle* is an angle defined by the edges of $\mathcal{C}$.

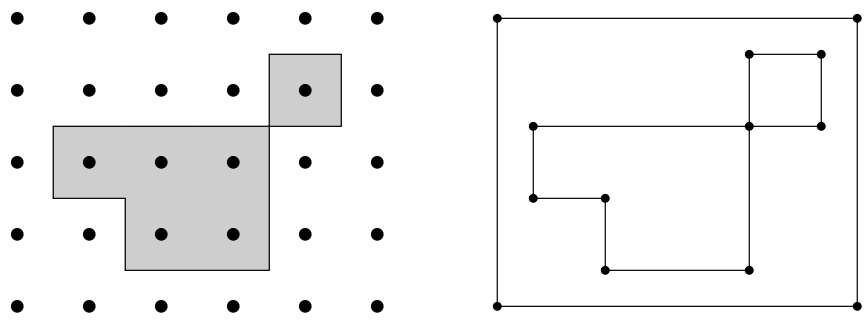


Figure 2.5: (a) Image grid and the continuous analog of the foreground pixels. (b) Polytopal complex describing the input for the meshing problem corresponding to the image in (a).

Besides being able to produce well-shaped triangles, some of these algorithms [116, 121, 95] produce meshes that are both *size nearly-optimal* and nicely graded. By size nearly-optimal, we mean that, for a given lower bound on the smallest angle, the algorithm produces a mesh whose number of triangles is at most a constant factor larger than the number of triangles of any triangular mesh that meets the same angle bound. Furthermore, the constant does not depend on the input complex, but on

the angle bound. By nicely graded, we mean that triangle sizes vary quickly over short distances and that they are adapted to the local geometric details of the mesh domain.

Despite the existence of meshing algorithms for generating triangular meshes with provably good quality guarantees, there are applications in which quadrilateral meshes may be more desirable than triangular ones. For instance, scattered data interpolation and approximation [78] and finite element analysis [3, 90]. Unfortunately, little is known about theoretical properties for generating quadrilateral meshes with provably good quality guarantees. A few algorithms exist to generate quadrilateral meshes of bounded size [44, 101, 9, 112, 17], bounded largest angle [9], and controlled grading and directionality [12, 123, 137].

The algorithm in [112] may generate non-convex quadrilaterals, which makes it unsuitable for various applications, including the finite element-based ones. Also, the theoretical guarantees of some of these algorithms are only valid to convex polygonal domains [101] or very small simple polygons [17]. Finally, to our best knowledge, there are no known algorithms to generate quadrilateral meshes from a given arbitrary complex that are provably guaranteed to simultaneously meet shape and size quality criteria.

In Chapter 3, we provide a new algorithm for converting triangular meshes of pure one-dimensional polytopal complexes with empty boundary into quadrilateral meshes of the same complex [114]. Our algorithm is guaranteed to produce a bounded number of quadrilaterals with respect to the number of triangles of the input triangular mesh and the number of connected components of its dual graph. This bound is tighter than previous known bounds for a similar algorithm [44]. Although our algorithm does not ensure that the output quadrilaterals are "well-shaped", they are all *strictly convex*, i.e., each internal angle is greater than 0^o and smaller than 180^o. Some of our experiments show that quadrilateral shape can be further improved by standard post-processing techniques.

We used our algorithm for generating quadrilateral meshes from two-dimensional images of the human brain. We ran an experiment in which a finite element based algorithm for image registration ([53]) was given meshes generated by our algorithm and their triangular counterparts. We then compared the accuracy of the registration based on our quadrilateral meshes with the one based on their triangular counterparts. Although the triangular meshes were generated by provably good triangular meshing algorithms, the quadrilateral meshes provided slightly inferior results (see [125, 113]).

Chapter 3

Quadrilateral Meshes of Bounded Size

This chapter describes a new algorithm for converting a triangular mesh of a polygonal region into a quadrilateral mesh of the same region. The polygonal region may contain interior vertices and edges. We apply our algorithm to the problem of generating meshes from a two-dimensional binary image, and we provide examples of its use on real data.

3.1 Background and Related Work

Throughout this chapter, we let Ω denote a polygonal region. We refer to the vertices and edges of $bd(\Omega)$ as vertices and edges of Ω. Recall that $bd(\Omega)$ consists of $1 + k$ disjoint polygonal curves, with $k \geq 0$. Whenever $k > 0$, we say that Ω is a polygonal region with k holes.

The problem of generating a quadrilateral mesh of a polygonal region Ω is more complex than that of producing a triangular mesh. In fact, if we require the set of vertices of the mesh to be the set of vertices of Ω, then a triangular mesh can always be obtained but it may not be possible to obtain a quadrilateral one. Hence,

additional vertices, called *Steiner points*, may be necessary in order to quadrangulate Ω.

More specifically, if $bd(\Omega)$ is a simple convex polygon and no interior vertex and edge is given, then a quadrilateral mesh of Ω with no Steiner points can be obtained if and only if the number of vertices of Ω is even [126]. If $bd(\Omega)$ is not convex, then Ω may require Steiner points in order to be quadrangulated. Furthermore, Lubiw [88] has shown that, if Ω is a polygonal region with k holes, where $k > 0$, the problem of deciding whether Ω can be decomposed into convex quadrilaterals, without adding Steiner points, is **NP**-complete.

As we pointed out in Section 2.4, in practice, we are often required to generate strictly convex quadrilateral meshes of bounded size and whose quadrilaterals are well-shaped. However, unlike the case of triangular meshes, very little is known about theoretical properties to generate provably good quality quadrilateral meshes. These facts have led several researchers to adopt an *indirect approach* to produce quadrilateral meshes [107]: the input domain is first triangulated and then the triangular mesh is converted into a quadrilateral one [44, 72, 112, 123, 106, 137]. This approach relies on the premise that a good quality quadrilateral mesh may be more easily generated from an existing good quality triangular mesh of the same input domain.

Let $\mathcal{T}$ be a triangular mesh of a polygonal region Ω. A very simple algorithm for converting such a triangular mesh into a strictly convex quadrilateral mesh was proposed by de Berg [44]: place a Steiner point in the interior of each edge and each triangle of the triangular mesh, and then connect the Steiner point in the interior of each triangle to the three Steiner points on its edges. Fig. 3.1 illustrates de Berg's algorithm. If the Steiner points are placed carefully, it is always possible to obtain a strictly convex quadrilateral mesh. If t is the number of triangles of $\mathcal{T}$, then de Berg's algorithm runs in $\mathcal{O}(t)$ time, produces $3t$ quadrilaterals, and inserts exactly $5m + 5k - 5 - 2m_b$ Steiner points, where m is the number of vertices of $\mathcal{T}$, k is

the number of holes of Ω, and m_b is the number of vertices of $\mathcal{T}$ on $bd(\Omega)$. This algorithm, however, is not practical, as it produces too many quadrilaterals on a large input triangular mesh.

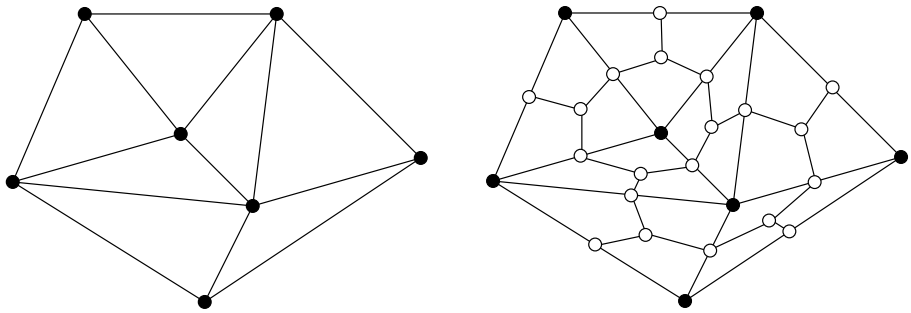


Figure 3.1: Example of de Berg's algorithm.

Everett, Lenhart, Overmars, Shermer and Urrutia [44] introduced another linear time algorithm to convert triangular meshes into strictly convex quadrilateral ones. Their algorithm also inserts a Steiner point in the interior of each edge of the triangular mesh, but only some of the mesh triangles contain Steiner points in their interiors. This algorithm generates at most $\lfloor \frac{8t}{3} \rfloor$ quadrilaterals and uses the same number of Steiner points as de Berg's. However, the number of output quadrilaterals may still be prohibitive in practice. An interesting feature of this algorithm, which is also present in de Berg's algorithm, is the preservation of the size grading of the triangular mesh.

Ramaswami, Ramos and Toussaint [112] presented a linear time and space algorithm to convert triangular meshes into quadrilateral ones that considerably improves upon the bounds on mesh size provided by the two previous algorithms. However, the quadrilateral meshes are not necessarily convex. Johnston, Sullivan, and Kwasnik [72] also apply the indirect approach to give an algorithm that uses several heuristics to obtain a strictly convex quadrilateral mesh from a triangular mesh. Their algorithm runs in $\mathcal{O}(t^2)$ time, and selectively combines adjacent triangles to obtain quadrilaterals. However, it is not clear from the description of the

algorithm in [72] that the heuristic procedures are always successful in producing a mesh consisting of quadrilaterals only.

Shimada, Liao and Itoh [123] proposed an algorithm for generating quadrilateral meshes that takes into account mesh directionality and grading, and quadrilateral shape. First, the problem domain is filled with square cells whose size is controlled by a user-defined, sizing function. Next, the orientation of each cell is adjusted by a physically-based relaxation process and a user-defined vector field that specifies directionality over the problem domain. Then, mesh vertices are placed at the center of every cell and connected to generate a triangular mesh of the entire domain. Finally, the triangular mesh is converted into a strictly convex quadrilateral mesh. Later, Viswanath, Shimada and Itoh [137] modified the algorithm in [123] by using rectangular cells, rather than square cells, to generate *anisotropic* quadrilateral meshes [47].

Owen, Staten, Cannan, and Saigal [106] presented another quadrilateral meshing algorithm that takes into account directionality and element shape, and it also preserves mesh grading. It converts a triangular mesh into a strictly convex quadrilateral one using advancing fronts initially defined by the boundary edges of the input mesh. Quadrilaterals are generated by combining and transforming triangles as the fronts move from the boundary to the interior of the input mesh. Local smoothing and topological improvements, commonly performed as post-processing steps, are part of the conversion process. One limitation of this method is that directionality cannot be arbitrarily specified as in [123, 137]. Although the algorithms in [106, 123, 137] do not provide any provable bounds on mesh size or mesh element shape, they have successfully been used in practice to generate good quality quadrilateral meshes of several complex domains.

Our quadrilateral meshing algorithm improves upon the bounds on mesh size provided by the algorithms of de Berg and Everett, Lenhart, Overmars, Shermer and Urrutia [44]. This improvement makes it possible to use our algorithm in practical

applications (e.g., see [125, 114]). Our algorithm is also simpler, and very likely faster, than the algorithms in [106, 123, 137]. Furthermore, our algorithm can also deal with triangular meshes of polygonal domains with interior vertices and edges, which has not been reported to be possible by the algorithms in [106, 123, 137]. On the other hand, algorithms in [106, 123, 137] are more likely to generate quadrilateral meshes in which overall element shape is better for finite element analysis than the overall element shape in the quadrilateral meshes generated by our algorithm. Fortunately, we can further improve the overall shape of the quadrilaterals generated by our algorithm, at the expense of runtime, by using standard post-processing techniques, as we shall see later.

3.2 Unconstrained Quadrilateral Meshes

For the sake of clarity, we split the description of our algorithm into two parts. In this section, we present an algorithm for converting a triangular mesh of a polygonal region without interior vertices and edges into a quadrilateral mesh. We refer to such a triangular (resp. quadrilateral) mesh as an *unconstrained* triangular (resp. quadrilateral) mesh. In the next section, we extend our algorithm to converting triangular meshes of polygonal regions with interior vertices and edges into quadrilateral ones. Likewise, we refer to these meshes as *constrained* triangular (resp. quadrilateral) meshes.

Our algorithm for handling unconstrained triangular meshes allows edges of the triangular mesh to be deleted, but not vertices. To construct the quadrilateral mesh, new vertices (i.e., Steiner points) may be inserted along with new edges between Steiner points and/or vertices of the input triangular mesh. We show that the mesh produced by our algorithm has small, bounded size and it consists of strictly convex quadrilaterals only. In particular, we show that if the input triangular mesh has t triangles, then our algorithm produces a strictly convex quadrilateral mesh with at

most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals and it adds at most $t+2$ Steiner points into the resulting mesh.

3.2.1 The Algorithm

The strategy used by our algorithm is to quadrangulate a small triangulated region of the input triangular mesh at a time until the entire input mesh is converted into a quadrilateral one. The domain of each such a triangulated region is a small and simple polygonal region (one with 7 or fewer vertices and no holes). Our algorithm builds a spanning tree of the dual graph of the entire triangular mesh, and uses a procedure to traverse and prune this spanning tree in a bottom-up fashion, to systematically groups triangles together in order to define and quadrangulate the small regions.

Let $\mathcal{T}$ be the input (unconstrained) triangular mesh of a polygonal region Ω, and let t be the number of triangles of $\mathcal{T}$. Our algorithm starts by building a rooted spanning tree T of the dual graph G of $\mathcal{T}$.

Definition 3.2.1. The *dual graph* of $\mathcal{T}$ is the graph that contains a node for every triangle of $\mathcal{T}$ and an edge between two nodes if and only if the corresponding triangles share an edge.

Figure 3.2(b) shows the dual graph of a triangular mesh of the polygonal region shown in Figure 3.2(a). A rooted spanning tree T of G is built as a breadth-first search (BFS) tree. The root of T is any node corresponding to a triangle containing a boundary edge of $\mathcal{T}$. Since every node of G can have degree at most 3, we have that T is a binary tree. Figure 3.2(c) shows such a spanning tree.

After constructing T, the algorithm builds the set V_l of all nodes of T at level l, for every $l \in \{0, 1, \ldots, d\}$, where d is the depth of T. The root node of T is the singleton node at level 0. Next, the algorithm visits the nodes of T one level at a time and in decreasing order of depth by processing the sets $V_d, V_{d-1}, \ldots, V_0$ in this order.

Let $par(v)$ denotes the parent of $v \in V$, $sib(v)$ the sibling of v, and $ele(v)$ the triangle of $\mathcal{T}$ corresponding to v. Note that $ele(v)$ and $ele(par(v))$ necessarily share an edge of $\mathcal{T}$. When visiting a node $v \in V_l$,$1 < l \leq d$, the algorithm considers the subtree rooted at either $par(v)$ or $par(par(v))$ (the nodes of V_0 and V_1 are handled separately at the end of the algorithm). We denote this subtree by T_v and its root by r_v. Let G_v denote the subgraph of G induced by T_v. As we shall see later, the subgraph G_v corresponds to a triangulated polygonal region $\mathcal{T}_v$ of $\mathcal{T}$ consisting of $4, 5, 6$, or 7 vertices. Our algorithm converts $\mathcal{T}_v$ into a partial or complete quadrilateral mesh of the domain of $\mathcal{T}_v$.

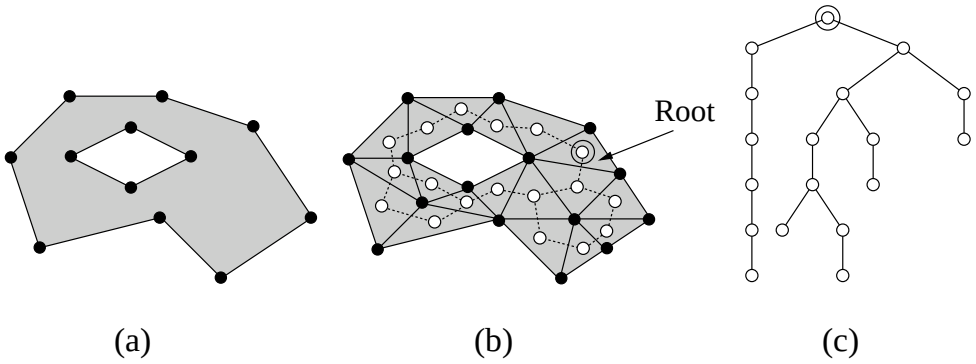


Figure 3.2: (a) A polygonal region Ω. (b) A triangular mesh of Ω and its dual graph. Vertices of the dual graph are marked white, and its edges are shown as dotted lines. (c) A rooted (BFS) spanning tree for the dual graph in (b).

During the conversion of $\mathcal{T}_v$ into a partial or complete quadrilateral mesh of its domain, Steiner points may be added to the interior and boundary of $\mathcal{T}_v$. If the result is a complete quadrilateral mesh of the domain of $\mathcal{T}_v$, the entire subtree T_v is eliminated from T. If the quadrilateral mesh is not complete, there will be only one leftover triangle inside the domain of $\mathcal{T}_v$. The root node r_v of T_v now represents this triangle and the remaining nodes of T_v are eliminated from T. Figure 3.3 illustrates both cases for a triangulated polygonal region $\mathcal{T}_v$ of $\mathcal{T}$ consisting of 5 vertices on the boundary.

After converting $\mathcal{T}_v$ into a partial or complete quadrilateral mesh, the node v is eliminated from T and V_l. Other nodes of T_v may or may not be eliminated from T and $V_l \cup V_{l-1} \cup V_{l-2}$. In either case, we show that all nodes of V_l are eliminated from T and V_l by algorithm, and hence the depth of T decreases by at least one. As a result, when the nodes of V_{l-1} are visited during the next step of the algorithm, they are all leaf nodes of (the pruned) T. The sets V_0 and V_1 are handled in a similar way as special cases in the last step, so that after they are processed, the spanning tree T is empty and the triangular mesh $\mathcal{T}$ has been converted into a strictly convex quadrilateral mesh. We will show that, except at the last step, for every two nodes eliminated from T, at most three quadrilaterals are created by using at most two Steiner points.

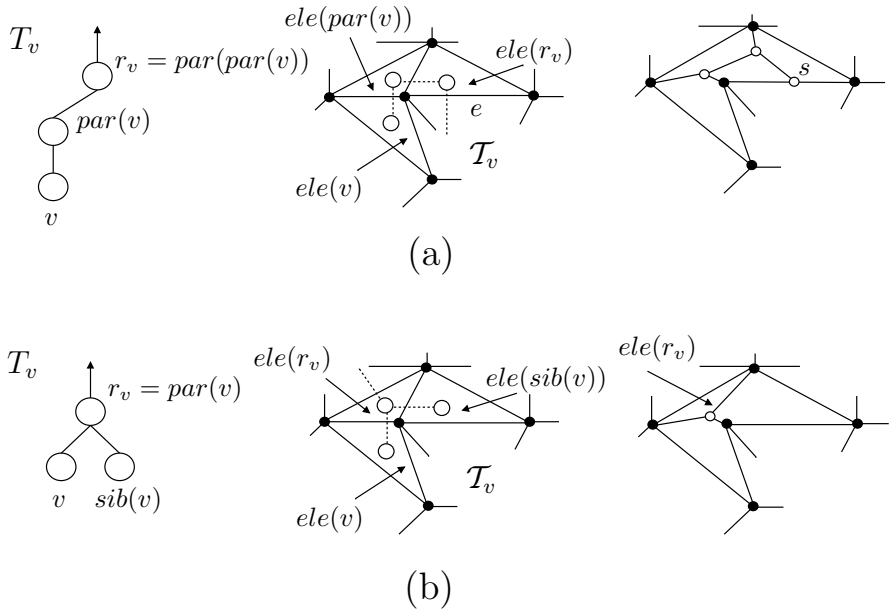


Figure 3.3: (a) Complete quadrilateral mesh of the domain of $\mathcal{T}_v$. (b) Partial quadrilateral mesh of the domain of $\mathcal{T}_v$.

Our algorithm has five steps. For each $l = d, d-1, \ldots, 2$, the algorithm repeatedly executes steps 1 to 4, in this order. The last step, step 5, is executed only once to process the sets V_1 and V_0, unless these sets are already empty after the algorithm

processes sets $V_d, V_{d-1}, \ldots, V_2$. Before describing the five steps, we discuss two special situations:

1. When processing T_v, the algorithm may place a Steiner point s on the edge e between $ele(r_v)$ and $ele(par(r_v))$, as shown in Fig. 3.3(a). If so, the triangle $ele(par(r_v))$ becomes a *degenerate quadrilateral*. Furthermore, this degenerate quadrilateral $ele(par(r_v))$ may further become a *degenerate pentagon* if the algorithm happens to add another Steiner point to it on the edge shared with $ele(sib(r_v))$ (see Figure 3.4).

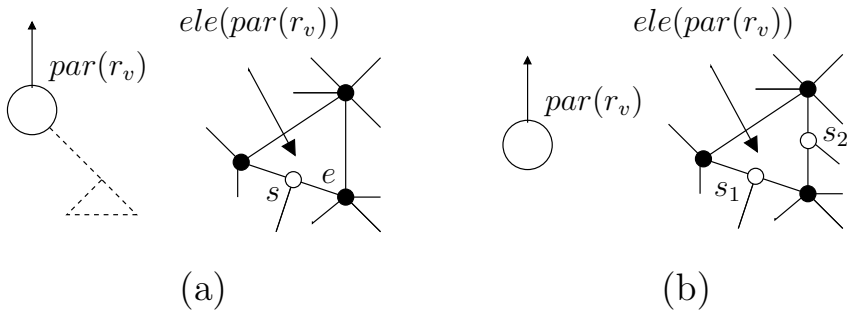


Figure 3.4: (a) Degenerate quadrilateral. (b) Degenerate pentagon.

Since T_v gets eliminated from T when a Steiner point is placed on the common edge e of r_v and $par(r_v)$, degenerate pentagons are leaves of T, and degenerate quadrilaterals are either leaves or internal nodes of degree 2. Furthermore, since all quadrilaterals in the mesh constructed so far are strictly convex, there must be an edge of the quadrilateral mesh incident on s and lying outside $ele(par(r_v))$. We can slightly perturb s along this edge to eliminate the degeneracy of $ele(par(r_v))$ without destroying the strict convexity of other quadrilaterals incident to s (see Figure 3.5(a)). We describe later how our algorithm handles degenerate pentagons.

2. When T_v is a subtree of three nodes, v, $r_v = par(v)$ and $sib(v)$, containing v and there is a cross-edge between v and $sib(v)$ in G_v, the triangulated polygonal

region $\mathcal{T}_v$ has three vertices of $\mathcal{T}$ on its boundary and a vertex of $\mathcal{T}$ in its interior (see Figure 3.6). In this case, the algorithm eliminates v and $sib(v)$ from T, so that $par(v)$ now represents the *non-empty triangle* Δ with an interior point (see Figure 3.6). Note that if v is at level l, then the node corresponding to Δ is a leaf at level $l-1$.

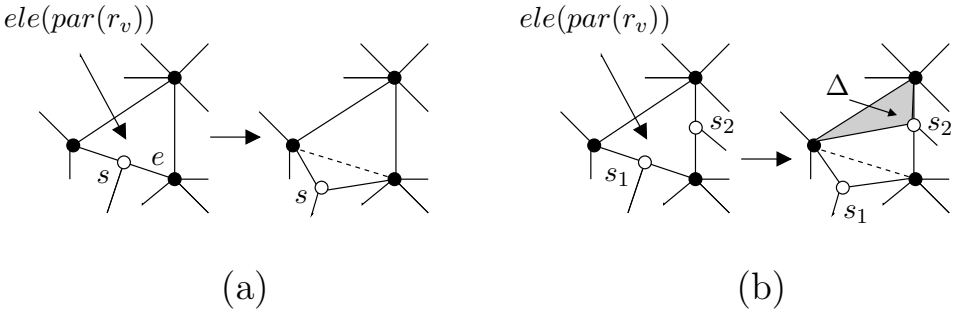


Figure 3.5: Elimination of degenerate quadrilaterals and pentagons.

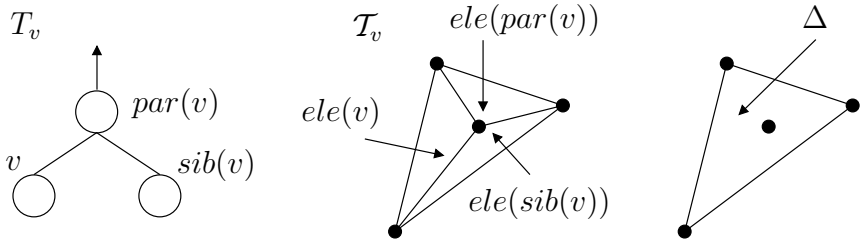


Figure 3.6: The non-empty triangle Δ.

We now describe the steps taken by our algorithm to process the set V_l ($1 < l \leq d$), where l is the current deepest level of T. We first eliminate all leaves v of T such that $ele(v)$ is a degenerate quadrilateral, degenerate pentagon, or a non-empty triangle. The first two types of leaves will exist only at levels l, $l-1$, or $l-2$, and the third type will exist only at level l, as each node u of level $l+1$ processed during the previous step had T_u rooted at either $par(u)$ or $par(par(u))$. During the course of our description, we refer to several lemmas concerning quadrilateral meshes

of small polygonal regions without holes. These lemmas are stated and proved in Appendix B.

Let l be the deepest level of T, and let v be a node (leaf) of T at level l. When visiting v, our algorithm considers the subtree T_v of T rooted at either $par(v)$ or $par(par(v))$. Since the spanning tree T is a result of a breadth-first search (BFS) on the dual graph G of $\mathcal{T}$, the subtree T_v rooted at $par(v)$ must be one of subtrees (1) and (2) shown in Figure 3.7 (up to isomorphism), while the subtree T_v rooted at $par(par(v))$ must be one of subtrees (3) to (10) shown in Figure 3.7 (up to isomorphism).

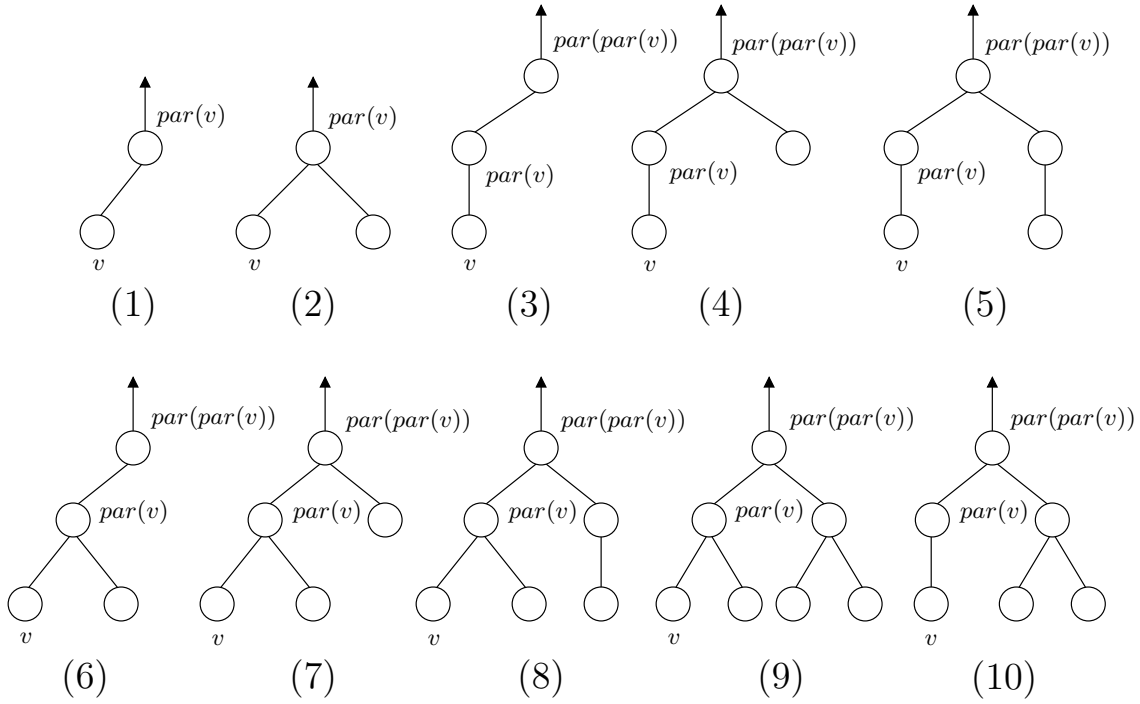


Figure 3.7: All possibilities for a subtree T_v of T rooted at $par(v)$ or $par(par(v))$, where v is a leaf of T at its deepest level.

Let l be the current deepest level of T, and assume that $l \geq 2$. Step 1 eliminates all leaves v from T such that v is in $v \in V_l \cup V_{l-1} \cup V_{l-2}$ and $ele(v)$ is a degenerate quadrilateral. Likewise, step 2 eliminates all leaves v from T such that v is in

$v \in V_l \cup V_{l-1} \cup V_{l-2}$ and $ele(v)$ is a degenerate pentagon. Both steps are described in details below:

Step 1. *Eliminate all $v \in V_l \cup V_{l-1} \cup V_{l-2}$ such that v is a leaf and $ele(v)$ is a degenerate quadrilateral.* Let s be the Steiner point of $ele(v)$, and let e_s be the edge of the quadrilateral mesh (constructed thus far) incident on s. Convert $ele(v)$ into a strictly convex quadrilateral by perturbing s along the edge e_s, as shown in Figure 3.5(a). Remove v from T and from $V_l \cup V_{l-1} \cup V_{l-2}$.

Step 2. *Eliminate all $v \in V_l \cup V_{l-1} \cup V_{l-2}$ such that $ele(v)$ is a degenerate pentagon.* Let s_1 and s_2 be the two Steiner points of $ele(v)$ and let e be the shared edge of $ele(v)$ and $ele(par(v))$. It is straightforward to convert $ele(v)$ into a strictly convex quadrilateral and a leftover triangle Δ, as shown in Figure 3.5(b). Now, the node v represents the leftover triangle, i.e., $ele(v) = \Delta$.

After steps 1 and 2 are carried out, every element v of V_l must correspond to either a triangle or a non-empty triangle, and $par(v) \in V_{l-1}$ must be either a triangle or a degenerate quadrilateral. The next step eliminates all nodes v from V_l such that $ele(v)$ is a non-empty triangle. Assume that $v \in V_l$ corresponds to a non-empty triangle and consider the subtree T_v containing v and rooted at $par(v)$. Up to isomorphism, the subtree T_v must be one of subtrees (1) and (2) in Figure 3.7. If T_v is isomorphic to subtree (1), then $par(v)$ is either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (2), then $par(v)$ is a triangle and $sib(v)$, the sibling of v, is either a triangle or a non-empty triangle. So, there are four cases to be considered (cases 3(a)-(d) below) in order to eliminate v from V_l when $ele(v)$ is a non-empty triangle.

Step 3. *Eliminate all $v \in V_l$ such that $ele(v)$ is a non-empty triangle.* Note that v corresponds to three nodes of the original spanning tree T. Let T_v be the subtree of T rooted at $par(v)$. We consider the following sub-steps (refer to Figure 3.8 and Figure 3.9):

Case 3(a). *Node $par(v)$ has degree 2, and $ele(par(v))$ is a degenerate quadrilateral.* Let s be the Steiner point of $ele(par(v))$. By connecting s to the node of $ele(par(v))$ that is not incident to the edge of $ele(par(v))$ containing s, the triangulated region $\mathcal{T}_v$ can be decomposed into a triangle Δ and a quadrilateral with a point in its interior (see Figure 3.8(a)). By Lemma B.1.2, the latter can be quadrangulated into five convex quads with three Steiner points in its interior. Carry out the appropriate decomposition and then remove v from T and V_l. Node $par(v)$ now corresponds to the triangle Δ.

Case 3(b). *Node $par(v)$ has degree 2, and $ele(par(v))$ is a triangle.* Note that the domain of $\mathcal{T}_v$ is a quadrilateral and this quadrilateral contains a vertex of $\mathcal{T}_v$ in its interior (see Figure 3.8(b)). Again, by Lemma B.1.2, the region can be quadrangulated into five convex quadrilaterals with three Steiner points in its interior. Carry out this decomposition and then remove v and $par(v)$ from T and from their corresponding node sets.

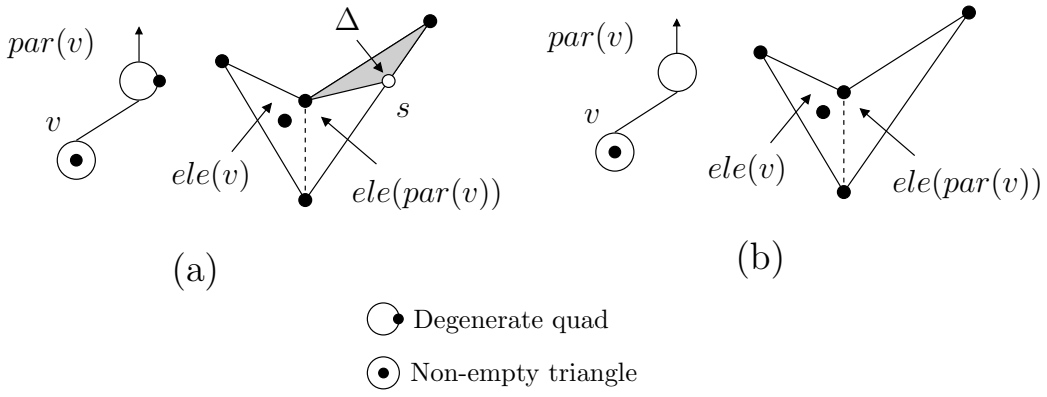


Figure 3.8: Cases 3(a) and 3(b) of Step 3.

Case 3(c). *Node $par(v)$ has degree 3, and $ele(sib(v))$ is a triangle.* If $G_v = T_v$, then the domain of $\mathcal{T}_v$ is a pentagon and this pentagon contains a vertex of $\mathcal{T}_v$ in its interior (see Figure 3.9(a)). Then by Lemma B.2.2, this region can be

decomposed into at most six convex quads and one triangle Δ by adding at most four Steiner points to its interior. If G_v contains a cross-edge between v and $sib(v)$, then $ele(v)$ and $ele(sib(v))$ form a quadrilateral with a point inside (see Figure 3.9(a)), and again Lemma B.1.2 is applied. In either case, carry out the appropriate decomposition and then remove v and $sib(v)$ from T and from their corresponding node sets. In the former case, the vertex r_v is made correspond to triangle Δ.

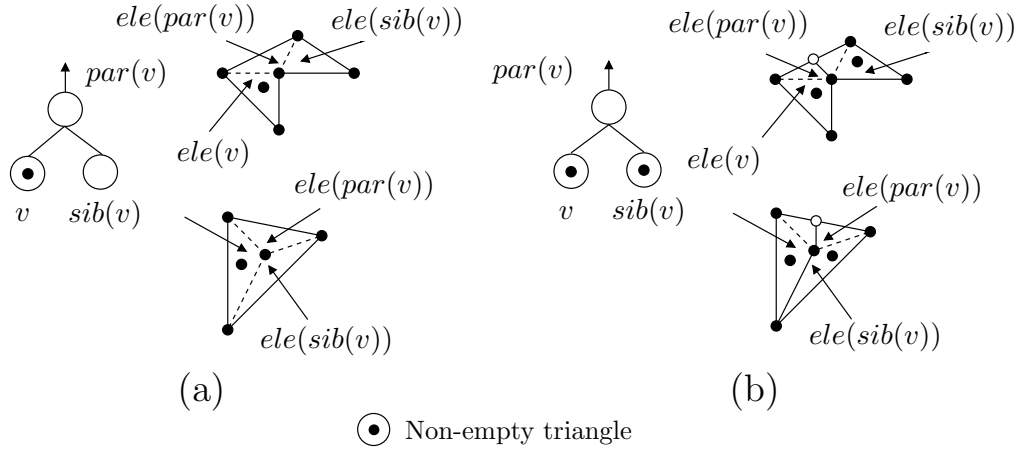


Figure 3.9: Cases 3(c) and 3(d) of Step 3.

Case 3(d). *Node $par(v)$ has degree 3, and $ele(sib(v))$ is a non-empty triangle.* If $G_v = T_v$, then the domain of $\mathcal{T}_v$ is a pentagon and this pentagon contains two vertices of $\mathcal{T}_v$ in its interior. If $G_v \neq T_v$, i.e., if G_v contains a cross-edge between v and $sib(v)$, then the domain of $\mathcal{T}_v$ is a triangle and this triangle contains three vertices of $\mathcal{T}_v$ in its interior. In either case, the triangulated region $\mathcal{T}_v$ can be decomposed into two quadrilaterals as follows: add a Steiner point on the edge shared by $ele(r_v)$ and $ele(par(r_v))$ and connect it to the vertex of $ele(r_v)$ that is not incident to the edge shared by $ele(r_v)$ and $ele(par(r_v))$ (see Figure 3.9(b)). By Lemma B.1.2, each quadrilateral can be decomposed

into five convex quadrilaterals using three Steiner points. Next, remove all nodes of T_v from T and from their corresponding node sets. So, a total of seven nodes were eliminated and ten convex quadrilaterals were created using seven Steiner points. Note that $ele(par(r_v))$ will be a degenerate quadrilateral or pentagon in the next step of the algorithm.

After steps 1, 2 and 3 are carried out, for every node $v \in V_l$, its corresponding element $ele(v)$ in $\mathcal{T}$ is a triangle, and the corresponding element $ele(par(v))$ of its parent $par(v) \in V_{l-1}$ (if any) is either a triangle or a degenerate quadrilateral. The algorithm proceeds by performing two more steps: step 4 and step 5. These steps eliminate all remaining nodes of V_l. Step 4 is carried out if and only if $l \geq 3$, i.e., if and only if the spanning tree T is currently a tree with at least three levels. In the remaining description of our algorithm, assume that $l \geq 3$ after steps 1, 2, and 3 are carried out.

Let v be any node of V_l. If $par(v)$ exists, then consider the subtree T_v containing v and rooted at $par(v)$. Up to isomorphism, the subtree T_v must be one of subtrees (1) and (2) in Figure 3.7. If T_v is isomorphic to subtree (1), then $ele(par(v))$ is either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (2), or equivalently, if $par(v)$ has degree 3, then $par(v)$ is a triangle and $sib(v)$, the sibling of v, must be a triangle, as no vertex in V_l can correspond to a non-empty triangle. Our algorithm considers T_v if and only if T_v is isomorphic to subtree (1) and $ele(par(v))$ is a degenerate quadrilateral (step 4(a)), or T_v is isomorphic to subtree (2) (step 4(b)). Whenever T_v is isomorphic to subtree (1) and $ele(par(v))$ is a triangle, the algorithm considers the subtree T_v containing v and rooted at $par(par(v))$ (step 4(c)). Up to isomorphism, the subtree T_v rooted at $par(par(v))$ is one of subtrees (3), (4), (5), and (10) in Figure 3.7.

Step 4. *Eliminate all remaining nodes $v \in V_l$.* We split this step into three cases, 4(a), 4(b), and 4(c), each of which corresponds to a distinct subtree in Figure 3.7.

Case 4(a): *Eliminate all $v \in V_l$ such that $ele(par(v))$ is a degenerate quadrilateral.* Let T_v be the subtree of T rooted at $r_v = par(v)$. Perturb the Steiner point s of $ele(r_v)$ along the edge incident to it. A Steiner point s' placed in $ele(r_v)$ decomposes the region $\mathcal{T}_v$ into two convex quadrilaterals and a triangle Δ adjacent to the edge shared by $ele(r_v)$ and $ele(par(r_v))$ (see Figure 3.10(a)). Remove v from T and V_l, and let r_v now represent triangle Δ.

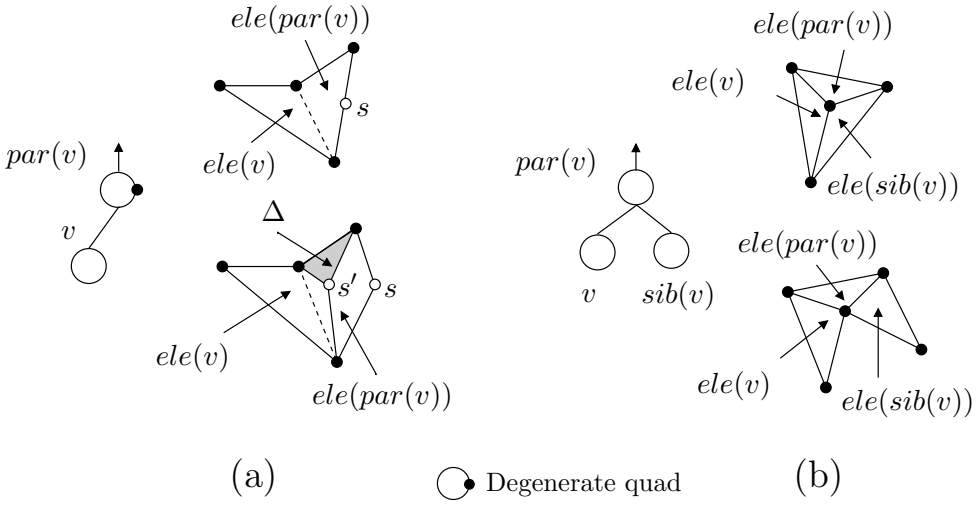


Figure 3.10: Cases 4(a) and 4(b) of Step 4.

Case 4(b). *Eliminate all $v \in V_l$ such that $par(v)$ is a node of degree 3.* Again, let T_v be the subtree of T rooted at $r_v = par(v)$ and refer to Figure 3.10(b). If G_v contains an edge between the nodes v and $sib(v)$, remove v and $sib(v)$ from T_v. Node $par(v)$ is now the non-empty triangle corresponding to the boundary of $\mathcal{T}_v$, with the fourth vertex of $\mathcal{T}_v$ in the interior of its domain (see Figure 3.6). If there is no cross-edge between v and $sib(v)$, then by Lemma B.2.1, the domain of $\mathcal{T}_v$ can be subdivided into two convex quadrilaterals and one triangle Δ (adjacent to the edge shared by $ele(r_v)$ and $ele(par(r_v))$) by adding one Steiner point. Carry out the appropriate decomposition and then remove v

and $sib(v)$ from T and the corresponding node sets. Node r_v now represents triangle Δ.

If $ele(par(v))$ is not a degenerate quadrilateral and $par(v)$ is not a node of degree 3, the algorithm considers the subtree T_v containing v and rooted at $par(par(v))$. As we mentioned before, up to isomorphism, the subtree T_v rooted at $par(par(v))$ is one of subtrees (3), (4), (5), and (10) in Figure 3.7. However, if we execute steps 4(a) and 4(b) for all vertices v of V_l *before* we consider step 4(c), no subtree T_v can be isomorphic to subtree (10) in Figure 3.7. This is because after steps 4(a) and 4(b) are executed for all $v \in V_l$, no v will have a parent, $par(v)$, of degree 3. For the very same reason, T_v cannot be isomorphic to subtrees (6), (7), (8) and (9) either. So, we can focus our attention on the cases in which T_v is isomorphic to the subtrees (3), (4), and (5) only.

If T_v is isomorphic to subtree (3), or equivalently, if node $par(v)$ has degree 2, then $par(par(v))$ can be either a triangle or a degenerate quadrilateral. If T_v is isomorphic to subtree (4), or equivalently, if node $par(par(v))$ has degree 3 and node $sib(par(v))$, the sibling of $par(v)$, is a leaf of T_v, then $sib(par(v))$ is either a triangle or a non-empty triangle previously created by step 4(b). If T_v is not isomorphic to subtree (3) nor subtree (4), then it is isomorphic to subtree (5), or equivalently, if $par(par(v))$ has degree 3 and node $sib(par(v))$ has degree 3. All cases are handled by step 4(c) below:

Case 4(c). *Eliminate all $v \in V_l$ such that $par(v)$ is a node of degree 2.* Let T_v be the subtree of T rooted at $r_v = par(v)$. If the domain of $\mathcal{T}_v$ is already a strictly convex quadrilateral, simply eliminate the common edge shared by $ele(v)$ and $ele(par(v))$, and then remove v and $par(v)$ from T and their corresponding node sets. Otherwise, let T_v be the subtree of T rooted at $r_v = par(par(v))$ and consider the following five sub-cases:

Sub-Case 4(c)(i). *Element $ele(r_v)$ is a degenerate quadrilateral.* Since T is a

BFS tree, we have that $G_v = T_v$. Perturb the Steiner point of $ele(r_v)$ along the edge incident to it so that the domain of $\mathcal{T}_v$ is now a hexagon (see Figure 3.11(a)). By Lemma B.1.1, this region can be subdivided into at most four convex quadrilaterals by using at most three Steiner points in its interior. Carry out this decomposition and then eliminate all the nodes of T_v from T and from their corresponding node sets.

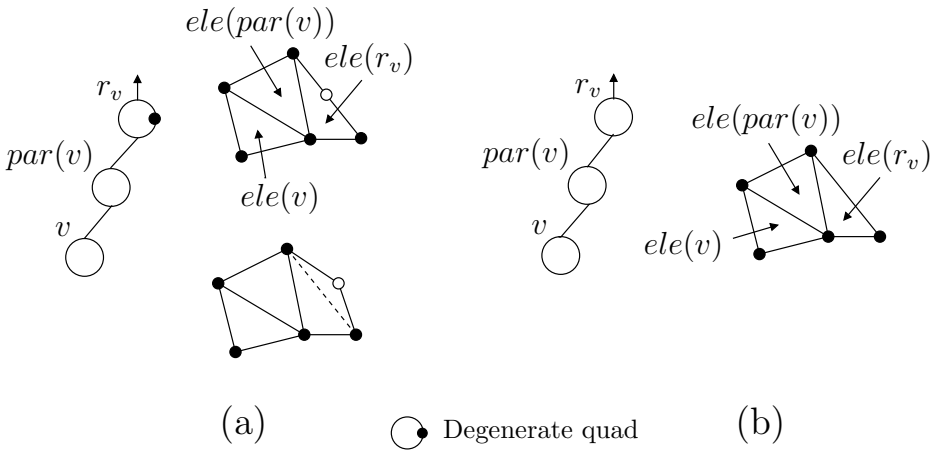


Figure 3.11: Sub-cases 4(c)(i) and 4(c)(ii) of step 4(c).

Sub-Case 4(c)(ii). *Node r_v has degree 2 and $ele(r_v)$ is a triangle.* Once again $G_v = T_v$ for this case. Therefore, the domain of $\mathcal{T}_v$ is a pentagon (see Figure 3.11(b)). Apply Lemma B.2.1. The common edge of $ele(r_v)$ and $ele(par(r_v))$ is designated as the "outgoing" edge. There are two possible outcomes from applying Lemma B.2.1. First, the domain of $\mathcal{T}_v$ is subdivided into three convex quadrilaterals and one triangle Δ adjacent to the outgoing edge by using two Steiner points. Second, the domain of $\mathcal{T}_v$ is subdivided into four convex quadrilaterals by using three Steiner points, one of which lies on the outgoing edge. In the first situation, remove v and $par(v)$ from T and from their corresponding node sets, and let r_v now represent triangle Δ. In the second

situation, remove all nodes of T_v from T and from their corresponding node sets. Note that triangle $ele(par(r_v))$ becomes a degenerate quadrilateral or a pentagon.

Sub-Case 4(c)(iii). *Node r_v has degree 3, node $sib(par(v))$ is a leaf, and element $ele(sib(par(v)))$ is a triangle.* If $G_v = T_v$, the domain of $\mathcal{T}_v$ is a hexagon. Otherwise, the domain of $\mathcal{T}_v$ is a quadrilateral that contains a vertex of $\mathcal{T}_v$ in its interior (see Figure 3.12). In the former case, carry out the decomposition from Lemma B.1.1. In the latter case, carry out the decomposition from Lemma B.1.2. In either case, remove all nodes of T_v from T and from their corresponding node sets.

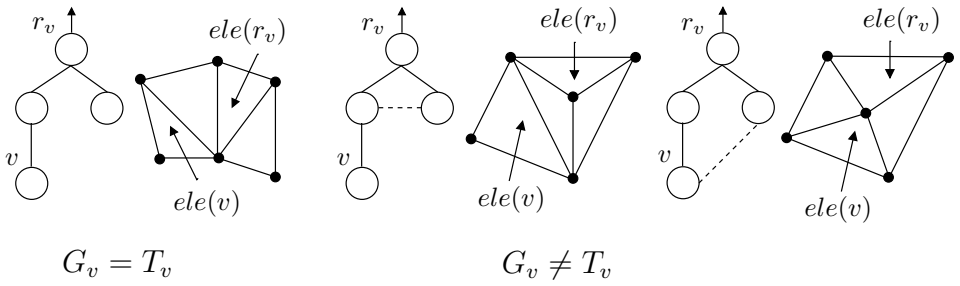


Figure 3.12: Sub-case 4(c)(iii) of step 4c (dashed edges are cross-edges).

Sub-Case 4(c)(iv). *Node r_v has degree 3, node $sib(par(v))$ is a leaf, and element $ele(sib(par(v)))$ is a non-empty triangle.* Note that such a non-empty triangle must have been created in Step 4(b). Refer to Figure 3.13. If $G_v = T_v$, the domain of $\mathcal{T}_v$ is a hexagon with a point in its interior. Otherwise, the domain of $\mathcal{T}_v$ is a quadrilateral that contains two vertices of $\mathcal{T}_v$ in its interior. In both cases, consider the triangulated pentagon P defined by $ele(v)$, $ele(par(v))$, and $ele(r_v)$. Apply Lemma B.2.1 to P, where the shared edge of $ele(r_v)$ and $ele(sib(par(v)))$ is designated as the "outgoing" edge. If there is a leftover triangle Δ, this triangle and $ele(sib(par(v)))$ form a triangulated quadrilateral

with a vertex of $\mathcal{T}_v$ in its interior. Otherwise, a Steiner point has been placed on the edge shared by $ele(r_v)$ and $ele(sib(par(v)))$. In either case, the element $ele(sib(par(v)))$ becomes a triangulated quadrilateral with a vertex of $\mathcal{T}_v$ in its interior. Apply Lemma B.1.2 to the resulting quadrilateral. Remove all nodes of T_v from T and from their corresponding node sets.

Sub-Case 4(c)(v). *Node r_v has degree 3 and node $sib(par(v))$ has degree 2.* The different possibilities for the graph G_v are derived from the fact that T is a BFS tree. All cases are illustrated in Figure 3.14. Cases 4(a)-(c) correspond to a pentagon with a point in its interior. For these cases, apply Lemma B.2.2. In cases (d)-(e), the non-root nodes of T_v (v, v_1, v_2 and v_3 in Figure 3.14) correspond to a quadrilateral with a point inside. Apply Lemma B.1.2 for these cases. Finally, case (f) corresponds to a heptagon. For this case, apply Lemma B.3.1. In all cases, remove all four non-root nodes of T_v from T and from their corresponding node sets.

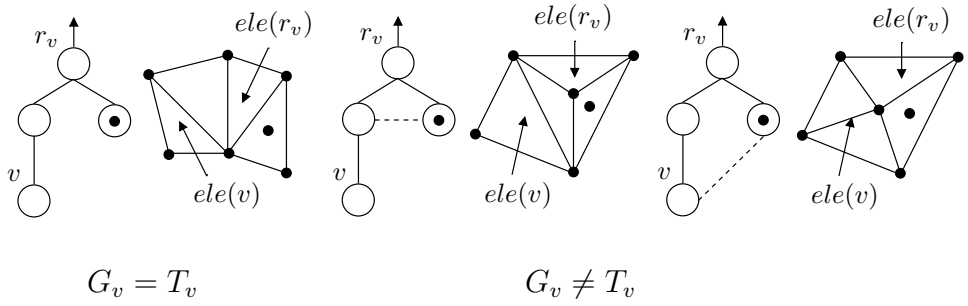


Figure 3.13: Sub-case 4(c)(iv) of step 4c (dashed edges are cross-edges).

Recall that our algorithm repeats steps 1 to 4 to process each of the sets V_d, $V_{d-1}, \ldots, V_2$, in this order. After these sets are processed, they are all empty and the sets V_0 and V_1 may or may not be non-empty. If both V_0 and V_1 are empty, we are done. Otherwise, the algorithm carries out the final step, step 5, which is executed only once.

Step 5. *Eliminate all nodes $v \in V_0 \cup V_1$.* Apply steps 1, 2, and 3 to $V_0 \cup V_1$. If $V_0 \cup V_1$ is now empty, we are done. Otherwise, the nodes in V_1, if any, correspond to triangles of $\mathcal{T}$. The singleton node in V_0, if any, corresponds to either a triangle or a degenerate quadrilateral that contains a boundary edge of $\mathcal{T}$ (because of how we choose the root node of T). We now have the following sub-steps:

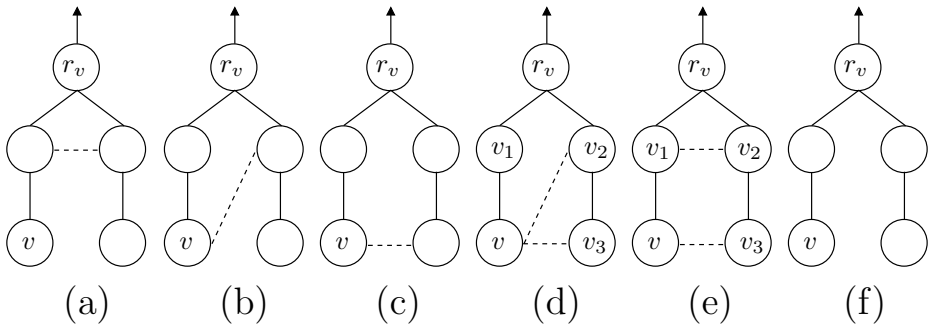


Figure 3.14: G_v for sub-case 4(c)(v) (dashed edges are cross-edges).

Case 5(a). *Set V_1 has two elements.* Let v and $sib(v)$ be the two elements of V_1, and let T_v be the subtree of T rooted at $r_v = par(v)$. The tree T_v corresponds to either a triangulated pentagon or a non-empty triangle. Place a single Steiner point on the boundary edge of $ele(r_v)$ (i.e., a boundary edge of $\mathcal{T}$ that belongs to r_v) and perturb it slightly so that it lies outside the domain of $\mathcal{T}_v$ (see Figure 3.15). The domain of $\mathcal{T}_v$ is now either a hexagon, or a quadrilateral with a point in its interior. By lemmas B.1.1 and B.1.2, respectively, each of these regions can be decomposed into strictly convex quadrilaterals by using at most three additional Steiner points in its interior. Eliminate all three nodes from T, V_0, and V_1.

Case 5(b). Set V_1 *has only one element and the singleton element of V_0 corresponds to a degenerate quadrilateral.* Let v be the singleton element of V_1,

and let T_v be the subtree of T rooted at $r_v = par(v)$. Proceed as in case 4a and let r_v correspond to the leftover triangle Δ. Place a Steiner point on the boundary edge of Δ and perturb it slightly as shown in Figure 3.16 to convert Δ into a strictly convex quadrilateral. Eliminate v and r_v from T, V_1, and V_0.

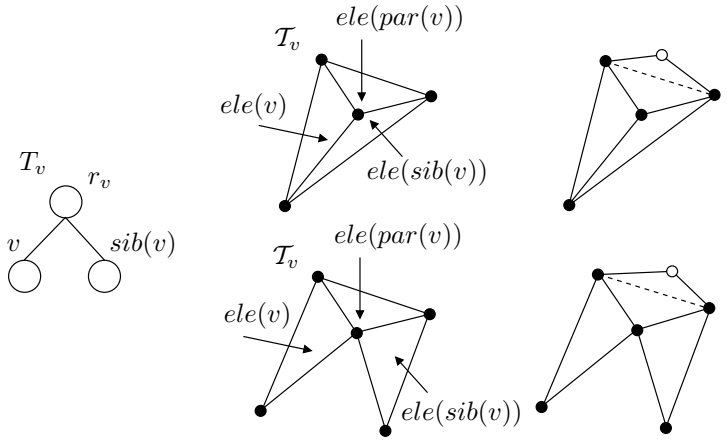


Figure 3.15: Decomposition of a non-empty triangle into two quadrilaterals in step 5(a).

Case 5(c). *Set V_1 has only one element and the singleton element of V_0 corresponds to a triangle.* Again, let v be the singleton element of V_1, and let $r_v = par(v)$. Let Q be the quadrilateral formed by $ele(v)$ and $ele(r_v)$. If Q is strictly convex, simply eliminate the common edge shared by $ele(v)$ and $ele(r_v)$. If Q is not strictly convex, add a Steiner point in the interior of Q and then apply Lemma B.1.2 to decompose it into five strictly convex quads by using three additional Steiner points. In either case, remove v and r_v from T and from their corresponding node sets.

Case 5(d). *Set V_1 is empty, V_0 is not empty and its singleton element corresponds to a triangle.* Let v be the singleton element of V_0. Place a Steiner point s on the edge of $ele(v)$ that is also a boundary edge of $\mathcal{T}$. Perturb s, as

shown in Figure 3.16, so that it lies outside the domain of $\mathcal{T}$ and Q becomes a strictly convex quadrilateral. Remove v from T and V_0.

Note that each case of step 5 can be executed at most once and they are all mutually exclusive. Furthermore, cases 5(a), 5(b), and 5(d) are the only cases when our algorithm adds a Steiner point outside the domain of $\mathcal{T}$. When $V_0 \cup V_1$ is empty, the spanning tree T is also empty and the triangular mesh $\mathcal{T}$ has been converted into a strictly convex quadrilateral mesh.

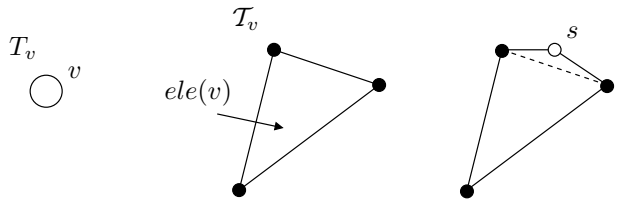


Figure 3.16: Converting a triangle into a quadrilateral.

3.2.2 Correctness, Complexity, and Bounded Size

The fact that our algorithm generates a strictly convex quadrilateral mesh from $\mathcal{T}$ is an immediate consequence of two observations and the lemmas in Appendix B. First, we notice that every vertex v in the set V_l is removed from T and V_l by the time the algorithm processed V_l, where l is the current deepest level of T, with $0 \leq l \leq d$, and d is the initial depth of T. So, T is empty after all sets V_d, $V_{d-1}, \ldots, V_0$ are processed. Second, we noticed that T_v can only be one of subtrees (1), (2), (3), (4) or (5) in Figure 3.7. Each of these trees represents one or more triangulated domains, each of which has been considered by our algorithm. Finally, the lemmas in Appendix B show that the triangulated domains are correctly converted into strictly convex quadrilateral meshes.

Our algorithm is linear in the number t of triangles of $\mathcal{T}$. This is because each node $v \in V_l$ is visited at most twice before being eliminated from V_l and T. Recall

that v is visited twice only if $ele(v)$ is a degenerate pentagon (see step 2), as v is not removed from V_l at that time, but when it is considered for the second time by some other step. Furthermore, when T_v is considered by the algorithm, each node of T_v is visited at most once. Since these nodes are in levels l, $l-1$, and $l-2$, we have that each node of T gets visited a constant number of times during the entire execution of the algorithm. So, the algorithm is indeed linear in t, and the space requirements are clearly linear in t as well.

The following theorem summarizes the above results and states another property of our algorithm:

Theorem 3.2.1. *Let Ω be a polygonal region with or without polygonal holes. Then, given any triangular mesh $\mathcal{T}$ of Ω with t triangles, the algorithm described above can convert $\mathcal{T}$ into a strictly convex quadrilateral mesh with at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals by using at most $t+2$ Steiner points, all except one of which lie within the boundary of Ω. The algorithm runs in $\mathcal{O}(t)$ time and space.*

Proof. We must show that our algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from the spanning tree T of the dual graph G of $\mathcal{T}$. In order to see this, consider Table 3.1. Each row of this table contains, for a given step of the algorithm, the maximum number of nodes of T eliminated, the maximum number of quadrilaterals created, and the maximum number of Steiner points inserted to create those quadrilaterals. Note that the algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from T in all steps except steps 2, 3(a), 4(a), 5(a), and 5(c). So, we focus our attention on these cases. In step 2, $ele(v)$ is a degenerate pentagon. In steps 3(a) and 4(a), $ele(par(v))$ is a degenerate quadrilateral. Degenerate quadrilaterals and pentagons can only arise as the result of the execution of step 4(c)(ii). Hence, step 2 must be preceded by two executions of step 4(c)(ii), while steps 3(a) and 4(a) must each be preceded by one execution of step 4(c)(ii). The combination of one execution of step 2 and two executions of step 4(c)(ii) eliminates 6 nodes from T and produces at most

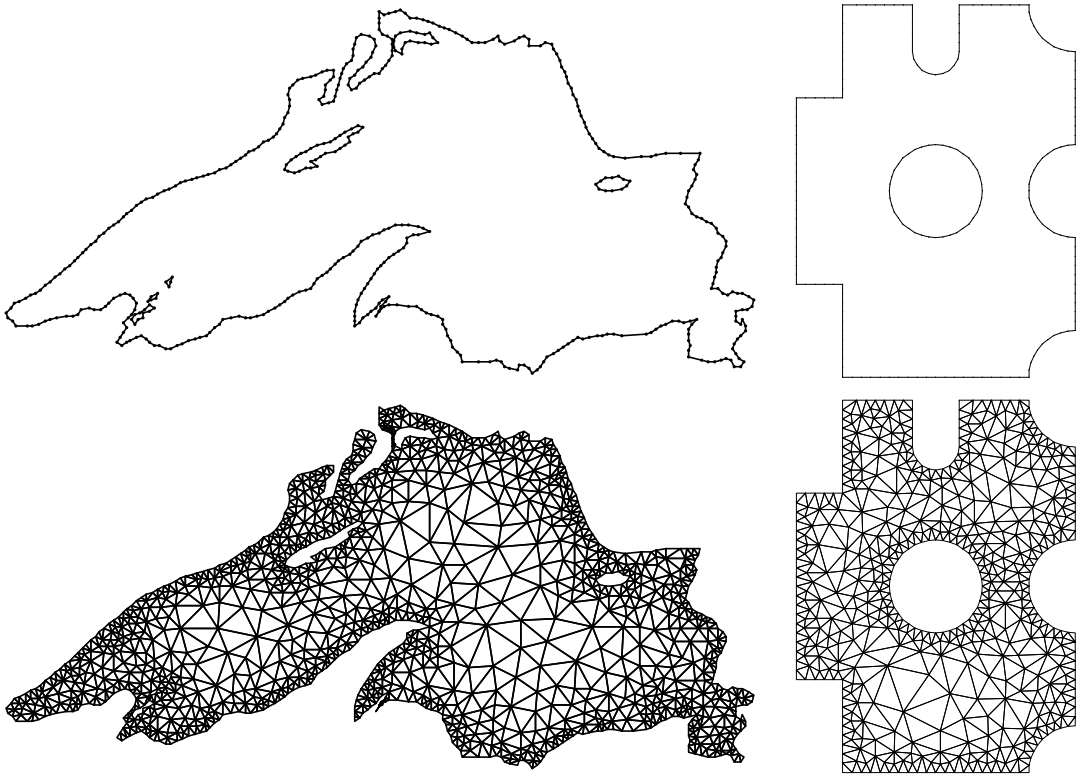
9 quadrilaterals. The combination of one execution of step 3(a) and one execution of step 4(c)(ii) eliminates 7 nodes from T and produces at most 8 quadrilaterals. The combination of one execution of step 4(a) and one execution of step 4(c)(ii) eliminates 5 nodes from T and produces at most 5 quadrilaterals. Thus, in all these situations, the algorithm produces at most three strictly convex quadrilaterals for every two nodes eliminated from T. Finally, only one of steps 5(a) and 5(c) can be executed and at most once. Step 5(a) creates 5 quads after eliminating three nodes, thus creating one more quad than the target ratio. Step 5(c) creates 5 quads after eliminating two nodes, thus creating two more quads than the target ratio. It follows that our algorithm constructs a strictly convex quadrilateral mesh with at most $\lfloor \frac{3t}{2} \rfloor + 2$ quadrilaterals. From Table 3.1, we also have that the algorithm uses at most k Steiner points for every k nodes eliminated from T in all steps except for steps 5(a) and 5(c), in which at most $k + 2$ Steiner points are used. Since only one of these steps is executed and at most once, it follows that our algorithm places at most $t + 2$ Steiner points. Furthermore, all Steiner points are placed within the boundary of Ω, except for one Steiner point inserted in Steps 5(a), 5(b), or 5(d). Since these steps are mutually exclusive and executed at most once, our algorithm places at most one Steiner point outside Ω. Correctness and complexity follow from our discussion in the beginning of this section. □

Step	**Nodes Eliminated**	**Quad. Created**	**Steiner Pts. Inserted**
1	1	1	0
2	0	1	0
3(a)	3	5	3
3(b)	4	5	3
3(c)	4	6	4
3(d)	7	10	7
4(a)	1	2	1
4(b)	2	2	1
4(c)(i)	3	4	3
4(c)(ii)	3	4	3
4(c)(iii)	4	6	4
4(c)(iv)	6	9	6
4(c)(v)(a)–(c)	4	5	3
4(c)(v)(d)–(e)	4	5	3
4(c)(v)(f)	4	6	4
5(a)	3	5	4 (1 outside Ω)
5(b)	2	3	2 (1 outside Ω)
5(c)	2	5	4
5(d)	1	1	1 (1 outside Ω)

Table 3.1: Upper bounds on number of quadrilaterals created and Steiner points inserted to quadrangulate Ω.

3.2.3 Results and Discussion

We implemented the algorithm described in Section 3.2.1 using C++ and the open source and freely available software *Computational Geometry and Algorithms Library* [45]. To illustrate the properties of our algorithm, we consider the triangular meshes in Figure 3.17(a) and Figure 3.17(b), which were both produced by the freely available software `Triangle` [119] from the polygonal regions shown on top of the meshes.

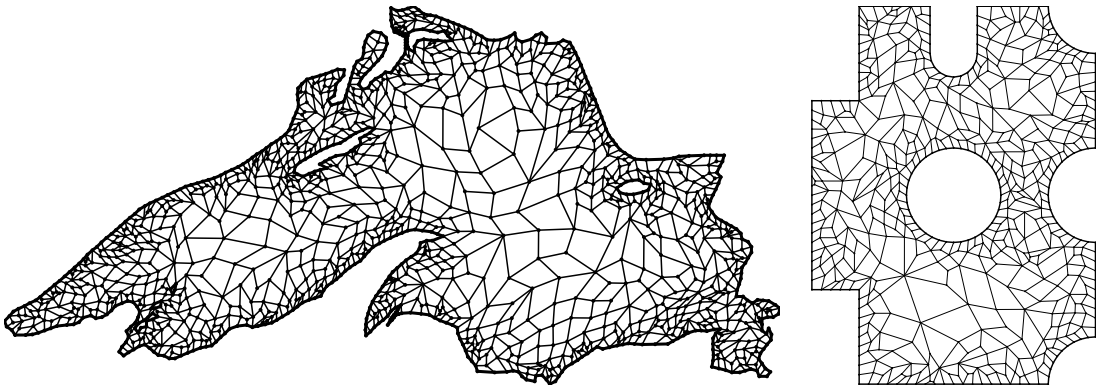


(a) (b)

Figure 3.17: (a) Triangular mesh of an outline of Lake Superior shown on the top. (b) Triangular mesh of a CAD model shown on the top. Both meshes were generated by `Triangle`.

Figure 3.18(a) and Figure 3.18(b) show the quadrilateral meshes obtained from

the triangular meshes in Figure. 3.17(a) and Figure 3.17(b), respectively, using our algorithm. The quadrilateral meshes in Figure 3.18(a) and Figure 3.18(b) have 1249 and 511 quadrilaterals, while their triangular counterparts in Figure 3.17(a) and Figure 3.17(b) have 2160 and 912 triangles, respectively. Our algorithm used 169 and 55 Steiner points to generate the quadrilateral meshes in Figure 3.18(a) and Figure 3.18(b), respectively.



(a) (b)

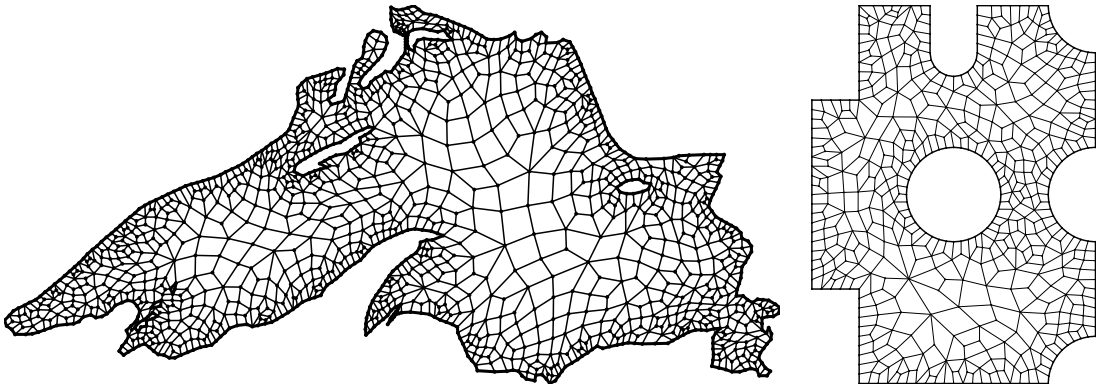
Figure 3.18: (a) Quadrilateral mesh of a polygonal region of Lake Superior's shape. (b) Quadrilateral mesh of a CAD model.

Note that even though our algorithm may, in the worst case, produce up to $\lfloor \frac{3t}{2} \rfloor$ quadrilaterals and use up to $t + 2$ Steiner points on an input triangular mesh with t triangles, in the previous examples the number of quadrilaterals and the number of Steiner points are about 60% and 8%, respectively, of the number of triangles of the input triangular meshes. It turns out that this reduction in mesh size and the use of only a few Steiner points were observed in almost all test cases on which we ran our algorithm.

Our algorithm does not provide any guarantee on the shape quality of the output quadrilaterals. This is true no matter what shape metric one might consider. Since element shape quality is one of the most desirable properties of any mesh, this is

a considerable weakness of our algorithm. Fortunately, we can improve the shape quality of the quadrilaterals generated by our algorithm by using post-processing techniques.

Post-processing techniques optimize the shape quality of a mesh according to some quality criterion and at the expense of runtime. Figure 3.19(a) and Figure 3.19(b) show quadrilateral meshes obtained from the ones in Figure 3.18(a) and Figure 3.18(b), respectively, by using a smoothing technique called *angle-based smoothing* [146] and a subset of the topological operations from [19]. The combination of both techniques help make quadrilaterals less distorted with respect to a reference square.

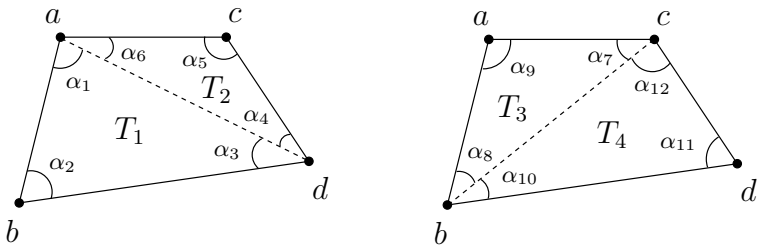


(a) (b)

Figure 3.19: (a) Quadrilateral mesh resulting from post-processing of the mesh in Figure 3.18(a). Quadrilateral mesh resulting from post-processing of the mesh in Figure 3.18(b).

We can quantitatively evaluate the effect of the post-processing techniques by considering a measure of shape quality defined by Joe [71] (refer to Figure 3.20). Let $Q = [a, b, c, d]$ be a strictly convex quadrilateral with vertices a, b, c, and d. Then, we define a quality measure μ as follows. The quadrilateral Q can be decomposed into two triangles by either adding the diagonal $\overline{bd}$ or the diagonal $\overline{ac}$ to it. In the

former case, we have the triangles $T_1 = [a, b, d]$ and $T_2 = [c, d, b]$. In the latter case, we have the triangles $T_3 = [a, b, c]$ and $T_4 = [d, a, c]$. We define μ to be the smallest angle of T_1, T_2, T_3, and T_4.



$$\mu = \min\{\alpha_i \mid 1 \leq i \leq 12\}$$

Figure 3.20: Illustration of the definition of the quality measure μ.

Note that the value of μ is 45° if Q is a square, and that the value of μ approaches 0° if one or two edges of Q is much shorter than the other edges or Q has an angle near 0° or 180°. In fact, it can be shown that the maximum value of μ is 45°, and that this value is only attained by the square [71]. The value of μ is a measure of how distorted the triangles T_1, T_2, T_3, and T_4 are with respect to a reference right, isosceles triangle. This interpretation was recently formalized by Pébay [109] in a study of the asymptotic behavior of a shape metric for quadrilaterals, which is very similar to μ.

Table 3.2 shows the frequency distribution of the μ values of the quadrilaterals of the meshes in Figure 3.18 and Figure 3.19. Although the values of μ were improved for the post-processed quadrilaterals, the time to produce each mesh in Figure 3.19 is at least seven times longer than the time our algorithm took to produce the corresponding mesh in 3.18. The angle-based smoothing also increases the uniformity of the area of the mesh elements. Table 3.3 illustrates this effect by showing the standard deviation of the average area of the elements of meshes 1, 2, 3, and 4 in Table 3.2.

Another interesting feature of our algorithm is that it tends to retain the input

triangular mesh grading. This is because the conversion of the triangular mesh into a quadrilateral one is done locally, and the size of the quadrilaterals are proportional to the size of the triangles in the small triangulated domains converted into small quadrilateral meshes.

Mesh	**# Quads.**	$[0^o, 5^o)$	$[5^o, 10^o)$	$[10^o, 25^o)$	$[25^o, 35^o)$	$[35^o, 45^o]$	**Runtime**
1	1249	11	430	566	228	14	1401 ms
2	511	2	170	225	110	4	429 ms
3	1249	2	166	587	458	36	10956 ms
4	511	2	38	266	191	14	4791 ms

Table 3.2: Meshes 1 and 2 are the quadrilateral meshes in Figure 3.18(a) and Figure 3.18(b), respectively. Meshes 3 and 4 are the post-processed quadrilateral meshes in Figure 3.19(a) and Figure 3.19(b), respectively. The second column from left to right shows the number of quadrilaterals of the meshes. The rightmost column shows the time our meshing algorithm took to generate the mesh.

Mesh	**Average Area**	**Standard Deviation**	**Mesh**	**Standard Deviation**
1	1.2×10^{-4}	1.6×10^{-4}	3	1.2×10^{-4}
2	7.1	6.5	4	4.1

Table 3.3: Average area of the quadrilaterals in the meshes in Table 3.2 before and after post-processing.

The fact that our algorithm places a Steiner point outside the triangular mesh domain may be highly undesirable in practice. However, our algorithm can be slightly modified so that no Steiner point is placed outside the triangular mesh domain. Note that a Steiner point is placed outside the triangular mesh domain only if one of steps 5(a), 5(b), and 5(d) is carried out. Instead of adding a Steiner point to the triangular mesh boundary to convert the leftover triangle into a quadrilateral in steps 5(b) and

5(d), we can subdivide the leftover triangle into five strictly convex quadrilateral by adding five Steiner point in the interior of the triangle and one on the boundary [71] (see Figure 3.21).

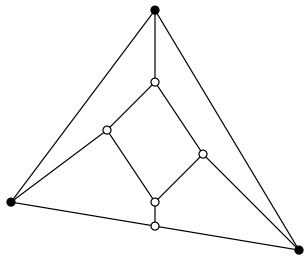


Figure 3.21: We can subdivide a triangle into five strictly convex quadrilaterals using five Steiner points, one of which is placed on the boundary of the triangle (see [71] for a proof).

Note that steps 5(b) and 5(d) insert 6 and 5 Steiner points into the resulting quadrilateral mesh, respectively, after our change. Now, consider step 5(a). Recall that we have two possible situations for the subtree T_v of T rooted at $par(v)$: T_v corresponds to either a triangulated pentagon or a non-empty triangle. If T_v corresponds to a triangulated pentagon, then we can apply Lemma B.2.1 to convert the triangular mesh $\mathcal{T}_v$ into a quadrilateral mesh and a leftover triangle that contains a boundary edge. Next, we subdivide the leftover triangle into five strictly convex quadrilaterals by using five Steiner points inside the triangle. So, we insert exactly six Steiner points into the resulting quadrilateral mesh.

If T_v corresponds to a non-empty triangle, then we can insert a Steiner point in the interior of $ele(v)$, so that the elements $ele(v)$ and $ele(sib(v))$ form a quadrilateral with a vertex in its interior. Hence, we can apply Lemma B.1.2 to subdivide the quadrilateral formed by $ele(v)$ and $ele(sib(v))$ into at most five strictly convex quadrilateral using at most three Steiner points. All Steiner points are placed in the interior of the quadrilateral formed by $ele(v)$ and $ele(sib(v))$. Now, we are left with only one triangle, $ele(par(v))$. We can subdivide $ele(par(v))$ into five strictly

convex quadrilaterals by adding five Steiner points to the interior of $ele(par(v))$. So, the total number of Steiner points inserted into the resulting quadrilateral mesh by step 5(a) is at most nine. This implies that our algorithm can convert a triangular mesh with t triangles into a strictly convex quadrilateral mesh by using at most $t + 7$ Steiner points, all of which are placed in the interior or on the boundary of the triangular mesh domain.

3.3 Constrained Quadrilateral Meshes

The algorithm described in Section 3.2.1 can convert unconstrained triangular meshes of arbitrary polygonal regions into quadrilateral meshes of the same domain. In this section, we extend this algorithm to handle a particular class of constrained triangular meshes. The ability to handle constrained triangular meshes is crucial for generating meshes from two-dimensional imaging data, as we have already seen in Chapter 2.

The class of constrained triangular meshes handled by our algorithm encompasses any triangular mesh generated from a two-dimensional binary image. More specifically, the input triangulated domain is an arbitrary polygonal region, Ω, with interior edges and vertices satisfying the *input restriction*: the vertices and edges of $bd(\Omega)$ and the interior vertices and edges are the underlying space of a pure one-dimensional polytopal complex with empty boundary; that is, each vertex is a vertex of an edge of the complex and every vertex is incident to exactly two edges of the complex.

We assume that our algorithm is informed about which edges of the constrained triangular mesh are boundary edges and *constraining edges*, i.e., edges that cover the interior edges. The output of our algorithm is a quadrilateral mesh that conforms to $bd(\Omega)$ and to the interior vertices and edges. The quadrilateral mesh retains all vertices of the input triangular mesh, and it is likely to contain extra vertices (Steiner

points).

3.3.1 The Extended Algorithm

Let $\mathcal{T}$ be a given constrained triangular mesh such that the domain of $\mathcal{T}$ satisfies the input restriction. Let G be the dual graph of $\mathcal{T}$ such that G does not include the dual of the constraining edges of $\mathcal{T}$, and let h be the number of connected components of G.In order to convert $\mathcal{T}$ into a strictly convex, constrained quadrilateral mesh, we first compute all connected components $\{G_1, G_2, \ldots, G_h\}$ of G and their corresponding spanning trees $\{T_1, T_2, \ldots, T_h\}$, and then we run the algorithm described in Section 3.2.1 on each triangular mesh $\mathcal{T}_i$ using T_i as the spanning tree of G_i, for each $i \in \{1, \ldots, h\}$.

The root node of each T_i represents a triangle adjacent to a boundary edge e_i of $\mathcal{T}_i$. Since the dual graph G of $\mathcal{T}$ does not include the dual of the constraining edges of $\mathcal{T}$, constraining edges cannot be interior edges of the small triangulated polygonal regions that are formed by the algorithm in Section 3.2.1 during the conversion of $\mathcal{T}_i$ into a quadrilateral mesh. Thus, the constraining edges of $\mathcal{T}$ are kept in the resulting quadrilateral mesh. However, at the very last step, the algorithm in Section 3.2.1 may place a Steiner point on at most one of the constraining edges of each $\mathcal{T}_i$. More specifically, the boundary edge belonging to the triangle that corresponds to the root node of T_i.

Note that the conversion of $\mathcal{T}_i$ into a quadrilateral mesh may "corrupt" the quadrilateral mesh resulting from the conversion of $\mathcal{T}_j$, with $i \neq j$ and $1 \leq i, j \leq h$. This may occur if the algorithm places one Steiner point on a boundary edge of $\mathcal{T}_i$ that is also a boundary edge of $\mathcal{T}_j$ after $\mathcal{T}_j$ has been converted into a quadrilateral mesh. This Steiner point will transform a quadrilateral obtained from $\mathcal{T}_j$ into a pentagon. Figure 3.22 illustrates this situation by using a constrained triangular mesh whose dual graph has three connected components. As we shall discuss next, this situation can be avoided by carefully choosing the root nodes of the spanning trees T_1,

$T_2, \ldots, T_h$.

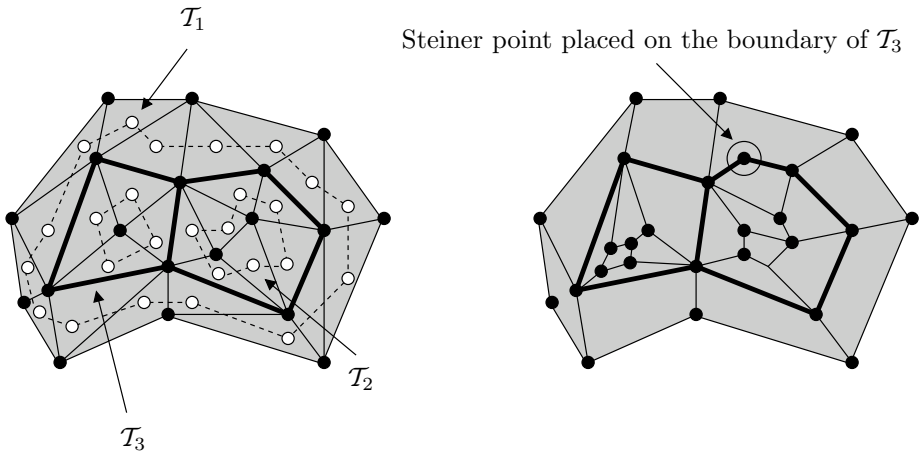


Figure 3.22: The conversion of the triangular mesh $\mathcal{T}_2$ into a quadrilateral mesh corrupted the quadrilateral mesh resulting from $\mathcal{T}_1$. Constraining edges are heavily drawn.

Definition 3.3.1. Let G^c be a graph such that G^c has a node v_i for every connected component G_i of G, and G^c has an edge connecting v_i to v_j if and only if there exist a node $u \in G_i$ and a node $w \in G_j$ such that $ele(u)$ and $ele(w)$ in $\mathcal{T}$ share a common edge, with $i \neq j$ and $1 \leq i, j \leq h$.

Note that the graph G^c is connected. Let T^c be any spanning tree of G^c such that the root of T^c is a node of G^c corresponding to a connected component of G whose underlying triangular mesh has a triangle containing a boundary edge of $\mathcal{T}$. From the definition of G^c, if $v_i \in T^c$ is not the root of T^c then it has a parent node, say v_j. Note that the triangular meshes $\mathcal{T}_i$ and $\mathcal{T}_j$, which correspond to nodes v_i and v_j of T^c, respectively, must share a common constraining edge of $\mathcal{T}$.

To compute a spanning tree T_i for the connected component G_i of G ($1 \leq i \leq h$), we consider two cases. First, the node v_i is the root of T^c, and hence the triangular mesh $\mathcal{T}_i$ has at least one triangle $\triangle$ that contains a boundary edge of $\mathcal{T}$. Second, the node v_i has a parent in T^c, say v_j, and hence the triangular mesh $\mathcal{T}_i$ has at

least one triangle $\triangle$ that shares a constraining edge with a triangle of $\mathcal{T}_j$, for some $j \neq i$ and $1 \leq j \leq h$. In either case, we choose the root of T_i to be the node of G_i corresponding to $\triangle$. Finally, we obtain T_i by performing a BFS on G_i, starting from the node of G_i corresponding to $\triangle$.

Suppose we run the algorithm in Section 3.2.1 on $\mathcal{T}_i$ using T_i as the spanning tree of G_i. If the algorithm places a Steiner point s on a boundary edge of $\mathcal{T}_i$, this boundary edge (call it e) is an edge of the triangle of $\mathcal{T}_i$ corresponding to the root node of T_i. Furthermore, e is also a boundary edge of exactly one other triangular mesh, namely $\mathcal{T}_j$, where v_j is the parent node of v_i in T^c. So, by converting $\mathcal{T}_i$ into a quadrilateral mesh, our algorithm may affect the triangular mesh $\mathcal{T}_j$, but no other triangular mesh.

The above observation suggests a straightforward way of avoiding the situation in Figure 3.22. We traverse T^c by level and in a bottom-up fashion, and for each visited node v_i of T^c, with $i \in \{1, \ldots, h\}$, we apply the algorithm in Section 3.2.1 to $\mathcal{T}_i$. Since we traverse T^c by level and in a bottom-up fashion, no quadrilateral mesh obtained before the conversion of $\mathcal{T}_i$ is corrupted by the conversion of $\mathcal{T}_i$. Furthermore, the conversion of $\mathcal{T}_i$ may only affect the triangular mesh $\mathcal{T}_j$, which corresponds to the parent v_j of v_i in T^c.

Note also the conversion of $\mathcal{T}_i$ affects $\mathcal{T}_j$ if and only if a Steiner point s is placed on a boundary edge of $\mathcal{T}_i$ and $\mathcal{T}_j$, which is an edge of the triangle of $\mathcal{T}_i$ corresponding to the root of T_i. If this is the case, then the triangle of $\mathcal{T}_j$ containing this edge becomes a quadrilateral. All we have to do is to split this quadrilateral into two triangles (see Figure 3.23). This repairing procedure modifies $\mathcal{T}_j$, but it guarantees that the modified $\mathcal{T}_j$ is a consistent triangular mesh before it is considered for conversion by the algorithm.

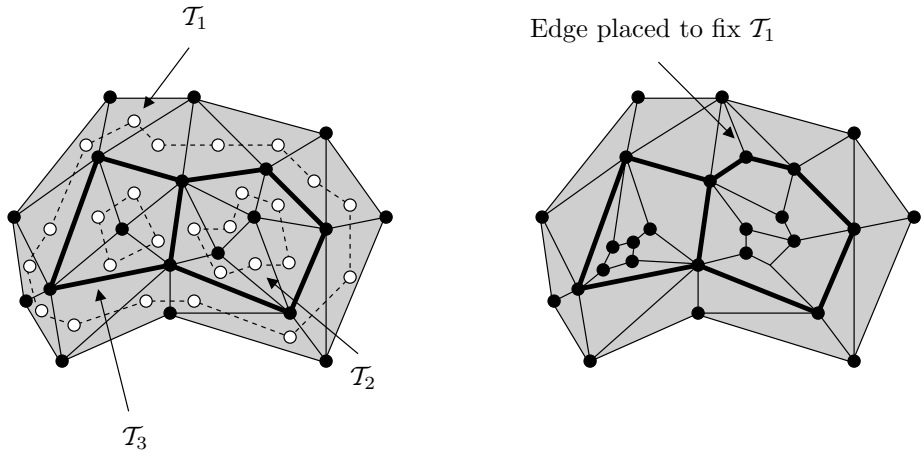


Figure 3.23: Consider the triangular mesh of Figure 3.22. If v_1 is the parent of v_2 in T^c, then we can fix $\mathcal{T}_1$ after the conversion of $\mathcal{T}_2$ by splitting the corrupted quadrilateral into two triangles.

3.3.2 Bounded Size

Our algorithm for producing constrained quadrilateral meshes has the same theoretical properties of the algorithm in Section 3.2.1. The following lemma provides upper bounds on the number of quadrilaterals and Steiner points generated and inserted by our algorithm, respectively.

Lemma 3.3.1. *Let $\mathcal{T}$ be a constrained triangular mesh satisfying the input restriction. The algorithm described in Section 3.3.1 converts $\mathcal{T}$ into a strictly convex, constrained quadrilateral mesh of the triangulated domain with at most $\lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals by inserting at most $t + 3h$ Steiner points, where t is the number of triangles of $\mathcal{T}$ and h is the number of connected components in the dual graph of $\mathcal{T}$.*

Proof. The conversion of each triangular mesh $\mathcal{T}_i$ into a quadrilateral one may only affect another triangular mesh $\mathcal{T}_j$ ($j \neq i$ and $1 \leq j \leq h$) that has not been converted into a quadrilateral mesh yet, except for the last triangular mesh considered by the algorithm, which does not affect any other triangular mesh or previously computed quadrilateral mesh. Every time a triangular mesh $\mathcal{T}_j$ is affected by the conversion

of another triangular mesh, the number of triangles in $\mathcal{T}_j$ is increased by 1 due to the repairing procedure shown in Figure 3.23. Furthermore, each triangular mesh $\mathcal{T}_j$ can be affected at most x_j times where x_j is the number of children of the node v_j of T^c corresponding to the spanning tree T_j of G_j. Note that $\sum_{i=1}^{h} x_i = h - 1$, as $h - 1$ is the number of edges in T^c. Let t_i $(1 \leq i \leq h)$ be the number of triangles in $\mathcal{T}_i$ before the algorithm converts any triangular mesh into quadrilateral one, and let t'_i be the number of triangles of $\mathcal{T}_i$ at the time the algorithm converts $\mathcal{T}_i$ into a quadrilateral mesh. Then, we have that $t_i \leq t'_i \leq t_i + x_i$. From Theorem 3.2.1, we know that the algorithm in Section 3.2.1 produces $\lfloor \frac{3t'_i}{2} \rfloor + 2$ quadrilaterals on input $\mathcal{T}_i$. It follows therefore that the above algorithm produces at most $\sum_{i=1}^{h} \big(\lfloor \frac{3t'_i}{2} \rfloor + 2 \big) \leq \sum_{i=1}^{h} \big(\lfloor \frac{3(t_i + x_i)}{2} \rfloor \big) + 2h \leq \sum_{i=1}^{h} \frac{3t_i}{2} + \sum_{i=1}^{h} \frac{3x_i}{2} + 2h = \frac{3t}{2} + \frac{3h-3}{2} + 2h < \lfloor \frac{3t}{2} \rfloor + 4h$ quadrilaterals to convert $\mathcal{T}$ into a constrained quadrilateral mesh. Furthermore, these quadrilaterals are all strictly convex, as the algorithm in Section 3.2.1 only generates strictly convex quadrilaterals. From Theorem 3.2.1, we also know that the algorithm in Section 3.2.1 inserts at most $t'_i + 2$ Steiner points to convert $\mathcal{T}_i$ into a quadrilateral mesh. Thus, the algorithm extension in Section 3.3.1 inserts at most $\sum_{i=1}^{h}(t' + 2) \leq \sum_{i=1}^{h}(t_i + x_i + 2) = t + h - 1 + 2h < t + 3h$ Steiner points to convert $\mathcal{T}$ into a constrained quadrilateral mesh. □

3.3.3 Meshes from Images

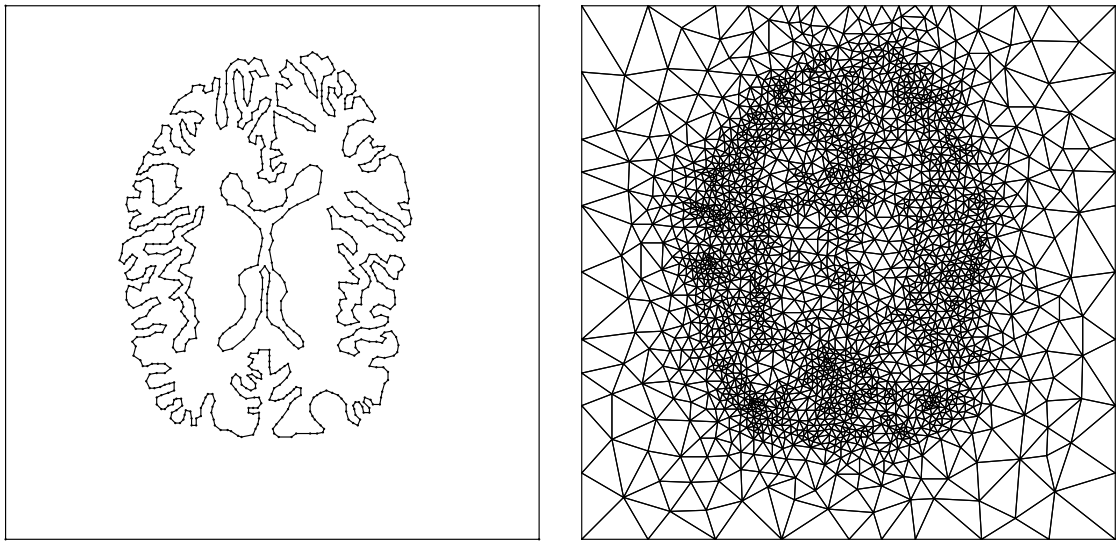
As we pointed out before, the algorithm in Section 3.3.1 can be used to generate strictly convex quadrilateral meshes from two-dimensional binary images. So, this algorithm is a solution for one of the two meshing problems studied in this thesis. Here, we show an example of a mesh generated by our algorithm from a human brain image (see Figure 3.24-3.25).

The human brain image is a two-dimensional binary image (D, X). The image grid D is such that

$$D = \{p = (p_1, p_2) \in \mathbb{R}^2 \mid p = O + \delta \cdot g\},$$

where $O = (o_1, o_2) \in \mathbb{R}^2$, δ is a positive real number (grid spacing), and $g = (g_1, g_2) \in \{0, n_1\} \times \{0, n_2\}$, for some positive integers n_1 and n_2. The set X is a nonempty subset of D such that if $p = (p_1, p_2) \in X$, then $p_i \neq o_i$ and $p_i \neq o_i + \delta \cdot n_i$, for all $i \in \{1, 2\}$.

To generate a mesh from (D, X), we first consider the rectangle $\Omega = conv(D)$ and the set of common edges (along with their vertices) of the pixels $CA(p)$ and $CA(q)$, for every (p, q) in $bd(X)$. These edges and vertices are interior edges and vertices of Ω. Let $\mathcal{C}$ denote the pure one-dimensional polytopal complex with empty boundary consisting of the edges and vertices of the boundary $bd(\Omega)$ of Ω and the interior edges and vertices.



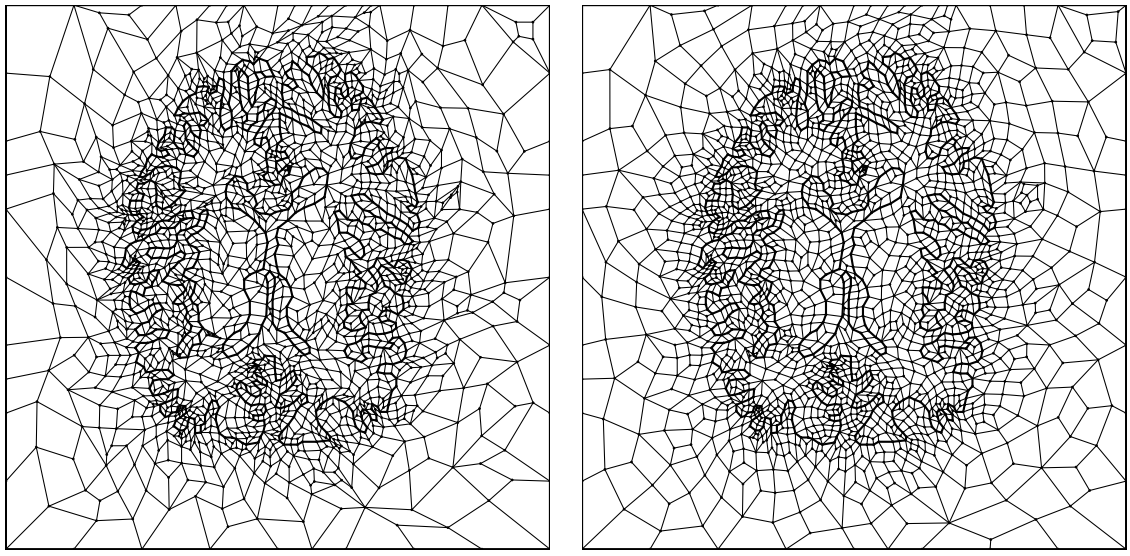
(a) (b)

Figure 3.24: (a) Polytopal complex representing an image domain. (b) A triangular mesh of the domain in (a) produced by `Triangle`.

Before we generate a triangular mesh from $\mathcal{C}$, we simplify $\mathcal{C}$. This is because $\mathcal{C}$ has a large number of interior edges and vertices. For the example shown here, this simplification has been carried out manually; that is, by selecting a subset of

vertices of $\mathcal{C}$, which were later connected to form a simpler homeomorphic complex. However, one can use some algorithm for curve simplification to automatically do the same task [67, 140].

Figure 3.24 shows the simplified polytopal complex and a triangular mesh produced by the software `Triangle` from the complex. The triangular mesh conforms to the interior edges and vertices. Figure 3.25(a) shows the corresponding quadrilateral mesh produced from the triangular mesh in Figure 3.24. Figure 3.25(b) shows the post-processed quadrilateral mesh after five iterations of the angle-based smoothing algorithm.



(a) (b)

Figure 3.25: (a) Quadrilateral mesh generated by our algorithm from the triangular mesh in Figure 3.24. (b) Quadrilateral mesh in (a) after five iterations of the angle-based smoothing.

A few years ago, we evaluated the quality of some meshes produced by our algorithm from two-dimensional binary images of the human brain [125, 113]. Our evaluation consisted of comparing the accuracy of the solution of a finite element (FE)

based image registration application when provided with *post-processed* quadrilateral meshes produced by our algorithm and their triangular counterparts. Although the triangular meshes were generated by a software that optimizes triangle shape quality, our experiment showed us that the quality of the quadrilateral meshes are comparable with their triangular counterparts with respect to the FE-based image registration application.

One may wonder if our algorithm can handle constrained triangular meshes of more general polytopal complexes. For instance, a polytopal complex in which an interior vertex is incident to only one edge. Unfortunately, our algorithm cannot handle such a complex (see Figure 3.26). To overcome this limitation, we must learn how to produce quadrilateral meshes from small triangulated polygons containing constraining edges.

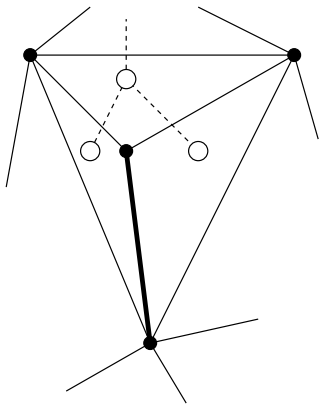


Figure 3.26: The triangulated domain corresponding to the dual graph is a non-empty triangle with a constraining edge. Such domain is not currently handled by our algorithm.

Although the combination of our algorithm in Section 3.2.1 with the angle-based smoothing can produce quadrilateral meshes that can be used by FE-based applications, the post-processing step is very time-consuming and may not improve the quality of the quadrilaterals beyond a certain limit. So, it is natural to try to improve

the shape quality of the quadrilaterals produced by our algorithm by modifying the algorithm itself. However, the problem of generating strictly convex quadrilateral meshes with theoretical guarantees on mesh size and element shape quality is a difficult one [9].

Nevertheless, we intend to investigate the possibility of generating quadrilateral meshes of small polygons (see Appendix B) where all quadrilateral are well-shaped and the number of Steiner points is still bounded. If we succeed, then we can easily modify the algorithm described in Section 3.2.1 to generate well-shaped quadrilateral meshes as well.

Chapter 4

Surface Meshes from 3D Images

This chapter introduces the problem of generating surface meshes from three-dimensional binary images. We also discuss our own solution for this problem, which consists of three parts. Each part is the subject of one forthcoming chapter of this thesis. Most of the material in this chapter are based on a book on digital geometry by Herman [65] and a paper on three-dimensional well-composed images by Longin Jan Latecki [79].

4.1 Three-Dimensional Digital Images

For any positive real number δ, we define

$$\delta\mathbb{Z}^3 = \{(\delta n_1, \delta n_2, \delta n_3) \in \mathbb{R}^3 \mid (n_1, n_2, n_3) \in \mathbb{Z}^3\}.$$

Definition 4.1.1. A *3D (gray-scale) digital image* $\mathcal{I} : D \to V$ is a function from a set $D \subset \mathbb{R}^3$ to a set $V \subseteq \mathbb{R}$ such that D is a translation of the set $\delta\mathbb{Z}^3$; that is, we can express D by the pair (O, δ),

$$D = \{p \in \mathbb{R}^3 \mid p = O + \delta \cdot g,\ g \in \mathbb{Z}^3\},$$

where $O \in \mathbb{R}^3$. We refer to D as the *image domain* or *image grid*, to the elements in V as *colors*, to δ as the *grid spacing*, and to the point O as the *grid origin*.

Definition 4.1.2. A *3D binary digital image* is a 3D digital image $\mathcal{I} : D \to V$ in which the set V of colors is the binary set $\{0, 1\}$. We shall denote a 3D binary image $\mathcal{I}$ by the pair (D, X), where D is the grid of $\mathcal{I}$ and X is the set $X = \{p \in D \mid \mathcal{I}(p) = 1\}$. We also refer to X and X^c as the *foreground* and the *background* of (D, X), respectively, and we may occasionally denote X by X_1 and X^c by X_0.

Definition 4.1.3. Let $CA : D \to \mathcal{P}(\mathbb{R}^3)$ be the function that identifies each point $p = (p_1, p_2, p_3)$ of D with the cube

$$\left[p_1 - \frac{\delta}{2}, p_1 + \frac{\delta}{2}\right] \times \left[p_2 - \frac{\delta}{2}, p_2 + \frac{\delta}{2}\right] \times \left[p_3 - \frac{\delta}{2}, p_3 + \frac{\delta}{2}\right],$$

where $\mathcal{P}(\mathbb{R}^3)$ denotes the power set of $\mathbb{R}^3$, and δ is the grid spacing of D. We refer to $CA(p)$ as the *continuous analog* of p or simply as the *voxel* of p.

We can extend the definition of continuous analog of a point to a set of points as follows:

Definition 4.1.4. Let X be any subset of D, and let $CA : \mathcal{P}(D) \to \mathcal{P}(\mathbb{R}^3)$ be the function such that

$$CA(X) = \bigcup_{p \in X} CA(p).$$

We refer to $CA(X)$ as the *continuous analog of the set* X.

Figure 4.1 shows the continuous analog of a set $X = \{a, b, c, d\}$ consisting of four points a, b, c, and d of a grid. We now define some important binary relations on D:

Definition 4.1.5. Two distinct points p and q of D are said to be *face-adjacent* if $CA(p)$ and $CA(q)$ share a face, or equivalently, if two of the coordinates of p and q are the same and the third coordinates differ by δ, where δ is the grid spacing of D.

Definition 4.1.6. Two distinct points p and q of D are said to be *edge-adjacent* if $CA(p)$ and $CA(q)$ share an edge but not a face, or equivalently, if one of the coordinates of p and q is the same and the other two coordinates differ by δ, where δ is the grid spacing of D.

Definition 4.1.7. Two distinct points p and q of D are said to be *corner-adjacent* if $CA(p)$ and $CA(q)$ share a vertex but not an edge, or equivalently, if the coordinates of p and q differ by δ, where δ is the grid spacing of D.

For an example, consider the set $\{a, b, c, d\}$ of points in Figure 4.1. Points a and b are face-adjacent, points b and c are edge-adjacent but not face-adjacent, and points c and d are corner-adjacent but not edge- nor face-adjacent. The face-adjacency relation is also known as 6-*adjacent*. Two other useful adjacency relations are the 18-*adjacency* and the 26-*adjacency* relations.

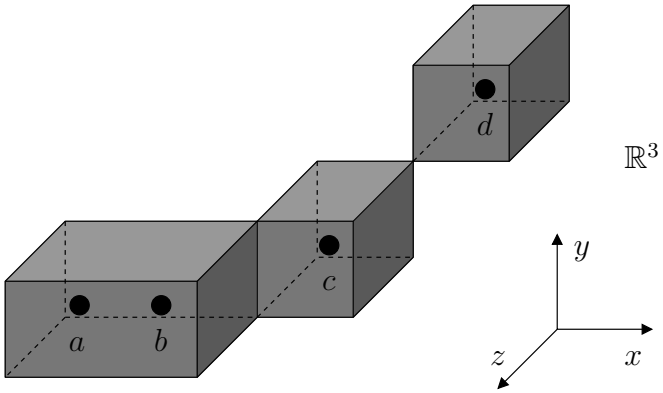


Figure 4.1: Continuous analog of four grid points.

Definition 4.1.8. Two points p and q of D are said to be 18-*adjacent* if they are face- or edge-adjacent, and 26-*adjacent* if they are face-, edge-, or corner-adjacent.

For instance, in Figure 4.1, a and b are 6-, 18-, and 26-adjacent, b and c are 18-, and 26-adjacent but not 6-adjacent, and c and d are 26-adjacent but not 6- nor 18-adjacent.

Let X be any subset of D, and let $\rho \in \{6, 18, 26\}$. We define ρ-connectedness and ρ-connected components in the same way we did for 2D digital images (see Definition 2.1.8 and Definition 2.1.9). For instance, if X is the set of points in Figure 4.1, then a is 6-, 18-, and 26-connected in X to b, c, and d, respectively. Furthermore, X is a 26-connected component of D.

We can extend the definition of continuous analog to pair of points of D and to sets of pairs of points of D as follows:

Definition 4.1.9. Let $CA : D \times D \to \mathcal{P}(\mathbb{R}^3)$ be the function that identifies a pair (p, q) of points of D with the intersection set $CA(p) \cap CA(q)$. Note that if p and q are face-adjacent then $CA((p, q))$ is precisely the face (a square) shared by $CA(p)$ and $CA(q)$. Likewise, we let $CA : \mathcal{P}(D \times D) \to \mathcal{P}(\mathbb{R}^3)$ be the function such that, for any subset Y of $D \times D$,

$$CA(Y) = \bigcup_{(p,q)\in Y} CA((p, q)).$$

Definition 4.1.10. Let X be a nonempty subset of D. Then, we define the *digital boundary* $bd(X)$ *between* X *and* X^c as

$$bd(X) = \{(p, q) \in D \times D \mid p \in X,\ q \in X^c,\ \text{and } p \text{ and } q \text{ are face-adjacent}\}.$$

Note that the continuous analog $CA(bd(X))$ of the digital boundary $bd(X)$ between X and X^c is precisely the (topological) boundary of $CA(X)$ in $\mathbb{R}^3$. From now on, we denote $CA(bd(X))$ by $bdCA(X)$.

Assume that D is the domain of a binary image, (D, X), whose foreground X is a nonempty and finite set. It can be shown that the collection of faces of $bd(X)$, along with their vertices and edges, consists of a pure 2-polytopal complex with empty boundary [76]. Furthermore, the underlying space of this polytopal complex (i.e., $bdCA(X)$) separates the continuous analog of the maximal 6-connected components of X and X^c, i.e., any curve connecting a point in the interior of the continuous analog of a maximal 6-connected component of X or X^c to its exterior must cross $bdCA(X)$ [65].

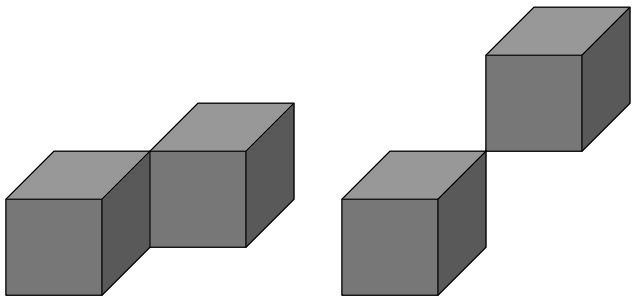
4.2 Three-Dimensional Well-Composed Images

Let (D, X) be a 3D binary digital image, and assume that X is a nonempty and finite subset of D. We say that (D, X) is *well-composed* if $bdCA(X)$ is a surface in $\mathbb{R}^3$, or

equivalently, if the pure 2-polytopal complex consisting of the faces of $bd(X)$, along with their vertices and edges, does not contain any singularity. Interestingly, well-composed images can be characterized by two local conditions depending only on points of (D, X) [79]. A 3D binary digital image (D, X) satisfies these two conditions if and only if (D, X) is well-composed, i.e., if and only if $bdCA(X)$ is a surface in $\mathbb{R}^3$.

Definition 4.2.1. Let Y be a any subset of four points of D. We say that Y is an instance of the *critical configuration (C1) in* (D, X) if the voxels of the four points of Y share an edge, two of them are in $CA(X)$ and the other two are in $CA(X^c)$, and the two voxels in $CA(X)$ (resp. $CA(X^c)$) are edge-adjacent.

Refer to Figure 4.2(a) for an illustration of Definition 4.2.1.



(a) (b)

Figure 4.2: (a) Instance of the critical configuration (C1). Only the two voxels of X (resp. X^c) are shown. (b) Instance of the critical configuration (C2). Only the two voxels of X (resp. X^c) are shown.

Definition 4.2.2. let Y be any subset of eight points of D. We say that Y is an instance of the *critical configuration (C2) in* (D, X) if the voxels of the eight points of Y share a vertex, two of them are in $CA(X)$ (resp. $CA(X^c)$) and the other six are in $CA(X^c)$ (resp. $CA(X)$), and the two voxels in $CA(X)$ (resp. $CA(X^c)$) are corner-adjacent.

Refer to Figure 4.2(a) for an illustration of Definition 4.2.2.

The following theorem stated and proved by Latecki in [79] establishes an important equivalence between the well-composedness property of a 3D binary digital image and the nonexistence of instances of critical configurations (C1) and (C2) in the image:

Theorem 4.2.1. *Let (D, X) be any 3D binary digital image such that X be any nonempty and finite subset of D. Then, the image (D, X) is well-composed if and only if it does not contain any instance of the critical configuration (C1) nor any instance of the critical configuration (C2).*

Theorem 4.2.1 also implies that there is only one kind of connectedness in well-composed images, i.e., if (D, X) is well-composed then any 26-connected component of X or X^c is also a 18-connected component, which in turn is also a 6-connected component. This is due to the fact that, if a 18-connected component of X is not 6-connected, then there must exist an instance of (C1) in (D, X). Similarly, if a 26-connected component of X is not 18-connected, then there must exist an instance of (C2) in (D, X).

4.3 Surface Meshes from Images

Three-dimensional digital images are used in several important applications, such as medical imaging, fluid dynamics simulation, and videoconferencing. In such applications, it is often needed to visualize the image [23]. If the image is binary, a common approach to visualize it is to extract and render the continuous analog of the digital boundary between background and foreground [58]. However, this approach has two major drawbacks.

First, the large number of vertices, edges, and faces of a piecewise linear representation of the continuous analog may slow down visualization and manipulation algorithms. Second, due to the nature of the digital geometry, the continuous analog

is not suitable for representing arbitrarily oriented "curved" objects without suffering from the "staircase" appearance artifact. Figure 4.3 below illustrates both problems:

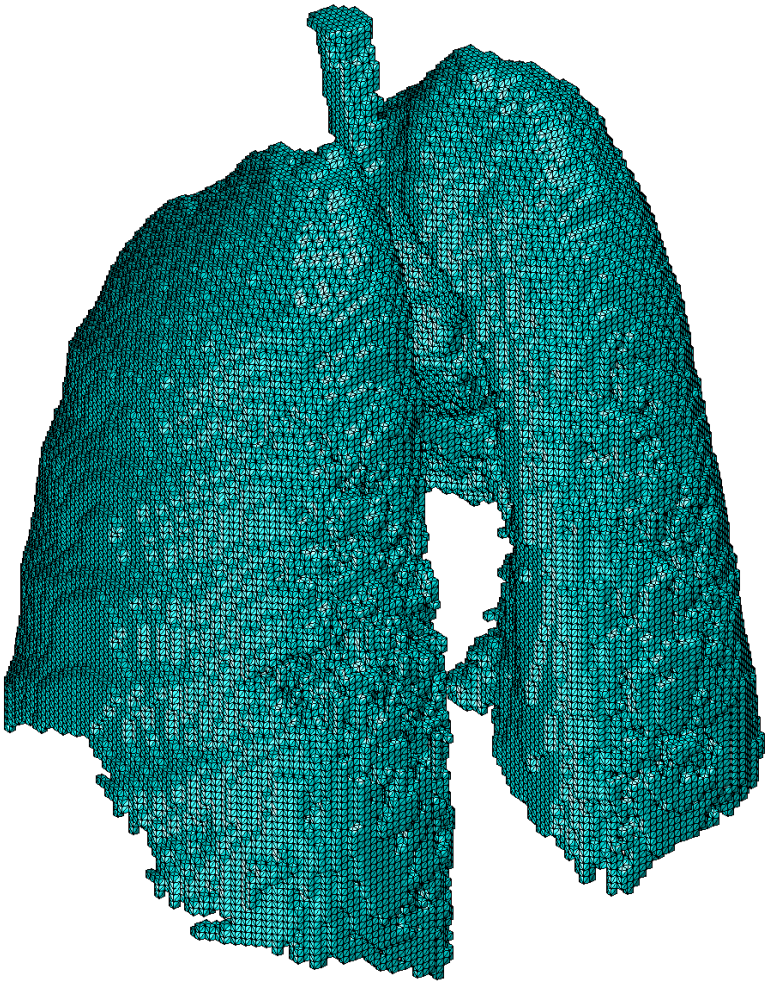
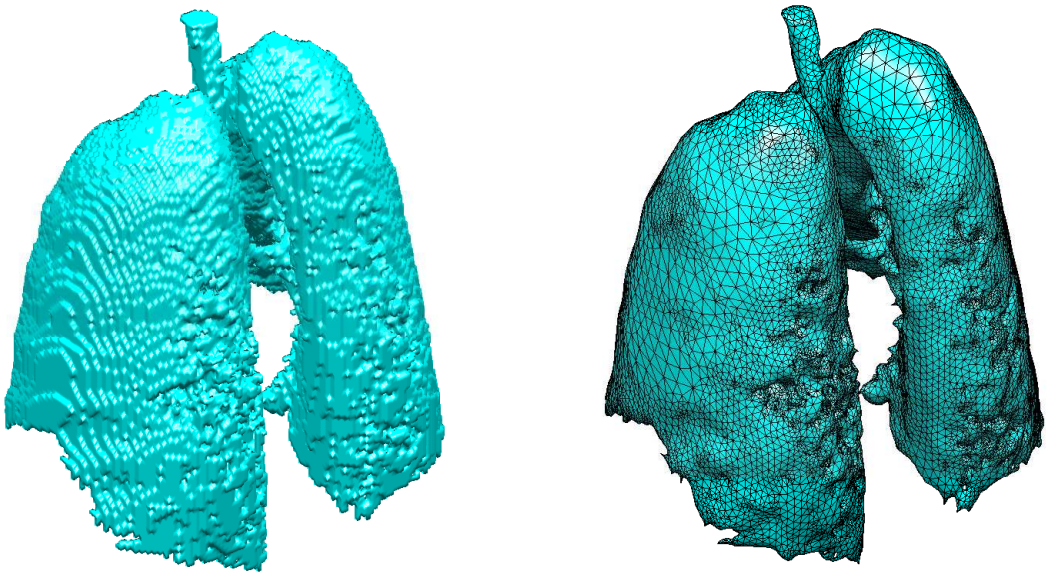


Figure 4.3: A piecewise-linear representation of the continuous analog of the digital boundary between the foreground and background of a binary image of a lung during inspiration.

Several different solutions have been proposed for both problems over the past 20 years [87, 117, 55, 139, 52, 73, 77]. A popular solution is to approximate the continuous analog by an adaptive and "smoother" surface, which is in general the

underlying surface of a simplicial surface [87, 117, 139, 52, 77] (see Figure 4.4). As we have seen in Section 4.1, the continuous analog of the digital boundary between the foreground and background of a binary image is not always a surface. Indeed, it is the underlying space of a pure two-dimensional polytopal complex with empty boundary, which may have singularities (see Section 4.2).



(a) (b)

Figure 4.4: (a) The surface in Figure 4.3 without the edges in black. (b) A simplicial surface that approximates the surface in (a) and does not contain the "staircase" appearance.

Although it may seem awkward to approximate a non-manifold by a surface, this is justified in practice for three main reasons. First, the fact that the continuous analog is not a surface is quite often the result of the lack of topology-preservation during the image formation process [61]. Besides, the problem of using the continuous analog to recover the correct topology of the surface of the object represented by the image is ill-posed. Second, there are several available data structures to represent

piecewise linear surface, as well as efficient algorithms for manipulating them. Third, for many applications, the approximation of a non-manifold by a surface may be entirely satisfactory, as long as the surface and the non-manifold are geometrically "close" to each other.

A piecewise-linear surface, such as the ones in Figure 4.3 and Figure 4.4(b), is called a *surface mesh*. More precisely, a *surface mesh* is a pure two-dimensional polytopal complex with empty boundary whose underlying space is a surface (see Appendix A). Here, we restrict our attention to simplicial surfaces, i.e., surface meshes whose 2-faces are triangles. Given a 3D binary digital image (D, X), we define *the surface mesh problem* as the one of obtaining a surface mesh whose underlying surface "approximates" $bdCA(X)$[1]. This problem is not precisely specified until we formalize the meaning of "approximates".

Here, we restrict our attention to 3D well-composed images. This makes it possible to precisely state the surface mesh problem, as $bdCA(X)$ is always a surface. So, by "approximates", we mean a surface mesh $\mathcal{M}$ such that its underlying surface, $|\mathcal{M}|$, is homeomorphic to $bdCA(X)$. In addition, we may require the triangles of $\mathcal{M}$ to have good aspect ratio, and the surface $|\mathcal{M}|$ to be a geometrically close approximation of $bdCA(X)$. The notion of geometrically close can be formalized as follows: there exists a homeomorphism $h : bdCA(X) \rightarrow |\mathcal{M}|$ such that, for every $p \in bdCA(X)$, we have $|h(p) - p| \leq \epsilon$, where ϵ is a given fixed and small positive real number.

A triangle is said to have good aspect ratio, or equivalently, to be well-shaped, if its smallest (resp. largest) angle is bounded away from 0^o (resp. 180^o). The aspect ratio of a triangle can be measured in terms of a shape metric called radius-edge ratio:

[1]This problem is commonly referred to as the *iso-surface extraction problem* in the graphics literature.

Definition 4.3.1. The *radius-edge ratio*, $\rho(t)$, of a triangle t is given by

$$\rho(t) = \frac{r}{l},$$

where r is the circumradius of t and l is the length of its shortest edge.

A triangle's radius-edge ratio is related to its smallest angle, θ, by the formula $r/l = 1/(2\sin\theta)$. So, an upper bound on the the triangle's radius-edge ratio implies a lower bound on its smallest angle. Since a lower bound on the smallest angle of a triangle implies an upper bound on its largest angle (the converse is not true), if a triangle's radius-edge ratio is small, its smallest (resp. largest) angle is bounded away from 0^o (resp. 180^o). Surface meshes in which all triangles have good aspect ratio are very desirable for rendering purposes and physical simulations based on the finite element method.

4.4 Our Solution and Its Contributions

Our solution for the surface mesh problem consists of three parts. First, we convert the input 3D binary digital image (D, X) into a well-composed image (D, X') such that $X \subseteq X'$. Next, we define an implicit function $f : conv(D) \rightarrow \mathbb{R}$ such that the zero level set $f^{-1}(0)$ of f is a smooth surface homeomorphic to $bdCA(X')$. Finally, by using $f^{-1}(0)$, we generate a simplicial surface $\mathcal{M}$ such that $|\mathcal{M}|$ is homeomorphic to $bdCA(X')$.

The algorithm used in the third and last step of our solution is a simplification of an algorithm developed by Chen, Dey, Ramos, and Ray [26]. To our best knowledge, the latter is one of two existing algorithms that is provably guaranteed to generate well-shaped triangles. The other algorithm was developed by Boissonnat and Oudot [15]. While the algorithm in [26] does not provide any guarantee on how close $|\mathcal{M}|$ and $bdCA(X')$ are, the algorithm in [15] does. However, the theoretical guarantees of [15] rely upon knowledge of the medial axis of the input implicit surface. Unfortunately, there is no known algorithm for computing the medial axis of

an arbitrary surface [22].

The fact that $\mathcal{M}$ is generated from $f^{-1}(0)$, rather than directly from $bdCA(X')$, may seem redundant. Indeed, an algorithm for computing $\mathcal{M}$ directly from $bdCA(X')$ has been developed [77], and one could also use some *mesh simplification* algorithm instead. However, the algorithm in [77] does not provide any guarantee on the shape of the triangles of $\mathcal{M}$. Besides, to our best knowledge, the only known mesh simplification algorithm that provides such a guarantee requires the input surface mesh to satisfy some constraints which rule out surface meshes such as the ones arising from imaging data [36].

Our simplification of [26] consists of replacing an algorithm step for computing critical and silhouette points of the surface $f^{-1}(0)$ by simple topological and geometric operations to detect the presence of handles of $f^{-1}(0)$ inside Voronoi regions of a given Voronoi diagram. This simplification makes the algorithm practical, as the computation of critical and silhouette points involves the computation of local minima and maxima of $f^{-1}(0)$, which is an expensive operation subject to geometric degeneracies. The authors of [26] themselves did not implement such computation, and we strongly believe that their algorithm did not become popular due to these computations.

The simplification of the algorithm in [26] was made possible by the first and second steps of our solution. More specifically, these two steps enable us to apply relatively simple topological and geometric operations to find out information about the topology of $f^{-1}(0)$, which is the same as the topology of $bdCA(X')$, and also to detect the presence of handles of $f^{-1}(0)$ inside Voronoi regions of a given Voronoi diagram.

Although the first and second steps of our solution were motivated by the simplification of the algorithm in [26], each of these steps can be used by (un)related algorithms as well. For instance, the algorithm for converting an "ill-composed" 3D binary digital image, (D, X), into a well-composed image, (D, X'), can be used

by any algorithm that assumes that $bdCA(X)$ is a surface, such as algorithms for computing discrete curvature [128] and algorithms for approximating $bdCA(X)$ directly from (D, X') [77]. Likewise, our approach to build a smooth surface, $f^{-1}(0)$, that is homeomorphic to $bdCA(X')$ yields an analytic expression for the function f. This expression can be used for computing higher order derivatives and for realistic rendering.

Chapter 5

Well-Composed Images

This chapter presents an algorithm for converting "ill-composed" 3D binary digital images into well-composed ones. The basic idea of this algorithm is to change the color assigned to some points of the input ill-composed image so that the resulting image is well-composed. This algorithm is used by the first step of our solution for the problem of generating a surface mesh from a given 3D binary digital image (see Chapter 4).

5.1 Preliminaries and Related Work

As we have seen in Chapter 4, our solution for the surface mesh problem relies upon the fact that the continuous analog of the digital boundary between the foreground and background of the input image is a surface, or equivalently, the input image is well-composed. Since most images encountered in practice are likely to be ill-composed, we developed an algorithm for converting 3D ill-composed binary images into well-composed ones.

Our algorithm works by changing the color of some points of the input image from 0 to 1, i.e., by making some background points into foreground ones. We measure the success of our algorithm by the similarity between the input and output images.

By similarity, we mean the ratio between the number of points of the input image whose color changed and the number of points of the image. The smaller this ratio the better.

The idea of changing the color of background points to produce a well-composed image is due to Latecki [80], who proposed a similar algorithm for handling 2D binary digital images[1]. Unfortunately, his algorithm cannot be easily extended to deal with 3D binary images. Later, Rosenfeld, Kong, and Nakamura [115] introduced an image operation, called simple deformation, that can also be used for making 2D binary digital images well-composed. This operation can be extended to deal with 3D images is a straightforward manner, but the resulting well-composed images are 27 times bigger than their corresponding ill-composed ones, which rules out its use in practice.

Our algorithm can also be used by image-based applications that can benefit from dealing with well-composed images. For instance, thinning algorithms do not suffer from the irreducible thickness problem if the image is well-composed [81], and thinning well-composed gray-scale images leads not only to theoretically better understood algorithms but also to practically relevant thinning and segmentation algorithms [91]. Algorithms for computing surface curvature ([128]) from 3D binary images can also be preceded by our algorithm in order to ensure that the continuous analog of the digital boundary between foreground and background is indeed a surface.

5.2 Description of the Algorithm

Our algorithm for converting 3D binary digital images that are not well-composed into well-composed ones takes as input a 3D binary digital image (D, X), where D is assumed to be a finite orthogonal grid as described in Section 4.1. The output is a 3D

[1] A 2D binary digital image (D, X) is well-composed if and only if $bdCA(X)$ is a curve.

well-composed image (D, X') such that $X = X'$ if (D, X) is already well-composed, and $X \subset X'$ otherwise. In other words, if (D, X) is not well-composed, then the foreground X'_1 of (D, X') is a proper superset of the foreground X_1 of (D, X), or equivalently, the background X'_0 of (D, X') is a proper subset of the background X_0 of (D, X).

The set X' is computed by finding a subset P of X_0 such that the image (D, X'), with $X' = X \cup P$, is well-composed. So, the set P can be viewed as the subset of the background of (D, X) whose assigned colors are changed from 0 to 1 to generate (D, X'). Note that such a set P always exists, as (D, X') is well-composed if we let $P = X_0$. However, since we want (D, X) and (D, X') to be as similar as possible, we ideally would like to find a smallest or *optimal* set P, with $P \subseteq X_0$, such that (D, X') is well-composed.

Since D is a finite set, the background X_0 of (D, X) is also finite. Hence, there is a very simple algorithm for finding a smallest P: Enumerate and test all subsets P of X_0 in increasing order of cardinality until a well-composed image $(D, X' = X \cup P)$ is found. Recall that we can determine if $(D, X' = X \cup P)$ is well-composed by checking the existence of an instance of (C1) or (C2) in $(\mathbb{Z}^3, X')$ (see Figure 4.2). But, because X_0 has $2^{|X_0|}$ subsets, where X_0 is the cardinality of X_0, and X_0 has typically one or two million points in images encountered in practice, this algorithm is impractical.

By realizing that critical configurations are *local* configurations of points of D, we devised a practical alternative to this naïve solution. Our algorithm is linear in the size of D, but it is not guaranteed to find a smallest P such that $(D, X' = X \cup P)$ is well-composed. However, our experiments in Section 5.5 show empirical evidences that our algorithm is very effective on typical practical data, and Section 5.4 derives an upper bound for the expected size of the set P that confirms the results described in Section 5.5.

Our algorithm starts by letting $P = \emptyset$. Then, it loops over all points in D to find

all instances of (C1) and (C2) in (D, X). If there is no instance of (C1) or (C2) in (D, X), the algorithm terminates. Otherwise, the algorithm eliminates all instances of (C1) and (C2) found in (D, X), and also the ones that may eventually arise during this elimination process.

The elimination of critical configurations is carried out by inserting points from $X_0 - P$ into P one at a time. Each point p inserted into P eliminates at least one instance of (C1) or (C2) from $(D, X \cup (P - \{p\}))$. However, this operation may also give rise to *new* critical configurations in $(D, X \cup P)$, which will trigger the insertion of at least one more point from $X_0 - P$ into P, and so on so forth. In the following three sections, we elaborate on the details of the steps of our algorithm.

5.2.1 Looking for Critical Configurations

This sections describes the details of the first step of our algorithm, which initializes P and finds all instances of (C1) and (C2) in (D, X). Recall that

$$D = \{p = (p_1, p_2, p_3) \in \mathbb{R}^3 \mid p = O + \delta \cdot g_i\},$$

where $O = (o_1, o_2, o_3) \in \mathbb{R}^3$, $\delta \in \mathbb{R}$, and $g = (g_1, g_2, g_3) \in \{0, n_1\} \times \{0, n_2\} \times \{0, n_3\}$, for some positive integers n_1, n_2, and n_3. For each $j \in \{0, \ldots, n_1 - 1\}$, $k \in \{0, \ldots, n_2 - 1\}$, and $l \in \{0, \ldots, n_3 - 1\}$, we consider the point $r \in D$ with coordinates $r = (r_1, r_2, r_3) = (o_1 + \delta \cdot j, o_2 + \delta \cdot k, o_3 + \delta \cdot l)$, and we define the subsets of D:

$$\begin{aligned}
\mathcal{N}_x(r) &= \{(r_1, y, z) \mid y \in \{r_2, r_2 + \delta\} \text{ and } z \in \{r_3, r_3 + \delta\}\}, \\
\mathcal{N}_y(r) &= \{(x, r_2, z) \mid x \in \{r_1, r_1 + \delta\} \text{ and } z \in \{r_3, r_3 + \delta\}\}, \\
\mathcal{N}_z(r) &= \{(x, y, r_3) \mid x \in \{r_1, r_1 + \delta\} \text{ and } y \in \{r_2, r_2 + \delta\}\},
\end{aligned}$$

and

$$\mathcal{N}(r) = \{r_1, r_1 + \delta\} \times \{r_2, r_2 + \delta\} \times \{r_3, r_3 + \delta\}.$$

Note that each instance of (C1) in $(D, X \cup P)$ is one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$, for some $r \in D$. Likewise, each instance of (C2) in $(D, X \cup P)$ is a set $\mathcal{N}(r)$, for

some $r \in D$. Furthermore, if $\mathcal{N}(r)$ is an instance of (C2) in $(D, X \cup P)$, then none of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in $(D, X \cup P)$. Conversely, if any of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in $(D, X \cup P)$, then $\mathcal{N}(r)$ is not an instance of (C2) in $(D, X \cup P)$.

Our algorithm starts by letting P be the empty set, and then it loops over D to find all instances of (C1) and (C2) in (D, X). For each point $r \in D$, the algorithm inserts r into an initially empty queue, R, if and only if any of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in (D, X) or if $\mathcal{N}(r)$ is an instance of (C2) in (D, X). The queue R is initially empty. Figure 5.1 shows the pseudo code for the first step of our algorithm. After the first step is over, if the queue R is still empty, then the algorithm terminates. Otherwise, the algorithm starts eliminating the critical configurations from (D, X).

```
P ← ∅
R ← ∅
for each r ∈ D do
    if any of N_x(r), N_y(r) and N_z(r) is an instance of (C1) then
        insert r into R
    else if N(r) is an instance of (C2) then
        insert r into R
    endif
endfor
```

Figure 5.1: Pseudo code for the first step of our algorithm.

5.2.2 Removing Critical Configurations

If the queue R is not empty after the first step is over, the algorithm carries out its second and last step. This step removes all critical configurations from $(D, X \cup P)$. This is done by removing one point r from R at a time, and then considers *all* instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ or the instance of (C2) in $\mathcal{N}(r)$. By inserting some points from $X_0 - P$ into P, our algorithm eliminates the instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ from $(D, X \cup P)$ or the instance $\mathcal{N}(r)$ of (C2) from $(D, X \cup P)$. The choice of points from $X_0 - P$ to insert into P is described in the next section.

If the insertion of points from $X_0 - P$ into P also gives rise to any *new* critical configuration in $(D, X \cup P)$, our algorithm finds every point r of D such that any of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ is a new critical configuration, and inserts r into R. Next, the algorithm removes another point r from R and considers *all* instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ or the instance of (C2) in $\mathcal{N}(r)$, just like before. This process is carried out until the queue R becomes empty. Whenever R becomes empty, the algorithm terminates. Figure 5.2 shows the pseudo code for this second and last step of our algorithm.

5.2.3 Choosing Points from the Background

This section describes which points from the background can be chosen by our algorithm to eliminate a given instance of (C1) or (C2). First, consider an instance Y of (C1) in $(D, X \cup P)$, and refer to Figure 5.3. The set Y has four points, two of them, say a and b, are from $X_0 - P$ and two of them are in $X \cup P$. Note that the insertion of either a or b into P eliminates Y from $(D, X \cup P)$. On the other hand, points a and b are the only points from $X_0 - P$ that can be inserted into P in order to eliminate Y from $(D, X \cup P)$. So, our choice of a point from $X_0 - P$ to eliminate Y from $(D, X \cup P)$ is restricted to the two background points, a and b, of the critical configuration Y.

Now, consider an instance Y of (C2) in $(D, X \cup P)$, and refer to Figure 5.4. The set Y has eight points, two of them are in $X_0 - P$ (resp. $X \cup P$) and the other six are in $X \cup P$ (resp. $X_0 - P$). First, consider the case in which Y has exactly two points from $X_0 - P$, say a and b, as illustrated by Figure 5.4. Note that the insertion of either a or b into P eliminates Y from $(D, X \cup P)$. Furthermore, points a and b are the only ones from $X_0 - P$ that can be inserted into P in order to eliminate the critical configuration Y from $(D, X \cup P)$. So, once again our choice of a point from $X_0 - P$ in order to eliminate Y from $(D, X \cup P)$ is restricted to the two background points, a and b, of Y.

while R is not empty **do**
    remove a point r from R
    if any of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) **then**
        insert points $X_0 - P$ into P to eliminate all instances of (C1)
        $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$
    else if $\mathcal{N}(r)$ is an instance of (C2) **then**
        insert points $X_0 - P$ into P to eliminate $\mathcal{N}(r)$
    endif
    for each critical new configuration $\mathcal{N}_x(p)$, $\mathcal{N}_y(p)$, $\mathcal{N}_z(p)$, or $\mathcal{N}(p)$ **do**
        insert y into R
    endfor
endwhile

Figure 5.2: Pseudo code for the second and last step of our algorithm.

Next, consider the case in which the critical configuration Y has six points from $X_0 - P$. Note that the insertion of any of these points into P eliminates Y from $(D, X \cup P)$. Note also that this insertion always gives rise to an instance of (C1) in

$(D, X \cup P)$. This situation is illustrated in Figure 5.5. On the other hand, only the insertion of one of the six points in $Y \cap (X_0 - P)$ can eliminate Y from $(D, X \cup P)$. So, unlike the two previous cases, our choice of a point from $X_0 - P$ in order to eliminate Y from $(D, X \cup P)$ is restricted to six points, which are the six background points of Y.

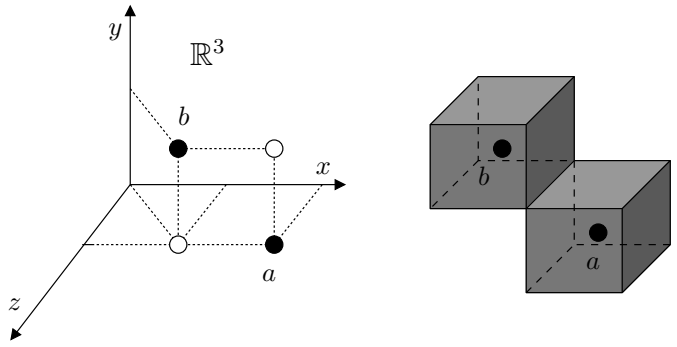


Figure 5.3: An instance Y of (C1). Points a and b are the background points of Y.

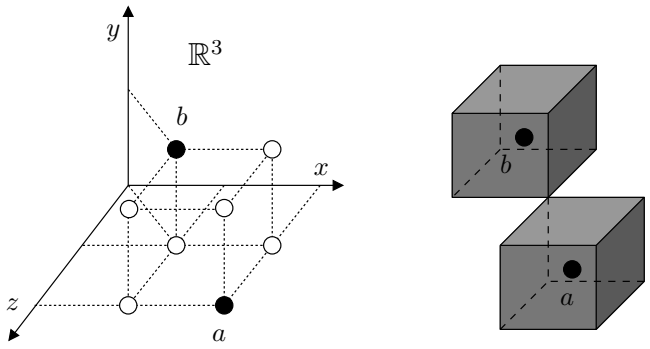


Figure 5.4: An instance Y of (C2). Points a and b are the background points of Y.

We are now ready to explain the choice of points from $X_0 - P$ to insert into P in order to eliminate critical configurations from $(D, X \cup P)$. Let r be a point of D for which one of the subsets $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ of D is an instance of (C1) in $(D, X \cup P)$.

If there is exactly one instance of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$, say Y, then

the algorithm inserts one of the two background points of Y into P, as we discussed before (see Figure 5.3). Our algorithm applies one of three rules to decide which point to choose. These rules are explained in the next section. However, regardless of the point chosen, all instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ are eliminated from $(D, X \cup P)$.

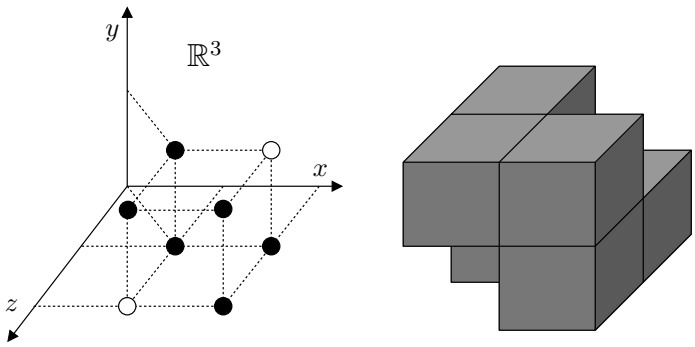


Figure 5.5: An instance Y of (C2). The six background points of Y are represented by black circles.

If there are exactly two instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$, say Y_1 and Y_2, then Y_1 and Y_2 have exactly *three* points of $X_0 - P'$, as any two instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ share two points, one of which is a point of $X_0 - P'$ and the other is a point of $X \cup P'$, as shown in Figure 5.6. In order to eliminate Y_1 and Y_2 from $(D, X \cup P)$, our algorithm inserts into P either the background point shared by Y_1 and Y_2 (point a in Figure 5.6) or the other two background points of $Y_1 \cup Y_2$ (points b and c in Figure 5.6). The exact choice of points is decided by the same three rules explained in the next section. Note that all instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ are eliminated from $(D, X \cup P)$ by either one of the two possible choices.

If $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains three instances of (C1) in $(D, X \cup P)$, then $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ has either three or four points of $X_0 - P$, which are background points of $(D, X \cup P)$ (see Figure 5.7). If $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains exactly four

points of $X_0 - P$, then our algorithm inserts either the common background point of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ (point a in Figure 5.7) or the other three background points of $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ into P (points b, c and d in Figure 5.7). Otherwise, $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ contains exactly three points of $X_0 - P$ (points b, c and d in Figure 5.7), and the algorithm inserts two of these three points into P. The exact choice of points is decided by the same three rules explained in the next section. Note that all instances of (C1) in $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ are eliminated from $(D, X \cup P)$ by either one of the two possible choices.

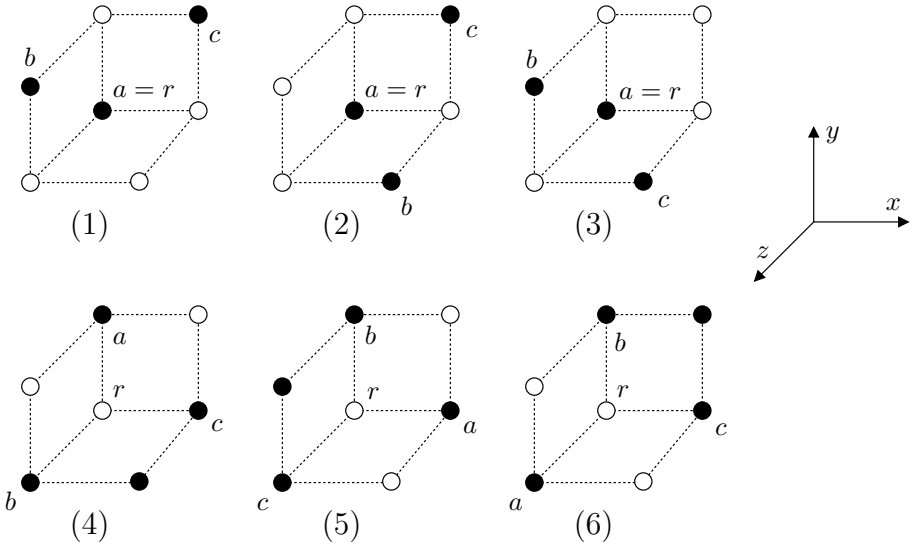


Figure 5.6: All possible cases in which the set $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains exactly two instances of (C1) in $(D, X \cup P)$. Points in $X \cup P$ are represented by white circles, while points in $X_0 - P$ are represented by black circles.

Finally, let r be a point of D such that $\mathcal{N}(r)$ is an instance of (C2) in $(D, X \cup P)$. As shown by Figure 5.4 and Figure 5.5, the set $\mathcal{N}(r)$ contains either two or six (background) points of $X_0 - P$. In either case, our repairing algorithm inserts only one of these points into P to eliminate the instance of (C2) in $\mathcal{N}(r)$ from $(D, X \cup P)$. The point is also chosen according to the three rules described in the next section. Recall that, if the set $\mathcal{N}(r)$ contains six (background) points from $X_0 - P$, then

the insertion of any of these six points into P always gives rise to a new critical configuration in $(D, X \cup P)$.

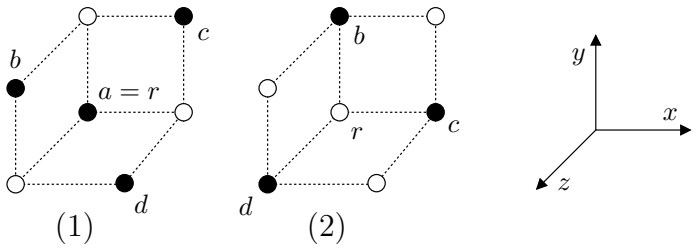


Figure 5.7: The two cases in which the set $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains three instances of (C1) in $(D, X \cup P)$. Points in $X \cup P$ are represented by white circles, while points in $X_0 - P$ are represented by black circles.

5.2.4 Rules for Choosing Points

Our repairing algorithm chooses one or more points from $X_0 - P$ to insert into P according to three rules. These rules are mutually exclusive. Before we describe the rules, we define three sets of subsets of points of $(X_0 - P) \cap \mathcal{N}(r)$ used in our description of the rules: $B(r)$, $B_1(r)$, and $B_2(r)$. We define $B(r)$ as the set of all subsets of points of $X_0 - P$ that our algorithm can insert into P in order to eliminate *all* instances of (C1) in $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$ or the instance $\mathcal{N}(r)$ of (C2), if any. So, we have the following possibilities:

- If $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains exactly one instance Y of (C1) in $(D, X \cup P)$, then $B(r) = \{\{a\}, \{b\}\}$, where a and b are the two background points of Y in $(X_0 - P)$.

- If $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains exactly two instances of (C1) in $(D, X \cup P)$, say Y_1 and Y_2, then $B(r) = \{\{a\}, \{b, c\}\}$, where a is the common point of Y_1 and Y_2 and $X_0 - P$, and b and c are the two other points of $X_0 - P$ in $Y_1 \cup Y_2$ (see Figure 5.6).

- If $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains three instances of (C1) in $(D, X \cup P)$, then we have two situations: If the common point a of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is in $X_0 - P$, then $B(r) = \{\{a\}, \{b, c, d\}\}$, where b, c, and d are the other background points of $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ in $X_0 - P$ (see Figure 5.7). Otherwise, $B(r) = \{\{b, c\}, \{b, d\}, \{c, d\}\}$, where b, c, and d are the points of the three background points of $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ in $X_0 - P$ (see Figure 5.7).

- Finally, if $\mathcal{N}(r)$ contains an instance of (C2) in $(D, X \cup P)$, then we also have two situations: If $\mathcal{N}(r)$ has exactly two points a and b of $X_0 - P$, then $B(r) = \{\{a\}, \{b\}\}$. Otherwise, $B(r) = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}\}$, where a, b, c, d, e, and f are the six background points of $\mathcal{N}(r)$ in $X_0 - P$.

We also define the partition $\{B_1(r), B_2(r)\}$ of $B(r)$ such that $S \in B_1(r)$ if and only if $S \in B(r)$ and the insertion of all elements of S into P does not give rise to any critical configuration in $(D, X \cup P)$, and $S \in B_2(r)$ if and only if $S \in B(r)$ and the insertion of all elements of S into P gives rise to at least one critical configuration in $(D, X \cup P)$. For each $r \in D$, if one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ is an instance of a critical configuration, our repairing algorithm chooses a set S from $B(r)$ according to the following three rules:

(1) If $B_1(r)$ is a singleton set, then let the set S be the single element of $B_1(r)$ and insert the elements of S into P.

(2) If $B_1(r)$ has two or more elements, then select a set S from $B_1(r)$ at random and insert the elements of S into P.

(3) If $B_1(r)$ is the empty set, or equivalently, if $B_2(r) = B(r)$, then select a set S from $B_2(r)$ at random and insert the elements of S into P.

Note that a new critical configuration in $(D, X \cup P)$ is created only if rule (3) is applied. Whenever rule (3) is applied, our algorithm finds every point $p \in D$ such that at least one of the sets $\mathcal{N}_x(p)$, $\mathcal{N}_y(p)$, $\mathcal{N}_z(p)$, and $\mathcal{N}(p)$ is a critical configuration

created by inserting the points of S into P, and then inserts p in the queue R, as shown by the pseudo code in Figure 5.2. Fortunately, we do not have to loop over D to find these points. Since every point inserted into P by rule (3) is a point of $\mathcal{N}(r)$, we just have to look for the new critical configurations in the grid $[k-1, k+2] \times [j-1, j+2] \times [l-1, l+2] \subset \mathbb{Z}^3$, where (j, k, l) are the integer coordinates of point r.

5.3 Termination, Correctness and Complexity

By examining the pseudo code for our algorithm (Figure 5.1 and Figure 5.2), we can conclude that our algorithm terminates if and only if the queue R eventually becomes empty. Recall that R is empty in the beginning of the first step, and that a point is inserted into R only if (D, X) contains a critical configuration. Since no point is removed from R in the first step, the queue R is empty at the end of the first step if and only if (D, X) is well-composed. So, if R is empty at the end of the first step, our algorithm terminates and outputs a well-composed image, which is (D, X) itself.

Assume that R is not empty after the first step is over. So, (D, X) is not well-composed and the algorithm correctly starts its second and last step. Each iteration of the second step removes exactly one point from R, and it may or may not insert more points in R. So, since all points inserted into and removed from R are points of D and D is a finite set, the algorithm terminates if the same point of D is inserted into R finitely many times only. It turns out that this is the case, as a point $r \in D$ is inserted into R at most four times. This claim follows from the following observations:

- The point r is inserted into R if and only if either any of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in $(D, X \cup P)$ or $\mathcal{N}(r)$ is an instance of (C2) in $(D, X \cup P)$.

- If a critical configuration Y is removed from $(D, X \cup P)$, then the same critical configuration Y cannot occur again in $(D, X \cup P)$, as our algorithm does not remove points from P.

So, our algorithm always terminates, as the queue R becomes empty after finitely many steps.

The correctness of the second and last step of our algorithm is an immediate consequence of the termination condition, as the image $(D, X \cup P)$ is well-composed whenever R becomes empty. To see why, recall that for every critical configuration Y of (D, X), the first step inserts a point r into R such that Y is one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$. In the second step, for every point r in R, the algorithm removes *all* instances of critical configurations in $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ from $(D, X \cup P)$.

Furthermore, if a new critical configuration Y arises, then a point r is inserted into R such that Y is one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$. So, at any given time, for every critical configuration Y of $(D, X \cup P)$, there exists a point r in R such that Y is one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$. Hence, whenever R becomes empty, we can conclude that $(D, X \cup P)$ does not contain any instance of a critical configuration.

Our algorithm is linear in the number $|D|$ of points of D. The first step is clearly $\mathcal{O}(|D|)$, as it considers every point of D once (see Figure 5.1). The second step is basically a loop that iterates until the queue R becomes empty. Each iteration is clearly $\mathcal{O}(1)$. So, the time complexity of the second step of our algorithm is given by the number of iterations of its loop. This number is certainly bounded by $4 \times |D|$, as each point of D can be inserted into R at most four times, and each iteration removes one point from R. So, the time complexity of our algorithm is indeed linear in the number $|D|$ of points of D.

5.4 An Upper Bound for the Size of P

Although our algorithm is linear in the size of the image domain, it is unlikely to produce a smallest set P such that $(D, X \cup P)$ is well-composed. Actually, in principle, this set P can be the entire set X_0 of background points of the input image (D, X). So, it is natural to ask ourselves how poor our algorithm can perform. This section addresses this question, and derives a probabilistic upper bound for the size of the set P computed by our algorithm.

Recall that each point inserted into P by our algorithm eliminates at least one critical configuration of $(D, X \cup P)$. So, the size of P is bounded above by $m + t$, where m is the number of critical configurations in the input image (D, X), and t is the number of critical configurations created (and later removed) by our algorithm in order to produce a well-composed image $(D, X \cup P)$. So, by deriving an upper bound for $m + t$, we also obtain an upper bound for the size of P. Here, we view m and t as an intrinsic feature of the input image and an intrinsic feature of our algorithm, respectively, and we obtain an upper bound for the expected value of t in terms of m.

From the description of our algorithm in Section 5.2, a new critical configuration is created if and only if rule (3) is applied to eliminate an existing critical configuration. We also know that any critical configuration in (D, X) or any new critical configuration created by our algorithm is one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$, for some $r \in D$. Since $\mathcal{N}_x(r_1) \neq \mathcal{N}_x(r_2)$, $\mathcal{N}_y(r_1) \neq \mathcal{N}_y(r_2)$, $\mathcal{N}_z(r_1) \neq \mathcal{N}_z(r_2)$, and $\mathcal{N}(r_1) \neq \mathcal{N}(r_2)$, for any two distinct points r_1 and r_2, we can express t as $\sum_{r \in D} C(r)$, where $C(r)$ is the number of new critical configurations resulting from the elimination of *all* instances of (C1) in $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$, or the instance of (C2) in $\mathcal{N}(r)$.

If we consider $C(r)$ as a random variable, then the expected value $E[t]$ of t is $E[t] = E[\sum_{r \in D} C(r)] = \sum_{r \in D} E[C(r)]$, where $E[C(r)]$ is the expected value of $C(r)$. By definition, the expected value $E[C(r)]$ of $C(r)$ is given by $E[C(r)] =$

$\sum_{i=1}^{K} i \cdot P[C(r) = i]$, where $P[C(r) = i]$ is the probability that $C(r)$ will have value i, and K is the largest value of $C(r)$. So, if we can obtain an expression for $P[C(r) = i]$, for each $1 \leq i \leq K$, we can provide an expression for the expected values $E[C(r)]$ and $E[t]$ of $C(r)$ and $E[t]$, respectively.

Suppose that rule (3) is used to eliminate an instance of (C1) in $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, or the instance of (C2) in $\mathcal{N}(r)$, for some $r \in D$. Let S be the set of points in $B(r)$ selected by rule (3) and whose elements are inserted into P. We know that every point $q \in S$ is a point of $\mathcal{N}(r)$, and that every new critical configuration created by the insertion of q into P is in the set $G(r) = [j-1, j+2] \times [k-1, k+2] \times [l-1, l+2]$ of points of $\mathbb{Z}^3$, where (j, k, l) are the integer coordinates of r. Note that $G(r)$ is a $4 \times 4 \times 4$ grid of points of $\mathbb{Z}^3$ whose interior points are exactly the points of $\mathcal{N}(r)$. So, we can compute $P[C(r) = i]$ by restricting our attention to the set $G(r)$, which contains exactly 64 points.

Let A be the set of all distinct binary images $(G(r), W)$. Since each point in $G(r)$ is assigned a color from the binary set $\{0, 1\}$, the cardinality $|A|$ is 2^{64}. We are interested in the subset A' of A such that $(G(r), W) \in A'$ if and only if $(G(r), W) \in A$ and at least one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in $(G(r), W)$ or $\mathcal{N}(r)$ is an instance of (C2) in $(G(r), W)$. There are exactly 84 ways of assigning a color from the set $\{0, 1\}$ to the points in $[j, j+1] \times [k, k+1] \times [l, l+1]$ such that at least one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in $(G(r), W)$ or $\mathcal{N}(r)$ is an instance of (C2) in $(G(r), W)$. Hence, the cardinality $|A'|$ of the set A' is $|A'| = 2^{56} \cdot 84$.

If we assume that each of the $2^{56} \cdot 84$ binary images $(G(r), W)$ of A' is *equally likely* to occur in any subset $G(r)$ of D, then we can define the conditional probability $P((G(r), W))$ that the points of $G(r) \subset D$ of (D, X) will be assigned the colors in $(G(r), W) \in A'$, given that at least one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, and $\mathcal{N}_z(r)$ is an instance of (C1) in (D, X) or $\mathcal{N}(r)$ is an instance of (C2) in (D, X): $P((G(r), W)) = 1/|A'| = 1/(2^{56} \cdot 84)$.

Let $(G(r), W)$ be any binary image of A'. Since $(G(r), W) \in A'$, either the set $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ contains one, two or three instances of (C1) or the set $\mathcal{N}(r)$ is an instance of (C2). Recall that our algorithm eliminates *all* critical configurations of $(G(r), W)$ in either $\mathcal{N}_x(r) \cup \mathcal{N}_y(r) \cup \mathcal{N}_z(r)$ or $\mathcal{N}(r)$ by selecting a set S of $B(r)$ and inserting all elements of S into P. Recall also that rule (3) is used to select S if and only if for any $S \in B(r)$, the insertion of the elements of S into P gives rise to at least one critical configuration in $(D, X \cup P)$. So, if we assume that each $S \in B(r)$ is equally likely to be selected by rule (3), we can define the probability $P(S)$ that S will be chosen from B by rule (3) as $P(S) = 0$ if $B_2(r) \neq B(r)$ and $P(S) = 1/|B_2(r)|$ otherwise, where $|B_2(r)|$ is the cardinality of $B_2(r)$.

Since the set $B(r)$ is not empty whenever one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ is an instance of a critical configuration in $(D, X \cup P)$, the probability $P(S)$ is well-defined. Now, we can define the probability that the insertion of the elements of S into P will give rise to exactly i critical configurations as $\sum_{S \in B(r)} P(S) \cdot N(S, W, i)$, where $N(S, W, i)$ is 1 if the insertion of the elements of S into P gives rise to exactly i critical configurations, and 0 otherwise.

Because the set $G(r)$ has only 64 points, we can easily build a computer program to obtain $N(S, W, i)$ for each binary image $(G(r), W)$ in A', for each set S in $B(r)$, and for each i, with $1 \leq i \leq K$. However, we must know the value of K. Since any set $S \in B(r)$ contains at most 3 elements, and instances of (C1) and (C2) are mutually exclusive in the same set $\mathcal{N}(p)$, for any $p \in D$, and since each point q in S can give rise to at most 12 instances of (C1) and 8 instances of (C2), we have that the largest value K of $C(r)$ is no larger than 36.

Finally, note that the probability $P[C(r) = i]$ that, for a given $r \in D$, the random variable $C(r)$ will have value i is given by the sum

$$P((G(r), W))[\sum_{S \in B(r)} P(S) \cdot N(s, W, i)]$$

of the conditional probabilities that the elimination of all instances of (C1) in $\mathcal{N}_x(r)$,

$\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, or the instance of (C2) in $\mathcal{N}(r)$ (if any) will give rise to exactly i new critical configurations, given that at least one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ is an instance of a critical configuration. So, for any $r \in D$, we can express the probability $P[C(r) = i]$ that $C(r)$ will have value i as

$$P[C(r) = i] = \sum_{(G(r),W)\in A} P((G(r), W))[\sum_{S\in B(r)} P(S) \cdot N(s, W, i)].$$

We built a computer program to compute $N(S, W, i)$[2], and then we used this program to compute the value of $P[C(r) = i]$, for each i such that $1 \leq i \leq 36$. Using this program, we found $P[C(r) = 1] = 0.0791921$, $P[C(r) = 2] = 0.0316537$, $P[C(r) = 3] = 0.0187593$, $P[C(r) = 4] = 0.0112136$, $P[C(r) = 5] = 0.00472093$, $P[C(r) = 6] = 0.00173523$, $P[C(r) = 7] = 0.00070917$, $P[C(r) = 8] = 0.000186012$, $P[C(r) = 9] = 2.32515 \cdot 10^{-5}$, and $P[C(r) = k]$ is approximately zero for $10 \leq k \leq 36$, $k \in \mathbb{Z}$. So, $E[C(r)]$ of $C(r)$ is 0.284309. From the fact that $E[C(r)] = \sum_{i=1}^{36} i \cdot P[C(r) = i] = 0.284309$, we can derive an upper bound for the expected value $E[t] = \sum_{r\in D} E[C(r)]$ of t in terms of m, as formally stated by the following theorem:

Theorem 5.4.1. *Let (D, X) be a 3D binary digital image. If each distinct configuration of eight pairwise corner adjacent points of D is equally likely, and if there are m instances of critical configurations in (D, X), then the expected value $E[t]$ of the number t of new critical configurations created by the repairing algorithm in Section 5.2 on input (D, X) is less than $m/2$.*

Proof. Let R be the set of points $r \in D$ such that one of $\mathcal{N}_x(r)$, $\mathcal{N}_y(r)$, $\mathcal{N}_z(r)$, and $\mathcal{N}(r)$ is an instance of a critical configuration in (D, X). Clearly, the number of points in R is less than or equal to m. For any $r \in D$, we have that $E[C(r)] = 0.284309$. So, since $C(r) = 0$ for $r \in D - R$, if there are m critical configurations in (D, X), then the expected number of critical configurations resulting from the elimination

[2]http://www.seas.upenn.edu/∼marcelos/probab.cpp

of m existing critical configurations in (D, X) is $E[\sum_{r\in D} C(r)] = E[\sum_{r\in R} C(r) + \sum_{r\in D-R} C(r)] = E[\sum_{r\in R} C(r)] = \sum_{r\in R} E[C(r)] = 0.284309 \cdot |R| \leq 0.284309 \cdot m$, where $|R|$ is the cardinality of R. Since the algorithm iterates until all critical configurations are eliminated from the binary image $(D, X \cup P)$, where P is the set of background points converted into foreground points at any given time, we have that the expected number $E[t] = \sum_{r\in D} E[C(r)]$ of new critical configurations created during the entire execution of the algorithm is bounded above by $\sum_{k=1}^{\infty}(0.284309)^k \cdot m < \sum_{k=1}^{\infty}(1/3)^k \cdot m = m/2$. □

Since the size of the set P obtained by our repairing algorithm is bounded above by $m + t$, Theorem 5.4.1 implies that the expected size of P is bounded above from $\frac{3m}{2}$. In the following section, we discuss the results of an experiment that applied our algorithm to several 3D binary digital images encountered in practical applications. As we shall see, this experiment confirms the theoretical bound on the size of P derived in this section.

5.5 Results and Discussion

We have shown an upper bound for the expected value of the number t of new critical configurations generated by our algorithm on an image with m critical configurations (see Section 5.4). Since the size of the set P generated by our algorithm is bounded above by $m + t$, this upper bound provides an idea of the effectiveness of our algorithm. We now show the results of an experiment that applied our algorithm to six 3D binary digital images.

Each image contains 129^3 = 2,146,689 points and was obtained from the segmentation of a gray-scale magnetic resonance (MR) digital image. More specifically, three of them were obtained from the segmentation of a gray-scale digital image of a normal brain into white matter, gray matter, and cerebrospinal fluid (CSF). The gray-scale brain image was produced by an on-line and freely available 3D MR image

simulator [34]. Two other images are the result of segmenting two lung gray-scale digital images during inspiration and expiration. The sixth image is the result of a fuzzy segmentation ([66]) of a male thorax gray-scale image from the dataset of the Visible Human Project [1].

We ran our algorithm on each of the six images 10 times. For each execution, we collected the size of the set P obtained by the algorithm and the number t of new critical configurations created by our algorithm in order to generate $(D, X' = X \cup P)$. Finally, for each image, we computed the average size of P and the average number of new critical configurations. Table 5.1 summarizes the results of the aforementioned experiment. Figures 5.8-5.13 show the continuous analog of the digital boundary between the foreground and background of the *well-composed* images produced by our experiment.

Note that, for each image, the average size of P is at most 0.32% of the size of the image. Furthermore, the average size of P is smaller than the total number of critical configurations. Note also that the average number of new critical configurations is no larger than $0.2 \cdot m$, where m is the number of critical configurations in the input image. This fact is in agreement with the upper bound on the expected value of t given by Theorem 5.4.1.

Although our experiment does not reveal how far from the optimal the size of P really is, the results in Table 5.1 show that the 3D well-composed images generated by our algorithm are very similar to their corresponding input ill-composed ones (according to the size of P). In principle, however, it is possible that our algorithm generates a set P very far from an optimal set. Besides, it is possible for the optimal size of P to be large, in which case any such conversion algorithm would not be successful.

It would be really interesting to know if it is possible to compute an optimal set P in polynomial time. Also, if we allow us to change the color of both background and foreground points, can we formulate an algorithm that can always terminate?

What about a polynomial time version of such an optimal algorithm? We can also think of modifying our algorithm to include a branch and bound strategy; that is, we can decide which points to change the color by looking ahead the number of critical configurations created by changing the color of distinct choices of points in a few iterations.

Table 5.1: Results from the application of the algorithm in Section 5.2 to six 3D binary digital images with $129^3 = 2{,}146{,}689$ points each. The first column identifies the images; the second and third columns contain the number of instances of the critical configuration (C1) and critical configuration (C2) in the input image, respectively; the fourth column shows the average size of the set P; and the fifth column shows the average number of new critical configurations generated by our algorithm in order to compute P. The average size of P and the average number of critical configurations were computed by executing our algorithm on each of the six binary digital images 10 times.

Image	**# (C1)**	**# (C2)**	**Avg. |P|**	**Avg. # C. C.**
Brain (White Matter)	2538	418	1833.5	249.9
Brain (Grey Matter)	9688	1316	6834.2	2115.9
Brain (CSF)	8574	676	5303.3	787.1
Lung (Inspiration)	1908	128	1213.7	288.4
Lung (Expiration)	3284	237	2068.7	483.2
Thorax	1962	84	1123.7	143.8

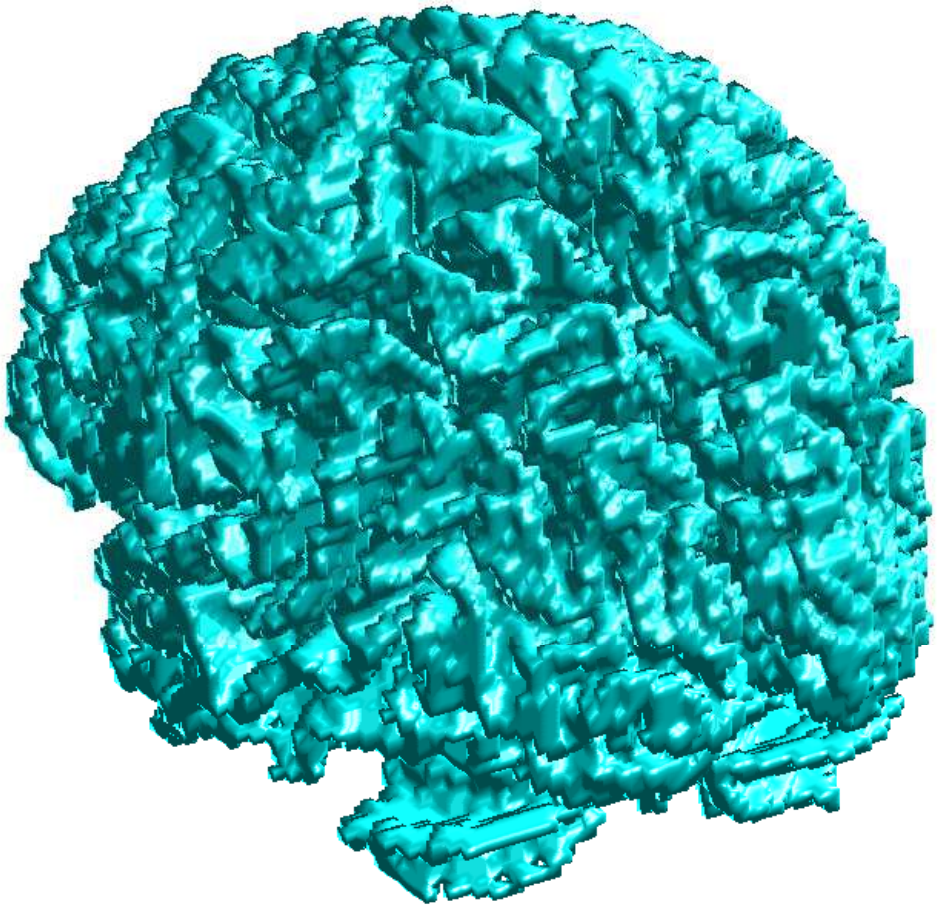


Figure 5.8: Continuous analog of the digital boundary between the foreground and background of the white matter segmentation of a well-composed, normal brain MR image.

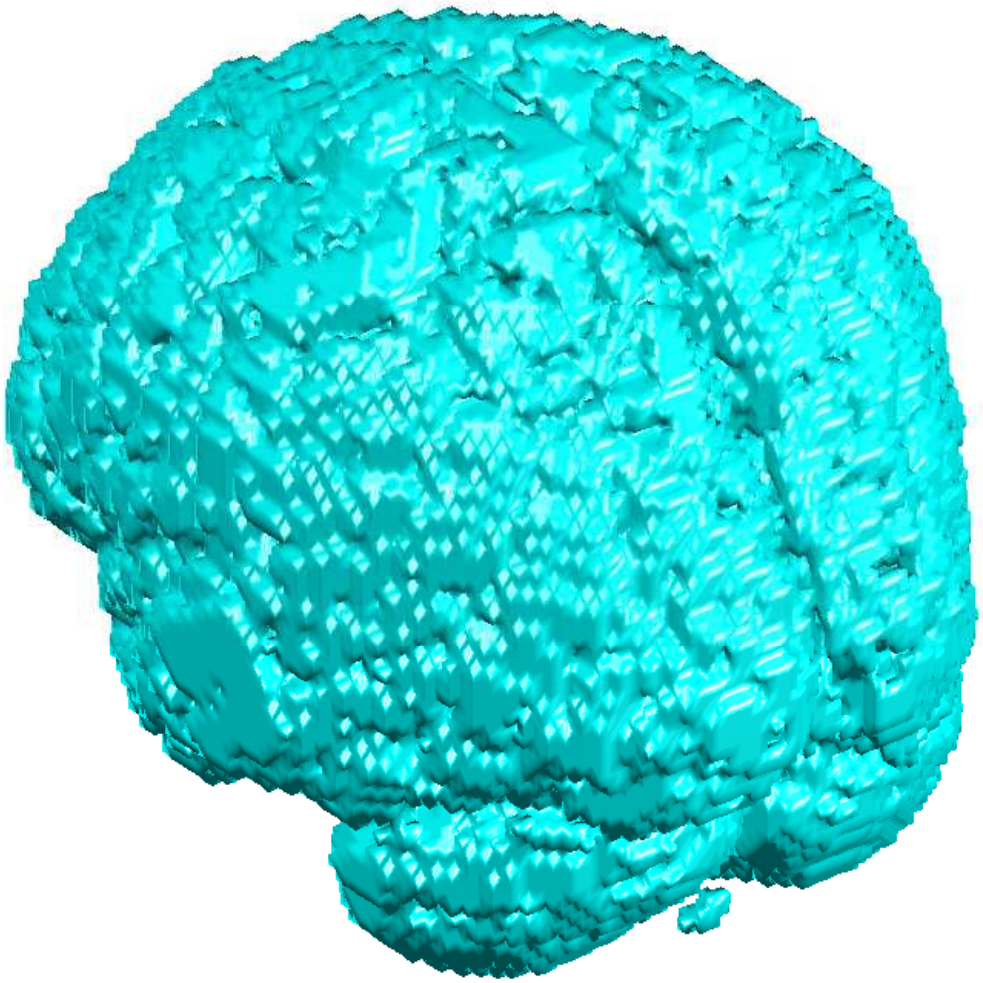


Figure 5.9: Continuous analog of the digital boundary between the foreground and background of the gray matter segmentation of a well-composed, normal brain MR image.

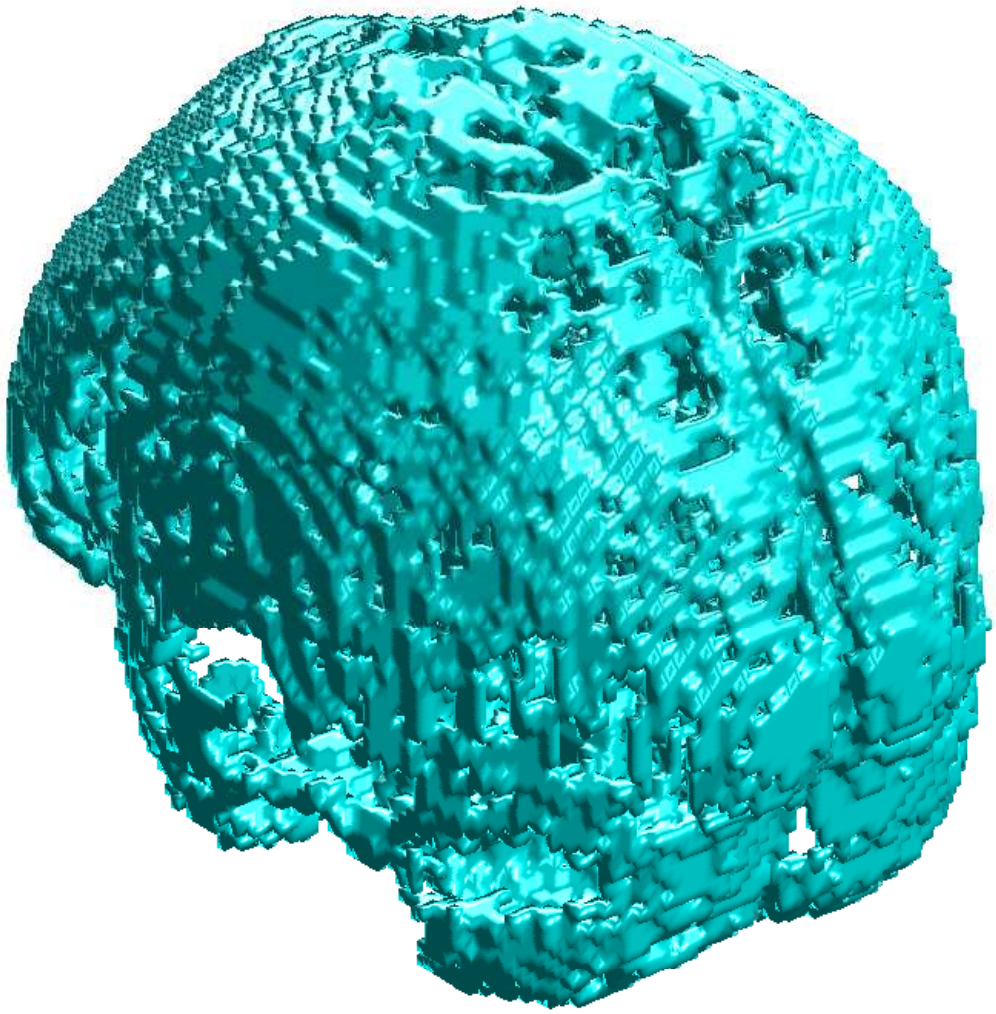


Figure 5.10: Continuous analog of the digital boundary between the foreground and background of the CSF segmentation of a well-composed, normal brain MR image.

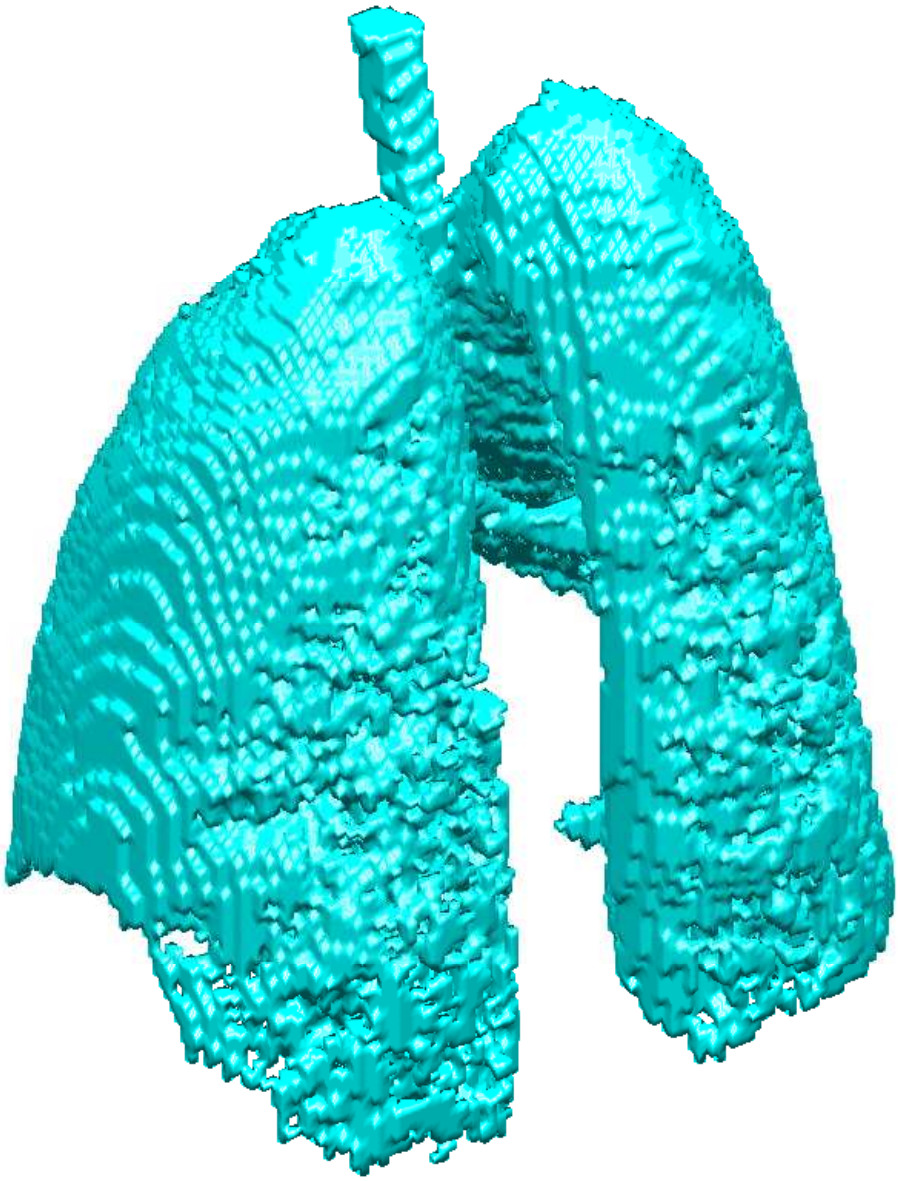


Figure 5.11: Continuous analog of the digital boundary between the foreground and background of the segmentation of a well-composed lung image during expiration.

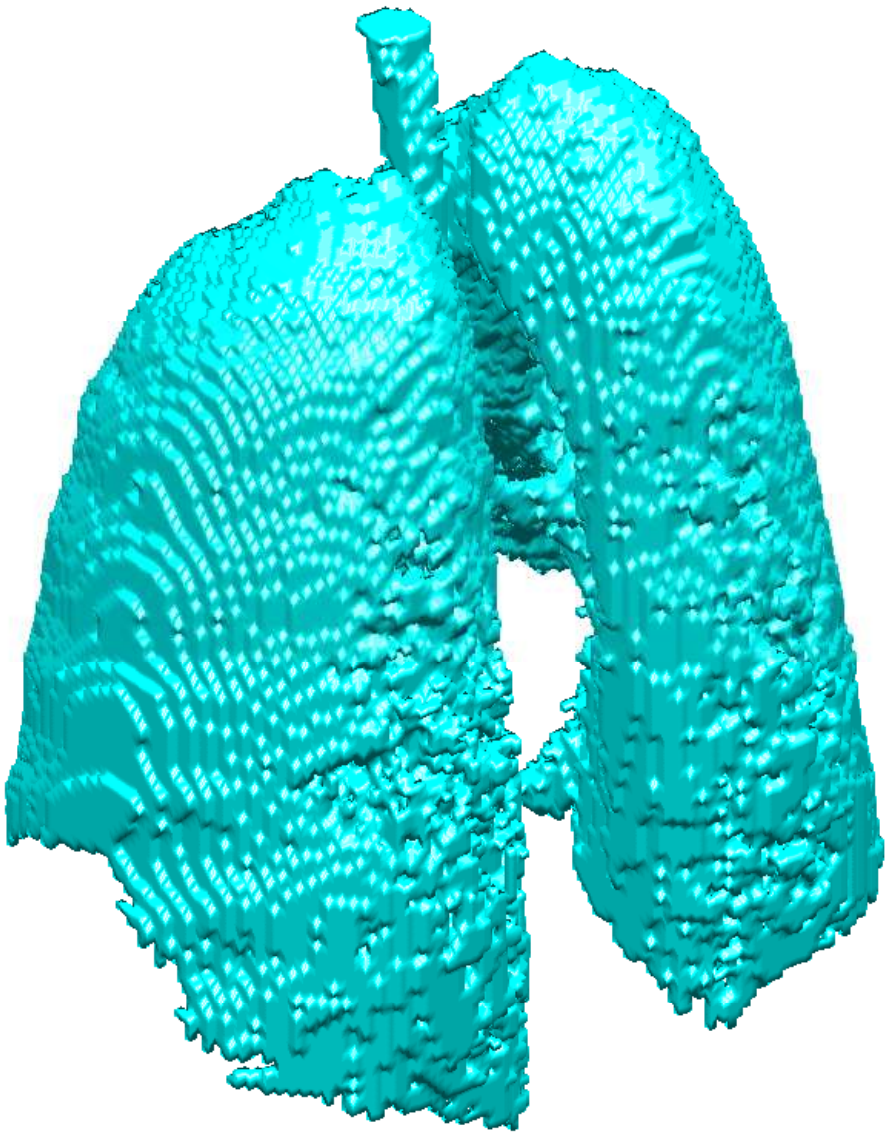


Figure 5.12: Continuous analog of the digital boundary between the foreground and background of the segmentation of a well-composed lung image during inspiration.

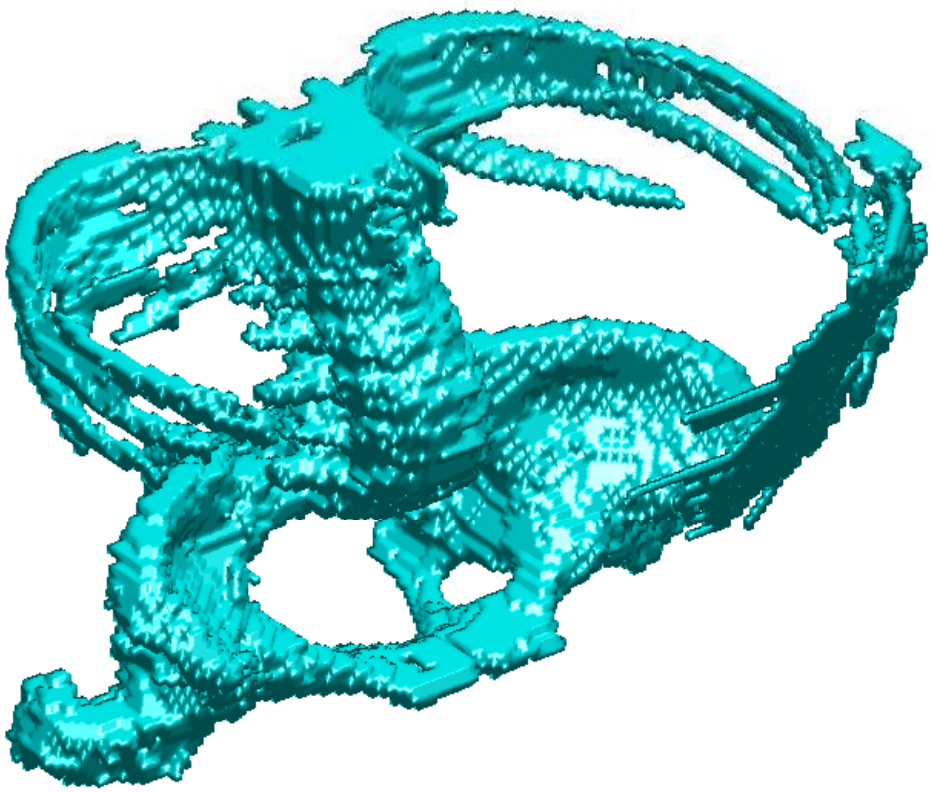


Figure 5.13: Continuous analog of the digital boundary between the foreground and background of the segmentation of a well-composed male thorax image from the Visible Human Project dataset.

Chapter 6

Smooth Surface Representation

Recall from Chapter 4 that the second step of our solution for the surface meshing problem is the generation of a *smooth* surface, which is homeomorphic and geometrically close to the continuous analog of the digital boundary between the foreground and the background of the 3D well-composed image resulting from the first step of our solution.

Several algorithms for constructing a surface from a 3D binary image have been developed in the last two decades [87, 94, 117, 139, 144, 52, 77]. However, most of them generate a piecewise linear surface rather than a smooth one [87, 102, 117, 52, 77]. Moreover, even if the input image is well-composed, the algorithms that are able to produce a smooth surface either do not offer any guarantee on the topology of the resulting surface or are not suitable for handling surfaces with complex topology [94, 139, 144].

This chapter describes a new algorithm for constructing a smooth surface from a 3D well-composed image. More specifically, given a 3D well-composed image (D, X), our algorithm constructs a smooth (i.e., a C^∞-continuous) surface S *which is provably guaranteed to have the same topology as the continuous analog* $bdCA(X)$ *of the digital boundary between the foreground* X *and the background* $X^c = D - X$ *of* (D, X)*. In addition,* S *is a "close" approximation to* $bdCA(X)$*, as the distance from any point*

in $bdCA(X)$ to S is bounded by a small constant depending on the spacing of D only.

6.1 Related Work

Most algorithms for constructing a smooth surface model from a binary image are either based on some interpolation scheme or on a deformable surface model. Interpolation-based algorithms build a smooth surface S by solving an interpolation problem: Given a set of distinct points $\{x_i\}_{i=1}^n \subset \mathbb{R}^3$ on and off S, along with a set of real values $\{h_i\}_{i=1}^n \subset \mathbb{R}$, find a smooth interpolant $f : \mathbb{R}^3 \to \mathbb{R}$ such that $f(x_i) = h_i$, for all $i \in \{1, \ldots, n\}$. The surface S is defined as the zero level set, $S = f^{-1}(0)$, of f.

Yoo, Morse, Subramanian, Rheingans, and Ackerman [144] chose f to be of the form

$$f(p) = \sum_{i=1}^{n} d_i \phi(p - x_i) + Q(p), \quad \forall p = (p_1, p_2, p_3) \in \mathbb{R}^3 \tag{6.1}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a compactly supported *radial basis function* (RBF) defined in an earlier work [100], $Q : \mathbb{R}^3 \to \mathbb{R}$ is the linear polynomial $Q(p) = a + p_1 b + p_2 c + p_3 d$, and $\{d_i\}_{i=1}^n$ is a set of weights associated with the given points in $\{x_i\}_{i=1}^n$. The coefficients a, b, c, and d of Q and the weights $\{d_i\}_{i=1}^n$ associated with $\{x_i\}_{i=1}^n$ are determined by solving an $(n+4) \times (n+4)$ sparse linear system that arises from the equations $f(x_i) = h_i$ and from the additional constraints $\sum_{i=1}^n d_i = \sum_{i=1}^n d_i x_{i,1} = \sum_{i=1}^n d_i x_{i,2} = \sum_{i=1}^n d_i x_{i,3} = 0$, where $x_{i,j}$ is the j-th coordinate of point x_i, for each $j \in \{1, 2, 3\}$.

The above approach is an attractive solution for the problem of finding an interpolant f satisfying $f(x_i) = h_i$, for all $i \in \{1, \ldots, n\}$. This is mainly due to the fact that the associated matrix of the linear system for determining a, b, c, d, and $\{d_i\}_{i=1}^n$ is invertible if the function ϕ is properly chosen or under very mild conditions on the location of the points $\{x_i\}_{i=1}^n$. Furthermore, since ϕ has compact support, the aforementioned system is sparse, and hence it can be solved significantly faster than

the linear systems that arise in the formulation of globally supported RBF-based interpolants [100, 20].

However, the results in [100] show that the computation of a, b, c, d, and $\{d_i\}_{i=1}^n$ can still take longer than one hour if n is of the order of hundreds of thousands, which is often the case if S is to approximate the continuous analog $bdCA(X)$ of the digital boundary between the foreground and background of a binary image (D, X) typically encountered in medical applications. Furthermore, even if (D, X) is a well-composed image, the approach by Yoo, Morse, Subramanian, Rheingans, and Ackerman [144] provides no guarantee on the topology of S, which makes it inappropriate to our purposes.

Ohtake, Belyaev, Alexa, Turk, and Seidel [103] provided an attractive alternative to the approach in [144]. Their work is based on the partition of unity approach (see Section 6.3.2), and was devised to solve the problem of reconstructing a surface from a given set $\{x_i\}_{i=1}^n$ of points on the surface. The reconstructed surface S is also defined as the zero level set $f^{-1}(0)$ of an implicit function $f : \mathbb{R}^3 \to \mathbb{R}$. However, the surface S does not necessarily contain the points of $\{x_i\}_{i=1}^n$, as f locally approximates rather than interpolates these points. Results in [103] show that f can be constructed in a matter of seconds for n as large as hundreds of thousands. However, the topology of S is not guaranteed to be the same as the topology of the surface in which the points $\{x_i\}_{i=1}^n$ lie either.

Although some algorithms for reconstructing topologically correct piecewise linear surfaces from point sets have been developed in previous years [14, 6, 38], only more recently an algorithm for reconstructing topologically correct smooth surfaces was presented [75]. This algorithm is based on an interpolation technique called *moving least squares* (MLS), and it is guaranteed to produce a smooth surface with the correct topology whenever the given point set satisfies a sampling density restriction. If the sampling density restriction does not hold, the correct topology guarantee no longer holds.

Unfortunately, the sampling density restriction is related to the values of the local feature size function at points on the surface we want to reconstruct. Since the value of the local feature size function at a point p on the surface is equal to the distance from p to the nearest point in the medial axis of the surface, both the verification of the sampling density restriction and the computation of a point set that satisfies the restriction may be very expensive. So, the guarantee provided by [75] is not helpful from a practical point of view.

Deformable models are also a powerful tool for creating surface models from imaging data. A deformable surface model can be thought as an "elastic" surface that can change its position, geometry and topology to approximate a given surface or to fit a given set of points. These changes are controlled by internal and external forces acting upon the surface and by user-defined constraints. Deformable models are broadly classified as either parametric (or explicit) or geometric (or implicit) models depending on whether the surface is defined by a parametric or an implicit function, respectively.

Parametric deformable models are suitable for approximating surfaces whose topology is known *a priori* [94], as the parametric representation automatically guarantees that the surface topology will not change while its geometry and position evolve. However, parametric deformable models require a collision detection test to avoid self-intersection, which leads in general to expensive computations. Furthermore, if the topology of the surface we want to approximate is arbitrarily complex, the design of a smooth parametric surface with the given topology is itself a difficult task.

Geometric deformable models on the other hand are defined as level sets of implicit functions, and therefore they inherently prevent self-intersections and they do not need explicit parametrizations. However, the topology can naturally changes during the model evolution process and there is no obvious way of dictating the final topology of the surface.

Whitaker [139] proposed an algorithm based on a geometric deformable model that can generate a smooth surface from a binary image (D, X). His algorithm was originally devised to create surface models that do not suffer from the aliasing artifacts typically encountered in voxel representations. Surfaces generated by the algorithm approximately separate the foreground from the background of (D, X), but they are not guaranteed to be homeomorphic to $bdCA(X)$ when $bdCA(X)$ is a surface.

Han, Xu, and Prince [63] and Bischoff and Kolbbet [11] developed similar and promising approaches that incorporate a topology-preservation mechanism into geometric deformable surface models. Their approaches apply a simple criterion based on the underlying digital topology of the image to prevent topological changes during the model evolution process. However, like parametric deformable models, the initial surface, i.e. level set, must have the desired topology before the evolution process starts, and therefore their approaches are better suitable for approximating surfaces with simple topology.

The algorithms in [144] and [139] do not assume that the input binary image is well-composed. By making this assumption, we either restrict the set of binary images we can handle or we slightly modify the input image to make it into a well-composed one [124]. On the other hand, we open up the possibility of using algorithms from the geometric modeling and computer graphics literature that generate smooth surfaces from piecewise linear surfaces, as the continuous analog of the digital boundary between the foreground and background of a well-composed image is always a piecewise linear surface.

There are many algorithms for building C^1-, C^2-, and even C^∞-continuous surfaces that either approximates or interpolates the vertices of a given arbitrary piecewise linear surface. In principle, the resulting surface can also have the same topology as the piecewise linear surface. Most algorithms are based on the paradigm of stitching parametric patches together [111, 110, 138]. Each patch approximates

or interpolates the vertices of one face of the piecewise linear surface and joins the patches corresponding to adjacent faces with parametric or geometric C^k-continuity, where the degree k of continuity is often 1 or 2.

A major drawback of the stitching paradigm is the need for additional points, called control points, to define the geometry of the patches. These points must be carefully placed to guarantee that adjacent patches join with C^k-continuity along their common boundaries and that non-adjacent patches do not intersect each other. Otherwise, the resulting surface will self-intersect. However, to our best knowledge, there is no known algorithm that provides a procedure to automatically create and place control points such that non-adjacent patches are guaranteed not to intersect each other.

Subdivision surfaces offer an interesting alternative to the stitching of parametric patches [21, 85]. A subdivision surface is the limit surface of a refinement process that recursively subdivides the faces of a piecewise linear surface. Subdivision surfaces are in general parametric C^2-continuous, except at some isolated vertices. Although subdivision surfaces do not require the placement of control points, they can still self-intersect, and hence an additional procedure is needed to detect and avoid self-intersection [60].

Ying and Zorin [142] described an algorithm for generating C^∞-continuous surface from piecewise linear surfaces which is based on the manifold approach of Grimm and Hughes [59]. The manifold approach defines a surface by a set of parametric patches that overlap with C^∞-continuity and cover the whole surface. Ying and Zorin [142] demonstrated that their algorithm produces surfaces with better visual quality than the ones generated by the algorithms based on the stitching of parametric patches. However, their algorithm does not ensure that non-overlapping patches do not intersect each other, which means that the resulting surface may contain self-intersections.

Shen, O'Brien, and Shewchuk [118] developed a method for building interpolating

or approximating implicit surfaces from a "polygon soup", which is a term used to designate an arbitrary set of polygons. A polygon soup may not define a piecewise linear surface and its polygons can arbitrarily intersect each other. Their method is based on the moving least squares interpolation technique, and it can in principle create a smooth surface that approximates the vertices of a given piecewise linear surface and has the same topology as the piecewise linear surface. A major drawback of the method in [118] is that each evaluation of the implicit function defining the approximating surface requires the solution of a linear system, whose number of equations is equal to the number of input polygons.

Unlike the algorithms in [144] and [139], given a well-composed image (D, X), our algorithm always generates a surface that is guaranteed to be homeomorphic to $bdCA(X)$. Our algorithm is faster than the RBF-interpolation based algorithm in [144], and likely faster than the one in [139] based on a deformable model. The implicit function defining the surface generated by our algorithm is smooth, which means that our algorithm can be used by applications that require the computation of higher order derivatives. Furthermore, this implicit function can be evaluated much faster than the function resulting from the method proposed by Shen, O'Brien, and Shewchuk [118].

6.2 Partition of Unity Approach

Our algorithm for constructing a smooth surface homeomorphic to $bdCA(X)$ is based on the notion of *partition of unity* [82]:

Definition 6.2.1. Given a bounded subset U of $\mathbb{R}^n$, a partition of unity $\{\varphi_k\}_{k\in K}$ on U is a set of nonnegative compactly supported functions $\varphi_k : \mathbb{R}^n \to \mathbb{R}$ that add up to 1 at every point of U. More precisely, for each $k \in K$ and for each $p \in U$, we have that $\varphi_k(p) \geq 0$, $\sum_{k\in K} \varphi_k(p) = 1$, and $\{\text{supp}(\varphi_k)\}_{k\in K}$ is a locally finite cover of U, where the support $\text{supp}(\varphi_k)$ of φ_k is the closure of the point set $\{p \in U \mid \varphi_k(p) \neq 0\}$.

A partition of unity is typically used to blend locally defined approximant functions into one global approximant function. More specifically, suppose that we want to approximate a function $g : U \to \mathbb{R}$, and consider a partition of unity $\{\varphi_k\}_{k\in K}$ on U. Suppose also that we associate a function $f_k : \mathbb{R}^n \to \mathbb{R}$ with each $\text{supp}(\varphi_k)$ such that f_k is an approximation for g in $\text{supp}(\varphi_k)$. We can now define a global approximation $f : U \to \mathbb{R}$ for g in terms of the set $\{f_k\}_{k\in K}$ of local approximations and $\{\varphi_k\}_{k\in K}$ as:

$$f(p) = \sum_{k\in K} \varphi_k(p) f_k(p), \quad \text{for all } p \in U. \tag{6.2}$$

Note that if the functions in $\{\varphi_k\}_{k\in K}$ and $\{f_k\}_{k\in K}$ are smooth, then f in (6.2) is also smooth. We use the terms *weight function* and *shape function* to denote φ_k and f_k, respectively.

The above partition of unity approach has long been a key ingredient of finite element meshless methods [129], and it has more recently been used for reconstructing surfaces from point sets [103] and for approximating iso-surfaces from multiple grids [33]. This approach is also used by our algorithm for constructing a smooth surface from a 3D well-composed image. The function $g : U \subset \mathbb{R}^3 \to \mathbb{R}$ approximated by our algorithm is a point-to-surface distance function whose zero level set $g^{-1}(0)$ is the surface $bdCA(X)$. The function f resulting from (6.2) approximates g in U, and the zero level set $f^{-1}(0)$ of f is a smooth surface that is guaranteed to be homeomorphic to $bdCA(X)$.

6.3 Overview of the Algorithm

The input of our algorithm is a 3D well-composed image (D, X) such that D and X satisfy the following two restrictions: (1) D is a finite orthogonal grid defined as

$$D = \{p \in \mathbb{R}^3 \mid p = O + \delta \cdot g\},$$

where $O = (o_1, o_2, o_3) \in \mathbb{R}^3$, $\delta \in \mathbb{R}$, and $g = (g_1, g_2, g_3) \in \{0, n_1\} \times \{0, n_2\} \times \{0, n_3\}$, for some positive integers n_1, n_2, and n_3, and (2) if $p = (p_1, p_2, p_3) \in X$, then $p_i \neq o_i$

and $p_i \neq o_i + \delta \cdot n_i$, for all $i \in \{1, 2, 3\}$. Note that both the continuous analog $CA(D)$ and the convex hull $conv(D)$ of D are "rectangular boxes", that and $conv(D)$ is contained in the interior of $CA(D)$.

The output of our algorithm is a function $f : U \to \mathbb{R}$ defined as in (6.2). The domain U of f is $conv(D)$. The function f is an approximation of the distance function defined as follows:

Definition 6.3.1. The function $g : conv(D) \to \mathbb{R}$ such that for every point $p \in conv(D)$,

$$g(p) = sign(p) \cdot \min\{\|p - q\| \mid q \in bdCA(X)\}, \tag{6.3}$$

is called the *distance function*, where $\|p - q\|$ is the Euclidean distance between p and a point $q \in bdCA(X)$, $sign(p) = 1$ if p is outside $bdCA(X)$, $sign(p) = 0$ if p is on $bdCA(X)$, and $sign(p) = -1$ if p is inside $bdCA(X)$.

Note that the zero level set $g^{-1}(0)$ of g is precisely $bdCA(X)$, as $bdCA(X) \subset conv(D)$, and $g(p) = 0$ if and only if p is a point of $bdCA(X)$. To obtain the partition of unity $\{\varphi_k\}_{k \in K}$ on $conv(D)$ and the set of shape functions $\{f_k\}_{k \in K}$ that define f in (6.2), our algorithm first considers a subdivision of $conv(D)$ into a set of same-sized cubes as follows: Let

$$K = [0, n_1 - 1] \times [0, n_2 - 1] \times [0, n_3 - 1],$$

and for each $k = (k_1, k_2, k_3) \in K$, let C_k denote the cube whose vertices are the points

$$\{o_1 + \delta \cdot k_1, o_1 + \delta \cdot (k_1 + 1)\} \times \{o_2 + \delta \cdot k_2, o_2 + \delta \cdot (k_2 + 1)\} \times \{o_3 + \delta \cdot k_3, o_3 + \delta \cdot (k_3 + 1)\}.$$

Next, for each vertex v of a cube of $\{C_k\}_{k \in K}$, the algorithm computes the value $g(v)$ and assigns it to v, where g is defined by the expression in (6.3). Note that the vertices of the cubes of $\{C_k\}_{k \in K}$ are the points of D.

Each cube C_k of $\{C_k\}_{k \in K}$ is then assigned a weight function $\{\varphi_k\}_{k \in K}$ and a shape function $\{f_k\}_{k \in K}$. The support $\text{supp}(\varphi_k)$ is a cube centered at the center of C_k, and

whose interior contains C_k. The size of this region depends on the minimum absolute value of g at the vertices of all cubes of $\{C_k\}_{k\in K}$ and on the largest magnitude of the gradient of f_k at the vertices of another cube containing C_k. In turn, each shape function f_k is a trilinear interpolation of the values of g at the vertices of C_k. Figure 6.1 illustrates the construction of f using an analogy in $\mathbb{R}^2$. We elaborate on the details of $\{f_k\}_{k\in K}$ and $\{\varphi_k\}_{k\in K}$ in the following two sections.

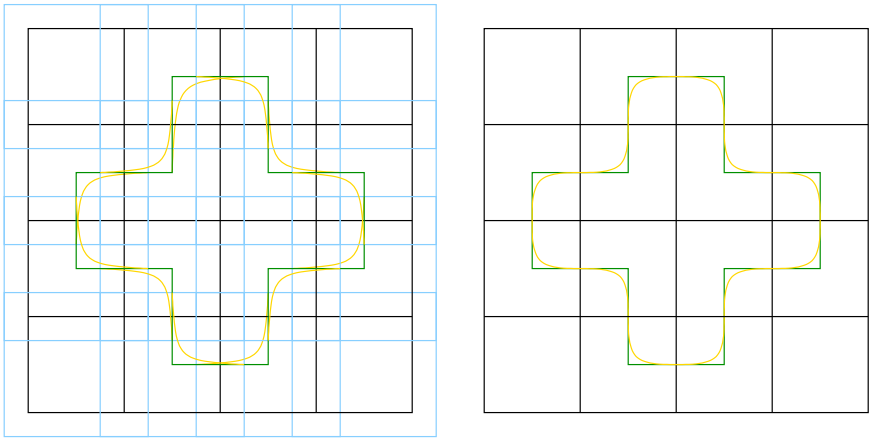


Figure 6.1: (a) The subdivision of U (black), the boundary of each $\text{supp}(\varphi_k)$ (blue), $bdCA(X)$ (green), and the zero level set $f_k^{-1}(0)$ of each shape function f_k (yellow). (b) The subdivision of U (black), $bdCA(X)$ (green), and the zero level set $f^{-1}(0)$ of f (yellow).

We evaluate g at the vertices of the cubes in $\{C_k\}_{k\in K}$ using an algorithm by Maurer, Qi, and Raghavan [93] for computing distance transforms. This algorithm has linear time in the total number of vertices, or equivalently, in the number of points of D. The construction and assignment of weight and shape functions to their corresponding cubes is also done in linear time in the number of cubes of $\{C_k\}_{k\in K}$, which is equal to $n_1n_2n_3$. So, the time complexity of our algorithm is $\mathcal{O}(|D|)$, as $|D| = (n_1+1)\cdot(n_2+1)\cdot(n_3+1)$.

Our algorithm is similar to the algorithms in [103] and [33], as they also subdivide a box-shaped region enclosing U into cubes, and assign a weight function φ_k and a shape function f_k with each cube. However, those algorithms differ from ours in two important ways. First, the support of each weight function of [103] and [33] is a ball centered at the center point of the cube assigned with the function, and each shape function is either a general quadric, a bivariate quadratic polynomial in local coordinates or a piecewise quadric surface [103], and a radial basis function (RBF) interpolant [33]. Second, the zero level set $f^{-1}(0)$ of the function f built by either algorithm is *not* guaranteed to be homeomorphic to the surface one wants to reconstruct from a point set [103] nor to the iso-surface one wants to approximate from multiple grids [33].

6.3.1 Shape Functions

Each function $f_k : \mathbb{R}^3 \to \mathbb{R}$ of the set $\{f_k\}_{k\in K}$ of shape functions used by our algorithm is a trilinear interpolant. This interpolant is defined in terms of the values of the function g in (6.3) at the vertices of the subdivision cube C_k. More specifically, let C be the unit cube $[0,1]^3$ in $\mathbb{R}^3$, and assume that, for each $m, n, l \in \{0,1\}$, the vertex of C with coordinates (m, n, l) is assigned a real number F_{mnl}. Then, we define the shape function f_k as a trilinear interpolation of the values F_{mnl} at the vertices of the cube as follows:

Definition 6.3.2. For every $(x, y, z) \in \mathbb{R}^3$, the *trilinear interpolant* $u : \mathbb{R}^3 \to \mathbb{R}$ is given by

$$u(x,y,z) = \sum_{m,n,l\in\{0,1\}} (1-x)^{1-m} x^m (1-y)^{1-n} y^n (1-z)^{1-l} z^l F_{mnl}. \tag{6.4}$$

Now, let $\{v^k_{mnl}\}_{m,n,l\in\{0,1\}}$ be the set of vertices of C_k such that

$$v^k_{mnl} = (v^k_{000,1} + m\cdot\delta, v^k_{000,2} + n\cdot\delta, v^k_{000,3} + l\cdot\delta)$$

where $v^k_{000,1}$, $v^k_{000,2}$, and $v^k_{000,3}$ are the coordinates of the vertex v^k_{000}, as illustrated by Figure 6.2.

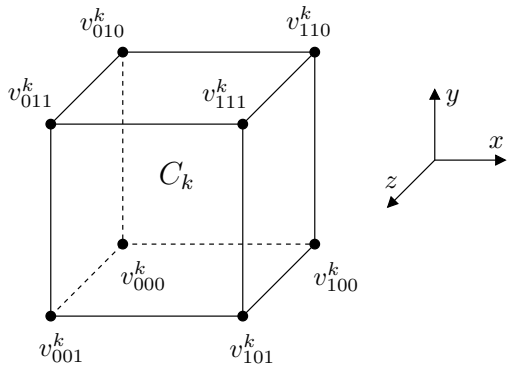


Figure 6.2: The set $\{v^k_{mnl}\}_{m,n,l\in\{0,1\}}$ of vertices of the cube C_k associated with the shape function f_k.

Definition 6.3.3. For every point $(x, y, z) \in \mathbb{R}^3$, the shape function f_k is defined as

$$f_k(x, y, z) = u\Big(\frac{x - v^k_{000,1}}{\delta}, \frac{y - v^k_{000,2}}{\delta}, \frac{z - v^k_{000,3}}{\delta}\Big) \tag{6.5}$$

where the value F_{mnl} of u in (6.4) is equal to $g(v^k_{mnl})$.

By definition, f_k coincides with g at the vertices of C_k, and f_k is an approximation for the distance function g in (6.3) elsewhere in C_k.

Note that the value of u at any point inside the unit cube is a convex combination of the values assigned with the vertices of the cube, the value of u at any point of a face of the cube is a convex combination of the values assigned with the four vertices of the face, and the value of u at any point of an edge of the cube is a convex combination of the values assigned with the two vertices of the edge. So, $f_k^{-1}(0)$ intersects C_k if and only if the values of g at the vertices of C_k are not all positive nor all negative, $f_k^{-1}(0)$ intersects a face of C_k if and only if the values of g at the four vertices of the face are not all positive nor all negative, and $f_k^{-1}(0)$ intersects an edge of C_k if and only if the values of g at the two vertices of the edge are not all positive nor all negative.

By construction, the vertices of each C_k are the centers of eight voxels of (D, X) that share a common vertex. So, the value of g is never zero at a vertex of C_k. Furthermore, there are $2^8 = 256$ possible sign configurations for the vertices of each C_k. However, Lopes and Brodlie [86] pointed out that up to rotational symmetry, reflectional symmetry, and complementarity (switching positive and negative vertices), these configurations are equivalent to the 14 canonical configurations in Figure 6.3. Configurations 3, 4, 6, 7, 10, 12, and 13 correspond to the "ambiguous" cases of the Marching Cubes algorithm, while configurations 0, 1, 2, 5, 8, 9, and 11 correspond to the "unambiguous" ones [87].

The works of Natarajan [102], Chernyaev [30], and Lopes and Brodlie [86] provide detailed analyses of the topology of the trilinear interpolant u inside the unit cube. Their work shows that if the sign configuration of the values assigned with the vertices of the cube is one of configurations 1, 2, 5, 8, 9, and 11 in Figure 6.3, the intersection of $u^{-1}(0)$ and the cube is homeomorphic to $\overline{\mathbb{D}}^2$, i.e., the closed unit disk in $\mathbb{R}^2$ (see Figure 6.4).

In particular, the work of Lopes and Brodlie [86] also shows that if we restrict the equation describing u to the supporting plane of any face of the unit cube, we obtain the equation of a hyperbola. Furthermore, if the sign configuration of the values assigned with the vertices of the unit cube is one of configurations 1, 2, 5, 8, 9, and 11, the intersection of $u^{-1}(0)$ and a face of the cube is exactly one hyperbolic arc.

Since (D, X) is a well-composed image, and since for every point $p \in D$, $g(p) < 0$ if p is inside $bdCA(X)$ (i.e., $p \in X$) and $g(p) > 0$ if p is outside $bdCA(X)$ (i.e., $p \notin X$), the sign configuration of the values of g at the vertices of any subdivision cube C_k must be one of configurations 0, 1, 2, 5, 8, 9, and 11 in Figure 6.3. Otherwise, there would be a critical configuration in (D, X). So, the intersection of any subdivision cube C_k and $f_k^{-1}(0)$ is always homeomorphic to $\overline{\mathbb{D}}^2$ (see Figure 6.4). Likewise, the intersection between C_k and $bdCA(X)$ is also homeomorphic to $\overline{\mathbb{D}}^2$, as the eight

corners of the cube are the centers of eight voxels of (D, X) that share a common vertex inside the cube (see Figure 6.5).

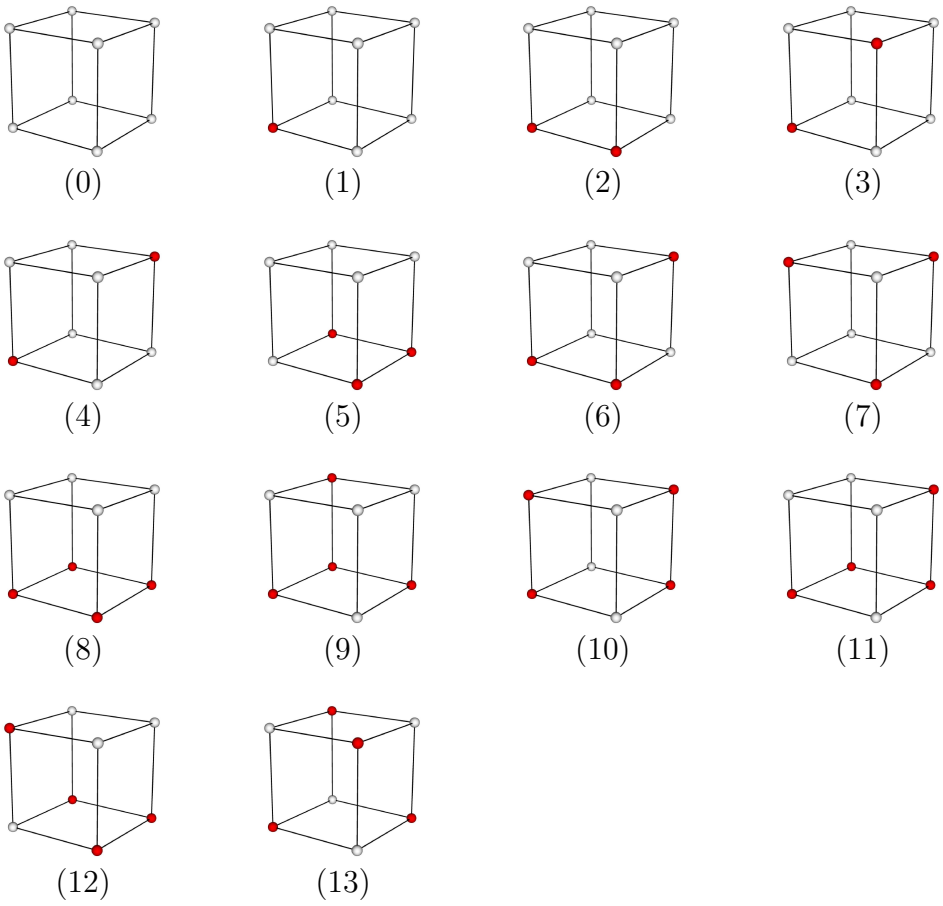


Figure 6.3: The canonical set of 14 sign configurations of g of at the vertices of a subdivision cube. The sign of g is positive at a vertex marked red, and it is negative at a vertex marked gray, or vice-versa.

In addition, $bdCA(X)$ intersects C_k if and only if $f_k^{-1}(0)$ intersects C_k, $bdCA(X)$ intersects a face of C_k if and only if $f_k^{-1}(0)$ intersects the face, and $bdCA(X)$ intersects an edge of C_k if and only if $f_k^{-1}(0)$ intersects the edge. So, the intersections between C_k and $bdCA(X)$ and C_k and $f_k^{-1}(0)$ are both empty or homeomorphic to $\overline{\mathbb{D}}^2$. Furthermore, $f_k^{-1}(0)$ approximates the nonempty intersections between $bdCA(X)$ and the edges, faces and interior of C_k.

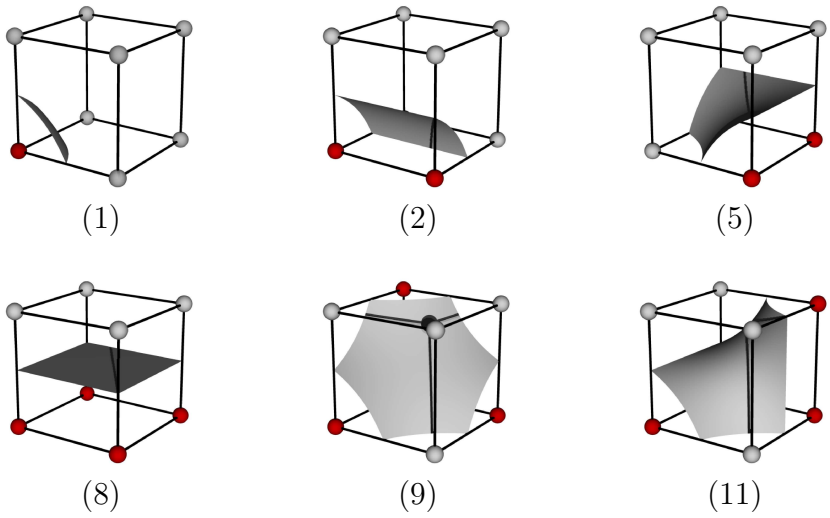


Figure 6.4: The intersection of $u^{-1}(0)$ and the cube $[0,1] \times [0,1] \times [0,1]$ when the values of g at the vertices of the cube define one of configurations 1, 2, 5, 8, 9, and 11 in Figure 6.3.

6.3.2 Partition of Unity

The partition of unity $\{\varphi_k\}_{k \in K}$ on $conv(D)$ used by our algorithm is defined in terms of another set $\{w_k\}_{k \in K}$ of nonnegative compactly supported functions from $\mathbb{R}^3$ to $\mathbb{R}$. Each w_k is also associated with the subdivision cube C_k from $\{C_k\}_{k \in K}$ and is defined as follows:

Definition 6.3.4. Let $w_k : \mathbb{R}^3 \to \mathbb{R}$ be the function such that

$$w_k(p) = \eta(p_1 - c_1^k) \cdot \eta(p_2 - c_2^k) \cdot \eta(p_3 - c_3^k), \tag{6.6}$$

where $p = (p_1, p_2, p_3) \in \mathbb{R}^3$, $c^k = (c_1^k, c_2^k, c_3^k)$ is the center of C_k, and $\eta : \mathbb{R} \to \mathbb{R}$ is given by

$$\eta(t) = \begin{cases} 1 & \text{if } 0 \leq |t| \leq \alpha_1 \\ \frac{h((|t|-\alpha_1)/\beta)}{h((|t|-\alpha_1)/\beta)+h((\alpha-|t|)/\beta)} & \text{if } \alpha_1 < |t| < \alpha_2 \\ 0 & \text{if } |t| \geq \alpha_2, \end{cases} \tag{6.7}$$

where $h : \mathbb{R} \to \mathbb{R}$ is defined as

$$h(t) = \begin{cases} e^{\frac{2e^{-\frac{1}{t}}}{t-1}} & \text{if } 0 \leq t < 1 \\ 0 & \text{otherwise,} \end{cases} \tag{6.8}$$

and $\alpha_2 > \alpha_1 > 0$ and $\beta = \alpha_2 - \alpha_1$ [142]. Here, we let $\alpha_1 = \frac{\delta}{2} - d$ and $\alpha_2 = \frac{\delta}{2} + d$, where

$$d = \min\Big\{\frac{\delta}{5}, \frac{g_{\min}}{2 \cdot L_{\max}}\Big\},$$

where $g_{\min}$ is the smallest absolute value of function g at a point of D, and $L_{\max} = \max_{k \in K} L_k$, with

$$L_k = \frac{\sqrt{3}}{\delta} \cdot \max\Big\{\Big|\frac{\partial f_k}{\partial x}(v)\Big|, \Big|\frac{\partial f_k}{\partial y}(v)\Big|, \Big|\frac{\partial f_k}{\partial z}(v)\Big|\Big\}$$

for all points $v \in \{c_1^k - 0.9\delta, c_1^k + 0.9\delta\} \times \{c_2^k - 0.9\delta, c_2^k + 0.9\delta\} \times \{c_3^k - 0.9\delta, c_3^k + 0.9\delta\}$.

Figure 6.6(a) shows the graph of h for $t \in [0, 1)$ and Figure 6.6(b) shows the graph of η for $\delta = 1$ and $d = 0.2$.

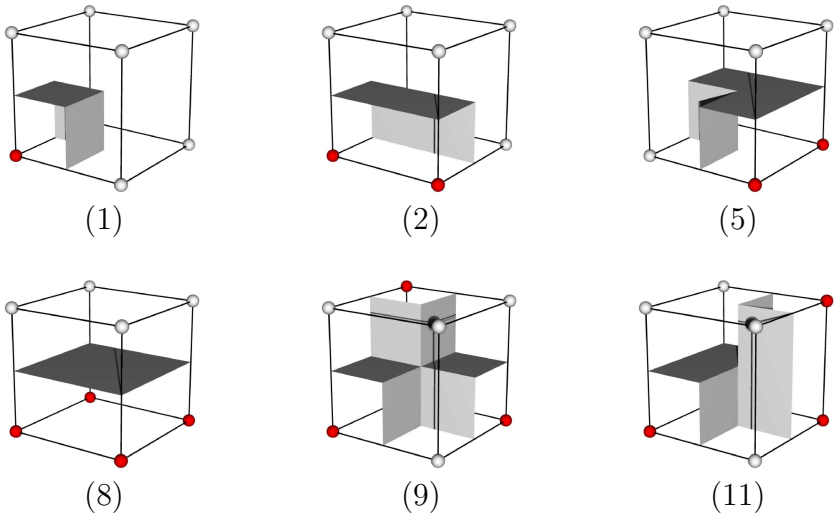


Figure 6.5: The intersection of $bdCA(X)$ and a cube from $\{C_k\}_{k \in K}$ where the sign configuration of C_k corresponds to configuration 0 (empty intersection), 1, 2, 5, 8, 9, or 11 in Figure 6.3.

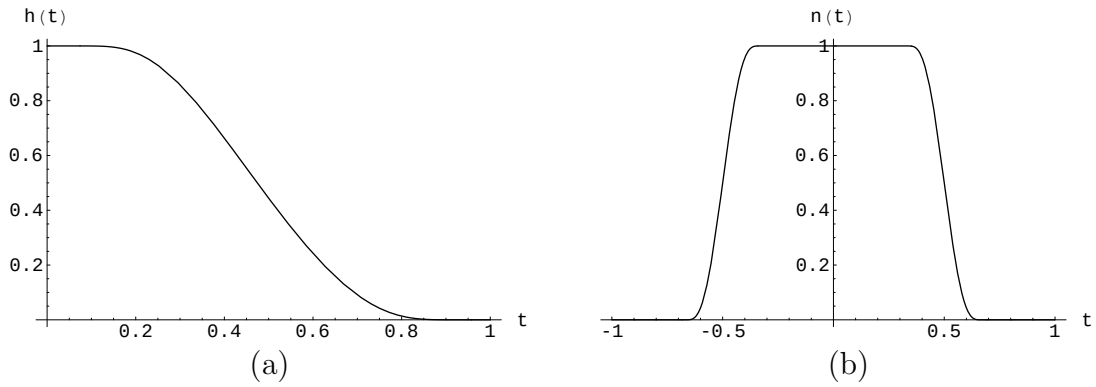


Figure 6.6: (a) Graph of h for $t \in [0, 1)$. (b) Graph of η for $\delta = 1$ and $d = 0.2$.

From the definition of η, it follows that the support $\text{supp}(w_k)$ of each w_k is a cube C_k^1 centered at $c^k = (c_1^k, c_2^k, c_3^k)$ and whose edge length is $\delta + 2d$. Furthermore, $w_k(p) = 1$ if p belongs to the cube C_k^2 centered at c^k and with edge length $\delta - 2d$; $w_k(p) > 0$ and $w_k(p) < 1$ if p belongs to the interior of the set $C_k^1 - C_k^2$; and $w_k(p) = 0$ if p is on the boundary of or outside C_k^1. Figure 6.7 illustrates these facts by showing the values of w_k in a plane through the center of C_k and parallel to two of the faces of C_k.

From the above observations, we have that $\{w_k\}_{k \in K}$ is a set of nonnegative compactly supported functions from $\mathbb{R}^3$ to $\mathbb{R}$. To define a partition of unity $\{\varphi_k\}$ on $conv(D)$, we let

$$\varphi_k(p) = \begin{cases} \dfrac{w_k(p)}{\sum_{j \in K} w_j(p)} & \text{if } p \in conv(D) \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

for each $k \in K$ and for every $p \in \mathbb{R}^3$. By definition, φ_k is nonnegative and has compact support $\text{supp}(\varphi_k) = (\text{supp}(w_k) \cap conv(D))$. Since $conv(D) = \bigcup_{k \in K} \text{supp}(\varphi_k)$, and since $\sum_{k \in K} \varphi_k(p) = 1$ for every $p \in conv(D)$, the set $\{\varphi_k\}$ is a partition of unity on $conv(D)$.

We should remark that for implementation purposes, one may consider letting

$\varphi_k(p) = w_k(p)$. The reason is that $\sum_{k \in K} w_k(p) = 1$ if $p \in conv(D)$ and the distance from p to its closest point on the boundary of $conv(D)$ is at least d, and $\sum_{k \in K} w_k(p) \in (0, 1)$ if $p \in conv(D)$ and the distance from p to its closest point on the boundary of $conv(D)$ is smaller than d. As a result, the approximation of g by $f(p) = \sum_{k \in K} w_k(p) f_k(p)$ in $conv(D)$ may be entirely satisfactory for most applications[1].

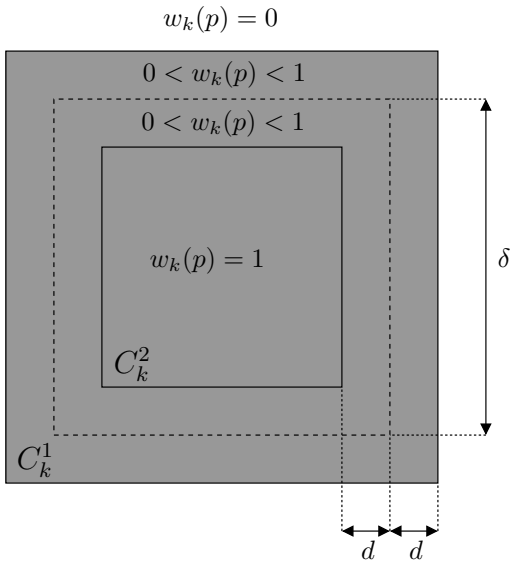


Figure 6.7: The values of w_k at a point p on a plane that intersects a subdivision cube C_k and is parallel to two of its faces. The intersection between the plane and the boundary of C_k is drawn with dashed lines.

6.3.3 Construction, Representation and Evaluation

To construct our function f, we compute the values of g at the points of D as well as the value of the constant d used to define the function η in Section 6.3.2. As we mentioned before, the values of g at the points of D are computed in $\mathcal{O}(|D|)$ using the algorithm in [93]. Likewise, the value of d can also be computed in $\mathcal{O}(|D|)$ time,

[1]It turns out that this is also the case here, as we are interested in $f^{-1}(0)$ only.

as we only need to know the smallest absolute value of g at the points of D, and the gradient of the shape functions at $8 \cdot n$ points of $\mathbb{R}^3$, where $n = n_1 \cdot n_2 \cdot n_3$ is the number of cubes in the uniform subdivision of $conv(D)$. So, we can construct f in $\mathcal{O}(|D|)$ time.

We represent f by an $n_1 \times n_2 \times n_3$ array M, where $(n_1+1)\cdot(n_2+1)\cdot(n_3+1) = |D|$. Each entry $M[k_1, k_1, k_3]$ of M contains the coefficients of the shape function f_k and the center point c_k of the subdivision cube C_k, where $k = (k_1, k_2, k_3)$. To evaluate the function f at a point p of $conv(D)$, we first identify a subdivision cube C_k containing p, and then, for each $j \in J$, where

$$J = [k_1 - 1, k_1 + 1] \times [k_2 - 1, k_2 + 1] \times [k_3 - 1, k_3 + 1],$$

we evaluate the values of φ_j and f_j at p. Finally, the value of f at p is given by the expression

$$f(p) = \sum_{j \in J} \varphi_j(p) f_j(p).$$

Note that the above summation is a restriction of the expression in (6.2), as it only takes into account the weight functions and shape functions of C_k and the 26 cubes that share a vertex, edge, or face with C_k. This is because the weight functions of the other subdivision cubes are zero at p, and consequently they need not be considered. So, the time complexity to evaluate the function f at a point of $conv(D)$ is constant.

6.4 Topological Equivalence

In this section we present the main result of this chapter, which is a theorem stating that the zero level set $S = f^{-1}(0)$ of the implicit function f generated by our algorithm is homeomorphic to $bdCA(X)$. We also present a corollary of this theorem that tells us that S also comes close to the vertices of $bdCA(X)$. Before stating the theorem and the corollary, we present a sequence of propositions and lemmas that

will lead us to a simple proof of the theorem. Let us start with a proposition that states an important fact concerning the trilinear interpolant $u : \mathbb{R}^3 \to \mathbb{R}$ defined in (6.4):

Proposition 6.4.1. *Let $C \subseteq \mathbb{R}^3$ be any cube of the form $C = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$, with $a_i < b_i$ and $|b_1 - a_1| = |b_2 - a_2| = |b_3 - a_3|$, for each $i \in \{1, 2, 3\}$. Then, the trilinear interpolant $u : \mathbb{R}^3 \to \mathbb{R}$ in (6.4) is Lipschitz continuous in C with Lipschitz constant L, where*

$$L \leq \sqrt{3} \cdot \max\Big\{\Big|\frac{\partial u}{\partial x}(v)\Big|, \Big|\frac{\partial u}{\partial y}(v)\Big|, \Big|\frac{\partial u}{\partial z}(v)\Big|\Big\}$$

and v is one of the eight vertices of C.

Proof. By expanding the summation in the expression (6.4) describing u, we obtain an expression of the form

$$u(x, y, z) = a + bz + cy + dyz + ex + fxz + gxy + hxyz$$

where $a = F_{000}$, $b = F_{001} - F_{000}$, $c = F_{010} - F_{000}$, $d = F_{000} + F_{011} - F_{001} - F_{010}$, $e = F_{100} - F_{000}$, $f = F_{000} + F_{101} - F_{001} - F_{100}$, $g = F_{000} + F_{110} - F_{010} - F_{100}$, and $h = F_{111} - F_{110} - F_{101} + F_{100} - F_{011} + F_{010} + F_{001} - F_{000}$. Since u is a polynomial, we know that u is a smooth function. The gradient ∇u of u is $\nabla u = (\partial u/\partial x, \partial u/\partial y, \partial u/\partial z)$, where

$$\begin{aligned}
\frac{\partial u}{\partial x}(x, y, z) &= e + fz + gy + hyz, \\
\frac{\partial u}{\partial y}(x, y, z) &= c + dz + gx + hxz, \\
\frac{\partial u}{\partial z}(x, y, z) &= b + dy + fx + hxy,
\end{aligned}$$

and

$$\|\nabla u\| = \sqrt{\Big(\frac{\partial u}{\partial x}\Big)^2 + \Big(\frac{\partial u}{\partial y}\Big)^2 + \Big(\frac{\partial u}{\partial z}\Big)^2}.$$

Since $\partial u/\partial x$, $\partial u/\partial y$, $\partial u/\partial z$ are continuous functions and since the cube $C = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$ is a bounded domain, $|\partial u/\partial x|$, $|\partial u/\partial y|$, $|\partial u/\partial z|$ have a maximum value in C. We claim that the maximum of each of $|\partial u/\partial x|$, $|\partial u/\partial y|$, $|\partial u/\partial z|$ in C is reached on a vertex of C. Let $p = (x, y, z)$ be a point in C at which $|\partial u/\partial x|$ reaches its maximum value in C. We claim that either p is a vertex of C or there must be a vertex q of C such that $|\frac{\partial u}{\partial x}(q)| = |\frac{\partial u}{\partial x}(p)|$. If the former case is true, we are done. So, assume that p is not a vertex of C. Since the value of $\frac{\partial u}{\partial x}$ does not depend on the first coordinate of p, we can assume that $x = a_1$ or $x = b_1$. Since p is not a vertex of C, either $y \notin \{a_2, b_2\}$ or $z \notin \{a_3, b_3\}$ or both. First, assume that $z \notin \{a_3, b_3\}$, and let t and v be the points $t = (a_1, y, a_3)$ and $v = (a_1, y, 2z - a_3)$ if $|z - a_3| \leq |b_3 - z|$, and let t and v be the points $t = (a_1, y, b_3)$ and $v = (a_1, y, 2z - b_3)$ otherwise. Note that both t and v are in C, and that $\frac{\partial u}{\partial x}(t) = \frac{\partial u}{\partial x}(p) + (f + hy)\delta_z$ and $\frac{\partial u}{\partial x}(v) = \frac{\partial u}{\partial x}(p) - (f + hy)\delta_z$, where $\delta_z = a_3 - z$ if $|z - a_3| \leq |b_3 - z|$, and $\delta_z = b_3 - z$ otherwise. If $(f + hy)\delta_z \neq 0$ then either $|\frac{\partial u}{\partial x}(t)| > |\frac{\partial u}{\partial x}(p)|$ or $|\frac{\partial u}{\partial x}(v)| > |\frac{\partial u}{\partial x}(p)|$, which contradicts our assumption that $|\frac{\partial u}{\partial x}|$ reaches its maximum value in C at p. So, $(f + hy)\delta_z$ must be zero, and therefore $|\frac{\partial u}{\partial x}(t)| = |\frac{\partial u}{\partial x}(p)|$. If $y = a_2$ or $y = b_2$, then t is a vertex of C, and hence we are done. So, assume that $y \notin \{a_2, b_2\}$, and let q and r be the points $q = (a_1, a_2, z_t)$ and $r = (a_1, 2y - a_2, z_t)$ if $|y - a_2| \leq |b_2 - y|$, and let q and r be the points $q = (a_1, b_2, z_t)$ and $r = (a_1, 2y - b_2, z_t)$ otherwise, where z_t is the third coordinate of point t, i.e., either $z_t = a_3$ or b_3. Note that q and r are in C and that q is a vertex of C. Note also that $\frac{\partial u}{\partial x}(q) = \frac{\partial u}{\partial x}(t) + (g + hz)\delta_y$ and $\frac{\partial u}{\partial x}(r) = \frac{\partial u}{\partial x}(t) - (g + hz)\delta_y$, where $\delta_y = a_2 - y$ if $|y - a_2| \leq |b_2 - y|$ and $\delta_y = b_2 - y$, otherwise. If $(g + hz)\delta_y \neq 0$ then either $|\frac{\partial u}{\partial x}(q)| > |\frac{\partial u}{\partial x}(t)|$ or $|\frac{\partial u}{\partial x}(r)| > |\frac{\partial u}{\partial x}(t)|$. Since $|\frac{\partial u}{\partial x}(t)| = |\frac{\partial u}{\partial x}(p)|$, we contradict our assumption that $|\frac{\partial u}{\partial x}|$ reaches its maximum value in C at p. So, $(g + hz)\delta_y$ must be zero, and therefore $|\frac{\partial u}{\partial x}(q)| = |\frac{\partial u}{\partial x}(t)| = |\frac{\partial u}{\partial x}(p)|$. Since q is a vertex point of C, our claim follows for $z \notin \{a_3, b_3\}$. If $z = a_3$ or $z = b_3$, then $y \notin \{a_2, b_2\}$, and we can proceed as we did before for the coordinate z. We can proceed in a similar way to show that $|\partial u/\partial y|$ and $|\partial u/\partial z|$ reach their maximum

values in C at a vertex of C. Finally,

$$\begin{aligned}\|\nabla u(p)\| &= \sqrt{\Big(\frac{\partial u}{\partial x}(p)\Big)^2 + \Big(\frac{\partial u}{\partial y}(p)\Big)^2 + \Big(\frac{\partial u}{\partial z}(p)\Big)^2} \\ &\leq \sqrt{\max\{|\frac{\partial u}{\partial x}(v)|\}^2 + \max\Big\{\Big|\frac{\partial u}{\partial y}(v)\Big|\Big\}^2 + \max\Big\{\Big|\frac{\partial u}{\partial z}(v)\Big|\Big\}^2} \\ &\leq \sqrt{3}\cdot\max\Big\{\Big|\frac{\partial u}{\partial x}(v)\Big|, \Big|\frac{\partial u}{\partial y}(v)\Big|, \Big|\frac{\partial u}{\partial z}(v)\Big|\Big\},\end{aligned}$$

for every point p in C, where v is a vertex of C. Since $\|\nabla u(p)\|$ is bounded in C by the above constant, the function u is Lipschitz continuous in C with Lipschitz constant L, where

$$L \leq \sqrt{3}\cdot\max\Big\{\Big|\frac{\partial u}{\partial x}(v)\Big|, \Big|\frac{\partial u}{\partial y}(v)\Big|, \Big|\frac{\partial u}{\partial z}(v)\Big|\Big\}.$$

□

Note that cube the C in Proposition 6.4.1 can be *any* cube of the form $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \subset \mathbb{R}^3$, with $a_i < b_i$ and $|b_1 - a_1| = |b_2 - a_2| = |b_3 - a_3|$, for each $i \in \{1, 2, 3\}$. Since, for every $p \in \mathbb{R}^3$, the gradient $\nabla f_k(p)$ of $f_k(p)$ is equal to $\frac{1}{\delta}\cdot\nabla u(p)$ of u, where δ is the grid spacing of D, we have that $\|\nabla f_k(p)\| = \frac{1}{\delta}\cdot\|\nabla u(p)\|$. So, from Proposition 6.4.1, the function f_k is also Lipschitz continuous in the cube C with Lipschitz constant L such that L is bounded from above by

$$\frac{\sqrt{3}}{\delta}\cdot\max\Big\{\Big|\frac{\partial u}{\partial x}(v)\Big|, \Big|\frac{\partial u}{\partial y}(v)\Big|, \Big|\frac{\partial u}{\partial z}(v)\Big|\Big\},$$

where v is one of the vertices of C.

For each cube C_k in the set $\{C_k\}_{k\in K}$ of cubes of the subdivision of $conv(D)$ built by our algorithm, consider a cube C'_k centered at the center of C_k, with faces aligned with the faces of C_k, and whose edge length is equal to $1.8\cdot\delta$, where δ is the edge length of C_k. For each cube C'_k, let $L'_k = \frac{\sqrt{3}}{\delta}\cdot\max\Big\{|\frac{\partial u}{\partial x}(v)|, |\frac{\partial u}{\partial y}(v)|, |\frac{\partial u}{\partial z}(v)|\Big\}$, where v is a vertex of C'_k. Since C_k is inside C'_k, Proposition 6.4.1 implies that L'_k is an upper bound for the Lipschitz constant of f_k in C_k. For each C_k, let g_k be the smallest

absolute value of g at any vertex of a cube of $\{C_k\}_{k\in K}$. We define the *Lipschitz ratio* of $\{C_k\}_{k\in K}$ as follows:

$$\frac{\min_{k\in K}\{g_k\}}{\max_{k\in K}\{L'_k\}}.$$

Using the above ratio, we can derive the following two results related to $\{C_k\}_{k\in K}$ and $\{f_k\}_{k\in K}$:

Proposition 6.4.2. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of conv(D) built by our algorithm. Then, for each vertex v of a cube C_k in $\{C_k\}_{k\in K}$, the value of f_k at v has the same sign as the value of f_k at any point inside a sphere centered at v and with radius α, where α is the smallest of $0.4\cdot\delta$ and the Lipschitz ratio of $\{C_k\}_{k\in K}$.*

Proof. For each $k \in K$, we know that f_k is Lipschitz continuous in a cube C'_k centered at the center of C_k, with faces aligned with the faces of C_k, and with edge length $1.8\cdot\delta$, where δ is the edge length of C_k. We also know that the Lipschitz constant of f_k in C'_k is bounded above by $L'_k = \frac{\sqrt{3}}{\delta}\cdot\max\{|\frac{\partial u}{\partial x}(p)|, |\frac{\partial u}{\partial y}(p)|, |\frac{\partial u}{\partial z}(p)|\}$, where p is a vertex of C'_k. Since C_k is inside C'_k, and since the edge lengths of C_k and C'_k are δ and $1.8\cdot\delta$, respectively, a sphere centered at a vertex v of C_k with radius α is entirely contained in C'_k. Consider any point q inside such a sphere. So, $\|v-q\| < \alpha$. From the Lipschitz condition of f_k in C'_k, we know that $|f(v) - f(q)| \le L'_k\cdot\|v-q\|$. Since $\|v-q\| < \alpha$, we have that $\|v-q\| < \frac{\min_{k\in K}\{g_k\}}{\max_{k\in K}\{L'_k\}}$, and hence $|f(v)-f(q)| < \min_{k\in K}\{g_k\}$. Since the value of g is never zero at a vertex of C_k, we must have $\min_{k\in K}\{g_k\} \neq 0$ and $f(v) \neq 0$. Since $|f(v)| \ge \min_{k\in K}\{g_k\}$, the value of $f(q)$ cannot be zero and the sign of $f(q)$ must be the same as the sign of $f(v)$. Otherwise, $|f(v)-f(q)| \ge \min_{k\in K}\{g_k\}$, which is a contradiction. □

Proposition 6.4.3. *Let C_k be any cube from the set $\{C_k\}_{k\in K}$ of cubes of the uniform subdivision of conv(D) built by our algorithm, and let C'_k be the cube centered at the center of C_k, with faces aligned with the faces of C_k, and with edge length equal to $\delta+\alpha$, where α is the smallest of $0.4\cdot\delta$ and the Lipschitz ratio of $\{C_k\}_{k\in K}$. Then,*

the sign configuration of the values of the shape function f_k at the vertices of C'_k is one of configurations 0, 1, 2, 5, 8, 9, *and* 11 *in Figure 6.3.*

Proof. Let v be a vertex of C_k, for any C_k in $\{C'_k\}_{k\in K}$. From the definition of C'_k, a sphere centered at v and with radius $\frac{\alpha}{2}$ contains exactly one distinct vertex p of C'_k in its interior. Since $\frac{\alpha}{2} < 0.2 \cdot \delta < 0.4 \cdot \delta$, Proposition 6.4.2 implies that the value of f_k at p is not zero, and the sign of f_k at p is equal to the sign of f_k at v. So, the sign configuration of f_k at the vertices of C'_k is the same as the one at the vertices of C_k. Since the sign configuration of f_k at the vertices of C_k is one of configurations 0, 1, 2, 5, 8, 9, and 11 in Figure 6.3, our result follows. □

An immediate consequence of Proposition 6.4.3 is the fact that the intersection of $f_k^{-1}(0)$ and C'_k is empty if and only if the intersection of $f_k^{-1}(0)$ and C_k is empty. Furthermore, $f_k^{-1}(0)$ and C'_k enjoy the same properties as $f_k^{-1}(0)$ and C_k. More specifically, since the sign configuration of f_k at the vertices of C'_k is the same as the sign configuration of f_k at the vertices of C_k, and since the value of f_k at any point in C'_k can be expressed as a trilinear interpolation of the values of f_k at the vertices of C'_k, if $f_k^{-1}(0) \cap C'_k \neq \emptyset$ then

(1) $f_k^{-1}(0) \cap C'_k$ is homeomorphic to $\overline{\mathbb{D}}^2$,

(2) the intersection between $f_k^{-1}(0)$ and an edge of C'_k is either empty or an interior point of the edge, and

(3) the intersection between $f_k^{-1}(0)$ and a face of C'_k is either empty or a curve segment whose endpoints are interior points of two distinct edges of the face.

Let d be half the smallest of $0.4 \cdot \delta$ and the Lipschitz ratio of $\{C_k\}_{k\in K}$, i.e.,

$$d = \min_{k\in K}\Big\{\frac{\delta}{5}, \frac{g_{\min}}{2 \cdot L_{\max}}\Big\},$$

where $g_{\min} = \min_{k\in K}\{g_k\}$ and $L_{\max} = \max_{kinK}\{L'_k\}$, and refer to Figure 6.8. For each vertex v of a cube in $\{C_k\}_{k\in K}$, we define the *vertex region* $V(v)$ of v as the set

of points q of $\mathbb{R}^3$ such that $\|v - q\| < 2d$. For each edge e of a cube in $\{C_k\}_{k\in K}$, we define the *edge region* $E(e)$ of e as the set of points q of $\mathbb{R}^3$ such that $\|p - q\| \leq d$, for some point p of e, and $q \notin V(v)$ for any vertex v of a cube in $\{C_k\}_{k\in K}$. For each face f of a cube in $\{C_k\}_{k\in K}$, we define the *face region* $F(f)$ of f as the set of points q of $\mathbb{R}^3$ such that $\|p - q\| \leq d$, for some point p of f, and $q \notin V(v)$ and $q \notin E(e)$, for every vertex v and for every edge e of a cube in $\{C_k\}_{k\in K}$. Finally, for each cube C_k in $\{C_k\}_{k\in K}$, we define the *center region* $C(C_k)$ of C_k as the set of points $q \in C_k$ such that $q \notin V(v)$, $q \notin E(e)$, and $q \notin F(f)$, for every vertex v, every edge e, and every face f of C_k.

Recall that d is the positive constant in the definition of the function η in (6.7). Recall also that the support $\text{supp}(w_k)$ of the function w_k in (6.6) is precisely a cube centered at the center of C_k and whose edge length is $\delta + 2d$. Now, consider the following claims:

Claim 6.4.1. *Let v be a vertex of any cube from the set $\{C_k\}_{k\in K}$ of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, $V(v) \cap \text{supp}(w_k) \neq \emptyset$ if and only if v is a vertex of C_k.*

Proof. Let v be a vertex of any cube in $\{C_k\}_{k\in K}$. By construction of the uniform subdivision of $conv(D)$, the vertex v is shared by either one, two, four, or eight cubes in $\{C_k\}_{k\in K}$. If v is a vertex of C_k, then v is in $\text{supp}(w_k)$, as C_k is inside $\text{supp}(w_k)$. Since $v \in V(v)$, we must have that $V(v) \cap \text{supp}(w_k) \neq \emptyset$. Conversely, if v is not a vertex of C_k, then v is a vertex of some other cube of $\{C_k\}_{k\in K}$, and hence the distance between any point p of $V(v)$ and the supporting plane of any face of C_k is at least $\delta - 2d$. Since $d < 0.2 \cdot \delta$, we have that $\delta - 2d > 0.6 \cdot \delta$. So, $p \notin \text{supp}(w_k)$, and hence $V(v) \cap \text{supp}(w_k) = \emptyset$. □

Claim 6.4.2. *Let e be an edge of any cube from the set $\{C_k\}_{k\in K}$ of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, $E(e) \subseteq \text{supp}(w_k)$ if and only if e is an edge of C_k.*

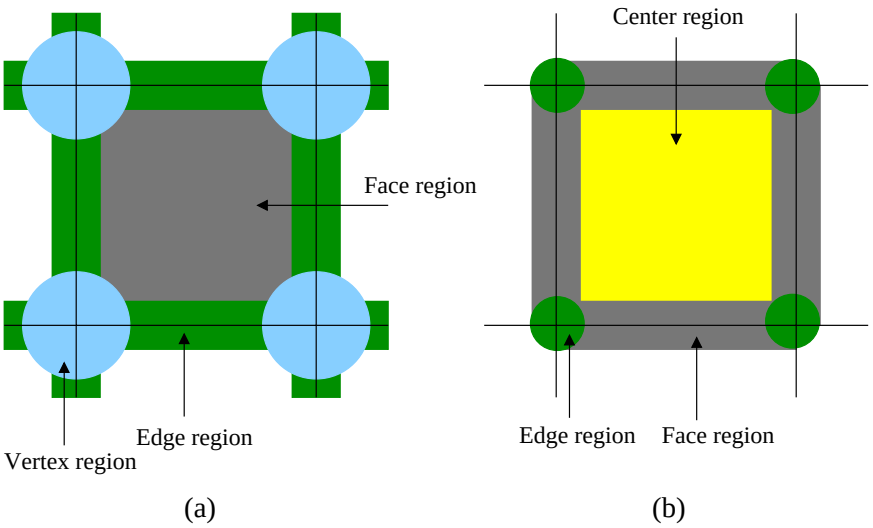


Figure 6.8: (a) Vertex, edge, and faces regions along the face of a cube. (b) Edge, face, and center regions along a plane through the center of a cube and parallel to two faces of the cube.

Proof. Let e be an edge of a cube C_k from $\{C_k\}_{k\in K}$. By definition, a point q is in $E(e)$ if $\|q-p\| \leq d$, for some point p on e, and $q \notin V(v)$, for any vertex v of a cube in $\{C_k\}_{k\in K}$. Since $\text{supp}(w_k)$ is a cube centered at the center of C_k and whose edge length is $\delta + 2d$, we have that $E(e) \subseteq \text{supp}(w_k)$. Conversely, if e is not an edge of C_k then the distance from any point q in $E(e)$ to any face of C_k is at least $\delta - 2d > \delta - 0.4 \cdot \delta = 0.6 \cdot \delta$. So, we must have $E(e) \cap \text{supp}(w_k) = \emptyset$, and our result follows. □

Claim 6.4.3. *Let f be a face of any cube from the set $\{C_k\}_{k\in K}$ of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, $F(f) \subset \text{supp}(w_k)$ if and only if f is a face of C_k.*

Proof. Let f be a face of a cube C_k from $\{C_k\}_{k\in K}$. By definition, a point q is in $F(f)$ if $\|q-p\| \leq d$, for some point p on f, and $q \notin V(v)$ and $q \notin E(e)$, for any vertex v and for every edge e of a cube in $\{C_k\}_{k\in K}$. Since $\text{supp}(w_k)$ is a cube centered at the center of C_k and whose edge length is $\delta + 2d$, we have that $F(f) \subseteq \text{supp}(w_k)$.

Conversely, if f is not a face of C_k, the distance from a point q of $F(f)$ to any face of C_k is greater than $\sqrt{2} \cdot d$, which means that $F(f) \cap \text{supp}(w_k) = \emptyset$, and our result follows. □

Claim 6.4.4. *Let C_k be any cube from the set $\{C_k\}_{k \in K}$ of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, $C(C_k) \subset \text{supp}(w_k)$ and $C(C_k) \cap \text{supp}(w_j) = \emptyset$, for every $j \in K$ with $j \neq k$.*

Proof. Let C_k be any cube from $\{C_k\}_{k \in K}$. Since C_k is inside $\text{supp}(w_k)$ and $C(C_k)$ is inside C_k, we must have that $C(C_k) \subset \text{supp}(w_k)$. Now, let C_j be any cube from $\{C_k\}_{k \in K}$ such that $j \in K$ and $j \neq k$. By construction, the vertex, edge, face, and center regions are pairwise disjoint. So, if q is a point in $C(C_k)$ the distance from q to any vertex, edge or face of C_j is greater than d, and therefore q is outside $\text{supp}(w_j)$. So, $C(C_k) \cap \text{supp}(w_j) = \emptyset$. □

We now prove that $f^{-1}(0)$ has the same topology as $bdCA(X)$. Our proof has two parts. First, we show that $bdCA(X)$ intersects a vertex, edge, face or center region if and only if $f^{-1}(0)$ intersects the same region. Second, we show that if the intersection of $bdCA(X)$ (resp. $f^{-1}(0)$) and a vertex, edge, face or center region is nonempty, then it must be homeomorphic to $\overline{\mathbb{D}}^2$. Since vertex, edge, face, and center regions are pairwise disjoint, and their union contains $conv(D)$, our claim immediately follows.

Lemma 6.4.1. *Let $\{C_k\}_{k \in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, for every vertex v, we have that $bdCA(X) \cap V(v) = \emptyset$ and $f^{-1}(0) \cap V(v) = \emptyset$.*

Proof. Since the vertices of the subdivision cubes are the centers of the voxels of (D, X), i.e., the points of D, the distance between a point $p \in bdCA(X)$ and any vertex v of the subdivision is at least $\frac{\delta}{2}$. Since $V(v)$ is the interior of a closed ball centered at v with radius $2d$, and since $2d < 0.4 \cdot \delta$, we must have $bdCA(X) \cap V(v) = \emptyset$.

Let $\{C_j\}_{j\in J}$ be the subset of cubes of $\{C_k\}_{k\in K}$ that share the vertex v with each other. Note that $\{C_j\}_{j\in J}$ contains at least one and at most eight cubes. From Claim 6.4.1, $\text{supp}(w_k) \cap V(v) \neq \emptyset$ if and only if $k \in J$. Recall that $\text{supp}(\varphi_k) = (\text{supp}(w_k) \cap \mathit{conv}(D))$, and that $\mathit{conv}(D)$ is the domain of f. So, for every $p \in (V(v) \cap \mathit{conv}(D))$, we have $\varphi_k(p) = w_k(p) = 0$ for every $k \notin J$, and $\sum_{j\in J} \varphi_j(p) = 1$. Thus, $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p)$, where f_j is the shape function associated with C_j. From proposition 6.4.2, either $f_j(p) > 0$ for every $j \in J$ or $f_j(p) > 0$ for every $j \in J$. Since $\sum_{j\in J} \varphi_j(p) = 1$, and since $\varphi_j(p)$ is nonnegative for every $j \in K$, we must have that either $f(p) > 0$ or $f(p) < 0$. As a result, $f^{-1}(0) \cap V(v) = \emptyset$, and our claim follows. □

Lemma 6.4.2. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $\mathit{conv}(D)$ built by our algorithm. Then, for every edge e, we have that $bdCA(X) \cap E(e) \neq \emptyset$ if and only if $f^{-1}(0) \cap E(e) \neq \emptyset$.*

Proof. Let e be a subdivision edge, and let v_1 and v_2 be the two subdivision vertices (endpoints) of e. Since v_1 and v_2 are points of D, we know that $bdCA(X)$ intersects e if and only if v_1 and v_2 are not both background nor both foreground points of (D, X). Furthermore, if $bdCA(X)$ intersects e then the intersection is the midpoint of e, and the intersection of $bdCA(X)$ and $E(e)$ is a circle perpendicular to e. On the other hand, if $bdCA(X)$ does not intersect e, the distance from a point in e to a point in $bdCA(X)$ is at least $\frac{\delta}{2}$, which means that $bdCA(X) \cap E(e) = \emptyset$. Let $\{C_j\}_{j\in J}$ be the subset of cubes of $\{C_k\}_{k\in K}$ that share the edge e. From Claim 6.4.2, $E(e) \subset \text{supp}(w_k)$ if and only if $k \in J$. Recall that $\text{supp}(\varphi_k) = (\text{supp}(w_k) \cap \mathit{conv}(D))$, and that $\mathit{conv}(D)$ is the domain of f. So, for every $p \in (E(e) \cap \mathit{conv}(D))$, we have $\varphi_k(p) = w_k(p) = 0$ for every $k \notin J$, and $\sum_{j\in J} \varphi_j(p) = 1$. This means that $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p)$, where f_j is the shape function associated with C_j. Assume that $bdCA(X) \cap E(e) = \emptyset$. We claim that $f^{-1}(0) \cap E(e) = \emptyset$. Since $bdCA(X) \cap E(e) = \emptyset$, either v_1 and v_2 are foreground points or they are both background points of (D, X). So, for every $j \in J$, either $f_j(v_1)$ and $f_j(v_2)$ are both positive or they are

both negative. Furthermore, for every $j, l \in J$, $sign(f_j(v_1)) = sign(f_l(v_1))$ and $sign(f_j(v_2)) = sign(f_l(v_2))$, as $f_j(v_1) = f_l(v_1) = g(v_1)$ and $f_j(v_2) = f_l(v_2) = g(v_2)$. For every point p in $(\bigcup_{j\in J} C_j) \cap (E(e) \cap conv(D))$, consider a line l through p and with the same direction as e. Note that l intersects at least one cube C_j at two points, say p_1 and p_2, where p_1 (resp. p_2) is in the face of C_j that contains v_1 (resp. v_2). If p is in e, then $p_1 = v_1$ and $p_2 = v_2$. Note that $p_1 \in V(v_1)$ and $p_2 \in V(v_2)$, as $\|p_1 - v_1\| \leq d$, $\|p_2 - v_2\| \leq d$. Since $p_1 \in V(v_1)$ and $p_2 \in V(v_2)$, Proposition 6.4.2 tells us that $sign(f_j(p_1)) = sign(f_j(v_1))$ and $sign(f_j(p_2)) = sign(f_j(v_2))$. So, either $f_j(p_1)$ and $f_j(p_2)$ are both positive or they are both negative. From the definition of the trilinear interpolant, the value of each f_j at p is a convex combination of the values of f_j at p_1 and p_2. Since $f_j(p_1)$ and $f_j(p_2)$ are both positive or they are both negative, we must have $f_j(p) > 0$ for every $p \in \overline{p_1p_2}$, or $f_j(p) < 0$ for every $p \in \overline{p_1p_2}$. Furthermore, for every $j, l \in J$, since $sign(f_j(v_1)) = sign(f_l(v_1))$ and $sign(f_j(v_2)) = sign(f_l(v_2))$, we also have that $sign(f_j(p_1)) = sign(f_l(p_1))$ and $sign(f_j(p_2)) = sign(f_l(p_2))$, and hence $f_j(p))$ and $f_l(p)$ are both positive or both negative. Since $\sum_{j\in J} \varphi_j(p) = 1$, and since $\varphi_j(p)$ is nonnegative for every $j \in K$, we can conclude that $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) > 0$ for every $p \in (E(e) \cap conv(D))$, or $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) < 0$ for every $p \in (E(e) \cap conv(D))$. Hence, $f^{-1}(0) \cap E(e) = \emptyset$. Conversely, assume that $bdCA(X) \cap E(e) \neq \emptyset$. We claim that $f^{-1}(0) \cap E(e) \neq \emptyset$. Since $bdCA(X) \cap E(e) \neq \emptyset$, either v_1 is a foreground point of (D, X) and v_2 is a background point of (D, X) or vice-versa. So, for every $j \in J$, either $f_j(v_1) > 0$ and $f_j(v_2) < 0$ or $f_j(v_1) < 0$ and $f_j(v_2) > 0$. Let p be any point in $(e \cap (E(e) \cap conv(D)))$. For any two $j, l \in J$, we must have $f_j(q) = f_l(q)$, as the value of $f_j(p)$ (resp. $f_l(p)$) is a convex combination of their values at v_1 and v_2, and $f_j(v_1) = f_l(v_1)$ and $f_j(v_2) = f_l(v_2)$. So, since $\sum_{j\in J} \varphi_j(p) = 1$, we must have $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) = f_l(p)$, for any $l \in J$. So, $f^{-1}(0) = f_l^{-1}(0)$ in $(e \cap E(e))$. Since $f_l(v_1) > 0$ and $f_l(v_2) < 0$ or $f_l(v_1) < 0$ and $f_l(v_2) > 0$, we know that $f_l^{-1}(0) \cap e$ is a point q of e. We claim that q is also in $E(e)$. This is because $sign(f_l(q)) = sign(v_1)$ if $q \in V(v_1)$ and

$sign(f_l(q)) = sign(v_2)$ if $q \in V(v_2)$. So, $f_l^{-1}(0) \cap (e \cap E(e)) \neq \emptyset$, and therefore $f^{-1}(0) \cap E(e) \neq \emptyset$, as $f^{-1}(0) = f_l^{-1}(0)$ in $(e \cap E(e))$. □

Lemma 6.4.3. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, for every face f, we have that $bdCA(X) \cap F(f) \neq \emptyset$ if and only if $f^{-1}(0) \cap F(f) \neq \emptyset$.*

Proof. Let f be a subdivision face. We know that $bdCA(X)$ intersects f if and only if the vertices of f are not all background nor foreground points of (D, X). Furthermore, if $bdCA(X)$ intersects f then $bdCA(X) \cap f$ is a curve whose endpoints are midpoints of two edge of f, and whose interior lies in the interior of f. Conversely, if $bdCA(X)$ does not intersect f, then the distance from a point p in f to a point in $bdCA(X)$ is at least $\frac{\delta}{2}$. These observations imply that $bdCA(X) \cap F(f) \neq \emptyset$ if and only if $bdCA(X) \cap f \neq \emptyset$. Let $\{C_j\}_{j\in J}$ be the subset of cubes of $\{C_k\}_{k\in K}$ that contain the face f. Note that $\{C_j\}_{j\in J}$ contains either one or two cubes, as f may be on the boundary of $conv(D)$ or it may be shared by two cubes of $\{C_k\}_{k\in K}$. Let p be any point in $(F(f) \cap conv(D))$. From Claim 6.4.3, $F(f) \subset \text{supp}(w_k)$ if and only if $k \in J$. Recall that $\text{supp}(\varphi_k) = (\text{supp}(w_k) \cap conv(D))$, and that $conv(D)$ is the domain of f. Since $p \in (F(f) \cap conv(D))$, we have $\sum_{k\in K} \varphi_k(p) = \sum_{j\in J} \varphi_j(p) = 1$, as $\sum_{k\in K} \varphi_k(p) = 1$ and $\varphi_k(p) = 0$ for $k \notin J$. So, $f(p) = \sum_{k\in K} \varphi_k(p) f_k(p) = \sum_{j\in J} \varphi_j(p) f_j(p)$. Let v_1, v_2, v_3, and v_4 be a circular enumeration of the vertices of f, and let π be a plane through p and parallel to f. Figure 6.9 shows the square $[p_1, p_2, p_3, p_4]$ resulting from the intersection of π and $\bigcup_{j\in J} C_j$. The circumferences around the vertices of $[p_1, p_2, p_3, p_4]$ are the intersections of π and the vertex regions $V(v_1)$, $V(v_2)$, $V(v_3)$, and $V(v_4)$. Since $p \in (F(f) \cap conv(D))$, the distance from p to f is at most d, and therefore the radius of the circumferences in Figure 6.9 are at least $\sqrt{3}d$, as each $V(v_i)$ is the interior of a closed ball with center at v_i and radius $2d$, for $i \in \{1, 2, 3, 4\}$. So, each vertex of $[p_1, p_2, p_3, p_4]$ is inside a vertex region $V(v_i)$. We assume that $p_i \in V(v_i)$. Otherwise, we can relabel the p_i's such that our assumption is true. From Proposition 6.4.2, since $p_i \in V(v_i)$, we must have

$sign(f_j(p_i)) = sign(f_j(v_i))$ for every $j \in J$ and for every $i \in \{1,2,3,4\}$. Furthermore, since $f_j(v_i) = g(v_i) \neq 0$, we can conclude that that $f_j(p_i) \neq 0$. Now, assume that $bdCA(X) \cap F(f) = \emptyset$. This means that the vertices of f are all background or all foreground points of (D, X), which in turn implies that either $f_j(v_i) > 0$ for every $j \in J$ and for every $i \in \{1,2,3,4\}$, or $f_j(v_i) < 0$ for every $j \in J$ and for every $i \in \{1,2,3,4\}$. From the definition of the trilinear interpolant, $f_j(p)$ can be expressed as a convex combination of the values of f_j at p_1, p_2, p_3, and p_4. Furthermore, since $\sum_{j\in J} \varphi_j(p) = 1$, and since $\varphi_j(p)$ is nonnegative, we must have that $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) > 0$ if $f_j(p_i) > 0$ for every $j \in J$ and for every $i \in \{1,2,3,4\}$, and $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) < 0$ if $f_j(p_i) < 0$ for every $j \in J$ and for every $i \in \{1,2,3,4\}$. So, $f^{-1}(0) \cap F(f) = \emptyset$, as p is any point of $(F(f) \cap conv(D))$. Conversely, assume that $bdCA(X) \cap F(f) \neq \emptyset$. So, at least one vertex of f is a background point of (D, X) and at least one vertex of f is a foreground point of (D, X). In other words, the sign of f_j at the vertices v_1, v_2, v_3, and v_4 are not all positive nor all negative. To show that $f^{-1}(0) \cap F(f) \neq \emptyset$, we consider $f(p)$ at a point $p \in (F(f) \cap f)$. For $q \in f$, for $t \in F(f)$, and for any two $j, l \in J$, we have that $f_j(q) = f_l(q)$ and $\varphi_j(t) = \varphi_l(t)$. So, $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p) = f_l(p)$, for any $l \in J$, and hence $f^{-1}(0) = f_l^{-1}(0)$ in $(F(f) \cap f)$. So, to show that $f^{-1}(0) \cap F(f) \neq \emptyset$, it suffices to show that $f_l^{-1}(0)$ intersects $(F(f) \cap f)$. From the properties of the trilinear interpolant, we know that $f_l^{-1}(0) \cap f$ is a hyperbolic arc with endpoints in two distinct edges of f and whose interior lies in the interior of f. We claim that this curve must intersect $(F(f) \cap f)$. Otherwise, it must intersect $V(v_i)$, for some $i \in \{1,2,3,4\}$, and this is not possible. So, $f_l^{-1}(0) \cap (F(f) \cap f) \neq \emptyset$, and therefore $f^{-1}(0) \cap F(f) \neq \emptyset$. □

Lemma 6.4.4. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Then, for every cube C_k of $\{C_k\}_{k\in K}$, we have that $bdCA(X) \cap C(C_k) \neq \emptyset$ if and only if $f^{-1}(0) \cap C(C_k) \neq \emptyset$.*

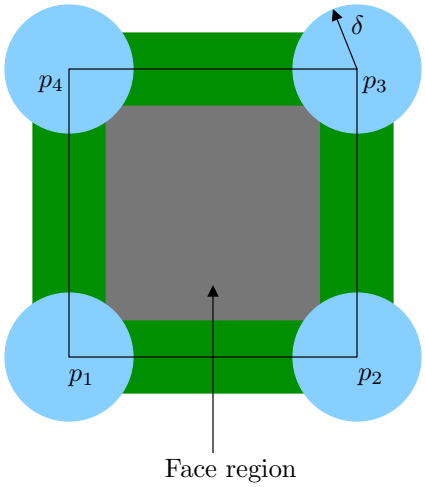


Figure 6.9: The square $[p_1, p_2, p_3, p_4]$ is the intersection of a plane perpendicular to a face f of the cube subdivision of $conv(D)$ and passing through a point p in the face region $F(f)$ of f.

Proof. From Claim 6.4.4, we know that $\varphi_k(p) = w_k(p) = 1$ for every $p \in C(C_k)$, and $\varphi_k(p) = 0$ for all $j \in K$, with $j \neq k$. So, if $p \in C(C_k)$, we must have that for $f(p) = f_k(p)$. Now, if $bdCA(X) \cap C(C_k) = \emptyset$ then the vertices of C_k are either all background or all foreground points of (D, X). This means that $f_k^{-1}(0) \cap C_k = \emptyset$. Since $C(C_k) \subset C_k$ and $f(p) = f_k(p)$ for $p \in C(C_k))$, we must have $f^{-1}(0) \cap C(C_k) = \emptyset$. Conversely, if $bdCA(X) \cap C(C_k) \neq \emptyset$ then the vertices of C_k are not all background nor all foreground points of (D, X). So, $f_k^{-1}(0) \cap C_k \neq \emptyset$. Let C'_k be the cube centered at the center of C_k and whose edge length is equal to $\delta - 2d$. Note that $C(C_k)$ contains all points of C'_k, except for the points that belong to the vertex regions of the vertices of C_k. So, from Claim 6.4.3, we know that C'_k has the same sign configuration as C_k with respect to f_k. Thus, $f_k^{-1}(0) \cap C'_k \neq \emptyset$. Let p be any point of $f_k^{-1}(0) \cap C'_k$. Since $f_k(p) = 0$, point p cannot belong to a vertex region of C_k. So, p must belong to $C(C_k)$, and therefore $f_k^{-1}(0) \cap C(C_k) = f_k^{-1}(0) \cap C'_k$. So, $f_k^{-1}(0) \cap C(C_k) \neq \emptyset$. Since $f(p) = f_k(p)$ for $p \in C(C_k))$, we must have $f_k^{-1}(0) = f^{-1}(0)$ in $C(C_k)$, and hence our result follows. □

To prove the next two lemmas, we make use of the Implicit Function Theorem and one of its many important consequences. Let $h : \mathbb{R}^3 \to \mathbb{R}$ be a continuously differentiable function, and let p be a point in $\mathbb{R}^3$ such that $h(p) = 0$. If the gradient ∇h of h does not vanish at p, then $h^{-1}(0)$ is locally a surface in a small neighborhood around p [62]. Furthermore, if this is true for every point $p \in \mathbb{R}^3$ such that $h(p) = 0$ then $h^{-1}(0)$ is a surface.

Lemma 6.4.5. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. For every edge e of a cube of $\{C_k\}_{k\in K}$, if $bdCA(X)\cap E(e) \neq \emptyset$, then $bdCA(X) \cap E(e)$ and $f^{-1}(0) \cap E(e)$ are homeomorphic to $\overline{\mathbb{D}}^2$.*

Proof. From Lemma 6.4.2, we know that $bdCA(X)\cap E(e) \neq \emptyset$ if and only if $f^{-1}(0)\cap E(e) = \emptyset$, for every edge region $E(e)$ of an edge e of a cube in $\{C_k\}_{k\in K}$. By construction, if $bdCA(X) \cap E(e) \neq \emptyset$ then $bdCA(X) \cap E(e)$ is a circle perpendicular to e and whose center is the midpoint of e and radius is d. So, $bdCA(X) \cap E(e)$ is homeomorphic to $\overline{\mathbb{D}}^2$. We claim that $f^{-1}(0) \cap E(e)$ is also homeomorphic to $\overline{\mathbb{D}}^2$. Let v_1 and v_2 denote the two vertices (endpoints) of e, and let $\{C_j\}_{j\in J}$ be the subset of cubes of $\{C_k\}_{k\in K}$ that share the edge e. From Claim 6.4.2, $E(e) \subset \text{supp}(w_k)$ if and only if $k \in J$. So, for every $k \notin J$ and for every $p \in (E(e) \cap conv(D))$, we have $\varphi_k(p) = w_k(p) = 0$ and $\sum_{k\in K} \varphi_k(p) = \sum_{j\in J} \varphi_j(p) = 1$. This means that $f(p) = \sum_{j\in J} \varphi_j(p) f_j(p)$, where f_j is the shape function associated with C_j. Let p be any point in $(\bigcup_{j\in J} C_j) \cap E(e)$, consider a line l through p and with the same direction as e. Note that l intersects at least one cube C_j at two points, say p_1 and p_2, where p_1 (resp. p_2) is in the face of C_j that contains v_1 (resp. v_2). If p is in e, then $p_1 = v_1$ and $p_2 = v_2$. Note that $p_1 \in V(v_1)$ and $p_2 \in V(v_2)$, as $\|p_1 - v_1\| \leq d$ and $\|p_2 - v_2\| \leq d$. To show that $f^{-1}(0) \cap E(e)$ is homeomorphic to $\overline{\mathbb{D}}^2$, we show that (1) $f^{-1}(0)$ intersects the line segment $(\overline{p_1p_2} \cap (E(e) \cap conv(D)))$ at exactly one point, and (2) the restriction of f to $(\overline{p_1p_2} \cap (E(e) \cap conv(D)))$ is either a strictly increasing or a strictly decreasing function.

From (1) we have that $f(q) = 0$ for exactly a point $q \in (\overline{p_1p_2} \cap (E(e) \cap conv(D)))$.

Since f is a smooth function (we tell why in Theorem 6.4.1), (2) implies that the gradient of f at q is not zero. So, $f^{-1}(0)$ is locally a surface in a small neighborhood around q. If (1) and (2) hold for every line l through a point $p \in (E(e) \cap conv(D))$ and parallel to e, the restriction of $f^{-1}(0)$ to $(E(e) \cap conv(D))$ must be a surface with boundary, which is homeomorphic to $\overline{\mathbb{D}}^2$. So, we proceed to prove (1) and (2).

Since $bdCA(X) \cap E(e) \neq \emptyset$, we know that, for every $j \in J$, either $f_j(v_1) > 0$ and $f_j(v_2) < 0$ or $f_j(v_1) < 0$ and $f_j(v_2) > 0$. Since $p_1 \in V(v_1)$ and $p_2 \in V(v_2)$, Proposition 6.4.2 tells us that $sign(f_k(p_1)) = sign(f_k(v_1))$ and $sign(f_k(p_2)) = sign(f_k(v_2))$, for every $k \in K$. Furthermore, for any two $l, m \in K$, since $f_l(v_1) = f_m(v_1)$ and $f_l(v_2) = f_m(v_2)$, we must have $sign(f_l(p_1)) = sign(f_m(p_1))$ and $sign(f_l(p_2)) = sign(f_m(p_2))$. From the definition of the trilinear interpolant, we know that $f_j(p)$ is a convex combination of $f_j(p_1)$ and $f_j(p_2)$, for every $j \in J$. Since $f_j(p_1) > 0$ and $f_j(p_2) < 0$ or $f_j(p_1) < 0$ and $f_j(p_2) > 0$, the restriction of f_j to the line segment $\overline{p_1 p_2}$ is a strictly decreasing function for every $j \in J$, or a strictly increasing function for every $j \in J$. By the definition of the functions η and w_j, the value of $\varphi_j(q)$ is the same for every $q \in (\overline{p_1 p_2} \cap (E(e) \cap conv(D)))$. Since $\sum_{j \in J} \varphi_j(p) = 1$ and $\varphi_j(p) \geq 0$ for every $p \in (\overline{q_1 q_2} \cap (E(e) \cap conv(D)))$, the above observations imply that $f(p) = \sum_{j \in J} \varphi_j(p)$ is either a strictly decreasing or a strictly increasing function. We also claim that f is zero at a point $p \in (\overline{q_1 q_2} \cap (E(e) \cap conv(D)))$. This is because, for every $i \in \{1, 2\}$ and for every $q \in V(v_i)$, since $\sum_{k \in K} \varphi_k(q) = 1$ and $\varphi_k(q) \geq 0$ and $f(q) = \sum_{k \in J} \varphi_k(q)$, we must have $sign(f(q)) = sign(f_k(q))$. So, for every $q_1 \in V(v_1)$ and $q_2 \in V(v_2)$, either $f(q_1) > 0$ and $f(q_2) < 0$ or $f(q_1) < 0$ and $f(q_2) > 0$, and therefore $f(p) = 0$ for some $p \in (\overline{q_1 q_2} \cap (E(e) \cap conv(D)))$. Finally, since f is either strictly decreasing or strictly increasing along $(\overline{q_1 q_2} \cap (E(e) \cap conv(D)))$, there can be only one point $p \in (\overline{q_1 q_2} \cap (E(e) \cap conv(D)))$ such that $f(p) = 0$. □

Lemma 6.4.6. *Let $\{C_k\}_{k \in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. For every face f of a cube of $\{C_k\}_{k \in K}$, if $bdCA(X) \cap F(f) \neq \emptyset$, then $bdCA(X) \cap F(f)$ and $f^{-1}(0) \cap F(f)$ are homeomorphic to $\overline{\mathbb{D}}^2$.*

Proof. From Lemma 6.4.3, we know that $bdCA(X) \cap F(f) \neq \emptyset$ if and only if $f^{-1}(0) \cap F(f) = \emptyset$, for every face region $F(f)$ of a face f of a cube in $\{C_k\}_{k \in K}$. From the construction of $\{C_k\}_{k \in K}$, if $bdCA(X) \cap F(f) \neq \emptyset$ then $bdCA(X) \cap F(f)$ is homeomorphic to $\overline{\mathbb{D}}^2$. We claim that $f^{-1}(0) \cap F(f)$ is also homeomorphic to $\overline{\mathbb{D}}^2$ whenever $f^{-1}(0) \cap F(f) \neq \emptyset$. Let $\{C_j\}_{j \in J}$ be the subset of cubes of $\{C_k\}_{k \in K}$ that contain f. Note that $\{C_j\}_{j \in J}$ contains either one or two cubes, as f may be on the boundary or in the interior of $conv(D)$, respectively. For each point $p \in (F(f) \cap \bigcup_{j \in J} C_j)$, consider a plane π through p and parallel to f, and refer to Figure 6.9. The intersection of π and $\bigcup_{j \in J} C_j$ is a square $[p_1, p_2, p_3, p_4]$ such that each p_i is in the vertex region $V(v_i)$ of a vertex v_i of f, for $i \in \{1, 2, 3, 4\}$, and p_1, p_2, p_3, and p_4 is a circular enumeration of the vertices of $[p_1, p_2, p_3, p_4]$. If $p \in f$ then $p_i = v_i$ and $[p_1, p_2, p_3, p_4] = f$. Since $bdCA(X) \cap F(f) \neq \emptyset$, we know that the value of f_j at the vertices of f cannot be all positive nor all negative. Since $p_i \in V(v_i)$, we have $sign(f_j(p_i)) = sign(f_j(v_i))$ and $sign(f(p_i)) = sign(f_j(p_i))$. From Lemma 6.4.5, for each edge $[p_l, p_m]$ of $[p_1, p_2, p_3, p_4]$, we have that $sign(f(p_l)) \neq sign(f(p_m))$ if and only if $f^{-1}(0)$ intersects $[p_l, p_m]$. Furthermore, if $f^{-1}(0)$ intersects $[p_l, p_m]$, then the intersection is a point in the edge region $E([v_l, v_m])$ of $[v_l, v_m]$, where $l, m \in \{1, 2, 3, 4\}$. From Proposition 6.4.3 and from the fact that $sign(f(p_i)) = sign(f_j(p_i))$ for every $i \in \{1, 2, 3, 4\}$, we know that $f^{-1}(0)$ intersects exactly two edges of $[p_1, p_2, p_3, p_4]$. These edges may or may not share a common vertex of f. We distinguish these two cases. Without loss of generality, if the edges share a vertex, assume that they are $[p_1, p_2]$ and $[p_2, p_3]$, and then consider the line segment $\overline{qt}$, where $q = p_2$ and t is a point on either $[p_3, p_4]$ or $[p_4, p_1]$. Otherwise, assume that the intersected edges are $[p_1, p_2]$ and $[p_3, p_4]$, and consider the line segment $\overline{qt}$, where q is a point on $[p_2, p_3]$ and t is a point on $[p_4, p_1]$. Now, we can proceed as in Lemma 6.4.5 to show that

(1) $f^{-1}(0)$ intersects $\overline{qt}$ at exactly one point in $(\overline{qt} \cap (F(f) \cap conv(D)))$, and

(2) the restriction of f to $\overline{qt}$ is a strictly decreasing or strictly increasing function.

From (1), (2), and the Implicit Function Theorem, $f^{-1}(0) \cap F(f)$ is homeomorphic to $\overline{\mathbb{D}}^2$. □

Lemma 6.4.7. *Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. For every cube C_k of $\{C_k\}_{k\in K}$, if $bdCA(X) \cap C(C_k) \neq \emptyset$, then $bdCA(X) \cap C(C_k)$ and $f^{-1}(0) \cap C(C_k)$ are homeomorphic to $\overline{\mathbb{D}}^2$.*

Proof. Let $\{C_k\}_{k\in K}$ be the set of cubes of the uniform subdivision of $conv(D)$ built by our algorithm. Lemma 6.4.1 tells us that $bdCA(X) \cap V(v) = \emptyset$ and $f^{-1}(0) \cap V(v) = \emptyset$, for every vertex region $V(v)$ of a vertex v of a cube in $\{C_k\}_{k\in K}$. From the proof of Lemma 6.4.4, for any cube C_k of $\{C_k\}_{k\in K}$ such that $bdCA(X) \cap C(C_k) \neq \emptyset$, we know that $f^{-1}(0) = f_k^{-1}(0)$ in $C(C_k)$. By construction of $\{C_k\}_{k\in K}$, $bdCA(X) \cap C(C_k)$ is homeomorphic to $\overline{\mathbb{D}}^2$. From Claim 6.4.3 and from the proof of Lemma 6.4.4, we can also conclude that $f^{-1}(0) \cap C(C_k)$ is homeomorphic to $\overline{\mathbb{D}}^2$. So, $bdCA(X) \cap C(C_k)$ and $f^{-1}(0) \cap C(C_k)$ are topologically the same. □

Theorem 6.4.1. *Let $f : conv(D) \to \mathbb{R}$ be the function produced by our algorithm on input (D, X), where (D, X) is a 3D well-composed image. Then, the function f is a smooth function such that $f^{-1}(0)$ is a surface homeomorphic to the continuous analog $bdCA(X)$ of the digital boundary between the foreground and background of (D, X).*

Proof. The fact that f is a smooth function follows immediately from the definition of our weight functions $\{\varphi_k\}_{k\in K}$ and shape functions $\{f_k\}_{k\in K}$, which are smooth functions, and from the definition of $f(p)$ as $\sum_{k\in K} \varphi_k(p) f_k(p)$, for every $p \in conv(D)$. The fact that $f^{-1}(0)$ is a surface follows from the fact that $f^{-1}(0)$ is homeomorphic to $bdCA(X)$. The latter fact follows from three observations. First, vertex, edge, face, and center regions are disjoint sets. Second, the vertex, edge, face, and center regions of all vertices, edges, faces, and cubes of the cube subdivision of $conv(D)$ cover $conv(D)$. Third, the intersection of $bdCA(X)$ and each of these regions is nonempty if and only if the intersection of $f^{-1}(0)$ and the same region is nonempty.

Furthermore, whenever the intersection of $f^{-1}(0)$ and a vertex, edge, face, and center region is nonempty, it is homeomorphic to the intersection of $bdCA(X)$ and the same region. This is a consequence of Lemmas 6.4.1, 6.4.2, 6.4.3, 6.4.4, 6.4.5, 6.4.6, and 6.4.7. So, $f^{-1}(0)$ must be homeomorphic to $bdCA(X)$. □

From Lemmas 6.4.1, 6.4.2, 6.4.3, and 6.4.4, we know that $f^{-1}(0)$ intersects a cube C_k of $\{C_k\}_{k\in K}$ if and only if $bdCA(X)$ intersects C_k. So, the distance from any point in $bdCA(X)$ to a point of $f^{-1}(0)$ is bounded by $\sqrt{3}\cdot\delta$, which is the length of the diagonal of C_k. Note that this number depends on the spacing between the points of D only. To formally state this property, we define an r-homeomorphism: Let A and B be two sets in $\mathbb{R}^3$, and let $h : A \to B$ be a homeomorphism. We say that h is a *r-homeomorphism*, where r is some nonnegative constant, if $\|p - h(p)\| \leq r$ for all $p \in A$. If h is an r-homeomorphism, then we say that the sets A and B are *r-homeomorphic*.

Another important feature of $f^{-1}(0)$ is that it separates the background points from the foreground points of (D, X). To see why, recall that, for every point $p \in D$, the value $g(p)$ of the function g in (6.3) at a point p of D is never zero, and $g(p) < 0$ if p is in X and $g(p) > 0$ if p is in $X^c = D \setminus X$. Since $f(p) = g(p)$ for every $p \in D$, we must have that p is the inside of $f^{-1}(0)$ if $p \in X$ and p is in the outside of $f^{-1}(0)$ if $p \in X^c$.

Corollary 6.4.1. *Let $f : conv(D) \to \mathbb{R}$ be the function produced by our algorithm on input (D, X), where (D, X) is a 3D well-composed image. Then, the following hold:*

1. *$bdCA(X)$ and $S = f^{-1}(0)$ are $\sqrt{3}\cdot\delta$-homeomorphic.*

2. *The surface $f^{-1}(0)$ separates the background from the foreground points of (D, X).*

6.5 Results and Discussion

We tested our algorithm against the 3D well-composed images resulting from the application of our algorithm in Chapter 5 to the six binary images images in Figures 5.8-5.13. For each image, we measured the time our algorithm took to construct the function $f : conv(D) \rightarrow \mathbb{R}$, where D is the image domain. In addition, we built a simple program for evaluating f and its gradient ∇f at 20,000,000 points randomly chosen from $conv(D)$. We measured the time this program took to finish when given the function f corresponding to each image. Table 6.1 shows the results of our time measurements.

We intend to compare the results in Table 6.1 with those obtained by some freely available and fast implementation of compactly supported RBF interpolants or by the MLS-based interpolant of Shen, O'Brien, and Shewchuk [118]. Since an RBF interpolant is built from a set of points, rather than a piecewise linear surface, we must carefully design an experiment that may be considered a "fair" comparison. This is not the case with the MLS-based interpolant of [118], as such an interpolant is built from a set of polygons.

Since the images have the same size, and since the time and space complexities to construct and evaluate f depend solely on the image size, the results shown in Table 6.1 are consistent, i.e., the construction and evaluation times of each image are basically the same. Figure 6.10(a) shows a ray traced image of a C^0-surface built by joining the level set of the trilinear interpolant. Figure 6.10(b) shows a ray traced image of C^∞-surface built by our algorithm from the trilinear interpolants used in Figure 6.10(a).

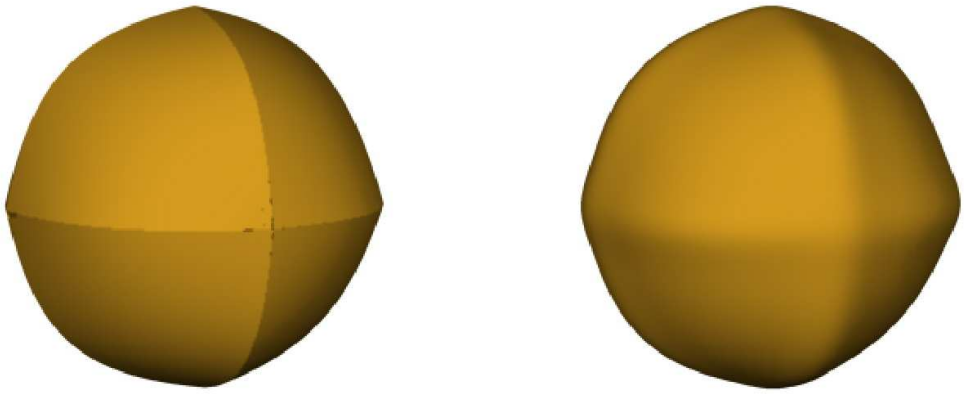
We intend to modify our algorithm by using an adaptive rather than uniform space decomposition of the image domain. The idea is to relax the approximation bound on the distance from points in $bdCA(X)$ to the resulting smooth surface S (see Corollary 6.4.1), so that S will look less "jaggy", and therefore it will be more suitable for visualization purposes. Furthermore, the time to evaluate the function

Table 6.1: The first column identifies the image. The second column shows the time in seconds that our algorithm took to construct f. The third column shows the time in seconds to evaluate f at 20,000,000 points randomly chosen from the domain of f. The fourth column shows the time in seconds to evaluate ∇f at 20,000,000 points randomly chosen from the domain of f. All tests were performed on a Powerbook G4 with a 1.5GHz processor and 512MB of RAM.

Image	**C. T.**	**E. T. of f**	**E. T. of ∇f**
Brain (CSF)	18.6	417	750
Brain (GM)	19.6	441	724
Brain (WM)	15.3	417	715
Lung (EXP)	18.7	416	710
Lung (INS)	18.8	417	715
Thorax	18.3	416	711

f will be shortened.

We also intend to extend our algorithm so that it can deal with arbitrary piecewise linear surfaces. By arbitrary, we mean that the piecewise linear surface is not restricted to be the continuous analog of the digital boundary between the foreground and background of a 3D well-composed image. To implement this extension, we will need an algorithm for computing a uniform cube subdivision of a box-shaped region containing the piecewise linear surface such that the intersection of any subdivision cube and the piecewise linear surface is either empty or homeomorphic to $\overline{\mathbb{D}}^2$. Also, we will need to compute the distance from each vertex of a cube subdivision to the piecewise linear surface. If both tasks can be done efficiently, then the resulting algorithm may be more attractive than previous algorithms for the same problem [143, 118].



(a) (b)

Figure 6.10: (a) C^0-continuous surface defined by the level set of eight trilinear interpolants (one for which octant of the "sphere"). (b) C^∞-continuous surface built by our algorithm from the trilinear interpolants used for defining the C^0-continuous surface in (a).

Chapter 7

Surface Meshing: Part I

As we saw in Chapter 4, the third and last step of our solution for the surface meshing problem generates a simplicial surface $S' \subset \mathbb{R}^3$ from a given smooth surface $S \subset \mathbb{R}^3$ such that S' is homeomorphic to S. The surface S is defined as the zero level set $f^{-1}(0)$ of an implicit function $f : U \subset \mathbb{R}^3 \to \mathbb{R}$. To compute $S' \subset \mathbb{R}^3$, we use a simplified version of an algorithm by Cheng, Dey, Ramos, and Ray [26]. Here, we describe the original algorithm. The details of our simplified version are described in the next chapter.

7.1 Preliminaries

Voronoi Diagram

Definition 7.1.1. Let $P \subset \mathbb{R}^n$ be a finite set of points. For each point q of P, we define the *Voronoi region* $\mathcal{V}(q)$ of q,

$$\mathcal{V}(q) = \{x \in \mathbb{R}^n \mid d(x,q) \leq d(x,r),\ \forall r \in P\}. \tag{7.1}$$

Note that each inequality in (7.1) defines a closed half-space, and thus $\mathcal{V}(q)$ is the

intersection of a finite set of such half-spaces, which implies that $\mathcal{V}(q)$ is a convex set.

The set $\mathcal{V}(q)$ can be unbounded (for instance, if the set P is a unit or a binary set). In fact, it can be shown that $\mathcal{V}(q)$ is unbounded if and only if the site q belongs to the convex hull $conv(P)$ of P [104]. Furthermore, if P has m sites then the boundary of $\mathcal{V}(q)$ consists of up to $m-1$ $(n-1)$-dimensional faces; the boundary of a $(n-1)$-dimensional face of $\mathcal{V}(q)$ consists of $(n-2)$-dimensional faces, and so on. In particular, a 1-dimensional face is called a *Voronoi edge* and a 0-dimensional face is called a *Voronoi vertex*.

Definition 7.1.2. All Voronoi regions of points of P together with their common faces, edges, and vertices form the *Voronoi diagram* $\mathcal{V}(P)$ of P. We call each point $q \in P$ a *site* or *generator* of $\mathcal{V}(P)$ in order to distinguish them from the other points in $\mathbb{R}^n$.

An example of a Voronoi diagram $\mathcal{V}(P)$ for a set P of points in $\mathbb{R}^2$ is given in Figure 7.1.

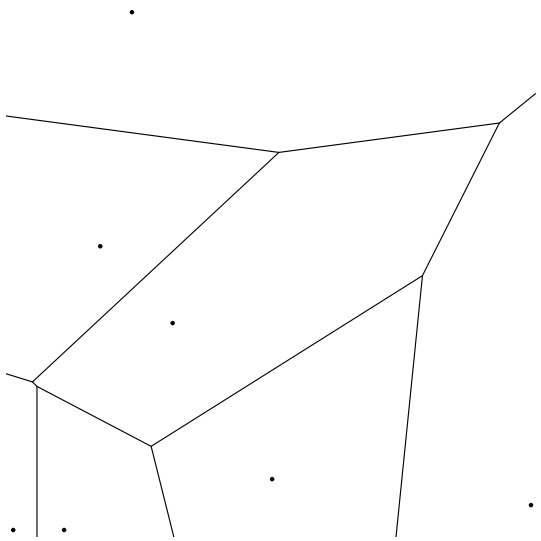


Figure 7.1: Voronoi diagram of a set of points in $\mathbb{R}^2$.

A point $p \in \mathbb{R}^n$ that belongs to m Voronoi regions is equally far from the m sites of these Voronoi regions. It follows that the m sites lie on a common $(n-1)$-sphere in $\mathbb{R}^n$. It can be shown that, if P is in *general position*, i.e., if no $n+2$ sites of P lie on a common $(n-1)$-sphere, then $m \leq n+1$ [16]. Furthermore, if r is a vertex of $\mathcal{V}(P)$ and P is in general position, then r is the common point of exactly $n+1$ Voronoi regions. If P is not in general position then r is the common point of at least $n+1$ Voronoi regions. In particular, if P is a subset of $\mathbb{R}^3$ and P is in general position and no four points of P lie in a common circle in $\mathbb{R}^3$, then every vertex r of a Voronoi region $V(r)$ in $\mathcal{V}(P)$ belongs to exactly three three-dimensional faces and three edges of $V(r)$.

From the definition of a Voronoi region, we have that the sphere centered at a vertex r of $\mathcal{V}(P)$ and containing the sites of the Voronoi regions that meet at r cannot contain any sites of P inside it. This is known as the *empty-sphere property*. The Voronoi diagram $\mathcal{V}(P)$ also satisfies the *nearest-neighbor property*: If $r \in P$ is the nearest-neighbor of $s \in P$, then $\mathcal{V}(r)$ and $\mathcal{V}(s)$ share a common $(n-1)$-dimensional face.

Delaunay Complex

Let $\mathcal{V}(P)$ be the Voronoi diagram generated by a finite set $P \subset \mathbb{R}^n$ of m distinct points in $\mathbb{R}^n$ such that $m \geq n+1$ and not all points in P lie in a common hyperplane of $\mathbb{R}^n$.

Definition 7.1.3. We define the *Delaunay complex* $\mathcal{D}(P)$ of P as the dual of $\mathcal{V}(P)$. That is, if $V = \{v_1, \ldots, v_k\}$ is the set of Voronoi vertices of $\mathcal{V}(P)$, then $\mathcal{D}(P)$ is the polytopal complex consisting of $T_1, \ldots, T_k$ and their boundary elements such that, for every $i \in \{1, \ldots, k\}$,

$$T_i = \Big\{ x \in \mathbb{R}^n \mid x = \sum_{j=1}^{m_i} \lambda_j q_{i,j},\ \sum_{j=1}^{m_i} \lambda_j = 1,\ \lambda_j \geq 0 \Big\},$$

where $\{q_{i,1}, \ldots, q_{i,m_i}\} \subseteq P$ is the set of sites of $\mathcal{V}(P)$ meeting at the Voronoi vertex v_i, and m_i ($m_i \leq m$) is the number of elements of $\{q_{i,1}, \ldots, q_{i,m_i}\}$.

Note that if $m_i = n+1$ and $\{q_{i,1}, \ldots, q_{i,m_i}\}$ is a set of $n+1$ affinely independent points then T_i is a n-simplex. In fact, if $m_i = n+1$ and $q_{i,1}, \ldots, q_{i,m_i}$ are affinely independent for every $i \in \{1, \ldots, k\}$, it can be shown that $\mathcal{D}(P)$ is a simplicial complex [16].

Definition 7.1.4. Whenever $\mathcal{D}(P)$ is a simplicial complex, we say that $\mathcal{D}(P)$ the *n-dimensional Delaunay simplicial complex* or *n-dimensional Delaunay tessellation* $\mathcal{D}_n(P)$ *of* P. If $n = 2$ (resp. $n = 3$), we also refer to $\mathcal{D}_n(P)$ as the *Delaunay triangulation* (resp. *Delaunay tetrahedrization* or *Delaunay tetrahedralization*) of P.

If $m_i > n+1$ for some T_i, and if there is at least one subset of $n+1$ affinely independent points of $\{q_{i,1}, \ldots, q_{i,m_i}\}$, then T_i is a n-dimensional convex polyhedron and we can partition T_i into $(m_i - n)$ openly disjoint n-simplices by cutting T_i with hyperplanes passing through its vertices. So, we can always define a n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$ of P from the Delaunay complex $\mathcal{D}(P)$. However, there can be more than one partition of T_i into $(n_i - m)$ n-simplices, and thus we can define more than one n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$ of P. Otherwise, $\mathcal{D}_n(P)$ is unique.

Figure 7.2 illustrates the duality between $\mathcal{D}_2(P)$ and $\mathcal{V}(P)$, where P is the set of points in Figure 7.1. Each vertex of $\mathcal{D}_2(P)$ is a site of $\mathcal{V}(P)$, each edge of $\mathcal{D}_2(P)$ connects two sites q and r of $\mathcal{V}(P)$ and is the dual of the edge of $\mathcal{V}(P)$ defined by the bisector line of p and q, and each triangle of $\mathcal{D}_2(P)$ is the dual of the Voronoi vertex of $\mathcal{V}(P)$ shared by the Voronoi regions whose sites are three vertices of the triangle. Similarly, the vertices of a Delaunay tetrahedrization $\mathcal{D}_3(Q)$ of a set $Q \subset \mathbb{R}^3$ are the sites of $\mathcal{V}(Q)$, the edges of $\mathcal{D}_3(Q)$ are the dual of facets of $\mathcal{V}(Q)$, the facets of $\mathcal{D}_3(Q)$ are the dual of the Voronoi edges of $\mathcal{V}(P)$, and the tetrahedra of $\mathcal{D}_3(Q)$ are the dual of the Voronoi vertices of $\mathcal{V}(Q)$.

Due to the empty-sphere property, the *circumsphere* of any n-simplex σ of a n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$, i.e., the (unique) $(n-1)$-sphere defined by the $n+1$ vertices of σ, contains no points of P inside it. Note that the center of the circumsphere (circumcenter) of an n-simplex $\sigma \in \mathcal{D}_n(P)$ is the Voronoi vertex which is the dual of σ in $\mathcal{V}(P)$. Likewise, the nearest-neighbor property implies that the edges of the n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$ of P connect the nearest vertices of $\mathcal{D}_n(P)$.

The underlying space of the n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$ of P is precisely the convex hull $conv(P)$. It follows that $\mathcal{D}_n(P)$ is a conforming mesh of $conv(P)$. There are several algorithms for computing the n-dimensional Delaunay tessellation $\mathcal{D}_n(P)$ of a given subset P of $\mathbb{R}^n$, and if the set P has m points, this simplicial complex can be computed in time $\mathcal{O}(m \log m + m^{\lceil n/2 \rceil})$, and this is optimal in the worst case [16].

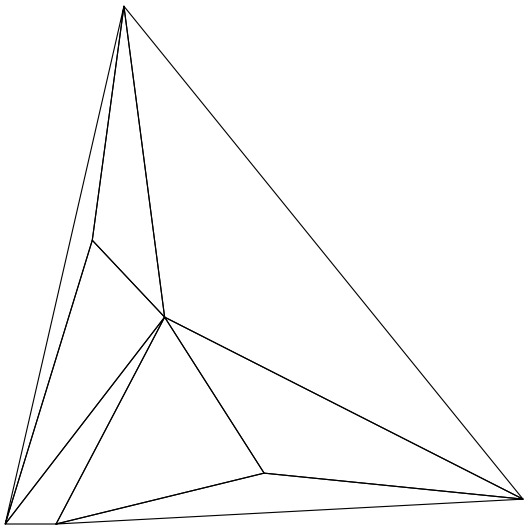


Figure 7.2: Delaunay triangulation of the set of points in Figure 7.1.

Restricted Voronoi Diagram and Its Dual

Throughout the remainder of this chapter, let $S \subset \mathbb{R}^3$ denote a smooth and compact surface, and we let $W \subset S$ be a finite set of points on S. We also assume that $|W| \geq 4$, where $|W|$ is the cardinality of W, and that the points of W are in *general position*, i.e., no four points of W lie in a circumference, and no five points of W are co-spherical.

Definition 7.1.5. The *Voronoi neighborhood restricted to S of $p \in W$*, $\mathcal{V}(S,p)$, is the set

$$\mathcal{V}(S,p) = S \cap \mathcal{V}(p),$$

where $\mathcal{V}(p)$ is the Voronoi neighborhood in $\mathbb{R}^3$ of $p \in W$.

Definition 7.1.6. The *Voronoi diagram restricted to S*, $\mathcal{V}(W,S)$, is the set

$$\mathcal{V}(W,S) = \{\mathcal{V}(S,p) \mid p \in W\}.$$

For any subset $Y \subseteq W$, we will consider the subsets

$$\mathcal{V}(Y) = \{\mathcal{V}(p) \mid p \in Y\} \subseteq \mathcal{V}(W) \quad \text{and} \quad \mathcal{V}(Y,S) = \{\mathcal{V}(S,p) \mid p \in Y\} \subseteq \mathcal{V}(W,S),$$

and their common intersections

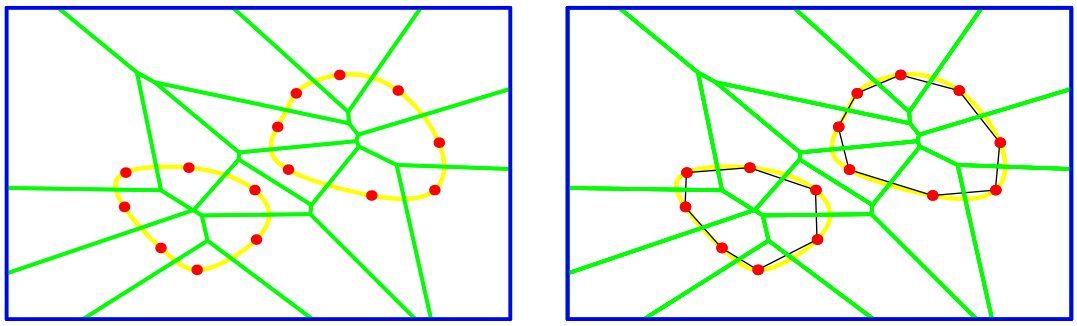
$$\bigcap \mathcal{V}(Y) \quad \text{and} \quad \bigcap \mathcal{V}(Y,S) = S \cap (\bigcap \mathcal{V}(Y)).$$

Definition 7.1.7. Dual to the restricted Voronoi diagram $\mathcal{V}(W,S)$ is the *Delaunay simplicial complex restricted by S*,

$$\mathcal{D}_3(W,S) = \{\sigma_Y \mid Y \subset W, \ \bigcap \mathcal{V}(Y,S) \neq \emptyset\},$$

where σ_Y is the simplex spanned by the point in Y, i.e., the convex hull, $conv(Y)$, of Y.

Figure 7.3 illustrates the concept of a Delaunay simplicial complex restricted by a surface using an analogy with the 2D case, i.e., a Delaunay simplicial complex restricted by a curve.



(a) (b)

Figure 7.3: (a) Voronoi diagram of a set of points sampled on a smooth curve. (b) Delaunay simplicial complex restricted by the curve in (a). The edges of the restricted Delaunay complex (black lines) are the dual of the Voronoi edges that intersect the curve.

7.2 Related Work

The goal of the algorithm in [26] is to compute a set of points W on a given smooth surface S such that $\mathcal{D}_3(W)$ satisfies the *topological property*, i.e., the triangles of $\mathcal{D}_3(W)$ (along with their vertices and edges), which are dual of the Voronoi edges of $\mathcal{V}(W)$ that intersect S, form a simplicial complex S' whose underlying surface $|S'|$ is homeomorphic to S. This simplicial complex S' is the Delaunay simplicial complex of W restricted by S. An important feature of the algorithm in [26] is that the triangles of S' are all well-shaped, i.e., the smallest angle of any triangle is bounded from below by about 30^o.

Chew [31] introduced the first algorithm for computing a restricted Delaunay simplicial complex for a given smooth surface. His algorithm is guaranteed to generate a restricted Delaunay simplicial complex whose triangles are all well-shaped. However, the underlying surface of the resulting restricted Delaunay complex is not guaranteed to be homeomorphic to the input surface. Later, Edelsbrunner and Shah [41] showed that if a set of sampled points on the given surface and its Voronoi diagram

satisfy certain conditions (see Section 7.3 and Section 7.4), then the underlying surface of the restricted Delaunay simplicial complex of the sampled points has the same topology as the given surface.

Using the sampling strategy of Chew's algorithm for obtaining well-shaped triangles, Cheng, Dey, Edelsbrunner, and Sullivan [24] proposed an algorithm for building restricted Delaunay simplicial complexes from a special type of surface, called skin surfaces, where the sampled point set and its Voronoi diagram satisfy the conditions given in [41]. Furthermore, the triangles of the output complex are also guaranteed to be well-shaped.

More recently, Boissonnat and Oudot [15] developed an algorithm that can compute restricted Delaunay simplicial complexes from general surfaces. This algorithm provides the same topological and geometric guarantees as the ones in [24] and [26]. In addition, it also ensures that the underlying surface of the resulting simplicial surface,$|S'|$, is r-homeomorphic to the input surface S for a small positive constant r.

The main drawback of the algorithm by Boissonnat and Oudot [15] is that it requires knowledge of the *distance to the medial axis*, i.e., the Euclidean distance from a point on the surface to its closest point in the medial axis of the surface. Since there is no known algorithm for computing the exact medial axis of a general smooth surface [22], and since existing algorithms for approximate computations require a dense sampling of the surface with respect to the *distance to the medial axis function* [37], the necessary condition for the guarantee provided in [15] is hardly obtained *in practice*.

Although the algorithm in [26] does not require any knowledge about the medial axis of the input surface S, it does require the computation of *critical and silhouette points* of S. There are numerical methods for computing these points, but several surfaces may contain geometric degeneracies, such as infinitely many critical points, that prevent the use of the available numerical routines. *Fortunately, if the input*

surface is a smooth surface defined by the construction in Chapter 6, such numerical computations can be avoided and replaced by much simpler geometric and topological operations.

7.3 Generic Intersection and Genericity

Most of the results in this chapter are concerned with the intersection of Voronoi faces of $\mathcal{V}(W)$ with S. To make the discussion of the results of this chapter clearer, we further assume that the position of the points of W are *non-degenerate*. By non-degenerate, we specifically mean that

(1) the Voronoi vertices of $\mathcal{V}(W)$ are not points of S,

(2) the supporting line of each Voronoi edge of $\mathcal{V}(W)$ is transverse to S, and

(3) any line that belongs to the supporting plane of a Voronoi facet of $\mathcal{V}(W)$ is transverse to S.

Definition 7.3.1. We say that a line l is *transverse* to S if either $l \cap S = \emptyset$ or if, for all $p \in (l \cap S)$, the unit vector $\vec{l}$ (or $-\vec{l}$), where $\vec{l}$ is the direction vector of l, does not belong to T_pS, where T_pS is the tangent space of S at p.

Refer to Figure 7.4 for an illustration of Definition 7.3.1.

Assumption (1) implies that the intersection between S and a Voronoi edge of $\mathcal{V}(W)$ cannot include an endpoint of the edge. This means that the intersection between S and a Voronoi edge of $\mathcal{V}(W)$ is equal to the intersection of S and the interior of the edge. Assumptions (1) and (2) imply that S intersects a Voronoi edge of $\mathcal{V}(W)$ in at most finitely many points, and that no Voronoi edge can tangentially meet S at a point. Assumption (3) implies that the intersection between S and a Voronoi facet of $\mathcal{V}(W)$ is either empty or a set of disjoint curves, and that no Voronoi facet can tangentially meet S at a point.

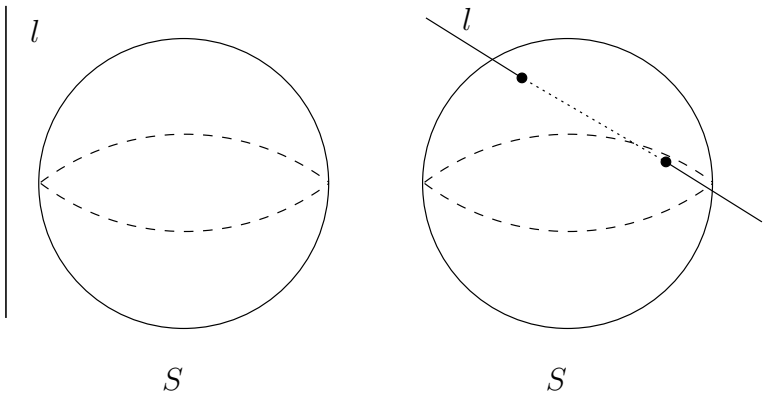


Figure 7.4: A line l transverse to a smooth surface S.

Generic Intersection Condition: We say that $\mathcal{V}(W)$ *intersects* S *generically* if the points of W are non-degenerate, or equivalently, if the aforementioned assumptions (1), (2), and (3) hold.

While assumptions (1), (2), and (3) are hardly satisfied in practice, we can numerically simulate these assumptions by using conceptual perturbation techniques [40, 141]. These techniques rely on the fact that the set of lines l that intersect S transversally is *dense* in $\mathbb{R}^3$. Roughly speaking, this means that a slight perturbation in the position of the points of W will cause assumptions (1), (2), and (3) to hold with high probability [62].

We now state a condition from [41] under which the Delaunay simplicial complex restricted by S, $\mathcal{D}_3(W, S)$, is the underlying space of a simplicial surface that is homeomorphic to S:

Closed Ball Property: Assume that $\mathcal{V}_W$ intersects S generically. Then, we say that $\mathcal{V}(W, S)$ satisfies the *closed ball property* if the intersection of S and each Voronoi face of dimension n of $\mathcal{V}(W)$ is either the empty set or it is homeomorphic to a closed ball in $\mathbb{R}^{(n-1)}$, for short, a closed $(n-1)$-ball.

The following important theorem from [41] establishes a relationship between the topology of S and the Delaunay simplicial complex restricted by S, $\mathcal{D}_3(W, S)$:

Theorem 7.3.1. *Let $S \subset \mathbb{R}^3$ be a surface, and let $W \subset S$ be a finite set of points on S such that the points in W are in general position and the Voronoi diagram $\mathcal{V}(W)$ of W in $\mathbb{R}^3$ intersects S generically. If the Voronoi diagram restricted to S, $\mathcal{V}(W, S)$, satisfies the closed ball property, then the Delaunay simplicial complex restricted by S, $\mathcal{D}_3(W, S)$, is a simplicial surface whose underlying surface, $|\mathcal{D}_3(W, S)|$, is homeomorphic to S.*

Let $\overline{\mathbb{B}}_r(c)$ be a closed 3-ball with center c and radius r, where $r > 0$. We say that $\overline{\mathbb{B}}_r(c)$ is empty with respect to S if the interior of $\overline{\mathbb{B}}_r(c)$ contains no point of S. We say that $\mathbb{B}_r(c)$ is a *medial ball* if (1) $\overline{\mathbb{B}}_r(c)$ is empty, and (2) $\overline{\mathbb{B}}_r(c)$ is a maximal ball, i.e., it is not properly contained in any other closed 3-ball. The center of a medial ball is either a point with more than one closest point to S, or a center of curvature of S [7]. The *medial axis* of S is the closure of the set of points that are the centers of medial balls of S.

Definition 7.3.2. We define the *distance to the medial axis function*, $d_M(p)$, as the minimum Euclidean distance from a point $p \in S$ to any point of the medial axis of S.

The distance to the medial axis function can be viewed as a local measure of "level of detail" on S; that is, if the medial axis is close to S then either the curvature of S is high or some other patch of S is nearby (the *thickness* of the surface is small). Since S is a smooth surface, the distance from any point on S to the medial axis is always greater than zero, and therefore $d_M(p) > 0$ for every $p \in S$. If that was not the case, the medial axis could meet S at a point p (e.g., when p is a sharp corner of S), and hence $d_M(p) = 0$. The d_M function also satisfies the following Lipschitz property proved in [5]:

Lemma 7.3.1. *Let $S \subset \mathbb{R}^3$ be a surface. Then, for any two points p and q of S, $|d_M(p) - d_M(q)| \leq \|p - q\|$.*

Assume that S is given as the zero level set, $S = f^{-1}(0)$, of a smooth function $f : \mathbb{R}^3 \to \mathbb{R}$.

Definition 7.3.3. Given a unit vector $\vec{d} = (d_1, d_2, d_3)$ in $\mathbb{R}^3$, a point $p \in S$ is said to be a *critical point of S in the direction $\vec{d}$* if and only if the normal to S at p is parallel to $\vec{d}$ (and $-\vec{d}$).

For any point $p \in S$, the gradient $\nabla f(p) = (\frac{\partial f}{\partial x}(p), \frac{\partial f}{\partial y}(p), \frac{\partial f}{\partial z}(p))$ of S at p is normal to S at p. So, the normal to S at p is parallel to $\vec{d}$ or $-\vec{d}$ when $\frac{\partial f/\partial x}{d_1} = \frac{\partial f/\partial y}{d_2} = \frac{\partial f/\partial z}{d_3}$. Hence, the critical points of S are the solution for the system of equations $f(p) = 0$, $\frac{\partial f}{\partial x} d_2 - \frac{\partial f}{\partial y} d_1 = 0$ and $\frac{\partial f}{\partial x} d_3 - \frac{\partial f}{\partial z} d_1 = 0$. We denote the set of critical points of S in the direction $\vec{d}$ as $Z_{\vec{d}}$.

Definition 7.3.4. The *silhouette* of S with respect to a unit vector $\vec{d}$ in $\mathbb{R}^3$ is the set of points $J_{\vec{d}}$,

$$J_{\vec{d}} = \{p \in S \mid \vec{n}_p \cdot \vec{d} = 0\},$$

where $\vec{n}_p$ is the unit normal to S at p.

Genericity Condition: We say that S satisfies the *genericity condition* if the set $Z_{\vec{d}}$ of critical points of S in the direction $\vec{d}$ has finitely many points and these points are "isolated", for all $\vec{d} \in \mathbb{R}^3$.

Under the genericity condition, the silhouette $J_{\vec{d}}$ of S is a set of smooth, pairwise disjoint closed curves [56]. The following lemma from [26] states an important fact regarding silhouette:

Lemma 7.3.2. *Let $S \subset \mathbb{R}^3$ be a smooth surface, and let $M \subset S$ be a connected, compact surface with boundary such that the boundary of M is a single cycle. Then, for any unit vector $\vec{d}$ in $\mathbb{R}^3$, if M does not intersect the silhouette $J_{\vec{d}}$ of S, the surface M is homeomorphic to $\overline{\mathbb{D}}^2$.*

7.4 Closed Ball Property

Recall that the goal of the algorithm in [26] is to build a Delaunay simplicial complex restricted by a given smooth surface S, $\mathcal{D}_3(W, S)$, such that the underlying surface $|\mathcal{D}_3(W, S)|$ of $\mathcal{D}_3(W, S)$ is guaranteed to be homeomorphic to S, where W is a finite set of points on S obtained by the algorithm. To ensure that $|\mathcal{D}_3(W, S)|$ is homeomorphic to S, the algorithm computes a (non-degenerate) point set W such that $\mathcal{V}(W, S)$ satisfies the closed ball property. From Theorem 7.3.1, if $\mathcal{V}(W, S)$ satisfies the closed ball property and $\mathcal{V}_W$ intersects S generically, then we know that $|\mathcal{D}_3(W, S)|$ is homeomorphic to S.

In what follows, we present several results from [26] that show that whenever the topological ball property is violated, we can find a point on S that is not in W and is at least $k \cdot d_M(p)$ away from all points in W, where $k > 0$ is a constant and p is a point in W. These results indicate which points from S the algorithm should include into W, and they will be used to show that the algorithm always terminates and produces a correct solution. Throughout this section we shall assume that $\mathcal{V}_W$ intersects S generically, and that S satisfies the genericity condition introduced in Section 7.3.

The closed ball property tells us that the intersection of S and a Voronoi edge e of the Voronoi neighborhood $\mathcal{V}(p)$ in $\mathbb{R}^3$ of any point p of W is either empty or a single point. The next lemma from [26] states that if this property is violated, then there is a point on S that is at least $k \cdot d_M(p)$ far away from all existing points in W, where k is a constant. All results from [26] described below can be shown to hold for k equal to 0.06.

Lemma 7.4.1. *Let $e \in \mathcal{V}(p)$ be a Voronoi edge of the Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$. If e intersects S in two or more points, then the intersection point of e and S which is furthest from p is at least $k \cdot d_M(p)$ away from p.*

The closed ball property tells us that the intersection between S and a Voronoi

facet f of a Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$ is either empty or homeomorphic to a closed arc. The next two lemmas from [26] state that if the closed ball property is violated for $f \cap S$, then there is a point on S that is at least $k \cdot d_M(p)$ far away from all existing points in W (refer to Figure 7.5).

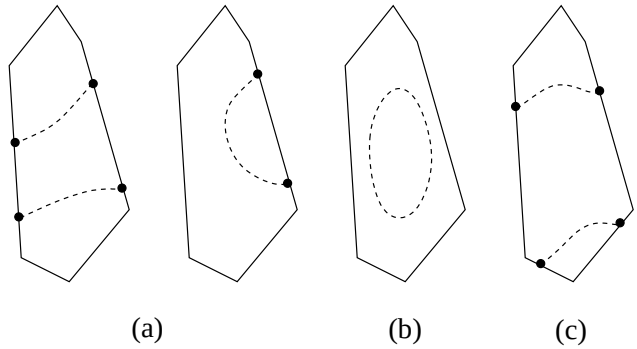


Figure 7.5: (a) An edge of a Voronoi facet intersects S in more than one point. (b) The intersection between S and a Voronoi facet is a cycle. (c) S intersects a Voronoi facets in two curve segments, and S does not intersect an edge of the facet in more than one point.

Lemma 7.4.2. *Let $f \in \mathcal{V}(p)$ be a Voronoi facet of the Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$ such that that $f \cap S$ contains a cycle C (simple and closed curve). Let l be any line in the supporting plane of f that is normal to C at a point $q \in C$. Then, the furthest point from p in the intersection $l \cap C$ between l and C is at least $k \cdot d_M(p)$ away from p.*

Lemma 7.4.3. *Let $f \in \mathcal{V}(p)$ be a Voronoi facet of the Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$ such that that $f \cap S$ contains at least two curve segments, each of which is homeomorphic to a closed arc. Further, assume that each Voronoi edge of $\mathcal{V}(p)$ intersects S in at most one point. Then, the furthest point from p which lies in a Voronoi edge of $\mathcal{V}(p)$ and in $f \cap S$ is at least $k \cdot d_M(p)$ away from p.*

The next two lemmas concern the intersection between S and the Voronoi neighborhoods of $\mathcal{V}(W)$:

Lemma 7.4.4. *Let $\mathcal{V}(p)$ be a Voronoi neighborhood of a point $p \in W$ such that $\mathcal{V}(p)$ contains a point q in the silhouette $J_{\vec{d}}$ of S, where $\vec{d}$ is the unit normal of S at p. Then, the Euclidean distance $\|p - q\|$ from p to q is at least $k \cdot d_M(p)$.*

Lemma 7.4.5. *Let $\mathcal{V}(p)$ be a Voronoi neighborhood of a point $p \in W$ such that $\mathcal{V}(p) \cap S$ is a surface with boundary. If the boundary of $\mathcal{V}(p) \cap S$ has at least two cycles, none of which resides on a single facet of $\mathcal{V}(p)$, then there is a point of S in a Voronoi edge of $\mathcal{V}(W)$ which is at least $k \cdot d_M(p)$ away from p.*

7.5 Numerical Computations

The algorithm in [26] for building a simplicial approximation to a given S uses the following subroutines to carry out numerical computations involving S and Voronoi edges and facets:

- CritSurf(S,$\vec{d}$): This subroutine solves the system of equations described before to obtain $Z_{\vec{d}}$.

- CritCurve(S,F): This subroutine computes the critical points of a height function defined on the curve of intersection between S and the supporting plane of a Voronoi face F. Let $p' = Mp$ be a linear transformation of the coordinate axes that identifies the x'-y' plane with the supporting plane of F and y' with the projection of the z-axis on the supporting plane of F. By letting $g(p') = f(M^{-1}p') = 0$, where $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ is the implicit function defining S, we obtain S in the new coordinate axis frame. The equation $e(p'_1, p'_2) = g(p'_1, p'_2, 0) = 0$ gives the implicit equation of the curve in which S intersects the supporting plane of F. The system of equations $e(p'_1, p'_2) = 0$, $\frac{\partial e}{\partial p'_1}(p'_1, p'_2) = 0$ gives the critical points of S that belong to the curve $e(p'_1, p'_2) = 0$.

- SilhFacet(S,F,$\vec{d}$): This subroutine determines the intersection points of the silhouette $J_{\vec{d}}$ of S with the supporting plane of the Voronoi facet F. let $\vec{n} \cdot (p -$

$p_0) = 0$ be the equation of the supporting plane of F, where $\vec{n}$ is a unit vector perpendicular to the supporting plane of F, and p_0 is a point in this plane. The required points are the solutions of the system of equations $f(p) = 0$, $\vec{n} \cdot (p - p_0) = 0$, and $\nabla f(p) \cdot \vec{d} = 0$.

- CritSilh(S,$\vec{d}$,$\vec{d'}$): This subroutine computes the critical points of S on the silhouette $J_{\vec{d}}$ of S, where $\vec{d'}$ is orthogonal to $\vec{d}$. The silhouette is given by two implicit equations $f(p) = 0$ and $g(p) = \nabla f(p) \cdot \vec{d} = 0$. The tangent to the silhouette at p is given by $\nabla f(p) \times \nabla g(p)$, where $\nabla g(p)$ is the gradient of g. Thus, the critical points of $J_{\vec{d}}$ along $\vec{d'}$ are the solutions of the system of equations $f(p) = 0$, $g(p) = 0$, and $(\nabla f(p) \times \nabla g(p)) \cdot \vec{d'} = 0$.

- EdgeSurface(S,e): This subroutine determines the intersection points of a Voronoi edge e with the surface S. One way to do this is to align the x-axis with the line e by using a linear transformation $p' = Mp$ and then setting p'_2 and p'_3 to zero in the equation $g(p') = f(M^{-1}p') = 0$. The equation $e(p') = g(p'_1, 0, 0) = 0$ is an equation in a single variable p'_1, and its solution gives the intersection points of S and the supporting line of e and S. Among these intersection points, we can select the ones delimited by the endpoints of e.

All subroutines above can be implemented using numerical methods available for this purpose [74]. However, the input surface S must satisfy the genericity condition stated in Section 7.3. Otherwise, the set of points computed by CritSurf(), CritCurve(), SilhFacet(), or CritSilh() may contain infinitely many points, and hence either this set will not be entirely computed or the computation will never end.

7.6 The Algorithm

The algorithm in [26] builds a simplicial surface (i.e., $\mathcal{D}_3(W, S)$) that approximates a given S. The simplicial surface is guaranteed to be homeomorphic to S, its triangles are guaranteed to be well-shaped, and its size is optimal with respect to a criterion discussed later.

7.6.1 Topological Sampling

The first step of the algorithm, called topological sampling, generates a simplicial surface that is homeomorphic to S. The idea behind this step is to insert points of S into W as long as $\mathcal{V}(W)$ does not satisfy the closed ball property. Initially, W is populated with all critical points $Z_{\vec{d}}$ of S, where $\vec{d} = (0, 0, 1)$. These points are called *seed points*, and they are computed by CritSurf(). Next, the algorithm uses four subroutines, namely VorEdge(), TopoDisk(), FacetCycle(), and Silhouette(), necessarily in this order, to insert additional points into W while $\mathcal{D}_3(W, S)$ does not satisfy the closed ball property.

We first describe VorEdge(), TopoDisk(), FacetCycle(), and, Silhouette(), and then we show that the topological sampling step ensures the closed ball property.

- VorEdge($e \in \mathcal{V}(p)$): If EdgeSurface(S,e) computes two or more points of intersection between S and e, insert the point that is furthest from p among them into W.

Let T_p be the set of triangles of $\mathcal{D}_3(W, S)$ that are incident to a point p of W. TopoDisk() verifies if the underlying space $|\mathcal{T}_p|$ of the simplicial complex $\mathcal{T}_p$ consisting of the triangles in T_p, along with their vertices and edges, is homeomorphic to a disk. This is done by first checking if each edge incident to p in T_p has exactly two incident triangles from T_p. If so, TopoDisk() checks if there is exactly one

cycle of triangles in T_p around p. Note that $|\mathcal{T}_p|$ is homeomorphic to $\overline{\mathbb{D}}^2$ if and only if none of the two tests fail.

- TopoDisk()(p): If $|\mathcal{T}_p|$ is not homeomorphic to $\overline{\mathbb{D}}^2$, insert into W the intersection point between S and a Voronoi edge of $V(p)$ which is furthest from p.

FacetCycle() checks if a facet of $\mathcal{V}(W)$ intersects S in a cycle. If so, it does insert a point from the cycle into W. This point is computed by CritCurve(). Otherwise, it does nothing. Silhouette() checks if a Voronoi neighborhood of $\mathcal{V}(W)$ contains a point of the silhouette. If so, this point is inserted into W. Silhouette() uses CritSilh() and SilhFacet(). Both FacetCycle() and Silhouette() are described in details below.

- FacetCycle($f \in \mathcal{V}(p)$): Let Y be the point set given by CritCurve(S, f). If $q \in Y$ lies inside $\mathcal{V}(p)$, consider the line l which is the projection onto f of the line through q and parallel to the z-axis. Note that l is normal to the intersection curve of S and f at q by construction. If l intersects S at any point other than q in $\mathcal{V}(p)$, insert into W the point among such intersection points which is furthest from p.

- Silhouette($\mathcal{V}(p)$): Compute the unit normal $\vec{n}_p$ of S at p. This is done by computing the gradient $\nabla f(p)$ of the implicit function f defining S. Choose a unit vector $\vec{d}$ orthogonal to $\vec{n}_p$. Let Y be the point set computed by CritSilh($S, \vec{n}_p, \vec{d}$). If $Y \cap \mathcal{V}(p) \neq \emptyset$ then insert one point of $Y \cap \mathcal{V}(p)$ into W. Otherwise, for each facet $g \in \mathcal{V}(p)$, let U be the point set computed by SilhFacet($S, g, \vec{d}$). If $U \cap g \neq \emptyset$ for some facet g then insert one point of $U \cap g$ into W.

VorEdge(), TopoDisk(), FacetCycle(), and Silhouette() are called by a subroutine called Topology() as follows:

- TOPOLOGY(W)

    1. Check if any of VOREDGE(), TOPODISK(), FACETCYCLE(), and SILHOUETTE(), necessarily in this order, inserts a point into W. If so, update $\mathcal{V}(W)$ and repeat this step.
    2. Return W.

As we mentioned before, topological sampling initializes W with the seed points, and then calls TOPOLOGY(W). The next lemma from [26] states that if TOPOLOGY() terminates, then $\mathcal{V}(W)$ satisfies the closed ball property.

Lemma 7.6.1. *If the input point set W to* TOPOLOGY*() includes all critical points of S for a direction and* TOPOLOGY*() terminates, the closed ball property holds for $\mathcal{V}(W)$ at the end of its execution.*

Proof. Since TOPOLOGY() terminates, VOREDGE() cannot insert a point into W. This means that the intersection between any edge of $\mathcal{V}(W)$ and S is either empty or a single point. So, the closed ball property holds for all edges of $\mathcal{V}(W)$. Aiming at a contradiction, assume that a Voronoi facet f of $\mathcal{V}(W)$ intersects S in more than one curve segment, or cycle, or a combination or both. If there is more than one curve segment, S must intersect more than two Voronoi edges of f, as no Voronoi edge intersects S in more than one point. This means that the dual Delaunay edge of f in $\mathcal{D}_W$ is incident to more than two triangles of $\mathcal{D}_3(W, S)$. But, this violates the topological disk condition imposed by TOPODISK() for some point $p \in W$. So, TOPODISK() would have inserted a point in W, which contradicts the fact that TOPOLOGY() terminates. Now, consider the case in which f intersects S in some cycle C. But, if this is the case, then FACETCYCLE() would have inserted a point into W, which contradicts the fact that TOPODISK() terminates. So, the intersection between S and a Voronoi facet f of $\mathcal{V}(W)$ must be either empty or a curve segment homeomorphic to a closed arc. Thus, the closed ball property also holds for the facets of $\mathcal{V}(W)$. Now, consider the intersection between S and a Voronoi neighborhood $\mathcal{V}(p)$

of $\mathcal{V}(W)$, for some $p \in W$. Since S is a surface without boundary, and since the closed ball property holds for the edges and facets of $\mathcal{V}(W)$, the intersection between S and $\mathcal{V}(p)$ is either empty or a surface possibly with boundary. This surface cannot contain a connected component M without boundary, as the maxima and minima of M are computed by CritSurf and inserted into W before Topology() is executed. So, we are left with the cases in which (1) M is a connected, compact surface with a single boundary cycle, and (2) M is a connected, compact surface with more than one boundary cycle. Consider case (1). We claim that M is homeomorphic to $\mathbb{D}^2$, (i.e., a disk or closed 2-ball). Otherwise, from Lemma 7.3.2, M must intersect the silhouette $J_{\vec{d}}$ of S, where $\vec{d}$ is the unit normal $\vec{n}_p$ of S at p. Let $\vec{d'}$ be the direction orthogonal to $\vec{d}$ chosen by Silhouette() for its computation of the set W. If M contains a closed curve from $J_{\vec{d}}$, then there exists a point in $J_{\vec{d}}$ which is critical along the direction $\vec{d'}$. This point is computed by Silhouette(), which would trigger its insertion into W, contradicting the fact that Topology() terminates. If M does not contain any closed curve from $J_{\vec{d}}$, a curve from it must intersect a Voronoi facet f of $\mathcal{V}(W)$. So, Silhouette() computes a point using SilhFacet() which lies in f. Again, this would trigger the insertion of this point into W, which contradicts the fact that Topology() terminates. Finally, consider case (2). Since the closed ball property holds for the edges and facets of $\mathcal{V}(W)$, each boundary cycle of M intersects a cycle of Voronoi edges. The dual of each cycle of edges is a set of triangles of $\mathcal{D}_3(W, S)$ around p. Since none of the Voronoi edges intersect S in more than one point, each cycle induces a distinct and disjoint cycle of triangles around p. But, this violates the topological disk condition imposed by TopoDisk(), and therefore TopoDisk() would trigger the insertion of a point into W, which contradicts the termination of Topology(). So, the intersection between S and $\mathcal{V}(p)$ must be either empty or homeomorphic to a closed 2-ball. □

The next lemma states a result that ensures the termination of Topology(). Basically, we can show that every point inserted by Topology() into W is at

least $k \cdot d_M(p)$ away from its closest point $p \in W$. Since S is compact, TOPOLOGY() cannot keep inserting points into W indefinitely. From Lemma 7.6.1, the termination of TOPOLOGY() implies that the closed ball property holds for $\mathcal{V}(W)$, and thus $\mathcal{D}_3(W, S)$ is indeed a simplicial surface whose the underlying surface is homeomorphic to S.

Lemma 7.6.2. *Any point inserted by* TOPOLOGY*() into W is more than $k \cdot d_M(p)$ away from its closest point p in W.*

Proof. Lemma 7.4.1, Lemma 7.4.2, and Lemma 7.4.4 imply that any point inserted by VOREDGE(), FACETCYCLE(), and SILHOUETTE() into W is at least $k \cdot d_M(p)$ away from its closest point p in W. So, we are left with the analysis of TOPODISK(). Recall that the triangles of $\mathcal{D}_3(W, S)$ incident to a point p of W violate the topological disk condition if (1) an edge is not incident to two triangles, or (2) two or more cycles of triangles are incident to p. Consider case (1). If an edge e of $\mathcal{D}_3(W, S)$ is incident to only one triangle of $\mathcal{D}_3(W, S)$, then S intersects the dual face f of e in a single Voronoi edge. Since S has no boundary, $f \cap S$ cannot have an endpoint that is not on e. So, S must intersect e in more than one point. But, this is not possible, as VOREDGE() has eliminated such cases before the execution of TOPODISK(). If there are three or more triangles of $\mathcal{D}_3(W, S)$ incident to e, then the dual face f of e in $\mathcal{V}(W)$ intersects S in more than one curve segment. From Lemma 7.4.3, TOPODISK() inserts a point into W that is at least $k \cdot d_M(p)$ away from p. Finally, consider case (2). If there are more than one cycle of triangles of $\mathcal{D}_3(W, S)$ around p, then $\mathcal{V}(p) \cap f$ has at least two boundary cycles, none of which lie completely in a Voronoi facet of $\mathcal{V}(p)$. From Lemma 7.4.5, TOPODISK() inserts a point into W that is at least $k \cdot d_M(p)$ away from p. □

7.6.2 Deleting Seed Points

As we shall see later, if *all* points of W are at least k times the local feature size distance away from its closest point in W, the cardinality $|W|$ of W is "optimal"

with respect to a criterion described later. From Lemma 7.6.2, the above property holds for all points of W inserted by TOPOLOGY(), but it may not hold for the seed points (i.e., the ones computed by CRITSURF()), as the set of seed points can be arbitrarily dense.

The next lemma states that we can achieve size optimality by deleting the seed points after TOPOLOGY() terminates, and then restoring the closed ball property (if needed) using TOPOLOGY() again. Since every point inserted into W by TOPOLOGY() is at least k times the local feature size distance away from its closest point in W, all points of the resulting set W are at least k times the local feature size distance away from their closest point in W.

Lemma 7.6.3. *If $\mathcal{V}(W)$ satisfies the closed ball property, then* TOPOLOGY*() can restore the closed ball property given $W \setminus \{p\}$ as input, where p is a point used to initialize W, i.e., computed by* CRITSURF*().*

Putting all together, the topological sampling step can be described as follows: Let $\vec{d} = (0, 0, 1)$ and $W =$ CRITSURF(S,$\vec{d}$), and call the subroutine SAMPLETOPOLOGY(W), where

- SAMPLETOPOLOGY(W)

    1. TOPOLOGY(W)
    2. While there is a point p in W that was created by CRITSURF(), delete p from W and call TOPOLOGY(W).

7.6.3 Quality

The next step of the algorithm ensures that all triangles of $\mathcal{D}_3(W, S)$ are well-shaped. Here, the shape quality of a triangle t of $\mathcal{D}_3(W, S)$ is measured by its radius-edge ratio, $\rho(t)$ (see Definition 4.3.1). A triangle is considered well-shaped if its radius-edge ratio is bounded from above [122].

To achieve shape quality, the algorithm in [26] inserts a point into W as long as there is a triangle t of $\mathcal{D}_3(W,S)$ such that $\rho(t) > \rho_0 = (1+k)^2$. More specifically, if t is a triangle such that $\rho(t) > \rho_0 = (1+k)^2$, then the algorithm computes the intersection point p between S and the dual Voronoi edge of t in $\mathcal{V}(W)$, and inserts p into W. Note that if this process terminates, every triangle of $\mathcal{D}_3(W,S)$ has radius-edge ratio no larger than ρ_0.

- QUALITY(W): While there is a triangle t with $\rho(t) > (1+k)^2$ in $\mathcal{D}_3(W,S)$, insert the point of intersection between S and the dual Voronoi edge of t in $\mathcal{V}(W)$ into W, and update $\mathcal{V}(W)$.

Note that the insertion of a point into W by QUALITY() may change the topology of $\mathcal{D}_3(W,S)$. Consequently, the dual edge of a triangle of $\mathcal{D}_3(W,S)$ may intersect S in more than one point. If this is the case, QUALITY() inserts the furthest intersection point from its closest point in W. Furthermore, TOPOLOGY() must be called next to restore the closed ball property. It is worth to notice that there is no need for searching the entire Voronoi diagram $\mathcal{V}(W)$ for topology violations, as each point insertion by QUALITY() changes $\mathcal{V}(W)$ locally only.

7.6.4 Smoothness

Unlike the algorithm by Boissonnat and Oudot [15], the algorithm in [26] does not provide any bound on how "close" $|\mathcal{D}_3(W,S)|$ is from S. However, the algorithm in [26] provides an interesting property that can be viewed as an indirect measure of how close $|\mathcal{D}_3(W,S)|$ is from S. Before defining this property, we define the *roughness* of an edge e of $\mathcal{D}_3(W,S)$ as follows: The roughness $g(e)$ of e is given as $g(e) = \pi - \theta$, where θ is the internal dihedral angle of $\mathcal{D}_3(W,S)$ at e, i.e., the angle defined by the normals of the two triangles of $\mathcal{D}_3(W,S)$ incident to e. We can improve the approximation $\mathcal{D}_3(W,S)$ of S by keeping sampling S until all edges have roughness below a threshold:

- SMOOTH(W): While there is an edge $[p,q] \in \mathcal{D}_3(W,S)$ such that $g([p,q]) > 2(\arcsin(k) + \arcsin(\frac{2k}{\sqrt{3}}))$, insert the point furthest from p among all intersections of the Voronoi edges of $\mathcal{V}(p)$ and S and update $\mathcal{V}(W)$.

Lemma 7.6.4. *Any point inserted by* SMOOTH*() is more than $k \cdot d_M(p)$ away from its closest neighbor in W.*

7.6.5 Termination

In what follows, we describe the entire algorithm. We also introduce a theorem from [26] that guarantees the termination of the algorithm:

- DELMESH(S)

  1. Compute $W =$ CRITSURF(S,(0,0,1))
  2. $W_0 = W$
  3. SAMPLETOPOLOGY(W)
  4. QUALITY(W)
  5. SMOOTH(W)
  6. Go back to 2 if $W_0 \neq W$
  7. Output $\mathcal{D}_3(W,S)$

Theorem 7.6.1. DELMESH*() terminates.*

Proof. We claim that other than the seed points, any two points p and q in W at any stage of DELMESH() have distance $\|p-q\| \geq k \cdot d_M(p)/(1+k)$. We proof this claim by induction on the number of points inserted into W after the seed points. Note that a point is inserted into W either by SAMPLETOPOLOGY(), QUALITY(), or SMOOTH(). From Lemma 7.6.2, if p is inserted into W by SAMPLETOPOLOGY(), then p is at least $k \cdot d_M(q)$ away from its closest point q in W. From Lemma 7.3.1, we have that

$d_M(q) \geq d_M(p) - \|p-q\|$, which implies that $\|p-q\| \geq k \cdot d_M(p)/(1+k)$. Note also that the first point inserted into W must be inserted by SAMPLETOPOLOGY(), as all seed points are deleted from W before QUALITY() or SMOOTH() is called. So, our claim holds for the first point inserted into W after the seed points. Now, assume that our claim holds for the n-th point inserted into W, where $n \geq 1$, and consider the insertion of the $(n+1)$-th point p. If p is inserted by SAMPLETOPOLOGY(), we are done. Otherwise, it must be inserted by either QUALITY() or SMOOTH(). If p is inserted by SMOOTH(), Lemma 7.6.4 tells us that $\|p-q\| > kd_M(p)$, where q is the closest point to p in W. Thus, $\|p-q\| > \frac{k}{k+1} d_M(p)$, and hence our claim holds. Now, consider the case in which p is inserted by QUALITY(). Recall that p is the intersection point between S and the dual Voronoi edge of a triangle t of $\mathcal{D}_3(W,S)$ such that $\rho(t) > \rho_0$. So, p is more than $(1+k)^2 l$ away from its closest existing points in W, where $l = \|q-s\|$ is the length of the edge of $\mathcal{D}_3(W,S)$ with endpoints q and s, and q is one of the closest points to p. This is because $\|p-q\|$ is no smaller than the circumradius of t. By the inductive hypothesis, $l = \|q-s\| \geq k \cdot d_M(q)/(1+k)$, as all seed points have been deleted from W. Hence, $\|p-q\| > k(k+1) \cdot d_M(q)$. From Lemma 7.3.1 again, we have that $\|p-q\| > k \cdot d_M(p) > k \cdot d_M(p)/(1+k)$. Let $f_m = \min_{p \in S} d_M(p)$. Since S is smooth, we must have $f_m > 0$. So, $l_m = kf_m/(k+1) > 0$. Since any two points inserted into W have more than l_m distance and $l_m > 0$, and since S is compact, DELMESH() can insert only finitely many points into W, and thus it must terminate. □

7.6.6 Size Optimality

We mentioned before that the cardinality $|W|$ of the set W computed by DELMESH() is a linear factor of the cardinality of the "optimal" set. By optimal set, we actually mean the smallest *ϵ-sample* set of S, for any $\epsilon < 0.2$. A set Y of points of S is said to be an ϵ-sample if, for each $p \in S$, there exists a point $q \in Y$ such that $\|p-q\| \leq \epsilon \cdot d_M(p)$. Amenta and Bern [5] showed that if Y is ϵ-sample of S for

$\epsilon \leq 0.1$, then $\mathcal{D}_{Y,S}$ is a simplicial surface whose underlying surface is homeomorphic to S.

Erickson [43] showed that the number of points in any ϵ-sample of S, for $\epsilon < 0.2$, is $\Omega(\int_S \frac{1}{d_M(p)} dp)$. The following theorem from [26] uses the result in [43] and states that W has at most $c \cdot \int_S \frac{1}{d_M(p)} dp$ points, where c is a positive constant. Note that W is not necessarily an ϵ-sample of S. So, the size optimality of W should be considered here with respect to the size of the smallest ϵ-sample Y of S, for $\epsilon < 0.2$, rather than the smallest point set Y of S for which the Voronoi diagram $\mathcal{V}_Y$ satisfies the closest ball property.

Theorem 7.6.2. *The number of points in W is within a constant factor of any ϵ-sample of S, for any $\epsilon < 0.2$.*

Chapter 8

Surface Meshing: Part II

This chapter presents our simplification of the algorithm by Cheng, Dey, Ramos, and Ray [26], which was described in Chapter 7. We also discuss some implementation details, and show some surface meshes generated by a preliminary implementation of the algorithm.

8.1 Overview of the Simplified Algorithm

Let S denote a smooth and compact surface such that $S = f^{-1}(0)$, where $f : conv(D) \rightarrow \mathbb{R}$ is an implicit function defined from a given well-composed image (D, X) by our construction in Chapter 6. Here, we describe a simplified version of the algorithm by Cheng, Dey, Ramos, and Ray [26] which computes a finite set W of points on S such that the Voronoi diagram restricted to S, $\mathcal{V}(W, S)$, satisfies the closed ball property.

Our simplification of the algorithm in [26] consists of replacing the computation of critical and silhouette points by simpler computations. Recall from Chapter 7 that critical and silhouette points are used by the algorithm for initializing W, so that W has at least two points of every connected component of S and no handle of S is entirely inside a Voronoi region of $\mathcal{V}(W)$. Critical and silhouette points are

also used by FacetCycle() to verify if the intersection of S and a Voronoi facet of $\mathcal{V}(W)$ contains a cycle.

We use the fact that S is homeomorphic to $bdCA(X)$ to carry out our simplification. Let $\{C_k\}_{k\in K}$ be the set of cubes of the subdivision of $conv(D)$ used by the construction of S in Chapter 6. We know that there exists a homeomorphism $h : bdCA(X) \rightarrow S$ such that h maps each connected component of $bdCA(X)$ into a connected component of S that intersects the same subset of cubes of $\{C_k\}_{k\in K}$. This observation enables us to easily obtain points in every connected component of S.

More specifically, to obtain at least two points from each connected component of S, we first obtain two points from each connected component of $bdCA(X)$. Next, we locate the cubes of $\{C_k\}_{k\in K}$ that contain the two points. Recall that the intersection of $bdCA(X)$ (resp. S) and each edge of such a cube is either empty or one point of $bdCA(X)$ (resp. S). Furthermore, if the cube intersects $bdCA(X)$ (resp. S) then at least three edges of the cube intersect $bdCA(X)$ (resp. S). So, for each cube, we choose an edge that intersects $bdCA(X)$ (resp. S), and we obtain the intersection point between S and the edge. These points are inserted into W. As a result, no Voronoi region of $\mathcal{V}_W$ can entirely contain a connected component of S, but they can contain handles.

Unlike the algorithm in [26], we do not insert points in W to prevent a Voronoi region of $\mathcal{V}_W$ from containing an entire handle of S. Instead, we developed a simple procedure to check for the existence of a handle of S inside a Voronoi region of $\mathcal{V}_W$. This procedure basically computes the Euler characteristic of a simplicial approximation for the portion of S inside Voronoi region. Also, the procedure is executed only if the algorithm finds out that the intersection of S and the facets of a Voronoi region is a simple cycle. We provide the details of this handle detection procedure in Section 8.3.

Note that the above operations do not involve any complicated numerical and

geometric calculations. *Unfortunately, we have not been able to come up with similar operations for verifying if the intersection of S and a Voronoi facet of $\mathcal{V}(W)$ contains a cycle. However, we were still able to go around the need for computing critical and silhouette points.* We actually use a "brute force" geometric approach to verify the existence of a cycle in the intersection of S and a Voronoi facet of $\mathcal{V}(W)$. This approach is used by the subroutine FacetCycle(), and its details are described in Section 8.2.

From the above discussion, we can see that our simplified algorithm does not contain any of the numerical subroutines in Section 7.5 (except for EdgeSurface()) nor the subroutine Silhouette(), but it does contain all remaining subroutines. Furthermore, out of all remaining subroutines, only EdgeSurface() and FacetCycle() are affected by our simplifications. We further comment on their changes in Section 8.2.

8.2 EdgeSurface() and FacetCycle()

EdgeSurface() takes as input a Voronoi edge e and returns *all* intersection points of S and e. This subroutine is used by the subroutine VorEdge() only, which in turn takes as input a point p of W (a site of $\mathcal{V}(W)$) and an edge e of $\mathcal{V}(p)$, and then calls EdgeSurface(). If EdgeSurface() returns two or more intersection points of e and S, VorEdge() inserts the point that is furthest from p into W. Otherwise, VorEdge() takes no action.

A Voronoi edge e can be a line, a half-line, or a line segment. However, since we are interested in the intersection points of e and S, and since S is inside a compact domain, $conv(D)$, we can always restrict our attention to the line segment that results from the intersection of e and $conv(D)$. So, the first step of EdgeSurface() is to compute this intersection, and therefore we can assume that the edge e is a line segment.

The next step is to compute the intersection points of e and S. Since this is a time-consuming and reasonably complicated operation, we borrowed some approaches from the ray tracing literature to speed up and facilitate our computations. First, we consider the set $\{C_k\}_{k \in K}$ of cubes of the subdivision of $conv(D)$ used by the construction of S (see Chapter 6). We first identify the cubes that are intersected by e and then discard the cubes that do not intersect S. To identify the cubes intersected by e, we use a fast and simple algorithm for voxel traversal by Amanatides and Woo [4].

We consider the cubes that intersect S one at a time according to a traversal of e from one of its endpoints to the other. For each cube, we first compute a very fine triangulation of the "disk-like patch" of S inside the cube, and then we look for an intersection point between S and the triangulation. If an intersection point exists, say q, we consider a small interval of e around q, and look for an intersection point between S and e in this interval. We use the bisection method to look for the intersection point.

If we find no intersection point after examining all cubes that intersect S, we are done. Otherwise, after finding out the first intersection point, we reverse the order in which the cubes are considered before looking for a second intersection point. The reason is that if there are two or more intersection points, only the furthest intersection point from a given Voronoi site will be of interest for VorEdge(). Since e is a convex set, such furthest point must be either the first or the last intersection point found by traveling from one endpoint of e to another. This means that we have to look for at most two intersection points. So, we can end our search as soon as we find a second intersection point or reach the first intersection point computed before.

FacetCycle() checks if the intersection of S and a given facet of $\mathcal{V}(W)$ is a cycle. If so, it inserts a point from the cycle into W. Rather than using critical and silhouette points to perform this verification, we look for cycles in a "brute force"

way. Given a facet h of a Voronoi region $\mathcal{V}(p)$ of $\mathcal{V}(W)$, where $p \in W$, we first identify the cubes of $\{C_k\}_{k \in K}$ that intersect both h and S. For each of these cubes, we compute a very fine triangulation of the "disk-like patch" of S inside the cube. Next, we compute the intersection of h and all triangulated patches, and look for a cycle.

Note that *the cycle computed by* FacetCycle*() is actually a closed polygon, which is assumed to be an approximation for a cycle (a smooth curve) resulting from the intersection of S and h.* To improve the approximation, we project the vertices of the cycle (i.e., a closed and simple polygon) onto S by iteratively changing the position of the vertices until $|f(v)|$ is smaller than a positive constant ϵ. In each iteration, the new position of a vertex v is given by by $v - t \cdot \text{sign}(f(v)) \cdot (\vec{p}_v / \|\vec{p}_v\|)$, where t is a positive number, f is the implicit function whose zero level set is S, and

$$\text{sign}(f(v)) = \begin{cases} 1, & \text{if } f(v) > 0 \\ -1, & \text{if } f(v) < 0 \\ 0, & \text{if } f(v) = 0 \end{cases}$$

and $\vec{p}_v$ is the orthogonal projection of $\nabla f(v)$ onto the supporting plane of h. Figure 8.1 illustrates this procedure. The value of t is set to 0.0001 for the first iteration. From the second iteration on, the value of t is half of its value in the previous iteration. Note that since the vertices of the cycle are projected onto S, the vertices of the cycle are all points of S.

Now, we need to find a vertex of the cycle that is at least $k \cdot d_M(p)$ away from the Voronoi site p whose region contains the facet h, where k is the constant 0.06 and $d_M(p)$ is the distance to the medial axis function (see Section 7.4). To do that, we choose any vertex q of the cycle. By construction, we know that $q \in S$. Next, we compute the unit normal $\vec{n}_q$ to S at q. Recall that $\vec{n}_q = \nabla f(q) / \|\nabla f(q)\|$. Finally, we consider the orthogonal projection $\vec{v}_q$ of $\vec{n}_q$ onto the supporting plane of the Voronoi facet h of $\mathcal{V}(W)$ that contains the cycle. Note that $\vec{v}_q$ must be orthogonal to the cycle at q.

Since $\vec{v}_q$ is orthogonal to the cycle at q, the line l passing through q in the direction of $\vec{v}_q$ must intersect the cycle $S \cap h$ in at least one more point, say r. To compute r, we first find one intersection point, r', between l and our approximate cycle (the polygonal one), and then project r' along l onto S using the same strategy we used before for computing the vertices of the cycle. When r' reaches S, r' becomes the point r. According to Lemma 7.4.2, either q or r or both are at least $k \cdot d_M(p)$ away from p.

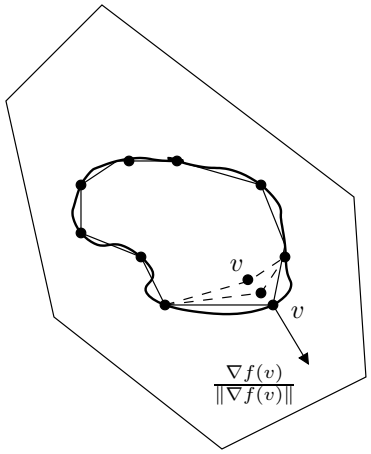


Figure 8.1: Example of how FacetCycle() works.

Recall from the description of the algorithm in [26] in Chapter 7 that VorEdge() (and therefore EdgeSurface()) is called by the subroutine Topology() for every edge e of $\mathcal{V}(W)$. Unlike, FacetCycle() is called by Topology() only if VorEdge() and TopoDisk() fail to insert a vertex into W. In other words, FacetCycle() is executed either if the intersection of S and the edges of a facet of $\mathcal{V}(W)$ is empty, or if S intersects only two edges of a facet of $\mathcal{V}(W)$ generically. Since FacetCycle() is more time-consuming than both VorEdge() and TopoDisk(), this order of execution benefits the performance of the algorithm, as FacetCycle() is executed less often.

8.3 Detecting Handles

As we pointed out before, our simplified version of the algorithm in [26] does not compute critical and silhouette points of S. So, our simplified algorithm does not contain the subroutine SILHOUETTE(), whose role is to prevent the existence of handles inside Voronoi regions of $\mathcal{V}(W)$. Instead, we developed a subroutine, called HANDLEDETECTION(), that detects (instead of preventing) the existence of handles inside a given Voronoi region of $\mathcal{V}(W)$.

HANDLEDETECTION() takes as input a Voronoi site p, i.e., a point of W, and inserts a point into W if $\mathcal{V}(W)$ contains a handle of S. Otherwise, it does nothing. HANDLEDETECTION() is called by TOPOLOGY() only if VOREDGE(), TOPODISK(), and FACETCYCLE() fail to insert a point into W while processing edges and facets of $\mathcal{V}(p)$.

Note that if VOREDGE(), TOPODISK(), and FACETCYCLE() fail to insert a point into W while processing edges and facets of $\mathcal{V}(p)$, then the intersection of S with the set of facets of $\mathcal{V}(p)$ is either empty or a single cycle. In the former case, we can conclude that the intersection of S and $\mathcal{V}(p)$ is empty, as we initialized W with at least two points per connected component. In the latter case, the intersection of S and $\mathcal{V}(p)$ is either (1) homeomorphic to $\mathbb{D}^2$ or (2) homeomorphic to a compact surface with boundary (the cycle) and with one or more handles (see Figure 8.2). Furthermore, the intersection of each facet of $\mathcal{V}(p)$ and S is either empty or homeomorphic to a line segment.

The goal of HANDLEDETECTION() is to detect case (2), and if it indeed occurs, HANDLEDETECTION() chooses a point q from $S \cap \mathcal{V}(p)$ and inserts it into W. If the point q inserted by HANDLEDETECTION() into W is $k \cdot d_M(q)$ away from p, then we can be sure that, after finitely many insertions of such points into W, the most recently updated Voronoi region $\mathcal{V}(p)$ will no longer contain a handle (see the proof of Theorem 7.6.1).

To detect a handle, HANDLEDETECTION() relies on the Euler characteristic of a

triangulation of the portion of S inside $\mathcal{V}(p)$. Let M be a compact surface (with or without boundary) in $\mathbb{R}^n$, and let $\mathcal{T}$ be any simplicial surface (triangulated surface) whose underlying surface $|\mathcal{T}|$ is homeomorphic to M. Then, we define the Euler characteristic of M as follows [13]:

Definition 8.3.1. The *Euler characteristic* $\chi(M)$ *of* M as the invariant number $\chi(M) = n_v - n_e + n_t$, where n_v is the number of vertices of $\mathcal{T}$, n_e is the number of edges of $\mathcal{T}$, and n_t is the number of triangles of $\mathcal{T}$.

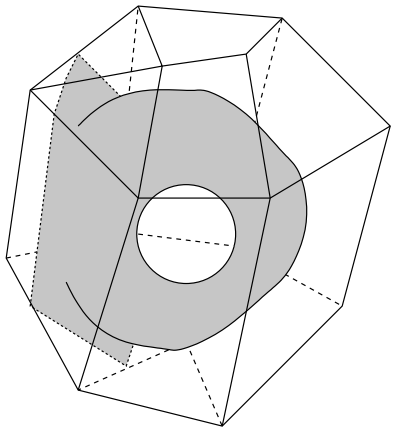


Figure 8.2: The intersection of a surface and the facets of a Voronoi region is a closed and simple polygon (a cycle). Each edge of the polygon belongs to a distinct facet of the Voronoi region. The portion of the surface inside the Voronoi region contains a handle.

Now, assume that the intersection of S and $\mathcal{V}(p)$ is either (1) homeomorphic to $\mathbb{D}^2$ or (2) homeomorphic to a compact surface with boundary (the cycle) and with one or more handles. If $\mathcal{T}$ is any simplicial surface whose underlying surface is homeomorphic to $S \cap \mathcal{V}(p)$, then $S \cap \mathcal{V}(p)$ is homeomorphic to $\mathbb{D}^2$ if and only if $\chi(S \cap \mathcal{V}(p))$ is equal to 1, as the Euler characteristic of a surface M with boundary is equal to $2 - c - 2g$, where c is the number of disjoint boundary contours of M and g is the genus of M [2].

To compute such a simplicial surface $\mathcal{T}$, HandleDetection() starts by finding the cubes of $\{C_k\}_{k\in K}$ that intersect the facets of $\mathcal{V}(p)$. Next, HandleDetection() computes a fine triangulation of the disk-like patches of S inside these cubes, and then computes the intersection of the facets of $\mathcal{V}(p)$ with the triangulations. Note that the result of this intersection is a closed polygon that approximates the cycle C defined by the intersection of S and the facets of $\mathcal{V}(p)$. Note also that C splits S into two connected components: the one inside $\mathcal{V}(p)$ (i.e., $S \cap \mathcal{V}(p)$) and the one outside $\mathcal{V}(p)$.

Let A be a set whose elements are the cubes of $\{C_k\}_{k\in K}$ that intersects an edge of our approximation for the cycle C. The cubes intersect $S \cap \mathcal{V}(p)$. Our next step is to search for the remaining cubes of $\{C_k\}_{k\in K}$ that intersect $S \cap \mathcal{V}(p)$. To do that, we first consider the set B of cubes of $\{C_k\}_{k\in K}$ that are not in A and that intersect both S and $\mathcal{V}(p)$. Note that a cube may intersect both S and $\mathcal{V}(p)$, but not $S \cap \mathcal{V}(p)$. In what follows, we explain how to find out which cubes of B do not intersect the component $S \cap \mathcal{V}(p)$.

The idea is to compute the triangulation of the disk-like patches of S inside the cubes of set B, so that these triangulations agree along the cube faces, and that they also agree with the triangulations inside the cubes of A along the common faces shared by a cube of A and a cube of B. Next, we compute the dual graph of the triangulations inside all cubes of A and B.

Note that the triangles of the triangulations inside the cubes of A must be represented by the same connected component G of the dual graph, as they cover the edges of our polygonal approximation for the cycle C. Note also that the triangles inside any cube of B that intersects $S \cap \mathcal{V}(p)$ must also be represented by vertices of G, while the triangles that are not represented by vertices of G must be in cubes of B that do not intersect $S \cap \mathcal{V}(p)$. So, we can compute $\chi(S \cap \mathcal{V}(p))$ by considering the dual triangulation of G.

If $\chi(S \cap \mathcal{V}(p))$ is not equal to 1, HandleDetection() chooses a point of $S \cap \mathcal{V}(p)$

to insert into W. This point is any vertex q of the dual triangulation of G that is in the *interior* of $\mathcal{V}(p)$ and maximizes $\|q-p\|$. This is because we want $\|q-p\|$ to be at least $k \cdot d_M(p)$. From Lemma 7.3.2, we know that such a q exists. However, since HANDLEDETECTION() does not compute silhouette points, we must guarantee that the point q returned by HANDLEDETECTION() is as far from p as some silhouette point.

Although we do not provide a guarantee that the point q returned by HANDLEDETECTION() is at least $k \cdot d_M(p)$ away from p, our implementation of HANDLEDETECTION() refines the triangulation of $S \cap \mathcal{V}(W)$ used to detect the handle, and then chooses an interior vertex q of this triangulation that maximizes $\|q-p\|$. Furthermore, we also mark q for deletion after TOPOLOGY() ensures the closed ball property. If deleting q from W causes TOPOLOGY() to call HANDLEDETECTION() again, we put q back in W.

The fact that the point q chosen by HANDLEDETECTION() may not be $k \cdot d_M(p)$ away from p does not imply that our algorithm does not terminate. This is because HANDLEDETECTION() will be called at most finitely many times by our algorithm, as the number of handles of S is finite and the points inserts into W by HANDLEDETECTION() are not too close to the existing Voronoi sites. However, the resulting simplicial surface may not satisfy the size optimality criterion described in Section 7.6.6, as the points inserted into W by HANDLEDETECTION() might not be removed from W later on.

8.4 Coping with Degeneracies

So far, we have discussed our simplified version of the algorithm in [26] without taking into account eventual degenerate cases. In other words, we have not addressed the fact that $\mathcal{V}(W)$ may not intersect S generically. More specifically, we have to deal with three issues:

(1) a vertex of $\mathcal{V}(W)$ is a point of S,

(2) an edge of $\mathcal{V}(W)$ does not intersect S generically (i.e., the edge intersects S at a single point and belongs to the tangent plane of S at this point), and

(3) a facet of $\mathcal{V}(W)$ does not intersect S generically (i.e., the intersection of S and the interior of the facet contains a point at which the facet is tangent to S).

Since S is defined by an implicit function, f, it is straightforward to verify if (1) occurs: we just have to test if $f(q)$ is 0 for every vertex q of $\mathcal{V}(W)$. If $f(q)$ is 0 for a vertex q of $\mathcal{V}(W)$, we perturb the position of any of the Voronoi sites whose Voronoi region contains q. By perturbing, we mean to slightly move the Voronoi site along S and recompute the Voronoi diagram. We check for (1) every time a new point is inserted into W.

The fact that S is defined by an implicit function, f, can also be used for detecting (2). If q is an intersection point between S and an edge e of $\mathcal{V}(W)$, we compute $\nabla f(q)$ and check if $\nabla f(q)$ is perpendicular to e. We check for (2) every time EdgeSurface() returns a single intersection point. Due to the following lemma from [26], if (2) occurs then the point q must be at least $k \cdot d_M(p)$ from the Voronoi site p of any Voronoi region $\mathcal{V}(p)$ that contains e. So, we insert q into W to remove the degeneracy, and this insertion will not affect the termination or size optimality of the algorithm.

Lemma 8.4.1. [26] *Let $e \in \mathcal{V}(p)$ be a Voronoi edge of the Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$. If e intersects S tangentially at a single point, then the intersection point of e and S which is furthest from p is at least $k \cdot d_M(p)$ away from p.*

In principle, we can deal with (3) in a way much like the one we dealt with (2). Suppose that the interior of a facet h of $\mathcal{V}(W)$ intersects S tangentially at a point q. Then, we can prove that q is at least $k \cdot d_M(p)$ away from the Voronoi site p of

any Voronoi region $\mathcal{V}(p)$ that contains h. So, to get rid of the degeneracy, we can insert q into W, and this insertion will not affect the termination or size optimality of the algorithm. To prove our claim, we will need a corollary from Lemma 2 of Amenta and Bernstein [5] and also a lemma from Cheng, Dey, Edelsbrunner, and Sullivan [24]:

Corollary 8.4.1. [5] *Let p and q be any two points on S so that $\|p-q\| \leq c \cdot d_M(p)$ for $c < 0.25$. Then, the acute angle $\angle \vec{n}_p, \vec{n}_q$ between the lines supporting the vectors $\vec{n}_p$ and $\vec{n}_q$ is no larger than $\frac{c}{1-4c}$, where $\vec{n}_p$ and $\vec{n}_q$ are the unit normals of S at p and q, respectively.*

Lemma 8.4.2. [24] *Let p and q be any two points on a smooth surface S. Then, the acute angle $\angle (p-q), \vec{n}_p$ between the lines supporting the vectors $(p-q)$ and $\vec{n}_p$ is at least $\frac{\pi}{2} - \arcsin(\frac{\|p-q\|}{2d_M(p)})$, where $\vec{n}_p$ is the unit normal of S at p.*

Lemma 8.4.3. *Let $h \in \mathcal{V}(p)$ be a Voronoi facet of the Voronoi neighborhood $\mathcal{V}(p)$ of a point $p \in W$. If the interior of h intersects S tangentially at a point q, then $\|p-q\|$ is at least $k \cdot d_M(p)$.*

Proof. If h intersects S tangentially at a point q, then the unit normal $\vec{n}_q$ of S at q is perpendicular to the supporting plane of h. Let s be the point of W such that $\mathcal{V}(s)$ shares s with h with $\mathcal{V}(q)$. Note that $(p-q)$ and $\vec{n}_q$ are parallel. Aiming at a contradiction, assume that $\|p-q\| < k \cdot d_M(p)$. From Lemma 8.4.1, the acute angle $\angle \vec{n}_p, \vec{n}_q$ between the lines supporting the vectors $\vec{n}_p$ and $\vec{n}_q$ is no larger than $\frac{k}{1-4k}$. On the other hand, from Lemma 8.4.2, the acute angle $\angle (p-s), \vec{n}_p$ between the lines supporting the vectors $(p-s)$ and $\vec{n}_p$ is at least $\frac{\pi}{2} - \arcsin(\frac{\|p-q\|}{2d_M(p)})$. Since $\angle (p-s), \vec{n}_p = \angle \vec{n}_p, \vec{n}_q$, we get that $\angle \vec{n}_p, \vec{n}_q \geq \frac{\pi}{2} - \arcsin(\frac{\|p-q\|}{2d_M(p)})$ and $\angle \vec{n}_p, \vec{n}_q \leq \frac{k}{1-4k}$. By assumption, we have $\|p-q\| < k \cdot d_M(p)$. So, $\angle \vec{n}_p, \vec{n}_q \geq \frac{\pi}{2} - \arcsin(\frac{k}{2})$. But, since $k = 0.06$, we have $\frac{k}{1-4k} = 0.07894$ and $\frac{\pi}{2} - \arcsin(\frac{k}{2}) = 1.54079$, which means that $\angle \vec{n}_p, \vec{n}_q \geq \frac{\pi}{2} - \arcsin(\frac{\|p-q\|}{2d_M(p)})$ and $\angle \vec{n}_p, \vec{n}_q \leq \frac{k}{1-4k}$ is not possible, and thus $\|p-q\|$ must be at least $k \cdot d_M(p)$. □

Although Lemma 8.4.3 tells us what to do when we detect the occurrence of (3), we still have not found a robust and practical way of detecting (3). One may think of trying to check for (3) while looking for cycles in the intersection of S and the facets of $\mathcal{V}(W)$ (see the description of FacetCycle() in Section 8.2). However, the numerical computations for detecting the occurrence of (3) using local piecewise linear approximations for S (which FacetCycle() is based upon) can be very cumbersome in practice. So, the current implementation of our algorithm ignores any occurrence of (3).

8.5 Results and Discussion

We implemented our algorithm using the GNU C++ compiler and the computational geometry and algorithms library, CGAL [45]. CGAL offers a robust implementation of an incremental algorithm for building Delaunay tetrahedrizations and Voronoi diagrams, which allowed us to develop code only for the routines related to our algorithm.

Our implementation takes as input a smooth surface S as defined in Chapter 6, a number ρ_0, with $\rho_0 > (1+k)^2$, which corresponds to the desired upper bound on the radius-edge ratio of each triangle of the output surface mesh, and a threshold g which is a lower bound for the smoothness of any two edge-adjacent triangles of the output surface; that is, if θ is the acute angle defined along the common edge of any two triangles of the output surface, then $\pi - \theta \geq g$. Recall from Section 7.6.4 that g must be larger than $2 \cdot \arcsin(k) + \arcsin(\frac{2k}{\sqrt{3}})$, so that the algorithm is guaranteed to terminate.

We have generated a few surface meshes using our implementation (see Figure 8.3-8.7). Figure 8.3 shows that the algorithm generated small triangles along regions of S with high curvature variation, and large triangles along regions of S with small curvature variation. This is an important feature of the algorithm. However, one

must keep in mind that the size of the triangles are actually related to the value of the function d_M. Since d_M incorporates curvature and *thickness* information, even flat regions of S may be approximated by small triangles. This is the case, for instance, when a small sphere centered at a point of the flat region intersects another region of the surface.

Although the input surface S is a smooth surface, it does retain (to some extent) the "staircase" look of the corresponding digital surface from which S is defined. This is an inherent feature of our construction in Chapter 6, and it causes the local curvature of S to vary abruptly over very small regions. Due to this fact, the surface meshes shown in figures 8.3-8.7 are not as smooth as they could be if another approach had been used to generate S, such as RBF interpolation or level sets method. Nevertheless, we do obtain a smoother piecewise linear approximation, as it can be seen in Figure 8.4. Furthermore, neither RBF interpolation nor level set methods can easily lead us to the simplification of the algorithm in [26] discussed earlier in this chapter.

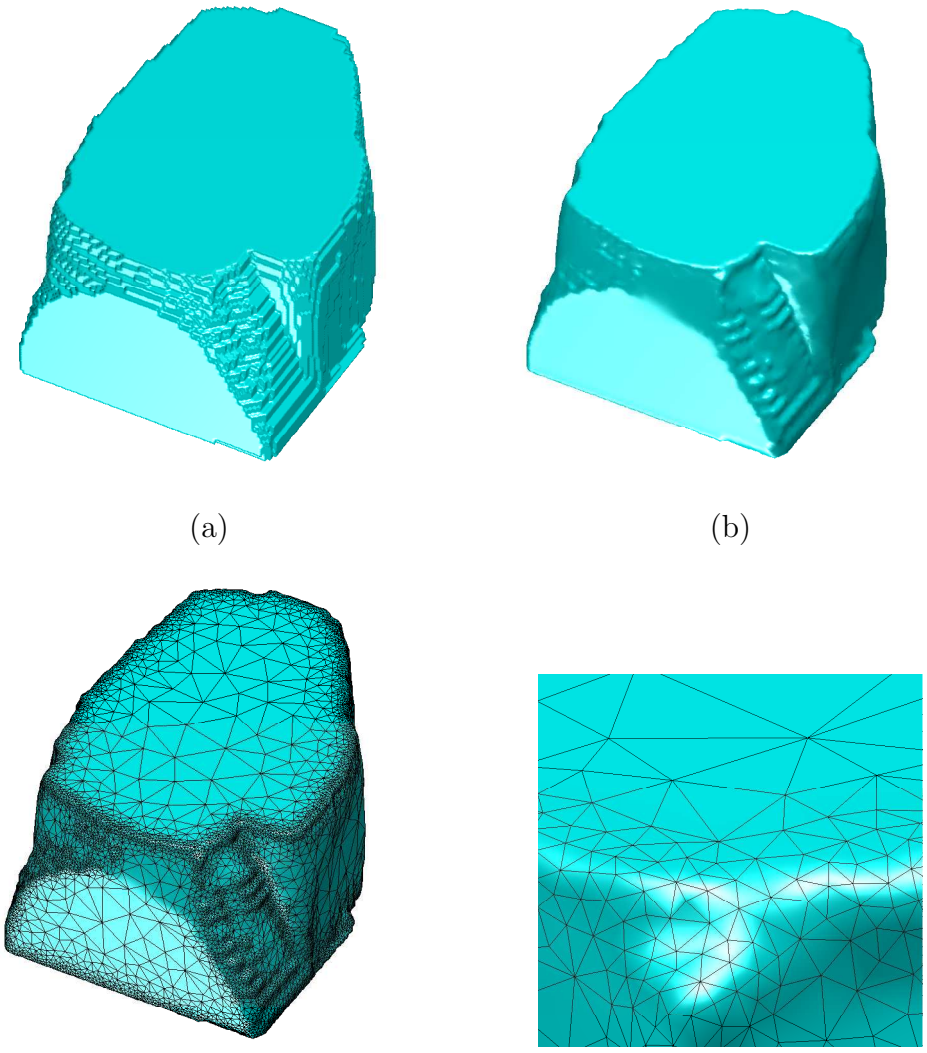
It is worth to mention that we can improve the smoothness of the surface meshes generated by our algorithm as a post-processing step. To do so, we can use one of several heuristic-based smoothing algorithms. For instance, [132]. Most smoothing algorithms improve mesh smoothness by moving its vertices one at a time. So, we can easily include a quality check into such an algorithm that allows a vertex to move only if the aspect ratios of the triangles incident to the vertex do not degenerate. Since our algorithm builds a Voronoi diagram in order to compute the output surface mesh, we can also take advantage of the existence of nice Voronoi-based smoothing algorithms [131].

If we choose ρ_0 to be around $(1+k)^2$ and g to be around $2\cdot\arcsin(k)+\arcsin(\frac{2k}{\sqrt{3}})$, the "staircase" artifact of S may cause our algorithm to perform very poorly. For such values of ρ_0 and g, our algorithm will take much longer to produce the output surface, and the surface will contain much more triangles than the trivial triangulation of the

continuous analog of the boundary between foreground and background of the input image. So, experimentally, we found that letting $\rho_0 = 2$ and $g = \frac{\pi}{3}$ is a good trade off between approximation quality and runtime. If we want our algorithm to run faster for smaller values for ρ_0 and g, we must give to it an input surface without the "staircase" artifact.

Even if we choose not so small values of ρ_0 and g, such as $\rho_0 = 2$ and $g = \frac{\pi}{3}$, we noticed that our algorithm is still very time-consuming when compared with previous and simpler surface meshing algorithms. However, the existing faster algorithms do not provide any guarantee on the quality of the triangles of the output surface. So, we can view this downside as a trade off. Some calculations of our algorithm are also sensitive to numerical problems. This is because they are often based on local triangulations of S (see Section 8.2 and Section 8.3), rather than on the exact representation of S. While an approximate representation speeds up (or makes feasible) the calculations, there is always the risk of producing a wrong result due to a bad approximation.

Recall that the algorithm in [26] does not offer any theoretical guarantee on how close the output surface is from S; that is, we cannot ensure that there is a r-homeomorphism between S and the underlying surface of the output surface mesh. All we can do to improve the geometric approximation is to set both ρ_0 and g to very small values. However, it would be extremely interesting to devise a mechanism to guarantee a r-homeomorphism between the input and output surfaces, for an r provided by the user. This mechanism should also guarantee that the algorithm will always terminate.

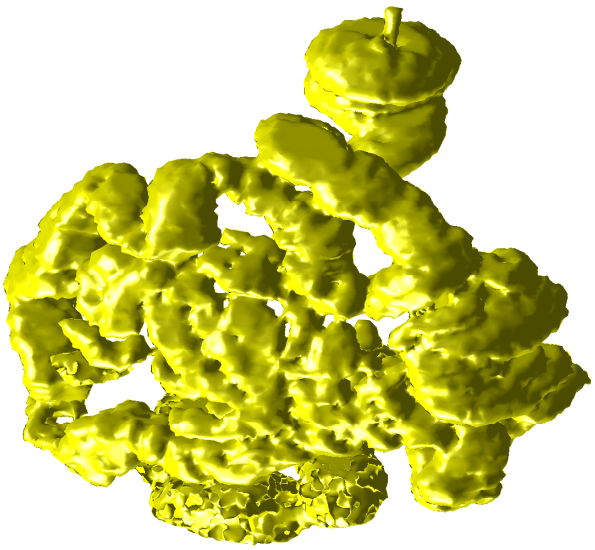


(a) (b)

(c) (d)

Figure 8.3: (a) Continuous analog of the foreground and background of a well-composed image of a "filled" lung. (b) (and (c)) is the simplicial surface produced by our algorithm with $\rho_0 = 2$ and $g = \frac{\pi}{3}$. (d) A close-up view of a region of the mesh in (c).

(a)



(b)

Figure 8.4: (a) Continuous analog of the foreground and background of a well-composed image of a human intestine. (b) Simplicial surface produced by our algorithm with $\rho_0 = 2$ and $g = \frac{\pi}{3}$.

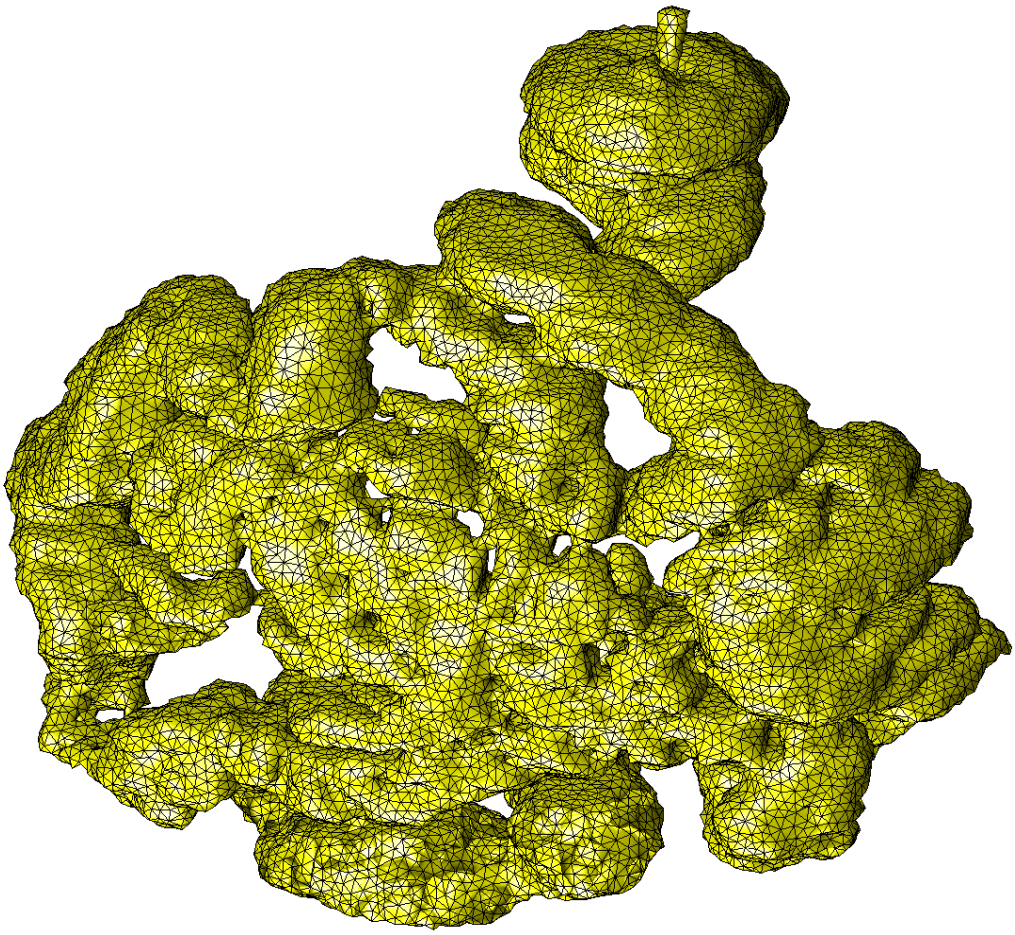


Figure 8.5: Simplicial surface in Figure 8.4(b) with its edges highlighted. This surface mesh was produced in 22 minutes and 7 seconds. It contains 49,640 triangles and 24,597 vertices.

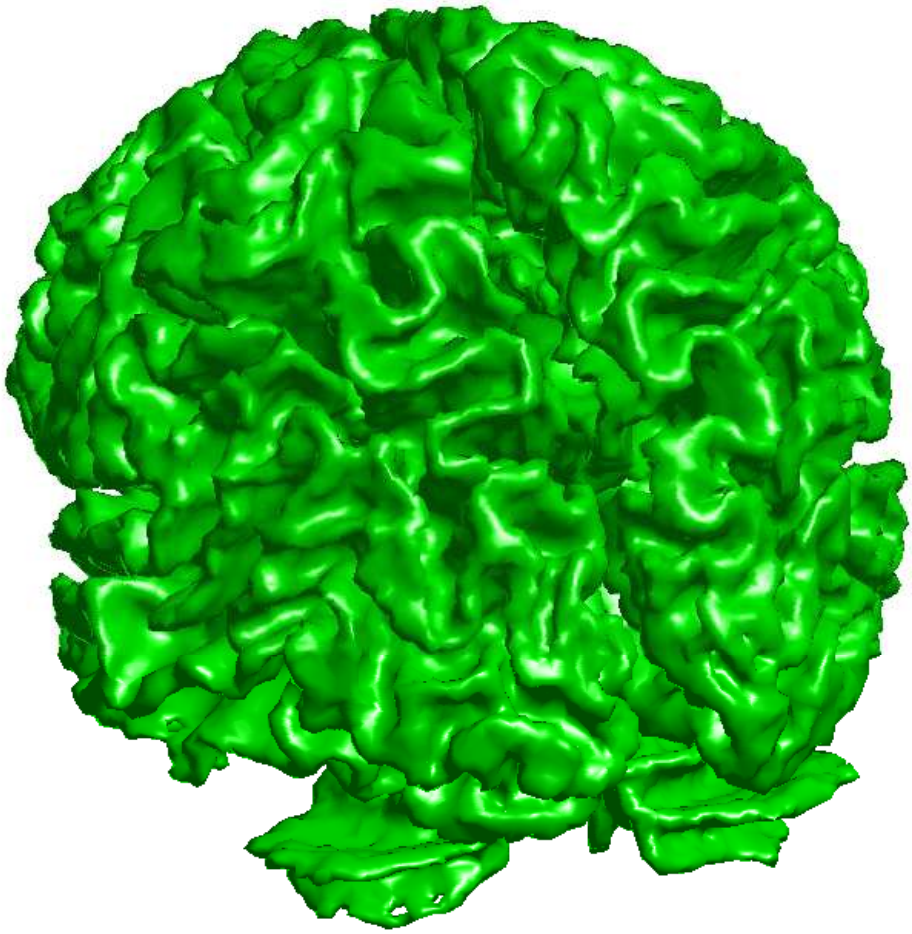


Figure 8.6: Simplicial surface produced by our algorithm from the brain image in Figure 5.8. We used $\rho_0 = 2.5$ and $g = \frac{\pi}{3}$.

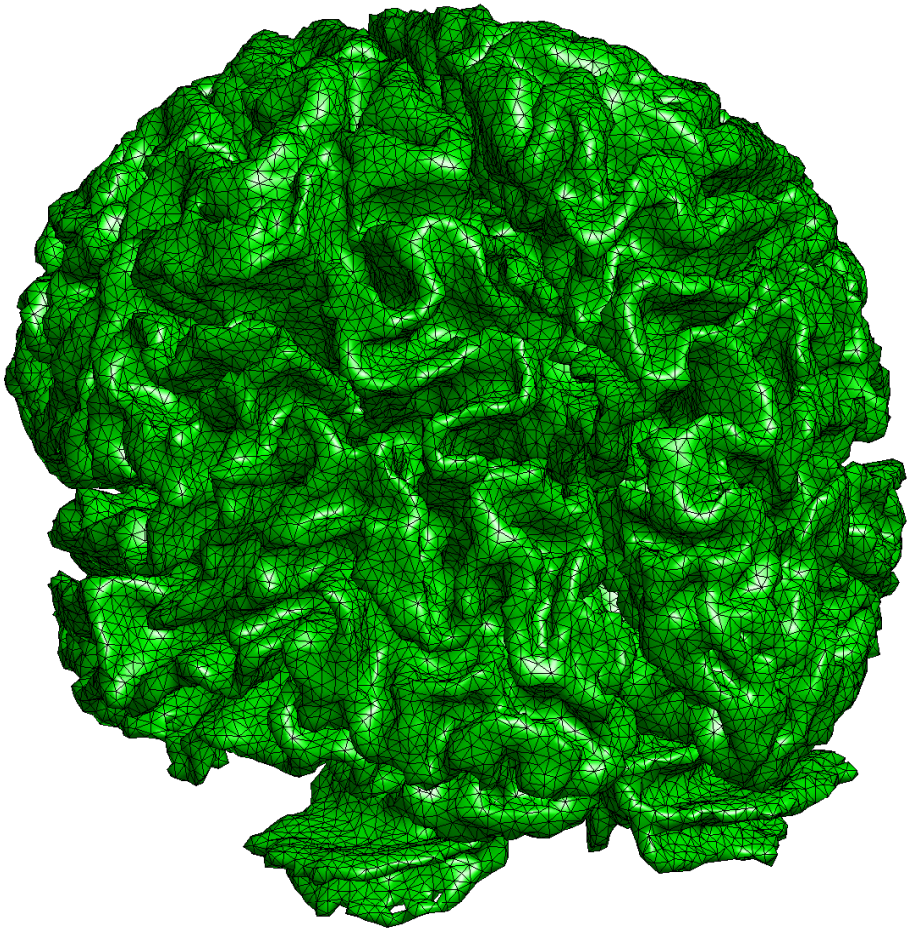


Figure 8.7: Simplicial surface in Figure 8.6(b) with its edges highlighted. This surface mesh was produced in 31 minutes and 29 seconds. It contains 117,182 triangles and 58,474 vertices.

Chapter 9

Conclusion

This thesis provides new solutions for two important problems, namely: the problem of generating *meshes from 2D binary digital images* and the problem of generating *surface meshes from 3D binary digital images*. In both cases, our solutions are characterized by their emphasis in correctness and other theoretical guarantees, which are related to either the size of the mesh or the aspect ratio of mesh elements. Our solutions are also practical and better than previous solutions in several aspects of the problems.

More specifically, for the problem of generating meshes from 2D binary digital images, we provided a new algorithm for converting triangular meshes of polygonal regions, with or without holes, into strictly convex quadrilateral meshes. Our algorithm has the following features:

- the algorithm is linear time in the number of triangles of the input triangular mesh,
- it generates a bounded number of quadrilaterals,
- it inserts a bounded number of Steiner points,
- it offers better bounds than similar algorithms that also produce strictly convex quadrilateral meshes of bounded size,

- it tends to preserve the grading of the input triangular mesh, and
- it is simpler and likely faster than algorithms that produce better quality quadrilateral meshes – in terms of the shape of the quadrilaterals – at the expense of runtime.

We applied our quadrilateral meshing algorithm to the problem of generating meshes from 2D binary digital images. In this problem, the input is a polygonal region possibly with interior vertices and edges. A mesh of the input domain must conform to both the domain boundary and interior vertices and edges. There are several provably good quality algorithms for generating triangular meshes that conform to this kind of domain, which are called constrained triangular meshes. Since our quadrilateral meshing algorithm can handle constrained triangular meshes, we were able to successfully use our algorithm for generating meshes from 2D binary digital images.

For the problem of generating surface meshes from 3D binary digital images, we presented a new solution that consists three steps:

- First, we convert the input image into a well-composed one (if the input image is ill-composed).
- Second, we build an implicit surface that has the same topology as the continuous analog of the digital boundary between the foreground and background of the input image.
- Third, we compute a simplicial approximation to the implicit surface defined in the second step by using a simplified version of an algorithm by Cheng, Dey, Ramos, and Ray [26].

The main advantage of our solution is the fact that the triangles of the output simplicial surface are guaranteed to have a good aspect ratio. This guarantee is a feature of the algorithm in [26]. This algorithm was originally developed to compute

simplicial approximations of implicitly defined surfaces, and it relies on numerically expensive and unstable computations of critical and silhouette points of the input implicit surface. These computations rule out the possibility of using this algorithm to approximate geometrically complex surfaces, such as the ones encountered in biomedical data.

The main contribution of our solution is the simplification of the algorithm in [26], which consists of replacing the aforementioned numerical computations by simpler topological and geometric operations. Recall that *our simplification was made possible by the first two steps of our solution, which allows us to find out all connected components of the implicit surface and to detect the presence of surface handles inside Voronoi regions.*

Although the first and second steps of our solution were meant to simplify the third step, they can be also viewed as independent contributions. More specifically, our algorithm for making 3D binary digital images well-composed can be used as a preprocessing step of algorithms for computing discrete curvature [128] and simplicial approximations [77] directly from the digital data. Likewise, our approach to generate smooth and implicitly defined surfaces from 3D binary digital images can be used for realistic rendering and for computing higher order derivatives in physical simulations.

Our solution for the surface mesh generation problem can in principle be applied to any 3D binary digital image, i.e., it is not restricted to MR images of human organs. However, since we approximate the image boundaries by smooth surfaces, and since the surface meshes are generated from such smooth surfaces, the output meshes do not retain eventual "sharp" edges of the image boundaries. While this is a desirable feature whenever the input image represents an anatomical shape (as the boundary of such shapes are in general smooth surfaces), it may be very undesirable if the input image represents, for instance, a mechanical part whose boundary is not a smooth surface.

9.1 Future Work

We intend to extend the algorithm in [26] to generate tetrahedral meshes from 3D binary digital images. The problem of generating meshes from 3D binary digital images is of great practical interest and has been addressed by several researchers in the past 15 years [49, 68, 48, 135, 18, 64, 130, 46, 89, 54, 99, 134, 35, 98, 145]. However, the theoretical issues in generating meshes of subsets of the Euclidean space are not so well understood as the ones concerning meshes of subsets of the Euclidean plane.

Although there are several algorithms for producing tetrahedral meshes of three-dimensional domains with provable quality guarantees [97, 96, 120, 84, 83, 29, 28, 25, 108, 27], they either provide quality bounds that are not entirely satisfactory in the context of several practical applications or they provide meaningful quality bounds for a restricted class of input domains only. Furthermore, to our best knowledge, there is no known algorithm for generating hexahedral meshes, with provable quality guarantees, of three-dimensional domains. So, the meshing problem in 3D is still a challenge.

By extending the algorithm in [26] to generate tetrahedral meshes from 3D binary digital images, we mean to create tetrahedral meshes of the interior and exterior (with respect to the image domain) of the surface mesh generated by our simplified version of the algorithm in [26]. Note that the algorithm in [26] already generates a tetrahedral mesh of the image domain, which is the dual of the Voronoi diagram of the set of vertices of the surface mesh. This mesh, however, is likely to have very poor quality, i.e., its tetrahedra are likely to be long and skinny. So, the idea is to insert more points in the mesh domain to refine the mesh in order to generate well-shaped tetrahedra.

The strategy mentioned above is known as *Delaunay refinement*, and it has been already used in a similar situation by Oudot, Rineau, and Yvinec [105]. The authors extended the surface meshing algorithm in [15], using the Delaunay refinement

approach, to produce a tetrahedral mesh of the interior of the input surface. Their algorithm offers provable bounds on the quality of the shape of the tetrahedra. These quality bounds are similar to the ones offered by previous algorithms for meshing 3D domains bounded by polyhedra rather than smooth surfaces [96, 120, 29, 25, 108]. Since the algorithm in [15] is very similar to the one in [26], we believe that the work of Oudot, Rineau, and Yvinec [105] can provide us with good insights for extending the algorithm in [26].

9.2 Related Open Problems

In what follows, we provide a small list of *open* problems related to the subject of this thesis:

1. Can we give an algorithm for converting a triangular mesh into a bounded size quadrilateral one such that the shape quality of every quadrilateral is provably guaranteed to be good with respect to some meaningful (in practice) shape metric?

2. Can we give an algorithm for converting a tetrahedral mesh into a hexahedral one? This problem is the three-dimensional version of the problem solved by our algorithm in Chapter 3. In general, the problem of generating good quality hexahedral meshing is a wide open, as not much is known in terms of theoretical properties for generating such meshes [42, 145, 127].

3. Can we extend the algorithm in [26] so that we provide an error bound for the geometric approximation of the input surface?

4. Can we give an algorithm for the surface meshing problem, which enjoys the same features as the one in [26], provides an error bound for the geometric approximation of the input surface, and is also guaranteed to produce a number

of triangles that is a constant factor of the number of triangles of any surface mesh satisfying the other quality requirements?

5. Can we give an algorithm for the surface meshing problem, which enjoys the same features as the one in [26], and is also able to deal with non-smooth surfaces? Such an algorithm would be very useful for approximating objects whose surface contains "creases", pockets, and sharp angles, such as mechanical parts.

Appendix A

Mathematical Preliminaries

This appendix introduces several basic concepts from point set topology, combinatorial topology, and discrete geometry, which are used throughout this thesis. Most of the concepts presented here are can be found in the books by Bloch [13], Ziegler [147], and Gallier [50].

A.1 Topological Surfaces

Let $p \in \mathbb{R}^n$ be a point, and let r be a positive real number. The *n-dimensional open ball* of radius r centered at p is the set $\mathbb{B}_r(p) = \{x \in \mathbb{R}^n \mid \|x - p\| < r\}$. More generally, let $A \subset \mathbb{R}^n$ be any subset, let $p \in A$ be a point, and let r be a positive real number. The *open ball* in A of radius r centered at p is the set $\mathbb{B}_r(p, A)$ defined by $\mathbb{B}_r(p) \cap A = \{x \in A \mid \|x - p\| < r\}$. The *closed ball* in A of radius r centered at p is the set $\overline{\mathbb{B}}_r(p, A) = \{x \in A \mid \|x - p\| \leq r\}$.

A subset $A \subset \mathbb{R}^n$ is an *open* subset of $\mathbb{R}^n$ if, for each point $p \in A$, there is an open ball centered at p that is entirely contained in A, i.e., there exists a positive real number r such that $\mathbb{B}_r(p) \subset A$. Likewise, a subset $S \subset A$ is a *relatively open* subset of A, often referred to simply as an *open* subset of A, if there exists an open set U in $\mathbb{R}^n$ such that $S = A \cap U$. If $p \in A$ is a point, then an *open neighborhood* in

A of p is an open subset of A containing the point p.

A subset $C \subset \mathbb{R}^n$ is *closed* in $\mathbb{R}^n$ if the complement of C, namely $\mathbb{R}^n - C$, is an open subset of $\mathbb{R}^n$. A subset $C \subset A$ is a *relatively closed* subset of A, often referred to as simply a *closed* subset of A, if $A - C$ is open in A. Let $D \subset A \subset \mathbb{R}^n$ be sets. The *closure* of D in A, denoted $\overline{D}$, is defined to be the intersection of all closed subsets of A containing D.

Definition A.1.1. Let $A \subset \mathbb{R}^n$ and $B \subset \mathbb{R}^m$ be sets, and let $f : A \to B$ be a function. The function f is *continuous* if, for every open subset $U \subset B$, the set $f^{-1}(U)$ is open in A, or equivalently, if for every point $p \in A$ and every number $\epsilon > 0$, there is a number $\delta > 0$ such that if $x \in A$ and $\|x - p\| < \delta$ then $\|f(x) - f(p)\| < \epsilon$. The function f is said to be a *homeomorphism* if it is bijective and both it and its inverse are continuous. If f is a homeomorphism, we say that A and B are *homeomorphic*, and we write $A \approx B$.

Definition A.1.2. A function $f : U \subset \mathbb{R}^n \to \mathbb{R}$ is said to be *Lipschitz continuous* if there exists a positive constant L such that

$$|f(p) - f(q)| \leq L \cdot \|p - q\| \tag{A.1}$$

for every $p, q \in U$. The smallest value of L satisfying the inequality in (A.1) is known as the *Lipschitz constant* of f in U. From the Mean Value Theorem, if f is differentiable in U and if, for every $p \in U$, the magnitude $\|\nabla f(p)\|$ of the gradient $\nabla f(p)$ of f at p is bounded by a constant, then f is Lipschitz continuous with Lipschitz constant L bounded from above by $\max_{p \in U} \|\nabla f(p)\|$ on U. If f is Lipschitz continuous with Lipschitz constant L, we say that f is a *L-Lipschitz continuous* function.

Definition A.1.3. Let $A \subset \mathbb{R}^n$ be a set. The *interior* of A, written $int(A)$, is the union of all open balls in $\mathbb{R}^n$ that are subsets of A. It is the largest open subset of A. The *boundary* of A, written $bd(A)$, consists of all points $p \in A$ with the property

that if B is any open set in $\mathbb{R}^n$ and $p \in B$, then $A \cap B \neq \emptyset$ and $(\mathbb{R}^n - A) \cap B \neq \emptyset$. Equivalently, $bd(A) = \bar{A} - int(A)$.

Definition A.1.4. We say that A is *disconnected* if A can be expressed as the union of two non-empty disjoint open subsets. Otherwise, A is said to be *connected*. A subset $C \subset A$ is a *(connected) component* of A if it is non-empty, connected, and not a proper subset of any connected subset of A.

Definition A.1.5. Let $A \subset \mathbb{R}^n$ be a set. A *cover* of A is a (finite or infinite) collection of subsets of A whose union is all of A. If $\mathcal{U} = \{\mathcal{U}_i\}_{i \in I}$ is a cover of A, a *sub-cover* of $\mathcal{U}$ is a sub-collection of the sets in $\mathcal{U}$ that is itself a cover of A (any sub-cover is of the form $\{\mathcal{U}_j\}_{j \in J}$ for a subset $J \subset I$). A *finite cover* is a cover of A with finitely many sets, and a *locally finite cover* is a cover of A such that, for each point $p \in A$ and each open ball $\mathbb{B}_r(p)$, we have that $\mathbb{B}_r(p)$ intersects only finitely many sets of the cover. An *open cover* of A is a cover of A such that all the sets in the cover are open subsets of A.

Definition A.1.6. We say that A is *compact* if every open cover of A has a finite sub-cover. We say that A is *bounded* if there exists some non-negative real number r such that A is contained in the open ball of radius r centered at the origin. A subset of $\mathbb{R}^n$ that is not bounded is called *unbounded*.

The following theorem states an important fact concerning the relationship between compact subsets and bounded and closed subsets of $\mathbb{R}^n$:

Theorem A.1.1 (Heine-Borel Theorem [13]). *A subset $A \subset \mathbb{R}^n$ is compact if, and only if, it is closed in $\mathbb{R}^n$ and bounded.*

Let $\mathbb{D}^2$ and $\overline{\mathbb{D}}^2$ denote the open and closed unit disks in $\mathbb{R}^2$, i.e., $\mathbb{D}^2 = \{x \in \mathbb{R}^2 \mid \|x\| < 1\}$ and $\overline{\mathbb{D}}^2 = \{x \in \mathbb{R}^2 \mid \|x\| \leq 1\}$. The (closed) disk $\overline{\mathbb{D}}^2$ is the union of two disjoint subsets, namely $\mathbb{D}^2$ and $\mathbb{S}^1$, where $\mathbb{S}^1$ is the unit circle in $\mathbb{R}^2$, $\mathbb{S}^1 = \{x \in \mathbb{R}^2 \mid \|x\| = 1\}$. A subset of $\mathbb{R}^n$ that is homeomorphic to the closed interval

$[-1, 1]$ is an *arc*; a subset of $\mathbb{R}^n$ that is homeomorphic to the disk $\overline{\mathbb{D}}^2$ is a *disk*; a subset of $\mathbb{R}^n$ that is homeomorphic to the unit circle is a 1-*sphere* (also known as a *simple closed curve*).

Definition A.1.7. A subset $S \subset \mathbb{R}^n$ is called a *topological surface*, or just *surface* for short, if each point $p \in S$ has an open neighborhood that is homeomorphic to $\mathbb{D}^2$.

Definition A.1.8. Let $U \subset \mathbb{R}^n$ be a set, and let $f : U \to \mathbb{R}^m$ be a function. We say that f is *smooth* if the set U is open in $\mathbb{R}^n$, and all partial derivatives of f of all orders exist and are continuous.

Definition A.1.9. Let $V \subset \mathbb{R}^2$ be an open set. A smooth function $g : V \to \mathbb{R}^3$ is a *coordinate patch* if it is injective and if $\frac{\partial g}{\partial u} \times \frac{\partial g}{\partial v} \neq \vec{0}$ at all points of V.

Definition A.1.10. A subset $S \subset \mathbb{R}^3$ is a *smooth surface* if it is a topological surface and if, for each point $p \in S$, there exists a coordinate patch $g : V \to S$ such that $p \in g(V)$.

Theorem A.1.2 (Classification of Compact Connected Surfaces [13]). *Any compact connected surface in $\mathbb{R}^3$ is homeomorphic to either a sphere or a n-torus (a connected sum of tori).*

Proof. For a detailed proof of the above theorem, we refer the reader to [51], which is available on-line at `http://www.cis.upenn.edu/~jean/gbooks/surftop.html` □

A.2 Simplicial Complexes

Let X be a subset of $\mathbb{R}^n$. We say that X is *convex* if for any two points $q, r \in X$, the point $p = (1 - \lambda)q + \lambda r$, with $0 \leq \lambda \leq 1$ and $\lambda \in \mathbb{R}$, is also a point of X. The empty set is trivially convex, every singleton set is convex, and the entire space $\mathbb{R}^n$ is convex.

Definition A.2.1. Given any nonempty subset X of $\mathbb{R}^n$, there is a smallest set containing X denoted by $conv(X)$ and called the *convex hull of* X. If X is any subset of $m > 0$ points $x_1, \ldots, x_m$ of $\mathbb{R}^n$ then the set of all *convex combinations* $\sum_{i=1}^{m} \lambda_i x_i$, where $\sum_{i=1}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$, is the convex hull $conv(X)$ of the set X.

Definition A.2.2. Given a collection of $m+1$ affinely independent points in $\mathbb{R}^n$, $V = \{v_0, v_1, \ldots, v_m\}$, the *m-simplex (or simplex)* $\sigma = [v_0, v_1, \ldots, v_m]$ *spanned by* V is the convex hull of V, $conv(V)$. That is, the set of all convex combinations $\sum_{i=0}^{m} \lambda_i v_i$, where $\sum_{i=0}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$, with $0 \leq i \leq m$. The *dimension* of σ, denoted by $dim(\sigma)$, is m. In $\mathbb{R}^n$, the largest number of affinely independent points is $n+1$, and we have simplices of dimension $0, 1, \ldots, n$.

Figure A.1 shows examples of the four simplices in $\mathbb{R}^3$. A 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron.

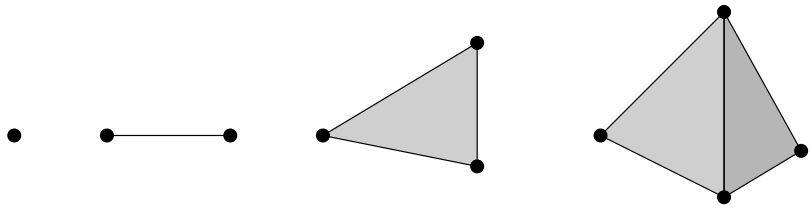


Figure A.1: Examples of simplices of dimension 0, 1, 2, and 3 in $\mathbb{R}^3$.

Note that the convex hull of any nonempty subset $V' \subseteq V$ is again a simplex. The simplex spanned by V', say τ, is called a *face* of σ and the inclusion relation is denoted as $\tau \prec \sigma$. If $dim(\tau) = d$ then τ is called an *d-face* of σ. If d is 0 then τ is called a *vertex*. If d is 1 then τ is called an *edge*. If $m = n$ and $dim(\tau) = n - 1$ then τ is called a *facet* of σ. If $\tau = \sigma$ then τ is an *improper* face, and all others are *proper* faces of σ. The *boundary* $bd(\sigma)$ of a simplex σ is the union of its proper faces. The relative interior of a simplex σ, denoted by $int(\sigma)$, arises by removing its boundary. The number of d-faces of σ is equal to the number of ways we can choose $d+1$ from $m+1$ points, which is $\binom{m+1}{d+1}$. Thus, the total number of faces of σ is given by the

following summation:

$$\sum_{d=0}^{m} \binom{m+1}{d+1} = 2^{m+1} - 1.$$

Definition A.2.3. A *simplicial complex* $\mathcal{K}$ is a finite collection of simplices in $\mathbb{R}^n$ satisfying the following two conditions:

1. If a simplex is in $\mathcal{K}$, then all its faces are in $\mathcal{K}$, i.e., if $\sigma \in \mathcal{K}$ and $\tau \prec \sigma$ then $\tau \in \mathcal{K}$;

2. If $\sigma, \tau \in \mathcal{K}$ are simplices such that $\sigma \cap \tau \neq \emptyset$, then $\sigma \cap \tau$ is a face of each of σ and τ, i.e., $\sigma \cap \tau \prec \sigma$ and $\sigma \cap \tau \prec \tau$.

Figure A.2 shows three sets of simplices in $\mathbb{R}^2$. The set on the left is not a simplicial complex because it is missing an edge and a vertex. The set in the middle contains two simplices that intersect each other but the intersection is not a face of either one, and therefore it cannot be a simplicial complex. The set on the right is a simplicial complex.

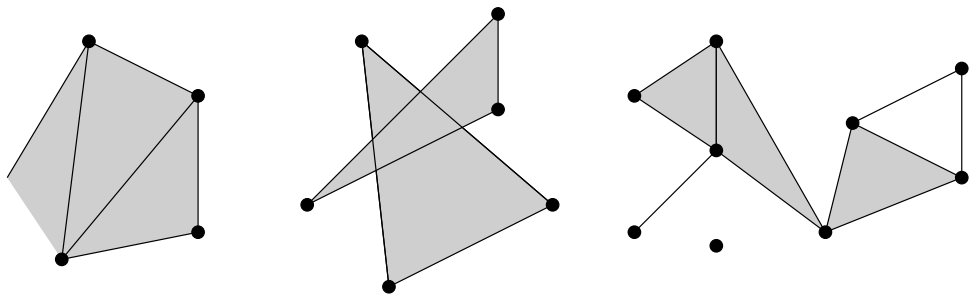


Figure A.2: Collections of simplices in $\mathbb{R}^2$. (a) and (b) are not simplicial complexes, but (c) is.

Definition A.2.4. Let $\mathcal{K}$ be a simplicial complex in $\mathbb{R}^n$. The *dimension* $dim(\mathcal{K})$ of $\mathcal{K}$ is the largest dimension of a face in $\mathcal{K}$, i.e., $dim(\mathcal{K}) = \max\{dim(\sigma) \mid \sigma \in \mathcal{K}\}$. Here, we will refer to an n-dimensional simplicial complex as simply an m-complex. The

set consisting of the union of all points in the simplices of $\mathcal{K}$ is called the *underlying space* of $\mathcal{K}$, or the *polyhedron* of $\mathcal{K}$, or the *geometric realization* of $\mathcal{K}$, and it is denoted by $|\mathcal{K}|$. Since $\mathcal{K}$ is a finite collection of simplices, the underlying space $|\mathcal{K}|$ of $\mathcal{K}$ is a compact set.

A *sub-complex* of a simplicial complex $\mathcal{K}$ is a subset of $\mathcal{K}$ that is itself a simplicial complex. An important example of a sub-complex is the d-skeleton $\mathcal{K}^{\leq d}$ of a simplicial complex $\mathcal{K}$. It consists of all simplices of $\mathcal{K}$ of dimension at most d, i.e., $\mathcal{K}^{\leq d} = \{\sigma \in \mathcal{K} \mid dim(\sigma) \leq d\}$. We define $\mathcal{K}^d = \{\sigma \in \mathcal{K} \mid dim(\sigma) = d\}$ to be the subset of simplices in $\mathcal{K}$ of dimension d. In particular, $\mathcal{K}^0$ is the set of vertices of $\mathcal{K}$. Note that $\mathcal{K}^d$ is not a simplicial complex. We say that a simplicial complex K' *subdivides* $\mathcal{K}$ if $|\mathcal{K}'| = |\mathcal{K}|$ and if every simplex of $\mathcal{K}'$ is a subset (not necessarily proper) of a simplex of $\mathcal{K}$.

Definition A.2.5. Let τ be a simplex in $\mathcal{K}$. The *star* of τ in $\mathcal{K}$, denoted by $st(\tau, \mathcal{K})$, is the set of all simplices that are (not necessarily proper) faces of a simplex in $\mathcal{K}$ that contains τ. That is,

$$st(\tau, \mathcal{K}) = \{\nu \in \mathcal{K} \mid \exists \sigma \in K \text{ such that } \nu \prec \sigma \text{ and } \tau \prec \sigma\}.$$

The *link* of τ in $\mathcal{K}$, denoted by $lk(\tau, \mathcal{K})$, is the set of all faces in $st(\tau)$ that do not intersect τ. That is,

$$lk(\tau, \mathcal{K}) = \{\sigma \in st(\tau, \mathcal{K}) \mid \sigma \cap \tau = \emptyset\}.$$

Let $\mathcal{K}$ be the simplicial complex in Figure A.3(a). The star $st([v], \mathcal{K})$ of $[v]$ consists of the 2-simplices $[r, s, v]$, $[p, s, v]$, $[t, v, x]$, and $[v, x, z]$, and all their 0- and 1-faces, including $[v]$ itself, as illustrated by Figure A.3(b). The link $lk([v], \mathcal{K})$ of $[v]$ consists of the 0-simplices $[r]$, $[s]$, $[p]$, $[x]$, $[z]$, $[t]$, and 1-simplices $[r, s]$, $[s, p]$, $[z, x]$, and $[x, t]$, as illustrated by Figure A.3(c).

Definition A.2.6. A 2-complex $\mathcal{K}$ is called a *simplicial surface* if $\mathcal{K}$ is a 2-complex such that each 1-simplex of $\mathcal{K}$ is the face of precisely two simplices, and the underlying

space of the link of each 0-simplex of $\mathcal{K}$ is homeomorphic to the topological 1-sphere, $\mathbb{S}^1$, i.e., a circle.

For instance, the simplicial complex consisting of all proper faces of a tetrahedron is a simplicial surface. However, the simplicial complex consisting of all proper faces of the two tetrahedra illustrated by Figure A.4 is not a simplicial surface, as the link of $[v]$ is not homeomorphic to $\mathbb{S}^1$.

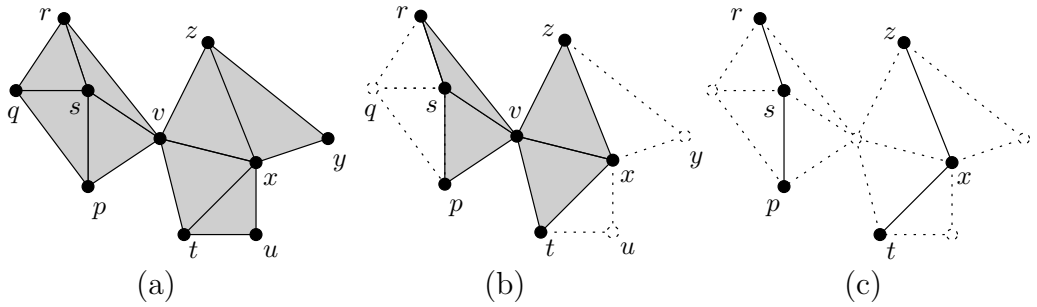


Figure A.3: (a) A simplicial complex. (b) Star of the vertex v in (a). (c) Link of vertex v in (a).

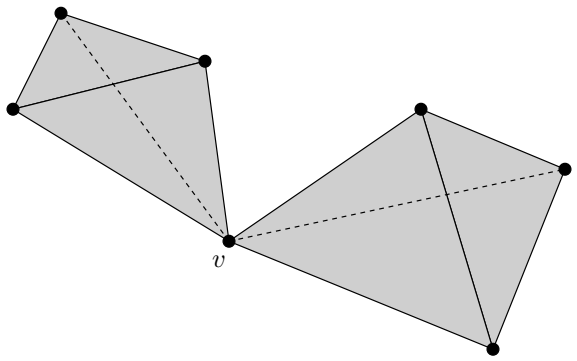


Figure A.4: The 2-complex consisting of the proper faces of the two tetrahedra is not a simplicial surface.

Definition A.2.7. The underlying space $|\mathcal{K}|$ of a simplicial surface $\mathcal{K}$ is called the *underlying surface* of $\mathcal{K}$.

The following lemma from [13] states that the underlying surface of a simplicial surface is a topological surface:

Lemma A.2.1. *Let $\mathcal{K}$ be a simplicial complex in $\mathbb{R}^n$. Then $|\mathcal{K}|$ is a topological surface if and only if $\mathcal{K}$ is a simplicial surface.*

Definition A.2.8. We say that an m-simplicial complex $\mathcal{K}$ is a *pure* or *homogeneous* simplicial complex if and only if any face of $\mathcal{K}$ belongs to the boundary of some m-simplex in $\mathcal{K}$.

Figure A.5(a) shows a simplicial complex that is not pure, and Figure A.5(b) shows a pure simplicial complex.

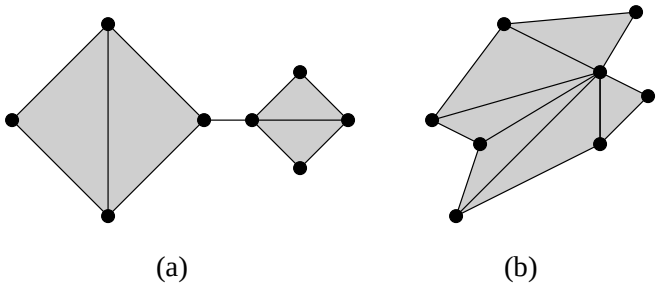


Figure A.5: (a) A simplicial complex that is not pure. (b) A pure simplicial complex.

Definition A.2.9. $\mathcal{K}$ is said to be *connected* if $|\mathcal{K}|$ is connected, and $\mathcal{K}$ is connected if and only if its 1-skeleton is connected.

Both Figure A.5(a) and Figure A.5(b) show connected simplicial complexes, and Figure A.2(c) shows a simplicial complex that is not connected.

Let $\mathcal{K}$ be a pure simplicial complex. The *boundary* of $\mathcal{K}$, $bd(\mathcal{K})$, is the pure simplicial complex of dimension $(m-1)$ consisting of all $(m-1)$-faces of $\mathcal{K}$ that belong to only one m-simplex of $\mathcal{K}$, and all the faces of these $(m-1)$-faces. A face of $\mathcal{K}$ is said to be *external* if it belongs to $bd(\mathcal{K})$ and *internal* otherwise.

Definition A.2.10. Let $\mathcal{K}$ be a simplicial complex, and let σ be any face of $\mathcal{K}$. We say that σ is *singular* if its link, $lk(\sigma)$, is neither homeomorphic to a ball in $\mathbb{R}^d$ nor to the $(d-1)$-sphere, $\mathbb{S}^{d-1} = \{x \in \mathbb{R}^d \mid \|x\| = 1\}$, for some $0 \leq d \leq dim(\mathcal{K})$.

For instance, vertex v in Figure A.3(a) is a singular vertex.

Lemma A.2.2 ([16]). *If $\mathcal{K}$ is a pure, connected simplicial complex of dimension $m \geq 1$, without singularities, then the pure simplicial complex, $bd(\mathcal{K})$, of dimension $m-1$, which is the boundary of $\mathcal{K}$, has itself an empty boundary.*

Proof. If $bd(\mathcal{K})$ is the empty complex or a complex of dimension at most 1, we are done. Then, assume that $bd(\mathcal{K})$ is a pure simplicial complex of dimension at least 2. Since $bd(\mathcal{K})$ is pure, every face in $bd(\mathcal{K})$ must be a face of some $(m-1)$-face in $bd(\mathcal{K})$. So, $bd(\mathcal{K})$ has an an empty boundary if and only if every $(m-2)$-face $bd(\mathcal{K})$ is a face of exactly two $(m-1)$-faces in $bd(\mathcal{K})$. Let σ be a $(m-2)$-face in $bd(\mathcal{K})$, and consider the link of σ, $lk(\sigma)$, in $\mathcal{K}$. Since $\sigma \in bd(\mathcal{K})$, $dim(\sigma) = m-2$ and $\mathcal{K}$ is pure, $lk(\sigma)$ in $\mathcal{K}$ must be a simple polygonal line whose boundary consists of two points, u and v. Because u belongs to a single edge of $lk(\sigma)$ in $\mathcal{K}$, the $(m-1)$-simplex, τ, defined by the convex hull of u and σ belongs to only one m-simplex of $\mathcal{K}$. Thus, $\tau \in bd(\mathcal{K})$. The same argument applies to v, and therefore σ belongs to exactly two $(m-1)$-faces in $bd(\mathcal{K})$. □

A.3 Polytopal Complexes

An $\mathcal{H}$-polyhedron is an intersection of finitely many closed half-spaces in some $\mathbb{R}^n$.

Definition A.3.1. A *polytope*[1] is an $\mathcal{H}$-polyhedron that is *bounded*.

It can be shown that a polytope is the convex hull of a finite set of points in some $\mathbb{R}^n$ [147].

[1]Some authors distinguish between non-convex and convex polytopes. Here, we consider only convex polytopes, and we omit the word "convex".

Definition A.3.2. The *dimension* of a polytope is the dimension of its affine hull, i.e., the dimension of the linear vector space associated with its affine hull. The *affine hull* of a subset $A \subset \mathbb{R}^n$ is the set of all affine combinations formed from all finite subsets of A, i.e., the union of all sets of the form

$$\{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{m} \lambda_i x_i \text{ and } \sum_{i=1}^{m} \lambda_i = 1\}$$

where $\{x_1, \ldots, x_m\}$ is any finite subset of points of A, and $\lambda_i \in \mathbb{R}$ for all $i \in \{1, \ldots, m\}$. Every non-empty affine hull H is an affine subspace, and hence H can be written as $H = p + \vec{V}$, where $p \in H$ is a point and $\vec{V}$ is a linear vector subspace of $\mathbb{R}^n$. The dimension of H, $dim(H)$, is the one of $\vec{V}$.

If $P \subset \mathbb{R}^n$ is a polytope, we denote the affine hull of P by $aff(P)$, and the dimension of P by $dim(P)$, with $dim(P) = dim(aff(P))$.

Definition A.3.3. Let $P \subset \mathbb{R}^n$ be a polytope. A linear inequality $\langle \vec{c}, \vec{x} \rangle \leq c_0$ is *valid* for P if it is satisfied for all points $x = O + \vec{x}$ of P, where O is the origin of $\mathbb{R}^n$, and $\langle \vec{c}, \vec{x} \rangle$ is the inner product of $\vec{c}$ and $\vec{x}$. A *face* F of P is a set of the form $F = P \cap \{x \in \mathbb{R}^n \mid \langle \vec{c}, \vec{x} \rangle = c_0,\ x = O + \vec{x}\}$, where $\langle \vec{c}, \vec{x} \rangle \leq c_0$ is a valid inequality of P. The *dimension* of F, $dim(F)$, is the dimension of its affine hull, $dim(F) = dim(aff(F))$.

Since $\langle \vec{O}, \vec{x} \rangle \leq 0$ is a valid inequality, P itself is a face of P. All other faces of P, satisfying $F \neq P$, are called *proper* faces. The faces of dimensions 0, 1, and $dim(P) - 1$ are called *vertices*, *edges*, and *facets*, respectively. Thus, in particular, the vertices are the minimal nonempty faces, and the facets are the maximal proper faces.

It can be shown that every polytope is the convex hull of its vertices [147]. Also, if a polytope can be written as the convex hull of a finite set of points, then this set contains all vertices of the polytope [147]. Let $P \subset \mathbb{R}^n$ be a polytope, and let F be any face of P. Then, it can also be shown that the following properties hold [147]:

1. The face F is also a polytope.

2. The vertices of F are exactly the vertices of P that belong to F.

3. The faces of F are exactly the faces of P that are contained in F.

4. $F = P \cap aff(F)$.

Definition A.3.4. A *polytopal complex* $\mathcal{C}$ is a finite collection of polytopes in $\mathbb{R}^n$ such that

1. the empty polytope is in $\mathcal{C}$,

2. if $P \in \mathcal{C}$, then all the faces of P are also in $\mathcal{C}$,

3. the intersection $P \cap Q$ of any two polytopes $P, Q \in \mathcal{C}$ is a face of both P and Q.

The dimension $dim(\mathcal{C})$ of $\mathcal{C}$ is the largest dimension of a polytope in $\mathcal{C}$. The *underlying space* of $\mathcal{C}$ is the point set $|\mathcal{C}| = \bigcup_{P \in \mathcal{C}} P$.

If every polytope P of a polytopal complex $\mathcal{C}$ is a simplex then $\mathcal{C}$ is a simplicial complex. Furthermore, notions such as *connectedness*, *pureness*, *singularity*, and *k-skeleton* are defined just as in the simplicial case (see Section A.2).

A.4 Final Remarks

Throughout most chapters of this thesis, we deal with compact surfaces, smooth surfaces, and *piecewise linear* surfaces, which are the underlying spaces of pure 2-dimensional polytopal complexes with empty boundary and without singularities. We often refer to homeomorphisms between any two of these types of surfaces in $\mathbb{R}^3$. Although we did not prove that these homeomorphisms always exist, in $\mathbb{R}^3$, it is true that every surface is homeomorphic to a piecewise linear surface, and that every piecewise linear surface is homeomorphic to a smooth surface. So, the class of

all surfaces, all smooth surfaces, and all piecewise linear surfaces are essentially the same in $\mathbb{R}^3$ [13].

We remark that our definition of (topological) surface is the one of a surface *without boundary* [92]. A (topological) surface *with boundary* is a subset $S \subset \mathbb{R}^n$ such that each point $p \in S$ has an open neighborhood that is homeomorphic to either $\mathbb{D}^2$ or to $\mathbb{H}^2 = (\mathbb{D}^2 \cap \{(x, y) \in \mathbb{R}^2 \mid x \geq 0\})$. Unless stated otherwise, we will use the term "surface" to mean term "surface without boundary", and whenever we need to refer to a "surface with boundary", we will do so by explicitly mentioning the qualification "with boundary".

Appendix B

Meshing Small Polygonal Regions

This appendix introduces several lemmas concerning strictly convex quadrilateral meshes of small polygonal regions, i.e., polygonal regions consisting of 4, 5, 6, or 7 boundary edges and no holes. These lemmas are used by our algorithm for converting triangular meshes into strictly convex quadrilateral meshes of bounded size (see Chapter 3).

B.1 Quadrilaterals and Hexagons

We start with two useful facts about strictly convex quadrilaterals meshes of 4- and 6-sided polygons, which are given and proved in Bremner, Hurtado, Ramaswami and Sacristán [17]:

Lemma B.1.1. *A hexagon can be decomposed into at most four strictly convex quadrilaterals by using at most three Steiner points in its interior.*

Lemma B.1.2. *A quadrilateral with a point in its interior can be decomposed into at most five strictly convex quadrilaterals by using at most three Steiner points in its interior.*

B.2 Pentagons

For polygonal regions bounded by an odd number of edges, one of the boundary edges is designated as an *outgoing edge*. (In the algorithm described in Section 3.2.1 the outgoing edge is simply the triangular mesh edge between the root of subtree T_v and its parent node.) When quadrangulating this region, all Steiner points except one are placed in the interior of the polygon, and one Steiner point may be placed on the outgoing edge. In the following lemmas, these relevant facts are stated and proved formally:

Definition B.2.1. *Given two points p and q, we denote by $L(p,q)$ (resp. $R(p,q)$) the left (resp. right) open half-space defined by the oriented line from p to q. Given a vertex v of a polygon P, we define wedge(v) as follows: If v is reflex then wedge(v) denotes the locus of points inside P that can be connected to v forming strictly convex angles at v. If v is convex then wedge(v) is the interior of the visibility region of v in P. Now, assume that P is a simple polygon. Given a triangular mesh $\mathcal{T}$ of P such that the vertices of $\mathcal{T}$ are the ones of P, a triangle $\Delta = (p,q,r)$ of $\mathcal{T}$ is said to be an* ear *if and only if p, q, and r are consecutive vertices of a counterclockwise (or clockwise) enumeration of the vertices of P.*

Definition B.2.2. *Let P be a pentagon and let e be an edge of P. Given a triangular mesh $\mathcal{T}$ of P such that the vertices of $\mathcal{T}$ are the ones of P, the mesh $\mathcal{T}$ necessarily consists of three triangles, two of which are ears (see Figure B.1). Each of these ears shares two vertices and a distinct diagonal of $\mathcal{T}$ with the third triangle, called the center triangle. Furthermore, the edge e is said to be of type 1 with respect to $\mathcal{T}$ if it is the edge of P shared with the center triangle of $\mathcal{T}$. If e is not of type 1 and e is adjacent to the type 1 edge, it is said to be of type 2 with respect to $\mathcal{T}$. If e is neither of type 1 nor of type 2, then e is incident to the common vertex of all triangles of $\mathcal{T}$ and is said to be of type 3.*

Lemma B.2.1. *Let P be a pentagon and let e be the outgoing edge of P. Then, given any triangular mesh $\mathcal{T}$ of P such that the vertices of $\mathcal{T}$ are the ones of P, we have the following:*

1. *If e is of type 1 with respect to $\mathcal{T}$, P can be decomposed into two strictly convex quadrilaterals and one triangle adjacent to e by adding one Steiner point inside P.*

2. *If e is of type 2 with respect to $\mathcal{T}$ then, P can be decomposed into three strictly convex quadrilaterals and one triangle adjacent to e by adding two Steiner points inside P.*

3. *If e is of type 3 with respect to $\mathcal{T}$ then, P can be decomposed into four strictly convex quadrilaterals by adding two Steiner points inside P and one more on the edge e.*

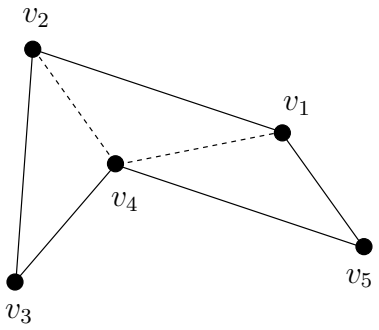


Figure B.1: Triangular mesh $\mathcal{T}$ of a pentagon P.

Proof. Let v_1, v_2, v_3, v_4, v_5 be a counterclockwise enumeration of the vertices of P. Let $\mathcal{T}$ be a triangular mesh of P such that the vertices of $\mathcal{T}$ are the ones of P and refer to Figure B.2. Without loss of generality, assume that the center triangle, T_c, of $\mathcal{T}$ is defined by the vertices v_4, v_1 and v_2, and v_4 is the common vertex of all three triangles of $\mathcal{T}$. If e is of type 1 then e is the edge $\overline{v_1v_2}$ of P. Let

$R_1 = wedge(v_3) \cap wedge(v_5) \cap T_c$. Note that the interior of R_1 cannot be empty, and clearly $R_1 \subset P$. Denote the barycenter of R_1 by p_1. By removing edges $\overline{v_4v_1}$ and $\overline{v_4v_2}$ from $\mathcal{T}$, and then adding the point p_1 and the edges $\overline{v_1p_1}$, $\overline{p_1v_4}$, and $\overline{p_1v_2}$, we obtain a decomposition of P into two quadrilaterals, $Q_1 = (v_1, p_1, v_4, v_5)$ and $Q_2 = (v_4, p_1, v_2, v_3)$, and one triangle, $T_1 = (v_1, v_2, p_1)$, which contains $e = \overline{v_1v_2}$. Since p_1 belongs to both $wedge(v_3) \cap T_c$ and $wedge(v_5) \cap T_c$, Q_1 and Q_2 are strictly convex quadrilaterals, and hence Claim 1 is true. If e is of type 2 then e is either $\overline{v_2v_3}$ or $\overline{v_5v_1}$. Assume that $e = \overline{v_2v_3}$ and refer to Figure B.3. By removing diagonals $\overline{v_4v_1}$ and $\overline{v_4v_2}$ from $\mathcal{T}$, and then adding the point p_1 above and the edges $\overline{v_1p_1}$ and $\overline{p_1v_4}$, we obtain a decomposition of P into the quadrilateral $Q_1 = (v_1, p_1, v_4, v_5)$ and the pentagon $P' = (v_2, v_3, v_4, p_1, v_1)$. Since diagonals $\overline{p_1v_2}$ and $\overline{v_3p_1}$ are inside P', the triangle (p_1, v_2, v_3) is entirely contained in P', and we can decompose P' into triangles $T'_c = (p_1, v_2, v_3)$, (p_1, v_1, v_2), and (v_3, v_4, p_1), where T'_c is the center triangle and (p_1, v_1, v_2), and (v_3, v_4, p_1) are ears of the triangular mesh (see Figure B.3(a)). Let R_2 be the region $R_2 = wedge(v_1) \cap wedge(v_4) \cap T'_c$. Note that $R_2 \subset P' \subset P$, and the interior of R_2 cannot be empty. Define p_2 to be the barycenter of R_2. We can now decompose P' as in (1) to obtain two strictly convex quads, $Q_3 = (v_2, p_2, p_1, v_1)$ and $Q_4 = (v_3, v_4, p_1, p_2)$, and a triangle $T_2 = (v_3, p_2, v_2)$ adjacent to e. If $e = \overline{v_5v_1}$ we have the symmetric case, and hence Claim 2 holds.

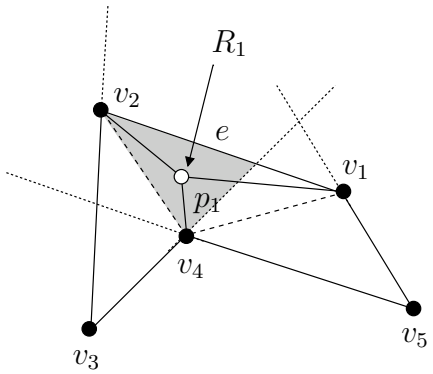


Figure B.2: Illustration of Claim 1 of Lemma B.2.1.

If e is of type 3, then e is either $\overline{v_3v_4}$ or $\overline{v_4v_5}$. Assume that $e = \overline{v_3v_4}$. By removing diagonals $\overline{v_4v_1}$ and $\overline{v_4v_2}$ from $\mathcal{T}$, and then adding the points p_1 and p_2 above plus the edges $\overline{v_1p_1}$, $\overline{p_1v_4}$, $\overline{p_1p_2}$ and $\overline{p_2v_2}$, we obtain a decomposition of P into strictly convex quads $Q_1 = (v_1, p_1, v_4, v_5)$ and $Q_3 = (v_2, p_2, p_1, v_1)$, and the pentagon $P'' = (v_2, v_3, v_4, p_1, p_2)$. Let $l = \overrightarrow{v_2p_2}$ be the oriented line from v_2 to p_2. Note that v_3 is on the right side of l and the line l. If v_4 is also on the right of l, we define p_3 to be the midpoint of the edge $\overline{v_3v_4}$. Otherwise, the line l intersects the edge $\overline{v_3v_4}$, and we define p_3 to be the midpoint of the segment defined by v_3 and the intersection point of l and $\overline{v_3v_4}$ (see Figure B.3(b)). By adding p_3 and the edge $\overline{p_2p_3}$ to P'', we obtain a decomposition of P'' into two quadrilaterals, $Q_5 = (v_2, v_3, p_3, p_2)$ and $Q_6 = (v_4, p_1, p_2, p_3)$. Since p_3 lies on the right side of l, both Q_5 and Q_6 are strictly convex quadrilaterals and therefore the set consisting of the convex quadrilaterals Q_1, Q_3, Q_5 and Q_6 is a decomposition of P that uses three Steiner points, two of which are inside P and the third is on the edge $e = \overline{v_3v_4}$. If $e = \overline{v_4v_5}$ we have the symmetric case, and hence Claim 3 also holds. □

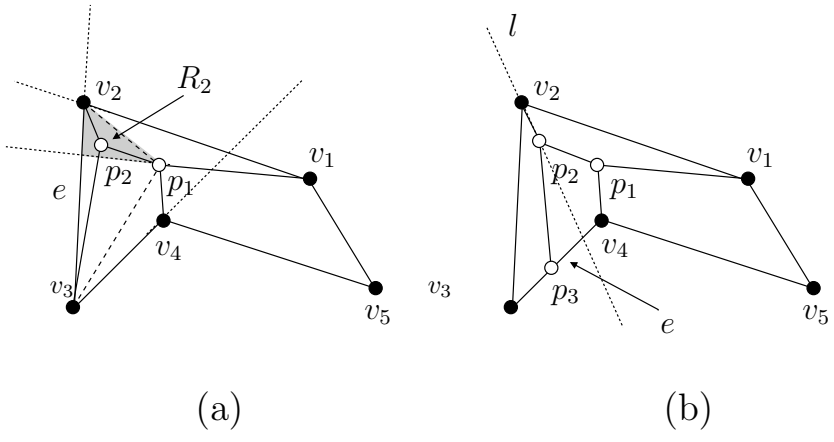


Figure B.3: (a) Illustration of Claim 2, and (b) Claim 3 of Lemma B.2.1.

In steps 3(c) and 4(c)(v) of our algorithm in Chapter 3, the subgraph G_v corresponds to a triangular mesh $\mathcal{T}_v$ of a pentagon P such that $\mathcal{T}_v$ has exactly one vertex q

inside P (see Figure 3.13). The following lemma is invoked by steps 3(c) and 4(c)(v) to convert T_v into a partial quadrilateral mesh:

Lemma B.2.2. *Let P be a pentagon, q a point in its interior, and e the outgoing edge of P. Then, P can be decomposed into at most six strictly convex quadrilaterals and one triangle adjacent to e by inserting at most four Steiner points inside P.*

Proof. Let $\mathcal{T}$ be any triangular mesh of P such that the vertices of $\mathcal{T}$ are the ones of P. Let T_1, T_2, and T_3 be the three triangles of $\mathcal{T}$, where T_2 is the center triangle and T_1 and T_3 are the ears. There are two possibilities: The point q belongs to the triangle containing e, or it does not. In the former case, let Δ be the triangle obtained by connecting q to the two endpoints of e. P is thus decomposed into Δ, which is adjacent to the outgoing edge, and a hexagon H (see Figure B.4(a)). From Lemma B.1.1, H can be decomposed into at most four strictly convex quadrilaterals using at most three Steiner points in its interior. Suppose now that q does not belong to the triangle containing e. If e belongs to T_2, decompose P into a triangle Δ adjacent to e and two strictly convex quadrilaterals, as in case (1) of Lemma B.2.1 (see Figure B.4(b)). One of these quadrilaterals must contain q, which can be decomposed into five convex quads using three more Steiner points by applying Lemma B.1.2. Thus, P is decomposed into six strictly convex quadrilaterals and a triangle adjacent to e by using four Steiner points. If e belongs to T_1, let $\Delta = T_1$. Triangles T_2 and T_3 form a quadrilateral with a point inside (see Figure B.4(c)), to which we apply Lemma B.1.2. Thus, P is decomposed into five strictly convex quadrilaterals and a triangle adjacent to e by using three Steiner points. The case when e belongs to T_3 is symmetric. □

B.3 Heptagons

In our algorithm in Chapter 3, a polygonal region S bounded by seven edges (heptagon) is obtained when the subtree T_v is a path of five nodes, with the middle node

as the root of T_v. (This is the only case that results in a heptagon.) Hence, in this case, the outgoing edge is always the edge of S belonging to the middle node. The triangular mesh $\mathcal{T}_v$ of S also has a center triangle, the triangle corresponding to the middle node of T_v. This triangle contains only one edge of S, the outgoing one. Figure B.5 illustrates a triangular mesh of a heptagon whose dual graph is a path of five nodes.

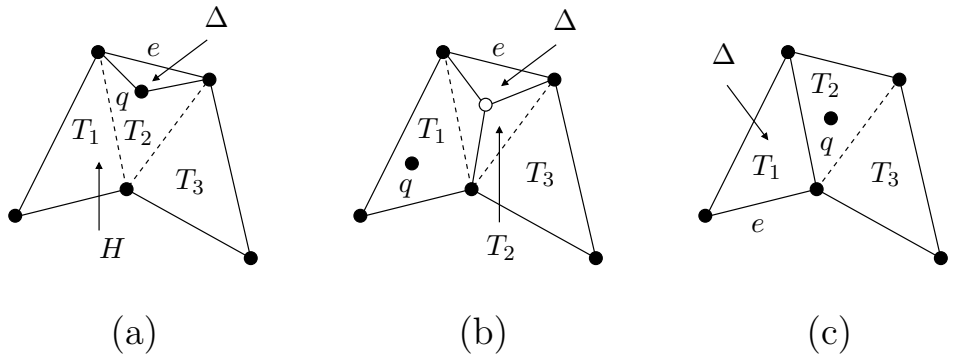


Figure B.4: Illustrations of the possible cases (up to symmetry) of Lemma B.2.2.

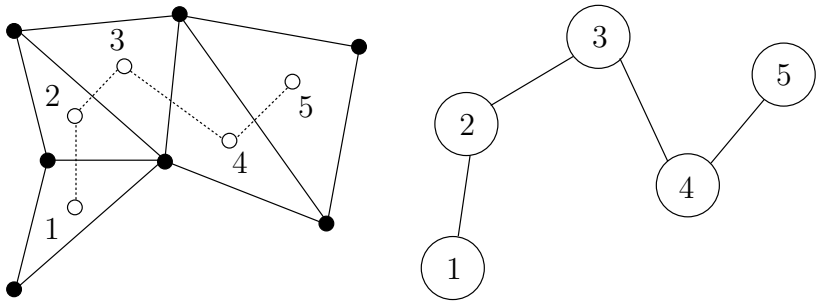


Figure B.5: Triangular mesh of a heptagon and its dual graph.

Lemma B.3.1. *Let S be a heptagon such that S admits a triangular mesh $\mathcal{T}$ such that the vertices of $\mathcal{T}$ are the ones of S, and whose dual graph is a path. Let the edge of S contained in the center triangle of $\mathcal{T}$ be the outgoing edge e. Then, S can be decomposed into six strictly convex quadrilaterals and one triangle adjacent to e by adding four Steiner points inside S.*

Proof. Let S be a heptagon such that S admits a triangular mesh $\mathcal{T}$ such that the vertices of $\mathcal{T}$ are the ones of S, and whose dual graph is a path. Let $v_1, v_2, \ldots, v_7$ be a counterclockwise enumeration of the vertices of S. Without loss of generality, assume that the center triangle T_c of $\mathcal{T}$ is $T_c = (v_1, v_2, v_5)$. So, there are at most four possibilities for the other four triangles of $\mathcal{T}$, T_1, T_2, T_3 and T_4 (see Figure B.6): (a) $T_1 = (v_5, v_3, v_4)$, $T_2 = (v_5, v_2, v_3)$, $T_3 = (v_5, v_7, v_1)$, and $T_4 = (v_5, v_6, v_7)$; (b) $T_1 = (v_5, v_3, v_4)$, $T_2 = (v_5, v_2, v_3)$, $T_3 = (v_5, v_6, v_1)$, and $T_4 = (v_6, v_7, v_1)$; (c) $T_1 = (v_4, v_2, v_3)$, $T_2 = (v_5, v_2, v_4)$, $T_3 = (v_5, v_7, v_1)$, and $T_4 = (v_5, v_6, v_7)$; and (d) $T_1 = (v_4, v_2, v_3)$, $T_2 = (v_5, v_2, v_4)$, $T_3 = (v_5, v_6, v_1)$, and $T_4 = (v_6, v_7, v_1)$. Assume that $\mathcal{T}$ consists of T_c and the four triangles shown in Figure B.6(a). Below, we describe how to place four Steiner points, p_1, p_2, p_3, and p_4, in order to decompose S into six convex quads and one triangle adjacent to e. The proofs for the other possibilities are very similar and we omit them here.

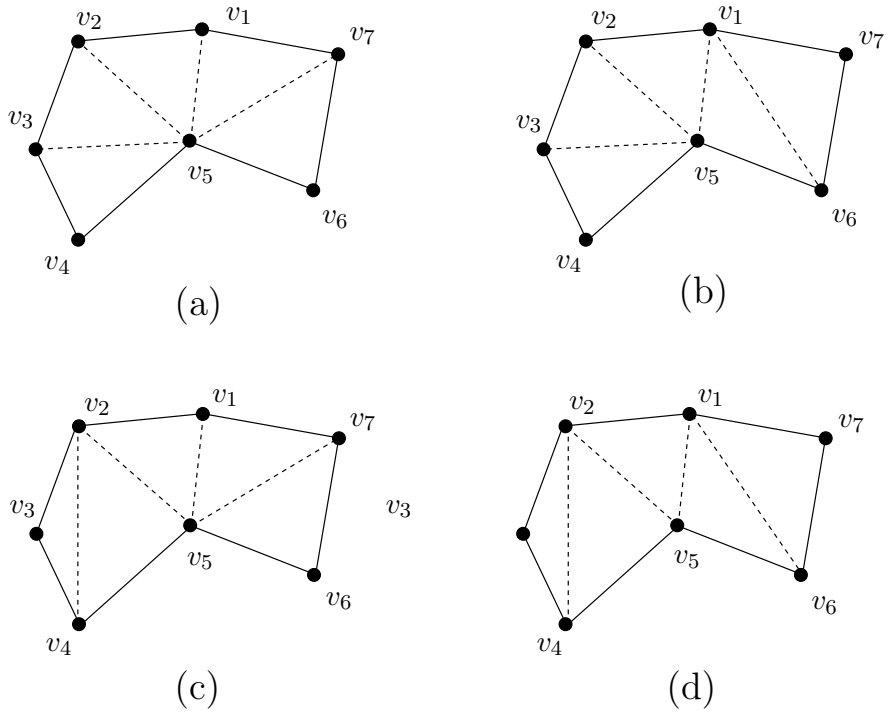


Figure B.6: All triangular meshes of a heptagon whose dual graphs are paths.

Let $R_1 = wedge(v_4) \cap wedge(v_1) \cap T_2$. Clearly, $R_1 \subseteq T_2$, $v_5 \in R_1$, and the interior of R_1 cannot be empty. Similarly, let R_2 be the non-empty region $R_2 = wedge(v_6) \cap wedge(v_2) \cap T_3$. Now, consider the two following cases (see Figure B.7): (i) $v_2 \in R(v_4, v_5)$ and (ii) $v_2 \notin R(v_4, v_5)$. Recall that $R(p, q)$ (resp. $L(p, q)$) is the right (resp. left) open half-space defined by the oriented line from p to q (see Definition B.2.1). In case (i), because $\overline{v_5v_2}$, $\overline{v_5v_1}$, and $\overline{v_5v_7}$ are edges of $\mathcal{T}$ and $v_2 \notin R(v_4, v_5)$, the intersection region $R_1 \cap R(v_7, v_5)$ cannot be empty. So, we choose p_1 and p_2 to be the barycenters of $R_1 \cap R(v_7, v_5)$ and R_2, respectively. In case (ii), we choose p_1 to be the barycenter of R_1, and we calculate the position of p_2 as follows: If $v_1 \in R(v_6, v_5)$ then p_2 is the barycenter of $R_2 \cap L(p_1, v_5)$. The region $R_2 \cap L(p_1, v_5)$ cannot be empty. Otherwise, the vertex v_6 would belong to $L(p_1, v_5)$ and the vertex v_7 would belong to the complement of $L(p_1, v_5)$, which is an absurd as $v_6 \in R(v_4, v_5)$ and $v_4 \in R(p_1, v_5)$.

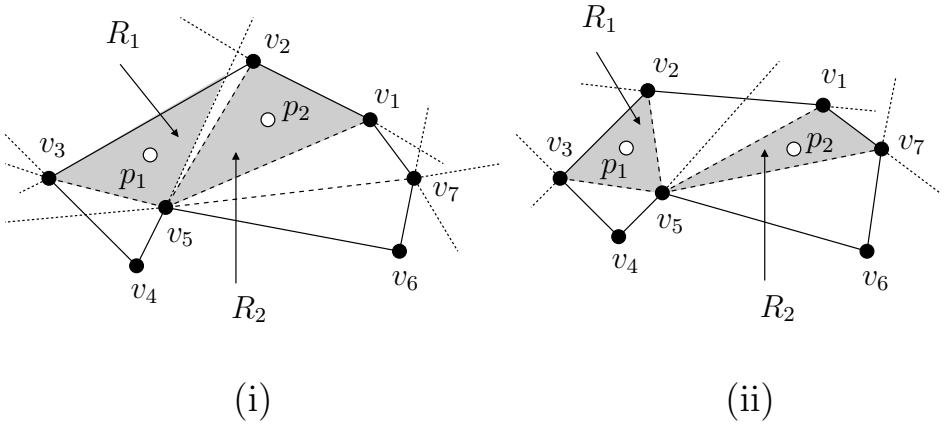


Figure B.7: Choosing points p_1 and p_2 when (i) $v_2 \in R(v_4, v_5)$ and (ii) $v_2 \notin R(v_4, v_5)$.

If $v_1 \notin R(v_6, v_5)$ then p_2 is the barycenter of $R_2 \cap L(v_3, v_5)$. Again, $R_2 \cap L(v_3, v_5)$ cannot be empty as the oriented line $\overrightarrow{v_3v_5}$ from v_3 to v_5 passes through the interior of $wedge(v_2) \cap wedge(v_6)$ and v_1 is on the left of $\overrightarrow{v_3v_5}$. In both cases (i) and (ii), we have that $p_2 \in L(p_1, v_5)$. Since $p_1 \in T_2$ and $p_2 \in T_3$, we have that the oriented lines $l_1 = \overrightarrow{v_3p_1}$ and $l_2 = \overrightarrow{v_7p_2}$ intersect the edges $\overline{v_5v_2}$ of T_2 and $\overline{v_5v_1}$ of T_3, respectively.

Besides, since $p_1 \in wedge(v_1)$ and $p_2 \in wedge(v_2)$, the oriented lines $l_3 = \overrightarrow{p_1v_1}$ and $l_4 = \overrightarrow{p_2v_2}$ intersect the edges $\overline{v_5v_2}$ of T_2 and $\overline{v_5v_1}$ of T_3, respectively. If q and r denote the intersection points of l_1 and l_3 with $\overline{v_5v_2}$, respectively, then v_2, q and r are collinear, and we define p_3 to be the midpoint of $\overline{v_2x}$, where $x = q$ if q is in between v_2 and r, and $x = r$ otherwise. Similarly, If s and t denote the intersection points of l_2 and l_4 with $\overline{v_5v_1}$, respectively, then v_1, s and t are collinear, and we choose p_4 to be the midpoint of $\overline{v_1y}$, where $y = s$ if s is in between v_1 and t, and $y = t$ otherwise. In this way, we have that $p_3 \in L(p_1, v_1)$ and $p_4 \in R(p_1, v_1)$. Now, by removing edges $\overline{v_3v_5}$, $\overline{v_2v_5}$, $\overline{v_1v_5}$, and $\overline{v_7v_5}$ from $\mathcal{T}$, and then adding points p_1, p_2, p_3 and p_4 and edges $\overline{v_5p_1}$, $\overline{p_1v_3}$, $\overline{p_1p_3}$, $\overline{p_1v_3}$, $\overline{v_5p_2}$, $\overline{p_2v_7}$, $\overline{p_2p_4}$, $\overline{p_1p_4}$, and $\overline{p_3v_1}$, we obtain a decomposition of S into six quadrilaterals, $Q_1 = (v_4, v_5, p_1, v_3)$, $Q_2 = (v_3, p_1, p_3, v_2)$, $Q_3 = (p_1, v_5, p_2, p_4)$, $Q_4 = (p_1, p_4, v_1, p_3)$, $Q_5 = (v_5, v_6, v_7, p_2)$, and $Q_6 = (v_7, v_1, p_4, p_2)$, and one triangle, $T = (p_3, v_1, v_2)$, as shown in Figure B.8.

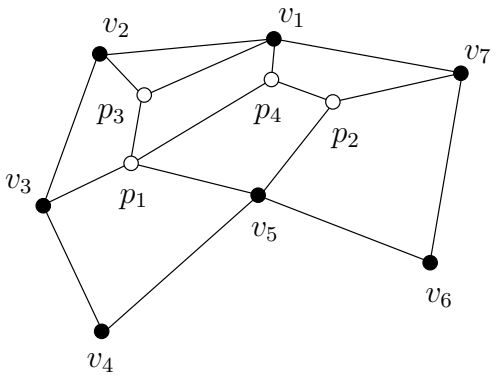


Figure B.8: Resulting decomposition from the triangular mesh in Figure B.6(a).

Since p_1 and p_2 belong to the interior of $wedge(v_4) \cap T_2$) and $wedge(v_6) \cap T_3$, respectively, we have that Q_1 and Q_5 are strictly convex quadrilaterals. Since $p_3 \in L(v_3, p_1)$ and $p_4 \in R(v_7, p_2)$, we get that Q_2 and Q_6 are also strictly convex quadrilaterals. Since $p_2 \in L(p_1, v_5)$, $p_2 \in R(v_5, v_1)$, $p_1 \in L(v_5, v_1)$, $p_4 \in L(p_1, p_2)$, and p_4 is a point on $\overline{v_5v_1}$, we get that Q_3 is a strictly convex quadrilateral. Finally, because $p_3 \in L(p_1, v_1)$, $p_4 \in R(p_1, v_1)$, p_3 is a point on $\overline{v_5v_2}$, and p_4 is a point on $\overline{v_5v_1}$, we get that Q_4 is also

a strictly convex quadrilateral. Thus, our claim holds when T_1, T_2, T_3 and T_4 are the triangles corresponding to possibility (a). $\square$

Bibliography

[1] M.J. Ackerman. The Visible Human Project. *Proceedings of the IEEE*, 86(3):504–511, 1998.

[2] L. V. Ahlfors and L. Sario. *Riemann Surfaces*. Number 2 in Princeton Mathematics Series. Princeton University Press, 1960.

[3] D.J. Allman. A Quadrilateral Finite Element Including Vertex Rotations for Plane Elasticity Analysis. *International Journal for Numerical Methods in Engineering*, 26:717–730, 1988.

[4] J. Amanatides and A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Proceedings of the Eurographics Conference*, pages 3–10, 1987.

[5] N. Amenta and M. Bern. Surface Reconstruction by Voronoi Filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[6] N. Amenta, S. Choi, T. Dey, and N. Leekha. A Simple Algorithm for Homeomorphic Surface Reconstruction. *International Journal of Computational Geometry and Applications*, 12(1-2):125–141, 2002.

[7] N. Amenta, S. Choi, and R. K. Kolluri. The Power Crust. In *Proceedings of the 6th ACM Symposium on Solid Modeling and Applications*, pages 249–260, Ann Arbor, Michigan, USA, June 4-8 2001.

[8] M. Bern and D. Eppstein. Mesh generation and Optimal Triangulation. In F.K. Hwang and D.-Z. Du, editors, *Computing in Euclidean Geometry*. World Scientific, 1992.

[9] M. Bern and D. Eppstein. Quadrilateral Meshing by Circle Packing. In *Proceedings of the 6th International Meshing Roundtable*, pages 7–19, Park City, Utah, USA, October 13-15 1997.

[10] M. Bern and P. Plassmann. Mesh Generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science, 2000.

[11] S. Bischoff and L. Kobbelt. Sub-Voxel Topology Control for Level-Set Surfaces. In *Proceedings of the Eurographics 2003*, pages 1008–1018, Granada, Spain, September 1-6 2003.

[12] T. Blacker. Paving: A New Approach To Automated Quadrilateral Mesh Generation. *International Journal For Numerical Methods in Engineering*, 32(4):811–847, 1991.

[13] E.D. Bloch. *A First Course in Geometric Topology and Differential Geometry*. Birkhäuser, 1997.

[14] J.-D. Boissonnat and F. Cazals. Smooth Surface Reconstruction via Natural Neighbour Interpolation of Distance Functions. In *Proceedings of the the 16th Annual ACM Symposium on Computational Geometry*, pages 223–232, New York, NY, USA, June 12-14 2000.

[15] J-.D. Boissonnat and S. Oudot. Provably Good Surface Sampling and Approximation. In *Proceedings of the 1st Eurographics Symposium on Geometry Processing*, pages 9–18, Aachen, Germany, June 23-25 2003.

[16] J-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, Cambridge, UK, 1998.

[17] D. Bremner, F. Hurtado, S. Ramaswami, and V. Sacristán. Small Convex Quadrangulations of Point Sets. In *Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, volume 2223 of *Lecture Notes in Computer Science*, pages 623–635. Spriger-Verlag, 2001.

[18] D.L.A. Camacho, R.H. Hopper, G.M. Lin, and B.S. Myers. An Improved Method for Finite Element Mesh Generation of Geometrically Complex Structures with Application to the Skullbase. *Journal of Biomechanics*, 30(10):1067–1070, 1997.

[19] S. Canann, S. Muthukrishnan, and R. Phillips. Topological Improvement Procedures for Quadrilateral Finite Element Meshes. *Engineering with Computers*, 14:168–177, 1998.

[20] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, and B.C. McCallum. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *SIGGRAPH 01: Proceedings of the 28th ACM Annual Conference on Computer Graphics and Interactive Techniques*, pages 67–76, August 12-17 2001.

[21] E. Catmull and J. Clark. Recursively Generated B-Spline Surfaces on Arbitrary Topological Surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.

[22] F. Cazals and M. Pouget. Smooth Surfaces, Umbilics, Lines of Curvatures, Foliations, Ridges and the Medial Axis: a Concise Overview. Technical Report 5138, INRIA, Sophia Antipolis, France, 2004.

[23] M. Chen, A. Kaufman, and R. Yagel, editors. *Volume Graphics*. Spriger-Verlag, 2000.

[24] H-.L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic Skin Triangulation. *Discrete & Computational Geometry*, 25(4):525–568, 2001.

[25] S.-W. Cheng, T. Dey, E. Ramos, and T. Ray. Quality Meshing for Polyhedra with Small Angles. In *Proceedings of the the 20th Annual ACM Symposium on Computational Geometry*, pages 290–299, Brooklyn, New York, USA, June 9-11 2004.

[26] S.-W. Cheng, T. Dey, E. Ramos, and T. Ray. Sampling and Meshing a Surface with Guaranteed Topology and Geometry. In *Proceedings of the the 20th Annual ACM Symposium on Computational Geometry*, pages 280–289, Brooklyn, New York, USA, June 9-11 2004.

[27] S.-W. Cheng, T. Dey, and T. Ray. Weighted Delaunay Refinement for Polyhedra with Small Angles. In *Proceedings of the 14th International Meshing Roundtable*, pages 325–342, San Diego, California, USA, September 11-14 2005.

[28] S.-W. Cheng and T.K. Dey. Quality Meshing with Weighted Delaunay Refinement. *SIAM Journal on Computing*, 33(1):69–93, 2003.

[29] S-W. Cheng and S.-H. Poon. Graded Conforming Delaunay Tetrahedralization with Bounded Radius-Edge Ratio. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 295–304, Baltimore, Maryland, USA, January 12-14 2003.

[30] E. Chernyaev. Marching Cubes 33: Construction of Topologically Correct Isosurfaces. Technical Report CN/95-17, CERN, 1995.

[31] L. P. Chew. Guaranteed-Quality Mesh Generation for Curved Surfaces. In *Proceedings of the 9th Annual ACM Symposium on Computational Geometry*, pages 274–280, San Diego, California, USA, May 18-21 1993.

[32] L.P. Chew. Guaranteed-Quality Triangular Meshes. Technical Report 89-983, Department of Computer and Information Science, Cornell University, Ithaca, NY, USA, 1989.

[33] C. S. Co, S. D. Porumbescu, and K. I. Joy. Meshless Isosurface Generation from Multiblock Data. In *Proceedings of the Joint Eurographics-IEEE TCVG Symposium on Visualization*, pages 273–282, Konstanz, Germany, May 19-21 2004.

[34] D. Collins, A. Zijdenbos, V. Kollokian, J. Sled, N. Kabani, C. Holmes, and A. Evans. Design and Construction of a Realistic Digital Brain Phantom. *IEEE Transactions on Medical Imaging*, 17(3):463–468, 1998.

[35] J. Crouch, S. Pizer, E. Chaney, and M. Zaider. Medial Techniques to Automate Finite Element Analysis of Prostate Deformation. *IEEE Transactions on Medical Imaging (to appear)*, 2005.

[36] T. Dey, G. Li, and T. Ray. Polygonal Surface Remeshing with Delaunay Refinement. In *Proceedings of the 14th International Meshing Roundtable*, pages 343–361, San Diego, California, USA, September 11-14 2005.

[37] T. K. Dey and W. Zhao. Approximating the Medial Axis from the Voronoi Diagram with a Convergence Guarantee. *Algorithmica*, 38(1):179–200, 2003.

[38] Tamal K. Dey and Samrat Goswami. Provable Surface Reconstruction from Noisy Samples. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, pages 330–339, June 9-11 2004.

[39] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, Cambridge, UK, 2001.

[40] H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[41] H. Edeslbrunner and N. Shah. Triangulating Topological Spaces. *International Journal of Computational Geometry & Applications*, 7(4):365–378, 1997.

[42] D. Eppstein. Topological Issues in Hexahedral Meshing. Talk at the Conference on Algebraic Topology Methods in Computer Science, 2001.

[43] J. Erickson. Nice Point Sets Can Have Nasty Delaunay Triangulations. In *Proceedings of the 17th Annual ACM Symposium on Computational Geometry*, pages 96–105, Medford, Massachusets, USA, June 3-5 2001.

[44] H. Everett, W. Lenhart, M. Overmars, T. Shermer, and J. Urrutia. Strictly Convex Quadrilateralizations of Polygons. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 77–82, St. John's, Newfoundland, Canada, 1992.

[45] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the Design of CGAL, the Computational Geometry Algorithms Library. *Software – Practice and Experience*, 30:1167–1202, 2000.

[46] M. Ferrant, A. Nabavi, B. Macq, F. Jolesz, R. Kikinis, and S. Warfield. Registration of 3-D Intraoperative MR Images of the Brain Using a Finite-Element Biomechanical Model. *IEEE Transactions on Medical Imaging*, 20(12):1384–1396, 2001.

[47] P. Frey and P.-L. George. *Mesh Generation: Application to Finite Elements*. Hermes Science Publishing, Oxford, UK, 2000.

[48] P. Frey, B. Sarter, and M. Gautherie. Fully Automatic Mesh Generation for 3-D Domains Based Upon Voxel Sets. *International Journal for Numerical Methods in Engineering*, 37:2735–2753, 1994.

[49] D.P. Fyhrie, M.S. Hamid, R.F. Kuo, and S.M. Lang. Direct Three-Dimensional Finite Element Analysis of Human Vertebral Cancelous Bone. In *Transactions of the 38th Annual Meeting of Orthopaedic Research Society*, volume 17, page 551, February 17-20 1992.

[50] J. Gallier. *Geometric Methods and Applications: For Computer Science and Engineering* . Springer-Verlag, 2000.

[51] J. Gallier. The Classification Theorem for Compact Surfaces and a Detour on Fractals. Preprint, 2005.

[52] M. Gavriliu, J. Carranza, D. Breen, and A. Barr. Fast Extraction of Adaptive Multiresolution Meshes with Guaranteed Properties from Volumetric Data. In *Proceedings of the 12th Annual IEEE Conference on Visualization*, pages 295–302, San Diego, California, USA, October 21-26 2001.

[53] J. Gee and R. Bajcsy. Elastic Matching: Continuum Mechanical and Probabilistic Analysis. In Arthur Toga, editor, *Brain Warping*. Academic Press, 1999.

[54] A.P. Gibson, J. Riley, M. Schweiger, J.C. Hebden, S.R. Arridge, and D.T. Delpy. A Method for Generating Patient-Specific Finite Element Meshes for Head Modelling. *Physics in Medicine and Biology*, 48(4):481–495, 2003.

[55] S. Gibson. Constrained Elastic Surface Nets: Generating Smooth Surfaces from Binary Segmented Data. In *First International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 888–898, Cambridge, MA, USA, October 11-13 1998.

[56] M. Golubitsky and V. Guillemin. *Stable Mappings and Their Singularities*. Springer-Verlag, 1973.

[57] R. Gonzalez and Richard Woods. *Digital Image Processing*. Prentice Hall, 2002.

[58] D. Gordon and J. K. Udupa. Fast Surface Tracking in Three-Dimensional Binary Images. *Computer Vision, Graphics, and Image Processing*, 45(2):196–214, 1989.

[59] C.M. Grimm and J.F. Hughes. Modeling Surfaces of Arbitrary Topology Using Manifolds. In *SIGGRAPH 95: Proceedings of the 22nd ACM Annual Conference on Computer Graphics and Interactive Techniques*, pages 359–368, August 6-11 1995.

[60] Eitan Grinspun and Peter Schröder. Normal bounds for subdivision-surface interference detection. In *Proceedings of the 12th Annual IEEE Conference on Visualization*, pages 333–340, San Diego, California, USA, October 21-26 2001.

[61] A. Gross and L. Latecki. Digitizations Preserving Topological and Differential Geometric Properties. *Computer Vision and Image Understanding*, 62(3):370–381, 1995.

[62] V. Guillemin and A. Pollack. *Differential Topology*. Prentice-Hall, 1974.

[63] X. Han, C. Xu, and J.L. Prince. A Topology Preserving Level Set Method for Generating Deformable Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):755–768, 2003.

[64] U. Hartmann and F. Kruggel. A Fast Algorithm for Generating Large Tetrahedral 3D Finite Element Meshes from Magnetic Resonance Tomograms . In *Proceedings of the IEEE Workshop on Biomedical Image Analysis*, pages 184–192, Santa Barbara, California, USA, June 26-27 1998.

[65] G. Herman. *Geometry of Digital Spaces*. Springer-Verlag, 1998.

[66] G. Herman and B. Carvalho. Multiseeded segmentation using fuzzy connectedness. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(5):460–474, 2001.

[67] J. Hershberger and J. Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In *Proceedings of the 5th International Symposium*

on Spatial Data Handling, volume 1, pages 134–143, Charleston, South Carolina, USA, 1992.

[68] S.J. Hollister, J.M. Brennan, and N. Kikuchi. Direct Analysis of Trabecular Bone Stiffness and Tissue Level Mechanics Using an Element-by-Element Homogeinization Method. In *Transactions of the 38th Annual Meeting of Orthopaedic Research Society*, volume 17, page 559, February 17-20 1992.

[69] C.R. Jacobs, J.A. Mandell, and G.S. Beaupre. A Comparative Study of Automatic Finite Element Mesh Generation Techniques in Orthopaedic Biomechanics. In *Proceedings of the ASME Bioengineering Conference*, volume BED-24, pages 512–514, 1993.

[70] C.R. Jacobs, J.A. Mandell, and G.S. Beaupre. Finite Element Solution Errors Associated with Digital Image-Based Mesh Generation. In *Proceedings of the ASME Bioengineering Conference*, volume BED-28, pages 147–148, 1994.

[71] B. Joe. Quadrilateral mesh generation in polygonal regions. *Computer-Aided Design*, 27(3):209–222, 1995.

[72] B.P. Johnston, J.M. Sullivan, and A. Kwasnik. Automatic Conversion of Triangular Finite Meshes to Quadrilateral Meshes. *International Journal of Numerical Methods in Engineering*, 31(1):67–84, 1991.

[73] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. *ACM Transactions on Graphics*, 21(3):339–346, 2002.

[74] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. *SIAM Reviews*, 45(3):385–482, 2003.

[75] R. Kolluri. Provably Good Moving Least Squares. In *Proceedings of the 2005 ACM-SIAM Symposium on Discrete Algorithms*, pages 1008–1018, Vancouver, British Columbia, Canada, January 23-25 2005.

[76] V.A. Kovalesky. Finite Topology as Applied to Image Analysis. *Computer Vision, Graphics, and Image Processing*, 46(2):141–161, 1989.

[77] N. Krahnstoever and C. Lorenz. Computing Curvature-Adaptive Surface Triangulations of Three-Dimensional Image Data. *The Visual Computer*, 20(1):17–36, 2004.

[78] Ming-Jun Lai. Scattered Data Interpolation and Approximation Using Bivariate C^1 Piecewise Cubic Polynomials. *Computer Aided Geometric Design*, 13(1):81–88, 1996.

[79] L. Latecki. 3D Well-Composed Pictures. *Graphical Models and Image Processing*, 59(3):164–172, 1997.

[80] L. Latecki. *Discrete Representation of Spatial Objects in Computer Vision.* Kluwer Academic Publishers, 1998.

[81] L. Latecki, U. Eckhardt, and A. Rosenfeld. Well-Composed Sets. *Computer Vision and Image Understanding*, 61(1):70–83, 1995.

[82] J. M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *GTM*. Springer-Verlag, 2002.

[83] X-Y Li. Spacing Control and Sliver-Free Delaunay Mesh. In *Proceedings of the 9th International Meshing Roundtable*, pages 295–306, New Orleans, Louisiana, USA, October 2-5 2000.

[84] X.-Y. Li, S.-H. Teng, and A. Üngör. Biting Spheres in 3D. In *Proceedings of the 8th International Meshing Roundtable*, pages 295–306, South Lake Tahoe, California, USA, October 10-13 2000.

[85] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

[86] A. Lopes and K. Brodlie. Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16–27, 2003.

[87] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *SIGGRAPH 87: Proceedings of the 14th ACM Annual Conference on Computer Graphics and Interactive Techniques*, volume 21, pages 163–169, July 27-31 1987.

[88] A. Lubiw. Decomposing Polygonal Regions into Convex Quadrilaterals. In *Proceedings of the 1st ACM Symposium on Computational Geometry*, pages 97–106, Baltimore, Maryland, USA, June 5-7 1985.

[89] F.H. Mabin, P. Brengard, R. David, and C. Mongenet. 3D Structured Mesh Generator for Volumetric Digital Data. In *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, Las Vegas, Nevada, USA, June 24-27 2002.

[90] A. Malanthara and W. Gerstle. Comparative Study of Unstructured Meshes Made of Triangles and Quadrilaterals. In *Proceedings of the 6th International Meshing Roundtable*, pages 437–447, Park City, Utah, USA, October 13-15 1997.

[91] J. Marchadier, D. Arques, and S. Michelin. Thinning Grayscale Well-Composed Images. *Pattern Recognition Letters*, 25(5):581–590, 2004.

[92] W.S. Massey. *Algebraic Topology: An Introduction*, volume 56 of *GTM*. Springer-Verlag, 1967.

[93] C. Maurer, R. Qi, and V. Raghavan. A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, 2003.

[94] T. McInerey and D. Terzopoulos. Deformable Models in Medical Image Analysis: A Survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[95] G. Miller, S. Pave, and N. Walkington. When and Why Ruppert's Algorithm Works. In *Proceedings of the 12th International Meshing Roundtable*, pages 91–102, Santa Fe, New Mexico, USA, September 14-17 2003.

[96] G. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computation*, pages 683–692, Las Vegas, Nevada, USA, May 29 - June 01 1995.

[97] S. Mitchell and S. Vavasis. Quality Mesh Generation in Three Dimensions. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, pages 212–221, Berlin, Germany, June 10-12 1992.

[98] A. Mohamed and C. Davatzikos. Finite Element Mesh Generation and Remeshing from Segmented Medical Images. In *Proceedings of the 2004 IEEE International Symposium on Biomedical Imaging*, Arlington, Virginia, USA, April 15-18 2004.

[99] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A Crystalline, Red Green Strategy for Meshing Deformable Objects with Tetrahedra. In *Proceedings of the 12th International Meshing Roundtable*, pages 103–114, Santa Fe, New Mexico, USA, September 14-17 2003.

[100] B. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating Implicit Surfaces from Scattered Surface Data Using Compactly

Supported Radial Basis Functions. In *Proceedings of IEEE International Conference on Shape Modeling and Applications*, pages 89–98, Genova, Italy, May 7-11 2001.

[101] Matthias Müller-Hannemann and Karsten Weihe. Minimum Strictly Convex Quadrangulations of Convex Polygons. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 193–202, Centre Universitaire Méditerranéen, Nice, France, June 4-6 1997.

[102] B. Natarajan. On Generating Topologically Consistent Isosurfaces from Uniform Samples. *The Visual Computer*, 11(1):52–62, 1994.

[103] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H-.P. Seidel. Multi-Level Partition of Unity Implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003.

[104] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons Ltd, 2nd edition, 2000.

[105] S. Oudot, L. Rineau, and M. Yvinec. Meshing Volumes Bounded by Smooth Surfaces. In *Proceedings of the 14th International Meshing Roundtable*, pages 203–220, San Diego, California, USA, September 11-14 2005.

[106] S. Owen, M. Staten, S. Cannan, and S. Saigal. Q-Morph: An Indirect Approach to Advancing Front Quad Meshing. *International Journal for Numerical Methods in Engineering*, 9(44):1317–1340, 1999.

[107] S.J. Owen. A Survey of Unstructured Mesh Generation Technology. In *Proceedings of the 7th International Meshing Roundtable*, pages 239–267, Dearborn, Michigan, USA, October 26-28 1998.

[108] S. Pav and N. Walkington. Robust Three Dimensional Delaunay Refinement. In *Proceedings of the 13th Internation Meshing Roundtable*, pages 145–156, Williamsburg, VA, USA, September 19-22 2004.

[109] P. Pébay. Planar Quadrangle Quality Measures: Is There Really a Choice? In *Proceedings of the 11th International Meshing Roundtable*, pages 53–62, Ithaca, New York, USA, September 2002.

[110] J. Peters. C^2 Free-Form Surfaces of Degree $(3, 5)$. *Computer Aided Geometric Design*, 19(2):113–126, 2002.

[111] J. Peters. Geometric Continuity. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. Elsevier Science, 2002.

[112] S. Ramaswami, P. Ramos, and G. Toussaint. Converting Triangulations to Quadrangulations. *Computational Geometry: Theory and Applications*, 9:257–276, 1998.

[113] S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, and J. Gee. A New Algorithm for Generating Quadrilateral Meshes and Its Application to FE-Based Image Registration. In *Proceedings of the 12th International Meshing Roundtable*, pages 159–170, Santa Fe, New Mexico, USA, September 14-17 2003.

[114] S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, and J. Gee. Constrained Quadrilateral Meshes of Bounded Size. *International Journal of Computational Geometry & Applications*, 15(1):55–98, 2005.

[115] A. Rosenfeld, T.Y. Kong, and A. Nakamura. Topology-Preserving Deformations of Two-Valued Digital Pictures. *Graphical Models and Image Processing*, 60(1):24–34, 1998.

[116] J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[117] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-Based Decimation of Marching Cubes Surfaces. In *Proceedings of the 7th Annual IEEE Conference on Visualization*, pages 335–342, San Francisco, California, USA, October 27 - November 1 1996.

[118] C. Shen, J.F. O'Brien, and J.R. Shewchuk. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics*, 23(3):896–904, 2004.

[119] J. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133, Philadelphia, PA, USA, May 1996.

[120] J. Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 86–95, Minneapolis, Minnesota, USA, June 7-10 1998.

[121] J. Shewchuk. Delaunay Refinement Algorithms for Triangular Mesh Generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, 2002.

[122] J. Shewchuk. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Proceeding of the 11th International Meshing Roundtable*, pages 115–126, Ithaca, New York, USA, September 15-18 2002.

[123] K. Shimada, J-H. Liao, and T. Itoh. Quadrilateral Meshing with Directionality Control through the Packing of Square Cells. In *Proceedings of the 7th International Meshing Roundtable*, pages 61–76, Dearborn, Michigan, USA, October 26-28 1998.

[124] M. Siqueira, L.J. Latecki, and J. Gallier. Making 3D Binary Digital Images Well-Composed. In *Proceedings of the IS&T/SPIE Conference on Vision Geometry XIII*, volume 5675, pages 150–163, San Jose, California, USA, January 16-20 2005.

[125] M. Siqueira, T. Sundaram, S. Ramaswami, J. Gallier, and J. Gee. Quadrilateral Meshes for the Registration of Human Brain Images.

[126] J. Staples, P. Eades, N. Katoh, and A. Moffat, editors. *No Quadrangulation is Extremely Odd*, volume 1004 of *Lecture Notes in Computer Science*, Cairns, Australia, December 4-6 1995.

[127] M. Staten, S. Owen, and T. Blacker. Unconstrained Paving and Plastering: A New Idea for All-Hexahedral Meshing. In *Proceedings of the 14th International Meshing Roundtable*, pages 399–416, San Diego, California, USA, September 11-14 2005.

[128] E. M. Stokely and S. Y. Wu. Surface Parametrization and Curvature Measurement of Arbitrary 3-D Objects: Five Practical Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(8):833–840, 1992.

[129] N. Sukumar. Meshless Methods and Partition of Unity Finite Elements. In *Proceedings of the 6th International ESAFORM Conference on Material Forming*, pages 603–606, Salerno, Italy, April 28-30 2003.

[130] J.M. Sullivan, Z. Wu, and A. Kulkarni. 3D Volume Mesh Generation of Human Organs Using Surface Geometries Created from the Visible Human Data Set. In *Proceedings of the 3rd Visible Human Project Conference*, NIH, Bethesda, Maryland, USA, October 5-6 2000.

[131] Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. Isotropic Remeshing of Surfaces: a Local Parameterization Approach. In *Proceedings of 12th International Meshing Roundtable*, pages 215–224, Santa Fe, New Mexico, USA, September 14-17 2003.

[132] Gabriel Taubin. Curve and Surface Smoothing without Shrinkage. In *Proceedings of the 5th International Conference on Computer Vision (ICCV)*, pages 852–857, Boston, Massachusetts, USA, June 20-23 1995.

[133] S.-H. Teng and C. Wong. Unstructured Mesh Generation: Theory, Practice, and Perspectives. *International Journal of Computational Geometry and Applications*, 10(3):227–266, 2000.

[134] T. Udeshi. Tetrahedral Mesh Generation from Segmented Voxel Data. In *Proceedings of the 12th International Meshing Roundtable*, pages 424–435, Santa Fe, New Mexico, USA, September 14-17 2003.

[135] B. van Rietbergen, H. Weinans, R. Huiskes, and A. Odgaard. A New Method to Determine Trabecular Bone Elastic Properties and Loading Using Micromechanical Finite-Element Models. *Journal of Biomechanics*, 28(1):69–81, 1995.

[136] L. Velho and J. Gomes. Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics Forum*, 19(4):195–214, 2000.

[137] N. Viswanath, K. Shimada, and T. Itoh. Quadrilateral Meshing with Anisotropy and Directionality Control via Close Packing of Rectangular Cells. In *Proceedings of the 9th International Meshing Roundtable*, pages 217–225, New Orleans, Louisiana, USA, October 2000.

[138] M. Wagner, K. Hormann, and G. Greiner. C^2-Continuous Surface Reconstruction with Piecewise Polynomial Patches. Preprint, September 2003.

[139] R.T. Whitaker. Reducing Aliasing Artifacts in Iso-Surfaces of Binary Volumes. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 23–32, Salt Lake City, UTAH, USA, October 9-10 2000.

[140] S.-T. Wu and M. Marquez. A Non-Self-Intersection Douglas-Peucker Algorithm. In *Proceedings of XVI Brazilian Symposium on Computer Graphics and Image Processing*, pages 60–66, São Carlos, São Paulo, Brazil, October 12-15 2003.

[141] C.-K. Yap. Symbolic Treatment of Geometric Degeneracies. *Journal of Symbolic Computation*, 10(3-4):349–370, 1990.

[142] L. Ying and D. Zorin. A Simple Manifold-Based Construction of Surfaces of Arbitrary Smoothness. *ACM Transactions on Graphics*, 23(3):271–275, 2004.

[143] G. Yngve and G. Turk. Robust Creation of Implicit Surfaces from Polygonal Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):346–359, 2002.

[144] T.S. Yoo, B. Morse, P. Rheingans, K.R. Subramanian, and M.J. Ackerman. Anatomic Modeling from Unstructured Samples Using Variational Implicit Surfaces. In *Proceedings of the Medicine Meets Virtual Reality Conference*, volume 81 of *Studies in Health Tecnology and Informatics*, pages 594–600, January 24-27 2001.

[145] Y. Zhang, C. Bajaj, and B.-S. Sohn. 3D Finite Element Meshing from Imaging Data. *Computer Methods in Applied Mechanics and Engineering (to appear)*, 2005.

[146] T. Zhou and K. Shimada. An Angle-Based Approach to Two-Dimensional Mesh Smoothing. In *Proceedings of the 9th International Meshing Roundtable*, pages 373–384, New Orleans, Louisiana, USA, October 2000.

[147] G.M. Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1994.