Three Dimensional Boundary Conforming Delaunay Mesh Generation

vorgelegt von
Master of Science
Hang Si
aus Hangzhou, China

Von der Fakultät II - Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

genehmigte Dissertation

Promotionsausschuß:

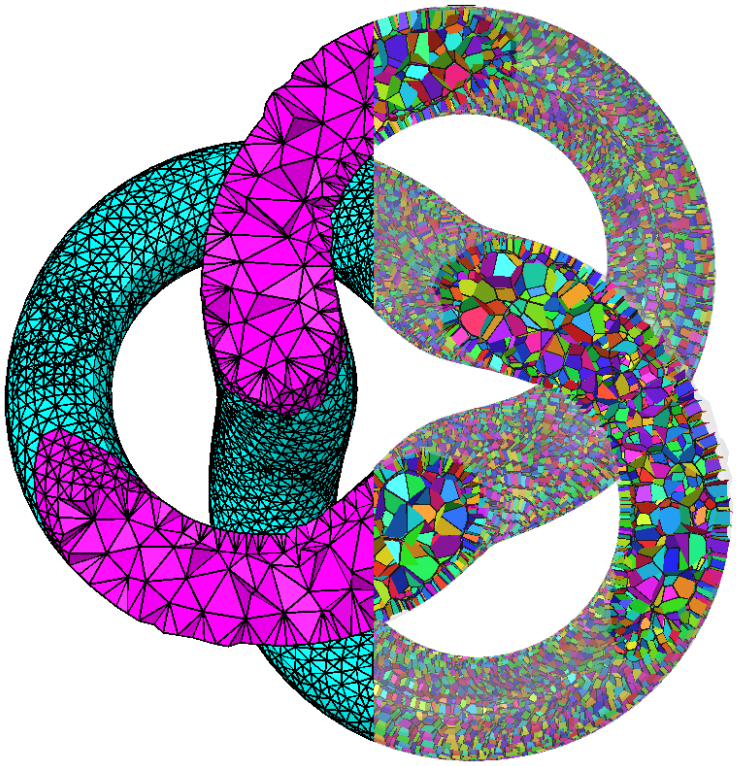
Vorsitzender: Prof. Dr. Dietmar Hömberg
Berichter: Prof. Dr. Günter M. Ziegler
Berichter: Prof. Dr. Herbert Edelsbrunner (Duke University, USA)

Tag der wissenschaftlichen Aussprache
7. Juli 2008

Berlin 2008

D 83

Three Dimensional Boundary Conforming Delaunay Mesh Generation



Hang Si

PhD Dissertation, TU Berlin

July 2008

⋆ The picture in the former page shows two partitions of a three-dimensional solid: a boundary conforming Delaunay mesh (left) consisting of Delaunay tetrahedra and a polyhedral subdivision (right) consisting of Voronoi cells. They are dual to each other. The Voronoi faces are randomly colored for visualization. The mesh was generated by the program `TetGen` [110] using an algorithm developed in Chapter 3 (see also Fig. 3.20) of this thesis.

Zusammenfassung

Diese Arbeit wird durch das Ziel motiviert, numerische Methoden für die Lösung partieller Differentialgleichungen zu unterstützen. Zu diesen Methoden zählen das Finite-Elemente- und das Finite-Volumen-Verfahren. Um diese zu nutzen, muss ein gegebenes Gebiet zunächst in viele einfache Zellen unterteilt werden. Die Genauigkeit und Konvergenz der Methode wird von der Qualität der Unterteilung stark beeinflusst. Ein *randkonformes Delaunay-Gitter* ist eine Unterteilung eines polyedrischen Gebiets in Delaunay-Simplices, sodaß alle Randsimplices die verallgemeinerte Gabriel-Eigenschaft haben. Diese Bedingungen sind für das *Voronoi-box basierte Finite-Volumen-Verfahren* wichtig, um wesentliche, qualitative Eigenschaften vom stetigen auf das diskrete Problem zu übertragen. In dieser Arbeit untersuchen wir das Problem der Erzeugung dreidimensionaler randkonformer Delaunay-Gitter mit guter Qualität.

Ein einfacher und effizienter Algorithmus, um randkonforme Delaunay-Gitter zu erzeugen, ist die Delaunay-Verfeinerung. Der Algorithmus von Shewchuk wird neu analysiert. Neue Ergebnisse für den Abschluß des Verfahrens in endliche vielen Schritten in Abhängigkeit vom Winkel der Ausgangsgeometrie, für die Grenzen geometrischer Kenngrößen der erzeugten Tetrader und für die Verteilung der Gittergröße werden erzielt. In der Praxis ist festzustellen, dass dieser Algorithmus gewöhnlich schneller als die bewiesenen Abschätzungen terminiert. Weiterhin schlagen wir einen adaptiven Algorithmus für die Delaunay-Verfeinerung vor, welcher die Terminierung für alle gültigen Eingabegitter garantiert.

Es ist bekannt, dass nicht alle Polyeder tetraedrisiert werden können, ohne zusätzliche Punkte (so genannte *Steiner-Punkte*) einzuführen. Die Erzeugung dreidimensionaler randkonformer Delaunay-Gitter ist mit vielen Schwierigkeiten verbunden. Ein Algorithmus zur Erzeugung *eingeschränkter Delaunay-Tetraedrisierungen* wird vorgestellt. Dieser Algorithmus fügt wenige Steiner-Punkte am Gebietsrand hinzu. Die Korrektheit wird für ein beliebiges, gültiges polyedrisches Gebiet bewiesen. Fragen der Komplexität werden diskutiert. Der Algorithmus ist praktisch effizient.

Abstract

This work is motivated by the aim to support numerical methods to solve partial differential equations. Among them are finite element and finite volume methods. For these, a given domain must first be subdivided into many simple cells. The quality of the subdivision will tremendously affect the accuracy and convergence of the method. A *boundary conforming Delaunay mesh* is a partition of a polyhedral domain into Delaunay simplices and all boundary simplices satisfy the Gabriel property. It is important in *Voronoi-box based finite volume schemes*. It allows to carry important qualitative properties from the continuous to the discrete level. In this work, we study meshing problems for the generation of three-dimensional good quality boundary conforming Delaunay meshes.

A simple and efficient algorithm to generate boundary conforming Delaunay meshes is Delaunay refinement. Shewchuk's algorithm is reanalyzed, achieving new results on the termination (angle) condition, on the bounds on characteristics of the tetrahedral shape and on the mesh size distribution. In practice, it is observed that this algorithm usually greatly outperforms the proven estimates. Next we propose an adaptive Delaunay refinement algorithm which guarantees the termination for all valid inputs.

It is known that not all polyhedra can be tetrahedralized without using additional points (so-called *Steiner points*). The three-dimensional boundary conforming Delaunay mesh generation problem has many difficulties. An algorithm to construct *constrained Delaunay tetrahedralizations* is presented. This algorithm adds few Steiner points on the domain boundary. The correctness is proven for an arbitrary valid polyhedral domain. The complexity issues are discussed. It is practically efficient.

Preface

In August 2001, I posted a piece of code for generating tetrahedral meshes in a mailing list discussing mesh generation techniques. I called it "tetgen". The code was primarily written for my Master's thesis in Computer Science from Zhejiang University in Hangzhou. A few months later, I got an email from Dr. Jürgen Fuhrmann from Weierstrass Institute for Applied Analysis and Stochastics (WIAS) Berlin. He asked me if I'd like to give a seminar in his research group. I eagerly accepted and traveled to Berlin in March, 2002. During my visit in WIAS, I had stimulating discussions with Dr. Klaus Gärtner and other colleagues. I became convinced that 3D Delaunay mesh generation is an important research topic. That's when I decided to work on it. In October of 2002, I joined the research group "numerical mathematics and scientific computing" of WIAS.

This work is supported by the project "Numerical Methods" of WIAS in the topics of "Numerical solution of partial differential equations".

I would like to express my sincere gratitude to Prof. Günter M. Ziegler, who agreed to be the advisor of my PhD thesis. I appreciate the time he invested in our discussions and email exchanges. He always gave me prompt answers to the questions on which I got stuck. He is without a doubt one of the greatest teachers. I've enjoyed the lectures he taught at TU Berlin over the last three semesters.

It was a great chance to meet Prof. Herbert Edelsbrunner from Duke University, who was invited by the Berlin Mathematical School for one year. I am grateful for our enlightening discussions and for his invaluable assistance with reading and editing my manuscripts. His comments always forced me to re-consider the questions like: Does this definition cause ambiguity? Is this proof complete? That greatly improved this thesis.

I'm grateful to Prof. Dietmar Hömberg for being the chairman of my defense committee.

I'm heartily grateful to all my colleagues in WIAS. First of all, I would like to extend my honest thanks to Jüergen Fuhrmann and Klaus Gärtner, who initially motivated this challenging topic and kept helping my work all the time. I would also like to thank Jürgen Fuhrmann for all the help he has given me. His `pdelib` is extremely useful in my work. Besides, all the meshes shown in my thesis were created through his visualization toolkit

`gltools`. I owe a lot to Klaus Gärtner, who, from the beginning of the project until now, has been a continuous help to me. I can't remember how many evenings and weekends we've spent in discussions. Our work on constrained Delaunay triangulations (CDTs) is one of the important parts of my thesis.

Thanks to Hartmut Langmach, who shared the same office with me for my first two years in WIAS, for all the assistance he's given me; Timo Streckenbach for the support in using `pdelib` and co-developing `TetView`, Thomas Koprucki for the help of migrating from Linux to MacOSX, Alexander Linke for explaining me finite element methods. Special thanks to Manfred Uhle, Jürgen Borchardt, Ralf Krahl for smoothing my "Umzug nach Berlin" (move to Berlin) with wall paper, paint, tiles, etc. Thanks to Frau Lawrenz, our secretary, who is an expert at organizing our work and life.

Special thanks goes to Jonathan Shewchuk from UC Berkeley. It was his pioneering work in Delaunay mesh generation and his excellent software `Triangle` that stimulated this work and the development of `TetGen`. His visit to WIAS in 2005, in which he shared his unpublished manuscript on CDTs and his code `Pyramid`, truly opened my mind. I would like to thank Drosos Kourounis, my friend from Greece, who all along has been an active critic of `TetGen`. It was refreshing to be in his beautiful island, Kalymnos, in the south-eastern Aegean Sea. I would also like to thank those users of `TetGen` who sent me feedbacks and suggestions and told me about their various applications of the software.

I thank my parents and two sisters for their love. I would not have completed my university and graduate education without the encouragement and financial support of my parents. I am also grateful for my son. Playing with him has filled my spare time with joy.

The last place of honor absolutely goes to my wife, Xiaojun, for her endless tolerance and love. Without her, this work would definitely not exist.

Berlin, July 2008 Hang Si

Contents

Zusammenfassung

Abstract **i**

Preface **ii**

1 Introduction **1**
1.1 Voronoi Finite Volumes 2
1.2 The Meshing Problems 3
1.2.1 Delaunay Mesh Generation 3
1.2.2 Mesh Adaptation 6
1.2.3 Boundary Conformity 7
1.3 Thesis Outline 10

2 Preliminaries **13**
2.1 Basic Notions 13
2.2 Polyhedra and Faces 16
2.3 Piecewise Linear Systems 18
2.4 Triangulations and Meshes 20

3 Boundary Conforming Delaunay Mesh Generation **23**
3.1 Boundary Conforming Delaunay Meshes 24
3.2 Delaunay Refinement 26
3.2.1 Tetrahedron Shape Measures 26
3.2.2 Point Generating Rules 27
3.2.3 The Algorithm 29
3.3 Analysis 30
3.3.1 Proof of Termination 30
3.3.2 Output Edge Lengths 35
3.3.3 Mesh Quality 46
3.3.4 Vertex Degree 51
3.3.5 Insertion Orders 53
3.3.6 Output Size 56
3.4 Improvement 59

3.4.1 Relaxed Input Angle Assumption 60
3.4.2 Experiments on Sliver Removal 62
3.5 Summary 65

4 Adaptive Delaunay Mesh Generation 67
4.1 Constrained Delaunay Refinement 68
4.1.1 Mesh Sizing Functions 68
4.1.2 Sparse and Protecting Balls 68
4.1.3 Point Generating Rules 69
4.1.4 Point Accepting Rules 70
4.1.5 The Algorithm 71
4.2 Analysis 72
4.2.1 Proof of Termination 72
4.2.2 Mesh Quality 74
4.2.3 Mesh Conformity 76
4.3 Examples 78
4.4 Summary 80

5 Constrained Delaunay Mesh Generation 83
5.1 Constrained Delaunay Triangulations 84
5.1.1 Definitions 84
5.1.2 Basic Properties 85
5.2 The CDT Algorithm 88
5.2.1 Correctness 88
5.3 Segment Recovery 89
5.3.1 Segment Splitting Rules 89
5.3.2 The Algorithm 92
5.3.3 Proof of Termination 92
5.3.4 Comparison with Other Algorithm 94
5.4 Facet Recovery 96
5.4.1 The Algorithm 96
5.4.2 Proof of Termination 98
5.4.3 Correctness 101
5.4.4 Complexity 102
5.5 Local Degeneracies Removal 103
5.5.1 Local Degeneracies and Break Points 104
5.5.2 The Algorithm 104
5.6 Examples 105
5.7 Summary 109

6 Conclusions 111

A Implementation **113**
A.1 Description of Mesh Domains 113
A.1.1 CSG and B-Rep Models 113
A.1.2 A PLS Boundary Description 114
A.1.3 Data File Description 116
A.2 Mesh Data Structures 116
A.2.1 Triangle-Edge Data Structure 118
A.2.2 Tetrahedron-based Data Structure 119
A.3 Implementing the CDT Algorithm 121
A.3.1 Tetrahedralizing the Vertices 121
A.3.2 Triangulating the Boundaries 121
A.3.3 Recovering the Segments 123
A.3.4 Recovering the Facets 125

Bibliography **127**

Notations **137**

Index **141**

Curriculum Vitae **145**

List of Figures

1.1 Numerical simulation of the airflow 1
1.2 Finite volume partitions and control volumes 2
1.3 A three-dimensional polyhedral domain 4
1.4 Boundary conforming Delaunay property 5
1.5 Adaptive numerical simulation 6
1.6 Untetrahedralizable polyhedra 8
1.7 Comparison of different approaches 9
1.8 A constrained Delaunay triangulation 10

2.1 A two-dimensional simplicial complex 15
2.2 A two-dimensional polyhedral complex 16
2.3 Polyhedra and faces 17
2.4 A convex piecewise linear system 19
2.5 A partition of PLS 20
2.6 Triangulations and meshes 21

3.1 A degeneracy 24
3.2 A conforming Delaunay mesh 25
3.3 Tetrahedra of different shapes 27
3.4 A three-dimensional PLS $\mathcal{X}$ 28
3.5 The Delaunay refinement algorithm 29
3.6 Proof of Lemma 3.3.2 32
3.7 Proof of Lemma 3.3.2 33
3.8 The resulting meshes with different ρ_0. 36
3.9 Proof of Lemma 3.3.6 37
3.10 Proof of Lemma 3.3.7 39
3.11 All possible cases of $\mathbf{b_V}$ 42
3.12 Illustration of Theorem 3.3.9 46
3.13 Proof of Lemma 3.3.12 48
3.14 Proof of Lemma 3.3.13 50
3.15 Proof of the upper bound of the vertex degree 52
3.16 PLS models 54
3.17 Relatively short edges 56
3.18 A modified Delaunay refinement algorithm 57

3.19 The PLS R_{10} 60
3.20 A boundary conforming Delaunay mesh 62
3.21 The remaining slivers 63

4.1 Vertex balls 69
4.2 Point accepting rules 70
4.3 The Adaptive Delaunay refinement algorithm 71
4.4 Proof of Lemma 4.2.1 73
4.5 Proof of Theorem 4.2.3 75
4.6 Test case for various combinations of parameters (ρ_0, α_2). 77
4.7 Meshes created by specifying different sizing functions 77
4.8 Adaptive tetrahedral mesh of the Boeing 747 model 79
4.9 Mesh quality plot of the Boeing 747 model 79
4.10 Adaptive meshes of a moving laser spot 80

5.1 Visibility and constrained Delaunay 84
5.2 Proof of the constrained Delaunay Lemma (Theorem 5.1.4) 86
5.3 Shewchuk's CDT Theorem [100] 89
5.4 A reference point 90
5.5 Segment splitting rules 91
5.6 An example of protecting ball 92
5.7 The segment recovery algorithm 93
5.8 Comparison with other algorithm 94
5.9 A counter example 95
5.10 A missing region 97
5.11 Tetrahedralize Cavity Algorithm 98
5.12 The verification of the cavity (illustrated in 2D) 99
5.13 Cavity tetrahedralization (illustrated in 2D) 99
5.14 The facet recovery algorithm 100
5.15 Examples (`Layers`) 102
5.16 Local degeneracies and the break points 104
5.17 The Local Degeneracy Removal algorithm 105
5.18 Example (`Cami1a`) 106
5.19 Example: `Wing-Iso` 107
5.20 Examples: `Crystal` (left) and `IFP` (right) 109
5.21 Example: `Wing-Aniso` 110

List of Tables

3.1 Tetrahedral shape measures 27
3.2 Point insertion orders 54
3.3 Comparison of the output sizes for point clouds 55
3.4 Comparison of the output sizes for PLSs 55
3.5 Tests on the relations between the input parameters ρ_0, Δ 59
3.6 Sliver removal experiments 64

4.1 Statistics of the ratios S_v and L_v 78

5.1 Runtime statistics of the CDT algorithm 108

Chapter 1

Introduction

This work studies unstructured mesh generation problems in three dimensions with an emphasis on its application in the computer simulation of physical and engineering problems described by partial differential equations (PDEs). Both finite element (FEM) and finite volume methods (FVM) are well suited for numerically solving PDEs, and are extensively used in various engineering fields. Fig. 1.1 illustrates such an example.

The very first step in solving the problem numerically is to create a suitable mesh for the simulation domain. This process, referred as *mesh generation*, has a tremendous effect on the computed results, either in the numerical error or in the computer resources used. FEM and FVM will benefit from improved mesh quality. On the other hand, many interesting theoretical properties have been discovered about the numerical methods, which in turn suggest the use of particular types of optimal meshes. *Voronoi-box based finite volume approximations* can be best solved on *boundary conforming Delaunay meshes*. In the plane, such meshes can be generated using optimal algorithms. For arbitrary three-dimensional domains, however, this problem remains a major challenge both in theory and practice.

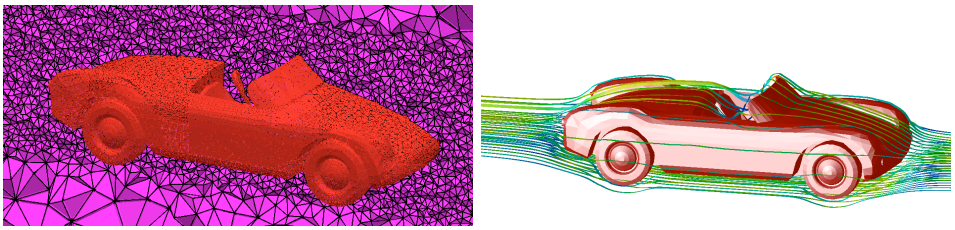


Figure 1.1: Numerical simulation of the airflow around an automobile (solving the Navier-Stokes equations). Left: a tetrahedral mesh (generated by TetGen [110]) used by a FEM solver (`AcuSolve` [1]). Right: stream lines visualize the velocity field (plotted by `OpenDX` [117]).

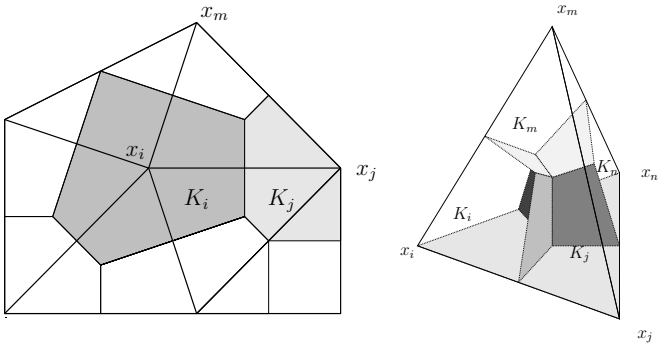


Figure 1.2: Examples of finite volume partitions and control volumes in 2D and 3D. K_i and K_j are two adjacent control volumes. The line segment going through $\mathbf{x}_i$ and $\mathbf{x}_j$ is orthogonal to the interface face $K_i \cap K_j$.

1.1 Voronoi Finite Volumes

Physical problems governed by conservation laws are often treated by finite volume methods. The motivation behind the focus on boundary conforming Delaunay meshing is the Voronoi box based finite volume method for PDEs. It allows to carry important qualitative properties from the continuous to the discrete level. To give a short description of this method, we consider the convection-diffusion problem

$$c_t - \nabla \cdot (D\nabla c - c\mathbf{v}) = 0, \quad (\mathbf{x}, t) \in \Omega \times [0, T] \tag{1.1}$$

in a 2D or 3D domain Ω with appropriate initial and boundary conditions. Here, c can be seen e.g. as a species concentration. D is a diffusion coefficient, and $\mathbf{v}$ is a velocity field or a potential gradient ($\nabla\phi$). This setting includes solute transport in fluids and charge carrier transport in semiconductor devices. For Dirichlet boundary conditions and $\nabla\cdot\mathbf{v} = 0$, c is bounded from below and above by its boundary and initial values. If the lower bound is nonnegative, c stays nonnegative. The *Voronoi box based finite volume method* [75] , known also as "box method", "control volume method", allows to carry over this maximum principle to the discretized equations.

Given a partition of Ω whose cells (called control volumes) are all Voronoi cells [118] (see Fig. 1.2 for examples) and a partition $0 = t^0 < t^1 < \ldots < t^n = T$, for each space-time control volume $K \times (t^{n-1}, t^n)$, we integrate (1.1), and apply the Gauss theorem to the integral of the flux divergence. After choosing a quadrature in time which yields an implicit scheme, and dividing by the time difference, with $c^n = c(t^n, \mathbf{x})$ we arrive at

$$\int_K \frac{c^n - c^{n-1}}{t^n - t^{n-1}} \mathrm{d}\mathbf{x} - \int_{\partial K} (D\nabla c^n - c^n\mathbf{v}) \cdot \mathbf{n}_K \mathrm{d}s = 0. \tag{1.2}$$

Splitting the surface integral into contributions from the facets common with the neighboring control volumes $L \in \mathcal{N}(K)$, applying quadrature rules, and introducing the flux function $g(c_K, c_L, v_{KL})$ to approximate the scaled normal flux $(-D\nabla c + c\mathbf{v})\cdot(\mathbf{x}_K - \mathbf{x}_L)$ through a common facet $\sigma_{KL} = \partial K \cap \partial L$ yields

$$|K|\frac{c_K^n - c_K^{n-1}}{t^n - t^{n-1}} + \sum_{L\in\mathcal{N}(K)} \frac{|\sigma_{KL}|}{|\mathbf{x}_K - \mathbf{x}_L|} g(c_K^n, c_L^n, v_{KL}) = 0 \qquad (1.3)$$

Here, c_K^n is the average value of c^n in K, and $v_{KL} = \frac{1}{|\sigma_{KL}|}\int_{\sigma_{KL}} \mathbf{v}\cdot\mathbf{n}ds$ is the average normal flux of $\mathbf{v}$ through σ_{KL}. Any consistent upwind 1D finite difference expression of the flux along $\mathbf{x}_K\mathbf{x}_L$ defines a flux function g which leads to a stable and converging scheme. A preferable choice is the exponential fitting scheme [2, 63, 94] which can be derived from the analytical solution of a two-point boundary value problem arising from the projection of (1.1) onto $\mathbf{x}_K\mathbf{x}_L$ [42].

Per time step, this approach yields a linear system of equations $c^n + \mathbf{A}c^n = \mathbf{B}c^{n-1}$ where $\mathbf{B}$ is a positive diagonal matrix, and the graph of $\mathbf{A}$ is connected, $\mathbf{A}$ has column sum zero, non-positive off-diagonal entries and nonnegative main diagonal entries. Therefore $\mathbf{I} + \mathbf{B}^{-1}\mathbf{A}$ has the M-property [115], immediately resulting in $c^n \geq 0$, once $c^{n-1} \geq 0$. A convergence theory for finite volume methods which covers the Voronoi case is available in [43]. Discrete theory for the nonlinear case, including a proof of the local maximum principle for interior points can be found in [48]. A generalization of exponential fitting to the nonlinear case is available in [42]. Coupled equations, where $\mathbf{v} = \nabla\phi$ are handled in [50, 51, 54]. In particular, [50] contains a proof of the dissipativity of the scheme. Coupling to the pointwise divergence free Scott Vogelius finite element for the Navier-Stokes equation has performed in [71].

A prerequisite for applying the Voronoi finite volume method for solving PDEs defined on an arbitrary domain Ω is *to generate a partition of* Ω *which has certain desired properties*. For example, one basic geometrical requirement is that all control volumes are Voronoi cells.

1.2 The Meshing Problems

This section gives a tour of the meshing problems treated in this thesis. We attempt to describe them as general as possible. Special terms will be explained when they are encountered. Formal mathematical definitions will be given in the later chapters.

1.2.1 Delaunay Mesh Generation

Let $\Omega \subset \mathbb{R}^3$ be a *physical domain* for numerical simulation. The boundary of Ω is denoted as $\partial\Omega$. An example of a polyhedral domain is shown in Fig. 1.3

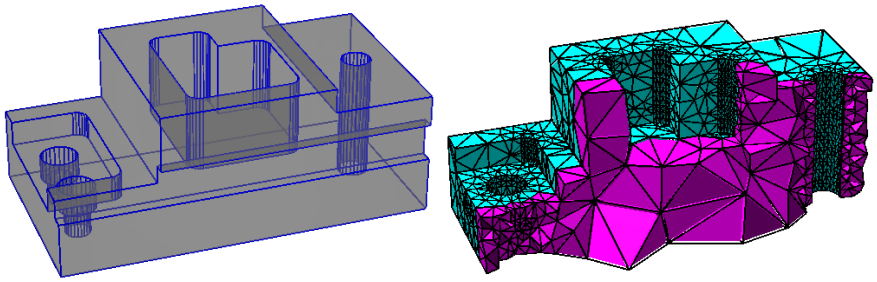


Figure 1.3: A three-dimensional polyhedral domain (left) and a partition (or subdivision) (right) of the left domain.

left. In general, $\partial\Omega$ may consist of curved surfaces. It is necessary that Ω contains *internal boundaries* which may separate itself into sub-domains so that the discontinuity between different materials can be modeled. Hence Ω is in general not a topological manifold.

A *partition* (or subdivision) of Ω is a finite set $\mathcal{T}$ of simple cells such that: *(i)* any two cells of $\mathcal{T}$ are either disjoint or intersect at their common face, *(ii)* the union of $\mathcal{T}$ equals to Ω, and *(iii)* $\partial\Omega$ is represented by the union of a subset of $\mathcal{T}$. Each cell in $\mathcal{T}$ may be any convex polytope. The simplest cell is called a *simplex*, e.g., edges, triangles, and tetrahedra are simplices in $\mathbb{R}^3$. Fig. 1.3 right shows a partition whose cells are all simplices.

Let S be a finite set of points in $\mathbb{R}^d$. A simplex σ in S is *Delaunay* [28] if it has a circumscribed sphere Σ such that no other point of S lies inside Σ. Moreover, σ is *Gabriel* [49] if no other point of S lies inside the *diametrical sphere* of σ, i.e., the smallest circumscribed sphere of σ. A *boundary conforming Delaunay mesh* of a domain Ω is a partition of Ω into a set $\mathcal{T}$ of simplices with the following properties

(i) every simplex in Ω is Delaunay, and
(ii) every simplex in $\partial\Omega$ is Gabriel.

Property *(i)* ensures that $\mathcal{T}$ is a *conforming Delaunay mesh* of Ω whose geometric dual is a Voronoi partition of Ω. However, such a partition may not satisfy the requirements of the Voronoi-box based finite volume scheme. A critical case is illustrated in Fig. 1.4 left. A Voronoi cell from one sub-domain may cross an internal boundary if the simplex on $\partial\Omega$ does not have the Gabriel property. If such case happens, equation (1.2) may not be evaluated correctly (since the diffusion coefficient D may be discontinuous at $\partial\Omega$). The property *(ii)* is included for avoiding such cases, see Fig. 1.4 right.

Problem 1.2.1 *Given a three-dimensional domain Ω, generate a boundary conforming Delaunay mesh $\mathcal{T}$ of Ω such that* (1) *the mesh size of $\mathcal{T}$ is bounded, and* (2) *the mesh quality of $\mathcal{T}$ is optimal.*

This two extra requirements are generally needed by numerical methods for achieving high accuracy with low computational costs.

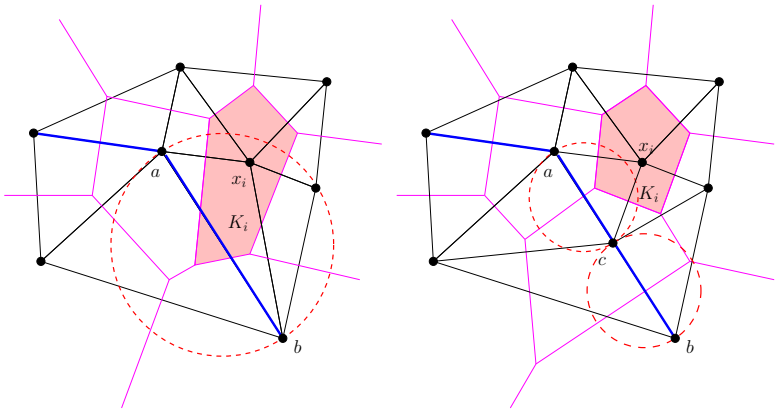


Figure 1.4: Edge **ab** is in $\partial\Omega$. Left is a Delaunay mesh of Ω whose dual is a Voronoi partion of Ω. Since **ab** does not have the Gabriel property, the Voronoi cell K_i of one subregion intersects with **ab**. The mesh in the right is a boundary conforming Delaunay mesh of Ω.

The first requirement means that for any input of size $O(n)$, where n is the number of vertices (or edges) of Ω, the output mesh size m should not arbitrarily large with respect to n. We say that the output mesh size is *bounded* if $m = O(n^p)$, where p is some fixed constant.

The second requirement needs further explanations. First of all, the term "mesh quality" depends on the applications. In numerical simulations, the mesh quality is justified by both the numerical accuracy of the solution and the used computing resources. It is usually determined by the combination of several measures on element shape, size, and orientation [61]. In general, the mesh quality can be expressed by an *object function* $f : \mathcal{P}_T \to \mathbb{R}$, where $\mathcal{P}_T$ is the collection of all meshes of Ω, and $f(\mathcal{T})$ gives a real value for each $\mathcal{T} \in \mathcal{P}_T$. The *optimal mesh* is defined to be the one on which f attains its minimum. For an example, if the mesh is used for function approximation, then Delaunay triangulations minimize the interpolation error for the lifting function $\|\mathbf{x}\|^2$ among all other triangulations of the same set of vertices, see [27, 17].

One of the essential requirements on mesh quality is the element shape. A general shape measure is the *aspect ratio* λ which is defined as the ratio between the longest and shortest diameter of the element. The smaller λ implies the better shape. For a tetrahedron τ, the *minimum dihedral angle* $\theta(\tau)$ between two faces of τ is also a shape measure.

A classical method to generate Delaunay meshes is *Delaunay refinement*, first introduced by Chew [23] and Ruppert [92] for meshing two-dimensional polygonal domains. The basic scheme is to insert the circumcenters of badly-shaped elements and maintain the Delaunay triangulation of the augmented

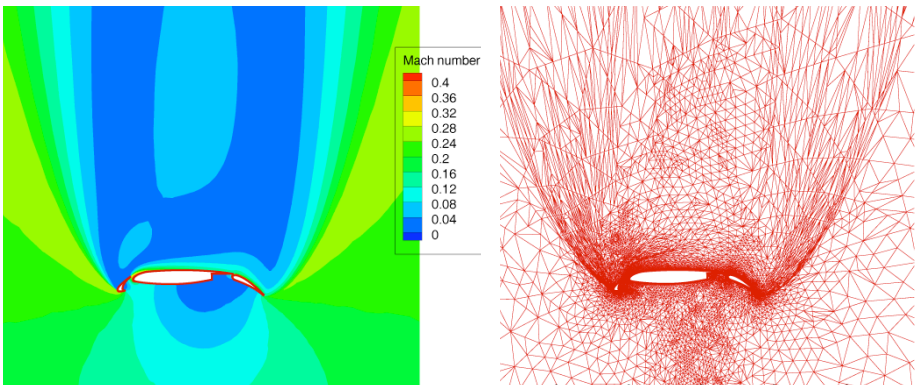


Figure 1.5: Adaptive numerical simulation of the lift versus angle-of-attack for a multi-element airfoil (`GGNS` [114]). Left: Mach distribution at 90° angle-of-attack. Right: An intermediate adapted mesh.

point set. It has theoretical guarantees on the shape (angle) of mesh elements and a graded mesh size distribution. Moreover, it results boundary conforming Delaunay meshes. The technique of Delaunay refinement has been generalized into three dimensions by many authors [30, 25, 99, 21, 20, 91]. However, most of the algorithms do not guarantee boundary conforming Delaunay meshes except the algorithm of Shewchuk [99].

Shewchuk's algorithm [99] is generalized from Ruppert's algorithm [92]. The boundary enforcement and the removal of badly-shaped tetrahedra are performed simultaneously through priority point insertion rules. It is proved that this algorithm has theoretical guarantees on both of the tetrahedral shape and the mesh size distribution. Moreover, it is practically efficient. However, it only works for restricted non-convex polyhedral domains, and the proven theoretical guarantees are weak. In Chpater 3, we will discuss this algorithm in great detail.

1.2.2 Mesh Adaptation

In general the nature of the exact solution of a given problem is not known beforehand. Then it is not clear how to generate a mesh with a small degree of freedom and resolve the detailed features of the solution, such as the edge or corner singularities and shock fronts. Adaptive numerical methods seek the best approximated solution at a low computational cost through a sequence of computed solutions on successively changed meshes, see e.g., [5, 66, 116, 4, 10]. Fig. 1.5 shows such an example.

Let $\mathbf{u}$ be the exact solution, and δ be a given tolerance. Each adaptive loop i mainly includes five subsequent phases: (1) the computation of an approximated solution $\mathbf{u}_i$ by FEM (or FVM), (2) the estimation of the error, e.g., $\mathbf{e}_i = \mathbf{u} - \mathbf{u}_i$, by an a posteriori error estimator, (3) the generation of

a *mesh sizing function* H_i from $\mathbf{e}_i$, (4) the generation of an adapted mesh $\mathcal{T}_{i+1}$ conforming to H_i, and (5) the interpolation of the solution $\mathbf{u}_i$ on $\mathcal{T}_{i+1}$. The whole adaptive process can then be viewed as a non-linear optimization problem [66]. The goal is to find a mesh $\mathcal{T}_k$ with as few degrees of freedom as possible, and satisfies the termination criterion, e.g., $\|\mathbf{u} - \mathbf{u}_k\|_{L^2} \leq \delta$. The convergence of adaptive finite element methods for elliptic problems has been theoretically proved [33].

Mesh adaptation, i.e., the phase (4) in the above process, is one of the key steps in adaptive numerical methods. Let Ω be a three-dimensional simulation domain, and let the appropriate mesh sizing function H be defined over Ω and it indicates the desired size of mesh elements, note that H may be anisotropic. One of the approaches to obtain a good quality tetrahedral mesh of Ω with the mesh size conforming to H, takes mainly three steps:

(1) construct an initial tetrahedral mesh $\mathcal{T}$ of Ω,
(2) add new points into $\mathcal{T}$ to conform H, and
(3) optimize $\mathcal{T}$ to improve the mesh quality.

Each of these steps can be extended into a more complex process and is a topic of interest in mesh generation. For a comprehensive survey of these technologies, we refer to [47] and [77]. In this work, we focus on how to efficiently perform the step *(2)* in the above process.

Problem 1.2.2 *Given an initial tetrahedral mesh $\mathcal{T}$ of a three-dimensional domain Ω and an isotropic mesh sizing function H defined on Ω, insert new points into $\mathcal{T}$ to form a good quality tetrahedral mesh $\mathcal{T}'$ of Ω such that the mesh size of $\mathcal{T}'$ conforms to H.*

A central question in the above problem is how to place the new points such that the requirements are simultaneously satisfied. Various approaches have been developed for this purpose, such as advancing-front methods [73, 74], Octree-based methods [121, 82], Delaunay-based methods [80, 25], and the combinations of them [119, 53]. Among these methods, the Delaunay-based methods which utilize the Delaunay criterion and the Delaunay triangulation [28] are the most robust and efficient. In Chpater 4 we present an incremental point insertion approach extended from the basic Delaunay refinement scheme.

1.2.3 Boundary Conformity

It is known that convex polyhedra are easy to tetrahedralize, but additional vertices, called *Steiner points*, may be needed to tetrahedralize a non-convex polyhedron, see examples in Fig. 1.6. The problem of determining whether or not a non-convex polyhedron can be tetrahedralized without Steiner points is NP-complete [93]. Chazelle [14] shows that a quadratic number of Steiner points may be required (see Fig. 1.6 right).

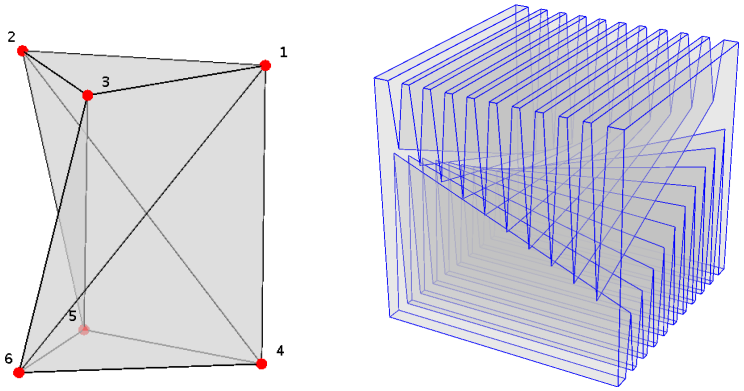
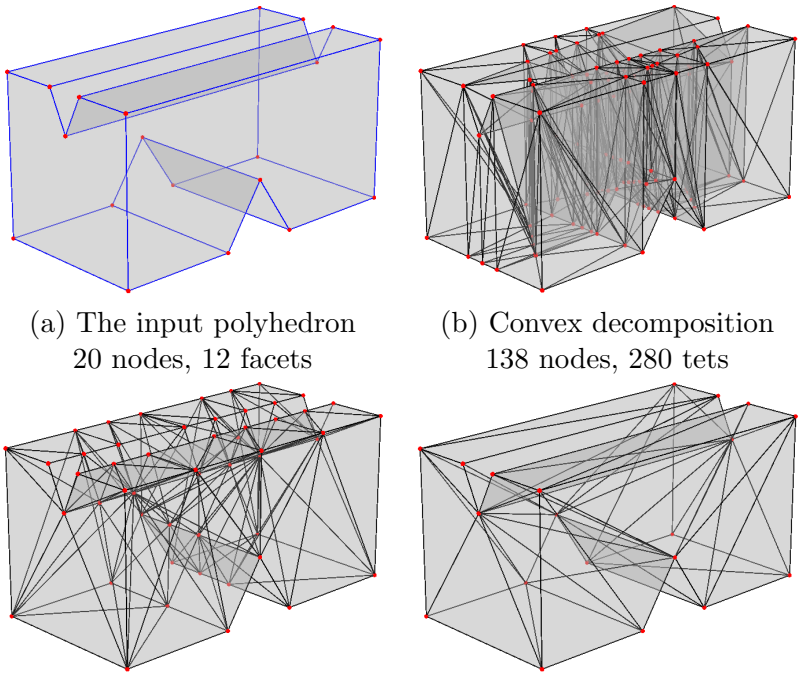


Figure 1.6: Two polyhedra which are not tetrahedralizable without using Steiner points. Left: The Schönhardt polyhedron [95] can be obtained by twisting the upper face around the axes of a parallel triangular prism by a small angle. Right: The Chazelle's polyhedron [14] which is formed by cutting wedges from a cube. In the middle of the polyhedron are two sets of orthogonal lines. The lower and upper lines lie on hyperbolic paraboloids $z = xy$, and $z = xy + \epsilon$, respectively.

Given a three-dimensional polyhedron P, the problem of *boundary conformity* asks to generate a tetrahedralization $\mathcal{T}$ containing the boundary of P as a subcomplex of $\mathcal{T}$. This problem is fundamental to many applications. For example, a better solution of this problem would imply a better solution to both of our problems 1.2.1 and 1.2.2.

Chazelle and Palios [16] proved a quadratic upper bound for the number of Steiner points by giving an algorithm to decompose a simple polyhedron P using $O(n + r^2)$ Steiner points, where r is the number of reflex edges (a quantitative measure of nonconvexity) of P. However, their algorithm will usually introduce an unnecessarily large number of Steiner points even for a simply-shaped polyhedron, see Fig. 1.7 (b). More practical approaches using conforming Delaunay triangulations [87, 26] and constrained Delaunay triangulations [103, 112] are proposed, see Fig. 1.7 (c) and (d). In practice, constrained Delaunay triangulations usually need less Steiner points than conforming Delaunay triangulations.

Let $\mathcal{G}$ be a planar straight line graph. Any two edges of $\mathcal{G}$ can only intersect at a common endpoint. A *constrained Delaunay triangulation* of $\mathcal{G}$ is a triangulation $\mathcal{T}$ of the vertices of $\mathcal{G}$ such that every edge of $\mathcal{G}$ is also an edge of $\mathcal{T}$, and the circumscribed circle of any triangle $\tau \in \mathcal{T}$ contains no vertex of $\mathcal{T}$ which is visible from the interior of τ, see Fig. 1.8. This definition is independently developed by Lee *et al.* [69] and Chew [23]. The same concept can be generalized into three and higher dimensions. While it is necessary to take Steiner points into account.



(a) The input polyhedron
20 nodes, 12 facets

(b) Convex decomposition
138 nodes, 280 tets

(c) Conforming Delaunay
51 nodes, 103 tets

(d) Constrained Delaunay
20 nodes, 29 tets

Figure 1.7: A comparison of meshing polyhedra by different approaches. (a) A simple polyhedron having 2 reflex edges. (b) Convex decomposition (by the program available from Palios's PhD thesis [89]). (c) Conforming Delaunay tetrahedralization (by `Pyramid` [107]). (d) Constrained Delaunay tetrahedralization (by `TetGen` [110]).

Let P be a d-dimensional polyhedron. The set of vertices of P is denoted as $V(P)$. Let S be a finite set of points in $\mathbb{R}^d$ and $V(P) \subseteq S$. Two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are *invisible* to each other if the line segments $[\mathbf{x}, \mathbf{y}]$ intersects any face F of P in a single point other than its endpoints. A simplex σ whose vertices are in S is *constrained Delaunay* if either it is Delaunay in S, or it has a circumscribed sphere does not contain any other vertices of S which is visible from the interior of σ.

A *constrained Delaunay triangulation* (abbreviated as CDT) of P is defined as a partition $\mathcal{T}$ of P such that $\mathcal{T}$ is a simplicial complex and every simplex of $\mathcal{T}$ is constrained Delaunay. By this definition, a CDT of P may contain Steiner points, i.e., those points in $S \setminus V(P)$. Lee *et al.* [69]'s and Chew [23]'s definition of a two-dimensional CDT is just a special case when $S = V(P)$. A three-dimensional CDT is also called a *constrained Delaunay tetrahedralization*. In general, there are infinitely many CDTs of P (by different choices of Steiner points). Our goal is to find a CDT of P which contains small number of Steiner points.

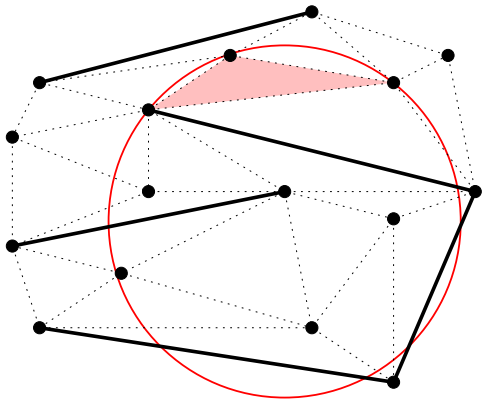


Figure 1.8: A constrained Delaunay triangulation of a set of vertices and edges (the solid lines). The circumcircle of the shaded triangle is not empty but no enclosed vertex is visible from its interior.

Problem 1.2.3 *Given a three-dimensional polyhedron P, generate a constrained Delaunay tetrahedralization $\mathcal{T}$ of P such that the number of Steiner points in $\mathcal{T}$ is as small as possible.*

A key question to the above problem is to determine under which condition Steiner points are not needed. Call an edge σ of P *strongly Delaunay* if there is a circumscribed sphere Σ of σ such that all other vertices lie strictly outside and not on Σ. Shewchuk [100] showed that if all edges of P are strongly Delaunay, then P has a CDT with no Steiner points. This condition is useful in practice. It suggests that Steiner points are only needed on some of the input edges. In [103, 112], practical algorithms for recovering Delaunay edges are proposed. Steiner points are inserted in such a way that no unnecessarily short edges will be introduced.

Another key question: Assume it is known that P has a CDT without Steiner points. How to efficiently construct such a CDT? So far, Shewchuk proposed several algorithms for this purpose [102, 103, 105]. Among them, the flip-based facet insertion algorithm [105] has better performance. Si and Gärtner [112] proposed another incremental facet recovery algorithm which is practically efficient and easy to implement.

A complete algorithm for Problem 1.2.3 is discussed in Chapter 5.

1.3 Thesis Outline

This thesis studies meshing problems for generating good quality boundary conforming Delaunay meshes for arbitrary three-dimensional domains. They are motivated by using the Voronoi-based finite volume method to solve partial differential equations. While its applications are far beyond,

the background of these problems connects different subjects in mathematics and computer science, such as numerical analysis, topology, discrete geometry, and computational geometry. The main objective of this thesis is to develop efficient algorithms to solve these problems, as good as possible, both in theory and practice. Below is the outline of the rest of this thesis.

Chapter 2 formalizes the mathematical definitions of the input and output objects for the algorithms discussed in this work. The input object, called *piecewise linear system* (abbreviated as PLS), is an approximation of a domain Ω. It is defined as a finite collection of (not necessarily convex) polytopes satisfying several properties. The *underlying space* of a PLS $\mathcal{X}$, denoted as $|\mathcal{X}|$, is the union of all polytopes of $\mathcal{X}$. A *triangulation* of a PLS $\mathcal{X}$ is defined as a simplicial complex $\mathcal{T}$ whose underlying space $|\mathcal{T}|$ equals to the convex hull of $|\mathcal{X}|$ and every polytope of $\mathcal{X}$ equals to the union of a subcomplex of $\mathcal{T}$. A *mesh* of $\mathcal{X}$ is defined as a subcomplex $\mathcal{K}$ of $\mathcal{T}$ such that $|\mathcal{K}| = |\mathcal{X}|$ (hence it may not be convex). The output object of our algorithms is either a triangulation or a mesh.

Chapter 3 studies the problem of good quality boundary conforming Delaunay mesh generation. A classical technique for generating such meshes is *Delaunay refinement*. Shewchuk's algorithm [99] is reanalyzed.

It is shown that this algorithm terminates for a larger class of inputs than previously proven. The restriction on the facet angle of a PLS $\mathcal{X}$ is reduced from 90° to $\arccos \frac{1}{2\sqrt{2}} \approx 69.3^\circ$. Moreover, the restriction on the planar angle of $\mathcal{X}$ (which is 60°) can be completely removed by a minor modification in the algorithm. The latter enables the generation of boundary conforming Delaunay meshes for a wider class of smooth surface meshes. We used a different approach to analyze the output edge lengths. New lower bounds on edge lengths are derived. Moreover, we introduce a general formula to describe the distribution of the edge lengths in the output mesh. As a consequence, the lower bound of a shape measure (the radius-edge ratio) on output tetrahedra can be reduced from 2 to $\sqrt{2}$ with a simple assumption on the inputs (which is unlikely needed in practice).

Previous work showed that the running time of the Delaunay refinement algorithm is nearly linear with respect to the output size of the mesh. It is observed that the change of point insertion order will result in various output sizes. Several point insertion orders are investigated for studying the complexity of the algorithm. Further on, the output size of the algorithm with a specific insertion order is analyzed. By a modest assumption on the inputs, an average upper bound depending on input only is given.

The only possible remaining badly-shaped tetrahedra are slivers. A heuristic method which uses the Delaunay refinement to remove slivers is tested. For PLSs satisfying the input angle restriction, the method success-

ful removes all slivers and the number of added Steiner points is at the same order as the number of the initial slivers.

Chapter 4 focuses on a problem in mesh adaptation, i.e., how to efficiently distribute new points in an initial mesh of a domain such that the output mesh has good quality and the mesh size conforms to a desired size distribution on the domain. Two basic issues need to be resolved, namely boundary conformity and field points distribution.

The basic Delaunay refinement scheme of Shewchuk is extended and modified for the above purpose. A new point insertion rule is added for taking a user-defined mesh sizing function into account. A point accepting rule is applied for rejecting new points which may cause no termination. Our analysis shows that these variants guarantee the termination on all initial meshes of a PLS without the input angle limitation. The shapes of output tetrahedra remain good inside the domain. The mesh size can be adapted to a desired size distribution. For a smoothly changing sizing function, the resulting mesh size conforms well to it.

Chapter 5 presents an algorithm for solving the boundary conformity problem to support boundary conforming Delaunay mesh generation. The main issues are to limit the needed number of Steiner points and to improve the output mesh quality. Given a three-dimensional PLS, the algorithm constructs a constrained Delaunay tetrahedralization of it.

We first give a definition for a constrained Delaunay triangulation (abbreviated as CDT) of a PLS in any dimensions. It generalizes the definition given by Lee *et al.* [69] and Chew [23] with a new property that it may contain Steiner points. We then show some basic properties of such objects which are very close to Delaunay triangulations.

Our algorithm for constructing a CDT of a three-dimensional PLS $\mathcal{X}$ is based on a theorem of Shewchuk [100] which guarantees the existence of a CDT without Steiner points. First, the edge set of $\mathcal{X}$ is enriched by adding Steiner points into $\mathcal{X}$. The result is a new PLS $\mathcal{X}'$ such that $|\mathcal{X}'| = |\mathcal{X}|$ and $\mathcal{X}'$ has a CDT without Steiner points. Then the CDT of $\mathcal{X}'$ is constructed by incrementally recovering the facets of $\mathcal{X}'$. The termination and correctness of the algorithm are proven. The complexity of the algorithm is analyzed and discussed. Experimental results are given to illustrate its robustness and efficiency.

Chpater 6 summaries the results achieved in this thesis and highlights some open questions which worth to be investigated in the future.

Chapter 2

Preliminaries

In this chapter we give the mathematical definitions of the input and output objects discussed in this work. The input objects – piecewise linear systems – are defined in Section 2.3. The output objects are triangulations and meshes of the inputs, they are defined in Section 2.4.

Throughout this thesis, some basic concepts from combinatorial topology and convex polytopes are needed. We briefly review some important notions in Section 2.1. Edelsbrunner provides a nice introductory text on combinatorial topology [35]. For a comprehensive discussion of convex polytopes, we refer to the lecture notes of Ziegler [122].

2.1 Basic Notions

We begin with the definition of the space. A *topological space* is a pair $(\mathbb{X}, \mathcal{O})$ where $\mathbb{X}$ is a set and $\mathcal{O} \subseteq 2^{\mathbb{X}}$ is the *topology* on $\mathbb{X}$, that is, a family of subsets of $\mathbb{X}$, called *open sets*, such that

(i) $\emptyset, \mathbb{X} \in \mathcal{O}$,
(ii) $\mathcal{Z} \subseteq \mathcal{O}$ implies $\bigcup \mathcal{Z} \in \mathcal{O}$, and
(iii) $\mathcal{Z} \subseteq \mathcal{O}$ and $\mathcal{Z}$ finite implies $\bigcap \mathcal{Z} \in \mathcal{O}$.

This definition is extremely general. For all discussions in this thesis, we specialize the set $\mathbb{X}$ to be the *d-dimensional Euclidean space*, denoted as $\mathbb{R}^d$. This is the set consisting of all ordered d-tuples of real numbers. Each element $\mathbf{x} \in \mathbb{R}^d$ is called a *point* (or a vertex), i.e., $\mathbf{x} = (x_0, x_1, ..., x_{d-1})$, where each x_k is a *coordinate* of $\mathbf{x}$. We use the Euclidean distance function $\|\cdot\|$ to define an *open d-ball*,

$$\mathbb{B}^d(\mathbf{p}, r) = \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{p}\| < r\},$$

as the set of all points closer than a given distance r from a given point $\mathbf{p}$. The *natural topology* of $\mathbb{R}^d$ is a system of open sets, where each open set is a union of open balls. In the rest of this thesis, we will typically omit the

topology $\mathcal{O}$ from the notation and simply say "$\mathbb{X}$ is a topological space" or "$\mathbb{X}$ is a space". Hence $\mathbb{R}^d$ is a space (with the natural topology).

If $\mathbb{Y} \subseteq \mathbb{X}$ is a subset, for a topological space $(\mathbb{X}, \mathcal{O})$, then $\mathbb{Y}$ is a *subspace* with the *induced* topology, whose open sets are of the form $U \cap \mathbb{Y}$ for $U \in \mathcal{O}$. For example, the *unit d-ball*, $B^d = \{\mathbf{x} \in \mathbb{R}^d \,|\, \|\mathbf{x}\| \leq 1\}$,is a subspace of $\mathbb{R}^d$. Its open sets are the intersections of itself with open sets of $\mathbb{R}^d$. Note that an open set in B^d is not necessarily open in $\mathbb{R}^d$. The spaces encountered in this work are all subspaces of $\mathbb{R}^d$ with the induced natural topology. We will then build special topology on a given subspace.

Let $\mathbb{X}$ and $\mathbb{Y}$ be two topological spaces. A map $f : \mathbb{X} \to \mathbb{Y}$ is *continuous* if the preimage $f^{-1}(U)$ of every open set $U \subseteq \mathbb{Y}$ is open (in $\mathbb{X}$). A *homeomorphism* is a map $f : \mathbb{X} \to \mathbb{Y}$ that is bijective, continuous, and has a continuous inverse. If a homeomorphism exists, then $\mathbb{X}$ and $\mathbb{Y}$ are *homeomorphic*, or $\mathbb{X}$ and $\mathbb{Y}$ are *topologically equivalent*. For example, every $\mathbb{B}^k$ is homeomorphic to $\mathbb{R}^k$ (by $f(\mathbf{x}) = \mathbf{x}/(1 - \|\mathbf{x}\|)$).

A *neighborhood* of a point $\mathbf{x} \in \mathbb{X}$ is an open set of $\mathbb{X}$ that contains $\mathbf{x}$. $\mathbb{X}$ is a *k-manifold* if every $\mathbf{x} \in \mathbb{X}$ has a neighborhood homeomorphic to $\mathbb{R}^k$. For an example, the *k-sphere*, $\mathbb{S}^k = \{\mathbf{x} \in \mathbb{R}^{k+1} \,|\, \|\mathbf{x}\| = 1\}$,is a k-manifold.

A *separation* of a topological space $\mathbb{X}$ is a pair of nonempty, disjoint open subsets $U, V \subset \mathbb{X}$ such that $\mathbb{X} = U \cup V$. We say that $\mathbb{X}$ is *connected* if there is no separation of $\mathbb{X}$.

A point set in $\mathbb{R}^d$ is *convex* if with any two points $\mathbf{v}_i, \mathbf{v}_j$ it also contains the straight line segment $[\mathbf{v}_i, \mathbf{v}_j] = \{(1 - \lambda)\mathbf{v}_i + \lambda\mathbf{v}_j \;:\; 0 \leq \lambda \leq 1\}$. The *convex hull* of a not necessarily convex set V, denoted as $\text{conv}(V)$, is the smallest convex set containing V. Let V be a finite set of $k+1$ points. The *affine hull* of V, $\text{aff}(V)$, is the smallest *affine subspace* (the translate of a linear subspace) that contains V, i.e.,

$$\text{aff}(V) = \left\{ \sum_{i=0}^{k} \lambda_i \mathbf{v}_i \;\middle|\; \mathbf{v}_i \in V, \lambda_i \in \mathbb{R}, \sum_{i=0}^{k} \lambda_i = 1 \right\}.$$

The *dimension* of $\text{aff}(V)$ is the dimension of the corresponding linear subspace. A set of $k+1$ points $\mathbf{v}_0, ..., \mathbf{v}_k \in \mathbb{R}^d$ is *affinely independent* if for a set of coefficients $\lambda_0, ..., \lambda_k \in \mathbb{R}$ with $\sum_{i=0}^{k} \lambda_i = 0$, the only solution for equation $\sum_{i=0}^{k} \lambda_i \mathbf{v}_i = \mathbf{0}$ to be hold is $\lambda_0 = \lambda_1 = ... = \lambda_k = 0$.

A *geometric k-simplex* σ in $\mathbb{R}^d$ is the convex hull of a collection of $k+1$ affinely independent points in $\mathbb{R}^d$. The *dimension* of σ is $\dim(\sigma) = k$. For example, the (-1)-simplex is the empty set ($\emptyset$), a 0-simplex is a vertex (or point), a 1-simplex is an edge, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. A *face* τ of σ is the convex hull of any subset of the vertices of σ. It is again a simplex. $\tau = \emptyset$ and $\tau = \sigma$ are the two trivial faces of σ, all others are *proper* faces of σ. We write $\tau \leq \sigma$ or $\tau < \sigma$ if τ is a face or a

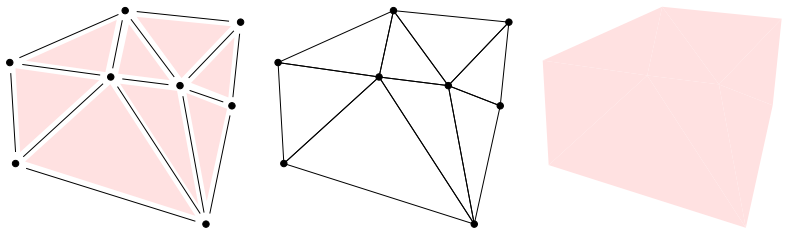


Figure 2.1: Left: A two-dimensional simplicial complex $\mathcal{D}$, which consists of the empty set, 8 vertices, 15 edges and 8 triangles. It is the Delaunay triangulation of $\text{vert}(\mathcal{D})$. Middle: A subcomplex – the 1-skeleton $\mathcal{D}^{(1)}$ of $\mathcal{D}$ (includes $\emptyset$). Right: The underlying space $|\mathcal{D}|$.

proper face of σ. The union of all proper faces of a simplex σ is called the *boundary* $\text{bd}(\sigma)$ of σ. The *interior* $\text{int}(\sigma)$ of σ is $\sigma - \text{bd}(\sigma)$. The interior of a point is the point itself. Note that in geometry the interior is often called the relative interior.

A *simplicial complex* $\mathcal{K}$ in $\mathbb{R}^d$ is a finite collection of simplices, such that

(i) $\sigma \in \mathcal{K}$ and $\tau \leq \sigma \Longrightarrow \tau \in \mathcal{K}$, and

(ii) $\sigma, \tau \in \mathcal{K} \Longrightarrow \sigma \cap \tau \leq \sigma, \tau$.

The first property implies $\emptyset \in \mathcal{K}$. The second property implies that any two different simplices in $\mathcal{K}$ have disjoint interiors, i.e., $\text{int}(\sigma) \cap \text{int}(\tau) = \emptyset$. The *dimension* $\dim(\mathcal{K})$ of $\mathcal{K}$ is the largest dimension of a simplex of $\mathcal{K}$. The *vertex set* $\text{vert}(\mathcal{K})$ of $\mathcal{K}$ is the set of all vertices of $\mathcal{K}$. A *subcomplex* is a subset of $\mathcal{K}$ that is a simplicial complex itself. A particular subcomplex is the *i-skeleton* $\mathcal{K}^{(i)}$ of $\mathcal{K}$, which consists of all simplices $\sigma \in \mathcal{K}$ whose dimension is i or less. Hence $\mathcal{K}^{(0)} = \text{vert}(\mathcal{K}) \cup \emptyset$. The *underlying space* $|\mathcal{K}|$ of $\mathcal{K}$ is the union of all simplices of $\mathcal{K}$, i.e., $|\mathcal{K}| = \bigcup_{\sigma \in \mathcal{K}} \sigma$. See Fig. 2.1 for examples.

$|\mathcal{K}|$ is a topological space with the subspace topology inherited from $\mathbb{R}^d$. Note that the interiors of the simplices partition the underlying space, i.e., each point of $|\mathcal{K}|$ belongs to the interior of exactly one simplex. Let $\pi : \coprod_{\sigma \in \mathcal{K}} \sigma \to \bigcup_{\sigma \in \mathcal{K}} \sigma = |\mathcal{K}|$ be the map from the *disjoint union* of simplices (which are subspaces of $\mathbb{R}^d$) to the union by sending each simplex σ to itself. We obtain a *quotient topology* on $|\mathcal{K}|$ with respect to π, that is, $U \subseteq |\mathcal{K}|$ is open if and only if $\pi^{-1}(U)$ is open in the space $\coprod_{\sigma \in \mathcal{K}} \sigma$ (which has the disjoint union topology). This is the same as saying that $U \subseteq |\mathcal{K}|$ is open (or closed) if and only if its intersection with each simplex of $\mathcal{K}$ is open (or closed). Note that every simplex of $\mathcal{K}$ is closed in $|\mathcal{K}|$.

A *convex polytope* $P \subseteq \mathbb{R}^d$ can be defined either as the convex hull of a finite set of n points, i.e.,

$$P = \text{conv}(V) \quad \text{for some } V \in \mathbb{R}^{d \times n}$$

or, equivalently, as the bounded intersection of finitely many closed half-

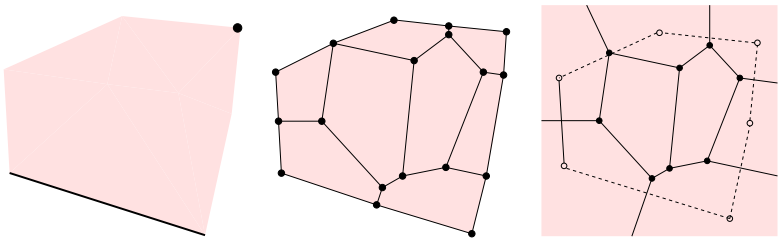


Figure 2.2: Left: A 2-polytope P (a polygon with 6 sides) and two of its faces, an edge (1-polytope) and a vertex (0-polytope), are shown. Middle: A polytopal complex $\mathcal{C}_0$ of dimension 2, containing the empty polytope, 18 vertices (0-polytopes), 25 edges (1-polytopes), and 8 cells (2-polytopes). Right: A polyhedral complex (which is a Voronoi diagram).

spaces in $\mathbb{R}^d$, i.e.,

$$P = P(\mathbf{A}, \mathbf{z}) = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{x} \leq \mathbf{z}\} \quad \text{for some } \mathbf{A} \in \mathbb{R}^{m \times d},\ \mathbf{z} \in \mathbb{R}^m.$$

Here "$\mathbf{A}\mathbf{x} \leq \mathbf{z}$" is the shorthand for a system of inequalities, namely $\mathbf{a}_i^T\mathbf{x} \leq z_i$, for $i = 0, ..., m-1$, where $\mathbf{a}_0^T, ..., \mathbf{a}_{m-1}^T$ are the rows of $\mathbf{A}$. The *dimension* of P is the dimension of its affine hull: $\dim(P) = \dim(\text{aff}(P))$. For example, any d-simplex is a d-polytope, a 0-polytope is a vertex (or point), a 1-polytope is an edge, and a 2-polytope is a polygon.

A *face* F of P is a subset of the form

$$F = P \cap \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^T\mathbf{x} = z_0\},$$

where $\mathbf{a}^T\mathbf{x} \leq z_0$ is a valid inequality of P. F is again a convex polytope whose dimension is $\dim(\text{aff}(F))$, see Fig. 2.2 left. $\emptyset$ and P itself are faces of P. All others are *proper* faces of P. We write $F \leq P$ or $F < P$ if F is a face or a proper face of P. The union of all proper faces of P is the *boundary* $\text{bd}(P)$ of P. The *interior* $\text{int}(P)$ of P is $P - \text{bd}(P)$.

A *polytopal complex* $\mathcal{C}$ in $\mathbb{R}^d$ is a finite collection of convex polytopes, such that it has the same properties as those of a simplicial complex, see Fig. 2.2 middle. A polytopal complex can be viewed as the geometrical realization of a *cell complex* which is a topological space covered by cells, each cell is topologically equivalent to a simplex.

A *convex polyhedron* in $\mathbb{R}^d$ is the intersection of finitely many closed half-spaces in $\mathbb{R}^d$. Note that a polyhedron may be unbounded. A polytope is a bounded polyhedron. For an example, the Voronoi diagram of a set of points is a polyhedral complex, see Fig. 2.2 right.

2.2 Polyhedra and Faces

In this section, we will define a general and therefore not necessarily convex polyhedron and its faces. Such polyhedron will be used as the basic

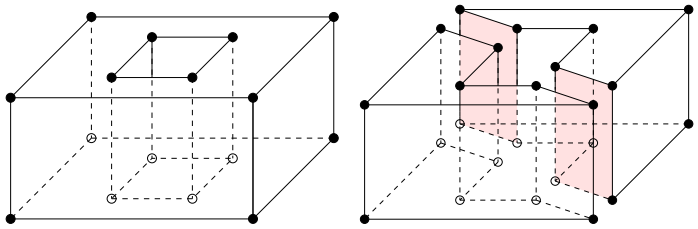


Figure 2.3: Polyhedra and faces. Left: A three-dimensional polyhedron (a torus) formed by the union of four convex polytopes. It consists of 16 vertices (zero-faces), 24 edges (1-faces), 10 two-faces (the faces at top and bottom are not simply connected), and 1 three-face (which is itself). Right: Two three-dimensional polyhedra. Each one has 12 vertices, 18 edges, 8 two-faces, and 1 three-faces. The shaded area highlights two 2-faces whose points have the same face figures.

component of our input objects defined in the next section.

Definition 2.2.1 (Polyhedron) *A polyhedron P in $\mathbb{R}^d$ is the union of a finite set $\mathcal{P}$ of convex polyhedra, i.e., $P = \bigcup_{U\in\mathcal{P}} U$, and the space of P is connected.*

The *dimension* $\dim(P)$ is the largest dimension of a convex polyhedron in $\mathcal{P}$. Note that P may contains holes in its interior. Whatever, we require that the space of P must be connected. See Fig. 2.3 for examples.

There are less definitions about faces of a non-convex polyhedron. The following definition of faces is given by Edelsbrunner [35] with a minor modification in the connectness of the faces.

For a point $\mathbf{x}$ in a polyhedron P we consider a sufficiently small neighborhood $N_\epsilon(\mathbf{x}) = (\mathbf{x} + \mathbb{B}(\mathbf{0}, \epsilon)) \cap P$. The *face figure* of $\mathbf{x}$ is the enlarged version of this neighborhood within P, i.e., $\mathbf{x} + \bigcup_{\lambda>0} \lambda(N_\epsilon(\mathbf{x}) - \mathbf{x})$.

Definition 2.2.2 (Face) *A face F of a polyhedron P is the closure of a maximal connected set of points with identical face figures.*

By this definition, a face of P may contain holes in its interior, but it is always connected. See Fig. 2.3 for examples.

A face F of P is again a polyhedron. Particularly, $\emptyset$ is a face of P. If all convex polyhedra in $\mathcal{P}$ have the same dimension, then P itself is a face of P. All other faces of P are *proper* faces of P. We also write $F \leq P$ or $F < P$ if F is a face or a proper face of P. The faces of dimension 0, 1, $\dim(P) - 2$, and $\dim(P) - 1$ are called *vertices*, *edges*, *ridges*, and *facets*, respectively. The set of all vertices of P, the *vertex set*, will be denoted by $\mathrm{vert}(P)$. The union of all proper faces of P is called the *boundary* of P, denoted as $\mathrm{bd}(P)$. The *interior* $\mathrm{int}(P)$ is $P - \mathrm{bd}(P)$.

Note that a polyhedron may be unbounded. In the scope of this work, we always assume that a polyhedron is bounded, i.e., it is a polytope. In our later discussions, a polytope can be convex or non-convex.

2.3 Piecewise Linear Systems

A *physical domain* Ω in $\mathbb{R}^3$ used for numerical simulation is the volume enclosed by the *boundary* $\partial\Omega$ of Ω. Usually, $\partial\Omega$ may consist of arbitrarily shaped (e.g., curved) edges and surfaces. It is necessary that $\partial\Omega$ includes *internal boundaries* which may separate Ω into sub-domains so that the discontinuity between different materials can be modeled. Moreover, an internal boundary may be *floated*, i.e., it does not separate the domain which contains it. Floated boundaries are useful as well. For instance, the user might want to model a feature line or an iso-surface inside Ω where the numerical solution changes rapidly near it. Hence $\partial\Omega$ is in general not a topological manifold.

A *mesh domain* is an object such that it preserves the topology of Ω and it approximates Ω geometrically. Miller *et al.* [80] introduced a geometric object which uses convex polytopes as the main components. In the following, we define a generalization of this object to represent mesh domains with piecewise linear boundaries.

Definition 2.3.1 (Piecewise Linear System) *A piecewise linear system (abbreviated as PLS) is a finite collection $\mathcal{X}$ of polytopes with the following properties:*

(i) If $P \in \mathcal{X}$, then all faces of P are in $\mathcal{X}$.

(ii) If $P, Q \in \mathcal{X}$, then $P \cap Q \subset \mathcal{X}$.

(iii) If $dim(P \cap Q) = dim(P)$ then $P \subseteq Q$, and $dim(P) < dim(Q)$.

A PLS $\mathcal{X}$ gives certain flexibility in representing non-manifold objects. The condition *(ii)* is relaxed from that of a complex. For example, an edge and a quadrilateral may intersect at a point in their interior as long as that point (a 0-polytope) is included in $\mathcal{X}$. In particular, it implies that $\emptyset \in \mathcal{S}$. Hence, any polytopal complex is a PLS but not vice versa. Since two non-convex polytopes P and Q may intersect at more than one faces of them, $P \cap Q$ is a subcollection of $\mathcal{X}$. Property *(iii)* allows a cube encloses an edge in its interior with no need to further decompose it. Furthermore, this condition excludes the cases which two polytopes have the same dimension and overlap each other. See Fig. 2.4 for an example.

The *dimension* $\dim(\mathcal{X})$ of a PLS $\mathcal{X}$ is the largest dimension of a polytope in $\mathcal{X}$. A *subsystem* of $\mathcal{X}$ is a subset of $\mathcal{X}$ which is again a PLS. A particular subsystem is the *i-skeleton* $\mathcal{X}^{(i)}$ of $\mathcal{X}$ which consists of all polytopes of $\mathcal{X}$

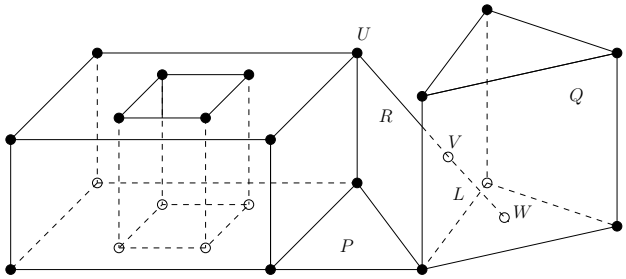


Figure 2.4: A piecewise linear system $\mathcal{X}$ which contains 24 vertices, 37 edges, 16 two-polytopes, and 2 three-polytopes. In this figure, the dimensions of polytopes $Q, P, R, L \in \mathcal{S}$ are $3, 2, 1, 1$, respectively. $U, V, W \in \mathcal{S}$ are vertices, where $V = R \cap L = R \cap Q$, $L = L \cap Q$, and $\dim(L) < \dim(Q)$.

with dimension $\leq i$. For example, $\mathcal{X}^{(0)}$ is the *vertex set* $\mathrm{vert}(\mathcal{X})$ plus $\emptyset$. We define $\partial\mathcal{X} = \mathcal{X}^{(k-1)}$ to be the *boundary system* of $\mathcal{X}$, where $k = \dim(\mathcal{X})$. The *underlying space* $|\mathcal{X}|$ of $\mathcal{X}$ is the union of all polytopes of $\mathcal{X}$. The *interior* of $\mathcal{X}$ is defined as $\mathrm{int}(\mathcal{X}) = |\mathcal{X}| - |\partial\mathcal{X}|$.

Note that in general a PLS $\mathcal{X}$ may not be a complex (although every complex is a PLS). Since the interiors of its polytopes may overlap (due to the properties *(ii)* and *(iii)*), hence they may not partition the underlying space $|\mathcal{X}|$. In the following, we show how to obtain a topology on $|\mathcal{X}|$. We use the concept of "carrier" from convex geometry [12].

Definition 2.3.2 (Carrier) *Let $\mathcal{X}$ be a PLS. The carrier of a point $\boldsymbol{x} \in |\mathcal{X}|$ or a subset $U \subseteq |\mathcal{X}|$ is the minimal polytope of $\mathcal{X}$ containing $\boldsymbol{x}$ or U.*

Equivalently, the carrier of a point is the unique polytope of $\mathcal{X}$ having it in its interior. We define an equivalence relation on $|\mathcal{X}|$ by identifying two points $\mathbf{x}, \mathbf{y} \in |\mathcal{X}|$ if they have the same carrier, denoted as $\mathbf{x} \sim \mathbf{y}$. Let $\mathcal{X}/\sim$ be the collection of all equivalence classes on $|\mathcal{X}|$, i.e., $\mathcal{X}/\sim$ is a partition of $|\mathcal{X}|$. See Fig. 2.5 for an example. Moreover, Each $U \in \mathcal{X}/\sim$ is a subspace of $\mathbb{R}^d$ (with the induced natural topology).

The topology on $|\mathcal{X}|$ can be defined in the following way. Let $\coprod_{U\in\mathcal{X}/\sim} U$ be the disjoint union of all sets of $\mathcal{X}/\sim$ with the disjoint union topology (a set is open in $\coprod_{U\in\mathcal{X}/\sim} U$ if and only if its intersection with each U is open in U), and let

$$\pi : \coprod_{U\in\mathcal{X}/\sim} U \to |\mathcal{X}|$$

be the map that sends each element U to itself in $|\mathcal{X}|$. We give $|\mathcal{X}|$ the quotient topology with respect to π, i.e., a subset of $|\mathcal{X}|$ is open (closed) if and only if its intersection with each set in $\mathcal{X}/\sim$ is open (closed). Later on, when a topology on $|\mathcal{X}|$ is needed, it always refers to this one.

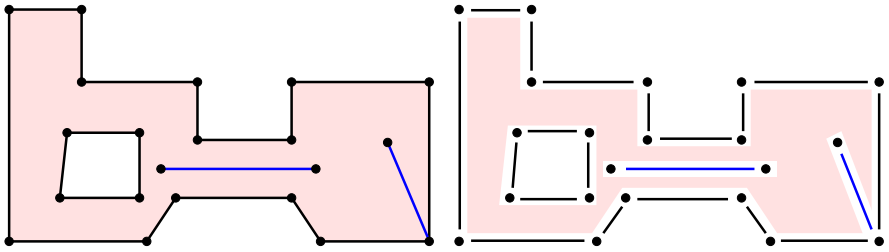


Figure 2.5: Left: A two-dimensionl PLS $\mathcal{X}$. It contains 21 vertices, 20 edges (the edges shown in blue are floating), and 1 two-face (the shaded region). Right: The collection $\mathcal{X}/\sim$ obtained by identifying points in $|\mathcal{X}|$ which have the same carrier. $\mathcal{X}/\sim$ is a partition of $|\mathcal{X}|$.

A mesh domain $\Omega \subset \mathbb{R}^d$ as well as its boundary can be approximated by a PLS $\mathcal{X}$ in $\mathbb{R}^d$. The shape of Ω is approximated by the union of all polytopes of $\mathcal{X}$, and the space of Ω is homeomorphic to $|\mathcal{X}|$.

Every PLS $\mathcal{X}$ can be refined into a convex PLS, e.g., by convex decomposition. Interestingly, when $\mathcal{X}$ has dimension 3 or higher, it is not always possible to do so without using additional points, so-called *Steiner points*[1]. The simplest example is the Schönhardt polyhedron (see Fig. 1.6). It is a three-dimensionl polyhedron P_S which has 6 vertices. However, any convex decomposition of P_S will need at least one additional vertex which is not a vertex of P_S. In our context, a *Steiner point* is a point which was not given initially and must be computed in order to build the desired object, e.g., the convex decomposition of a PLS. It turns out, one of our major problems (in Chapter 5) is how to construct the Steiner points such that they are located at the "best" places in the space of the given PLS.

2.4 Triangulations and Meshes

Triangulations and meshes are commonly used terms in literatures. Depending on the contexts, they may refer to different objects. In this section, we will distinguish two types of triangulations, namely *triangulations of topological spaces* and *triangulations of geometrical objects* (e.g., PLSs). They are related but different objects. Simply saying, both of them associate to a space. The topology of the space is preserved in both of them. While the shape of the space is only captured by the latter.

[1] Jakob Steiner (1796 – 1863), a Switzerland Mathematician who spent almost all of his life in the University of Berlin, described several types of interesting points (see `http://mathworld.wolfram.com/SteinerPoints.html`) which later were called Steiner points. However, none of these Steiner points is related to the one discussed here. It is not clear how this name came about. A closely related type of Steiner points were from the *Steiner tree problem* [62].

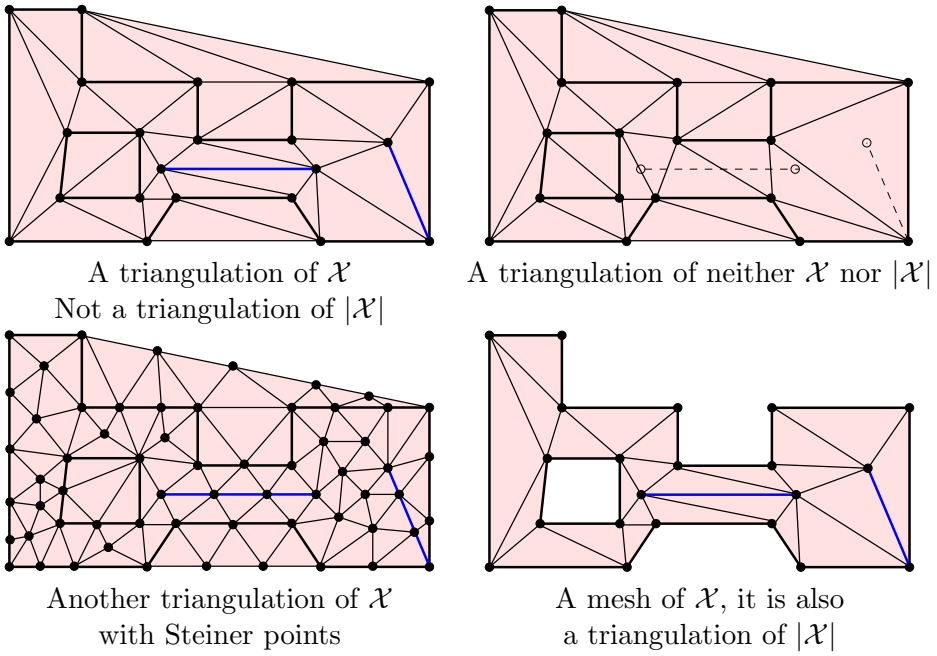


Figure 2.6: Triangulations and meshes. The PLS $\mathcal{X}$ is shown in Fig. 2.5.

Moreover, we require that a triangulation of a geometric object always covers the convex hull of that object (so we can shortly say a "triangulation of a finite set of points" instead of saying "a triangulation of the convex hull of that point set"). A *mesh of a geometrical object* is defined as a triangulation of the space of that object such that it also has exactly the same shape of the object (so a finite set of points is a mesh of itself).

Definition 2.4.1 (Triangulation of Space) *Let $\mathbb{X}$ be a topological space in $\mathbb{R}^d$. A triangulation of $\mathbb{X}$ is a simplicial complex $\mathcal{T}$ whose underlying space is homeomorphic to $\mathbb{X}$.*

By this definition, a triangulation $\mathcal{T}$ of a space $\mathbb{X}$ preserves the topology of $\mathbb{X}$ (by the homeomorphism). It also implies that the topology of $|\mathcal{T}|$ is finer than $\mathbb{X}$'s. Hence every closed set of $\mathbb{X}$ is triangulated into a subcomplex of $\mathcal{T}$. While $|\mathcal{T}|$ and $\mathbb{X}$ may not have the same shape.

Definition 2.4.2 (Triangulation of PLS) *Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. A triangulation of $\mathcal{X}$ is a simplicial complex $\mathcal{T}$ such that*

(i) For each $F \in \mathcal{X}$, there is a subcomplex $\mathcal{K} \subseteq \mathcal{T}$ and $|\mathcal{K}| = F$, and

(ii) $|\mathcal{T}| = conv(|\mathcal{X}|)$.

A triangulation of a PLS always covers the convex hull of its underlying space, and the shape of $\mathcal{X}$ is respected. Property *(i)* implies that the any

simplex $\sigma \in \mathcal{T}$ does not cross $F \in \mathcal{X}$. It can be shown that the topology of $|\mathcal{X}|$ is preserved by $|\mathcal{T}|$. In particular, every vertex of $\mathcal{X}$ must also be a vertex of $\mathcal{T}$. While $\mathcal{T}$ may contain Steiner points. This ensures that every $\mathcal{X}$ can be triangulated for $\dim(\mathcal{X}) = 3$. See Fig. 2.6 for examples.

Definition 2.4.3 (Mesh of PLS) *Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. A mesh of $\mathcal{X}$ is a simplicial complex $\mathcal{T}$ such that*

(i) For each $F \in \mathcal{X}$, there is a subcomplex $\mathcal{K} \subseteq \mathcal{T}$ and $|\mathcal{K}| = F$, and

(ii) $|\mathcal{T}| = |\mathcal{X}|$.

By this definition, if $\mathcal{K}$ is a mesh of $\mathcal{X}$, then the underlying space $|\mathcal{K}|$ preserves both of the topology and the geometry of $|\mathcal{X}|$. It also implies that if $\mathcal{T}$ is a triangulation of $\mathcal{X}$, then $\mathcal{T}$ contains a mesh of $\mathcal{X}$ as a subcomplex. See Fig. 2.6 for examples.

Chapter 3

Boundary Conforming Delaunay Mesh Generation

This chapter studies algorithms for generating boundary conforming Delaunay meshes. Both the mesh quality and mesh size are considered. A simple and efficient technique to generate Delaunay meshes is *Delaunay refinement*. It was introduced by Chew [23] and Ruppert [92] for meshing two-dimensional domains, and later generalized into three dimensions by Dey *et al.* [30], Chew [24], and Shewchuk [99].

Although Delaunay refinement has achieved great development over the past 10 years, our understanding of its basic scheme is still limited. For instances, the proved theoretical guarantee on termination and mesh quality does not match the practical observations; the existing upper bounds on edge length, vertex degree, and mesh size are obviously too large; and the time and space complexity are not clear with respect to a given input. For these reasons, it is necessary to reanalyze the basic scheme and to gain more insight. The discussion of the classical method is important because many powerful algorithms are based on its simple and elegant ideas.

This chapter is organized as follows. We first give the definition of boundary conforming Delaunay meshes of a polyhedral domain (of any dimension) in Section 3.1. Then we briefly describe Shewchuk's Delaunay refinement algorithm in Section 3.2. A detailed reanalysis and new results are found in Section 3.3. Two heuristic variants of the algorithm are presented in Section 3.4. We summarize our results in Section 3.5.

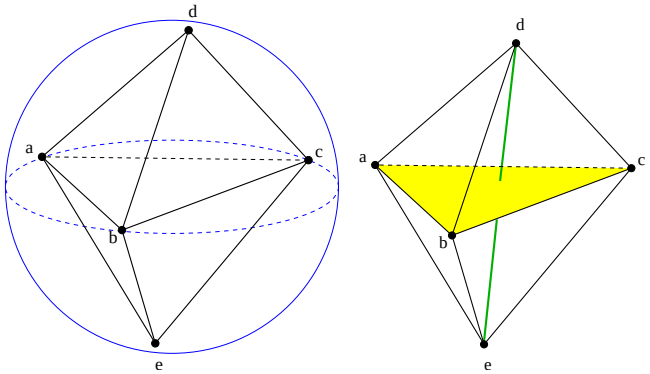


Figure 3.1: A degeneracy. Left: Five points in $\mathbb{R}^3$ lie on a common sphere. Right: A Delaunay triangulation of these five points consists of either two tetrahedra: **abcd**, **abce**, or three tetrahedra: **deab**, **debc**, and **deca**. That is, inside the convex hull of the five points, one can either choose face **abc** (in yellow) or the edge **de** (in green), they are all Delaunay simplices.

3.1 Boundary Conforming Delaunay Meshes

We begin with the definitions of Voronoi diagram and Delaunay triangulations. Let S be a finite set of points in $\mathbb{R}^d$. A *Voronoi cell* V_p of a point $\mathbf{p} \in S$ is defined as:

$$V_p = \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\| \; \forall \mathbf{q} \in S\}.$$

V_p is a convex polyhedron which may not be bounded. The collection $\mathcal{C} = \{V_p \mid \mathbf{p} \in S\}$ gives a partition of the d-dimensional space. A *Voronoi k-face* is the intersection of $d - k + 1$ Voronoi cells in $\mathcal{C}$. The *Voronoi diagram* $\mathrm{Vor}(S)$ of S is the collection of all Voronoi faces [118]. The dual of $\mathrm{Vor}(S)$ is a polytopal complex $\mathcal{D}$ whose vertex set is S and $|\mathcal{D}| = \mathrm{conv}(S)$. If S is in *general position*, i.e., no $d + 2$ vertices of S share a $(d - 1)$-dimensional sphere, then $\mathcal{D}$ is the *Delaunay triangulation* $\mathrm{Del}(S)$ of S [28].

Simplices in the Delaunay triangulation are characterized by the *Delaunay criterion* (known as the "empty sphere" property). A *circumball* of a simplex σ is a d-ball B_σ with all vertices of σ on its boundary. A simplex σ in S is *Delaunay* if there exists a circumball B_σ of σ, such that B_σ is *empty*, i.e., $\mathrm{int}(B_\sigma) \cap S = \emptyset$ [28]. Any simplex in $\mathrm{Del}(S)$ is a Delaunay simplex. However, not every Delaunay simplex of S may appear in $\mathrm{Del}(S)$. In other words, $\mathrm{Del}(S)$ is not unique. See Fig. 3.1 for an example.

Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. It is well known that a Delaunay triangulation of $\mathrm{vert}(\mathcal{X})$ does not necessarily contain every polytope $F \in \mathcal{X}$. A *conforming Delaunay mesh* of $\mathcal{X}$ is defined as below.

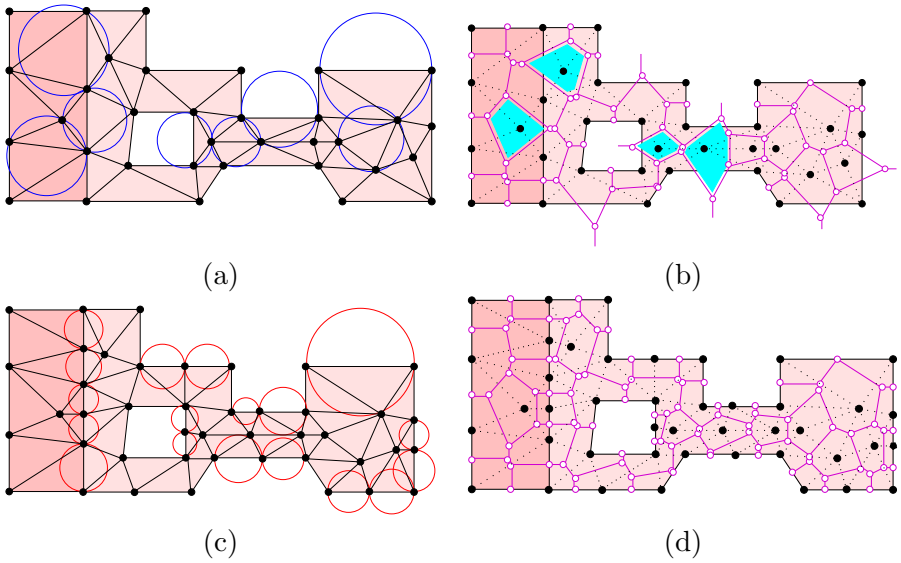


Figure 3.2: (a) A conforming Delaunay mesh $\mathcal{T}$ of a two-dimensional PLS $\mathcal{X}$. $\mathcal{X}$ has two regions (shown in different colors) separated by an internal edge. (b) The Voronoi partition of $\mathcal{X}$ obtained from taking the dual of $\mathcal{T}$. Some Voronoi edges (highlighted in cyan) cross the internal boundaries are not orthogonal. (c) A boundary conforming Delaunay mesh $\mathcal{T}'$ of $\mathcal{X}$. Some diametric spheres of the boundary edges in $\mathcal{T}'$ are shown. (d) The corresponding Voronoi partition of $\mathcal{X}$.

Definition 3.1.1 (Conforming Delaunay Mesh) *A conforming Delaunay mesh of a PLS $\mathcal{X}$ is a mesh $\mathcal{T}$ of $\mathcal{X}$ such that every simplex of $\mathcal{T}$ is Delaunay.*

A conforming Delaunay mesh $\mathcal{T}$ of $\mathcal{X}$ is a subcomplex of a Delaunay triangulation of $\text{vert}(\mathcal{T})$, see Fig. 3.2 (a).

The Voronoi-boxed based finite volume scheme has special requirement for its boundary cells, i.e., a Voronoi edge crossing the boundary must be orthogonal to the boundary faces. However, the Voronoi diagram obtained from a conforming Delaunay mesh may not satisfy this requirement, see Fig. 3.2 (b). The *boundary conforming Delaunay* property is thus introduced to characterize meshes that avoid such cases.

Let $\mathcal{T}$ be a mesh of $\mathcal{X}$ in $\mathbb{R}^d$. Let σ be a simplex in $\mathcal{T}$. If $\dim(\sigma) = d$, then σ has a unique circumball. While σ has infinitely many circumballs if $\dim(\sigma) < d$. The smallest circumball of σ is called the *diametric ball* of σ. For example, if σ is an edge in $\mathbb{R}^3$, then the midpoint and half of the edge length of σ define its diametric ball. The diametric ball of a vertex is itself. A simplex $\sigma \in \mathcal{T}$ is *boundary conforming Delaunay* if the diametric ball B_σ of σ is empty. It is also known as Gabriel property [49].

Definition 3.1.2 (Boundary Conforming Delaunay Mesh) *A boundary conforming Delaunay mesh of a PLS $\mathcal{X}$ is a mesh $\mathcal{T}$ of $\mathcal{X}$ such that*

(i) every simplex of $\mathcal{T}$ is Delaunay, and

(ii) every simplex contained in $|\partial\mathcal{X}|$ is Gabriel.

Fig. 3.2 (c) and (d) illustrate a two-dimensional boundary conforming Delaunay mesh and its Voronoi diagram. A similar definition for this property appeared in both dissertations of Fleischmann [44] and Krause [67] about mesh generation for semiconductor device simulations. Our definition is more concise and it generalizes to any dimension.

Let $\mathcal{X}$ be a two-dimensional PLS and $\mathcal{T}$ is a (triangular) mesh of $\mathcal{X}$. There is an easy criterion to check whether $\mathcal{T}$ is boundary conforming or not: Let $\sigma \in \mathcal{T}$ be an edge contained in $|\partial\mathcal{X}|$, let $\tau \in \mathcal{T}$ be a triangle containing σ. If the angle in τ opposite σ is larger than 90°, then the circumcenter of τ lies outside τ, i.e., σ is not boundary conforming Delaunay, which implies that $\mathcal{T}$ is not boundary conforming Delaunay.

3.2 Delaunay Refinement

In this section, we describe Shewchuk's Delaunay refinement algorithm [99]. It is included for the completion of our analysis.

3.2.1 Tetrahedron Shape Measures

A *tetrahedron shape measure* is a continuous function which evaluates the shape of a tetrahedron by a real number. Various tetrahedron shape measures have been suggested, some of them are equivalent, see [72, 32, 104]. This section describes three tetrahedron shape measures which will be encountered in our analysis of the algorithm.

The most general shape measure for a simplex is the aspect ratio. The *aspect ratio*, $\eta(\tau)$, of a tetrahedron τ is the ratio between the longest edge length l_{max} and the shortest height h_{min}, i.e., $\eta(\tau) = l_{max}/h_{min}$. The aspect ratio measures the "roundness" of a tetrahedron in terms of a value between $\sqrt{2}/\sqrt{3}$ and $+\infty$. Low aspect ratio implies better shape. Other possible definitions of aspect ratio exist, such as the ratio between the circumradius and inradius. These definitions are equivalent in the sense that if a tetrahedron is bounded under one ratio implies a bound on the other.

In Delaunay refinement algorithm, however, a weaker tetrahedron shape measure is used. The *radius-edge ratio*, $\rho(\tau)$ of a tetrahedron τ is the ratio between the radius r of its circumscribed ball and the length l_{min} of its shortest edge, i.e., $\rho(\tau) = r/l_{min}$. The radius-edge ratio $\rho(\tau)$ is at least $\sqrt{6}/4 \approx 0.612$, achieved by the regular tetrahedron. Most of the badly shaped tetrahedra will have a big radius-edge ratio (e.g., > 2.0) except the

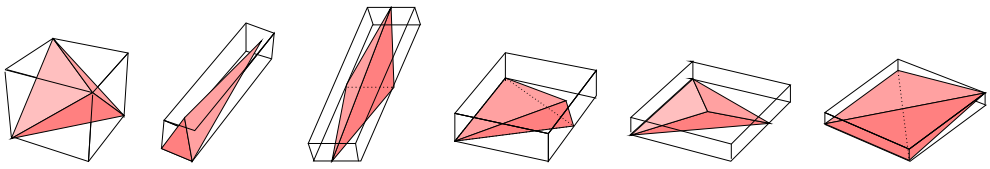


Figure 3.3: Tetrahedra of different shapes classified by Bern *et al.* [8].

	Regular	Needle	Spindle	Wedge	Cap	Sliver
η	0.75	4.74	70.47	7.74	8.83	8.15
ρ	0.612	4.10	26.27	6.74	3.57	0.707
ϕ_{min}	70.53°	60.12°	22.15°	5.77°	15.79°	4.98°

Table 3.1: The aspect ratio (η), radius-edge ratio (ρ), and the minimum dihedral angle (ϕ_{min}) of the tetrahedra shown in Figure 3.3.

sliver, which is a type of very flat tetrahedra having no small edges but nearly zero volume. A sliver can have a minimal value $\sqrt{2}/2 \approx 0.707$. Hence the radius-edge ratio is not equivalent to aspect ratio (due to the slivers). Beside this, the radius-edge ratio is a useful shape measure. It can be shown that if a tetrahedral mesh has a radius-edge ratio bounded for all of its tetrahedra, then the point set of the mesh is well spaced, and each node of the mesh has bounded degree [79].

Each of the six edges of a tetrahedron τ is surrounded by two faces. At a given edge, a *dihedral angle* between two faces is the angle between the intersection of these faces and a plane perpendicular to the edge. A dihedral angle in τ is between 0° and 180°. The minimum dihedral angle $\phi_{min}(\tau)$ of τ is a tetrahedron shape measure.

Figure 3.3 shows six differently shaped tetrahedra classified in [8]. The three shape values of each tetrahedron are reported in Table 3.1.

3.2.2 Point Generating Rules

Let $\mathcal{X}$ be a three-dimensional PLS. Consider a Delaunay tetrahedralization $\mathcal{D}$ of vert($\mathcal{X}$). It may contain many badly-shaped tetrahedra (classified by a tetrahedral shape measure). The basic idea of Delaunay refinement is to choose the circumcenters of the badly-shaped tetrahedra as the candidates, add them into $\mathcal{D}$, update $\mathcal{D}$ with the new points into a new Delaunay tetrahedralization of the augmented point set. The process repeats until no badly-shaped tetrahedra remain in $\mathcal{D}$. The boundary of $\mathcal{X}$ must be preserved in $\mathcal{D}$ as well. This leads to the point generating rules described below.

We call 1- and 2-polytopes of $\mathcal{X}$ *segments* and *facets*. A tetrahedralization $\mathcal{T}$ of $\mathcal{X}$ triangulates every segment and every facet by a subcomplex each. We call 1- and 2-simplices of the subcomplex *subsegments* and *sub-*

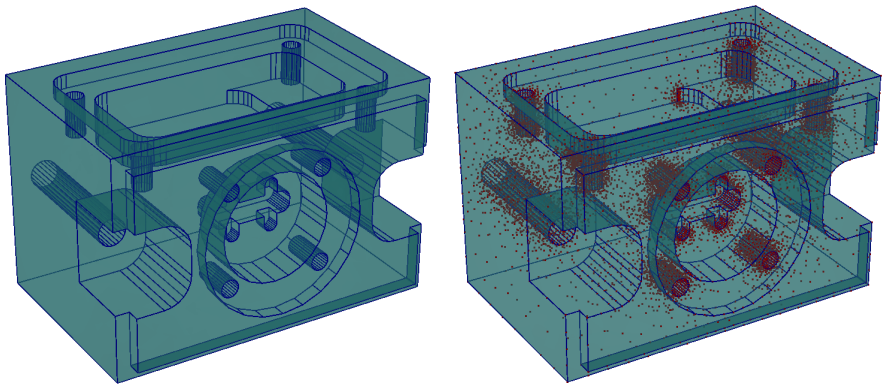


Figure 3.4: A three-dimensional PLS $\mathcal{X}$ (left) representing a mechanical part (`Thepart`) is shown. Right shows the set of new points generated by the Delaunay refinement algorithm.

faces of $\mathcal{T}$ to distinguish them from other simplices of $\mathcal{T}$. Recall that the diametric ball of a subsegment (or a subface) of $\mathcal{T}$ is the smallest circumscribed ball of it. A subsegment (or a subface) σ is said to be *encroached* if its diametric ball B_σ is not empty.

A new point $\mathbf{v}$ is generated by one of the three *point generating rules* described below.

$R1$ If a subsegment is encroached, let $\mathbf{v}$ be its midpoint.

$R2$ If a subface is encroached, let $\mathbf{v}$ be the center of its diametric ball. However, if $\mathbf{v}$ encroaches upon some subsegments, then do not choose $\mathbf{v}$, but use $R1$ to return a $\mathbf{v}$ on one of the encroached subsegments.

$R3$ If a tetrahedron τ has a radius-edge ratio $\rho(\tau) > \rho_0$, let $\mathbf{v}$ be its circumcenter. However, if $\mathbf{v}$ encroaches upon some subsegments or subfaces, then do not choose $\mathbf{v}$, but use $R1$ or $R2$ to return a $\mathbf{v}$ on one of the encroached subsegments or subfaces.

Among these three rules, $R1$ has the highest priority, and $R3$ has the lowest priority, i.e, it is only applied when there are no encroached subsegments and subfaces. In $R3$, if the circumcenter $\mathbf{c}$ of the bad tetrahedron τ encroaches upon both subsegments and subfaces, one should try $R1$ first, if τ still exists after all encroached subsegments are split, then use $R2$. The priorities of the rules are important. They ensure the following facts [99]: (1) If a circumcenter of a badly-shaped tetrahedra does not encroach upon any subface and subsegment, then it must lie inside the mesh domain. (2) If a circumcenter of a subface does not encroach upon any subsegment, then it must lie inside the facet containing this subface.

Unlike the point generation rules, there is no definitive order on point insertion, i.e., given two badly-shaped tetrahedra, they may be split in arbi-

DelaunayRefinement ($\mathcal{X}$, ρ_0)
// $\mathcal{X}$ is a 3D PLS; ρ_0 is a radius-edge ratio bound.
1. initialize a DT $\mathcal{D}$ of vert($\mathcal{X}$);
2. **while** ($\exists$ subsegment or subface σ not in $\mathcal{D}$)
3. **or** ($\exists\ \tau \in \mathcal{D}$ such that $\rho(\tau) > \rho_0$), **do**
4. create a new point $\mathbf{v}$ by rule i, $i \in \{1, 2, 3\}$;
5. update $\mathcal{D}$ to be the DT of vert($\mathcal{D}$) $\cup \{\mathbf{v}\}$;
6. **endwhile**
7. $\mathcal{D} = \mathcal{D} \setminus \{\tau \,|\, \tau \in \mathcal{D}$ and $\tau \nsubseteq |\mathcal{X}|\}$
8. **return** $\mathcal{D}$;

Figure 3.5: The Delaunay refinement algorithm.

trary order. This makes the algorithm simple. However, it has been shown that point insertion order will affect the mesh size, running time, etc [92, 78]. This leaves a big question to the implementation, which order of point insertion will lead to the best performance? We will discuss this issue in the next section.

3.2.3 The Algorithm

The DelaunayRefinement algorithm is summarized in above figure. Given a three-dimensional PLS $\mathcal{X}$ and a radius-edge ratio ρ_0. It first initializes a Delaunay tetrahedralization (DT) $\mathcal{D}$ of vert($\mathcal{X}$). Then it repeatedly adds new points into $\mathcal{D}$ by one of the three *point generating rules*. After a new point $\mathbf{v}$ is added, $\mathcal{D}$ is updated into a DT of vert($\mathcal{D}$)$\cup\{\mathbf{v}\}$ (by either Boywer-Watson [11] or the randomized flip [39] algorithm). The algorithm stops when no rule can be applied. Return $\mathcal{D}$ after removing its tetrahedra outside $|\mathcal{X}|$. Fig. 3.4 illustrates an example of this algorithm.

3.3 Analysis

Ruppert provided a framework to analyze the Delaunay refinement algorithm [92]. Shewchuk extended Ruppert's analysis and showed that under a 90° input angle condition, this algorithm will terminate with a bounded radius-edge ratio of 2.0. The output mesh size is well-graded. Our practical experiments suggest that the 90° angle limitation is too strong. The algorithm usually terminates at a bounded radius-edge ratio much smaller than 2.0. The grading effect of the mesh is different between interior and boundary. It is noticed that the algorithm generally runs efficiently and produces meshes of small size. However, the theoretical time and space complexities of the algorithm are not known yet. The large discrepancies between the theory and practice suggests that a more detailed analysis is necessary.

Our analysis is organized as follows. At first, the termination of this algorithm are reanalyzed in Section 3.3.1. Then we derive new lower bounds on the output edge lengths for the mesh grading effect in Section 3.3.2. The upper bound on mesh quality is improved in Section 3.3.3. A new upper bound on vertex degree is proved in Section 3.3.4. The complexity issues are discussed in Sections 3.3.5 and Sections 3.3.6, respectively.

3.3.1 Proof of Termination

Suppose $\rho_0 > 2$, Shewchuk [99] proved that the Delaunay refinement algorithm terminates if there is no *input angle* (defined below) of the PLS is smaller than 90°. He also showed (in [99] Theorem 28) that this angle condition can be relaxed to 60° on any two incident segments, and 69.3° on any incident segment and facet. But the 90° bound has to be imposed for two incident facets. In this section, we show that a 69.3° bound of the input angle is sufficient for the termination.

Definitions Points are added in a sequence, and for each new point there are predecessors that we can make responsible for the addition. If $\mathbf{v}$ is inserted by $R1$ or $R2$, we define the *parent* $p(\mathbf{v})$ of $\mathbf{v}$ as the encroaching point that causes the insertion of $\mathbf{v}$. The point $p(\mathbf{v})$ may be a Delaunay vertex or a rejected circumcenter. If there are several encroaching points then $p(\mathbf{v})$ is the one closest to $\mathbf{v}$. If $\mathbf{v}$ is inserted by $R3$, let τ be the tetrahedron $\mathbf{v}$ splits, and $\iota < \tau$ is the shortest edge of τ, then $p(\mathbf{v})$ is one of the endpoints of ι which is added more recently. Define the *insertion radius* $r(\mathbf{v})$ of $\mathbf{v}$ to be $r(\mathbf{v}) = \|\mathbf{v} - p(\mathbf{v})\|$.

We define an *input angle* θ of $\mathcal{X}$ as follows: If two segments share a common vertex, θ is the interior angle in $|\mathcal{X}|$, it is a *segment-segment angle*; if a segment and a facet share a single common vertex, then θ is the smallest angle between the segment and a line in the facet, it is a *segment-facet angle*; if two facets are incident at a common segment, then θ is the smallest

interior dihedral angle formed by them, it is a *facet-facet angle*. Let θ_{min} be the smallest angle among all input angles of $\mathcal{X}$.

The *local feature size* [92] is the function lfs : $|\mathcal{X}| \to \mathbb{R}$ that maps any point $\mathbf{x} \in |\mathcal{X}|$ to a positive value, such that lfs($\mathbf{x}$) is the radius of the smallest closed ball centered at $\mathbf{x}$ that intersects at least two disjoint faces of $\mathcal{X}$. It is easy to show that lfs satisfies the Lipschitz condition $\text{lfs}(\mathbf{x}) \leq \text{lfs}(\mathbf{y}) + \|\mathbf{x}-\mathbf{y}\|$.

Let $\mathbf{v}$ be a newly inserted point, and $\mathbf{p} = p(\mathbf{v})$ is its parent. The following lemma relates $r(\mathbf{v})$ and $r(\mathbf{p})$ when no input angle is smaller than 90°.

Lemma 3.3.1 (In [101] Lemma 3) *Let $\boldsymbol{v}$ be a newly inserted mesh vertex, and $\boldsymbol{p} = p(\boldsymbol{v})$. If $\theta_{min} \geq 90°$, then:*

- $r(\boldsymbol{v}) \geq lfs(\boldsymbol{v})$, *if $\boldsymbol{v}$ is an input vertex.*
- $r(\boldsymbol{v}) \geq \frac{1}{\sqrt{2}}\, r(\boldsymbol{p})$, *if $\boldsymbol{v}$ is an $R1$- or $R2$-vertex.*
- $r(\boldsymbol{v}) \geq \rho_0\, r(\boldsymbol{p})$, *if $\boldsymbol{v}$ is an $R3$-vertex.*

The above lemma excludes the cases when $\mathbf{v}$ and $\mathbf{p} = p(\mathbf{v})$ lie on two incident boundaries. In such cases, there must exist an input angle $\theta < 90°$. The next lemma is crucial. It gives the relations between $r(\mathbf{v})$ and $r(\mathbf{p})$ in such cases.

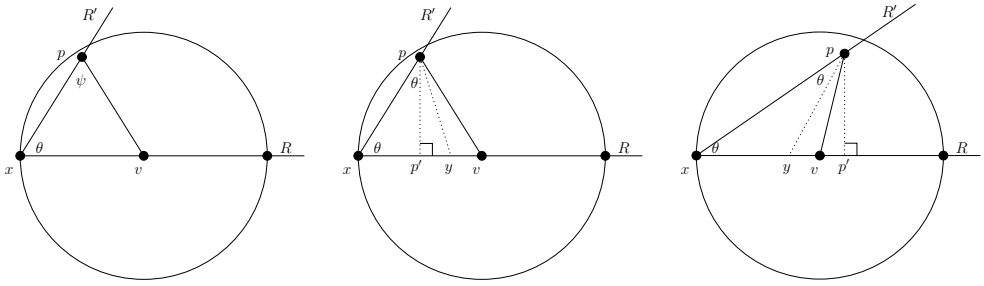
Lemma 3.3.2 *Let $\boldsymbol{v}$ be a newly inserted vertex by $R1$ or $R2$, and $\boldsymbol{p} = p(\boldsymbol{v})$ is an existing Delaunay vertex. Furthermore, $\boldsymbol{v}$ and $\boldsymbol{p}$ lie on two incident boundaries forming an input angle $\theta < 90°$. Then:*

- $r(\boldsymbol{v}) \geq \frac{1}{2\cos\theta}\, r(\boldsymbol{p})$, *when $\theta \geq 45°$ and θ is not a facet-facet angle;*
- $r(\boldsymbol{v}) \geq \frac{1}{2\sqrt{2}\cos\theta}\, r(\boldsymbol{p})$, *when $\theta \geq 54.7°$ and θ is a facet-facet angle;*
- $r(\boldsymbol{v}) \geq \sin\theta\, r(\boldsymbol{p})$, *$\theta < 54.7°$.*

Proof There are four cases classified by the locations of $\mathbf{v}$ and $\mathbf{p}$. *(1)*: both $\mathbf{v}$ and $\mathbf{p}$ lie on segments; *(2)*: $\mathbf{v}$ lies on a segment, $\mathbf{p}$ lies on a facet; *(3)*: $\mathbf{v}$ lies on a facet, $\mathbf{p}$ lies on a segment; and *(4)*: both $\mathbf{v}$ and $\mathbf{p}$ lie on facets. *(1)* has been proved by Pav [90], it is included here since *(2)* and *(3)* are similar to it and we will need the result for the proof of *(4)*.

Let R and R' be two segments incident at a point $\mathbf{x}$ with angle θ between them, $\mathbf{v} \in R$ and $\mathbf{p} \in R'$, see Fig. 3.6 (a). Note that $\mathbf{p}$ is inside the ball $B(\mathbf{v}, r)$, where $r \leq \|\mathbf{v} - \mathbf{x}\|$, implying that $\theta \leq \psi$. We can then draw an isosceles triangle of base angle θ and base edge $\mathbf{xp}$, and let $\mathbf{y}$ be the apex of this triangle, see Fig. 3.6 (b). When $45° \leq \theta$, the apex angle of this isosceles triangle will be acute, so $\|\mathbf{p} - \mathbf{y}\| \leq \|\mathbf{p} - \mathbf{v}\|$. Using the cosine relation it is easy to show that $\|\mathbf{p} - \mathbf{y}\| = \frac{1}{2\cos\theta}\|\mathbf{p} - \mathbf{x}\|$, and thus

$$\|\mathbf{p} - \mathbf{v}\| \geq \frac{1}{2\cos\theta}\, \|\mathbf{p} - \mathbf{x}\| \tag{3.1}$$



(a) the configuration (b) $\theta \geq 45°$ (c) $\theta < 45°$

Figure 3.6: Proof of Lemma 3.3.2. A possible configuration of the points and segments is shown in (a). Since $\mathbf{p}$ is inside the sphere centered at $\mathbf{v}$, so $\theta \leq \psi$. Two auxiliary edges are drawn in both (b) and (c), where the edge $\mathbf{pp'}$ is vertical to the segment R, and the edge $\mathbf{py}$ is one leg of the isosceles triangle with base edge $\mathbf{px}$ and base angle θ.

When $\theta < 45°$, the apex angle may be obtuse (Fig. 3.6 (c)), which makes $\|\mathbf{p}-\mathbf{v}\| < \|\mathbf{p}-\mathbf{y}\|$. However,

$$\|\mathbf{p}-\mathbf{v}\| \geq \sin\theta \, \|\mathbf{p}-\mathbf{x}\|. \tag{3.2}$$

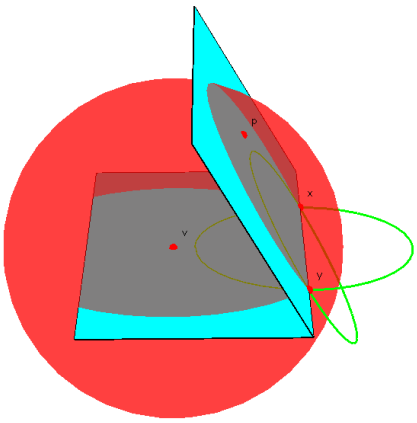
By the same construction, we can show that the inequalities (3.1) and (3.2) also hold in the cases *(2)* and *(3)*.

Now we consider case *(4)*. Let F and F' be two facets incident at a segment R with a dihedral angle θ between them, $\mathbf{v} \in F$ and $\mathbf{p} \in F'$. Fig. 3.7 (a) shows a possible configuration. $\mathbf{xy}$ is a subsegment of R, both $\mathbf{v}$ and $\mathbf{p}$ lie outside the diametric ball $B_{\mathbf{xy}}$ of $\mathbf{xy}$, $\mathbf{p}$ lies inside the ball $B(\mathbf{v}, r)$, where $r \leq \min\{\|\mathbf{v}-\mathbf{x}\|, \|\mathbf{v}-\mathbf{y}\|\}$. Let $\mathbf{p'}$ be the projection of $\mathbf{p}$ onto R, we add two auxiliary edges to our configuration, $\mathbf{pp'}$ and $\mathbf{vp'}$, see Fig. 3.7 (b), and let $\theta' = \angle\mathbf{vp'p}$ be the angle formed by the two added edges. It can be shown that $\theta \leq \theta'$ (see a proof in [20] Lemma 2.1).

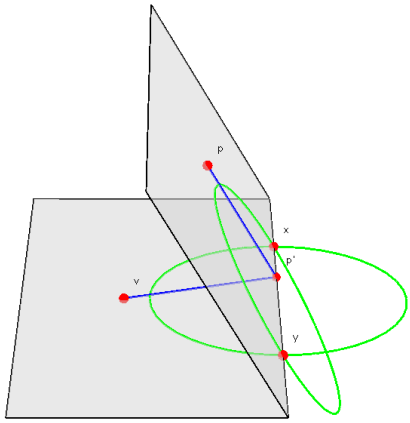
The proof of *(1)* applies to the edges $\mathbf{pv}$ and $\mathbf{pp'}$, that is, if $45° \leq \theta'$, $\|\mathbf{p}-\mathbf{v}\| \geq \frac{1}{2\cos\theta'} \|\mathbf{p}-\mathbf{p'}\|$, else, $\|\mathbf{p}-\mathbf{v}\| \geq \sin\theta' \, \|\mathbf{p}-\mathbf{p'}\|$. Since $\theta \leq \theta'$, we can replace θ' by θ and the inequalities do not change, i.e., if $45° \leq \theta'$,

$$\|\mathbf{p}-\mathbf{v}\| \geq \frac{1}{2\cos\theta} \|\mathbf{p}-\mathbf{p'}\|. \tag{3.3}$$

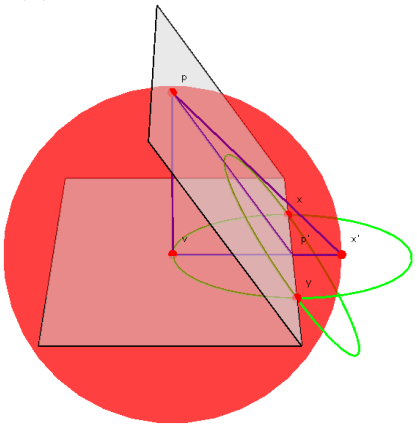
Note that the inequality (3.3) may not hold for $\theta = 45°$ (or slightly larger than it). The reason is that $\mathbf{p'}$ is not a mesh vertex and it is possible that $\|\mathbf{v}-\mathbf{p'}\| < r \leq \|\mathbf{v}-\mathbf{x}\|$, refer to Fig. 3.6 (c). Hence the triangle $\mathbf{vpp'}$ may be obtuse at $\mathbf{v}$ when $\theta = 45°$ and $\mathbf{p}$ is still inside the ball $B(\mathbf{v}, r)$. An extreme case which guarantees that the angle at $\mathbf{v}$ is non-obtuse is shown in Fig. 3.7 (c). In this case, $\mathbf{v}$ lies strictly on both the diametric sphere and the bisector of $\mathbf{xy}$, $\mathbf{p}$ lies strictly on the ball centered at $\mathbf{v}$ with $r = \|\mathbf{v}-\mathbf{p}\| =$



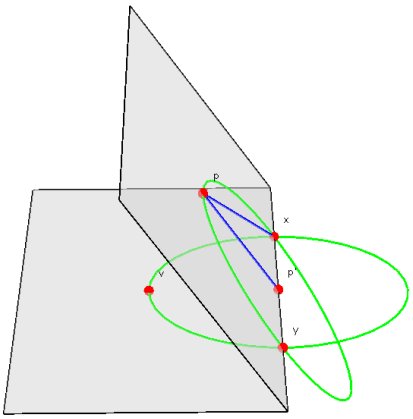
(a) The general configuration



(b) Two auxiliary edges $\mathbf{pp}'$ and $\mathbf{vp}'$.



(c) The case when $\theta \approx 54.7°$



(d) The case when $\theta \approx 69.3°$.

Figure 3.7: Proof of Lemma 3.3.2. **(a)** F and F' are two incident facets forming a dihedral angle $\theta < 90°$. $\mathbf{v} \in F$, $\mathbf{p} \in F'$, and $\mathbf{xy} \subset F \cap F'$ is a subsegment. The diametric ball $B_{\mathbf{xy}}$ of $\mathbf{xy}$ is shown in green. The ball $B(\mathbf{v}, r)$, $r \leq \min\{\|\mathbf{v}-\mathbf{x}\|, \|\mathbf{v}-\mathbf{y}\|\}$, is shown in red. Both $\mathbf{p}$ and $\mathbf{v}$ lie outside $B_{\mathbf{xy}}$, $\mathbf{p}$ is inside $B(\mathbf{v}, r)$. **(b)** $\mathbf{p}'$ is the orthogonal projection of $\mathbf{p}$ onto $\mathbf{xy}$. $\theta' = \angle\mathbf{vp}'\mathbf{p}$, and it must be $\theta \leq \theta'$. **(c)** The first extreme case: $\mathbf{p}$ is on $B(\mathbf{v}, r)$, and $\mathbf{v}$ is the orthogonal projection of $\mathbf{p}$ onto F. Hence $\theta = \arctan\sqrt{2} \approx 54.7°$. **(d)** The second extreme case: $\mathbf{p}$ is on $B_{\mathbf{xy}}$, $\|\mathbf{p}-\mathbf{p}'\|$ reaches its minimum. Hence $\|\mathbf{p}-\mathbf{p}'\| = \|\mathbf{p}-\mathbf{x}\|/\sqrt{2}$, and $\theta = \arccos\frac{1}{2\sqrt{2}} \approx 69.3°$.

$\|\mathbf{v}-\mathbf{x}'\|$, i.e., the angle formed by edges $\mathbf{vx}'$ and $\mathbf{px}'$ is 45°. Hence we have $\theta = \arctan\sqrt{2} \approx 54.7°$. So the inequality (3.3) holds if $54.7° \leq \theta$.

The inequality (3.3) only bounds the distance $\|\mathbf{p}-\mathbf{p}'\|$ which may be smaller than $\|\mathbf{p}-\mathbf{x}\|$. However, $\|\mathbf{p}-\mathbf{p}'\|$ will be not much smaller than $\|\mathbf{p}-\mathbf{x}\|$. The extreme configuration is shown in Fig. 3.7 (d), where $\|\mathbf{p}-\mathbf{p}'\| = \frac{1}{\sqrt{2}}\|\mathbf{p}-\mathbf{x}\|$. Hence, if $54.7° \leq \theta$, the following inequality holds.

$$\|\mathbf{p}-\mathbf{v}\| \geq \frac{1}{2\sqrt{2}\cos\theta}\|\mathbf{p}-\mathbf{x}\|. \tag{3.4}$$

Finally, note that $r(\mathbf{v}) = \|\mathbf{p}-\mathbf{v}\|$, and $r(\mathbf{p}) \leq \|\mathbf{p}-\mathbf{x}\|$, the lemma is proved by replacing the corresponding terms in the (3.1), (3.2), and (3.4). ■

Lemma 3.3.1 and Lemma 3.3.2 together describe the relation between the insertion radii of a newly inserted vertex and its parent. Before we prove the termination, a restriction on the input angle is needed.

Assumption 3.3.3 (Input Angle) *Let $\mathcal{X}$ be a three-dimensional PLS. Assume that $\mathcal{X}$ contains no segment-segment angle and segment-facet angle smaller than 60°, and no facet-facet angle smaller than* $\arccos\frac{1}{2\sqrt{2}} \approx 69.3°$.

Theorem 3.3.4 *If $\mathcal{X}$ satisfies the Input Angle Assumption and $\rho_0 \geq 2$, then the Delaunay refinement algorithm terminates.*

Proof Let $\text{lfs}_{min} = \min\{\text{lfs}(\mathbf{x}) \mid \mathbf{x} \in |\mathcal{X}|\}$. Our goal is to show that no output edge of this algorithm will have length shorter than lfs_{min}.

Suppose for the sake of contradiction that the algorithm introduces an edge shorter than lfs_{min} into the mesh. Let $\mathbf{xy}$ be the first such edge introduced. Clearly, $\mathbf{x}$ and $\mathbf{y}$ cannot both be input vertices, nor can they lie on non-incident boundaries. Assume $\mathbf{x}$ was inserted after $\mathbf{y}$. By assumption, no edge shorter than lfs_{min} exists before $\mathbf{x}$ was inserted, i.e., for any existing vertex $\mathbf{q}$, $r(\mathbf{q}) \geq \text{lfs}_{min}$. Now let $\mathbf{x}$ be an inserted vertex, and $\mathbf{p} = p(\mathbf{x})$, we consider the following cases:

- If $\mathbf{x}$ is an $R3$-vertex, then $r(\mathbf{x}) \geq \rho_0\, r(\mathbf{p}) \geq \text{lfs}_{min}$.
- If $\mathbf{x}$ is an $R1$-vertex, and $\mathbf{p}$ is a rejected circumcenter of a bad tetrahedron τ. Let $\mathbf{g} = p(\mathbf{p})$, then $r(\mathbf{x}) \geq \frac{1}{\sqrt{2}} r(\mathbf{p}) \geq \frac{\rho_0}{\sqrt{2}} r(\mathbf{g}) \geq \text{lfs}_{min}$.
- If $\mathbf{x}$ is an $R1$-vertex, and $\mathbf{p}$ is a rejected circumcenter of a an encroached subface σ, and σ is encroached by a rejected circumcenter of a bad tetrahedron τ. Let $\mathbf{g} = p(\mathbf{p})$, and $\mathbf{e} = p(\mathbf{g})$, then $r(\mathbf{x}) \geq \frac{1}{\sqrt{2}} r(\mathbf{p}) \geq \frac{1}{2} r(\mathbf{g}) \geq \frac{\rho_0}{2} r(\mathbf{e}) \geq \text{lfs}_{min}$.
- If $\mathbf{x}$ is an $R1$-vertex, and $\mathbf{p}$ lies on an incident segment or facet. Then $r(\mathbf{x}) \geq \frac{1}{2cos\theta}\, r(\mathbf{p}) \geq \text{lfs}_{min}$.

- If $\mathbf{x}$ is an $R2$-vertex, and $\mathbf{p}$ is a rejected circumcenter of a bad tetrahedron t. Let $\mathbf{g} = p(\mathbf{p})$, then $r(\mathbf{x}) \geq \frac{1}{\sqrt{2}} r(\mathbf{p}) \geq \frac{\rho_0}{\sqrt{2}} r(\mathbf{g}) \geq \text{lfs}_{min}$.
- If $\mathbf{x}$ is an $R2$-vertex, and $\mathbf{p}$ lies on an incident facet. Then $r(\mathbf{x}) \geq \frac{1}{2\sqrt{2}cos\theta}\, r(\mathbf{p}) \geq \text{lfs}_{min}$.

We have shown that $r(\mathbf{x}) \geq \text{lfs}_{min}$ in all cases, contradicting the assumption that $\mathbf{xy}$ has length shorter than lfs_{min}. No edge shorter than lfs_{min} is ever introduced, hence the algorithm must terminate. ■

3.3.2 Output Edge Lengths

It is first proved by Ruppert [92] that the Delaunay refinement results a *well-graded* triangular mesh, i.e., each output edge has a length which is provably not smaller than a constant factor times the local feature size of its endpoints. Shewchuk [101] used Ruppert's technique to show that the 3D Delaunay refinement algorithm results well-graded tetrahedral meshes as well. He provided three constants, one for each rule, which are

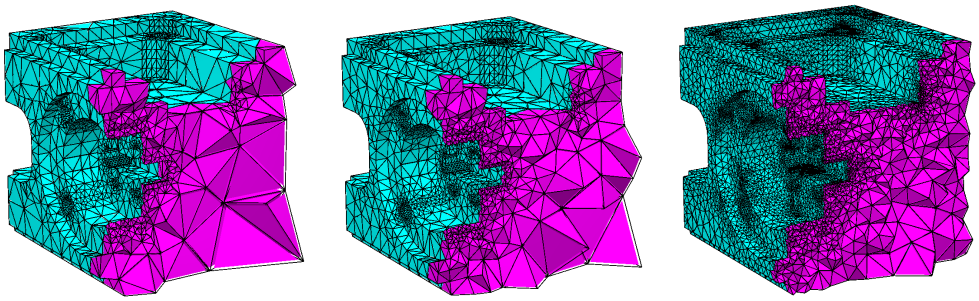
$$D_1 = \frac{(3+\sqrt{2})\rho_0}{\rho_0 - 2}, \quad D_2 = \frac{(1+\sqrt{2})\rho_0 + \sqrt{2}}{\rho_0 - 2}, \quad D_3 = \frac{\rho_0 + 1 + \sqrt{2}}{\rho_0 - 2}.$$

For any vertex $\mathbf{v}$ inserted by the i-th rule, $i \in \{1, 2, 3\}$, $r(\mathbf{v}) \geq \frac{\text{lfs}(\mathbf{v})}{D_i}$ holds. Because D_1 is the largest of the three constants, we can simplify the invariant to $r(\mathbf{v}) \geq \frac{\text{lfs}(\mathbf{v})}{D_1}$ for every Delaunay vertex.

Theorem 3.3.5 ([92, 101]) *Let $\boldsymbol{v}$ and $\boldsymbol{w}$ be any two output mesh vertices. If $\rho_0 > 2$, then $\|\boldsymbol{v} - \boldsymbol{w}\| \geq \frac{lfs(\boldsymbol{v})}{D_1+1}$.* ■

The above theorem guarantees that the spacing of the output mesh vertices is separated by the local feature sizes of the vertices. Clearly the theorem does not hold if $\rho_0 \leq 2$. In practice, however, we observe well-graded meshes for a much smaller ρ_0 (even if ρ_0 is very close to 1) (see Fig. 3.8) though the mesh size is increasing dramatically as ρ_0 approaches to 1. In the following, we analyze the relation of a new vertex $\mathbf{v}$ and $r(\mathbf{v})$.

Definitions For any newly inserted vertex $\mathbf{v}$, we can form a *sequence of parents* (SoP), $\{\mathbf{v} = \mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_m, \mathbf{v}_{m+1} = \mathbf{w}\}$, where for $1 \leq i \leq m$, $\mathbf{v}_i$ was inserted by $R3$, $\mathbf{v}_{i+1}$ is the parent of $\mathbf{v}_i$, and $\mathbf{w}$ is called the *grandparent* of $\mathbf{v}$. It is either an input vertex or a vertex inserted by $R1$ or $R2$ ($\mathbf{w}$ is a boundary vertex). m denotes the number of $R3$-vertices in this sequence. A trivial sequence ($m = 0$) has at least two elements, namely, $\mathbf{v}$ and $\mathbf{w}$. Given a SoP of $\mathbf{v}$, it can be extended to a *maximum* sequence of parents whose



$\rho_0 = 2.0$ $\rho_0 = 1.4$ $\rho_0 = 1.1$

Figure 3.8: The resulting meshes with different ρ_0. From left to the right, the number of mesh nodes are: 7884, 15044, 42686. In our tests, the algorithm still terminates when $\rho_0 = 1.04$, and results a mesh of 1700906 nodes and 9451816 tetrahedra.

last element $\mathbf{w}$ is either a segment midpoint or an input point. We call such a $\mathbf{w}$ the *ancestor* of $\mathbf{v}$.

Remark One has to be careful, if $\mathbf{v}$ is an $R1$- or $R2$-vertex, $\mathbf{v}_1$ may not be the parent of $\mathbf{v}$, although there are special cases in which $\mathbf{v}_1$ is the parent of $\mathbf{v}$. For instance, $\mathbf{v}$ is an $R2$-vertex, $\mathbf{v}_1$ is an $R1$-vertex which encroaches upon the subface whose circumcenter is $\mathbf{v}_1$. However, such special cases do not influence the derived bounds in below.

Lemma 3.3.6 *Let $\boldsymbol{v}$ be a newly inserted vertex, $\boldsymbol{w}$ be its grandparent, and $\rho_0 > 1$. Let $x_1 = \frac{2}{\rho_0}$, $x_2 = \frac{\sqrt{2}}{\rho_0}$, and $x_3 = \frac{1}{\rho_0}$. If $\rho_0 > 1$.*

(c1) $r(\boldsymbol{v}) \geq x_1^{-1}\rho_0^m\, r(\boldsymbol{w})$, *and* $\|\boldsymbol{v} - \boldsymbol{w}\| \leq \left(1 + \sqrt{2} + 2\, C_1\right) r(\boldsymbol{v})$, *if $\boldsymbol{v}$ is an $R1$-vertex.*

(c2) $r(\boldsymbol{v}) \geq x_2^{-1}\rho_0^m\, r(\boldsymbol{w})$, *and* $\|\boldsymbol{v} - \boldsymbol{w}\| \leq \left(1 + \sqrt{2}\, C_1\right) r(\boldsymbol{v})$, *if $\boldsymbol{v}$ is an $R2$-vertex.*

(c3) $r(\boldsymbol{v}) \geq x_3^{-1}\rho_0^m\, r(\boldsymbol{w})$, *and* $\|\boldsymbol{v} - \boldsymbol{w}\| \leq C_1\, r(\boldsymbol{v})$, *if $\boldsymbol{v}$ is an $R3$-vertex, and*

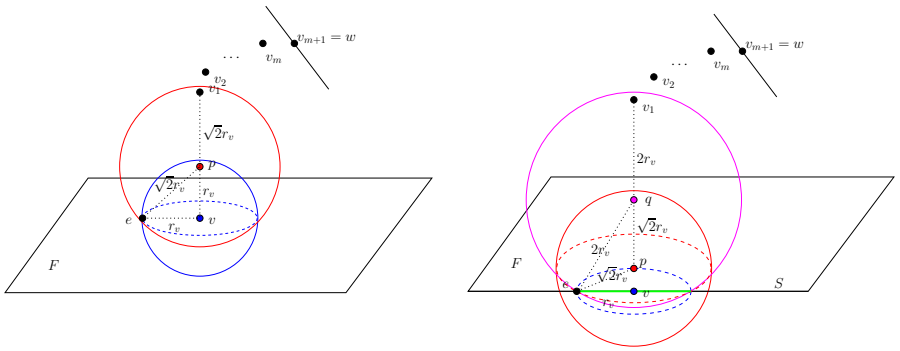
(c3') $lfs(\boldsymbol{v}) \leq C_1\, r(\boldsymbol{v})$, *if $\boldsymbol{w}$ is an input vertex.*

m is the number of $R3$-vertices between $\boldsymbol{v}$ and $\boldsymbol{w}$. $C_1 = \frac{\rho_0}{\rho_0 - 1}$.

Proof Consider the SoP of $\mathbf{v}$, $\{\mathbf{v}_i\}_{i=0}^{m+1}$.

We first prove *(c3)*. Let $\mathbf{v}$ be an $R3$-vertex. Lemma 3.3.1 gives the relations between $r(\mathbf{v}_i)$ and $r(\mathbf{v}_{i+1})$, which are,

$$\begin{array}{rcl} & & r(\mathbf{v}_0) \geq \rho_0\, r(\mathbf{v}_1), \\ r(\mathbf{v}_1) \geq \rho_0\, r(\mathbf{v}_2) & \Longrightarrow & r(\mathbf{v}_0) \geq \rho_0^2\, r(\mathbf{v}_2), \\ & \cdots & \\ r(\mathbf{v}_{m-1}) \geq \rho_0\, r(\mathbf{v}_m) & \Longrightarrow & r(\mathbf{v}_0) \geq \rho_0^m\, r(\mathbf{v}_m). \end{array}$$



(a) (b)

Figure 3.9: Proof of Lemma 3.3.6. (a): $\mathbf{v}$ is an $R2$-vertex on facet F, $\mathbf{v}_1 = p(\mathbf{p})$, $\mathbf{p} = p(\mathbf{v})$, $\mathbf{p}$ is the center of a circumsphere (shown in red) of a bad quality tetrahedron with $\mathbf{v}_1$ as a vertex. $\mathbf{p}$ is rejected since it encroaches upon the circumsphere (shown in blue) of a subface on F centered at $\mathbf{v}$. This figure also shows that $\|\mathbf{v} - \mathbf{v}_1\| \leq (1 + \sqrt{2})r(\mathbf{v})$. (b) $\mathbf{v}$ is an $R1$-vertex on a segment S incident with F. $\mathbf{v}_1 = p(\mathbf{q})$, $\mathbf{q} = p(\mathbf{p})$, and $\mathbf{p} = p(\mathbf{v})$. $\mathbf{q}$ is rejected since it encroaches upon the subface centered at $\mathbf{p}$ on F (shown in red), and $\mathbf{p}$ is rejected too since it encroached upon the subsegment centered at $\mathbf{v}$ (shown in blue). It can be shown that $\|\mathbf{v} - \mathbf{v}_1\| \leq (3 + \sqrt{2})r(\mathbf{v})$.

Since $\mathbf{w} = \mathbf{v}_{m+1}$, we have:

$$r(\mathbf{v}) \geq \rho_0 \, \rho_0^m \, r(\mathbf{w}) = x_3^{-1} \, \rho_0^m \, r(\mathbf{w}).$$

Next we estimate $\|\mathbf{v} - \mathbf{w}\|$ by the total length in the SoP,

$$\begin{aligned} \|\mathbf{v} - \mathbf{w}\| &\leq \|\mathbf{v}_1 - \mathbf{v}_0\| + \|\mathbf{v}_2 - \mathbf{v}_1\| + \cdots + \|\mathbf{v}_{m+1} - \mathbf{v}_m\|, \\ &= r(\mathbf{v}_0) + r(\mathbf{v}_1) + \cdots + r(\mathbf{v}_m), \\ &\leq \left(1 + x_3 + x_3^2 + \cdots + x_3^m\right) \, r(\mathbf{v}), \\ &\leq \frac{\rho_0}{\rho_0 - 1} \, r(\mathbf{v}) \\ &= C_1 \, r(\mathbf{v}). \end{aligned}$$

If $\mathbf{w}$ is an input vertex, by Lemma 3.3.1, $r(\mathbf{w}) \geq \text{lfs}(\mathbf{w})$. Together with *(c3)*, *(c3')* is proved by,

$$\begin{aligned} \text{lfs}(\mathbf{v}) &\leq \|\mathbf{v} - \mathbf{w}\| + \text{lfs}(\mathbf{w}), \\ &\leq (r(\mathbf{v}_0) + r(\mathbf{v}_1) + \cdots + r(\mathbf{v}_m)) + r(\mathbf{w}), \\ &\leq \left(1 + x_3 + x_3^2 + \cdots + x_3^m + x_3^{m+1}\right) \, r(\mathbf{v}), \\ &\leq C_1 \, r(\mathbf{v}). \end{aligned}$$

Proof of *(c2)*. If $m = 0$, $\mathbf{w} = p(\mathbf{v})$. By Lemma 3.3.1, our claim holds. Now assume $m \geq 1$ and $\mathbf{v}_1 \neq p(\mathbf{v})$. There must be a rejected vertex $\mathbf{p}$, such

that $\mathbf{p} = p(\mathbf{v})$, and $\mathbf{v}_1 = p(\mathbf{p})$ (see Fig. 3.9 (a)). By Lemma 3.3.1:

$$\left.\begin{array}{rcl} r(\mathbf{v}_0) & \geq & \frac{1}{\sqrt{2}}\, r(\mathbf{p}), \\ r(\mathbf{p}) & \geq & \rho_0\, r(\mathbf{v}_1) \end{array}\right\} \Longrightarrow r(\mathbf{v}_0) \geq \tfrac{1}{\sqrt{2}}\, \rho_0\, r(\mathbf{v}_1),$$
$$r(\mathbf{v}_1) \geq \rho_0\, r(\mathbf{v}_2) \implies r(\mathbf{v}_0) \geq \tfrac{1}{\sqrt{2}}\, \rho_0^2\, r(\mathbf{v}_2),$$
$$\cdots$$
$$r(\mathbf{v}_{m-1}) \geq \rho_0\, r(\mathbf{v}_m) \implies r(\mathbf{v}_0) \geq \tfrac{1}{\sqrt{2}}\, \rho_0^m\, r(\mathbf{v}_m).$$

Since $\mathbf{w} = \mathbf{v}_{m+1}$, we have:

$$r(\mathbf{v}) \geq \frac{\rho_0}{\sqrt{2}}\, \rho_0^m\, r(\mathbf{w}) = x_2^{-1}\, \rho_0^m\, r(\mathbf{w}).$$

Next we estimate $\|\mathbf{v} - \mathbf{w}\|$ by the total length in the SoP,

$$\begin{aligned} \|\mathbf{v} - \mathbf{w}\| &\leq (\|\mathbf{p} - \mathbf{v}_0\| + \|\mathbf{v}_1 - \mathbf{p}\|) + \|\mathbf{v}_2 - \mathbf{v}_1\| + \cdots + \|\mathbf{v}_{m+1} - \mathbf{v}_m\|, \\ &\leq (r(\mathbf{v}_0) + \sqrt{2}\, r(\mathbf{v}_0)) + r(\mathbf{v}_1) + \cdots + r(\mathbf{v}_m), \\ &\leq r(\mathbf{v}_0) + \sqrt{2}\left(1 + x_3 + x_3^2 + \cdots + x_3^m\right)\, r(\mathbf{v}_0), \\ &\leq \left(1 + \sqrt{2}\, C_1\right)\, r(\mathbf{v}). \end{aligned}$$

Proof of *(c1)*. If $m = 0$, $\mathbf{w} = p(\mathbf{v})$. By Lemma 3.3.1, our claim holds. Now assume $m \geq 1$ and $\mathbf{v}_1 \neq p(\mathbf{v})$. In the worst case, there will be two rejected vertices $\mathbf{p}$, and $\mathbf{q}$, such that $\mathbf{p} = p(\mathbf{v})$, $\mathbf{q} = \mathbf{p}$, and $\mathbf{v}_1 = \mathbf{q}$ (see Fig. 3.9 (b)). By Lemma 3.3.1:

$$\left.\begin{array}{rcl} r(\mathbf{v}_0) & \geq & \frac{1}{\sqrt{2}}\, r(\mathbf{p}), \\ r(\mathbf{p}) & \geq & \frac{1}{\sqrt{2}}\, r(\mathbf{q}), \\ r(\mathbf{q}) & \geq & \rho_0\, r(\mathbf{v}_1) \end{array}\right\} \Longrightarrow r(\mathbf{v}_0) \geq \tfrac{1}{2}\, \rho_0\, r(\mathbf{v}_1).$$
$$r(\mathbf{v}_1) \geq \rho_0\, r(\mathbf{v}_2) \implies r(\mathbf{v}_0) \geq \tfrac{1}{2}\, \rho_0^2\, r(\mathbf{v}_2),$$
$$\cdots$$
$$r(\mathbf{v}_{m-1}) \geq \rho_0\, r(\mathbf{v}_m) \implies r(\mathbf{v}_0) \geq \tfrac{1}{2}\, \rho_0^m\, r(\mathbf{v}_m).$$

Since $\mathbf{w} = \mathbf{v}_{m+1}$, we have:

$$r(\mathbf{v}) \geq \frac{\rho_0}{2}\, \rho_0^m\, r(\mathbf{w}) = x_1^{-1}\, \rho_0^m\, r(\mathbf{w}).$$

Next we estimate $\|\mathbf{v} - \mathbf{w}\|$ by the total length in the SoP,

$$\begin{aligned} \|\mathbf{v} - \mathbf{w}\| &\leq (\|\mathbf{p} - \mathbf{v}_0\| + \|\mathbf{q} - \mathbf{p}\| + \|\mathbf{v}_1 - \mathbf{q}\|) + \|\mathbf{v}_2 - \mathbf{v}_1\| + \cdots + \|\mathbf{v}_{m+1} - \mathbf{v}_m\|, \\ &\leq (r(\mathbf{v}_0) + \sqrt{2}\, r(\mathbf{v}_0) + 2\, r(\mathbf{v}_0)) + r(\mathbf{v}_1) + \cdots + r(\mathbf{v}_m), \\ &\leq (1 + \sqrt{2}) r(\mathbf{v}_0) + 2\left(1 + x_3 + x_3^2 + \cdots + x_3^m\right)\, r(\mathbf{v}_0), \\ &\leq \left(1 + \sqrt{2} + 2\, C_1\right)\, r(\mathbf{v}). \end{aligned}$$

■

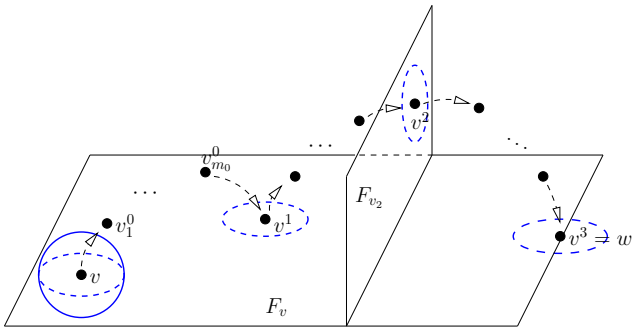


Figure 3.10: Proof of Lemma 3.3.7. v is an $R2$-vertex inserted in facet F_v. w is the ancestor of v. There are $k = 3$ SoPs bewteen v and w.

Lemma 3.3.7 *Let $\boldsymbol{v}$ be a newly inserted $R2$-vertex, $\boldsymbol{w}$ be its ancestor, and $\rho_0 > \sqrt{2}$.*

(c4) $r(\boldsymbol{v}) \geq x_2^{-k}\, \rho_0^m\, r(\boldsymbol{w})$, and $\|\boldsymbol{v} - \boldsymbol{w}\| \leq C_2\, C_1\, r(\boldsymbol{v})$, and

(c4') $lfs(\boldsymbol{v}) \leq C_2\, C_1\, r(\boldsymbol{v})$, if $\boldsymbol{w}$ is an input vertex.

Where $k \geq 1$ is the number of SoPs between $\boldsymbol{v}$ and $\boldsymbol{w}$, m is the total number of $R3$-vertices between $\boldsymbol{v}$ and $\boldsymbol{w}$. $C_2 = \frac{(1+\sqrt{2})\rho_0 - 1}{\rho_0 - \sqrt{2}}$.

Proof Let F_v denote the facet on which $\mathbf{v}$ lies. Let $\mathbf{v}^0 = \mathbf{v}$, we find $\mathbf{v}^1$ which is the grandparent of $\mathbf{v}^0$. If $\mathbf{v}^1 \in F_{v^1}$, and either $F_{v^1} = F_{v^0}$ or $F_{v^1} \cap F_{v^0} = S$, where S is a segment shared by F_{v^1} and F_{v^0}, then $\mathbf{v}^1 \neq \mathbf{w}$. We continue to find its grandparent $\mathbf{v}^2$. This process can be repeated until we find a $\mathbf{v}^k = \mathbf{w}$ ($k \geq 1$). Since $\mathbf{w}$ must exist, k is finite (see Fig. 3.10).

The complete expansion of SoPs is listed in (3.5). Note that $\mathbf{p}^i$ is not an existing mesh vertex, it shows up since it influences the insertion radii.

$$k \begin{cases} \mathbf{v} = \mathbf{v}^0, & (\mathbf{p}^0), & \underbrace{\mathbf{v}_1^0, \mathbf{v}_2^0, \cdots,}_{m_0} & \mathbf{v}_{m_0+1}^0, \\ \mathbf{v}_{m_0+1}^0 = \mathbf{v}^1, & (\mathbf{p}^1), & \underbrace{\mathbf{v}_1^1, \mathbf{v}_2^1, \cdots,}_{m_1} & \mathbf{v}_{m_1+1}^1, \\ \cdots, & & \cdots, & \\ \mathbf{v}_{m_{k-2}+1}^{k-2} = \mathbf{v}^{k-1}, & (\mathbf{p}^{k-1}), & \underbrace{\mathbf{v}_1^{k-1}, \mathbf{v}_2^{k-1}, \cdots,}_{m_{k-1}} & \mathbf{v}_{m_{k-1}+1}^{k-1} = \mathbf{w}. \end{cases} \tag{3.5}$$

The relation between $r(\mathbf{v}^i)$ and $r(\mathbf{v}^{i+1})$ is given by *(c2)*. Let $S_j = \sum_{i=0}^{j} m_i$. Then:

$$\begin{array}{rcl@{\qquad}rcl} & & & r(\mathbf{v}^0) & \geq & x_2^{-1}\rho_0^{m_0}\, r(\mathbf{v}^1), \\ r(\mathbf{v}^1) & \geq & x_2^{-1}\rho_0^{m_1}\, r(\mathbf{v}^2) & \Longrightarrow\ r(\mathbf{v}^0) & \geq & x_2^{-2}\rho_0^{m_0+m_1}\, r(\mathbf{v}^2), \\ & & & \cdots & & \\ r(\mathbf{v}^{k-1}) & \geq & x_2^{-1}\, \rho_0^{m_{k-1}}\, r(\mathbf{v}^k) & \Longrightarrow\ r(\mathbf{v}_0) & \geq & x_2^{-k}\, \rho_0^{S_{k-1}}\, r(\mathbf{v}^k). \end{array} \tag{3.6}$$

Next we estimate an upper bound of $\|\mathbf{v}-\mathbf{w}\|$ by summing the distances traversed from $\mathbf{v}$ to $\mathbf{w}$ in (3.5) and together with *(c2)*, we get,

$$\begin{aligned}
\|\mathbf{v}-\mathbf{w}\| &\leq \|\mathbf{v}^1-\mathbf{v}^0\|+\|\mathbf{v}^2-\mathbf{v}^1\|+\cdots+\|\mathbf{v}^k-\mathbf{v}^{k-1}\|,\\
&\leq (1+\sqrt{2}\,C_1)\,\left(r(\mathbf{v}^0)+r(\mathbf{v}^1)+\cdots+r(\mathbf{v}^{k-1})\right),\\
&\leq (1+\sqrt{2}\,C_1)\,\left(1+x_2x_3^{S_0}+x_2^2x_3^{S_1}+\cdots+x_2^{k-1}x_3^{S_{k-2}}\right)\,r(\mathbf{v}^0),\\
&\leq (1+\sqrt{2}\,C_1)\,\left(1+x_2+x_2^2+\cdots+x_2^{k-1}\right)\,r(\mathbf{v}^0),\\
&\leq \left(1+\frac{\sqrt{2}\rho_0}{\rho_0-1}\right)\,\frac{\rho_0}{\rho_0-\sqrt{2}}\,r(\mathbf{v}),\\
&= \frac{(1+\sqrt{2})\rho_0-1}{\rho_0-\sqrt{2}}\,\frac{\rho_0}{\rho_0-1}\,r(\mathbf{v}),\\
&= C_2\,C_1\,r(\mathbf{v}).
\end{aligned}$$

If $\mathbf{w}$ is an input vertex, then $r(\mathbf{w})\geq \mathrm{lfs}(\mathbf{w})$, and $\mathrm{lfs}(\mathbf{v})\leq\|\mathbf{v}-\mathbf{w}\|+r(\mathbf{w})$. We can add $r(\mathbf{w})$ at the end of the summation in the above equation and the result is still bounded by $C_2\,C_1\,r(\mathbf{v})$, this proofs *(c4')*. ■

Lemma 3.3.8 *Let $\boldsymbol{v}$ be a newly inserted R1-vertex, and $\rho_0>2$.*

(c5) lfs($\boldsymbol{v}$) $\leq C_3\,C_1\,r(\boldsymbol{v})$,

where $C_3=\frac{(3+\sqrt{2})\rho_0-(1+\sqrt{2})}{\rho_0-2}$.

Proof We first find an input vertex $\mathbf{w}$ through the parent chain of $\mathbf{v}$ (such $\mathbf{w}$ must exist). Let $\mathbf{v}^0=\mathbf{v}$, we find $\mathbf{v}^1$ which is the grandparent of $\mathbf{v}^0$. If $\mathbf{v}^1\neq\mathbf{w}$, we continue to find its grandparent $\mathbf{v}^2$. This process can be repeated until we find a $\mathbf{v}^k=\mathbf{w}$ ($k\geq 1$). The complete expansion of SoPs is similar to (3.5).

For each $\mathbf{v}^i$, $r(\mathbf{v}^i)$ is its insertion radius. The relation between $r(\mathbf{v}^i)$ and $r(\mathbf{v}^{i+1})$ is given by *(c1)*. Similar to (3.6), we get:

$$\begin{aligned}
r(\mathbf{v}^0) &\geq x_1^{-1}\rho_0^{m_0}\,r(\mathbf{v}^1),\\
&\geq x_1^{-2}\rho_0^{m_0+m_1}\,r(\mathbf{v}^2),\\
&\cdots\ ,\\
&\geq x_1^{-k}\,\rho_0^{S_{k-1}}\,r(\mathbf{v}^k).
\end{aligned}$$

Since $\mathbf{v}^k$ is an input vertex, $r(\mathbf{v}^k)\geq\mathrm{lfs}(\mathbf{v}^k)$ (Lemma 3.3.1), with *(c1)*, we get:

$$\begin{aligned}
\mathrm{lfs}(\mathbf{v}) &\leq \|\mathbf{v}-\mathbf{v}^k\|+\mathrm{lfs}(\mathbf{v}^k),\\
&\leq \left(\|\mathbf{v}^1-\mathbf{v}^0\|+\|\mathbf{v}^2-\mathbf{v}^1\|+\cdots+\|\mathbf{v}^k-\mathbf{v}^{k-1}\|\right)+r(\mathbf{v}^k),
\end{aligned}$$

$$
\begin{aligned}
&\leq (1+\sqrt{2}+2\,C_1)\,\left(r(\mathbf{v}^0)+r(\mathbf{v}^1)+\cdots+r(\mathbf{v}^{k-1})+r(\mathbf{v}^k)\right),\\
&\leq (1+\sqrt{2}+2\,C_1)\,\left(1+x_1x_3^{S_0}+x_1^2x_3^{S_1}+\cdots+x_1^kx_3^{S_{k-1}}\right)\,r(\mathbf{v}^0),\\
&\leq (1+\sqrt{2}+2\,C_1)\,\left(1+x_1+x_1^2+\cdots+x_1^k\right)\,r(\mathbf{v}^0),\\
&\leq \left(1+\sqrt{2}+\frac{2\rho_0}{\rho_0-1}\right)\,\frac{\rho_0}{\rho_0-2}\,r(\mathbf{v}),\\
&= \frac{(3+\sqrt{2})\rho_0-(1+\sqrt{2})}{\rho_0-2}\,\frac{\rho_0}{\rho_0-1}\,r(\mathbf{v}),\\
&= C_3\,C_1\,r(\mathbf{v}).
\end{aligned}
$$

■

Our goal is to show that there exists a constant D between $r(\mathbf{v})$ and $\mathrm{lfs}(\mathbf{v})$, such that $r(\mathbf{v})$ is bounded by the $\frac{\mathrm{lfs}(\mathbf{v})}{D}$. D should not be arbitrarily large. Lemmata 3.3.6 to 3.3.8 give some of these bounds in *(c3')*, *(c4')*, and *(c5)*. But these constants only apply for special cases. Below, we introduce a general formula to represent D.

Definitions Let $\mathbf{v}$ be an inserted vertex. The type and location of $\mathbf{v}$ determine the parameters: (k_1, m_1), and (k_2, m_2). Where

(k_1, m_1) are the exponents in *(c3)*, i.e., $x_3^{k_1}x_3^{m_1}$. k_1 can only be 1 or 0, which means $\mathbf{v}$ has one or none SoP, and m_1 counts the number of $R3$-vertices in the SoP of $\mathbf{v}$. If $\mathbf{v}$ is an $R3$-vertex, then $k_1 = 1$, $m_1 \geq 0$. If $\mathbf{v}$ is an $R2$- or $R1$-vertex, then $k_1 = m_1 = 0$.

(k_2, m_2) are the exponents in *(c4)*, i.e., $x_2^{k_2}x_3^{m_2}$. Let $\mathbf{w}$ be $\mathbf{v}$'s grandparent, $\mathbf{u}$ be $\mathbf{w}$'s ancestor. Then k_2 counts the number of SoPs between $\mathbf{w}$ and $\mathbf{u}$, and m_2 counts the total number of $R3$-vertices appearing in the k_2-SoPs (refer to Fig. 3.10). If $\mathbf{v}$ is an $R1$-vertex, then $k_2 = m_2 = 0$.

The variables (k_1, m_1), and (k_2, m_2) are generally different from vertex to vertex. Let $\mathbf{c} \in \mathbb{R}^{3\times1}$, and $M \in \mathbb{R}^{3\times3}$ defined as follows:

$$
\mathbf{c} = \begin{pmatrix} 1 \\ C_2 \\ C_3 \end{pmatrix},\ M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & x_3^{k_1+m_1} & 0 \\ 0 & 0 & x_2^{k_2}x_3^{k_1+m_1+m_2} \end{pmatrix}. \tag{3.7}
$$

Let $\mathbf{b} \in \{0,1\}^{3\times1}$, i.e., $\mathbf{b}$ is a three-dimensional column vector whose components are either 0 or 1. $\mathbf{b}$ depends on the type of $\mathbf{v}$, the type of $\mathbf{v}$'s grandparent $\mathbf{w}$, and the type of $\mathbf{w}$'s ancestor $\mathbf{u}$. For an example, if $\mathbf{v}$ is an $R3$-vertex, $\mathbf{w}$ is an $R2$-vertex, and $\mathbf{u}$ is an $R1$-vertex, then $\mathbf{b}_\mathbf{V} = (1,1,1)^T$. Fig. 3.11 lists the 7 possible cases of $\mathbf{b}$. Note that $\mathbf{b}$ will never be $\mathbf{0}$.

We define the function $D(\mathbf{v})$ as the inner product of $\mathbf{c}$ and $M\mathbf{b}$, i.e.,

$$
D(\mathbf{v}) = \mathbf{c}^T M_\mathbf{V}\mathbf{b}_\mathbf{V}. \tag{3.8}
$$

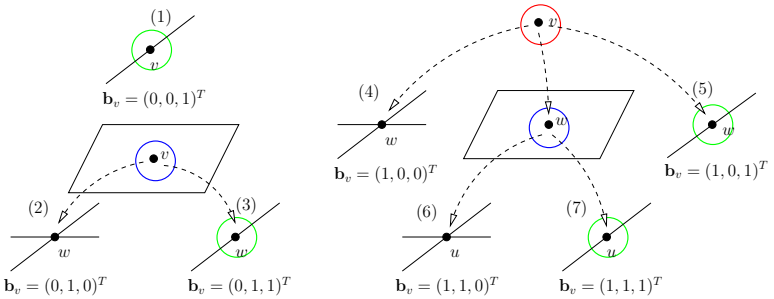


Figure 3.11: All possible cases (*(1)* – *(7)*) of $\mathbf{b_V}$. In the above figure, vertices inside green circles are $R1$-vertex, $R2$-vertices are in blue circles, and a red circle labels the $R3$-vertex. Vertices not in circles are input vertices.

In the above equation, the vector $\mathbf{c}$ only depends on ρ_0. While the matrix $M_\mathbf{V}$ and the vector $\mathbf{b_V}$ depend on $\mathbf{v}$.

Theorem 3.3.9 *Let $\boldsymbol{v}$ be a newly inserted vertex, $\boldsymbol{p} = p(\boldsymbol{v})$, and $\rho_0 > 2$.*

(c6) $lfs(\boldsymbol{v}) \leq C_1\, D(\boldsymbol{v})\, r(\boldsymbol{v})$.

(c7) $lfs(\boldsymbol{p}) \leq (1 + C_1\, D(\boldsymbol{v}))\; r(\boldsymbol{v})$.

(c8) $D(\boldsymbol{v}) \leq D(\boldsymbol{p})$, *if $\boldsymbol{p}$ is an inserted vertex.*

Proof First note that if claim *(c6)* is true, one can easily get *(c7)*:

$$\begin{aligned} \text{lfs}(\mathbf{p}) &\leq \|\mathbf{v} - \mathbf{p}\| + \text{lfs}(\mathbf{v}), \\ &\leq r(\mathbf{v}) + D(\mathbf{v})\; C_1\; r(\mathbf{v}), \\ &= (1 + D(\mathbf{v})\; C_1)\; r(\mathbf{v}). \end{aligned}$$

Proof of *(c6)*. This claim can be proved by directly verifying all the possible cases listed in Fig. 3.11. One shows that each has a $M_\mathbf{V}$ and $\mathbf{b_V}$ which define $D(\mathbf{v})$, and $D(\mathbf{v})$ satisfies our claim. Below we only verify the cases *(1)*, *(3)* and *(7)*.

(1): $\mathbf{v}$ is an $R1$-vertex. Then $k_1 = m_1 = k_2 = m_2 = 0$. By *(c5)*, we get,

$$\text{lfs}(\mathbf{v}) \leq C_3\; C_1\; r(\mathbf{v}) = \mathbf{c}^T M_\mathbf{V} \mathbf{b_V}\; C_1\; r(\mathbf{v}),$$

where,

$$M_\mathbf{V} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},\; \mathbf{b_V} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{3.9}$$

(3): $\mathbf{v}$ is an $R2$-vertex, and $\mathbf{w}$ is an $R1$-vertex. Then $k_1 = m_1 = 0$. By *(c3)* and *(c4)*, we get,

$$\text{lfs}(\mathbf{v}) \leq \|\mathbf{v} - \mathbf{w}\| + \text{lfs}(\mathbf{w}),$$

$$\begin{aligned} &\leq C_2\, C_1\, r(\mathbf{v}) + C_3\, C_1\, r(\mathbf{w}), \\ &\leq \left(C_2 + C_3\, x_2^{k_2} x_3^{m_2}\right)\, C_1\, r(\mathbf{v}), \\ &= \mathbf{c}^T M_{\mathbf{V}} \mathbf{b}_{\mathbf{V}}\, C_1\, r(\mathbf{v}), \end{aligned}$$

where

$$M_{\mathbf{V}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & x_2^{k_2} x_3^{m_2} \end{pmatrix},\ \mathbf{b}_{\mathbf{V}} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}. \tag{3.10}$$

(7): $\mathbf{v}$ is an $R3$-vertex, $\mathbf{w}$ is an $R2$-vertex, $\mathbf{u}$ is an $R1$-vertex. Then $K_1 = 1$. By *(c3)* and *(c4)*, we get,

$$\begin{aligned} \mathrm{lfs}(\mathbf{v}) &\leq \|\mathbf{v} - \mathbf{w}\| + \|\mathbf{w} - \mathbf{u}\| + \mathrm{lfs}(\mathbf{u}), \\ &\leq C_1\, r(\mathbf{v}) + C_2\, C_1\, r(\mathbf{w}) + C_3\, C_1\, r(\mathbf{u}), \\ &\leq \left(1 + C_2\, x_3^{k_1} x_3^{m_1} + C_3\, x_2^{k_2} x_3^{k_1+m_1+m_2}\right)\, C_1\, r(\mathbf{v}), \\ &= \mathbf{c}^T M_{\mathbf{V}} \mathbf{b}_{\mathbf{V}}\, C_1\, r(\mathbf{v}), \end{aligned}$$

where,

$$M_{\mathbf{V}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & x_3^{k_1+m_1} & 0 \\ 0 & 0 & x_2^{k_2} x_3^{k_1+m_1+m_2} \end{pmatrix},\ \mathbf{b}_{\mathbf{V}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{3.11}$$

Proof of *(c8)*. This claim can be proved by directly verifying all the possible cases of $\mathbf{v}$ and $\mathbf{p}$ and showing that $D(\mathbf{p}) - D(\mathbf{v}) \geq 0$. Below follows three verifications of the critical cases.

Case 1: Assume both $\mathbf{v}$ and $\mathbf{p}$ are $R3$-vertices. Then $\mathbf{b}_{\mathbf{V}} = \mathbf{b}_{\mathbf{P}}$ (since they have the same grandparent and ancestor). Suppose $M_{\mathbf{V}}$ is given as (3.11). $M_{\mathbf{P}}$ differs $M_{\mathbf{V}}$ in such a way that the sum $k_1 + m_1$ is reduced by 1, i.e.,

$$M_{\mathbf{P}} = M_{\mathbf{V}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & x_3^{-1} & 0 \\ 0 & 0 & x_3^{-1} \end{pmatrix}.$$

While $x_3 = \frac{1}{\rho_0}$ and $\rho_0 > 2$ follows

$$\begin{aligned} D(\mathbf{p}) - D(\mathbf{v}) &= \mathbf{c}^T M_{\mathbf{P}} \mathbf{b}_{\mathbf{P}} - \mathbf{c}^T M_{\mathbf{V}} \mathbf{b}_{\mathbf{V}}, \\ &= \mathbf{c}^T \left(M_{\mathbf{P}} - M_{\mathbf{V}}\right) \mathbf{b}_{\mathbf{V}}, \\ &= \mathbf{c}^T M_{\mathbf{V}} \begin{pmatrix} 0 & 0 & 0 \\ 0 & \rho_0 - 1 & 0 \\ 0 & 0 & \rho_0 - 1 \end{pmatrix} \mathbf{b}_{\mathbf{V}}, \\ &> 0. \end{aligned} \tag{3.12}$$

Case 2: Assume $\mathbf{v}$ is an $R3$-vertex, but $\mathbf{p}$ is an $R2$-vertex. Then $M_{\mathbf{v}}$, $\mathbf{b_v}$ are as (3.11), with $k_1 + m_1 = 1$. While $M_{\mathbf{p}}$, $\mathbf{b_p}$ are as (3.10). Then

$$\begin{aligned}
D(\mathbf{p}) - D(\mathbf{v}) &= \mathbf{c}^T M_{\mathbf{p}} \mathbf{b_p} - \mathbf{c}^T M_{\mathbf{v}} \mathbf{b_v}, \\
&= \mathbf{c}^T \left(M_{\mathbf{p}} \mathbf{b_p} - M_{\mathbf{v}} \mathbf{b_v} \right), \\
&= \mathbf{c}^T \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & x_2^{k_2} x_3^{m_2} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & x_3 & 0 \\ 0 & 0 & x_2^{k_2} x_3^{m_2+1} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right], \\
&= \mathbf{c}^T \left[\begin{pmatrix} 0 \\ 1 \\ x_2^{k_2} x_3^{m_2} \end{pmatrix} - \begin{pmatrix} 1 \\ x_3 \\ x_2^{k_2} x_3^{m_2+1} \end{pmatrix} \right], \\
&= (1, C_2, C_3) \begin{pmatrix} -1 \\ 1 - x_3 \\ x_2^{k_2} x_3^{m_2} (1 - x_3) \end{pmatrix}, \\
&= -1 + C_2 \, (1 - x_3) + C_3 \, x_2^{k_2} x_3^{m_2} (1 - x_3), \\
&\geq -1 + C_2 \, (1 - x_3). \qquad (3.13)
\end{aligned}$$

Since $C_2 = \frac{(1+\sqrt{2})\rho_0 - 1}{\rho_0 - \sqrt{2}}$, and $x_3 = \frac{1}{\rho_0}$, one verifies

$$C_2 \, (1 - x_3) > 1. \qquad (3.14)$$

So (3.13) is positive, which implies that $D(\mathbf{p}) - D(\mathbf{v}) > 0$.

Case 3: Assume $\mathbf{v}$ is an $R2$-vertex, but $\mathbf{p}$ is an $R1$-vertex. Then $M_{\mathbf{v}}$, $\mathbf{b_v}$ are as (3.10), with $k_2 = 1, m_2 = 0$. While $M_{\mathbf{p}}$, $\mathbf{b_p}$ are as (3.9). Then,

$$\begin{aligned}
D(\mathbf{p}) - D(\mathbf{v}) &= \mathbf{c}^T M_{\mathbf{p}} \mathbf{b_p} - \mathbf{c}^T M_{\mathbf{v}} \mathbf{b_v}, \\
&= \mathbf{c}^T \left(M_{\mathbf{p}} \mathbf{b_p} - M_{\mathbf{v}} \mathbf{b_v} \right), \\
&= \mathbf{c}^T \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & x_2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right], \\
&= \mathbf{c}^T \left[\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ x_2 \end{pmatrix} \right], \\
&= (1, C_2, C_3) \begin{pmatrix} 0 \\ -1 \\ 1 - x_2 \end{pmatrix}, \\
&= -C_2 + C_3 \, (1 - x_2), \qquad (3.15)
\end{aligned}$$

and $C_3 = \frac{(3+\sqrt{2})\rho_0 - (1+\sqrt{2})}{\rho_0 - 2}$, $x_2 = \frac{\sqrt{2}}{\rho_0}$, hence

$$C_3 \, (1 - x_2) > C_2. \qquad (3.16)$$

So (3.15) is positive, which implies that $D_p - D_v > 0$. ■

(c6) (in Theorem 3.3.9) shows that for any inserted vertex $\mathbf{v}$, the shortest edge length at $\mathbf{v}$ is proportional to its local feature size. Hence, edge lengths are determined by local considerations, i.e., the mesh size will be graded from vertices close to small features to those located in empty regions.

Discussion. This definition of $D(\mathbf{v})$ well explains the behavior of Delaunay refinement and the output mesh size distribution.

Let the input PLS is only a set of points, i.e., no segments and facets. All inserted vertices are $R3$-vertices. For any inserted vertex $\mathbf{v}$, $\mathbf{b_V} = (1,0,0)^T$. Then

$$D(\mathbf{v}) = \mathbf{c}^T M_{\mathbf{V}} \mathbf{b_V} = 1.$$

Hence, lfs($\mathbf{v}$) is bounded by $C_1 = \frac{\rho_0}{\rho_0 - 1}$. This implies that the algorithm will terminate and the output edge lengths will be well-graded as long as $\rho_0 > 1$.

For a vertex $\mathbf{v}$ inserted in the bulk of the mesh, i.e., $\mathbf{v}$ is far from the boundary, the sum $k_1 + m_1$ is usually large (see Fig. 3.12 left), hence in (3.11), $1 >> x_3^{k_1+m_1} >> x_2^{k_2} x_3^{k_1+m_1+m_2}$. $D(\mathbf{v})$ can be estimated by:

$$D(\mathbf{v}) = \mathbf{c}^T M_{\mathbf{V}} \mathbf{b_V} \approx 1.$$

Hence, lfs($\mathbf{v}$) is approximately bounded by $C_1 = \frac{\rho_0}{\rho_0 - 1}$. This implies that the mesh in the bulk of the domain will be graded if $\rho_0 > 1$.

For a vertex $\mathbf{v}$ inserted on the facet, k_2, and m_2 is large (see Fig. 3.12 right), then in (3.10) $1 >> x_2^{k_2} x_3^{m_2}$ holds. We can estimate $D(\mathbf{v})$ by:

$$D(\mathbf{v}) = \mathbf{c}^T M_{\mathbf{V}} \mathbf{b_V} \approx C_2.$$

Hence, lfs($\mathbf{v}$) is approximatedly bounded by C_2 $C_1 = \frac{(1+\sqrt{2})\rho_0 - 1}{\rho_0 - \sqrt{2}} \frac{\rho_0}{\rho_0 - 1}$. This implies that the mesh on the boundary will be graded if $\rho_0 > \sqrt{2}$.

Theorem 3.3.9 implies the following corollary which is comparable to the Theorem 3.3.5.

Corollary 3.3.10 *Let $\boldsymbol{v}$ and $\boldsymbol{w}$ be any two output mesh vertices. If $\boldsymbol{v}$ is an input vertex, define $D(\boldsymbol{v}) \equiv C_3$. If $\rho_0 > 2$, then,*

(c9) $\|\boldsymbol{v} - \boldsymbol{w}\| \geq \frac{lfs(\boldsymbol{v})}{D(\boldsymbol{v})\, C_1 + 1}$.

Proof For any inserted vertex $\mathbf{v}$, *(c8)* implies that through the SoP of $\mathbf{v}$,

$$D(\mathbf{v}_0) \leq D(\mathbf{v}_1) \leq \cdots \leq D(\mathbf{v}_m),$$

where the parent of $\mathbf{v}_m$ is an input vertex. By *(c3')*, *(c4')*,*(c5)*, and since $1 < C_1 < C_2 < C_3$, hence $D(\mathbf{v}_m) \leq C_3$. This implies:

$$C_3 = \max\{D(\mathbf{v}) \mid \mathbf{v} \text{ is an inserted vertex.}\}. \tag{3.17}$$

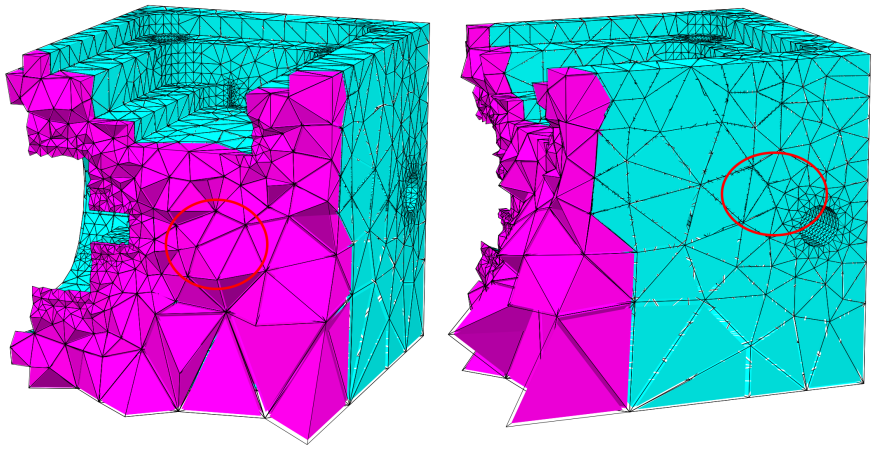


Figure 3.12: Illustration of Theorem 3.3.9. *(c6)* implies that the mesh in the bulk of the domain is well graded when $\rho_0 > 1$ (left), and the mesh on the surface is well graded when $\rho_0 > \sqrt{2}$ (right). (This mesh was generated with $\rho_0 = \sqrt{2}$.)

Now suppose $\mathbf{v}$ is any output mesh vertex, and $\mathbf{w}$ be its closest neighbor. We have the following cases.

Case 1: Both $\mathbf{v}$ and $\mathbf{w}$ are input vertices. Then by definition, $\|\mathbf{v}-\mathbf{w}\| \geq \text{lfs}(\mathbf{v})$.

Case 2: $\mathbf{v}$ was inserted after $\mathbf{w}$. Then by *(c6)*, $\|\mathbf{v}-\mathbf{w}\| \geq \frac{\text{lfs}(\mathbf{v})}{D(\mathbf{v})\,C_1} \geq \frac{\text{lfs}(\mathbf{v})}{C_3\,C_1}$.

Case 3: $\mathbf{v}$ is an input vertex or was inserted before $\mathbf{w}$. Then by *(c7)*, $\|\mathbf{v}-\mathbf{w}\| \geq r(\mathbf{w}) \geq \frac{\text{lfs}(\mathbf{v})}{1+D(\mathbf{v})\,C_1} \geq \frac{\text{lfs}(\mathbf{v})}{1+C_3\,C_1}$. ■

Corallary 3.3.10 shows that all edges in the output mesh have a bounded length with respect to the local feature sizes of their endpoints. Thus the output mesh size must be bounded as well. A common approach is to get an upper bound of the number of output vertices in terms of the integral $\int_\Omega \frac{\mathrm{d}x}{\text{lfs}(x)}$, similarly to what is done in the 2D case, see [92, 81, 35, 18].

3.3.3 Mesh Quality

Theorem 3.3.4 shows that the Delaunay refinement algorithm terminates and no output tetrahedron has radius-edge ratio larger than ρ_0, when $\rho_0 > 2$. Theorem 3.3.9 also requires that bound of ρ_0 to bound the mesh size. However, in practice this algorithm works well when ρ_0 is much smaller. In this section, we show that ρ_0 can be reduced to $\sqrt{2}$ if we impose an

additional but easy to satisfied condition on the input segments. Later on, we comment that this condition is unlikely to be needed in practice.

Assumption 3.3.11 (Input Segment) *If S_1 and S_2 are two adjoining segments that are non-collinear, then they have the same length.*

It is simple to show that this assumption can be satisfied by effectively re-defining the input PLS $\mathcal{X}$ with no more than $2m$ new points, where m is the number of input segments of $\mathcal{X}$.

We first need two geometric lemmas for any $R1$-vertex $\mathbf{v}$ whose insertion was caused by a *double rejection*: Let $\mathbf{p} = p(\mathbf{v})$, and $\mathbf{p}$ is a rejected circumcenter of an encroached subface, let $\mathbf{q} = p(\mathbf{p})$, and $\mathbf{q}$ is a rejected circumcenter of a bad quality tetrahedron (refer to Fig. 3.10 Right). Let $\mathbf{w} = p(\mathbf{q})$ and is an existing mesh vertex, we call $\mathbf{w}$ the *source vertex*. A double rejection imposes that $r(\mathbf{v}) \geq \frac{\rho_0}{2} r(\mathbf{w})$. This is the reason why we need a bound of $\rho_0 > 2$ in both proofs of the termination and mesh size. The next two lemmas relax this relation in special cases.

Lemma 3.3.12 *Suppose that the input PLS satisfies the Input Segment Assumption. Let $\boldsymbol{v}$ be the midpoint of a segment whose insertion is caused by a double rejection. Suppose that the source $\boldsymbol{w}$ is on an adjoining input segment. If $\rho_0 > \sqrt{2}$, then $r(\boldsymbol{v}) \geq r(\boldsymbol{w})$.*

Proof Let $\mathbf{xy}$, $\mathbf{xz}$ be the two input segments containing $\mathbf{v}$ and $\mathbf{w}$, respectively. Let $\mathbf{ab}$ be the subsegment of which $\mathbf{v}$ is midpoint. Let $\mathbf{cd}$ be that for which $\mathbf{w}$ is midpoint. Assume $\mathbf{a}$ is closer to $\mathbf{x}$ than $\mathbf{b}$, and assume $\mathbf{c}$ is closer to $\mathbf{x}$ than $\mathbf{d}$. It may be the case that $\mathbf{x} = \mathbf{a}$, or $\mathbf{x} = \mathbf{c}$.

Case 1: Suppose that $\mathbf{xy}$ and $\mathbf{xz}$ are non-collinear. It is to easy see there exist integers n and m, where $n = \log_2 \frac{\|\mathbf{x}-\mathbf{y}\|}{\|\mathbf{a}-\mathbf{b}\|}$, $m = \log_2 \frac{\|\mathbf{x}-\mathbf{z}\|}{\|\mathbf{c}-\mathbf{d}\|}$. Then,

$$\begin{aligned} r(\mathbf{v}) &= \frac{1}{2^n}\|\mathbf{x}-\mathbf{y}\|, \\ r(\mathbf{w}) &= \frac{1}{2^m}\|\mathbf{x}-\mathbf{z}\|, \\ r(\mathbf{v}) &\geq \frac{\rho_0}{2}\, r(\mathbf{w}), \\ \frac{1}{2^n}\|\mathbf{x}-\mathbf{y}\| &\geq \frac{\rho_0}{2^{m+1}}\|\mathbf{x}-\mathbf{z}\|. \end{aligned} \tag{3.18}$$

Since we assume $\|\mathbf{x}-\mathbf{y}\| = \|\mathbf{x}-\mathbf{z}\|$, and $\rho_0 > \sqrt{2}$. Then,

$$2^{m-n+1} \geq \rho_0 \geq 2^{\frac{1}{2}} \Longrightarrow m + \frac{1}{2} \geq n. \tag{3.19}$$

Since both m and n are integers, (3.19) can only be satisfied when $m \geq n$. Then by (3.18), it must be $r(\mathbf{v}) \geq r(\mathbf{w})$.

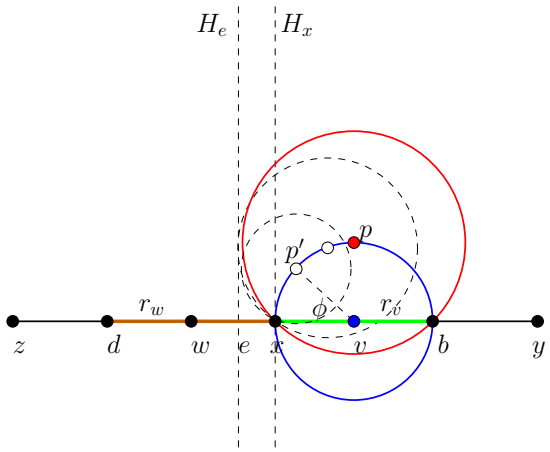


Figure 3.13: Proof of Lemma 3.3.12. **xy** and **xz** are two collinear segments sharing at vertex **x**. **v** on **xy** was caused by a double rejection, where **w** on **xz** is the source of **v**. The goal is to show that $r(\mathbf{v}) \geq r(\mathbf{w})$. The blue circle (having radius $r(\mathbf{v})$) is the largest possible region where a rejected circumcenter **p** of a subface can appear. The red circle (having radius $\|\mathbf{p} - \mathbf{x}\| = \sqrt{2}r(\mathbf{v})$) is the largest possible region where a rejected circumcenter **q** (not explicitly shown) of a bad quality tetrahedra can appear. H_e and H_x are two hyperplanes, **q** can be located only between H_e and H_x.

Case 2: Suppose that **xy** and **xz** are collinear. The sketch for proving is shown in Fig. 3.13. This figure shows the extreme case, that is $\mathbf{a} = \mathbf{c} = \mathbf{x}$, which results in the largest possible $r(\mathbf{w})$. Let **e** be the bisector of **wx**. By the double rejection assumption, we have the following facts:

Fact 1: **q** is the circumcenter of a bad quality tetrahedron (not shown in this figure) whose shortest edge is **wx**, hence **q** must lie on the hyperplane H_e which vertically bisects **wx**.

Fact 2: **p** is the circumcenter of a subface (not shown) whose diametric ball contains **q**, i.e., **q** encroaches upon it. But **q** must not lie on the hyperplane H_x since we assume **w** and **x** are distinct. Hence the hyperplane containing **q** can be only between H_e and H_x.

What remains is to find the relation between $\|\mathbf{e} - \mathbf{x}\|$ and $r(\mathbf{v})$. Since **p** must be inside the sphere centered at **v** with radius $r(\mathbf{v})$. Hence the largest possible $\|\mathbf{e} - \mathbf{x}\|$ can be found by moving **p** along the blue circle toward **x**. Let $\mathbf{p}'$ be the moving point, ϕ be the angle between edges $\mathbf{p}'\mathbf{v}$ and **vx** (refer to Fig. 3.13). Then,

$$\begin{aligned} \|\mathbf{v} - \mathbf{e}\| &= r(\mathbf{v})\ \cos\phi + \|\mathbf{p}' - \mathbf{x}\|, \\ &= \|\mathbf{v} - \mathbf{p}'\|\ \cos\phi + \sqrt{r^2(\mathbf{v})\sin\phi^2 + r^2(\mathbf{v})(1-\cos\phi)^2}, \\ &= r(\mathbf{v})\ (\cos\phi + \sqrt{2}\ \sqrt{1-\cos\phi}), \end{aligned}$$

$$\leq \frac{3}{2}\, r(\mathbf{v}). \tag{3.20}$$

Since $r(\mathbf{w}) = 2\|\mathbf{e}-\mathbf{x}\|$, and $\|\mathbf{e}-\mathbf{x}\| = \|\mathbf{v}-\mathbf{e}\| - r(\mathbf{v})$, we get,

$$\begin{aligned} r(\mathbf{w}) &= 2(\|\mathbf{v}-\mathbf{e}\| - r(\mathbf{v})), \\ &\leq 2(\frac{3}{2}\, r(\mathbf{v}) - r(\mathbf{v})), \\ &\leq r(\mathbf{v}). \end{aligned}$$

■

The above lemma shows that vertices from adjacent segments do not cause an infinite loop of double rejections. The next lemma shows, the same fact holds for those vertices from disjoint segments.

Lemma 3.3.13 *Let $\boldsymbol{v}$ be the midpoint of a segment $S_{\boldsymbol{v}}$ whose insertion was caused by a double rejection. Let $\boldsymbol{u}$ be another point on a segment $S_{\boldsymbol{u}}$ which is disjoint with $S_{\boldsymbol{v}}$ ($\boldsymbol{u}$ may be either an input or an inserted vertex). If the length of the segment $S_{\boldsymbol{v}}$ (denoted as $\|S_{\boldsymbol{v}}\|$) is sufficiently small, then $\boldsymbol{u}$ does not cause the insertion of $\boldsymbol{v}$.*

Proof Since $S_{\mathbf{V}}$ and $S_{\mathbf{U}}$ are disjoint, let $d_{\mathbf{VU}}$ be the shortest distance between them. Clearly, we have

$$d_{\mathbf{VU}} \leq \|\mathbf{v}-\mathbf{u}\|. \tag{3.21}$$

Now suppose that the contrary of our claim is true, that is, no matter how small $\|S_{\mathbf{V}}\|$ is, $\mathbf{u}$ may cause the insertion of $\mathbf{v}$. We're going to check all the possible cases and show a contradiction of (3.21).

Case 1: Suppose that $\mathbf{u}$ is just the source of $\mathbf{v}$, we have,

$$\|\mathbf{v}-\mathbf{u}\| \leq (3+\sqrt{2})\, r(\mathbf{v}) \leq \frac{1}{2}(3+\sqrt{2})\, \|S_{\mathbf{V}}\|. \tag{3.22}$$

Assume that $\|S_{\mathbf{V}}\|$ is small enough, so let it satisfy the following inequality:

$$\frac{1}{2}(3+\sqrt{2})\, \|S_{\mathbf{V}}\| < d_{\mathbf{VU}}. \tag{3.23}$$

(3.22) and (3.23) together implies that $\|\mathbf{v}-\mathbf{u}\| < d_{\mathbf{VU}}$, a contradiction to (3.21).

Case 2: The source $\mathbf{w}$ is on a facet $F_{\mathbf{W}}$ which is incident with $S_{\mathbf{V}}$, and $\mathbf{u}$ is the ancestor of $\mathbf{w}$ (see Fig. 3.14), by *(c4)*, we get,

$$\begin{aligned} \|\mathbf{v}-\mathbf{u}\| &\leq \|\mathbf{v}-\mathbf{w}\| + \|\mathbf{w}-\mathbf{u}\|, \\ &\leq (1+\sqrt{2})\, r(\mathbf{v}) + C_2\, C_1\, r(\mathbf{w}), \\ &\leq (1+\sqrt{2}+C_2\, C_1\, \frac{2}{\rho_0})\, r(\mathbf{v}), \\ &\leq \frac{1}{2}(1+\sqrt{2}+C_2\, C_1\, \frac{2}{\rho_0})\, \|S_{\mathbf{V}}\|. \end{aligned} \tag{3.24}$$

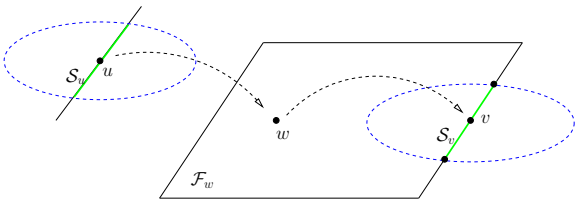


Figure 3.14: Proof of Lemma 3.3.13. $\mathbf{u}, \mathbf{v}, \mathbf{w}$ are three inserted vertices, where $\mathbf{u}$ and $\mathbf{v}$ are on two disjoint segments, and $\mathbf{w}$ is on a facet. Moreover, the insertion of $\mathbf{w}$ is caused by $\mathbf{u}$, and $\mathbf{w}$ cause the insertion of $\mathbf{v}$. The goal is to show, if the subsegment which is split by $\mathbf{v}$ is sufficiently short, then this case does not happen.

Assume that $\|S_{\mathbf{V}}\|$ is small enough, so let it satisfy the following inequality:

$$\frac{1}{2}(1 + \sqrt{2} + C_2\ C_1\ \frac{2}{\rho_0})\ \|S_{\mathbf{V}}\| < d_{\mathbf{VU}}. \tag{3.25}$$

Put (3.24) and (3.25) together, $\|\mathbf{v} - \mathbf{u}\| < d_{\mathbf{VU}}$, a contradiction to (3.21).

The last case is that the source $\mathbf{w}$ is an $R3$-vertex, and $\mathbf{u}$ is the grandparent of $\mathbf{w}$. The proof is identical to *Case 2*. ■

The next theorem summaries the main result of this section, that is, the Delaunay refinement algorithm will terminate when $\rho_0 > \sqrt{2}$ if the Input Segment Assumption is satisfied. This improves the original mesh quality guarantee which requires $\rho_0 \geq 2$.

Theorem 3.3.14 *Let $\mathcal{X}$ be a three-dimensional PLS. If $\mathcal{X}$ satisfies both the Input Angle Assumption and the Input Segment Assumption and $\rho_0 > \sqrt{2}$, then the Delaunay refinement algorithm terminates.*

Proof As long as $\mathbf{v}$ is not an $R1$-vertex or $\mathbf{v}$ is an $R1$-vertex but was not caused by a double rejection. By *(c2)* and *(c3)*, $r(\mathbf{v}) \geq \frac{1}{\sqrt{2}}\rho_0\ r(\mathbf{w}) > r(\mathbf{w})$, if $\rho_0 > \sqrt{2}$. Hence $r(\mathbf{v})$ does not decrease.

Assume that $\mathbf{v}$ is an $R1$-vertex and was caused by a double rejection. Let $S_{\mathbf{V}}$ denote the segment containing $\mathbf{v}$. For the termination proof only, we can assume that $\|S_{\mathbf{V}}\|$ is sufficiently small, so that by Lemma 3.3.13, other $R1$-vertices and input vertices on disjoint segments of $S_{\mathbf{V}}$ are not the source or the cause of the insertion of $\mathbf{v}$. Hence the source $\mathbf{w}$ can be either an $R2$-vertex or an $R3$-vertex.

If $\mathbf{w}$ is an $R3$-vertex, then let $\mathbf{u}$ be the grandparent of $\mathbf{w}$. By *(c3)*, the relation between $r(\mathbf{v})$ and $r(\mathbf{u})$ is, $r(\mathbf{v}) \geq \frac{1}{2}\rho_0\ r(\mathbf{w}) \geq \frac{1}{2}\rho_0\ \rho_0^{m_u+1}\ r(\mathbf{u}) \geq \frac{1}{2}\rho_0^{m_u+2}\ r(\mathbf{u})$. Since $\rho_0 > \sqrt{2}$, this implies that $r(\mathbf{v}) \geq r(\mathbf{u})$. Hence $r(\mathbf{v})$ does not decrease.

If $\mathbf{w}$ is an $R2$-vertex, consider the worst case, which is $r(\mathbf{v}) \geq \frac{2}{\rho_0}\, r(\mathbf{w}) \geq \sqrt{2}\, r(\mathbf{v})$, i.e., it may be $r(\mathbf{v}) < r(\mathbf{w})$. Note that we have the following facts when $\rho_2 > \sqrt{2}$: *(1)*, $r(\mathbf{v})$ will not create shorter edges on adjacent segments (by Lemma 3.3.12), and *(2)*, $r(\mathbf{v})$ does not cause infinitely many new double rejections on disjoint segments (by Lemma 3.3.13). Both facts guarantee that the insertion of $\mathbf{v}$ does not create an infinitely short edge. ■

In fact, the only case which needs the Assumption 3.3.11 is: if $\mathbf{v}$ is inserted on $S_{\mathbf{v}}$ by a double rejection and its source $\mathbf{w}$ is on $S_{\mathbf{w}}$, such that $S_{\mathbf{v}}$ and $S_{\mathbf{w}}$ are incident and are non-collinear. Let $\mathbf{e} = S_{\mathbf{v}} \cap S_{\mathbf{w}}$. If Assumption 3.3.11 is not made but only $\rho_0 > \sqrt{2}$, it is possible that $\|\mathbf{v}-\mathbf{e}\| < \|\mathbf{w}-\mathbf{e}\|$ which may endanger the termination, for instance, when the same configuration of points congruently repeats again and again during the Delaunay refinement. However, such a configuration will need at least three (or even four) other inserted vertices which are restricted to certain positions. In general, this can not often happen unless the input is constructed accordingly (Such an example is not known yet). Hence, in practice the Assumption 3.3.11 is unlikely to be needed.

3.3.4 Vertex Degree

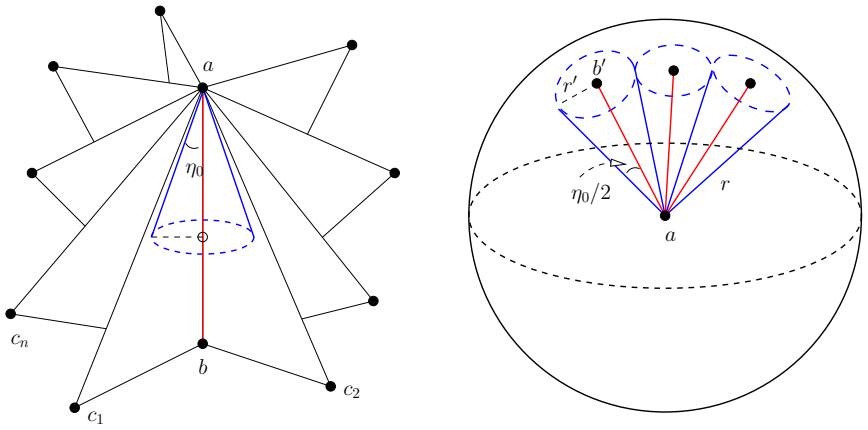
It was first proved by Miller *et al.* [79] that vertices of a bounded radius-edge ratio mesh have bounded degree, i.e., each vertex only belongs to at most some constant number of edges, and the constant only depends on ρ_0. This result is essential in proving many other properties of the meshes generated by the Delaunay refinement algorithm. However, Miller *et al.* [79] did not explicitly give the constant[1]. The one given in [35] is extremely large. In this section we prove a new upper bound of vertex degree which is more realistic. The new upper bound is not only valid for meshes created by Delaunay refinement, but also holds for all meshes with a bounded radius-edge ratio.

The proof of the upper bound is based on the "empty cone" observation, i.e., every edge of a mesh with a bounded radius-edge ratio defines a cone which is empty of incident edges.

Lemma 3.3.15 *Let $\mathcal{T}$ be a mesh with a radius-edge ratio bounded by ρ_0. Let $\boldsymbol{ab}$ be an edge of $\mathcal{T}$. Then the cone whose apex is $\boldsymbol{a}$, axis is $\boldsymbol{ab}$, and cone angle is η_0 contains no other edges of $\mathcal{T}$ incident at $\boldsymbol{a}$ in its interior. Where*

$$\eta_0 = \arcsin \frac{1}{2\rho_0}. \tag{3.26}$$

[1] In Talmor's PhD thesis [113], the Theorem 3.4.4. The constant is $C = C_2^{C_3}$, but C_3, which is the number of circular caps that form a cover of the sphere, is not explicitly given.



(a) (b)

Figure 3.15: Proof of the upper bound of the vertex degree. **(a)**: **ab** is an edge of the mesh shared by faces $\mathbf{abc}_i$, where $i = 1, 2, ..., n$. The cone (shown in blue) whose apex is **a**, axis is **ab** and angle is η_0 contains no other edge of the mesh. **(b)**: Form a maximal packing of the sphere centered at **a** by circular caps (shown in blue) which are not overlapping.

Proof Consider the set of faces sharing **ab** in the mesh. Let their apexes be $\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_n$, see Fig. 3.15 (a). Since no tetrahedra of the mesh has a radius-edge ratio larger than ρ_0, it holds on each face as well. This implies that:

$$\min\{\angle \mathbf{bac}_i \mid i = 1, ..., n\} \geq \arcsin \frac{1}{2\rho_0} = \eta_0.$$

Hence, the cone which has apex **a**, axis **ab**, and the cone angle η_0 must contain none of the edges $\mathbf{ac}_1, ..., \mathbf{a}, \mathbf{c}_n$ in its interior. ■

Theorem 3.3.16 *Let $\mathcal{T}$ be a tetrahedral mesh with a radius-edge ratio bounded by ρ_0. Then every vertex **a** of $\mathcal{T}$ belongs to at most δ_0 edges. Where*

$$\delta_0 = \frac{2}{1 - \cos(\eta_0/2)}. \tag{3.27}$$

Proof By Lemma 3.3.15 every edge containing **a** has an empty cone whose angle is η_0. Particularly, the cones around edges at **a** with angle $\eta_0/2$ do not overlap each other. It turns out that the number of edges at **a** is never larger than the maximal number of non-overlapping cones whose angles are all at least $\eta_0/2$ around **a**.

To bound the maximal number of such cones, we place a sphere Σ centered at **a** with radius r smaller than the shortest edge length at **a**. For each edge **ab** in the star of **a**, let $\mathbf{b}' \in \Sigma$ be the radial projection of **b**. Now we

place a circular cap centered at each $\mathbf{b}'$ with a radius $r' = r\sqrt{2 - 2\cos(\eta_0/2)}$, refer to Fig. 3.15 (b). Clearly, any two of such caps do not overlap. The area of each cap is $(1-\cos(\eta_0/2))/2$ times the area of Σ, which implies there are at most $2/(1 - \cos(\eta_0/2)) = \delta_0$ caps. ■

The upper bound δ_0 given in (3.27) depends only on the radius-edge ratio ρ_0. The smaller ρ_0 the smaller the bound on the vertex degree. It improves the one, denoted as δ_{old}, given in [35]. For example, for $\rho_0 = 2$, $\delta_0 \approx 251$, while $\delta_{old} > 2^{72000}$.

A straightforward consequence of the above theorem is that an improved upper bound on the longest edge length at each vertex.

Corollary 3.3.17 *Let $\mathcal{T}$ be a Delaunay mesh with a radius-edge ratio bounded by ρ_0. Among all edges sharing a vertex $\boldsymbol{a}$, let $\boldsymbol{ap}$ and $\boldsymbol{aq}$ be the shortest and longest edge, respectively. Then $\frac{\|\boldsymbol{a}-\boldsymbol{q}\|}{\|\boldsymbol{a}-\boldsymbol{p}\|} \leq \nu_0$. Where*

$$\nu_0 = (2\rho_0)^{\lfloor \delta_0/2 \rfloor}. \tag{3.28}$$

Proof Re-use the construction of Σ and the circular packing of Σ in the proof of Theorem 3.3.16. For any two adjacent points $\mathbf{x}$ and $\mathbf{y}$ on Σ, form an arc connecting $\mathbf{x}$ and $\mathbf{y}$. The points and arcs on Σ form a 3-connected planar graph. Let $\mathbf{p}'$ and $\mathbf{q}'$ be the radial projections of $\mathbf{p}$ and $\mathbf{q}$ on Σ. We walk in the graph from $\mathbf{p}'$ to $\mathbf{q}'$, and count the smallest nodes we've passed. Since there are at most δ_0 nodes, the number of visited nodes is no larger than $\nu_0 = \lfloor \delta_0/2 \rfloor$.

If we pass from one node to its connected neighbor, we are in the same tetrahedron. The bounded radius-edge ratio property implies that the maximum edge length change in a tetrahedron is at most a factor $1/(2\rho_0)$. Hence we have $\frac{\|\mathbf{a}-\mathbf{q}\|}{\|\mathbf{a}-\mathbf{p}\|} \leq \nu_0$. ■

3.3.5 Insertion Orders

In principle, the Delaunay refinement algorithm does not depend on the order of point insertion, i.e., if both $\mathbf{c}_1$ and $\mathbf{c}_2$ are circumcenters of bad quality tetrahedra, they may be added in an arbitrary order and the theoretical guarantees hold. In practice, it is noticed that the actual mesh size and running time will vary if the order of point insertion is changed. An important question regarding the implementation is: which point insertion order will lead to the best performance? This question is currently open. In this section, we take several different point insertion orders to evaluate the mesh sizes produced by the Delaunay refinement algorithm.

Orders	Rules	Comments
$O1$	If $l_i < l_j$, then $\mathbf{c}_i \prec \mathbf{c}_j$	Large radius-edge ratio first.
$O2$	If $l_i < l_j$, then $\mathbf{c}_i \prec \mathbf{c}_j$	Short edge first.
$O3$	If $r_i < r_j$, then $\mathbf{c}_i \prec \mathbf{c}_j$	Small radius first.
$O4$	If $r_i > r_j$, then $\mathbf{c}_i \prec \mathbf{c}_j$	Large radius first.
$O5$	Either $\mathbf{c}_i \prec \mathbf{c}_j$ or $\mathbf{c}_j \prec \mathbf{c}_i$	Random.

Table 3.2: Point insertion orders.

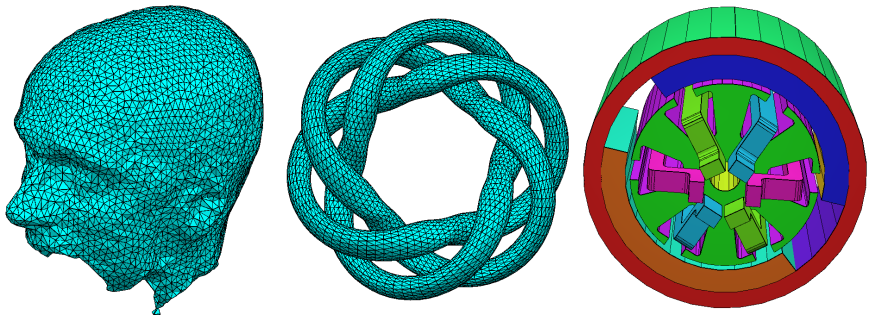


Figure 3.16: PLS models. Left: `Head`, 4684 nodes, 9364 triangles. Middle: `Hose`, 6000 nodes, 12000 triangles. Right: `Pmdc`, 972 nodes, 502 facets.

Insertion Orders To implement different insertion orders, we maintain a priority queue Q to order the circumcenters to be inserted. Let τ_i be a bad quality tetrahedra, $\mathbf{c}_i$ and r_i respectively denote the center and radius of the circumsphere of τ_i, and let l_i denote the shortest edge length of τ_i. For any $\mathbf{c}_i$ and $\mathbf{c}_j$, $\mathbf{c}_i \prec \mathbf{c}_j$ means that $\mathbf{c}_i$ has a higher priority than $\mathbf{c}_j$. The tested insertion orders are described in Table 3.2. Once $\mathbf{c}_i$ is inserted, new circumcenters of bad quality tetrahedra are inserted into Q according to the given order.

Point Clouds We first compare the behavior of the Delaunay refinement on point clouds inputs. The purpose is to study the case: only circumcenters of bad quality tetrahedra are involved. Table 3.3 shows a comparison of the output sizes (nodes and tetrahedra) for using different insertion orders on four data sets: (1) `Line-and-Circle`, it is 1000 points on a line going through 1000 points on a circle, plus a $4 \times 4 \times 4$ bounding box (it is a constructed input to produce quadratic complexity); (2) `2-Spheres`, a small sphere lies inside a big sphere; (3) `Head`, a set of sample points from the skin of a human head plus a bounding box (see Fig. 3.16 left), and (4) `Hose`, the point set of the surface mesh model (shown in Fig. 3.16 middle) plus a bounding box.

	Line-and-Circle		2-Spheres		Head		Hose	
$\mathcal{D}_{ini}$	2056	1004079	3076	13551	4698	28252	6008	40827
$\mathcal{D}_{O1}$	14111	86638	4412	21253	7494	44724	10126	61363
$\mathcal{D}_{O2}$	*12522*	*77098*	*4226*	*20042*	*7038*	*41770*	*9496*	*57185*
$\mathcal{D}_{O3}$	13674	86236	4294	20714	7118	42310	9597	57914
$\mathcal{D}_{O4}$	$--$	$--$	4464	21565	7656	45745	10457	63766
$\mathcal{D}_{O5}$	12924	79868	4292	20475	7165	42757	9827	59991

Table 3.3: Comparison of the output sizes (nodes and tetrahedra) of Delaunay refinement algorithm under different point insertion orders, i.e., $O1, ..., O5$ (defined in Table 3.2). Where $\mathcal{D}_{ini}$ means the initial size of the Delaunay tetrahedralization, $\mathcal{D}_{O1}, ..., \mathcal{D}_{O5}$ are the corresponding output sizes. In each column, the smallest size is shown in italic.

PLSs Next we compare the behavior of the Delaunay refinement on PLSs. In all the tested insertion orders, we unconditionally give higher priorities to the midpoints of encroached subsegments and the circumcenters of encroached subfaces. Table 3.4 shows a comparison of the output sizes (nodes and tetrahedra) for using different insertion orders on four data sets: (1) `DEMO03` (see Fig. 1.3); (2) `Pmdc` (see Fig. 3.16 Right); (3) `Thepart` (see Fig. 3.4 Top-Left); and (4) `Hose` (see Fig. 3.16 Middle).

	DEMO03		Pmdc		Thepart		Hose	
$\mathcal{D}_{ini}$	368	1116	972	5704	994	6413	6000	39701
$\mathcal{D}_{O1}$	2960	12150	*12050*	*43219*	7954	35076	19942	73730
$\mathcal{D}_{O2}$	3067	12019	12691	44958	*7379*	31182	*19933*	*72617*
$\mathcal{D}_{O3}$	*2912*	*11297*	12756	46102	7442	*31169*	20017	72801
$\mathcal{D}_{O4}$	2937	12105	12194	43859	8002	35251	21651	82922
$\mathcal{D}_{O5}$	2948	11773	12641	45430	7653	32747	20768	77921

Table 3.4: Comparison of the output sizes (nodes and tetrahedra) of Delaunay refinement algorithm under different point insertion orders, i.e., $O1, ..., O5$ (defined in Table 3.2). Where $\mathcal{D}_{ini}$ means the initial size of the Delaunay tetrahedralization, $\mathcal{D}_{O1}, ..., \mathcal{D}_{O5}$ are the corresponding output sizes. In each column, the smallest size is shown in italic.

Comparison In all the above tests, the bad quality tetrahedra (the circumcenters) were sorted strictly according to the chosen order. For point clouds, the "short-edge-frist" ($O2$), "small-radius-first" ($O3$), and "random" ($O5$) orders are similar and produce smaller output sizes. The "large-radius-edge-ratio-first" ($O1$) and "largest-radius-first" ($O4$) orders always have bigger output sizes. For PLSs, the difference becomes complicated because of the interactions with the boundary. For the `Pmdc` model, which has rela-

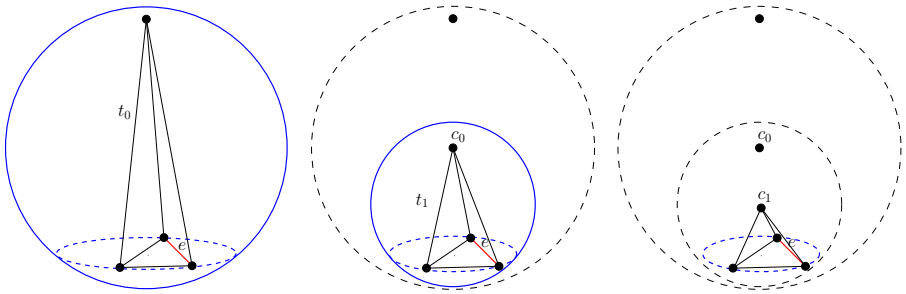


Figure 3.17: Left: Edge e is relatively short, since the tetrahedron $t_0 \ni e$ has large radius-edge ratio. Middle: c_0 is the circumcenter if t_0. e is relatively short because $t_1 \ni e$ has large radius-edge ratio. Right: c_1 is the circumcenter of t_1. e is not relatively short.

tively complicated interior boundaries, the output sizes are smaller by $O1$ and $O4$, but are bigger for others.

3.3.6 Output Size

We have shown that the output size of Delaunay refinement depends on the point insertion order. Hence the complexity of the algorithm will correspond to the worst insertion order. Unfortunately, it is very difficult to decide such an order due to the interaction with PLS boundaries. In this section, we study the output size (the total number of nodes and tetrahedra) of a specialized Delaunay refinement algorithm with no interaction with boundaries and a modest assumption on the input.

We assume that the input is only a finite set S of points in $\mathbb{R}^3$. Furthermore, we assume that S is a *periodic point set* [19], i.e., $S \subseteq [0,1)^3$ is a finite set of points, and duplicated within each integer unit cube: $S' = S + \mathbb{Z}^3$, where $\mathbb{Z}^3$ is the three-dimensional integer grid. For refining such a point set, only the rule $R3$ is needed, i.e., to generate the circumcenters of bad quality tetrahedra. Let L denote the diameter of S, while s denotes the smallest pairwise distance among input features, the ratio $\Delta = L/s$ is referred as *spread* of S [41].

Let $\mathcal{D}(S)$ be the initial Delaunay tetrahedralization. An edge $e \in \mathcal{D}(S)$ is *relatively short* if e is the shortest edge of a tetrahedron $t \in \mathcal{D}(S)$ and t has radius-edge ratio larger than ρ_0, see Fig. 3.17 Left. Obviously, not all edges of $\mathcal{D}(S)$ can be relatively short.

A modified Delaunay refinement algorithm using a specified point insertion order is shown in below. The new points are iteratively inserted. At each iteration i, it first collects a queue Q of relatively short edges in the current $\mathcal{D}(S)$ (line 3). Then new points are generated and added into $\mathcal{D}(S)$ until Q is empty (lines from 4 to 11).

```
DelaunayRefine2 (S, ρ0)
// S is a periodic point set; ρ0 is a radius-edge ratio bound.
 1. initialize a Delaunay tetrahedralization D of S;
 2. i = 0;
 3. while there exists relatively short edges, do
 4.     form a queue Q of all relatively short edges in D;
 5.     while Q ≠ ∅, do
 6.         pop an alive edge e from Q;
 7.         choose a tet t ∋ e, such that r_t/|e| > ρ0, set v = c_t;
 8.         update D to be a DT of vert(D) ∪ {v};
 9.         if e is still a relatively short edge of D, then
10.             push e back into Q;
11.         endif
12.     endwhile
13.     i = i + 1;
14. endwhile
15. return D.
```

Figure 3.18: A modified Delaunay refinement algorithm.

It is easy to show that the total number of the iterations needed by the above algorithm is limited as long as $\rho_0 > 1$.

Lemma 3.3.18 *If $\rho_0 > 1$, the Delaunay refinement algorithm takes at most $k = \lceil \log_{\rho_0} \Delta \rceil$ iterations.*

Proof At the first iteration, $i = 0$, the shortest edge length in Q is s. Since the algorithm only creates new edges that are at least of length $s\rho_0$, after the first iteration, the shortest edge length in Q increases to $s\rho_0$. By induction, after jth iteration ($j > 0$), the shortest edge length is at least $s\rho_0^j$. Clearly, there are at most $k = \lceil \log_{\rho_0} \Delta \rceil$ iterations. ■

The following lemma shows that in each iteration, the total number of added points is bounded by the initial size of the queue.

Lemma 3.3.19 *In each iteration i, where $0 \leq i < k$. Let m_i denote the initial size of Q, The algorithm adds at most $O(m_i \log_2 \Delta)$ points.*

Proof Let e be a relatively short edge in Q, and let $t_0 \ni e$ be a bad quality tetrahedron (Fig. 3.17 Left). The worst radius-edge ratio of t_0 is no larger than $\Delta = L/s$. t_0 is removed by adding its circumcenter c_{t_0}. Let t_1 be the tetrahedron $t_1 \ni e$, created by the insertion of c_{t_0} (Fig. 3.17 middle). t_1 may still be bad, however, it's radius-edge ratio is not larger than $\frac{1}{2}\Delta$. It is easy

to see that after the insertion of at most $\log_2 \Delta$ new points, e has at least one close vertex. If e is still in Q, then again after at most $\log_2 \Delta$ new point insertions e gets another close vertex. By Theorem 3.3.16, e can have at most a constant number of close points. Hence it needs at most $O(\log_2 \Delta)$ new points to remove e from Q. Since there are total m_i edges, then it needs at most $O(m_i \log_2 \Delta)$ new points. ■

An Upper Bound on the Average Output Size

Let S be a set of n periodic points in $\mathbb{R}^3$. Assume that $\mathcal{D}(S)$ has complexity $O(n)$, i.e., the number of edges of $\mathcal{D}(S)$ is $O(n)$, and $\rho_0 > 1$. We estimate the total number of output size produced by the algorithm given in this section. We will show that the average output number of nodes is bounded by the parameters which only depend on the input data.

By Lemma 3.3.18, there are at most $k = \lceil \log_{\rho_0} \Delta \rceil$ iterations. Let m_i denote the initial size of Q at iteration i. By repeatedly applying Lemma 3.3.19, the total number n_{total} of new points is within

$$n_{total} = O((m_0 + m_1 + \cdots + m_{k-1}) \log_2 \Delta). \tag{3.29}$$

Note that the above iterations are nothing else a packing process. At the beginning, there are m_0 edges with length $\geq s$, after iteration i, $i = 1, ..., k$, it packs m_i new edges with length $\geq s\rho_0^i$ into the volume. If we estimate the volume occupied by each edge to be l^3, where l is the edge length, the total volume is less than L^3. Then $m_0 s^3 + m_1(s\rho_0)^3 + m_2(s\rho_0^2)^3 + \cdots + m_k(s\rho_0^k)^3 \leq C\, L^3$, where C is a bounded constant, i.e.,

$$m_0 + m_1\rho_0^3 + m_2\rho_0^6 + \cdots + m_k\rho_0^{3k} \leq O(\Delta^3). \tag{3.30}$$

Combining (3.29) and (3.30) we get an upper bound for n_{total}, which is

$$n_{total} = O(\Delta^3 \log_2 \Delta). \tag{3.31}$$

Note that the above bound does not reflect both the initial size n and ρ_0. We further extend (3.29).

After iteration i, the number of new points is $O(m_i \log_2 \Delta)$. All new edges (including relatively short edges) are connecting to the new points. Since the mesh has linear complexity, it can have at most

$$m_{i+1} \leq D_i\, m_i \tag{3.32}$$

relatively short edges, for some constant D_i. (3.30) implies that D_i is related to ρ_0 and $D_i \sim O(\frac{1}{\rho_0^3})$. Let $D = \max\{D_i | i = 1, ..., k\}$, D is a bounded constant.

Inserting (3.32) into (3.29) we get

$$\begin{aligned} n_{total} &= O((1 + D_1 + \cdots + \prod_{i=1}^{k-1} D_i) m_0 \log_2 \Delta) \\ &= O(kD^k \, m_0 \log_2 \Delta) \\ &= O(n \lceil \log_{\rho_0} \Delta \rceil \log_2 \Delta). \end{aligned} \tag{3.33}$$

Equation (3.33) implies that the total number of output nodes is bounded by n, Δ, ρ_0, which are all input parameters.

Table 3.5 shows some tests on the relations between $\log_{\rho_0} \Delta$ and the output sizes. The results of this experiment suggest that this upper bound is far more tight.

PLS	n_{in}	ρ_0	Δ	$\log_{\rho_0} \Delta$	n_{out}	n_{add}
R_{10}	772	2.0	130.10	7.022	1085	313
R_{100}			1301.08	10.350	1153	381
R_{1k}			13010.80	13.670	1208	436
R_{10}	772	1.4	130.10	14.050	1586	814
R_{100}			1301.08	20.700	1821	1049
R_{1k}			13010.80	27.345	2034	1262
R_{10}	772	1.2	130.10	26.700	2560	1788
R_{100}			1301.08	39.330	3380	2608
R_{1k}			13010.80	51.960	4081	3309

Table 3.5: Tests on the relations between the input parameters ρ_0, Δ and the output sizes. R_{10}, R_{100}, and R_{1k} are three PLSs. R_{10} is formed by two spheres, one inside the other, the outer sphere has radius $r = 10$ (see Fig. 3.19). R_{100} and R_{1k} are obtained from R_{10} by scaling the outer sphere to the radii $r = 100$ and $r = 1000$, respectively. n_{in}, n_{out}, and n_{add} are the number of input nodes, output nodes, and added nodes, respectively.

We assume that the initial $\mathcal{D}(S)$ has linear complexity. We conjecture that this assumption is not necessary. Bern *et al.* [9] showed that any Delaunay tetrahedralization can be reduced to $O(n)$ complexity by adding $O(n)$ Steiner points, see also [15]. The example "Line-and-Circle" for instance, after the first iteration, $\mathcal{D}(S)$ shows linear complexity.

3.4 Improvement

There are two major problems (and big challenges as well) that prevent the three-dimensional Delaunay refinement algorithm from being practical: first, the input angle restriction 3.3.3 limits the class of models that can be processed; second, the resulting meshes will contain slivers.

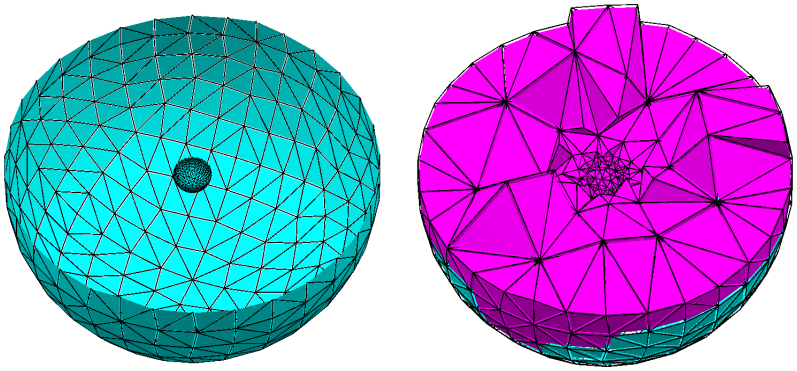


Figure 3.19: Left: The PLS R_{10} (772 nodes, 1536 faces), formed by two spheres, one inside the other, the outer sphere has radius $r = 10$, the inner one has $r = 1$. Right: The quality boundary conforming Delaunay mesh (1085 nodes, 5121 tetrahedra) by using $\rho_0 = 2$.

In this section, we first propose a relaxed input angle assumption, that is, we remove the 60° restriction on input segment-segment angles. Then we show that the Delaunay refinement algorithm will terminate with a slightly modified first point insertion rule (e.g., $R1^*$). The result implies that the three-dimensional Delaunay refinement algorithm is able to generate quality-guaranteed boundary conforming Delaunay meshes for a wider class of inputs. Next we present experimental results which show that the Delaunay refinement algorithm is indeed able to generate good shaped meshes without slivers, despite of the lack of a termination guarantee.

3.4.1 Relaxed Input Angle Assumption

The Input Angle Assumption (given in Section 3.3.3) requires that no segment-segment angle is smaller than 60°. This limitation is too strong in practice. For instance, many three-dimensional CAD models are represented by surface meshes. Even an extremely good-quality surface mesh made up by triangles may not satisfy this condition, refer to Fig. 3.16. In the following, we propose a Relaxed Input Angle Assumption.

Assumption 3.4.1 (Relaxed Input Angle) *Let $\mathcal{X}$ be a three-dimensional PLS. Assume that $\mathcal{X}$ contains* (1) *no segment-facet angle smaller than* 60°, *and* (2) *no facet-facet angle smaller than* $\arccos \frac{1}{2\sqrt{2}} \approx 69.3°$.

The new assumption accepts arbitrarily small input segment-segment angles. The condition *(1)* of this assumption is not necessary for most of the practical inputs. It is kept only for the proof of the termination. Only the facet-facet dihedral angle restriction is necessary in practice.

Modified Point Generation Rule Now arbitrarily small segment-segment angles are allowed, a segment is called *sharp* if it forms an angle strictly less than 60° with an incident segment. Sharp segments are easy to detect and they have to be treated differently. The idea is fairly simple: Do not split a sharp segment if it is boundary conforming Delaunay. Below is the modified first point generating rule.

$R1^*$ If a segment S is encroached. Let $\mathbf{v}$ be its midpoint. $\mathbf{v}$ may be inserted in one of the following two cases:

(1) S is not sharp, or

(2) S is sharp, and the cause of splitting S is an existing mesh vertex.

Otherwise, do not insert $\mathbf{v}$.

The case *(2)* of $R1^*$ implies two sub-cases, which are, *(a)*: S is encroached by an existing mesh vertex, and *(b)*: there is a subface F which is encroached by an existing mesh vertex, and the circumcenter of F encroaches S.

Termination and Mesh Quality Now we show that by using *(1)*: the Relaxed Input Angle Assumption, *(2)*: the modified point generation rule $R1^*$, and *(3)*: the Input Segment Assumption (given in Section 3.3.3). The Delaunay refinement algorithm given in Table 3.2.3 will terminate.

Theorem 3.4.2 *If $\mathcal{X}$ satisfies both the Relaxed Input Angle Assumption and the Input Segment Assumption and $\rho_0 > \sqrt{2}$. Assume that the $R1^*$ is used in stead of $R1$. The Delaunay refinement algorithm terminates.*

Proof The basic proof is given for Theorem 3.3.14. The exception here is the existence of input sharp segments. One needs to show that they don't cause infinite point insertion. The termination guarantee is by the using of $R1^*$ which sacrifices the quality of the mesh.

By the Input Segment Assumption, incident sharp segments must have an equal length. The split of one sharp segment will also cause the split of its incident sharp segments (since they form an angle strictly less than 60°), and after splits, the lengths of both subsegments are equal. This guarantees no infinite loop can happen after the split of two incident sharp segments. There are only finitely many splits on subsegments. Similarly we can show that $R1^*$ and $R2$ together guarantee all encroached subfaces will be split and the number of splits are finite. Hence, this algorithm guarantees that the output mesh is boundary conforming Delaunay.

Finally, removing the bad-quality tetrahedra will terminate in finite time. Since it is forbidden by $R1^*$ to split a sharp segment s which is also

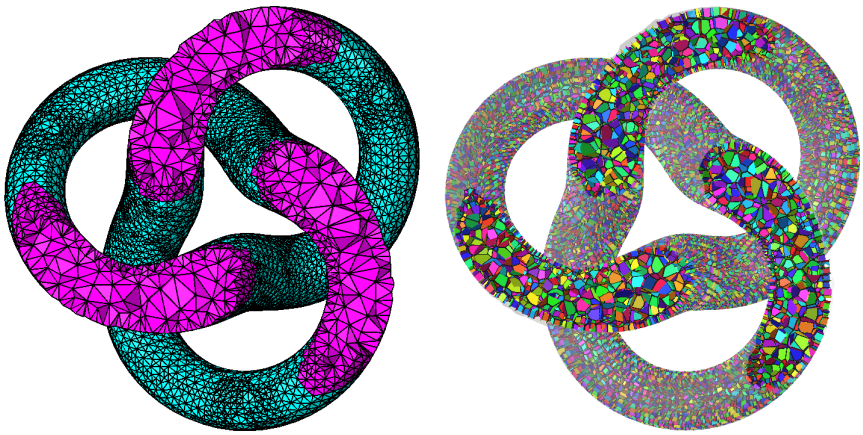


Figure 3.20: Left: A boundary conforming Delaunay mesh of a three-dimensional PLS (`hose_2_3`). The minimum input segment-segment angle is 24°, the minimum input face-facet dihedral angle is 154°. The mesh was generated by `TetGen` [110] using the modified Delaunay refinement algorithm. Right: The dual Voronoi diagram of the left mesh. The Voronoi faces are randomly colored.

boundary conforming, some badly-shaped tetrahedra main survive. However, these tetrahedra must be close to the sharp segments. Let t be such a tetrahedron, $\mathbf{c}_t$ be its circumcenter, and $\mathbf{c}_s$ be the midpoint of s, then $\|\mathbf{c}_t - \mathbf{c}_s\| \leq \frac{1}{2}(3+\sqrt{2})\,\|S\|$. ■

Remark. In the above theorem, the Input Segment Assumption is used. It is easy to satisfy (by simply splitting input segments). This may cause unnecessarily many points to be inserted. In Chapter 5, a set of segment splitting rules are proposed, which can protect sharp segments adaptively. In the Modified Point Generation Rule, instead of inserting the midpoint, use of these rules, the sharp segments will be automatically protected. Hence the Input Segment Assumption is not needed anymore. Our implementation verifies that this combination works very well in practice, i.e., a quality boundary conforming Delaunay mesh will be generated for inputs containing no small dihedral angles but having arbitrary small face angles. The number of additional points for protecting sharp segments is small. An example is shown in Fig 3.20.

3.4.2 Experiments on Sliver Removal

Another remaining theoretical problem of the Delaunay refinement algorithm is that the resulting meshes may contain slivers which may have arbitrary large aspect ratio. Fig. 3.21 highlights the remaining slivers in two

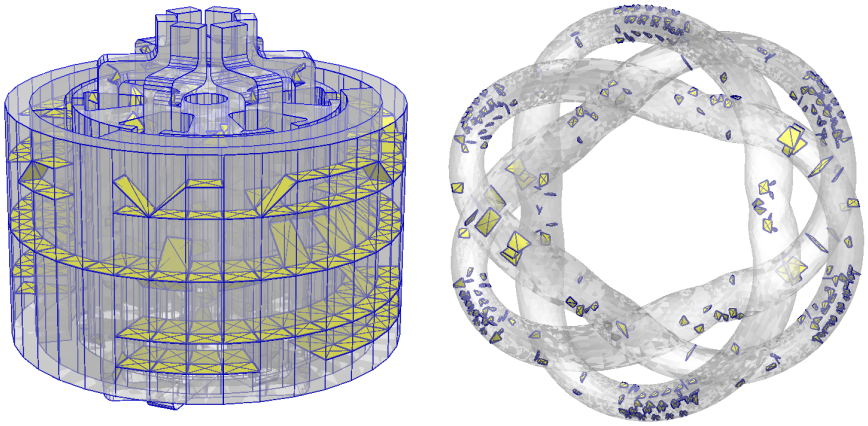


Figure 3.21: The remaining slivers in bounded radius edge ratio meshes are highlighted. Left: Pmdc, and Right: Hose.

bounded radius-edge ratio meshes.

Many algorithms have been proposed that take as input a good radius-edge ratio tetrahedral mesh, and refine it into a sliver-free, good aspect ratio mesh [25, 19, 37, 70]. Only the *Sliver Exudation* algorithm of Cheng *et al.* [19] is implemented and experimental results are reported. Besides the theoretical work, heuristic methods [65, 46] are proposed too. In particular, Shewchuk [101] showed experiments that Delaunay refinement indeed is effective in removing slivers. The purpose of this section is to give more quantitative evidences, that Delaunay refinement can generate boundary conforming Delaunay meshes with bounded aspect ratio.

Method Description The minimum dihedral angle, ϕ_{min}, is used to identify slivers. As suggested in [36], we fix a threshold and call a tetrahedron a *sliver* if its $\phi_{min} < 5°$. Assume a bounded radius-edge ratio Delaunay mesh containing slivers to be given.

Starting from that mesh, each sliver is removed by inserting a point at its circumcenter. If a circumcenter encroaches upon any subsegment or subface, it is rejected, and the circumcenters of the encroached boundaries are inserted. A new inserted point may create new large radius-edge ratio ($> \rho_0$) tetrahedra and new slivers, they are processed by the same way. The tetrahedra with large radius-edge ratios have priority over slivers. The process repeats until neither sliver nor bad quality tetrahedra exist. This method is essentially the same as that of Li and Teng [70]. However, we omit using *picking regions* and allow the creation of *small slivers*. Hence the termination is not theoretically guaranteed.

	Line & Circle	2-Spheres	Head	Hose
In Nodes	14111	4288	7331	9993
In Tets	86638	20643	43824	60818
In Min ϕ	0.51°	0.28°	$8.5e-07$°	0.20°
In Max ϕ	178.76°	179.40°	179.99°	179.56°
In Max η	143.4	333.3	$6.0e+14$	346.8
Slivers	*298*	*118*	*324*	*453*
Steiner Points	*397*	*151*	*431*	*532*
Out Min ϕ	5.01°	5.02°	5.00°	5.01°
Out Max ϕ	171.50°	171.91°	172.73°	171.89°
Out Max η	29.1	28.1	35.1	25.3

	DEMO03	Pmdc	Thepart	Hose
In Nodes	2960	12050	7954	19941
In Tets	12150	43219	35076	73720
In Min ϕ	0.007°	0.001°	0.002°	0.324°
In Max ϕ	179.98°	179.99°	179.99°	179.09°
In Max η	9435.9	71128	37316.0	194.1
PLS Facets	744	502	1995	12000
Slivers	*110*	*629*	*363*	*324*
Steiner Points	*349*	*1607*	*1075*	*691*
Out Min ϕ	5.01°	5.01°	5.02°	5.03°
Out Max ϕ	172.01°	175.60°	171.91°	171.94°
Out Max η	22.2	25.5	28.1	21.9

Table 3.6: Sliver removal experiments on meshes from point clouds (top) and PLS models (bottom). Both the initial and final meshes have a bounded radius-edge ratio below 2. The number of slivers and the inserted Steiner points are highlighted. Initial and final mesh quality are reported by the min-max dihedral angle (ϕ) and the maximum aspect ratio (η).

Results and Discussion Table 3.6 show the experiments on point clouds and PLS models. The test data sets are the same as those in Section 3.3.5. The PLS models satisfy the Input Angle Assumption 3.3.3.

Li and Teng claim that their algorithm needs at most $O(n)$ Steiner points to generate a sliver-free Delaunay mesh, where n is the number of input nodes of the initial bounded radius-edge ratio mesh [70]. For point cloud inputs, i.e., Table 3.6, one can observe that the number of initial slivers and the number of Steiner points are similar, the latter is only slightly larger. However, the difference between these two numbers gets larger for PLS models (see Table 3.6), but they are still in the same order. Our tests suggest that the number of initial slivers is crucial. To find the relation between the numbers of initial slivers and Steiner points will be meaningful.

3.5 Summary

In this chapter we defined boundary conforming Delaunay meshes and studied a Delaunay refinement algorithm for generating such meshes with good quality and well distributed mesh size. We performed a reanalysis on the basic scheme of this algorithm. The main reason was the observed large discrepancies between theory and practice. The gain of this reanalysis is a better understanding of the basic scheme, and a variety of improvements to the algorithm and its analysis. Our new results are summarize below.

• The original restriction on angles between two facets of the input can be reduced from the 90° to $\arccos \frac{1}{2\sqrt{2}} \approx 69.3^\circ$. The proof is based on the constructions of two extreme cases (shown in Fig. 3.7 (c) and (d)) which bound the limiting dihedral angle and the insertion radii, respectively. The formula (3.1) describing the relation between insertion radii and acute angle was first proved by Pav (In [90] Lemma 3.1.4) in the planar case.

• By using a different approach to analyze the insertion radii, new lower bounds on the shortest output edge lengths are derived. The new bounds can be represented by one formula, such that the constant for each edge is different depending on the location and the insertion time of its endpoints. The derived constants for edges in interior, on facets, and on segments have the denominators, $\rho_0 - 1$, $\rho_0 - \sqrt{2}$, and $\rho_0 - 2$, respectively. Pav [90] first used this approach in re-analyzing Ruppert's algorithm [92].

• Benefit from the reanalysis of the output edge lengths, we proved that the original lower bound on the tetrahedral shape, i.e., the radius-edge ratio, can be reduced from 2.0 to $\sqrt{2}$ if all adjacent non-collinear segments have equal length. Furthermore, our proof shows that this requirement of equal length is unlikely needed in practice.

• We proved a new upper bound for the vertex degree in a mesh whose all tetrahedra have a radius-edge ratio bounded. The proof is based on the fact that each edge e of such mesh can form an empty cone (no other edge inside) with e as the axis, one of its endpoints as apex, and the largest cone angle only depends on the radius-edge ratio.

• We investigated different point insertion orders on both point cloud and PLS inputs for analyzing the complexity of the algorithm. For point cloud inputs, our experiments showed that the "short-edge-first" and "small-radius-first" orders always result smaller output sizes. While the "large-radius-first" order results output sizes which are always larger. In between there are "large-radius-edge-ratio-first" and "randomized" orders. However, the cases are more complicated for PLS models.

The analysis of the Delaunay refinement is far from completed. Some issues are still widely open and worth to be investigated in the future.

◦ Our experiments on sliver removal suggests that Delaunay refinement is able to generate a sliver-free quality mesh for PLSs satisfying the input angle restriction. Is it possible to prove a non-trivial output dihedral angle?

◦ The space and time complexities of the discussed algorithm for input PLSs remain open. The difficulties are mainly in two issues: the choice of a point insertion order, and the interaction with the PLS boundaries. Even for inputs which are periodic point sets, the average upper bound given in Section 3.3.6 appears too big comparing with the experimental results.

Chapter 4

Adaptive Delaunay Mesh Generation

In this chapter, we consider an adaptive meshing problem. Here the central questions are how to efficiently distribute new mesh points and to conform the boundary simultaneously.

A variant of the Delaunay refinement algorithm is presented. The algorithm starts with an initial (coarse) tetrahedral mesh of a domain. It place new points in the domain by modified point generating rules. The control of the mesh size distribution is through either the input geometric data or a user-defined mesh sizing function. The mesh conformity can be guaranteed for smooth sizing functions. Most of the output tetrahedra have guaranteed quality. Possibly remaining badly-shaped tetrahedra are well-located in the vicinity of the small input angles. Application examples are given to illustrate the aspects of this algorithm. Part of this chapter will appear in a future issue of IJNME [111].

This chapter is organized as follows. The proposed algorithm is presented in detail in Section 4.1. In Section 4.2 we analyze the conditions on mesh quality and mesh conformity. Some practical aspects, e.g., the efficiency and robustness, are demonstrated by some application examples in Section 4.3. We end this chapter with a summary in Section 4.4.

4.1 Constrained Delaunay Refinement

In this section, we present an algorithm which makes the use of the Delaunay refinement technique to conform the domain boundary and create field points simultaneously. The algorithm is analyzed in the next section.

4.1.1 Mesh Sizing Functions

Let $\mathcal{X}$ be a three-dimensional PLS. Define a *mesh sizing function* $H : |\mathcal{X}| \rightarrow \mathbb{R}$, such that for each point $\mathbf{p} \in |\mathcal{X}|$, $H(\mathbf{p})$ specifies the desired length of edges to the vertex inserted at the location $\mathbf{p}$. For example, the *local feature size* $\text{lfs}(\mathbf{p})$ at a point $\mathbf{p} \in |\mathcal{X}|$ is defined as the radius of the smallest ball centered at $\mathbf{p}$ that intersects two non-incident polytopes of $\mathcal{X}$. lfs defines a default distance field on $|\mathcal{X}|$ based on its boundary information.

H is *isotropic* if the edge length does not vary with respect to the directions at $\mathbf{p}$, otherwise, it is *anisotropic*. Generally, H can be represented by a 3×3 metric tensor M which defines a field of symmetric positive definite matrices. In isotropic case, for any $\mathbf{p} \in |\mathcal{X}|$, $M(\mathbf{p}) = \frac{1}{H^2(\mathbf{p})} I_3$, where I_3 is the identity matrix in $\mathbb{R}^{3\times 3}$. For a more detailed discussion of the anisotropic case, see e.g., [47]. In the scope of this work, we assume that H is isotropic. An ideal sizing function is C^∞, $\forall \mathbf{p} \in |\mathcal{X}|$. However, in most cases, H is approximated by a discrete function specified at some points in $|\mathcal{X}|$. The size on other points is obtained by means of interpolation. A background mesh can be used for this purpose.

One of our goals is to create a tetrahedral mesh $\mathcal{T}$ of $\mathcal{X}$ such that the mesh size of $\mathcal{T}$ conforms to H. The conformity of the mesh size can be measured by the following criterion. Let $\mathbf{p}$ be a vertex in $\mathcal{T}$. Let $S(\mathbf{p})$ and $L(\mathbf{p})$ denote the shortest and longest edge lengths at $\mathbf{p}$, respectively. We say that the size of $\mathcal{T}$ *conforms* to H if there exist two constants C_S and C_L, where $0 < C_S \leq C_L < \infty$, such that for every vertex $\mathbf{p} \in \mathcal{T}$, the following relation holds

$$C_S \leq \frac{S(\mathbf{p})}{H(\mathbf{p})} \leq \frac{L(\mathbf{p})}{H(\mathbf{p})} \leq C_L.$$

The best conformity would be the case $C_S = C_L = 1$. This is just the definition of the *unit mesh* in [47]. Except for very regular geometries, it is generally not possible to obtain the best conformity. One goal is to minimize the ratio $\frac{C_L}{C_S}$.

4.1.2 Sparse and Protecting Balls

Starting with a Delaunay tetrahedralization, the classical Delaunay refinement scheme uses three rules to add new points. One adds a point $\mathbf{v}$ at the circumcenter of a badly-shaped tetrahedron τ. τ will be removed after reconnecting the local mesh edges to $\mathbf{v}$ using the Delaunay criterion. The

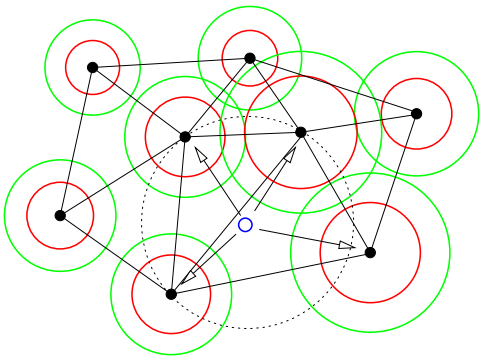


Figure 4.1: For each point $\mathbf{p}$ of $\mathcal{T}$, assume there are two virtual balls, one protect ball (shown in red), and one sparse ball (shown in green).

other two rules are used for boundary protection, i.e., split a segment or subface in the boundaries of $\mathcal{X}$ by adding its circumcenter. Boundary protection has higher priority than removing badly-shaped tetrahedron. The point insertion continues until no badly-shaped tetrahedron remains.

Our algorithm makes use of these rules. While two important variants are made: *(1)* try to insert a new point at the location where the local mesh is either badly-shaped or "sparse", the sparseness is indicated by the sizing function at mesh vertices, and *(2)* the new point can be inserted only if it is not "close" to any existing vertex. Intuitively, one can assume that each vertex $\mathbf{p}$ of the mesh is surrounded by two virtual balls, one *sparse ball* with radius $\alpha_1 H(\mathbf{p})$, and one *protecting ball* with radius $\alpha_2 H(\mathbf{p})$ (see Fig. 4.1). The space outside the sparse ball of $\mathbf{p}$ is *sparse* from the viewpoint of $\mathbf{p}$, while any point inside the protecting ball of $\mathbf{p}$ is *close* to $\mathbf{p}$.

The two parameters α_1 and α_2 are used to scale the size of the balls. In particular, we get the basic Delaunay refinement scheme by setting $\alpha_1 = \infty$ (no sparse ball), and $\alpha_2 = 0$ (no protect ball).

4.1.3 Point Generating Rules

A candidate $\mathbf{v}$ for insertion is found by the basic Delaunay refinement scheme, i.e., $\mathbf{v}$ is found by the following three *point generating rules*.

$R1$ If a subsegment is encroached, then let $\mathbf{v}$ be its midpoint.

$R2$ If a subface is encroached, then let $\mathbf{v}$ be its circumcenter. However, if $\mathbf{v}$ encroaches upon some subsegments, then reject $\mathbf{v}$. Instead, use $R1$ to return a $\mathbf{v}$.

$R3$ If a tetrahedron τ satisfies one of the following two cases, $R3.1$ or $R3.1$:

$R3.1$ τ has a radius-edge ratio greater than ρ_0.

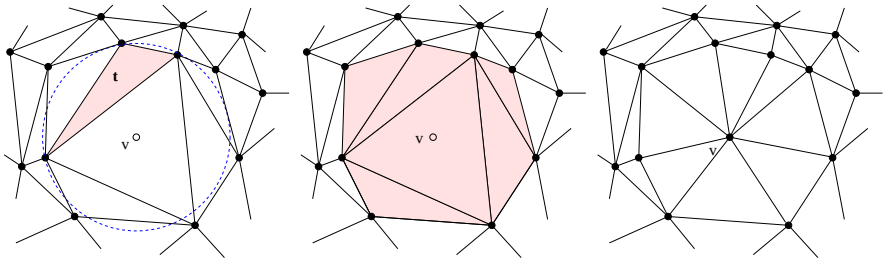


Figure 4.2: Left: $\mathbf{v}$ is a point found by $R2$. Middle: the points of the shaded region form the set P of points collected by the point accepting rule. Right: if $\mathbf{v}$ get inserted, the shaded region is re-triangulated according to the Delaunay criterion.

$R3.2$ there is a vertex $\mathbf{p}$ of τ, such that $\alpha_1 H(\mathbf{p}) < r$, where r is the radius of the circumscribed ball of τ,

then $\mathbf{v}$ is inserted in the circumcenter of τ. However, if $\mathbf{v}$ encroaches upon any subsegment or subface, then reject $\mathbf{v}$. Instead, use $R1$ or $R2$ to return a $\mathbf{v}$.

$R3.1$ tests if τ has bad quality, and $R3.2$ checks the H-conformity of the vertices of τ. $R3.1$ has a priority higher than $R3.2$. Hence $R3.2$ is triggered only if all tetrahedra have radius-edge ratio greater than ρ_0.

Unlike the Delaunay refinement algorithm in which the candidate $\mathbf{v}$ is always inside the mesh domain or on the boundary, here $\mathbf{v}$ may lie outside the mesh domain. The reason is that some candidates may not be inserted if they are too close to existing mesh vertices, hence the mesh may not boundary conforming Delaunay.

4.1.4 Point Accepting Rules

Once a candidate $\mathbf{v}$ is found, $\mathbf{v}$ is not inserted immediately. Instead, the *point accepting rule* will be called. It decides whether or not $\mathbf{v}$ can be inserted into the mesh. Let V be a set of vertices collected as follows:

- If $\mathbf{v}$ is found by $R1$, then V contains two endpoints of the segment that $\mathbf{v}$ will split.
- If $\mathbf{v}$ is found by $R2$, then V contains vertices of the subfaces whose diametrical circumspheres are encroached by $\mathbf{v}$ (see Fig. 4.2).
- If $\mathbf{v}$ is found by $R3$, then V contains vertices of the tetrahedra whose circumspheres contain $\mathbf{v}$.

Then $\mathbf{v}$ can be inserted if $\alpha_2 H(\mathbf{p}) < \|\mathbf{v} - \mathbf{p}\|$ for all $\mathbf{p} \in V$. Otherwise, $\mathbf{v}$ is not inserted.

```
AdaptiveDelaunayRefinement (𝒯, ρ0, H, α1, α2)
// 𝒯 is a tetrahedral mesh of a PLS 𝒳; ρ0 is a radius-edge ratio bound;
// H : |𝒳| → ℝ is a sizing function; α1, α2 are two parameters.
1.  initialize a queue Q of all tetrahedra of 𝒯;
2.  while Q ≠ ∅, do
3.    pop a tetrahedron τ from Q;
4.    if (ρ(τ) > ρ0) or (∃p < τ, ‖c_τ − p‖ > α1 H(p)), then
5.      create a vertex v by R3 (or R1 or R2);
6.      if v ∈ |𝒯|, then
7.        collect vertices in P by the point accepting rule;
8.        if ∀p ∈ P, ‖v − p‖ > α2 H(p), then
9.          update 𝒯 to be the mesh of 𝒯 ∪ {v};
10.         if v ≠ c_τ, then; Q = Q ∪ {τ}; endif
11.         Q = Q ∪ {ν ∈ 𝒯 | v < ν};
12.       endif
13.     endif
14.   endif
15. endwhile
16. return 𝒯;
```

Figure 4.3: The Adaptive Delaunay refinement algorithm.

In the point accepting rule, if $\mathbf{v}$ is found by $R1$ or $R2$, only the endpoints of the subsegment or subfaces of the same facet on which $\mathbf{v}$ lies have the right to accept or reject $\mathbf{v}$. $\mathbf{v}$ may be very close to some existing vertices, i.e., there exist a point $\mathbf{q} \notin V$, such that $\alpha_2 H(\mathbf{q}) > \|\mathbf{v} - \mathbf{q}\|$. However, we will show in the next section that the distance $\|\mathbf{v} - \mathbf{q}\|$ is always bounded.

4.1.5 The Algorithm

The AdaptiveDelaunayRefinement algorithm is described in Fig. 4.3. It first initialize a queue Q of all tetrahedra of $\mathcal{T}$. Then it runs into a loop until Q is empty. At each step, a tetrahedron τ is removed from Q (line 3). Let $\mathbf{c}_\tau$ be its circumcenter. If τ is badly-shaped or the space at $\mathbf{c}_\tau$ is sparse (line 4), then a new point $\mathbf{v}$ is generated (line 5). Since $\mathcal{T}$ may not be a boundary conforming Delaunay mesh, $\mathbf{v}$ may lie outside $|\mathcal{T}|$, $\mathbf{v}$ can be considered for insertion if $\mathbf{v} \in |\mathcal{T}|$ (lines $6-14$). Finally, $\mathbf{v}$ can be inserted if it passes the point insertion rule (lines $8-12$). The new mesh of $\mathcal{T} \cup \{\mathbf{v}\}$ is created by the Delaunay criterion (line 9). If $\mathbf{v}$ was generated by $R1$ or $R2$, i.e., $\mathbf{v} \neq \mathbf{c}_\tau$, the insertion of $\mathbf{v}$ may not delete τ, τ is queued in Q for later process (line 10). All the newly generated tetrahedra are added into Q as well (line 11).

4.2 Analysis

In the following, we provide conditions on the sizing function H, and the parameters α_1, α_2, and ρ_0 to ensure the theoretical guarantees of our algorithm. Specifically, we will show: (1) The termination of the algorithm only depends on α_2; (2) The mesh quality is governed by both ρ_0 and α_2; (3) The mesh conformity and mesh size depend on α_1, α_2, and H.

4.2.1 Proof of Termination

The termination of this algorithm can be proved by showing that for any newly inserted vertex, the distance to its nearest mesh vertex is bounded by some positive value. Then the algorithm will stop since no arbitrarily short edge can be introduced.

For each new vertex $\mathbf{v}$, its *parent* $p(\mathbf{v})$ is defined as follows: if $\mathbf{v}$ is an input vertex, $p(\mathbf{v})$ is the closest input vertex to $\mathbf{v}$; if $\mathbf{v}$ is an inserted vertex, and if V denotes the set of vertices collected by the point accepting rule, then $p(\mathbf{v}) \in V$ is the closest vertex to $\mathbf{v}$ immediately after $\mathbf{v}$ is inserted. If there are several such vertices, then choose the one which is the most recently inserted. Note that in the final mesh, $p(\mathbf{v})$ may not be the closest vertex to $\mathbf{v}$.

Let θ_m be the smallest acute input angle of X. In case there is no acute angle, set $\theta_m = 90°$.

Lemma 4.2.1 *Let* $\mathbf{v}$ *be a newly inserted vertex, and* $\mathbf{w}$ *is the closet mesh vertex of* $\mathbf{v}$*, then* $\|\mathbf{v}-\mathbf{w}\| \geq \min\{\alpha_2 H(\mathbf{v}), C\alpha_2 H(\mathbf{p}), lfs(\mathbf{v})\}$*, where* $\mathbf{p} = p(\mathbf{v})$ *and* $C = \sin\theta_m/\sqrt{2}$*.*

Proof We proof this lemma by enumerating all the cases which cause the existence of $\mathbf{v}$ and $\mathbf{w}$. For each the length bound for $\|\mathbf{v} - \mathbf{w}\|$ and the constant C are derived, respectively.

Case 1: If $\mathbf{w} = \mathbf{p}$, i.e., $\mathbf{w}$ is just the parent of $\mathbf{v}$, then by the point accepting rule: $\|\mathbf{v} - \mathbf{w}\| \geq \alpha_2 H(\mathbf{w})$.

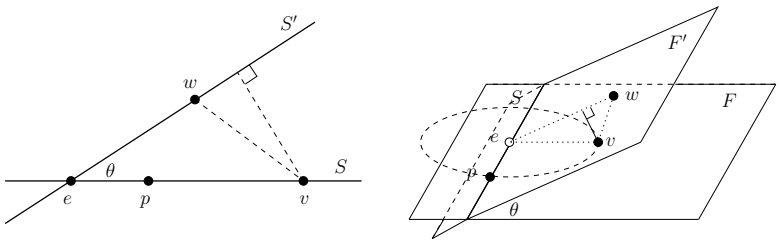
Case 2: If $\mathbf{w} \neq \mathbf{p}$, but $\mathbf{v}$ and $\mathbf{w}$ lie on two disjoint boundaries, then by the definition of the lfs, $\|\mathbf{v} - \mathbf{w}\| \geq \text{lfs}(\mathbf{v})$.

In the remaining cases, we assume $\mathbf{w} \neq \mathbf{p}$, $\mathbf{v}$ and $\mathbf{w}$ lie on two incident boundaries (segments or facets), let θ be the angle (or dihedral angle) of them, $\theta_m \leq \theta < 90°$.

Case 3: If $\mathbf{v}$ was inserted by $R1$. Let S be the segment containing $\mathbf{v}$.

3(a): If $\mathbf{w}$ lies on a segment S', and $S \cap S' = \mathbf{e}$, where $\mathbf{e}$ is an input vertex (see Fig. 4.4 (a)). Since $\|\mathbf{v} - \mathbf{e}\| \geq \|\mathbf{v} - \mathbf{p}\|$, we get,

$$\begin{aligned} \|\mathbf{v} - \mathbf{w}\| &\geq \sin\theta \, \|\mathbf{v} - \mathbf{e}\|, \\ &\geq \sin\theta \, \|\mathbf{v} - \mathbf{p}\|, \\ &\geq \sin\theta_m \, \alpha_2 H(\mathbf{p}). \end{aligned} \tag{4.1}$$



(a) (b)

Figure 4.4: Proof of Lemma 4.2.1. **(a)**: Two segments S and S' form an angle θ, $\mathbf{v}$ and $\mathbf{w}$ lie on each of them, $\|\mathbf{v}-\mathbf{w}\| \geq \sin\theta\ \|\mathbf{v}-\mathbf{p}\|$. **(b)**: Two facets F and F' form a dihedral angle θ, $\mathbf{v}$ and $\mathbf{w}$ lie on each of them. $\|\mathbf{v}-\mathbf{w}\| \geq (\sin\theta/\sqrt{2})\ \|\mathbf{v}-\mathbf{p}\|$.

3(b): If $\mathbf{w}$ lies on a facet F' incident with S. There are two sub-cases: (1) $F' \cap S = \mathbf{e}$; and (2) $F' \cap S = S$. In sub-case (1) the inequality (4.1) applies. If it is sub-case (2), since we assume that $\|\mathbf{v}-\mathbf{w}\| < \|\mathbf{v}-\mathbf{p}\|$, but $\mathbf{w}$ must not encroach upon the segment that $\mathbf{v}$ splits, i.e., $\|\mathbf{v}-\mathbf{w}\| \geq \|\mathbf{v}-\mathbf{p}\|$, this is a contradiction, i.e., sub-case (2) is not possible.

Case 4: If $\mathbf{v}$ is inserted by $R2$. Let F be the facet containing $\mathbf{v}$.

4(a): If $\mathbf{w}$ lies on a segment S'. There are two sub-cases: (1) $F \cap S' = \mathbf{e}$; and (2) $F \cap S' = S'$. (4.1) holds for sub-case (1). In sub-case (2), since $\mathbf{w}$ was inserted before $\mathbf{v}$, hence $\mathbf{w} = \mathbf{p}$, this contradicts our assumption, i.e., sub-case (2) is not possible.

4(b): If $\mathbf{w}$ lies on a facet F'. There are two sub-cases: (1) $F \cap F' = \mathbf{e}$; and (2) $F \cap F' = S$, where S is an segment. (4.1) holds for sub-case (1). In sub-case (2), let $\mathbf{e}$ be the projection of $\mathbf{v}$ onto S (see Fig. 4.4 (b)). Since $\mathbf{v}$ must not encroach upon the subsegment $s \in S$, so $\sqrt{2}\|\mathbf{v}-\mathbf{e}\| \geq \|\mathbf{v}-\mathbf{p}\|$, we get

$$\begin{aligned} \|\mathbf{v}-\mathbf{w}\| &\geq \sin\theta\ \|\mathbf{v}-\mathbf{e}\|, \\ &\geq (\sin\theta/\sqrt{2})\ \|\mathbf{v}-\mathbf{p}\|, \\ &\geq (\sin\theta_m/\sqrt{2})\ \alpha_2 H(\mathbf{p}). \end{aligned} \tag{4.2}$$

With $C = \sin\theta_m/\sqrt{2}$ our claim holds. ∎

Lemma 4.2.1 shows that the shortest edge at every new inserted vertex $\mathbf{v}$ is bounded by H, lfs, and θ_m which are all positive. The termination of the algorithm is guaranteed if α_2 is non-zero.

Theorem 4.2.2 *The algorithm terminates as long as* $\alpha_2 > 0$. ∎

4.2.2 Mesh Quality

In this section, we consider the output mesh quality. The goal is to show that the algorithm is able to create a mesh with most of the tetrahedra having their radius-edge ratio bounded from above, while only a few poor-quality tetrahedra remain in well defined locations.

We say a segment S is *sharp* if either: (1) it is incident with another segment S', such that the angle between S and S' is smaller than $60°$, or (2) it is the intersection of two facets F_1 and F_2, such that the dihedral angle between F_1 and F_2 is smaller than $69.3°$.

For a PLS $\mathcal{X}$, let $\mathcal{S}$ be the set of all sharp segments of $\mathcal{X}$. Define a sizing function H_s as follows:

$$H_s(\mathbf{x}) = \begin{cases} \text{lfs}(\mathbf{x}), & \text{for } \mathbf{x} \in S,\ S \in \mathcal{S}, \\ 0, & \text{for other } \mathbf{x} \in |\mathcal{X}|. \end{cases} \tag{4.3}$$

Note that H_s has zero values everywhere in X except on sharp segments, hence it is not the standard sizing function. However, if $R3.2$ is skipped, this algorithm works as usual.

Theorem 4.2.3 *Suppose H_s is used, $R3.2$ is not executed, and $\rho_0 > 2$. There exists an $\alpha_2 > 0$, such that any output tetrahedron t has a radius-edge ratio smaller than ρ_0, or the circumcenter $\mathbf{c}_t$ of t satisfies:*

$$\|\mathbf{c}_t - \mathbf{p}\| \leq \sqrt{2}\alpha_2 H_s(\mathbf{p}). \tag{4.4}$$

where $\mathbf{p} \in S$ is a mesh vertex, and S is a sharp segment.

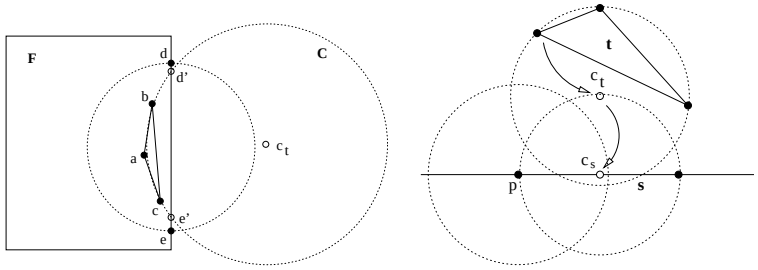
Proof If there is no sharp segment, then α_2 can be an arbitrarily small value, the algorithm behaves like the basic Delaunay refinement algorithm, hence the theorem holds.

Now assume there are sharp segments. We prove the theorem by a "bad tetrahedra elimination" procedure.

At the initialization, choose any $d_0 > 0$, and execute the algorithm with $\alpha_2 = d_0$. Assume the output mesh contains bad-quality tetrahedra (otherwise we are done). Let t be such a tetrahedron, $\mathbf{c}_t$ be its circumcenter. Then t can be categorized into one of the three sets Φ_1, Φ_2, and Φ_3:

- Φ_1: $\mathbf{c}_t$ lies outside the mesh;
- Φ_2: $\mathbf{c}_t$ is in the mesh and $\mathbf{c}_t$ does not encroach any segment or subface;
- Φ_3: $\mathbf{c}_t$ encroaches upon at least a segment or a subface.

If $t \in \Phi_1$, one can show that at least one corner of t encroaches upon a segment or subface. Let $\mathbf{v}$ be the corner of t, and let S be the segment (or subface) which is encroached by $\mathbf{v}$ (see Fig. 4.5 (a)). S is not split because



(a) (b)

Figure 4.5: Proof of Theorem 4.2.3. (a): $t \in \Phi_1$, one case of the existence of an encroached segment. The triangle **abc** is a face of t and lies inside a facet F. C and $\mathbf{c}_t$ are the circumcircle and circumcenter of **abc**, respectively. **de** is the segment of F which is both encroached upon and visible by **a**. (b): $t \in \Phi_2$. The circumcenter $\mathbf{c}_s$ of s lies inside the protecting ball of **p**, one of the endpoints of s.

its circumcenter $\mathbf{c}_s$ is rejected by the point accepting rule, i.e., $\mathbf{c}_s$ lies inside of at least one of the protecting balls of its corners. t will be eliminated if s is split. This can be achieved by shrinking the protecting balls of the endpoints of s such that $\mathbf{c}_s$ lies outside all of them. It is possible to choose a sufficiently small $d_1 > 0$, such that all tetrahedra of Φ_1 can be removed by executing the algorithm with $\alpha_2 = d_1$. The newly inserted vertices may create new poor quality tetrahedra which are in Φ_2 and Φ_3.

If $t \in \Phi_2$, there exits a vertex $\mathbf{p} \in S$ (S is sharp) in the neighborhood of $\mathbf{c}_t$, such that:

$$\|\mathbf{c}_t - \mathbf{p}\| \leq \alpha_2 H_s(\mathbf{p}).$$

If $t \in \Phi_3$, the circumcenter $\mathbf{c}_s$ of the encroached segment (or subface) is rejected by the point accepting rule (see Fig. 4.5 (b)). Hence there exits a vertex $\mathbf{p} \in S$ (S is sharp), $\|\mathbf{c}_s - \mathbf{p}\| \leq \alpha_2 H_s(\mathbf{p})$. Since $\mathbf{c}_t$ is inside the circumsphere centered at $\mathbf{c}_s$ with radius $\|\mathbf{c}_s - \mathbf{p}\|$, hence,

$$\|\mathbf{c}_t - \mathbf{p}\| \leq \sqrt{2}\|\mathbf{c}_s - \mathbf{p}\| \leq \sqrt{2}\alpha_2 H_s(\mathbf{p}).$$

In both of the above cases, our claim holds. ■

The above theorem shows that by using H_s and an appropriate α_2, the resulting mesh quality can be guaranteed as in the basic Delaunay refinement, and the remaining badly-shaped tetrahedra are all close to sharp segments. Moreover, the distance of bad-quality tetrahedra can not be too far from the sharp segments, it is restricted by (4.4).

To illustrate the algorithm, different parameters (ρ_0, α_2) are chosen. The geometry of the example (Fig. 4.6) contains both acute angles and dihedral

angles. We use a sizing function H_1 which is smoother than H_s, it is defined as follows:

- For any input point $\mathbf{p} \in |\mathcal{X}|$, $H_1(\mathbf{p}) = \text{lfs}(\mathbf{p})$.
- For any newly inserted point $\mathbf{p}$,

$$H_1(\mathbf{p}) = \frac{\sum_{i=1}^{n} \|\mathbf{p} - \mathbf{v}_i\|^{-2} H_1(\mathbf{v}_i)}{\sum_{i=1}^{n} \|\mathbf{p} - \mathbf{v}_i\|^{-2}}. \tag{4.5}$$

  where $\mathbf{v}_i$ is the Delaunay vertex connecting to p in the current mesh.

H_1 guarantees that to every point of the mesh a size is assigned. H_1 can be easily computed based on the input geometric data. For any new point $\mathbf{p}$, H_1 is computed on the fly – equation (4.5) is a Shepard interpolation such that the closet point has the biggest influence on the value of $H_1(\mathbf{p})$.

Fig. 4.6 shows two sequences of meshes created by various combinations of (ρ_0, α_2). In one sequence, ρ_0 was fixed at 2, and α_2 varied. While in the other sequence $\alpha_2 = 0.2$, and ρ_0 varied. Remaining poor-quality tetrahedra are plotted for selected meshes. The size statistics of each mesh is given in the form $n_v/n_t/n_b$, where n_v denotes the number of nodes, n_t denotes the number of tetrahedra, and n_b denotes the number of remaining poor-quality tetrahedra. The results satisfy our claim (Theorem 4.2.3) on the mesh quality, i.e., for an appropriate α_2, most of tetrahedra have a bounded radius-edge ratio, poor-quality tetrahedra are all close to small input angles. A few slivers may remain in the volume, they can be removed by a mesh smoothing step.

4.2.3 Mesh Conformity

Next, we consider the mesh conformity with respect to the sizing function H. For each vertex $\mathbf{v}$, recall that $S(\mathbf{v})$ and $L(\mathbf{v})$ denote the lengths of the shortest and the longest edge among all edges containing $\mathbf{v}$, respectively. Define two ratios:

$$S_v = \frac{S(\mathbf{v})}{H(\mathbf{v})}, \text{ and } L_v = \frac{L(\mathbf{v})}{H(\mathbf{v})}.$$

We will use these two ratios to measure the mesh conformity. S_v and L_v should be bounded from below and above. If they are all 1, the conformity would be perfect.

We analyze a special case where $H = \text{lfs}$ and $\theta_m = 90°$. Hence the mesh quality is guaranteed with a sufficiently small α_2. Theorem 4.2.4 establishes bounds for these quantities and output vertices.

Theorem 4.2.4 *Suppose $H = lfs$, $\theta_m = 90°$, $\rho_0 > 2$, and α_2 is small enough such that all output tetrahedra have a radius-edge ratio smaller than ρ_0. Then*

(i) $S_v \geq \frac{1}{C_3\, C_1 + 1}$;

(ii) $L_v \leq 2\alpha_1$.

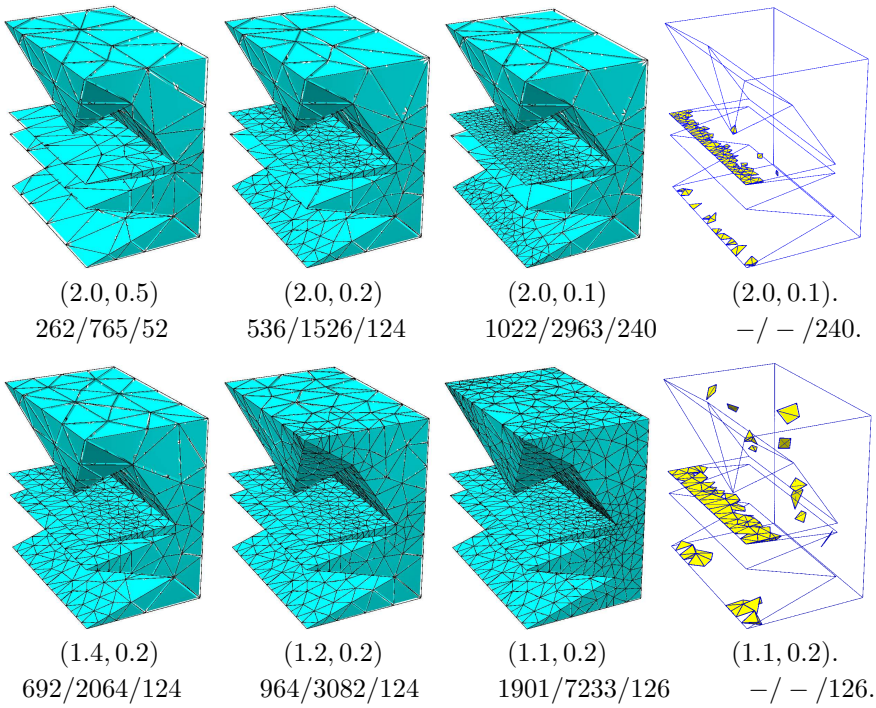


Figure 4.6: Test case for various combinations of parameters (ρ_0, α_2). Remaining bad quality tetrahedra are shown in yellow (right).

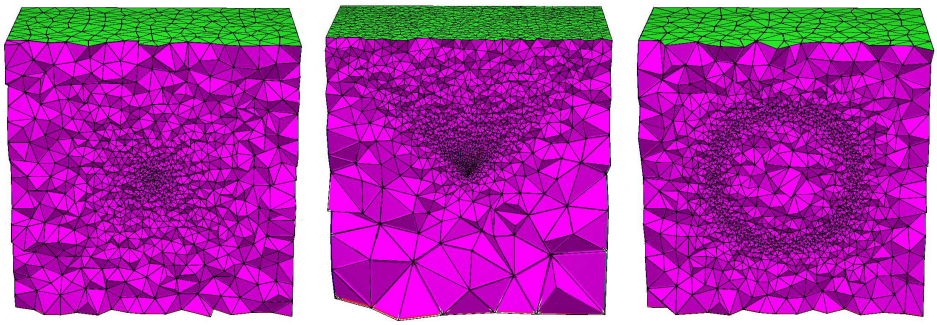


Figure 4.7: Meshes created by specifying different sizing functions.

			Mesh 1		Mesh 2		Mesh 3	
			S_v	L_v	S_v	L_v	S_v	L_v
	$<$	0.5	0	0	0	0	0	0
0.5	–	$1/\sqrt{2}$	58	0	0	0	0	0
$1/\sqrt{2}$	–	1	3221	1	283	0	0	0
1	–	$\sqrt{2}$	15062	113	10778	14	1927	49
$\sqrt{2}$	–	2	4246	3867	1187	1044	94186	12594
2	–	$2\sqrt{2}$	0	18606	0	11190	12276	95746
	$>$	$2\sqrt{2}$	0	0	0	0	0	0

Table 4.1: Statistics of the ratios S_v and L_v (defined in Theorem 4.2.4) at mesh vertices of the meshes shown in Fig. 4.7.

Proof *(i)* follows directly from Corollary 3.3.10. Consider *(ii)*. Let t be a tetrahedron which contains v and has a longest edge of length $L(v)$. Moreover, let r be the circumradius of t. Then: $\frac{L(v)}{2} \leq r \leq \alpha_1 H(v) \Longrightarrow \frac{L(v)}{H(v)} \leq 2\alpha_1$. ■

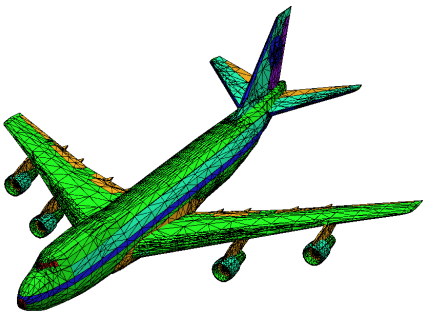
Fig. 4.7 shows three refined meshes resulting from a unit cube. Each mesh is obtained by specifying a piecewise smooth sizing function on a background mesh. The parameters ($\rho_0 = 2.0$, $\alpha_1 = \sqrt{2}$, and $\alpha_2 = 0.05$) are chosen in such a way that the hypothesizes of Theorem 4.2.4 are satisfied. Table 4.1 lists the statistics of the ratios S_v and L_v measured at mesh vertices. For all three meshes, L_v is strictly bounded by $2\alpha_1$, hence verifies *(ii)* of Theorem 4.2.4. The smallest S_v is much larger than α_2. All the three meshes conform well to the specified sizing functions.

4.3 Examples

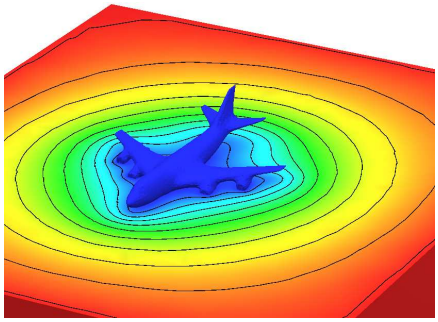
The algorithm has been integrated into `TetGen` – a quality Delaunay tetrahedral mesh generator [110]. The input can be either a PLS or a constrained tetrahedral mesh. A sizing function H can be optionally specified through a background mesh. Parameters ρ_0, α_1, and α_2 are all adjustable at runtime. Below are three selected application examples which generated by `TetGen` using the algorithm discussed in this chapter.

The first example is a Boeing 747 model. The input is a the surface mesh of the plane (2,874 nodes, 5,738 triangles) plus a bounding box. The tetrahedral mesh is constructed to compute a potential flow around the Boeing 747. The sizing function is explicitly given by a smoothed function of the Euclidian distance from the surface mesh (Fig. 4.8 (b)). The resulting tetrahedral mesh (Fig. 4.8 (c) and (d) 490,692 nodes, 2,709,770 tetrahedra) was generated with parameters: $\rho_0 = 2.0, \alpha_1 = 0.5, \alpha_2 = 0.25$.

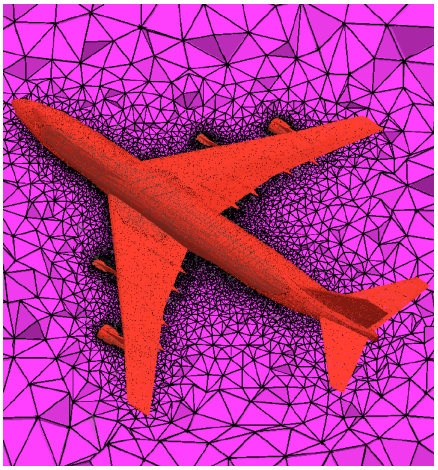
Remaining bad quality tetrahedra are all close to the sharp segments of



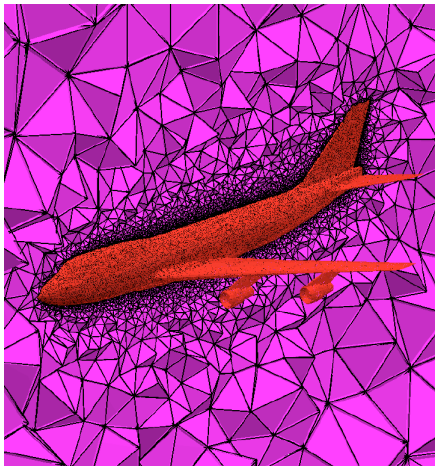
(a) The surface mesh, 2,874 nodes and 5,738 triangles.



(b) Tetrahedra size information is shown on a cut through the box, blue - small size tetrahedra, red - large size tetrahedra.

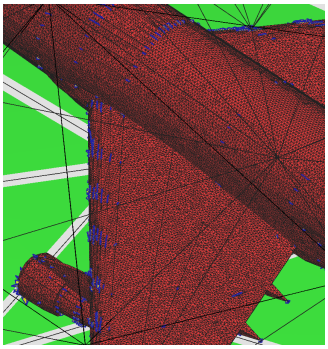


(c) Mesh detail (490,692 nodes and 2,709,770 tetrahedra)



(d) Mesh detail, refine time 59 s (44.5k tet/sec), 3.60GHz Intel PC.

Figure 4.8: Adaptive tetrahedral mesh of the Boeing 747 model.



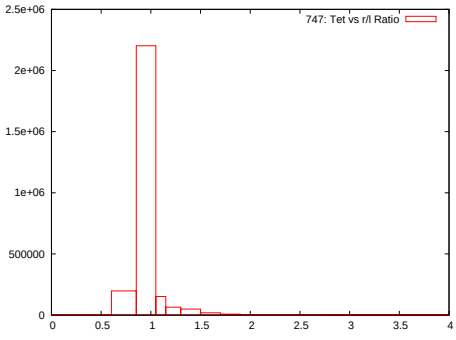


Figure 4.9: Mesh quality plot of the Boeing 747 model. Left: The remaining bad quality tetrahedra (shown in blue) are located close to the surface and sharp input segments. Right: The radius-edge ratio histogram.

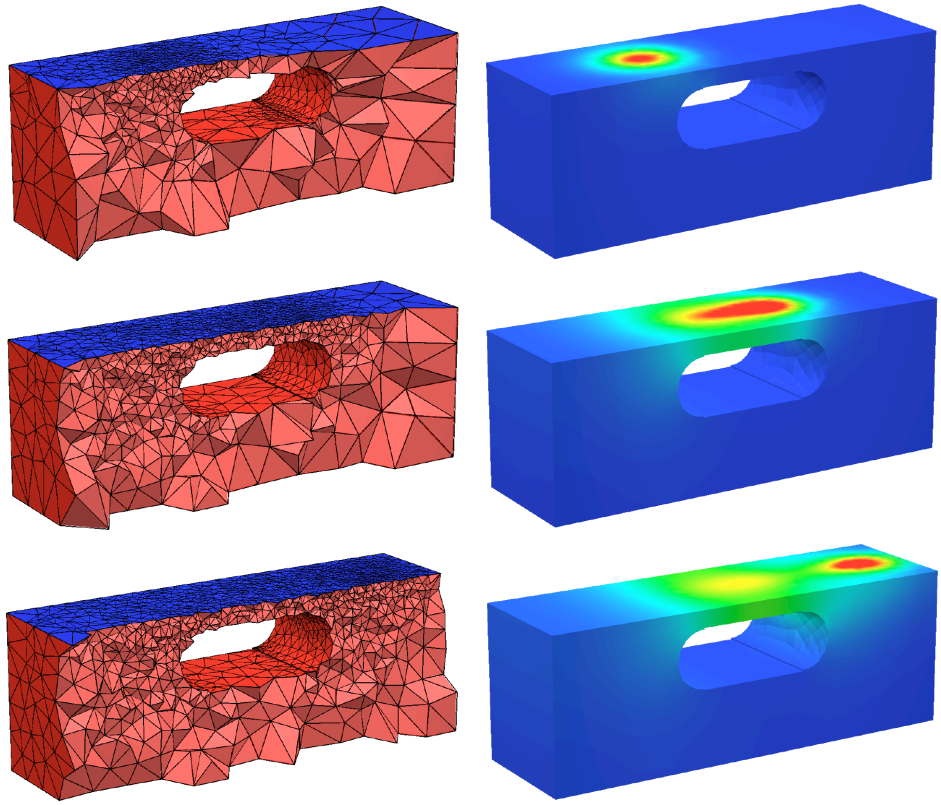


Figure 4.10: A sequence of adaptive meshes related to the moving laser spot (left) and the temperature distributions (right).

the plane's surface (Fig. 4.9 left). The histogram of radius-edge ratios of the tetrahedral mesh is shown in Fig. 4.9 right. The mesh contains high-quality tetrahedra in the bulk of the meshed domain. For example, over 94% of the tetrahedra of the Boeing 747 mesh have radius-edge ratios between 0.612 and 1.1. Only about 0.4% of the tetrahedra are bad-quality.

The second example (Fig. 4.10) illustrates the use of adaptivity and is part of the project `WIAS-SHarP` [120]. A moving laser spot heats a workpiece. Adaptive boundary conforming Delaunay meshes are generated whenever the estimated error in the solution gets larger than a user specified bound. The mesh gets coarse again in the already cooled down regions, where the estimated error is small.

4.4 Summary

This chapter is dealing with some modern aspects of adaptive meshing and solving PDEs. We presented a variant three-dimensional Delaunay refine-

ment algorithm for mesh adaptation purpose. The input is the constrained tetrahedralization of a PLS. It completely eliminates the input angle restrictions and it supports adaptive refinement through a user-defined sizing function. We summarize this chapter in the following issues.

• We extended the Delaunay refinement algorithm for the mesh adaptation purpose, the conformity of the mesh size can be theoretically guaranteed for a smooth sizing function H. More precisely, in a bounded radius-edge ratio mesh, for an interior mesh vertex $\mathbf{p}$, the shortest and longest edges at $\mathbf{p}$ are bounded with respect to $H(\mathbf{p})$. Moreover, there is no limitation on input angles and shapes. Prior to this algorithm, Oudot *et al.* [88] proposed a similar approach. However, they provide no guarantee on the mesh conformity, and the input is required to be a smooth 2-manifold.

• We introduced the using of both protecting and sparse balls into the Delaunay refinement scheme. The sizes of the balls can be adjusted through the parameters: α_1 and α_2, where α_1 can be used to adjust the the mesh conformity, and $\alpha_2 > 0$ affects the mesh quality. Although protecting balls are already used in [20, 91], our approach is much simpler and efficient.

• We proved that the locations of remaining bad quality tetrahedra are close to input sharp segments. Moreover, we explicitly give a bound of the closeness, which is $\sqrt{2}\alpha_2 H(\mathbf{p})$, where $\mathbf{p}$ is a vertex of a sharp subsegment of the resulting mesh. It implies that one can control the locations of the remaining bad quality tetrahedra by adjusting α_2.

A number of theoretical questions remain regarding this algorithm. Below are some questions which worth to be investigated in the future.

◦ If there are small input angles ($< 69.3^\circ$) in the initial mesh, the result mesh $\mathcal{T}$ is not guaranteed to be boundary conforming Delaunay. $\mathcal{T}$ even may not be conforming Delaunay. The algorithms of Cheng and Dey [20], Pav and Walkington [91] always guarantee the latter though at the expense of extra complications. It is still an open question to achieve the boundary conforming Delaunay property.

◦ Slivers are not handled in this algorithm. It is possible to apply the simple heuristic method described in Section 3.4.2 to remove all the slivers from the interior of the mesh. However, the slivers close to sharp segments can not be removed by this way. A post mesh optimization algorithm, e.g., [46], is necessary to improve the mesh quality.

◦ The mesh conformity guarantee is based on the assumption that the user-defined sizing function is smooth. Indeed, an appropriate mesh sizing

function needs to be adapted to the input geometry (e.g., the local feature sizes, curvatures) and smoothed.

◦ The mesh coarsening is another important ingredient for mesh adaptation. We do not discuss this here. However, it is straightforward to include a mesh coarsening operation in the main algorithm.

◦ Adaptation of the algorithm to anisotropic H. For this purpose, the usual ways of measuring edge length, updating the locally Delaunay property around Steiner points, and the interpolation of H must be modified to handle anisotropy [47, 52].

Chapter 5

Constrained Delaunay Mesh Generation

A fundamental problem in Delaunay mesh generation is to conform the boundary of the mesh domain, also known as *boundary conformity*. This chapter studies the three-dimensional boundary conformity problem to support the generation of boundary conforming Delaunay meshes.

A *constrained Delaunay triangulation* (CDT) of a polyhedral domain (e.g., a PLS) is a Delaunay like triangulation which contains the domain boundary. It has many nice properties as close as those of Delaunay triangulations. We propose an algorithm to construct a CDT for an arbitrary PLS. Steiner points will be introduced on the boundary of the PLS. The choice of Steiner points avoids creating unnecessarily short edges hence it also helps to reduce the total number of Steiner points. The termination and correctness of this algorithm are proved. Practical examples are used to illustrate the robustness and efficiency of this algorithm. Part of this chapter is based on a joint work with Gärtner [112].

This chapter is organized as follows. Our definition of CDT as well as some basic properties are found in Section 5.1. The proposed algorithm and its correctness are discussed in Section 5.2. The details of the individual steps of the algorithm are described in the consecutive sections, i.e., Sections 5.3, 5.4, and 5.5. The analysis of the algorithms are found therein. Some experimental results from publicly available examples are shown in Section 5.6. This chapter ends with a summary in Section 5.7.

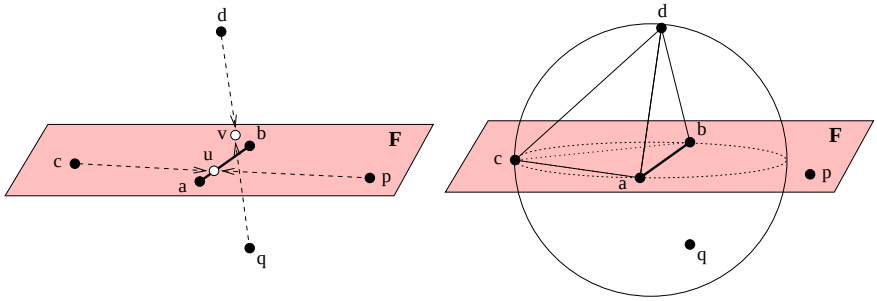


Figure 5.1: Visibility and constrained Delaunay. The shaded region represents a 2-polytope F of a PLS $\mathcal{X} \in \mathbb{R}^d$, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{p} \in F$, and $\mathbf{d}$, $\mathbf{q}$ are in opposite sides of F. $\mathbf{ab}$ is a 1-polytope of $\mathcal{X}$. Left: $\mathbf{d}$ and $\mathbf{q}$ are invisible to each other since $\mathbf{dq} \cap F = \mathbf{v}$. $\mathbf{c}$ and $\mathbf{p}$ are invisible to each other since $\mathbf{cp} \cap \mathbf{ab} = \mathbf{u}$. On the other hand, $\mathbf{u}$ sees both $\mathbf{c}$ and $\mathbf{p}$. Right: A circumball of the tetrahedron $\mathbf{abcd}$ is shown. It contains $\mathbf{q}$ in its interior. $\mathbf{abcd}$ is constrained Delaunay since $\mathbf{q}$ is not visible from its interior. The triangle $\mathbf{abc} \subset F$ is constrained Delaunay since $\mathbf{p}$ is outside its diametric ball.

5.1 Constrained Delaunay Triangulations

The two-dimensional *constrained Delaunay triangulation* of a planar straight line graph $\mathcal{G}$ (which is a 1-dimensional PLS) is a Delaunay-like triangulation that contains the set of edges of $\mathcal{G}$. It was independently defined by Lee and Lin [69] and Chew [22]. Such triangulations have many nice properties which are close to those of Delaunay triangulations. Moreover, any 2-dimensional PLS has a constrained Delaunay triangulation with no Steiner point.

The concept of constrained Delaunay triangulations has long been used in numerical mesh generation, see e.g., [6, 59, 13]. However, one of the difficulties in defining such objects in three dimensions is the well-known fact that not every three-dimensional polyhedron admit a tetrahedralization without Steiner points. In the following, we propose a definition for a *constrained Delaunay triangulation* of a PLS of any dimension.

5.1.1 Definitions

Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. A crucial concept is the *visibility* of points in $\mathbb{R}^d$. The basic idea is: every polytope $F \in \mathcal{X}$ is an obstacle which will block the visibility for points which are not in F. While F does not block the visibility for its own points. We formalize the idea in the following definition.

Definition 5.1.1 (Visibility) *Two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ are invisible to each other if the interior of the line segment $\boldsymbol{xy}$ intersects a polytope $F \in \mathcal{X}$ in a single point. Otherwise $\boldsymbol{x}$ and $\boldsymbol{y}$ are visible to each other.*

See Fig. 5.1 left for examples. The next definition relaxes the usual Delaunay criterion by using the concept of visibility, it is called the *constrained Delaunay crietrion*. Recall that the *carrier* of a subset $U \subseteq |\mathcal{X}|$ is the smallest polytope of $\mathcal{X}$ that contains U.

Definition 5.1.2 (Constrained Delaunay) *Let S be a finite set of points and $\mathcal{X}$ be a PLS in $\mathbb{R}^d$ with vert($\mathcal{X}$) $\subseteq S$. A simplex σ whose vertices are in S is constrained Delaunay if it is in one of the two cases:*

(i) There is a circumscribed sphere Σ of σ does not have any vertex of S lies inside Σ.

(ii) Let $F \in \mathcal{X}$ be the carrier of σ. Let $K = S \cap$ aff(F), then no vertex of K inside Σ is visible from any point in int(σ).

It is easy to see that every Delaunay simplex of S is constrained Delaunay. In *(ii)*, F is the the lowest-dimensional polytope of $\mathcal{X}$ that contains σ, and K is the subset of S in the affine hull generated by F. A simplex $\sigma \subset F$ is constrained Delaunay or not depends only on the vertices of K. See Fig. 5.1 right for examples.

A *constrained Delaunay triangulation* (abbreviated as CDT) of $\mathcal{X}$ is then defined as follows:

Definition 5.1.3 (Constrained Delaunay Triangulation) *Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. A constrained Delaunay triangulation (CDT) of $\mathcal{X}$ is a triangulation $\mathcal{T}$ of $\mathcal{X}$ such that every simplex of $\mathcal{T}$ is constrained Delaunay.*

If a CDT of $\mathcal{X}$ contains Steiner points, we call it a *Steiner* CDT. Otherwise, it is called a *pure* CDT. Hence a 2-dimensional pure CDT is the same as the one defined by Lee and Lin [69] and Chew [22]. It is well known that a pure CDT of a 3-dimensional PLS may not exist. For instance, if $\mathcal{X}$ contains the Schönhardt polyhedron [95], and the problem to decide whether a pure CDT of $\mathcal{X}$ exists or not is NP-complete [93]. On the other hand, there are infinitely many Steiner CDTs of $\mathcal{X}$. It is still an open question how to efficiently find a Steiner CDT of a three-dimensional PLS with the number of Steiner points is as small as possible.

5.1.2 Basic Properties

In the following, we derive some basic properties of the CDTs we defined. These properties show that a CDT is very close to a (conforming) Delaunay triangulation.

An important property of Delaunay triangulations is the locality property. Let S be a finite set of points in $\mathbb{R}^d$, and $\mathcal{K}$ be any triangulation of S. A $(d-1)$-simplex σ of $\mathcal{K}$ is *locally Delaunay* if either σ is on the convex

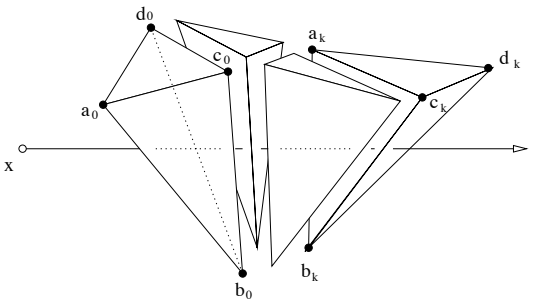


Figure 5.2: Proof of the constrained Delaunay Lemma (Theorem 5.1.4). A sequence of tetrahedra intersect the half-line starting from point **x**.

hull, or the opposite vertex of τ is not in $\text{int}(B_\nu)$ of ν, where $\tau, \nu \in \mathcal{K}$ and $\sigma = \tau \cap \nu$. $\mathcal{K}$ is a Delaunay triangulation if every $(d-1)$-simplex of $\mathcal{K}$ is locally Delaunay. TWe show that this property holds in CDTs as well.

Let $\mathcal{X}$ be a d-dimensional PLS. Let $\mathcal{T}$ be any triangulation of $\mathcal{X}$. A $(d-1)$-simplex σ of $\mathcal{T}$ is *locally Delaunay* if either *(i)* σ is on the convex hull, or *(ii)* $\sigma \subset |\partial\mathcal{X}|$, or *(iii)* the opposite vertex of τ is not in $\text{int}(B_\nu)$ of ν, where $\tau, \nu \in \mathcal{T}$ and $\sigma = \tau \cap \nu$. Note that *(ii)* implies that one can ignore the $(d-1)$-simplices contained in $|\partial\mathcal{X}|$.

Theorem 5.1.4 (Delaunay Lemma) *If every $(d-1)$-simplex of $\mathcal{T}$ is locally Delaunay, then $\mathcal{T}$ is a CDT of $\mathcal{X}$.*

Proof Consider a d-simplex $\tau \in \mathcal{T}$ and a vertex $\mathbf{p} \in \mathcal{T}$ such that $\mathbf{p}$ is not a vertex of τ and $\mathbf{p}$ is visible from the interior of τ. We show that $\mathbf{p}$ lies outside the circumball B_τ of τ. Because this is then true for every $\mathbf{p}$, B_τ is either empty or contains no vertex which is visible from its interior, and because this is then true for every d-simplex of $\mathcal{T}$, $\mathcal{T}$ is a CDT of $\mathcal{X}$.

Choose a point $\mathbf{x} \in \text{int}(\tau)$ such that the line segment $\mathbf{xp}$ does not contain any vertex of $\mathcal{T}$. Let $\sigma_0, \sigma_1, ..., \sigma_k$ be the sequence of $(d-1)$-simplices that intersect $\mathbf{x}, \mathbf{p}$, where $\sigma_0 < \tau$ and σ_k and $\mathbf{p}$ form a d-simplex of $\mathcal{T}$, see Fig. 5.2. Note that all $\sigma_i, 0 \leq i \leq k$ are locally Delaunay, and no $\sigma_i, 0 \leq i \leq k$ is contained in a polytopes of $\mathcal{X}^{(d-1)}$ (since $\mathbf{p}$ is visible by $\mathbf{x}$). By the Acyclicity Theorem for cell complexes [34], the sequence of d-simplices, $\tau = \tau_0, \tau_1, ..., \tau_k$ is acyclic, where $\sigma_i = \tau_{i-1} \cap \tau_i, 0 < i < k$ and $\mathbf{p} < \sigma_k$. This is equivalent to saying that $\mathbf{p}$ is lying outside of the circumball of τ. ■

If a set of vertices is in general position, i.e., no $d+2$ points share a common $(d-1)$-sphere, then the Delaunay triangulation of this vertex set is unique. By the above theorem, this property holds for CDT as well.

Theorem 5.1.5 *Let $\mathcal{T}$ be a CDT of $\mathcal{X}$. If vert($\mathcal{T}$) is in general position, then $\mathcal{T}$ is the unique CDT of $\mathcal{X}$ with respect to vert($\mathcal{T}$).* ■

Let $\mathcal{X}$ be a k-dimensional PLS in $\mathbb{R}^d$. The i-skeleton $\mathcal{X}^{(i)}$ of $\mathcal{X}$ is an i-dimensional PLS, where $-1 < i < k$. It is useful to know the properties of a CDT of $\mathcal{X}^{(i)}$.

Proposition 5.1.6 *Let $\mathcal{X}$ be a k-dimensional PLS.*

(i) A CDT of $\mathcal{X}^{(k-1)}$ is a CDT of $\mathcal{X}$.

(ii) A CDT of $\mathcal{X}^{(k-2)}$ is a conforming Delaunay triangulation of $\mathcal{X}^{(k-1)}$.

(iii) A CDT of $\mathcal{X}^{(i)}$ is a conforming Delaunay triangulation of $X^{(i)}$, where $-1 < i < k-1$.

Proof Let $\mathcal{T}$ be a CDT of $\mathcal{X}^{(k-1)}$. Any polytope $F \in \mathcal{X}$ is a union of simplices in $\mathcal{T}$, this implies that $\mathcal{T}$ is a triangulation of $\mathcal{X}$. Any simplex $\sigma \in \mathcal{T}$ is constrained Delaunay within $\mathcal{X}$, hence $\mathcal{T}$ is a CDT of $\mathcal{X}$ (by the definition of CDT). This proves claim *(i)*.

Now let $\mathcal{T}$ be a CDT of $\mathcal{X}^{(k-2)}$. Claim *(ii)* can be proved by showing that any d-simplex $\sigma \in \mathcal{T}$ is Delaunay within vert($\mathcal{X}$), where $d = \dim(\text{vert}(\mathcal{X}))$ ($d \geq k$). Suppose that σ is constrained Delaunay but not Delaunay. Then the circumball B_σ of σ is not empty. Let $\mathbf{p} \in \text{int}(B_\sigma) \cap \mathcal{T}$. Choose an arbitrary point $\mathbf{x} \in \text{int}(\sigma)$, and consider the line segment $\mathbf{px}$. If $\mathbf{px} \cap \mathcal{X}^{(k-2)} \neq \emptyset$, then there is a polytope $F \in \mathcal{X}^{(k-2)}$ such that F blocks the visibility of $\mathbf{p}$ and $\mathbf{x}$. Note that the dimension of F is at most $k-2$, while the dimension of σ is at least k. One can always find a $\mathbf{x}' \in \text{int}(\sigma)$ such that $\mathbf{x}'$ does not in aff($F \cup \mathbf{p}$) (which has dimension at most $k-1$) and $\mathbf{px}' \cap \mathcal{X}^{(k-2)} = \emptyset$. Hence $\mathbf{p}$ is visible by int(σ). This implies that σ is not constrained Delaunay which leads to a contradiction.

Note that claim *(ii)* is just a special case of *(iii)*. The above arguments apply to claim *(iii)* as well. ■

5.2 The CDT Algorithm

Let $\mathcal{X}$ be a three-dimensional PLS, i.e., $\mathcal{X}$ is a collection of polytopes of dimensions up to 3. The algorithm to construct a constrained Delaunay tetrahedral mesh $\mathcal{T}$ (with Steiner points) of $\mathcal{X}$ works in the following steps:

1. Form the Delaunay tetrahedralization $\mathcal{D}_0$ of $\text{vert}(\mathcal{X})$.
2. Let $\mathcal{D}_1 = \mathcal{D}_0$. Recover segments of $\mathcal{X}$ in $\mathcal{D}_1$ (by adding Steiner points in $\mathcal{D}_1$) such that $\mathcal{D}_1$ is a CDT of $\mathcal{X}^{(1)}$.
3. Let $\mathcal{D}_2 = \mathcal{D}_1$. Recover facets of $\mathcal{X}$ in $\mathcal{D}_2$ (without adding Steiner points) such that $\mathcal{D}_2$ is a CDT of $\mathcal{X}^{(2)}$.
4. Let $\mathcal{T} = \mathcal{D}_2 \setminus \{\tau \in \mathcal{D}_2 \,|\, \tau \nsubseteq |\mathcal{X}|\}$. Return $\mathcal{T}$.

This algorithm proceeds in increasing order of the dimensions of the skeletons of $\mathcal{X}$. It initializes a CDT of $\mathcal{X}^{(0)}$ (which is the Delaunay tetrahedralization of $\text{vert}(\mathcal{X})$). In each step i, where $i = 1, 2$, it constructs a CDT of $\mathcal{X}^{(i)}$ from the available CDT of $\mathcal{X}^{(i-1)}$. Lastly, a constrained Delaunay mesh of $\mathcal{X}$ is obtained by removing simplices not in $|\mathcal{X}|$.

5.2.1 Correctness

In this algorithm, Steiner points are only introduced in the step *2* (segment recovery). In order to show the correctness of this algorithm, we will first need the following assumption.

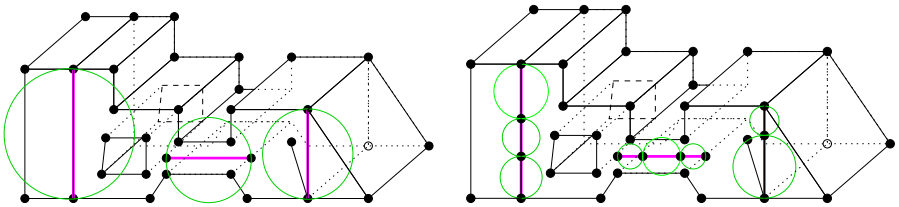
Assumption 5.2.1 *Assume the vertex set of the CDT $\mathcal{D}_1$ of $\mathcal{X}^{(1)}$ is in general position, i.e., no five vertices of vert($\mathcal{D}_1$) share a common sphere.*

Although this assumption is very strong, it is easy to satisfied by applying the techniques of symbolic perturbations [38, 103, 29] in both steps *1* and *2*. Hence, (theoretically) there is no need to actually perturb the vertices.

The following theorem proved by Shewchuk [100] ensures the correctness of the algorithm. Let S be a finite set of vertices. A simplex σ whose vertices are in S is *strongly Delaunay* if there exists a circumscribed ball B_σ of σ, such that $B_\sigma \cap S = \emptyset$.

Theorem 5.2.2 ([100]) *If every segment of a PLS $\mathcal{Y}$ is strongly Delaunay in vert($\mathcal{Y}$), then $\mathcal{Y}$ has a CDT with no Steiner point.* ■

After step *2* of the algorithm, each segment of $\mathcal{X}$ is a union of edges in $\mathcal{D}_1$. Moreover, $\mathcal{D}_1$ is the Delaunay tetrahedralization of $\text{vert}(\mathcal{D}_1)$. Hence, each subsegment of $\mathcal{X}$ is strongly Delaunay (by the Assumption 5.2.1) in



(a) A PLS $\mathcal{X}$ (b) A PLS $\mathcal{Y}$

Figure 5.3: Shewchuk's CDT Theorem [100]. (a): The PLS $\mathcal{X}$. Several segments of $\mathcal{X}$ are not Delaunay. (b): A refined PLS $\mathcal{Y}$ which is topologically and geometrically equivalent to $\mathcal{X}$. All segments of $\mathcal{Y}$ are Delaunay.

$\mathcal{D}_1$. Let $\mathcal{Y}$ be a PLS which is the *refinement* of $\mathcal{X}$, that is, each segment of $\mathcal{X}$ is union of segments in $\mathcal{Y}$, and the segments of $\mathcal{Y}$ are all edges in $\mathcal{D}_1$. Then $\mathcal{Y}$ has a CDT with no Steiner point. See Fig. 5.3 for an example.

This condition is also useful in practice since it suggests that the Steiner points can be inserted on segments only.

Once the existence of a CDT with no Steiner points is known, we still need to show that the step *3*, i.e., facet recovery, can be done without using Steiner points. We will postpone the proof of the correctness of the algorithm until the end of Section 5.4.

5.3 Segment Recovery

This section gives an algorithm for recovering segments. The inputs are $\mathcal{X}$ and $\mathcal{D}_0$, where $\mathcal{X}$ is a three-dimensional PLS, and $\mathcal{D}_0$ is the Delaunay tetrahedralization of $\text{vert}(\mathcal{X})$. A segment of $\mathcal{X}$ is *missing* in $\mathcal{D}_0$ if it is not an edge of $\mathcal{D}_0$. The purpose of the segment recovery algorithm is to update $\mathcal{D}_0$ into $\mathcal{D}_1$ such that $\mathcal{D}_1$ is a CDT of $\mathcal{X}^{(1)}$. Hence every segment of $\mathcal{X}$ is the union of edges of $\mathcal{D}_1$.

5.3.1 Segment Splitting Rules

A vertex in $\mathcal{X}$ is *acute* if at least two segments of $\mathcal{X}$ incident at it form an angle smaller than 90°. We distinguish two types of segments in $\mathcal{X}$: a segment is *type*-0 if its both endpoints are not acute, it is *type*-1 if only one of its endpoints is acute. If both endpoints of a segment are acute, it can be transformed into two type-1 segments by inserting a vertex.

Let $\mathbf{e}_i\mathbf{e}_j$ be a segment of $\mathcal{X}$ with endpoints $\mathbf{e}_i$ and $\mathbf{e}_j$. $\mathbf{e}_i\mathbf{e}_j$ is split by adding a Steiner point in the interior of it. The two resulting edges are subsegments of $\mathbf{e}_i\mathbf{e}_j$. Subsegments inherit types from the original segments. For example, let $\mathbf{e}_i\mathbf{e}_j$ be a subsegment of $\mathbf{e}_1\mathbf{e}_2$ which is a type-1 segment and $\mathbf{e}_1$ is acute, $\mathbf{e}_i\mathbf{e}_j$ is type-1 although none of its endpoints is acute. For any vertex $\mathbf{v}$ inserted on a type-1 segment (or subsegment), $R(\mathbf{v})$ denotes

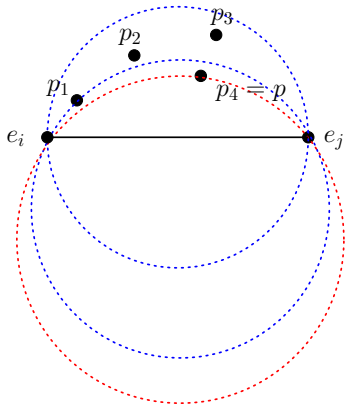


Figure 5.4: The reference point $\mathbf{p}$ of a missing segment $\mathbf{e}_i\mathbf{e}_j$. $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$ all encroach upon $\mathbf{e}_i\mathbf{e}_j$. $\mathbf{p}_4$ is chosen as the reference point because it forms the biggest the diametric circumball (shown in red) with $\mathbf{e}_i\mathbf{e}_j$.

its original acute vertex. A tacit rule is used throughout this section, if $\mathbf{e}_i\mathbf{e}_j$ is type-1, it implies either $\mathbf{e}_i$ or $R(\mathbf{e}_i)$ is the acute vertex. In the following, unless it is explicitly mentioned, a segment can be a segment or a subsegment.

A vertex *encroaches upon* a segment if it lies inside the diametric circumball of that segment. Obviously, if a segment of $\mathcal{X}$ is missing in $\mathcal{D}_0$, it must be encroached by at least one vertex of $\mathcal{D}_0$.

Let $\mathbf{e}_i\mathbf{e}_j$ be a missing segment, it will be split by a vertex $\mathbf{v}$ located inside it. The *reference point* $\mathbf{p}$ of $\mathbf{v}$, which is responsible for the insertion of $\mathbf{v}$, is defined as follows

(i) $\mathbf{p}$ encroaches upon $\mathbf{e}_i\mathbf{e}_j$, and

(ii) the radius of the diametric circumball of triangle $\mathbf{e}_i\mathbf{e}_j\mathbf{p}$ is maximum over other encroaching points of $\mathbf{e}_i\mathbf{e}_j$.

Figure 5.4 illustrates a reference point. In *(ii)*, the choice of taking the $\mathbf{p}$ which forms the largest diametric circumball with $\mathbf{e}_i$ and $\mathbf{e}_j$ is purely empirical. Notice that $\mathbf{p}$ may not be unique (because several points can share the same sphere), randomly choose one in this case.

The insertion of $\mathbf{v}$ to split $\mathbf{e}_i\mathbf{e}_j$ is governed by three rules given below. Let $\Sigma(\mathbf{c}, r)$ denote a sphere centered at $\mathbf{c}$ with radius r:

1. $\mathbf{e}_i\mathbf{e}_j$ is type-0 (see Figure 5.5 left), then $\mathbf{v} = \mathbf{e}_i\mathbf{e}_j \cap \Sigma$, where $\Sigma(\mathbf{c}, r)$ is defined as follows:

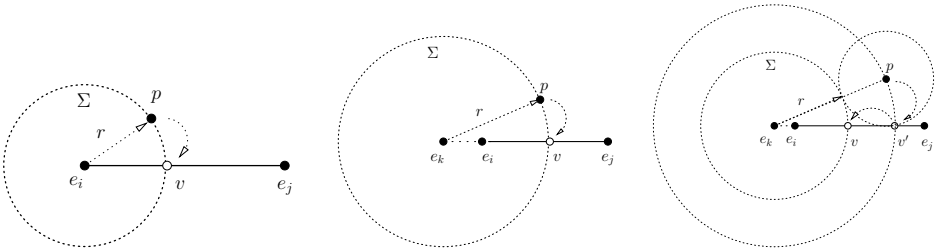


Figure 5.5: Illustrations of the segment splitting rules. Rule 1 (left), $\mathbf{e}_i\mathbf{e}_j$ is type-0, because $\|\mathbf{e}_i - \mathbf{p}\| > \|\mathbf{e}_j - \mathbf{p}\|$, and $\|\mathbf{e}_j - \mathbf{p}\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$, the cutting sphere Σ is centered at $\mathbf{e}_j$, its radius $r = \|\mathbf{e}_j - \mathbf{p}\|$. Rule 2 (middle), $\mathbf{e}_i\mathbf{e}_j$ is type-1, where $\mathbf{e}_k = R(\mathbf{e}_i)$ is acute, Σ is centered at $\mathbf{e}_k$, and $r = \|\mathbf{e}_k - \mathbf{p}\|$ if $\|\mathbf{v} - \mathbf{e}_j\| \geq \|\mathbf{v} - \mathbf{p}\|$, otherwise $\mathbf{v}$ is rejected. Rule 3 (right), $\mathbf{e}_i\mathbf{e}_j$ is type-1, $\mathbf{e}_k = R(\mathbf{e}_i)$, $\mathbf{v}'$ is the rejected point by rule 2, Σ is centered at $\mathbf{e}_k$, because $\|\mathbf{p} - \mathbf{v}'\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$, $r = \|\mathbf{e}_k - \mathbf{e}_i\| + \|\mathbf{e}_i - \mathbf{v}'\| - \|\mathbf{p} - \mathbf{v}'\|$.

if $\|\mathbf{e}_i - \mathbf{p}\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$, **then**
  $\mathbf{c} = \mathbf{e}_i, r = \|\mathbf{e}_i - \mathbf{p}\|$.
else if $\|\mathbf{e}_j - \mathbf{p}\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$, **then**
  $\mathbf{c} = \mathbf{e}_j, r = \|\mathbf{e}_j - \mathbf{p}\|$.
else
  $\mathbf{c} = \mathbf{e}_i, r = \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$.
end

2. $\mathbf{e}_i\mathbf{e}_j$ is type-1 (see Figure 5.5 middle), let $\mathbf{e}_k = R(\mathbf{e}_i)$, then $\mathbf{v} = \mathbf{e}_k\mathbf{e}_j \cap \Sigma$, where $\Sigma(\mathbf{c}, r)$ is defined as follows:
   $\mathbf{c} = \mathbf{e}_k, r = \|\mathbf{e}_k - \mathbf{p}\|$.
   However, if the subsegment $\mathbf{v}\mathbf{e}_j$ has length $\|\mathbf{v} - \mathbf{e}_j\| < \|\mathbf{v} - \mathbf{p}\|$, then we do not insert v and use rule 3 to split the segment.

3. $\mathbf{e}_i\mathbf{e}_j$ is type-1 (see Figure 5.5 right), let $\mathbf{e}_k = R(\mathbf{e}_i)$, and $\mathbf{v}'$ is the rejected vertex by rule 2. then $\mathbf{v} = \mathbf{e}_k\mathbf{e}_j \cap \Sigma$, where $\Sigma(\mathbf{c}, r)$ is defined as follows:
   $\mathbf{c} = \mathbf{e}_k$
   if $\|\mathbf{p} - \mathbf{v}'\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$, **then**
     $r = \|\mathbf{e}_k - \mathbf{e}_i\| + \|\mathbf{e}_i - \mathbf{v}'\| - \|\mathbf{p} - \mathbf{v}'\|$.
   else
     $r = \|\mathbf{e}_k - \mathbf{e}_i\| + \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$.
   end

The idea of these segment splitting rules is to avoid create too short edges, and the choices of the locations should not cause endless loop. All the three Rules guarantee that the newly inserted vertex does not too close to the existing vertices. Note that Rule 1 and 2 never create an edge shorter than the distance $\|R(\mathbf{e}_i) - v\|$. Rule 3 may create an edge which is at most

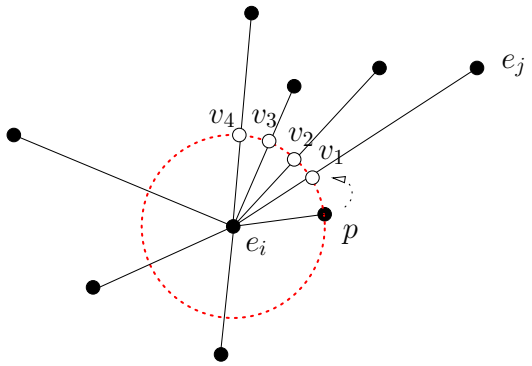


Figure 5.6: The protecting ball of an acute vertex $\mathbf{e}_i$. $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, and, $\mathbf{v}_4$ are points inserted on segments (by rule 2) sharing $\mathbf{e}_i$. They automatically create a protecting ball of $\mathbf{e}_i$.

one third of the length of $\|R(\mathbf{e}_i) - \mathbf{e}_j\|$. Our analysis will show the total number of applying Rule 3 is bounded.

For several segments sharing an acute vertex, by repeatedly using Rule 2 or 3, a protecting ball is automatically created which ensures: no other vertex can be inserted inside the ball. The effect is shown in Figure 5.6. Notice, the protecting ball is not necessarily completely created, only the missing segments will be split and protected. Existing segments remain untouched. This reduces the number of Steiner points.

5.3.2 The Algorithm

The SegmentRecovery algorithm is described in Fig. 5.7. The inputs are a three-dimensional PLS $\mathcal{X}$ and a Delaunay tetrahedralization $\mathcal{D}_0$ of vert($\mathcal{X}$). Some segments of $\mathcal{X}$ may be missing in $\mathcal{D}_0$. The algorithm first initialize a queue Q of all segments of $\mathcal{X}$. Then the algorithm runs into a loop until Q is empty.

For each segment (or subsegment) $\mathbf{e}_i\mathbf{e}_j \in Q$. If it is missing in $\mathcal{D}_1$, a Steiner point $\mathbf{v}$ inside $\mathbf{e}_i\mathbf{e}_j$ is generated by one of the three rules described in previous section (line 6). $\mathbf{v}$ will split $\mathbf{e}_i\mathbf{e}_j$ into two subsegments $\mathbf{e}_i\mathbf{v}$ and $\mathbf{e}_j\mathbf{v}$, they are queued in Q (line 7). Moreover, the insertion of $\mathbf{v}$ may cause other existing segments (subsegments) of $\mathcal{X}$ missing in $\mathcal{D}_1$, they are queued in Q and will be recovered later (line 8). Then $\mathcal{D}_1$ is updated to a Delaunay tetrahedralization of the vertex set including $\mathbf{v}$ (line 9).

5.3.3 Proof of Termination

The termination of this algorithm can be proved by showing that the length of every subsegment is bounded by the local feature size of its endpoints divided by a constant depending only on the input.

SegmentRecovery ($\mathcal{X}$, $\mathcal{D}_0$)
// $\mathcal{X}$ is a three-dimensional PLS; $\mathcal{D}_0$ is the DT of vert($\mathcal{X}$).
1. $\mathcal{D}_1 = \mathcal{D}_0$;
2. Initialize a queue Q of all segments of $\mathcal{X}$;
3. **while** $Q \neq \emptyset$ **do**
4. get a segment $\mathbf{e}_i\mathbf{e}_j \in Q$; $Q = Q \setminus \{\mathbf{e}_i\mathbf{e}_j\}$;
5. **if** $\mathbf{e}_i\mathbf{e}_j$ is missing in $\mathcal{D}_1$, **then**
6. find a Steiner point $\mathbf{v} \in \text{int}(\mathbf{e}_i\mathbf{e}_j)$ by rule i, $i \in \{1, 2, 3\}$;
7. $Q = Q \cup \{\mathbf{e}_i\mathbf{v}, \mathbf{e}_j\mathbf{v}\}$;
8. $Q = Q \cup \{\sigma \in \mathcal{X}^{(1)}, \sigma \in \mathcal{D}_1 \,|\, \mathbf{v} \in \text{int}(B_\sigma)\}$;
9. update $\mathcal{D}_1$ to be the DT of vert($\mathcal{D}_1$) $\cup \{\mathbf{v}\}$;
10. **endif**
11. **endwhile**
12. **return** $\mathcal{D}_1$;

Figure 5.7: The segment recovery algorithm. The B_σ (in line 8) means the diametric circumball of the segment σ.

Theorem 5.3.1 *Let $\boldsymbol{e}_i\boldsymbol{e}_j$ be a subsegment,*

- *if $\boldsymbol{e}_i\boldsymbol{e}_j$ is type-1, then:*

  $\|\boldsymbol{e}_i - \boldsymbol{e}_j\| \geq \min\{lfs(\boldsymbol{e}_i), lfs(\boldsymbol{e}_j)\}$.

- *if $\boldsymbol{e}_i\boldsymbol{e}_j$ is type-2, let $\boldsymbol{e}_k = R(\boldsymbol{e}_i)$, then:*

  $\|\boldsymbol{e}_i - \boldsymbol{e}_j\| \geq \frac{1}{C} lfs(\boldsymbol{e}_k)$ *when* $\boldsymbol{e}_i = \boldsymbol{e}_k$,
  $\|\boldsymbol{e}_i - \boldsymbol{e}_j\| \geq lfs(\boldsymbol{e}_k)\sin(\theta)$ *when* $\boldsymbol{e}_i \neq \boldsymbol{e}_k$.

  where C is a bounded constant and θ is the smallest input angle.

hence algorithm SegmentRecovery *terminates.*

Proof Every inserted vertex has a reference point which is responsible for the insertion. Let $\mathbf{v}$ be an inserted vertex, and $P(\mathbf{v})$ be its reference point.

If $\mathbf{e}_i\mathbf{e}_j$ is type-1. If $\mathbf{e}_i$ is an inserted vertex, then $\mathbf{p}_i = P(\mathbf{e}_i)$ is either an input vertex or an inserted vertex on a segment which is not incident with $\mathbf{e}_i\mathbf{e}_j$. Rule 1 guarantees that $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \|\mathbf{e}_i - \mathbf{p}_i\| \geq \text{lfs}(\mathbf{e}_i)$. The same relation $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \|\mathbf{e}_j - \mathbf{p}_j\| \geq \text{lfs}(\mathbf{e}_j)$ holds for $\mathbf{e}_j$ if it is an inserted vertex ($\mathbf{p}_j = P(\mathbf{e}_j)$).

If $\mathbf{e}_i\mathbf{e}_j$ is type-2 and $\mathbf{e}_k = R(\mathbf{e}_i)$. It is a result of splitting the segment using either rule 2 or rule 3.

If $\mathbf{e}_i = \mathbf{e}_k$ ($\mathbf{e}_i$ is the acute vertex), let $\mathbf{e}_i\mathbf{e}_s$ be the segment containing $\mathbf{e}_i\mathbf{e}_j$. Without loss of generality, assume there are n segments incident at $\mathbf{e}_i$ (including $\mathbf{e}_i\mathbf{e}_s$), and let $\mathbf{e}_i\mathbf{e}_a$ be the shortest segment among them, then $\text{lfs}(\mathbf{e}_i) \leq \|\mathbf{e}_i - \mathbf{e}_a\|$.

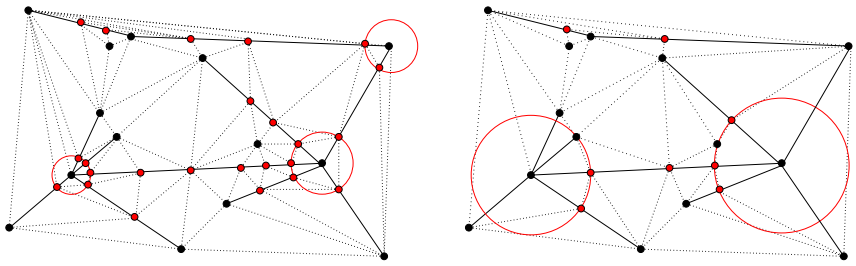


Figure 5.8: Comparison with other algorithm on a 2D PLS. The Steiner vertices are shown in red. The protecting balls around acute vertices are shown (in red). Left: a result of Shewchuk's algorithm [103], 24 Steiner points. Right: a result of our algorithm, 8 Steiner points.

Consider the worst case, that $\mathbf{e}_i\mathbf{e}_a$ is first halved by a splitting using rule 3, and due to multi-encroachment, $\mathbf{e}_i\mathbf{e}_s$ is also be split into $\mathbf{e}_i\mathbf{v}_1$ and $\mathbf{v}_1\mathbf{e}_s$. If $\mathbf{v}_1 = \mathbf{e}_j$, then $\|\mathbf{e}_i - \mathbf{e}_j\| = \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_a\| \geq \frac{1}{2}\text{lfs}(\mathbf{e}_i)$, else $\mathbf{e}_i\mathbf{v}_1$ will be split again. Assume the worst case again happens, i.e., $\mathbf{e}_i\mathbf{v}_1$ is halved by a splitting using rule 3 into $\mathbf{e}_i\mathbf{v}_2$ and $\mathbf{v}_2\mathbf{v}_1$. But the cause of inserting $\mathbf{v}_2$ must not due to $\mathbf{e}_i\mathbf{e}_a$ because the incident parts (at $\mathbf{e}_i$) of both segments have been split into the same length. We again consider the worst case, the insertion of $\mathbf{v}_2$ is caused by a split of another segment $\mathbf{e}_i\mathbf{e}_b$. If $\mathbf{v}_2 = \mathbf{e}_j$, then $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \frac{1}{4}\|\mathbf{e}_i - \mathbf{e}_1\| \geq \frac{1}{4}\text{lfs}(\mathbf{e}_i)$. If $\mathbf{e}_i\mathbf{e}_j$ still doesn't appear, the splitting procedure will continue. In the worst case, $\mathbf{e}_i\mathbf{e}_s$ is split at most n times, and each time is originally caused by a rule 3. Hence $\|\mathbf{e}_i - \mathbf{v}_n\| \geq \frac{1}{2n}\text{lfs}(\mathbf{e}_i)$. If $\mathbf{v}_n = \mathbf{e}_j$, then $C = 2n$.

Now it is possible that $\mathbf{v}_n \neq \mathbf{e}_j$, i.e., $\mathbf{e}_i\mathbf{e}_s$ will be split again. However, after at most n times rule 3 splits, the new split of $\mathbf{e}_i\mathbf{e}_s$ must not be caused by any incident segment at $\mathbf{e}_i$. Hence the cause of the split is either from a disjoint segment or an existing vertex. Since there are only finite number of input vertices and input segments, C is finite.

If $\mathbf{e}_i \neq \mathbf{e}_k$, then $\mathbf{p} = P(\mathbf{e}_i)$, and $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \|\mathbf{e}_k - \mathbf{e}_i\| \sin(\theta) \geq \text{lfs}(\mathbf{e}_i)\sin(\theta)$ (θ be the angle between $\mathbf{e}_k\mathbf{e}_i$ and $\mathbf{e}_k\mathbf{p}$). ■

Theoretically, the constant C in the above theorem could be very large. However, it will almost never happen in practice. In our experiment, the algorithm terminates within a few steps creating the protection ball sector. C is usually no larger than 4.

5.3.4 Comparison with Other Algorithm

Shewchuk proposed a segment recovery algorithm [103] for the same purpose. This algorothm proceeds in two steps. The first step uses protecting

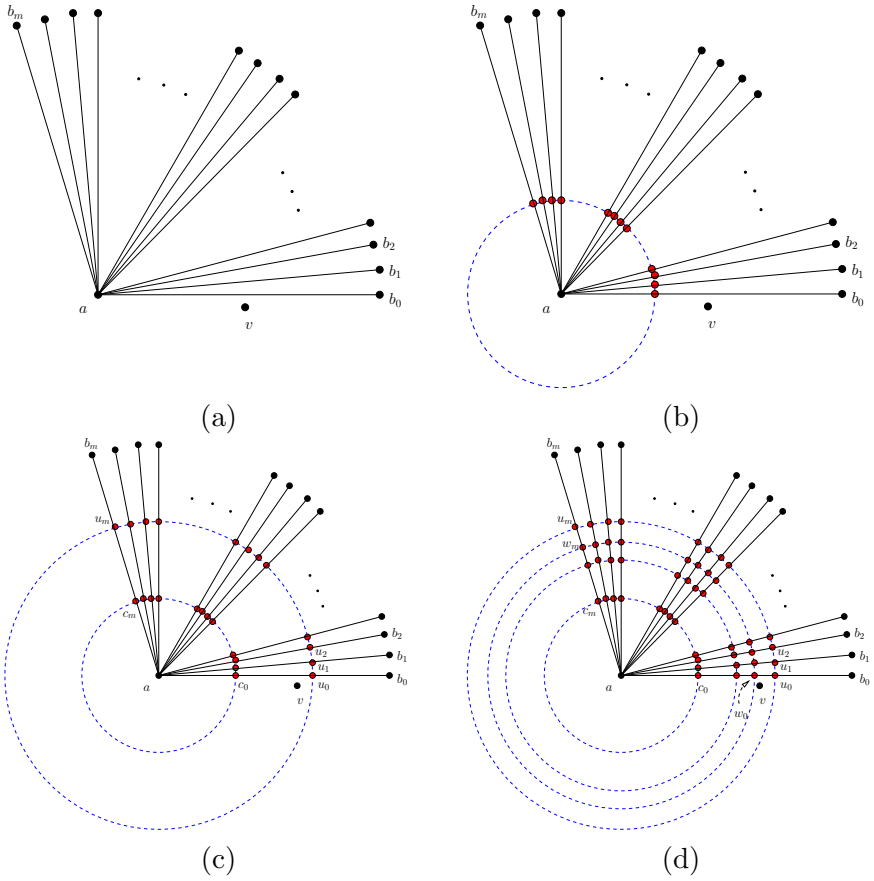


Figure 5.9: A counter example of the edge length claim of [103].

spheres centered at acute vertices of $\mathcal{X}$. Then places new points at where the segments and spheres intersect (see Fig. 5.8). The radii of these spheres are chosen priori to be not unnecessarily small and to guarantee that the subsegments inside the spheres are strongly Delaunay and no later Steiner points can be inside the spheres. The second step recovers the non-Delaunay segments by recursively bisection.

This algorithm is claimed to have provably good edge length bounds, i.e., unnecessary short edges are never created. The claim (in [103] Theorem 2) is that no output segment S has length shorter than $\text{lfs}(\mathbf{u})/4$, where $\mathbf{u}$ is any point in S. However, it is wrong. Below is a counter example.

The input PLS is shown in Fig. 5.9 (a). It consists of $m+1$ segments all meet at one single point $\mathbf{a}$. Assume that all segments have the same length. An additional point $\mathbf{v}$ lies slightly below the segment $\mathbf{ab}_0$, and let $\|\mathbf{a}-\mathbf{v}\| = \frac{2}{3}\|\mathbf{a}-\mathbf{b}_0\| - \epsilon$, where $0 < \epsilon << \|\mathbf{a}-\mathbf{b}_0\|$. The first step of the algorithm protects $\mathbf{a}$ by adding Steiner points $\mathbf{c}_0, ..., \mathbf{c}_m$ on segments (Shown in (b)) such that $\|\mathbf{a}-\mathbf{c}_m\| = \frac{1}{3}\|\mathbf{a}-\mathbf{b}_m\|$.

If $\mathbf{v}$ does not exist, then the algorithm stops and the claim holds. Now because of the presence of $\mathbf{v}$, segment $\mathbf{c}_0\mathbf{b}_0$ is non-Delaunay. The second step of this algorithm will bisect $\mathbf{c}_0\mathbf{b}_0$ by adding $\mathbf{u}_0$, and so do for $\mathbf{c}_i\mathbf{b}_i$ by adding $\mathbf{u}_i$, for $i \leq m$. (see (c)). One can easily see $\|\mathbf{c}_m - \mathbf{u}_m\| = 2^{-1}\,\frac{2}{3}\,\|\mathbf{a} - \mathbf{b}_m\|$.

The bisection process will continue until all subsegments are Delaunay. The final status is shown in (d). One can deduce that the length:

$$\|\mathbf{w}_m - \mathbf{u}_m\| = 2^{-k}\,\frac{2}{3}\,\|\mathbf{a} - \mathbf{b}_m\|,$$

where k is the number of bisections on each segment. In particular,

$$\frac{2}{3}\,\|\mathbf{a} - \mathbf{b}_m\| \geq \mathrm{lfs}(\mathbf{w}_m).$$

Put these together, we have

$$\|\mathbf{w}_m - \mathbf{u}_m\| \geq 2^{-k}\,\mathrm{lfs}(\mathbf{w}_m).$$

One can make k arbitrarily large by moving $\mathbf{v}$ arbitrarily close to $\mathbf{ab}_0$. Hence the edge length $\|\mathbf{u}_m - \mathbf{w}_m\|$ may not be bounded by $\mathrm{lfs}(\mathbf{w}_m)/4$.

Comparing with Shewchuk's algorithm, our algorithm does not need a pre-processing step for protecting the acute vertices. The protecting balls are automatically formed (see Fig. 5.6). Moreover, our algorithm tends to add fewer Steiner points. For example, in the above counter example, only $m + 1$ Steiner points are needed.

5.4 Facet Recovery

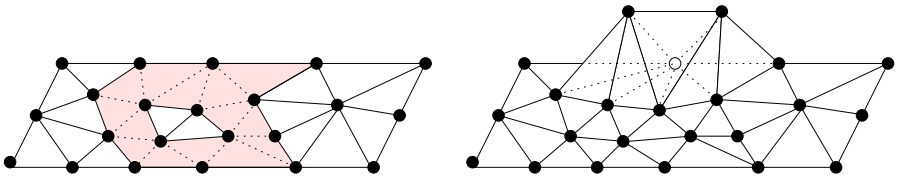
Given a CDT $\mathcal{D}_1$ of $\mathcal{X}^{(1)}$, the step *3* of our algorithm recovers the missing facets of $\mathcal{X}$ in $\mathcal{D}_1$. The assumption 5.2.1 that the vertex set of $\mathcal{D}_1$ is in general position is especially important. It guarantees that the facets of $\mathcal{X}$ can be recovered without using Steiner points.

5.4.1 The Algorithm

Each facet $F \in \mathcal{X}$ is first triangulated into a two-dimensional CDT $\mathcal{T}_F$ which includes the Steiner points inserted on segments of F. Hence each F is a union of subfaces of $\mathcal{X}$. Some subfaces may be missing in $\mathcal{D}_1$. The facet recovery algorithm incrementally recover missing subfaces of $\mathcal{X}$.

At initialization, let $\mathcal{D}_2 = \mathcal{D}_1$; add all missing subfaces of $\mathcal{X}$ into a queue Q. The algorithm iteratively recovers the subfaces in Q and update $\mathcal{D}_2$, it stops when Q is empty.

At each iteration i, several missing subfaces are recovered together. We define a *missing region* Ω to be a set of subfaces of $\mathcal{X}$ such that



(a) (b)

Figure 5.10: (a) The shaded area highlights a non-simply connected missing region Ω. (b) A cavity C at (step i) is illustrated.

(i) all subfaces of Ω belong to a facet of $\mathcal{X}$,
(ii) the edges on $\partial\Omega$ are edges of $\mathcal{D}_2$, and
(iii) the edges in $\text{int}(\Omega)$ are missing in $\mathcal{D}_2$.

Hence Ω is a connected set of missing coplanar subfaces. It may not be simply connected, i.e., Ω can contain a hole inside (see Figure 5.10 (a)). Each missing subface belongs to one missing region. A facet can have more than one missing regions.

When a Ω is found, one can derive a cavity in $\mathcal{D}_2$ by removing all its tetrahedra whose interiors intersect with Ω. This cavity can be further subdivided into two cavities by inserting the subfaces of Ω in it, see Figure 5.10 (b). Each resulting cavity is a polyhedron C which is not necessarily convex. The boundary $\text{bd}(C)$ of C is the union of triangular faces which are either subfaces of Ω or faces in $\mathcal{D}_2$.

The next step is to tetrahedralize each cavity C without using Steiner points. The TetrahedralizeCavity algorithm (given in Fig. 5.11) has two phases, *(1)* cavity verification (lines $2-13$), and *(2)* cavity tetrahedralization (lines $15-20$).

In phase *(1)*, each face $\sigma \subset \text{bd}(C)$ is verified. If σ is not Delaunay with respect to the vertex set of C, then C is enlarged by including the tetrahedron $\tau \in \mathcal{D}_2$, $\sigma < \tau$ and τ is removed from $\mathcal{D}_2$ (line 7). $\text{bd}(C)$ is also changed, hence the three faces of τ are added into Q_C for later processing (line 8). See Fig. 5.12 for an example. If σ is a subface of $\mathcal{X}$, the enlargement of C will cause σ missing from $\mathcal{D}_2$. For this reason, we have to queue σ in Q (lines $9-11$) so it will be recovered later.

Phase *(2)* first constructs the Delaunay tetrahedralization $\mathcal{D}_C$ of $\text{vert}(C)$ (line 16). By assumption 5.2.1, $\text{vert}(C)$ is in general position, hence $\mathcal{D}_C$ will include all faces contained in $\text{bd}(C)$. In other words, the interior of C is filled by a subset of tetrahedra of $\mathcal{D}_C$. The next step is to remove those tetrahedra which are not in $\text{int}(C)$ from $\mathcal{D}_C$ (lines $16-20$). Fig. 5.13 illustrates the idea of phase *(2)* in two dimensions.

The FacetRecovery algorithm is given in table 5.14. The termination and correctness of this algorithm are proved in the following sections.

```
TetrahedralizeCavity (D_2, C, Q)
// D_2 is a tetrahedralization; C is a cavity; Q is a queue.
 1.  // Phase (1): cavity verification.
 2.  initialize a queue Q_C of all faces contained in bd(C);
 3.  while Q_C ≠ ∅ do
 4.      get a face σ ∈ Q_C; Q_C = Q_C \ {σ};
 5.      if σ ⊂ bd(C) and σ is non-Delaunay wrt. vert(C), then
 6.          get the tetrahedron τ ∈ D_2 such that σ < τ;
 7.          C = C ∪ τ, D_2 = D_2 \ {τ};
 8.          Q_C = Q_C ∪ {ν < τ | ν ≠ σ};
 9.          if σ is a subface of X, then
10.              Q = Q ∪ {σ};
11.          endif
12.      endif
13.  endwhile
14.  // Phase (2): cavity tetrahedralization.
15.  form the Delaunay tetrahedralization D_C of vert(C);
16.  for each tetrahedron τ ∈ D_C, do
17.      if τ ⊈ int(C), then
18.          D_C = D_C \ {τ};
19.      endif
20.  endfor
21.  return D_2 = D_2 ∪ D_C;
```

Figure 5.11: Tetrahedralize Cavity Algorithm.

5.4.2 Proof of Termination

First, we need to show that the enlargement of the cavity C will terminate, i.e., the **do-while** loop (lines $3-14$) of the TetrahedralizeCavity algorithm will not run forever.

Lemma 5.4.1 *Suppose that the assumption 5.2.1 is satisfied. The* TetrahedralizeCavity *algorithm terminates.*

Proof The phase *(1)* (cavity verification) of the algorithm guarantees that the *(2)* can be correctly executed since every Delaunay simplex in vert(C) will appear in the Delaunay tetrahedralization of vert(C) (providing the fact that vert(C) is in general position). What remains here is to show that the **do-while** loop (lines $3-14$) must terminate.

Let $\mathcal{F}$ be the set of faces of $\mathcal{D}_2^{(2)}$ such that no face of $\mathcal{F}$ is crossed by any subfaces of $\mathcal{X}$. Clearly, $\mathcal{F} \neq \emptyset$ and any face $\sigma \in \mathcal{F}$ is Delaunay with respect to vert($\mathcal{D}_2$). It is also true that any face $\sigma \in \mathcal{F}$ is Delaunay with respect to vert(C). The set $\mathcal{F}$ limits the enlargement of C, i.e., C stops expanding at

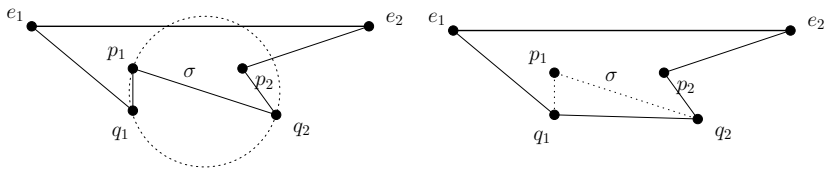


Figure 5.12: The verification of the cavity (illustrated in 2D). Left: $\mathbf{e}_1\mathbf{e}_2$ is the segment going to recover. The cavity C formed from $\mathbf{e}_1\mathbf{e}_2$ is a non-convex polygon. The edge $\mathbf{p}_1\mathbf{q}_2 \subset \text{bd}(C)$ is not Delaunay with respect to the vertex set of C. Right: C is enlarged by adding the triangle $\mathbf{q}_1\mathbf{q}_2\mathbf{p}_1$ into C. As a result, the edges $\mathbf{p}_1\mathbf{p}_2$ and $\mathbf{p}_1\mathbf{q}_1$ are not on $\text{bd}(C)$ anymore, and the edge $\mathbf{q}_1\mathbf{q}_2$ is included in $\text{bd}(C)$.

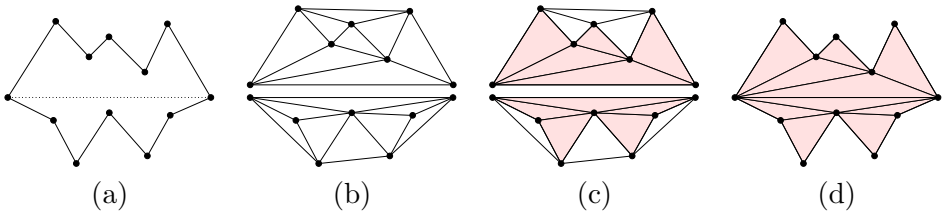


Figure 5.13: Cavity tetrahedralization (illustrated in 2D). (a) Two initial cavities separated by a constraining segment. (b) The two Delaunay triangulations are constructed at each side of the segment. (c) Classify triangles as "inside" or "outside". (d) Remove "outside" triangles.

a $\sigma \in \mathcal{F}$. Hence the **do-while** loop will not run forever. ■

There is a potential case of deadlock between the algorithms. In the TetrahedralizeCavity algorithm, after the recovery of a missing region Ω, some originally existed subfaces may be missing from $\mathcal{D}_2$, they are queued and will be recovered later (lines $9-11$). If the recovery of such subfaces (by the FacetRecovery algorithm) will cause Ω (or part of it) missing from $\mathcal{D}_2$ again, then a deadlock may appear. In the next lemma, we show that if the missing subfaces are recovered immediately after the recovery of Ω, the deadlock will never happen.

Lemma 5.4.2 *Let Ω be a missing region, C is one of the cavity formed from Ω. Suppose there is a subface $\sigma \subset C$ (hence it is missing from $\mathcal{D}_2$ after the recovery of Ω). Let Ω_σ be the missing region containing σ formed immediately after the recovery of Ω, and let C_σ be any of the two cavities formed from Ω_σ in the* TetrahedralizeCavity *algorithm. Then $C_\sigma \subset C$.*

Proof By the definition of a missing region, Ω_σ is contained in C (since all edges on $\text{bd}(C)$ are edges of $\mathcal{D}_2$). It further implies that the initial cavity

```
FacetRecovery (𝒳, 𝒟1)
// 𝒳 is a three-dimensional PLS; 𝒟1 is the CDT of 𝒳^(1).
 1.  𝒟2 = 𝒟1;
 2.  Initialize a queue Q of all subfaces of 𝒳;
 3.  while Q ≠ ∅ do
 4.      get a subface σ ∈ Q; Q = Q \ {σ};
 5.      if σ is missing in 𝒟2, then
 6.          form a missing region Ω containing σ;
 7.          form two cavities C1, C2 from Ω;
 8.          for each Ci, i = {1, 2} do
 9.              TetrahedralizeCavity(𝒟2, Ci, Q);
10.          endfor
11.      endif
12.  endwhile
13.  return 𝒟2;
```

Figure 5.14: The facet recovery algorithm.

C_σ formed from Ω_σ is contained in C. Note that C_σ may be enlarged in the TetrahedralizeCavity algorithm.

Since any face $\nu \subset \text{bd}(C)$ is Delaunay with respect to $\text{vert}(C)$, ν is also Delaunay with respect to the initial $\text{vert}(C_\sigma)$. Hence C_σ could not include any tetrahedron outside C since any face $\nu \subset \text{bd}(C)$ limits the enlargement of C_σ. This implies that $C_\sigma \subset C$. ■

Note that there is no ordering on the subfaces in Q in the FacetRecovery algorithm. While Lemma 5.4.2 only guarantees no deadlock if the algorithm performs a specific recovery ordering on the subfaces in Q. We thus prove a "weak" termination of the algorithm which depends on such a specific ordering. In general, the termination of this algorithm should not depend on any ordering. A random ordering in Q should be sufficient.

Theorem 5.4.3 *Suppose that the assumption 5.2.1 is satisfied. Moreover, the subfaces in Q are randomly ordered with one exception, that is, the subfaces queued in the* TetrahedralizeCavity *algorithm are always processed first. Then the* FacetRecovery *algorithm terminates.*

Proof Lemma 5.4.1 ensures that each missing region will be recovered. Lemma 5.4.2 and the specific ordering in Q together ensure that there is no two missing regions can form a deadlock. Hence Q will be empty and $\mathcal{D}_2$ contains all subfaces of $\mathcal{X}$. ■

5.4.3 Correctness

In this section, we establish the correctness of the FacetRecovery algorithm. That is, the result tetrahedralization $\mathcal{D}_2$ is a CDT of $\mathcal{X}$.

Theorem 5.4.4 *$\mathcal{D}_2$ is a CDT of $\mathcal{X}$.*

Proof The FacetRecovery algorithm is an iterative process. In each iteration, a missing region is recovered and $\mathcal{D}_2$ is updated. Let $\mathcal{D}_2^i$ denote the tetrahedralization after the recovery of i-th missing region, where $i = 0, 1, ..., m$. Hence initially, $\mathcal{D}_2^0 = \mathcal{D}_1$. Since the algorithm terminates, m is a finite number. We want to show that $\mathcal{D}_2^m$ is a CDT of $\mathcal{X}$.

We first show that $\mathcal{D}_2^1$ is a CDT. Tetrahedra of $\mathcal{D}_2^1$ which are outside the two cavities remain Delaunay. Let $\tau \in \mathcal{D}_2^1$ be a tetrahedron created inside a cavity C, and B_τ be its circumball. Let $\tau' \in \mathcal{D}_2^1$ be another tetrahedron sharing a face σ with τ. Let $\mathbf{v}$ be the vertex of τ' opposite to σ. We have the following cases:

(1) σ is a face inside C. Then $\tau' \subset C$, and B_τ must not contain $\mathbf{v}$ (guaranteed by the TetrahedralizeCavity algorithm).

(2) σ is a subface (hence $\sigma \subset \text{bd}(C)$). Then τ is constrained Delaunay even if B_τ contains $\mathbf{v}$, i.e, the inside of τ is not visible by $\mathbf{v}$. Otherwise, one can show that at least a segment of $\mathcal{X}$ is non-Delaunay, hence it could not exist in $\mathcal{D}_1$;

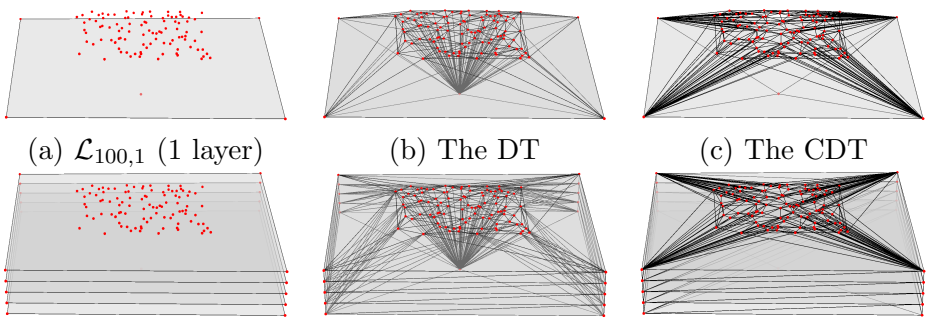
(3) σ is not a subface and $\sigma \subset \text{bd}(C)$. Then $\tau' \nsubseteq C$, and B_τ must not contain $\mathbf{v}$. Otherwise, $\mathcal{D}_1$ is not a Delaunay tetrahedralization since the circumball of τ' is not empty, i.e., it contains the vertex of τ opposite to σ.

By induction, after iteration i, where $i > 1$, $\mathcal{D}_2^i$ is a CDT. Now we want to show that $\mathcal{D}_2^{i+1}$ is a CDT. Using the similar arguments as above, the only difference is in the case *(3)* which is given below:

(4) σ is not a subface and $\sigma \subset \text{bd}(C)$. Then B_σ must not contain $\mathbf{v}$. Otherwise, $\mathcal{D}_2^i$ is not a CDT since the circumball of τ' contains the vertex of τ opposite to σ which is also visible from the interior of τ'.

Thus after all missing regions are recovered, $\mathcal{D}_2 = \mathcal{D}_2^m$ is a CDT of $\mathcal{X}$. ■

Theorem 5.4.3 and Theorem 5.4.4 together show that the FacetRecovery algorithm will construct a CDT of $\mathcal{X}$. This completes the proof of the correctness of the CDT algorithm given in Section 5.2.



(a) $\mathcal{L}_{100,1}$ (1 layer) (b) The DT (c) The CDT

(d) $\mathcal{L}_{100,5}$ (5 layers) (e) The DT (f) The CDT

Figure 5.15: Examples (Layers). The PLS $\mathcal{L}_{100,1}$ shown in (a) has 1 facet (1 layer), 100 vertices lie on top of the facet, one vertex below the center of the facet. The facet is missing in the Delaunay tetrahedralization of vert($\mathcal{L}_{100,1}$) shown in (b). The CDT of $\mathcal{L}_{100,1}$ is shown in (c). The PLS $\mathcal{L}_{100,5}$ shown in (d) is extended from $\mathcal{L}_{100,1}$ by including 5 parallel facets. (e) and (f) respectively show the Delaunay tetrahedralization and the CDT of $\mathcal{L}_{100,5}$.

5.4.4 Complexity

The complexity of the FacetRecovery algorithm is generally very hard to analysis. The main difficulty lies in the cavity verification phase of the CavityTetrahedralization algorithm, where some existing subfaces may be queued again and will be recovered later. Hence the number of total missing subfaces may be much larger than the initial size of the queue. While it is possible to analysis the general behavior of the algorithm by temporarily ignoring this issue.

The following theorem states the worst behavior of the FacetRecovery algorithm with respect to the number of vertices and facets of the input PLS.

Lemma 5.4.5 *Let $\mathcal{X}$ be a three-dimensional PLS which has v vertices. If the time for cavity verification is not considered, then one missing facet can be recovered in time $O(v^2 \log v)$.*

Proof We prove this lemma by constructing a PLS which needs such running time, then showing that it is indeed the worst case.

The PLS $\mathcal{L}_{v,1}$ (1-layer) shown in Fig. 5.15 (a) has only 1 facet which is a perturbed square, hence its four corners are in general position. $\mathcal{L}_{v,1}$ contains a set of v points (here $v = 100$) which are randomly distributed on top of the facet. The diameter of the facet is much larger than that of the point set. There is another point in $\mathcal{L}_{v,1}$, it lies straightly below the center of the facet. The Delaunay tetrahedralization of vert($\mathcal{L}_{v,1}$) is shown in (b). The facet of $\mathcal{L}_{v,1}$ is missing in it, there are exactly v edges (connecting top and bottom vertices) crossing the facet. The cavity formed from the missing facet has size $O(v)$. The CDT of $\mathcal{L}_{v,1}$ is shown in (c).

If we ignore the time for the cavity verification (lines $2-13$ in Fig. 5.11), and note that the removal of outside tetrahedra (lines $16-20$) takes linear time, the time for recovery of this facet is dominated by the time for constructing the Delaunay tetrahedralization of the set of v vertices. Note that there exist point sets with linear size Delaunay tetrahedralizations that reach quadratic intermediate size with positive constant probability. By using the randomized incremental flip algorithm [39], the worst time for constructing the Delaunay tetrahedralization is $O(v^2 \log v)$. Note that the largest possible size of a cavity is v. These together prove the claim. ■

Theorem 5.4.6 *Let $\mathcal{X}$ be a three-dimensional PLS which has v vertices and f facets. Assume the time for cavity verification is not considered, the* FacetRecovery *algorithm runs in time $O(fv^2 \log v)$.*

Proof Since the time for recovery of one facet may be $O(v^2 \log v)$, we just need to show that it is indeed possible that all facets of $\mathcal{X}$ may be missing and the recovery of any facet may require this time.

Let $\mathcal{L}_{v,f}$ be the PLS extended from $\mathcal{L}_{v,1}$ by including a set of f parallel facets. Fig. 5.15 (d) shows an example for $f = 5$. Clearly, the Delaunay tetrahedralization of $\text{vert}(\mathcal{L}_{v,f})$ will not contain the f facets, see (e). If the set of missing facets are recovered in an order which is from bottom to top, then the size of each cavity remains $O(v)$. This implies that the total time for recovery of the f facets is $O(fv^2 \log v)$. ■

Note that the order of the recovery of the missing facets is important. For example, if we reverse the recovering order in the above proof, i.e., missing facets are recovered from top to bottom. Only the top facet needs time $O(v^2 \log v)$, while the size of all other cavities is a constant (here is 4) which is much smaller than v. Hence if the missing facets are randomly ordered for recovery, the general behavior of the FacetRecovery algorithm is much better than the one given in Theorem 5.4.6.

5.5 Local Degeneracies Removal

Theoretically, the assumption 5.2.1 can be satisfied by applying symbolic perturbation [38, 103, 29] on the geometric predicates, e.g., *point-in-sphere* test, during the CDT algorithm. If the input is a finite set of vertices, and the point-in-sphere test is calculated exactly, then the symbolic perturbation is enough to satisfy the assumption.

Note that the facets of a PLS can be defined by any number of coplanar vertices. Hence the point-in-sphere test is essentially reduced to the *point-in-circle* test on the vertices of a facet. However, the latter may not be

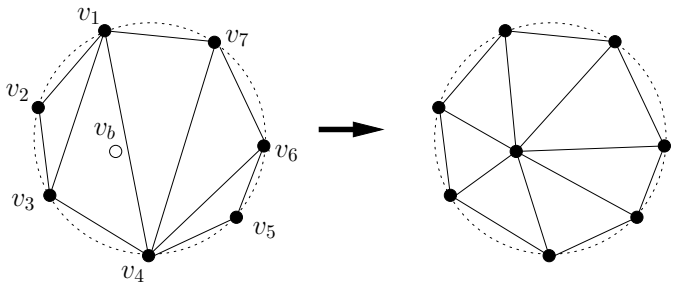


Figure 5.16: Local degeneracies and the break points. On the left, a set of seven coplanar vertices $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_7\}$ which share a common sphere. A triangulation of the vertices is shown. The subset $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ is a local degeneracy, while the subset $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_5\}$ is not. There are 4 local degeneracies (corresponding to the four interior edges) in this triangulation. $\mathbf{v}_b$ is a break point for removing the local degeneracies. On the right is a triangulation after $\mathbf{v}_b$ is inserted, no local degeneracy exists.

calculated exactly due to the possible input error, e.g., the vertices defining a facet may not be exactly coplanar. Even the input data is correct, it may not be represented exactly by the computer floating-point numbers [55]. For these reasons, a preprocessing on the input PLS is necessary. The purpose is to detect and break the (possible) degeneracies in the PLS. Steiner points (called *break point*) may be added.

5.5.1 Local Degeneracies and Break Points

Let $\mathcal{X}$ be a three-dimensional PLS. Let $F \in \mathcal{X}$ be a facet, and $\mathcal{K}_F$ be a triangulation of F. A set V of four vertices in F is called a *local degeneracy* if there are two triangles $\sigma, \nu \in \mathcal{K}_F$, where $\sigma \cap \nu \neq \emptyset$ and $V = \text{vert}(\sigma) \cup \text{vert}(\nu)$, such that the four vertices of V share a common sphere. See Fig. 5.16 left for examples of local degeneracies.

Once the triangulation of a facet is formed, the local degeneracies can be detected efficiently by locally checking all adjacent pairs of triangles. Let V be a local degeneracy. Let $\Sigma(\mathbf{c}, r)$ be the common sphere shared by the four vertices of V, where $\mathbf{c}$ and r are the center and radius of Σ, respectively. We introduce a Steiner point $\mathbf{v}_b$, called *break point*, such that $\mathbf{v}_b = \mathbf{c} + \epsilon\,(\mathbf{c} - \mathbf{v})$, where $\mathbf{v} \in V$ and $0 < \epsilon < 1$. v_b locates inside Σ and will break the local degeneracy after it is inserted in to the triangulation by a Delaunay algorithm. See Fig. 5.16 right for an example.

5.5.2 The Algorithm

The LocalDegeneracyRemoval algorithm is described in Fig. 5.17. Let $\mathcal{X}$ be a three-dimensional PLS. The triangulation $\mathcal{F}$ of all facets of $\mathcal{X}$ is

```
LocalDegeneracyRemoval (F)
// F is a triangulation of all facets of a three-dimensional PLS,
 1.   initialize a queue Q of all local degeneracies in F;
 2.   while Q ≠ ∅, do
 3.     get a set V ∈ Q; Q = Q \ {V};
 4.     if V is a local degeneracy in F, then
 5.       calculate a break point v_b for V;
 6.       if v_b encroaches upon any segment σ ∈ F, then
 7.         let v_b be the midpoint of σ;
 8.         Q = Q ∪ {V};
 9.       endif;
10.       update F to be a triangulation of F ∪ {v_b};
11.     endif
12.  endwhile
13.  return F;
```

Figure 5.17: The Local Degeneracy Removal algorithm.

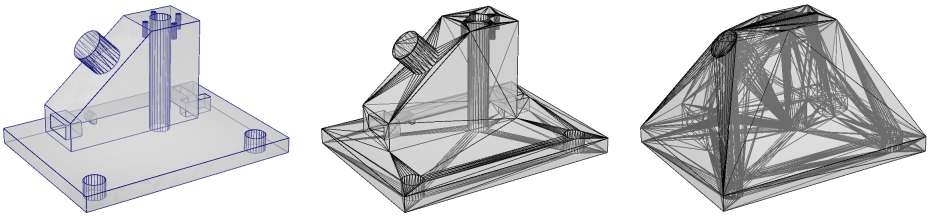
the input. The algorithm first initializes a queue Q containing all local degeneracies in $\mathcal{F}$. Then it runs in a loop until Q is empty.

For each local degeneracy of $\mathcal{F}$, the break point $\mathbf{v}_b$ is calculated. v_b may encroach upon one or more segments of the facet. In this case, $\mathbf{v}_b$ is shifted to the midpoint of an encroached segment. This avoids the case that $\mathbf{v}_b$ locates too close to a segment. $\mathcal{F}$ is then updated by including $\mathbf{v}_b$.

The success of this algorithm relies on the following hypothesis: the insertion of $\mathbf{v}_b$ does not create any new local degeneracy in $\mathcal{F}$. If we choose the parameter ϵ randomly for each $\mathbf{v}_b$, this hypothesis could be true in highly probability. If we ignore the time for updating $\mathcal{F}$ to include $\mathbf{v}_b$, this algorithm runs in linear time with respect to the initial size of Q, i.e., the number of local degeneracies in $\mathcal{F}$.

5.6 Examples

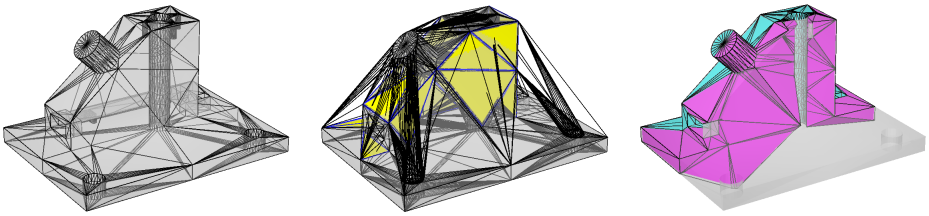
The CDT algorithm discussed in this paper has been implemented in the program `TetGen` [110]. Given a three-dimensional PLS, the LocalDegeneracyRemoval, SegmentRecovery, and FacetRecovery algorithms are called subsequently to create a CDT of that PLS. Note that the Steiner points will be only inserted on the boundary (segments and facets) of the PLS. A post-processing step for removing Steiner points on boundaries is also implemented. This step can be called optionally so that the output mesh can conform to the input surface mesh of the PLS with possibly few Steiner points inside the PLS.



(a) A 3D PLS
460 vertices,
328 facets

(b) The surface mesh
954 subfaces,
706 segments

(c) The DT
460 vertices,
2637 tetrahedra



(d) Segment recovery
213 break points
292 protect points

(e) Facet recovery
6446 subfaces
22 are missing

(f) A CDT
505 Steiner points
2738 tetrahedra

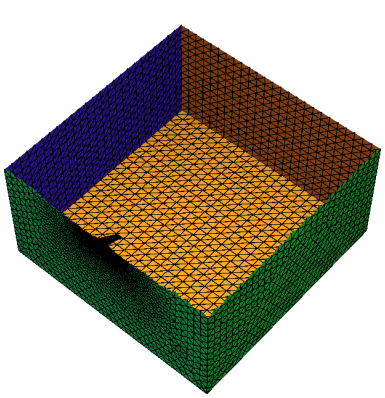
Figure 5.18: Example (`Cami1a`). The input PLS and the constructed CDT are shown in (a) and (f), respectively. Pictures from (b) to (e) show the intermediate status of the different steps of the CDT algorithm.

In the following, we provide several application examples to illustrate the practical behavior and the effectiveness of the CDT algorithm. Following the examples, we discuss the issues about the complexities of this algorithm and possible improvements.

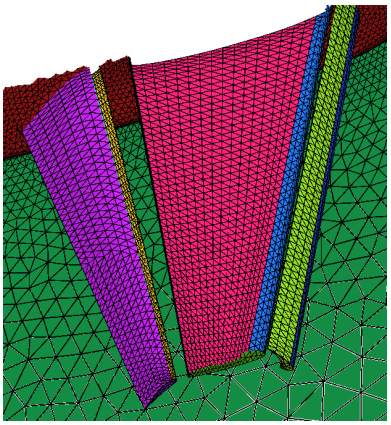
Fig. 5.18 illustrates an example of one run of the CDT algorithm on a mechanical part (Cami1a, available from [64]) with the intermediate status of the different steps. The input PLS shown in (a) has 460 vertices, 706 segments, and 328 facets. The surface mesh shown in (b), which is the input of the LocalDegeneracyRemoval algorithm, contains 954 subfaces. (c) is the initial Delaunay tetrahedralization of the vertex set. The status after the LocalDegeneracyRemoval and SegmentRecovery algorithms is shown in (d). The number of break points and protect points are 213 and 292, respectively. (e) shows the initial status of the FacetRecovery algorithm, there are total 6446 subfaces in which 22 are missing (highlighted in yellow). The resulting CDT is shown in (f). A vertical cut is made for visualizing the interior constrained Delaunay tetrahedra.

In the above example, the number of Steiner points is in the same order of the input size. This is the most often case that we have observed from our test inputs not having anisotropic features. Furthermore, the quadratic time behavior with respect to the number of Steiner points has never been

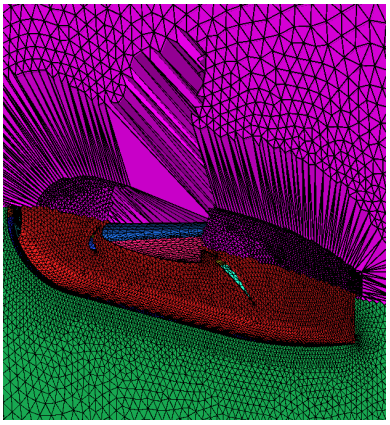
observed. Note that it may be further reduced the number of Steiner points by using appropriately point insertion order (currently it is random). Also note that the LocalDegeneracyRemoval algorithm may be called only when it is necessary. Current we call it in advance which may add at most $O(n)$ Steiner points (n is the input size).



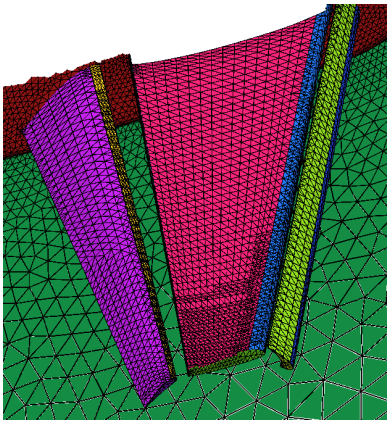
(a) A view of the input PLS
22905 nodes



(b) A view of the input surface mesh
45806 triangles



(c) A view of the output CDT
43044 nodes
134700 tetrahedra



(d) A view of the output surface mesh
2542 break points
17597 protect points

Figure 5.19: Example: `Wing-Iso`. A global and a local views of the input PLS are shown in (a) and (b), respectively. Two detailed views of the output CDT are shown in (c) and (d).

The geometry of the next example shown in Fig. 5.19 is the wing of an airplane placed inside a large bounding box, see (a). The surface of the wing and the bounding box were triangulated by 22905 nodes and 45806 triangles. A detailed view of the surface triangulation of the wing is shown in (b). To generate the CDT from the surface mesh, `TetGen` added total

	Heart	Fan	Crystal	Dragon	IFP
Input nodes	3,588	6,516	11,706	21,104	57,270
Input segments	11,205	19,709	37,785	63,461	172,001
Input facets	7,620	13,180	26,399	41,963	114,680
Break points	0	102	8	207	0
Protect points	3,675	5,017	10,894	1,817	1,963
Delaunay tetra.	0.22	0.41	0.81	1.61	4.85
Surface mesh	0.03	0.10	0.15	0.13	0.38
Degeneracy removal	0.10	0.18	0.42	0.65	1.80
Segment recovery	0.28	0.37	0.97	0.24	0.51
Facet recovery	0.06	0.07	0.64	0.12	0.31
Total time	0.69	1.13	2.99	2.75	7.85

Table 5.1: Runtime statistics of the CDT algorithm on selected examples. Tested by `TetGen` (compiled by g++ with -O3 option) on a linux workstation (Intel CPU 3.60GHz, 8GB). Times are reported in seconds.

20139 Steiner points in which 2542 are break points and 17597 are protect points. A view of the inside of the CDT near the wing is shown in (c). In (d), the modified surface mesh of the CDT is shown.

This example shows that the CDT algorithm is able to recover the boundary of certain complexity with a reasonable output size. From our tests, our implementation works nicely and efficiently when the surface mesh contains isotropically well-shaped triangles, no matter how complicated the geometrical shape is and how the mesh size is varying.

We next report the detailed running times of the CDT algorithm on some selected examples in Table 5.1. The input PLSs (whose boundary are triangular surface meshes) are available from the repository of `3D Meshes Research Database` maintained by INRIA's GAMMA project [64]. Two of the generated CDTs are shown in Fig. 5.20. Table 5.1 is divided into four parts: the input sizes (the number of nodes, segments, and facets) are reported in rows $2-4$, they are increasing from left to right; the number of break points and protect points are listed in rows $5-6$, respectively; then the running time statistics in individual steps of the CDT algorithm are given in rows $7-11$, the time is reported in seconds; and the total running time (which is the sum of the detailed times) is given in row 12.

From Table 5.1 we see that the majority Steiner points of these examples are inserted in the segment recovery step (line 10). Hence this step took the most time comparing to other steps. While the times for facet recovery (line 11) were relatively small which mean there were relatively small number of missing facets. The degeneracy removal step (line 9) uses linear time with respect to the input number of facets (line 4).

So far, the boundaries of our examples are formed by close to isotropic

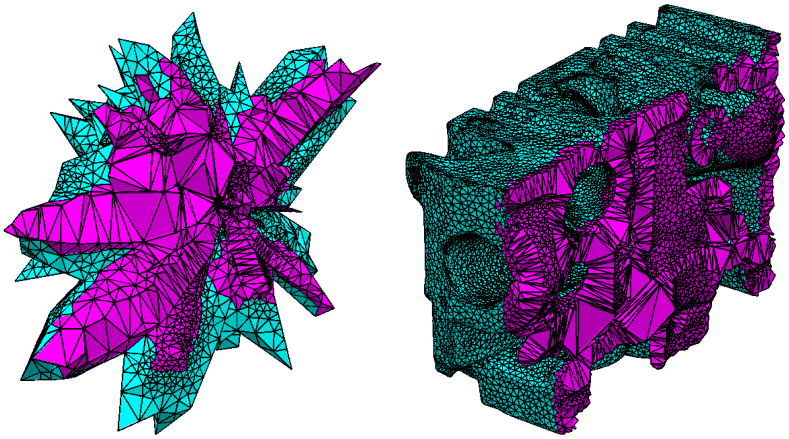


Figure 5.20: Examples: `Crystal` (left) and `IFP` (right). The generated CDTs are shown. See Table 5.1 for their statistics.

facets. Such inputs worked well with the CDT algorithm. The last example we want to present here has a surface mesh having anisotropic features, see Fig. 5.21 left. The input PLS has 50525 nodes and 101046 triangles. An enlarged view of the anisotropic triangles near the wing is shown in the middle. The generated CDT contains 181164 nodes including 6 break points and 130633 protect points. As we can see from the right figure, most of the Steiner points were inserted on the edges of anisotropic triangles. As a result, the input anisotropic features were destroyed. Although it is possible to remove these points by the post point removal process, it would be better to avoid inserting so many Steiner points on such features. This is not handled in the CDT algorithm. While it should be considered in the future.

5.7 Summary

Three-dimensional boundary recovery is a fundamental problem in Delaunay mesh generation. Theoretical questions of this problem like complexity, optimality, and output size are either NP-complete or still open. In this chapter, we proposed a practical and robust approach for solving this problem. Our approach is based on the construction of a constrained Delaunay tetrahedralization (CDT) for a three-dimensional PLS. We summarize this chapter in the following issues.

• We generalize the *constrained Delaunay triangulation* (CDT) of a PLS into any dimension. Our definition is, in some sense, more general than that of Shewchuk's [108]. We showed several fundamental properties of CDTs which are very close to those of Delaunay triangulations.

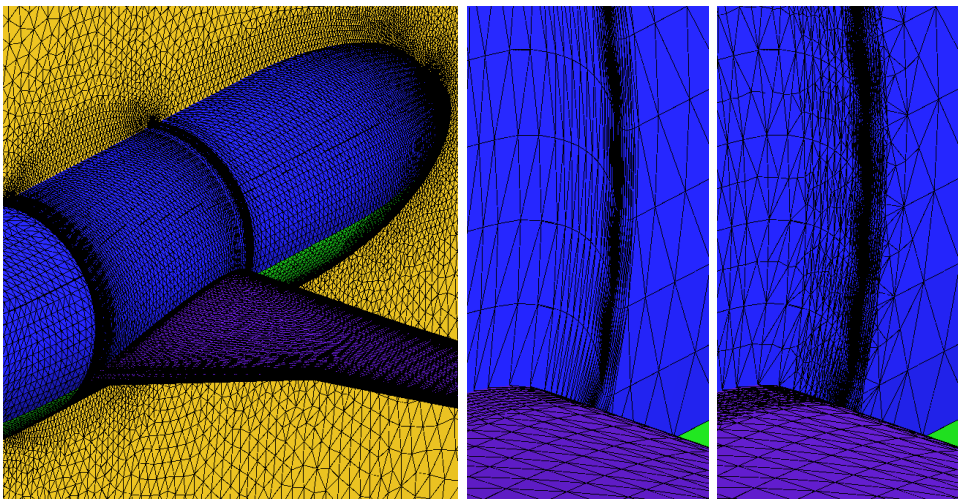


Figure 5.21: Example: `Wing-Aniso`. Left: A half plane. The surface mesh of the body and wing contains anisotropic triangles. Middle: An enlarged view of the anisotropic triangles near the wing. Right: A view of the result surface mesh of the CDT.

• We presented a practical algorithm for constructing a CDT of any three-dimensional PLS. Steiner points are added only on the boundary edges. The termination and correctness of the proposed algorithms are proved. We discussed the complexity issues of the algorithm.

For future work, some theoretical issues regarding this algorithm worth to be investigated.

◦ The complexity of the segment recovery algorithm with regard to the input size is not known yet. This is closely related to an open question [35] in computational geometry, i.e., the upper bound of the number of Steiner points needed to construct a conforming Delaunay tetrahedralization of a three-dimensioanl PLS?

◦ To limit the number of Steiner points is important since it directly influence the performance of the facet recovery algorithm. Some technical detail of the segment recovery algorithm need to be investigated. Currently, the segments are split in a randomized order. It generally works well, but sometimes results unnecessary Steiner points. It would be better to sort the segments and choose the ones which are grouped to split earlier than others.

Chapter 6

Conclusions

In this thesis, the problem of generating three-dimensional boundary conforming Delaunay meshes is discussed. The motivation of this study is to support numerically solving partial differential equations by finite element or finite volume methods. Three related meshing problems, i.e., Delaunay mesh generation, mesh adaptation, and boundary conformity, have been studied in detail. We reanalyzed a Delaunay refinement algorithm for Delaunay mesh generation and mesh adaption. We developed an algorithm for solving the boundary conformity problem.

Summary of Results

Based on the the basic geometrical requirement of a Voronoi finite volume method [43, 48], we defined a *boundary conforming Delaunay mesh* as a Delaunay mesh whose boundary simplices all satisfy the Gabriel property. It is so far the most concise definition for such objects.

We showed that a good aspect ratio boundary conforming Delaunay mesh of a three-dimensional piecewise linear system (PLS) $\mathcal{X}$ can be generated efficiently if no *facet angle* of $\mathcal{X}$ (i.e., the dihedral angle between any two facets in $\mathcal{X}$) is less than $\arccos \frac{1}{2\sqrt{2}} \approx 69.3^\circ$. This fact is proved by showing that the slightly modified Delaunay refinement algorithm of Shewchuk [99] terminates on any PLS satisfying this condition. Note that the 69.3° bound is only a sufficient condition. In practice, Shewchuk's algorithm usually terminates at an facet angle no less than 60°. Our reanalysis of Shewchuk's algorithm showed that this algorithm gives better theoretical guarantees on mesh quality and mesh size than those previously proven.

A simple variant of Shewchuk's algorithm is presented for generating isotropic adaptive Delaunay meshes of PLSs. It guarantees the termination on any PLS (no limitation on facet angle) and it inherits all theoretical properties of the original algorithm. However, a boundary conforming Delaunay mesh is not guaranteed if the input contains a facet angle less than 69.3°.

Boundary conformity is one of the bottlenecks in three-dimensional Delaunay mesh generation. We defined a *constrained Delaunay triangulation* (CDT) of a PLS $\mathcal{X}$ as a triangulation of $\mathcal{X}$ whose simplices are constrained Delaunay. This definition allows Steiner points in a CDT. It includes the definitions of Lee *et al.* [69] and Chew [23] (only in two dimensions) and Shewchuk [108] (in any dimensions) as special cases.

We presented an algorithm for constructing three-dimensional CDTs from arbitrary PLSs. It only uses Steiner points in the recovery of segments. Facet recovery requires no Steiner points. The termination of this algorithm is proved. We showed that the complexity of this algorithm can be determined if the number of Steiner points can be bounded. However, the upper bound of the number of Steiner points is not yet proved.

Outlook

The only obstacle for obtaining a boundary conforming Delaunay mesh is the limitation (e.g., $\geq 69.3°$) on facet angles of the input PLS. It seems that the only possible way to circumvent it, is to create points at the neighborhoods of small facet angles by special constructions. Such algorithms are proposed by Cheng *et al.* [20] and Pav *et al.* [91]. However, both approaches are complicated and may introduce arbitrarily many new points. A remaining problem is how to reduce the complexity for such a special construction, so that it is efficient.

Experiments show that Delaunay refinement can remove slivers by adding the same amount of Steiner points. However, it is not proved yet. It is an open problem to determine a non-trivial lower bound (or an expected bound) on the dihedral angles of the output tetrahedra.

Another open problem is to find an upper bound on the number of Steiner points for recovering all segments of a PLS $\mathcal{X}$ in a Delaunay triangulation. So far, only Edelsbrunner *et al.* [40] provided an upper bound for the problem in two-dimensional cases. It is necessary to analyze our segment recovery algorithm further or to develop a new algorithm which has asserted bounds on the resulting size of the CDT.

Despite the mentioned open questions, others exist too. A whole complex of very interesting ones is connected with anisotropic meshes and should attract more emphasis in the future.

Appendix A

Implementation

The meshing algorithms discussed and developed in this thesis have been implemented in the software `TetGen` [110] – a quality Delaunay tetrahedral mesh generator. The source code of `TetGen` is written in C++ and is freely available for academic and research use.

This chapter gives some implementation details including the structural description of the mesh domains, i.e., PLSs, the used mesh data structures, and the implementation of the CDT algorithm.

A.1 Description of Mesh Domains

At first we need to represent a mesh domain $\Omega \subset \mathbb{R}^3$ by a format that can be easily described and handled. In this work, Ω is approximated by a three-dimensional piecewise linear system (PLS) whose boundary consists of a set of vertices, segments, and facets. There are several possibilities to describe a PLS. In the following, we first review some concepts in geometric and solid modeling for describing three-dimensional solids. Then we introduce a simple boundary description of a three-dimensional PLS.

A.1.1 CSG and B-Rep Models

In geometric and solid modeling, CSG (constructive solid geometry) and B-Rep (boundary representation) are two popular representations for three-dimensionl objects, see e.g., [83, 76, 60].

A CSG model implicitly describes Ω as combination of simple primitives or other solids in a series of Boolean operations. The CSG model can describe rather complicated shapes simply. However, the domain boundary $\partial\Omega$ must be calculated numerically in order to find the intersecting points, curves, and patches. This involves solving many non-linear equations in three variables. The successfully obtaining a PLS from a CSG model is generally not a simple task.

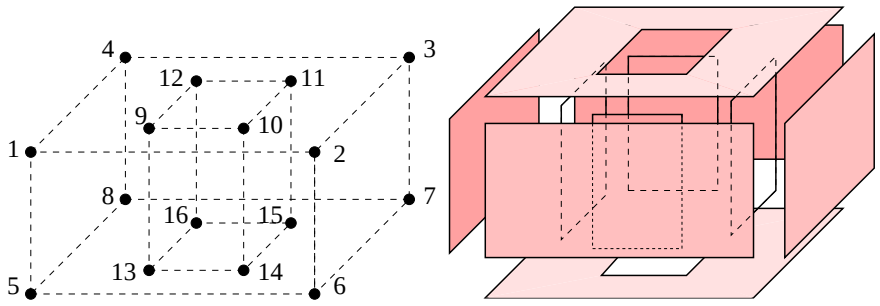


Figure A.1: A description of a three-dimensional PLS (Handle). The list of vertices (left) and the list of facets (right) are shown.

A B-Rep model explicitly describes $\partial\Omega$ by a set of non-overlapping facets (may be curved surfaces) together with topological informations (such as incidence and adjacency) between the facets. The volume of Ω is implicitly bounded by them. However, it is not trivial to correctly define such a model for a complex object. Whatever, a B-Rep model with only linear facets is eligible to describe the boundary of a PLS.

Popular B-Rep data structures are *winged-edge* [7], *doubly-connected edge list* [86], and *quad-edge* [58]. The quad-edge data structure is flexible to model 2-manifolds and their duals. However, it can not describe non-manifolds, which is possible in the winged-edge data structure. The *edge-facet* data structure [31] extends the quad-edge data structure for manipulating three-dimensional cell complexes and subdivisions. It can describe the boundaries of a PLS which are non-manifolds. However, it does not handle facets which are not simply connected.

A.1.2 A PLS Boundary Description

Let $\mathcal{X}$ be a three-dimensional PLS. This section presents a description of $\partial\mathcal{X}$ which includes a list of vertices, a list of facets, a list of volume hole descriptions, and possibly a list of subregion descriptions, see Fig. A.1 for an example. The central part is the description of each facet, which is a two-dimensional PLS, i.e., it may contain facet holes, isolated segments and vertices. Our description can be viewed as a B-Rep without topological information, i.e., there are no informations like incidences and orientations about facets. This makes the description simple and it can describe non-manifolds. The topological information can be recovered and validated from this description during later processing.

Vertex Description

Each vertex $\mathbf{p}$ in the list of vertices contains three floating point numbers to store its coordinates. $\mathbf{p}$ may additionally contain a number k of attributes

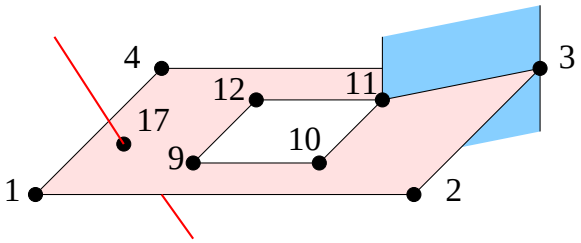


Figure A.2: The facet (shown in pink) consists of four polygons and one hole. The ordered vertex lists of the polygons are: $(1, 2, 3, 4)$, $(9, 10, 11, 12)$, $(11, 3)$, and (17). The last two polygons are degenerate.

which are floating point numbers. The meaning of these attributes depends on the applications. For instance, they may represent a weight ($k = 1$), or a vector ($k = 3$), or a metric tensor ($k = 6$), etc. In addition, the user can assign **p** a boundary marker (an integer). In FEM or FVM, the boundary marker may represent a specific boundary condition assigning at this point.

Facet Description

Each facet F in the list of facets is used to represent a two-dimensional PLS. It may contain facet holes, isolated segments and vertices. F is described by a list of polygons and a list of facet holes.

Each polygon of F is described by an ordered list of vertices (using their indices in the vertex list). The order of the vertices can be in either clockwise or counterclockwise order. A polygon may be *degenerate*, i.e., it may contain only one or two vertices. A degenerate polygon is used to represent an isolated vertex or segment in F, see Fig. A.2.

Since F may contain holes in it, each facet hole C of F is given in the facet hole list. Note that C is topologically equivalent to a disk. C is specified by choosing an arbitrary point **v** in the interior of C. **v** acts like a signal point. It informs `TetGen` that its location is at the interior of a facet hole. Note that **v** s not a vertex of F. See Fig. A.2 for an example.

A facet is also assigned a boundary marker (an integer). The boundary marker of a facet has several usages. For example, it may associate a boundary condition to the facet for FEM or FVM calculations.

Remark. Since a facet F is a two-dimensional PLS, all vertices of F must lie in the same affine subspace that F belongs. Unless F is a triangle, it is generally impossible to enforce this condition due to the floating-point numbers used in computer. The technique of *Epsilon geometry* [57] is applied, i.e., the facet is "fattened" by $\epsilon \geq 0$. The facet hole point **v** is not required to be exactly coplanar with F. **v** is acceptable as long as the orthogonal projection of **v** onto F lies inside the facet hole.

Volume Hole Description

A volume hole C of a PLS $\mathcal{X}$ is given in the volume hole list. Note that here C is topologically equivalent to a ball. Similar to the specification of the facet holes, a volume hole is specified by choosing an arbitrary point $\mathbf{v}$ in the interior of C. Note that $\mathbf{v}$ is not a vertex of $\mathcal{X}$.

Sub-Region Description

A PLS $\mathcal{X}$ may contain more than one subregion separated by internal facets. Subregions can be used to modeling different materials. Similar to specify the volume holes, a sub-region is specified by choosing an arbitrary point $\mathbf{v}$ lies in its interior. Each subregions must assign a marker (an integer) and possibly an attribute (a floating-point number).

Remark. Unlike the volume holes which must be specified on the input, the specification of subregions is not mandatory. Once $\mathcal{X}$ is tetrahedralized, the subregions can be automatically determined.

A.1.3 Data File Description

A data file `filename.poly` can be used to describe the boundary of a PLS, where `filename` is the base name, and `.poly` is the extension name for the PLS boundary file format. A detailed description of the file format is found in the User's manual of `TetGen` [109]. Fig. A.3 illustrates a data file, `handle.poly`, which describes the boundary of PLS shown in Fig. A.1. Characters of the same line after a '#' are comments.

The `.poly` file format is suitable for describing the boundaries of any PLS. However, certain PLSs only needs a simplified description. For example, each facet of a PLS may have only one polygon (no hole), or just one triangle. Several file formats exist for describing such PLSs, such as `.off`, `.ply`, `.mesh`, and `.stl` formats (links for the descriptions are found in[110]). These formats are supported by `TetGen`.

`TetGen` uses a set of zero-based arrays to store the list of vertices and facets as well as their attributes and boundary markers. These arrays are declared within the class `tetgenio`. Functions are available for reading (and writing) the file in `.poly` as well as other supported formats into (and out of) these arrays.

A.2 Mesh Data Structures

A tetrahedral mesh $\mathcal{T}$ of a mesh domain Ω is a three-dimensional simplicial complex. $\mathcal{T}$ contains a subcomplex $\mathcal{K}$ which is a triangular mesh of the boundary $\partial\Omega$. It is convenient to explicitly represent $\mathcal{K}$ separately from $\mathcal{T}$. `TetGen` combines two mesh data structures to store and dynamically update $\mathcal{T}$ and $\mathcal{K}$, respectively.

```
# Num of vertices, dimension, no attribute, no boundary marker.
  16  3  0  0
# Begin of the list of vertices.
  1  0.0 0.0 0.6  # index, x, y, z
  2  1.0 0.0 0.6
  3  1.0 1.0 0.6
  4  0.0 1.0 0.6
  ...

# Num of facets, no facet marker.
  10  0
# Begin of the list of facets.
# The top facet
  2  1  0    # 2 polygons, 1 hole, no boundary marker
  4  1  2  3  4     # A cycle (1,2,3,4) of length 4.
  4  9  10  11  12
  1  0.5  0.5  0.6  # A hole point given by its coordinates
# The four outer facets
  1  0  0
  4  1  2  6  5
  1  0  0
  4  2  3  7  6
  1  0  0
  4  3  4  8  7
  1  0  0
  4  4  1  5  8
  ...

# Num of volume holes
  0
# Num of sub-regions
  0
```

Figure A.3: A PLS data file description. An example file, `handle.poly`, describes the boundary of PLS shown in Fig. A.1. The whole file is available from `TetGen`'s website, in the file format examples page.

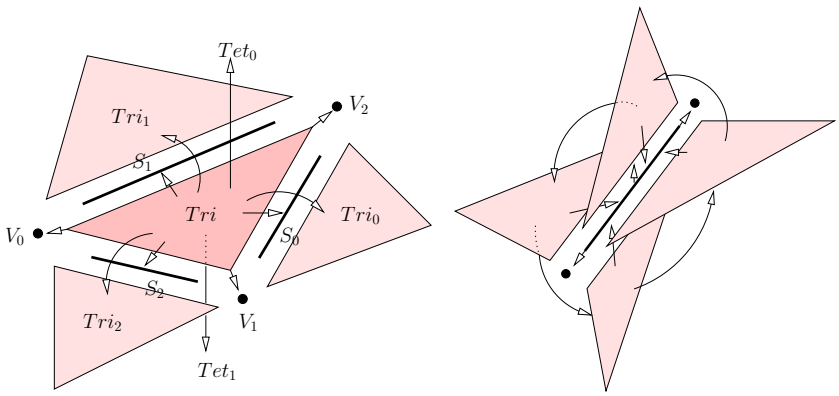


Figure A.4: The triangle-based data structure. Left: The data fields of the triangle Tri are illustrated as arrows. Right: The connections between triangles and a shared segment. The face ring is also shown.

A.2.1 Triangle-Edge Data Structure

The *triangle-edge* data structure introduced by Mücke [84] is used to represent a triangular mesh of $\partial\Omega$. It is a simplified version of the Dobkin and Laszlo's *edge-facet* data structure [31]. Several modifications in the original data structure are made for fitting our purposes. We refer to Mücke's thesis [84] for a detailed description of the original data structure. Below we give a brief description of our modified version of this data structure.

The triangle-edge data structure is triangle-based. Given a triangular mesh $\mathcal{K}$ of $\partial\Omega$, it stores the set of triangles in $\mathcal{K}$ and the connectivities between them. It also stores the connectivities to the segments of $\partial\Omega$ and the adjacent tetrahedra of $\mathcal{T}$. Hence, each triangle $Tri \in \mathcal{K}$ is declared as a structure containing the following data fields (see Fig. A.4 left):

- three vertices: V_0, V_1, V_2;
- three adjacent triangles: Tri_0, Tri_1, Tri_2;
- three adjacent segments: S_0, S_1, S_2;
- two adjacent tetrahedra: Tet_0, Tet_1.

These data fields are all C++ pointers. V_0, V_1, V_2 are pointers to the vertices in a vertex array (a dynamic list) in which the coordinates and attributes of the vertices are stored.

The original data structure uses six pointers for adjacent triangles such that they form a doubly linked list at each edge of Tri. To save memory, we only apply a single link list (in one direction) at each edge, i.e., each pointer points to the next adjacent triangle in the list. Although the number of

adjacent triangles at a segment of $\partial\Omega$ may be arbitrary, each interior edge has only two adjacent triangles.

The segments of $\partial\Omega$ need to be explicitly represented. For this purpose, each Tri is allocated three pointers to segments attached at its edges. Some segments in $\{S_0, S_1, S_2\}$ may not be used if the corresponding edges of Tri are interior edges, such segments are set to null pointers. This causes redundancy in the memory usage. However, the cardinality of $\mathcal{K}$ is usually much smaller than that of $\mathcal{T}$, hence the amount of memory used to store $\mathcal{K}$ is not dominate. In our implementation, there is no special structure declared for segments. Each segment is represented as a degenerated triangle (by letting one of its vertices to be a null pointer). Each segment only has one pointer to a triangle containing it as an edge, see Fig. A.4 right.

The pointers Tet_0 and Tet_1 serve the connections between the tetrahedral mesh $\mathcal{T}$ and the surface mesh $\mathcal{K}$. Each Tri determines two closed halfspaces, denoted as Tri^+ and Tri^-, one at each side of Tri. We assume that $Tet_0 \subset Tri^+$ and $Tet_1 \subset Tri^-$.

Assuming the usual 32 bits pointers, this gives a total of 44 bytes for a triangle structure. The original data structure uses 36 bytes without the fields of segments and tetrahedra.

The atomic unit for traversing a triangular mesh is a triangle-edge pair, denoted as (T, v), where T is a pointer to a triangle Tri and v is an integer between $[0, 5]$, called *edge version*. Each triangle-edge pair determines one of the six directed edges incident to Tri, refer to the Figure 3.1 in [84].

In the original data structure, every directed edge of Tri belongs to two rings: an *edge ring*, which traverses all directed edges of Tri having the same direction; and a *face ring*, which traverses all adjacent triangles having the same directed edge, refer to the Figure 3.2 in [84]. In our modified data structure, the edge ring is the same. Because we only use a single link list at each edge of Tri, one always traverses in one face ring regardless of the edge direction. At each interior edge of a Tri, this makes no difference (since there are only two triangles in its face ring). It may take more time on traversing the triangles around a segment.

A.2.2 Tetrahedron-based Data Structure

The *tetrahedron-based* data structure introduced by Shewchuk [99] is eligible for representing tetrahedralizations. We adopted this data structure and combined it with the triangle-edge data structure (in Section A.2.1) to represent and dynamically maintain tetrahedral meshes. Given a tetrahedral mesh $\mathcal{T}$ of Ω, it stores the set of vertices and tetrahedra of $\mathcal{T}$ and the connections between them. It also stores the connections between $\mathcal{T}$ and the boundary mesh $\mathcal{K}$ of $\partial\Omega$.

The set of vertices of $\mathcal{T}$ is stored as a dynamic list of vertices. Each *vertex* is a structure containing its coordinates and attributes.

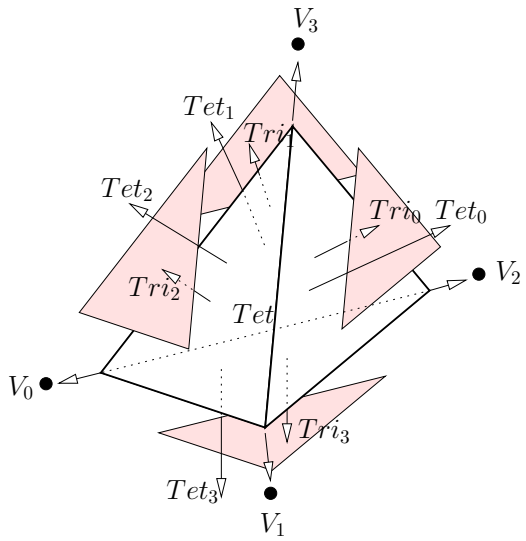


Figure A.5: The tetrahedron-based data structure. The data fields of the tetrahedron Tet are illustrated as arrows.

The set of tetrahedra of $\mathcal{T}$ is stored as a dynamic list of tetrahedra. Each *tetrahedron* $Tet \in \mathcal{T}$ is a structure containing the following data fields (which are C++ pointers):

- four vertices: V_0, V_1, V_2, V_3,
- four adjacent tetrahedra: Tet_0, Tet_1, Tet_2, Tet_3,
- four adjacent subfaces: Tri_0, Tri_1, Tri_2, Tri_3.

In this structure, each tetrahedron has four pointers to its vertices, four pointers to its adjacent tetrahedra, and four pointers to its adjacent subfaces which are triangles from the surface mesh of $\partial\Omega$, see Fig. A.5 left.

The four pointers to subfaces serve the link from tetrahedral mesh to its boundary mesh, Note that there is no direct connections between tetrahedra and segments. These connections can be recovered from the pointers in triangle-edge data structure.

Some subfaces are not used if the corresponding faces of the tetrahedron are interior faces. They are set to null pointers. Usually the majority faces of tetrahedra in $\mathcal{T}$ are interior faces. For saving the memory, the four subfaces are stored in a separated dynamic list. In the structure of a tetrahedron, only one pointer is allocated for pointing to its subfaces in the list. If all faces of the tetrahedron are interior faces, this pointer is null.

Assuming the usual 32 bits pointers, this gives a total of 36 bytes for a tetrahedron structure.

A.3 Implementing the CDT Algorithm

This section gives some implementation details of the CDT algorithm discussed in Chapter 5.

Given a PLS $\mathcal{X}$, a CDT of $\mathcal{X}$ is constructed in the following steps:

(1) Form the Delaunay tetrahedralization $\mathcal{T}$ of vert($\mathcal{X}$).
(2) Form the surface triangulation $\mathcal{F}$ of $\partial\mathcal{X}$.
(3) (optional) Perturb the vertices in $\mathcal{F}$.
(4) Recover the segments of $\mathcal{F}$ in $\mathcal{T}$.
(5) Recover the subfaces of $\mathcal{F}$ in $\mathcal{T}$.
(6) Remove tetrahedra outside $|\mathcal{X}|$ from $\mathcal{T}$.

In the following, we discuss the implementations of some of the key steps in `TetGen`. The functions are found in the class `tetgenmesh`.

A.3.1 Tetrahedralizing the Vertices

The randomized incremental flip algorithm of Edelsbrunner and Shah [39] is implemented to construct the Delaunay tetrahedralization $\mathcal{T}$. This algorithm runs in expected time $O(n \log n + n^{\lceil d/2 \rceil})$ for n input vertices. It is generally very efficient.

A robust implementation based on a triangle-edge data structure is discussed in detail by Mücke [84]. Our implementation of this algorithm is exactly the same as Mücke's, but it is based on the tetrahedron-based data structure discussed in Section A.2.2.

The robustness of the implementation is achieved by using the adaptive exact floating-point arithmetic of Shewchuk [97] to perform the two geometric predicates, i.e., the three-dimensional orientation test (`orient3d`) and the point-in-sphere test (`insphere`). In addition, a simple symbolic perturbation approach [100] is implemented in the point-in-sphere test to handle degenerate inputs.

The efficiency of the algorithm largely depends on the speed of point location. The simple and efficient "jump-and-walk" algorithm by Mücke *et al.* [85] is implemented in function `locate`. This algorithm needs not preprocessing and additional storage, and runs in expected time close to $O(n^{1/(d+1)})$ per point location.

The randomized incremental approach may not work efficiently when the input data set becomes very large. A pre-sorting on the point insertion order is expected to improve the efficiency and the local memory access [3].

A.3.2 Triangulating the Boundaries

The surface mesh $\mathcal{F}$ of $\partial\mathcal{X}$ is created in two steps: *(i)* triangulate each facet of $\mathcal{X}$ separately; and *(ii)* recover the incidence and adjacency relations between the facets (see Fig. A.6). The is done by the function `meshsurface`.

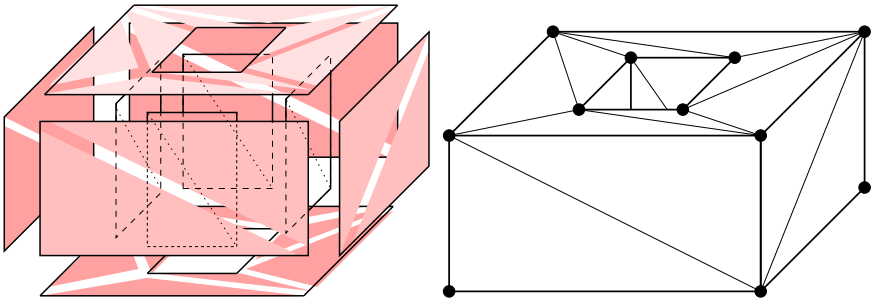


Figure A.6: Triangulating the facets. Left: each facet of the PLS is triangulated separately. Right: the surface mesh is formed after the unifying the common segments at each facets.

Triangulate a Facet

The boundary of each facet $F \in \mathcal{X}$ is a two-dimensional PLS. The pure CDT of F exists and can be constructed efficiently. For instance, Chew proposed a divide-and-conquer algorithm [22] that can build such a CDT in optimal $O(n \log n)$ time. Seidel [96] also sketches a time optimal algorithm based on Fortune's sweep-line technique [45].

Our approach to build the CDT of F is adapted from the approach used by Shewchuk in his program `Triangle` [98]. It works in three steps. The first step creates the Delaunay triangulation $\mathcal{D}$ of the vertices of F. The incremental flip algorithm of Guibas *et al.* [56] is used. It takes expected time $O(n \log n)$. The second step is to recover the missing segments of F in $\mathcal{D}$. Lawson's edge flip algorithm [68] is adapted for this job. Each missing segment is recovered by a sequence of edge flips. Once a segment appears in $\mathcal{D}$, it will never be flipped again. The total number of flips is fewer than $\binom{n}{2}$. In the last step, triangles not belong to F are removed (this also creates the holes in F). The function `triangulate` realizes this approach. The three steps are implemented as functions (`incrflipdelaunaysub`, `recoversegment`, and `carveholessub`) called subsequently in `triangulate`.

Our implementation of the above approach does not need to project the vertices into a plane. It is important to correctly decide whenever an edge flip is needed. That is, let $\mathbf{ab} \in F$ be an edge, and $\mathbf{abc}, \mathbf{abd} \in F$ be two triangles sharing at $\mathbf{ab}$, the *question* is whether or not $\mathbf{d}$ is inside the circumcircle of $\mathbf{abc}$. Since $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{R}^3$, the point-in-circle test does not apply. Instead, one can use the point-in-sphere test by find a point $\mathbf{e} \in \mathbb{R}^3$ of the PLS, such that $\mathbf{e}$ does not lie on the plane containing F. However, inconsistency may happen when $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ are not strictly lie on F.

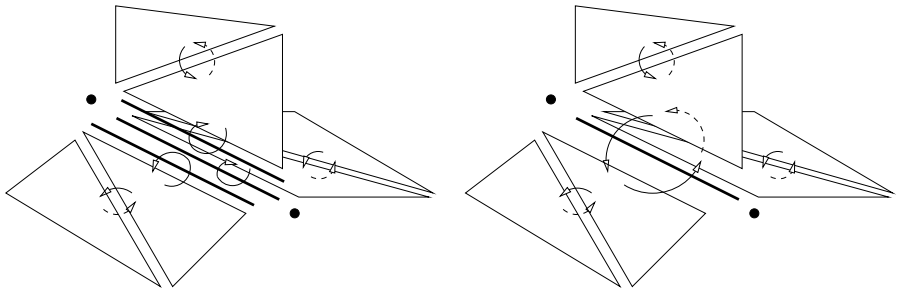


Figure A.7: Connect incident facets. Left: three facets are incident, each CDT of the facet contains a copy of the same segment. Right: two of the three identical segments are removed, and the face ring of the remaining segment is reformed.

Connect Incident Facets

After the first step, each facet is represented by a CDT of it. However, there are no connections between incident facets yet, i.e., if $k \geq 2$ facets are incident along a segment of $\mathcal{X}$, there are k copies of this segment in each of their CDTs, see Fig. A.7 left. The next task is to keep a unique segment of $\mathcal{X}$, and remove other duplicated segments. This is a combinatorial search which can be done efficiently. During the removing of the duplicated segments, the face ring at this segment is created, see Fig. A.7 right. Hence all incident facets are connected each other through the segment. This is done in the function `unifysegment`.

A.3.3 Recovering the Segments

Here the inputs are: a Delaunay tetrahedralization $\mathcal{T}$ and a surface mesh $\mathcal{F}$ of $\partial\mathcal{X}$. $\mathcal{T}$ and $\mathcal{F}$ share the same set of vertices of $\mathcal{X}$. Some segments of $\mathcal{F}$ may not exist in $\mathcal{T}$. The function `delaunizesegments` implements the segment recovery algorithm described in Section 5.3.

For each segment of $\mathcal{X}$, the segment type (sharp or non-sharp) is determined from the type (acute or non-acute) of its endpoints. A pre-process for the segment recovery algorithm is to detect and save the types of the vertices. This process can be completed in $O(n)$ time through the face rings constructed at segments. (See function `markacutevertices`).

Since a newly inserted Steiner point may cause some existing segments (and subsegments) non-Delaunay (so they must be split). For efficiently detecting such segments (and subsegments), we let $\mathcal{T}$ contain extra pointers to the segments of $\mathcal{F}$. The tetrahedron-based data structure is extended. Each tetrahedron contains six extra pointers to store the segments attached at its edges. This uses more memory which will be released on finishing of this algorithm. Hence if a segment of $\mathcal{F}$ matches an edge of $\mathcal{T}$, every

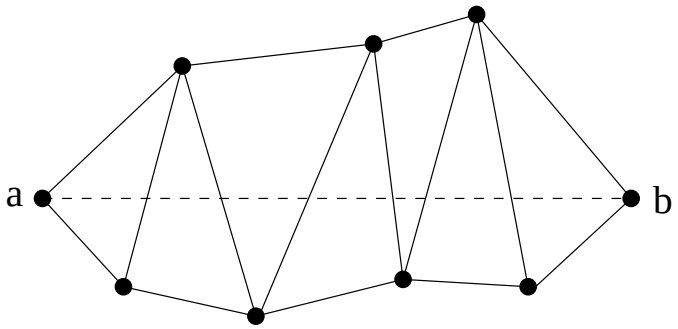


Figure A.8: Searching for the reference point of a missing segment (explained in 2D). The set of tetrahedra (here are triangles) that intersect with the segment is shown. The reference point is chosen from the set of vertices of the faces that the segment crosses.

tetrahedra of $\mathcal{T}$ sharing at this edge is assigned a pointer to this segment, we say that the segment is *inserted* into $\mathcal{T}$.

The first step of the segment recovery algorithm is to collect a queue Q of all missing segments. Let $\mathbf{ab}$ be a segment to be matched in $\mathcal{T}$. The *star* of $\mathbf{a}$, $\mathrm{St}(\mathbf{a})$, contains all simplices of $\mathcal{T}$ that contain $\mathbf{a}$, and the *link* of $\mathbf{a}$, $\mathrm{Lk}(\mathbf{a})$, contains all faces of simplices in $\mathrm{St}(\mathbf{a})$ that do not contain $\mathbf{a}$. We first locate an arbitrary tetrahedron $\tau \in \mathcal{T}$ having $\mathbf{a}$ as a vertex (by the function `locate`), the function `finddirection` searches an edge matching $\mathbf{ab}$ in $\mathrm{St}(\mathbf{a})$. If $\mathbf{ab} \in \mathrm{St}(\mathbf{a})$, then $\mathbf{ab}$ is inserted in $\mathcal{T}$. Otherwise, $\mathbf{ab}$ is put in Q. The function `insertallsegments` returns Q. The main cost of this step is the time for point locations. By maintaining a map from vertices to tetrahedra to accelerate the point location, the insertion of a segment can be done in constant time. Hence the expected total time is $O(n)$.

Let $\mathbf{ab} \in Q$ be a missing segment. The reference point $\mathbf{p}$ for splitting $\mathbf{ab}$ is found by performing a line search from $\mathbf{a}$ towards $\mathbf{b}$ (see the function `scoutrefpoint`). Let Λ be the set of tetrahedra in $\mathcal{T}$ that intersect with $\mathbf{ab}$ (see Fig. A.8). Then $\mathbf{p}$ is chosen from $V \backslash \{\mathbf{a}, \mathbf{b}\}$, where V is the set of vertices of Λ. Since $\mathcal{T}$ is a Delaunay tetrahedralization, $\mathbf{p}$ must exist in V. Once $\mathbf{p}$ is found, the function `getsplitpoint` is called to find a point $\mathbf{v}$ in $\mathbf{ab}$ by one of the segment splitting rules. The worst-case time complexity for searching a reference point can be $O(n^2)$, such an example is found in [106]. For normal data sets, each segment only spans locally, the average complexity for one search is a constant time independent of n.

The next step is to insert $\mathbf{v}$ into $\mathcal{T}$ and $\mathcal{F}$. Our implementation maintains $\mathcal{T}$ and $\mathcal{F}$ separately. The face and edge flips are used to update $\mathcal{T} \cup \{\mathbf{v}\}$ into a Delaunay tetrahedralization, and the edge flips are used to update $\mathcal{F} \cup \{\mathbf{v}\}$ into a constrained Delaunay triangulation. After the insertions of $\mathbf{v}$, $\mathbf{ab}$ is replaced by two subsegments $\mathbf{av}$ and $\mathbf{vb}$ in $\mathcal{F}$. They are inserting in

$\mathcal{T}$ immediately. If any of them is not inserted, it is queued in Q. Moreover, the insertion of $\mathbf{v}$ may cause one or more segments or subsegments of $\mathcal{F}$ missing in $\mathcal{T}$, they are queued in Q too. We temporarily ignore the time for inserting $\mathbf{v}$, in $\mathcal{T}$ and $\mathcal{F}$. The rest of the time is proportional to the number of flips needed to update $\mathcal{T} \cup \{\mathbf{v}\}$. By the analysis of Guibas [56], the expected number of flips to add any vertex is a constant.

The whole process is repeated until Q is empty. On finish, $\mathcal{T}$ does not need to hold the segments of $\mathcal{F}$ anymore. The extra memory for holding segments in $\mathcal{T}$ are released. The complexity of the segment recovery algorithm has been roughly analyzed. In the worst-case, the algorithm may need $O(n_S^2)$ time, where n_S is the total number of Steiner points. While for normal data sets, the average time is $O(n_S)$.

The robustness of this implementation takes advantage of the separation of $\mathcal{T}$ and $\mathcal{F}$. The updating of $\mathcal{T} \cup \{\mathbf{v}\}$ is just an incremental Delaunay tetrahedralization. Hence the exact point-in-sphere test can be performed. Cares must be taken to handle the degeneracies. One such case is that a $\mathbf{v}$ may lies slightly outside $\mathcal{T}$ due to the imprecision of the floating-point arithmetics. Then the convex hull of $\mathcal{T}$ is enlarged by including $\mathbf{v}$.

A.3.4 Recovering the Facets

All subfaces of $\mathcal{F}$ are boundary faces which we want to recover. The premise is that $\mathcal{T}$ contains all segments of $\mathcal{F}$. The process of recovering facets can be imagined as "merging" the subfaces of $\mathcal{F}$ into $\mathcal{T}$, i.e., The subfaces of $\mathcal{F}$ which are not in $\mathcal{T}$ will be forced to appear in $\mathcal{T}$ by locally update the region where subfaces of $\mathcal{F}$ and faces of $\mathcal{T}$ are intersecting. The function `constrainedfacets` realizes the facet recovery algorithm. This process does not introduce Steiner points.

At initialization, it searches for each subface $\sigma \in \mathcal{F}$ a matched face in $\mathcal{T}$. It first finds a vertex of σ in $\mathcal{T}$ (by `locate`), then finds an edge of σ in $\mathcal{T}$ (by `finddirection`). If no edge is found, then σ is missing. Otherwise, σ is searched in the set of faces of $\mathcal{T}$ having this edge. If a matched face exists, σ is inserted at the location of the matched face, i.e., it is connected to the tetrahedra of $\mathcal{T}$ that contain it (If the face is a boundary face, it only connects to one tetrahedron). If σ does not match any face in $\mathcal{T}$, it is added into a queue Q, see Fig. A.9.

The remainder of the facet recovery process incrementally updates the Delaunay tetrahedralization $\mathcal{T}$ into a CDT which contains the surface mesh $\mathcal{F}$. The process loops until Q is empty. At each loop, one or more missing subfaces will be forced to be presented in $\mathcal{T}$. Here the idea is quite simple, i.e., all tetrahedra intersecting with these subfaces are removed and the empty region is filled by a set of new tetrahedra which respect the subfaces. While a robust and efficient implementation is not trivial. Two crucial procedures (described in Section 5.5.1): *formcavity*, and *TetrahedralizeCavity*,

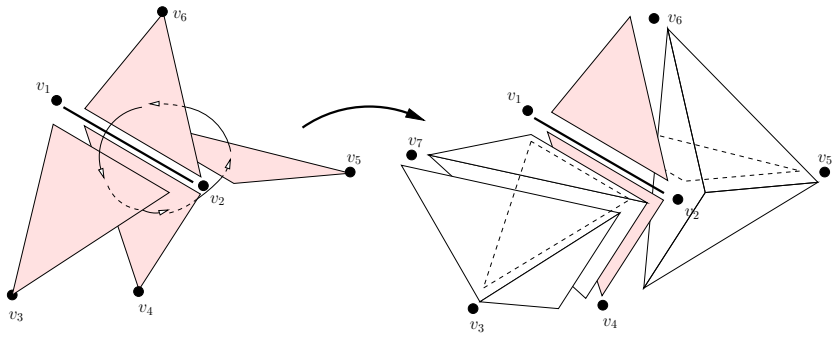


Figure A.9: Insert subfaces into the tetrahedralization $\mathcal{T}$. Left: four subfaces $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_i$, $i = \{3, 4, 5, 6\}$ sharing at a segment $\mathbf{v}_1\mathbf{v}_2$. Right: Subfaces $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_4$ and $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_6$ are inserted, while $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$ and $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_5$ are missing from $\mathcal{T}$.

need to be correctly implemented.

Let $\sigma \in Q$ be a missing subface, the missing region Ω containing σ is a collection of missing subfaces which are coplanar and connected. It can be formed by a depth-first searching through the connections of the subfaces. (See the function `formmissingregion`.)

Since Ω is missing in $\mathcal{T}$, it is expected that some tetrahedra in $\mathcal{T}$ are intersecting with it, i.e., a cavity exists. However, it is possible that Ω (or a portion of Ω) may not intersect with any tetrahedra of $\mathcal{T}$ due to the degenerate data set or numerical errors. Therefore the search for intersecting tetrahedra must be done carefully.

Even if a set of crossing tetrahedra Λ is found, not all subfaces of Ω may be crossed by the tetrahedra in Λ. One reason is that these subfaces are crossed by other tetrahedra which are disjoint with Λ. Before returning the two cavities $\mathcal{C}_1$ and $\mathcal{C}_2$, Ω must be validated. This is simply done by checking for each subface of Ω to see if it is crossed by at least one tetrahedra in Λ. The traingle-triangle test is used. The validation process has a quadratic time complexity. The hope is that both sizes of Ω and Λ are small. Subfaces in Ω which are not crossed by any tetrahedra in Λ are removed from Ω and put back into Q. They will be recovered later.

The cavity retetrahedralization procedure described in Section 5.5.1 uses two phases. The purpose of the first phase is to check all faces in F to see if they are strongly Delaunay within V. If a face $F \in \partial\mathcal{C}$ is not strongly Delaunay, $\mathcal{C}$ is expanded by removing F and adding new faces adjacent to F. However, the check for strongly Delaunay of a face within V without any neighbor information is not efficient. For this reason, our implementation does not need the first phase. The validation process is actually done inside the second phase. The cavity expansion is called if the validation fails. It is directly done by updating the current DT of vert($\mathcal{C}$).

Bibliography

[1] ACUSIM Software, Inc. AcuSolve, a general purpose finite element flow solver. `http://www.acusim.com`, 2007.

[2] D. Allen and R. Southwell. Relaxation methods applied to determine the motion, in two dimensions, of a viscous fluid past a fixed cylinder. *Quart. J. Mech. and Appl. Math.*, 8:129–145, 1955.

[3] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proc. 19th annual Symposium on Computational Geometry*, pages 211–219, 2003.

[4] I. Babuška, J. E. Flaherty, W. D. Henshaw, J. E. Hopcroft, J. E. Oliger, and T. Tezduyar, editors. *Modeling, mesh generation, and adaptive numerical methods for partial differential equations*, volume 75 of *The IMA Volumes in Mathematics and its Applications*. Springer, 1995.

[5] I. Babuška and W. C. Rheinboldt. Adaptive approaches and reliability estimates in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 17/18:519–540, 1987.

[6] T. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, 5:161–175, 1989.

[7] B. G. Baumgart. A polyhedron repersentation for computer vision. In *Proc. National Computer Conference*, 1975. AFIPS Conf. Proc., 589 596.

[8] M. Bern, P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proc. 6th Symp. Discrete Algorithms, ACM and SIAM*, pages 186–196, January 1995.

[9] M. Bern, D. Eppstein, and J. R. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[10] M. W. Bern, J. E. Flaherty, and M. Luskin, editors. *Grid generation and adaptive algorithms*, volume 113 of *The IMA Volumes in Mathematics and its Applications*. Springer, 1999.

[11] A. Bowyer. Computing Dirichlet tessellations. *Comp. Journal*, 24(2):162–166, 1987.

[12] G. E. Bredon. *Topology and Geometry*, volume 139 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993.

[13] P. R. Cavalcanti and U. T. Mello. Three-dimensional constrained Delaunay triangulation: a minimalist approach. In *Proc. 8th International Meshing Roundtable*, pages 119–129. Sandia National Laboratories, 1999.

[14] B. Chazelle. Convex partition of a polyhedra: a lower bound and worest-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984.

[15] B. Chazelle, H. Edelsbrunner, L. Guibas, J. Hershberger, and M. Seidel, R. amd Sharir. Selecting heavily covered points. *SIAM Journal on Computing*, 23:1138–1151, 1994.

[16] B. Chazelle and L. Palios. Triangulating a nonconvex polytope. *Discrete and Computational Geometry*, 5:505–526, 1990.

[17] L. Chen and J.-C. Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[18] S.-W. Cheng. On the sizes of Delaunay meshes. *Computational Geometry: Theory and Applications*, 33:130–138, 2006.

[19] S.-W. Cheng, T. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *J. Assoc. Comput. Mach.*, 47:883–904, 2000.

[20] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Quality meshing for polyhedra with small angles. *International Journal on Computational Geometry and Applications*, 15:421–461, 2005.

[21] S.-W. Cheng and S. H. Poon. Graded conforming Delaunay tetrahedralization with bounded radius-edge ratio. In *Proc. 14th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 295–304, 2003.

[22] P. L. Chew. Constrained Delaunay triangulation. *Algorithmica*, 4:97–108, 1989.

[23] P. L. Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Department of Computer Science, Cornell University, 1989.

[24] P. L. Chew. Guaranteed-quality meshing generation for curved surfaces. In *Proc. 9th annual ACM Symposium on Computational Geometry*, pages 274–280, 1993.

[25] P. L. Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proc. 13th annual ACM Symposium on Computational Geometry*, pages 391–393, Nice, France, June 1997.

[26] D. Cohen-Steiner, E. C. De Verdière, and M. Yvinec. Conforming Delaunay triangulation in 3D. In *Proc. 18th annual ACM Symposium on Computational Geometry*, 2002.

[27] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing*, 6:1063–1075, 1989.

[28] B. N. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[29] O Devillers and M. Teillaud. Perturbations and vertex removal in Delaunay and regular 3D triangulations. Technical Report 5968, INRIA, 2006.

[30] T. K. Dey, C. L. Bajaj, and K. Sugihara. On good triangulations in three dimensions. *International Journal of Computational Geometry & Applications*, 2(1):75–95, 1992.

[31] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[32] J. Dompierre, P. Labbe, F. Guibault, and R. Camarero. Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization. In *Proc. 7th International Meshing Roundtable*. Sandia National Laboratories, 1998.

[33] W. Dörfler. A convergent adaptive algorithm for Poisson's equation. *SIAM Journal on Numerical Analysis*, 33:1106–1124, 1996.

[34] H. Edelsbrunner. An acyclicity theorem for cell complex in d dimension. *Combinatorica*, 10(3):251–260, 1990.

[35] H. Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge University Press, England, 2001.

[36] H. Edelsbrunner and D. Guoy. Sink-insertion for mesh improvement. *Internat. J. Found. Comput. Sci.*, 13:223–242, 2002.

[37] H. Edelsbrunner, X.-Y. Li, G. L. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng, A. Üngör, and N. Walkington. Smoothing and cleaning up slivers. In *Proc. 32nd ACM Sympos. Theory Comput.*, pages 273–277, Portland, Oregon, United States, 2000.

[38] H. Edelsbrunner and M.P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithm. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[39] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[40] H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *SIAM Journal on Computing*, 22:527–551, 1993.

[41] J. Erickson. Nice point sets can have nasty Delaunay triangulations. *Discrete and Computational Geometry*, 30(1):109–132, 2003.

[42] R. Eymard, J. Fuhrmann, and K. Gärtner. A finite volume scheme for nonlinear parabolic equations derived from one-dimensional local Dirichlet problems. *Numerische Mathematik*, 102(3):463–495, 2006.

[43] R. Eymard, T. Gallouet, and R. Herbin. The finite volume method. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of Numerical Analysis*, volume VII, pages 715–1022. North-Holland, Amsterdam, 2000.

[44] P. Fleischmann. *Mesh generation for technology CAD in three dimensions*. PhD thesis, Fakultät für Elektrotechnik, Technischen Universität Wien, Wien, Dezember 1999.

[45] S. Fortune. Sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987.

[46] L. Freitag and C. F. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[47] P. J. Frey and P. L. George. *Mesh generation, application to finite elements*. Hermes Science, Oxford, United Kingdom, 2000.

[48] J. Fuhrmann and H. Langmach. Stability and existence of solutions of time-implicit finite volume schemes for viscous nonlinear conservation laws. *Applied Numerical Mathematics*, 37:201–230, 2001.

[49] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic analysis. *Systematic Zoology*, 18(3):259–278, 1969.

[50] H. Gajewski and K. Gärtner. On the discretization of van Roosbroeck's equations with magnetic field. *Z. Angew. Math. Mech.*, 76(5):247–264, 1996.

[51] K. Gärtner. Existence of bounded discrete steady state solutions of the van Roosbroeck system on boundary conforming Delaunay grids. Technical report, WIAS, Preprint 1258, 2007.

[52] P. L. George, H. Borouchaki, P. J. Frey, P. Laug, and E. Saltel. Mesh generation and mesh adaptivity: theory and techniques. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*. John Wiley & Sons, 2004.

[53] P. L. George, H. Borouchaki, and P. Laug. An efficient algorithm for 3D adaptive meshing. *Advances in Engineering Software*, 33:377–387, 2002.

[54] A. Glitzky and K. Gärtner. Energy estimates for continuous and discretized electro-reaction-diffusion systems. Technical report, WIAS, Preprint 1222, 2007.

[55] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[56] L. Guibas, D. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[57] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proc. 5th annual ACM Symposium on Computational Geometry*, pages 208–217, 1989.

[58] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4:75–123, 1985.

[59] C. Hazlewood. Approximating constrained tetrahedralizations. *Computer Aided Geometric Design*, 10:67–87, 1993.

[60] C. M. Hoffmann. *Geometric and Solid Modeling – An Introduction*. Morgan Kaufmann, San Mateo, CA, 1989. `http://www.cs.purdue.edu/homes/cmh/distribution/books/geo.html`.

[61] W. Huang. Measuring mesh qualities and application to variational mesh adaptation. *SIAM Journal on Scientific Computing*, 26(5):1643–1666, 2005.

[62] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, North-Holland, 1992.

[63] A. M. Il'in. A difference scheme for a differential equation with a small parameter multiplying the second derivative. *Mat. Zametki*, 6(2):237–248, 1969.

[64] INRIA. GAMMA, automatic mesh generation and adaption methods. `http://www-c.inria.fr/gamma`, 2007.

[65] B. Joe. Delaunay versus max-min solid angle triangulations for three-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 31:987–997, 1991.

[66] C. Johnson and P. Hansbo. Adaptive finite element methods in computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 101:143–181, 1992.

[67] J. Krause. *On boundary conforming anisotropic Delaunay meshes*. PhD thesis, Swiss Federal Institute of Technology Zurich, Zurich, 2001.

[68] C. L. Lawson. Software for c^1 surface interpolation. *Mathematical Software III, Academic Press, New York*, pages 161–194, 1977.

[69] D. T. Lee and A. K. Lin. Generalized Delaunay triangulations for planar graphs. *Discrete and Computational Geometry*, 1:201–217, 1986.

[70] X.-Y. Li and S.-H. Teng. Generating well-shaped Delaunay meshes in 3D. In *Proc. 12th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–37, Washington, DC, USA, 2001.

[71] A. Linke. *Divergence-free Mixed Finite Elements for the Incompressible Navier-Stokes Equation*. PhD thesis, Univ. Erlangen, 2008.

[72] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT Numerical Mathematics*, 34:268–287, 1994.

[73] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering*, 21:1403–1426, 1985.

[74] R. Löhner and P. Parikh. Three-dimensional grid generation by the advancing front method. *International Journal for Numerical Methods in Fluids*, 8:1135–1149, 1988.

[75] R. H. Mac Neal. An asymmetrical finite difference network. *Quart. Appl. Math.*, 11:295–310, 1953.

[76] M. Mäntylä. *An introduction to solid modeling.* Computer Science Press, New York, 1988.

[77] D. J. Mavriplis. Unstructured mesh generation and adaptivity. Technical Report 95-26, NASA Contractor Report 195069, ICASE, 1995.

[78] G. L. Miller. A time efficient Delaunay refinement algorithm. In *Proc. 15th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 400–409, 2004.

[79] G. L. Miller, D. Talmor, S.-H. Teng, and N. J. Walkington. A Delaunay based numerical method for three dimensions: Generation, formulation, and partition. In *Proc. 27th Annual ACM Symposium on Theory of Computing*, Las Vegas, Nevada, United States, 1995.

[80] G. L. Miller, D. Talmor, S.-H. Teng, N. J. Walkington, and H. Wang. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Proc. 5th International Meshing Roundtable.* Sandia National Laboratories, 1996.

[81] S. A. Mitchell. Cardinality bounds for triangulations with bounded minimum angle. In *Proc. 6th Canadian Conference on Computational Geometry*, pages 326–331, 1994.

[82] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in higher dimensions. *SIAM Journal on Computing*, 29:1334–1370, 2000.

[83] M. Mortenson. *Geometric Modeling.* John Wiley & Sons, College Park, Maryland, 1985.

[84] E. P. Mücke. *Shapes and Implementations in Three-Dimensions Geometry.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.

[85] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th annual Symposium on Computational Geometry*, pages 274–283, 1996.

[86] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236, 1978.

[87] M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralizations. In *Proc. 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 67–74, 2000.

[88] S. Oudot, L. Rineau, and M. Yvinec. Meshing volumes bounded by smooth surfaces. In *Proc. 14th International Meshing Roundtable*, pages 203–220. Sandia National Laboratories, 2005.

[89] L. Palios. *Decomposition Problems in Computational Geometry*. PhD thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1992.

[90] S. E. Pav. *Delaunay Refinement Algorithm*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2003.

[91] S. E. Pav and N. J. Walkington. Robust three dimensional Delaunay refinement. In *Proc. 13th International Meshing Roundtable*. Sandia National Laboratories, 2004.

[92] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[93] J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete and Computational Geometry*, 7:227–253, 1992.

[94] D. L. Scharfetter and H. K. Gummel. Large signal analysis of a silicon Read diode. *IEEE Trans. Electron. Dev.*, 16:64–77, 1969.

[95] E. Schönhardt. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98:309–312, 1928.

[96] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. In *1978-1988 Ten Years IIG*, pages 178–191, 1988.

[97] J. R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th annual Symposium on Computational Geometry*, pages 141–150, 1996.

[98] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, may 1996. `http://www.cs.cmu.edu/~quake/triangle.html`.

[99] J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997. Available as Technical Report CMU-CS-97-137.

[100] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th Annual Symposium on Computational Geometry*, pages 76–85, June 1998.

[101] J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proc. 14th Annual Symposium on Computational Geometry*, pages 86–95, Minneapolis, Minnesota, United States, June 1998.

[102] J. R. Shewchuk. Sweep algorithm for constructing higher-dimensional constrained Delaunay triangulations. In *Proc. 16th Annual Symposium on Computational Geometry*, pages 350–359, June 2000.

[103] J. R. Shewchuk. Constrained Delaunay tetrahedralization and provably good boundary recovery. In *Proc. 11th International Meshing Roundtable*, pages 193–204. Sandia National Laboratories, September 2002.

[104] J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proc. 11th International Meshing Roundtable*, pages 115–126, Ithaca, New York, September 2002. Sandia National Laboratories.

[105] J. R. Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. 19th Annual Symposium on Computational Geometry*, pages 181–190, June 2003.

[106] J. R. Shewchuk. Stabbing delaunay tetrahedralizations. *Discrete and Computational Geometry*, 32:339–343, 2004.

[107] J. R. Shewchuk. Pyramid. Unpublished, 2006.

[108] J. R. Shewchuk. General-dimensional constrained Delaunay and constrained regular triangulations I: Combinatorial properties. To appear in Discrete and Computational Geometry, 2007.

[109] H. Si. TetGen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3, user's manual. Technical Report 9, Weierstrass Institute for Applied Analysis and Stochastics, 2004.

[110] H. Si. TetGen. `http://tetgen.berlios.de`, 2007.

[111] H. Si. Adaptive tetrahedral mesh generation by constrained delaunay refinement. *International Journal for Numerical Methods in Engineering*, 75(7):856–880, 2008.

[112] H. Si and K. Gärtner. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proc. 14th International*

Meshing Roundtable, pages 147–163, San Diego, CA, USA, 2005. Sandia National Laboratories.

[113] D. Talmor. *Well-spaced points for numerical methods*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997. Available as Technical Report CMU-CS-97-164.

[114] The Boeing Company. GGNS, the General Geometry Navier-Stokes solver. not published, 2007.

[115] R. S. Varga. *Matrix Iterative Analysis*. Printice-Hall Inc., Englewood Cliffs, New Jersey, 1962.

[116] R. Verfürth. *A Review of Posteriori error estimation and adaptive mesh refinement techniques*. Wiley-Teubner, 1996.

[117] VIS, Inc. OpenDX. `http://www.opendx.org`, 2007.

[118] G. Voronoi. Nouvelles applications des parametrès continus à la théorie de formas quadratiques. *Reine Angew. Math.*, 133:97–178, 1907.

[119] N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.

[120] W. Weiss, T. Streckenbach, and D. Hömberg. WIAS-SHarP, a surface hardening program. `http://www.wias-berlin.de/software/sharp`, 2007.

[121] M Yerry and M. S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, 20:1965–1990, 1984.

[122] G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995.

Notations

In this thesis, the following general rules are applied for notations. There may have exceptions.

- Topological spaces are denoted by uppercase blackboard letters, *eg.* , $\mathbb{H}$, $\mathbb{R}$, $\mathbb{X}$, and $\mathbb{Y}$. In particular, $\mathbb{R}^d$ denotes the d-dimensional Euclidean space. (This font requires the `amssymb` package.)

- Elements of $\mathbb{R}^d$ (points, vectors) are denoted by boldface lowercase letters, *eg.* , $\mathbf{p}$, $\mathbf{v}$, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, etc.

- Elements of $\mathbb{R}$ (coordinates, scalars) are denoted by lowercase letters, *eg.* , x_k, y_k, z_k, etc. Exceptionally, the greek letters α, β, and λ are used for coefficients in formulae.

- Geometric objects (subsets of $\mathbb{R}^d$) are denoted by uppercase letters, *eg.* , F, P, Q, S, U, V, etc. Our preference is to use P, Q for (convex or non-convex) polytopes, F for faces of polytopes, and S and V for point sets. Exceptionally, geometrical simplices are denoted by the greek letters σ, τ, and ν.

- Families of subsets of $\mathbb{R}^d$ (geometrical objects) are denoted by uppercase calligraphic letters, *eg.* , $\mathcal{K}$, $\mathcal{T}$, $\mathcal{X}$, and $\mathcal{Y}$. This includes simplicial complexes, piecewise linear systems. Our preference is to use $\mathcal{K}, \mathcal{T}$ for simplicial complex, $\mathcal{X}, \mathcal{Y}$ for piecewise linear systems.

- Standard asymptotic notations (a shorthand used to give a quick measure of the behavior of a function $f(n)$ as n grows large): $O(f(n))$, the class of functions that are at most as large as $f(n)$, and $\Omega(f(n))$, the class of functions that are at least as large as $f(n)$, are used to express the algorithm performs, such as the running times, output sizes.

List of Symbols

$\leq$ ..... $F \leq P$, F is a face of P
$<$ ..... F is a proper face of P
$\|\cdot\|$ ..... the Euclidean distance function
$|\cdot|$ ..... the underlying space of a simplicial complex or a PLS
$\mathbf{A}$ ..... a matrix in $\mathbb{R}^{m\times d}$
$\text{aff}(V)$ ..... the affine hull of V
α_1 ..... parameter for scaling the size of sparse ball
α_2 ..... parameter for scaling the size of protecting ball
$\mathbb{B}^d(\mathbf{x}, r)$ ..... the open d-ball at a point $\mathbf{x} \in \mathbb{R}^d$ with radius r
B^d ..... the d-dimensional unit closed ball
$\text{bd}(P)$ ..... the bounday of P
$\mathcal{C}$ ..... a polytopal complex
C_1, C_2, C_3 ..... constants bounding output edge lengths
$\text{conv}(V)$ ..... the convex hull of V
$\mathcal{D}$ ..... a Delaunay triangulation
$D(\mathbf{v})$ ..... lower bound function on output edge length at $\mathbf{v}$
$\dim(P)$, $\dim(\mathcal{X})$ ..... the dimension of P, $\mathcal{X}$
Δ ..... the spread of a point set
$\emptyset$ ..... the empty set
$\mathcal{F}$ ..... a surface mesh of a PLS
$f : \mathbb{X} \to \mathbb{Y}$ ..... a (continuous) map from $\mathbb{X}$ to $\mathbb{Y}$
f^{-1} ..... the inverse of f
$\mathcal{G}$ ..... a planar straight line graph
$H : |\mathcal{X}| \to \mathbb{R}$ ..... a mesh sizing function defined on $|\mathcal{X}|$
$\text{int}(P)$ ..... the interior of P
$\mathbf{I}_n$ ..... the identity matrix of size n
K_i, K_j ..... control volumes
$\mathcal{L}_{v,f}$ ..... a special PLS with f facets and v vertices, see Fig. 5.15
$\text{lfs}(\mathbf{x})$ ..... the local feature size of $\mathbf{x}$
$\text{lfs}_{\min}$ ..... the minimum local feature size on points in $|\mathcal{X}|$
$\mathcal{O}(\subseteq 2^{\mathbb{X}})$ ..... $\mathcal{O}$ is a topology on $\mathbb{X}$
$O(f(n))$ ..... asymptotic notation for upper bound
Ω ..... a physical domain
$\partial\Omega$ ..... the boundary of Ω
$\Omega(f(n))$ ..... asymptotic notation for lower bound
$\mathcal{P}$ ..... a collection of polytopes
$p(\mathbf{v})$ ..... the parent of $\mathbf{v}$
$R1$, $R2$, $R3$ ..... point generating rules
$R(\mathbf{v})$ ..... the original acute vertex of a segment $\mathbf{ve}_j$
$r(\mathbf{v})$ ..... the insertion radius of $\mathbf{v}$
ρ_0 ..... the radius edge ratio bound
$\mathbb{S}^k$ ..... the k-dimensional unit sphere

Σ (of σ) ..... a circumscribed sphere of σ
$\Sigma(\mathbf{c}, r)$ ..... a sphere centered at $\mathbf{c}$ with radius r
$\mathcal{T}$ (of $\mathbb{X}$) ..... a triangulation of space
$\mathcal{T}$ (of conv($|\mathcal{X}|$)) ..... a triangulation of convex hull of a PLS $\mathcal{X}$
$\mathcal{T}$ (of $\mathcal{X}$) ..... a mesh of a PLS $\mathcal{X}$
θ (of $\mathcal{X}$) ..... an input angle of $\mathcal{X}$
θ_m (of $\mathcal{X}$) ..... the smallest input angle of $\mathcal{X}$
V_p ..... Voronoi cell of point $\mathbf{p}$
vert($\mathcal{X}$) ..... the vertex set of $\mathcal{X}$
$\mathbf{v}^k$ ..... the grandparent of $\mathbf{v}^{k-1}$, $\mathbf{v}^0 = \mathbf{v}$
$\mathcal{X}^{(i)}$ ..... the i-skeleton of $\mathcal{X}$
$\partial\mathcal{X}$ ..... the boundary system of $\mathcal{X}$ $(= \mathcal{X}^{(\dim(\mathcal{X})-1)})$
$\mathbf{x} \sim \mathbf{y}$ ..... an equivalence relation, $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ have the same carrier
$\mathcal{X}/\sim$ ..... a partition of $|\mathcal{X}|$ by the equivalence relation $\sim$
$\coprod_{U \in \mathcal{X}/\sim} U$ ..... the disjoint union of sets in $\mathcal{X}/\sim$

Index

acute vertex, 89
adaptive numerical methods, 6
affine hull, 14
affine subspace, 14
affinely independent, 14
ancestor, 36
anisotropic, 68
aspect ratio, 5, 26

B-Rep, *see* boundary representation
ball, open, 13
    unit, 14
boundary
    of a polytope, 16
    of a physical domain, 3, 18
    of a polyhedron, 17
    of a simplex, 15
boundary conforming Delaunay
    mesh, 4, 26
    property, 25
boundary conformity problem, 8, 10, 83
boundary representation, 113
boundary system, 19
break point, 104

carrier, 19, 85
cavity, 97
cavity tetrahedralization algorithm, 97
CDT, *see* constrained Delaunay triangulation
CDT algorithm, 88
cell complex, 16
Chazelle's polyhedron, 8
circumball, 24
conforming Delaunay
    mesh, 4, 25
connected, 14
constrained Delaunay
    criterion, 9, 85
    locally Delaunay, 86
    tetrahedralization, 9
    triangulation
        of a PLS, 85
        of a polyhedron, 9
        of a PSLG, 8
constructive solid geometry, 113
continuous map, 14
control volume, 2
convection-diffusion problem, 2
    concentration, 2
    continuous equation, 2
    diffusion coefficient, 2, 4
    discrete equation, 3
    integration equation, 3, 4
convex, 14
convex decomposition, 20
convex hull, 14
convex polyhedron, 16
convex polytope, 15
coordinate, 13
CSG, *see* constructive solid geometry

Delaunay
    criterion, 4, 24
    locally Delaunay, 85
    strongly Delaunay, 10, 88
    triangulation, 24
Delaunay Lemma, 86
Delaunay meshing problem, 4
Delaunay refinement, 5, 26

a modified algorithm, 56
adaptive algorithm, 71
Shewchuk's algorithm, 6, 29
sliver removal, 63
diametric ball, 25
diametric sphere, 4
dihedral angle, 27
dimension
of a PLS, 18
of a polyhedron, 17
of a polytope, 16
of a simplex, 14
of a simplicial complex, 15
of an affine hull, 14
disjoint union, 15
domain
internal boundary, 4
mesh domain, 18
physical domain, 3, 18
sub-domain, 4
double rejection, 47

edge ring, 119
edge, relatively short, 56
edge-facet data structure, 118
element shape, 5
empty cone lemma, 51
empty sphere property, *see* Delaunay criterion
encroached, 28, 90
equivalence relation
on the same carrier, 19
Euclidean distance function, 13
Euclidean space, 13

face
of a convex polytope, 16
of a polyhedron, 17
of a simplex, 14
face figure, 17
face ring, 119
facet, 17, 27
facet recovery algorithm, 97
FEM, *see* finite element methods
finite element methods, 1
adaptive, 7
finite volume methods, 1, 2
Voronoi-box based, 2, 4
FVM, *see* finite volume methods

Gabriel property, 4, 25
general position, 24, 86
grandparent, 35

homeomorphic, 14
homeomorphism, 14
hyperbolic paraboloids, 8

input angle, 30
facet-facet angle, 31
segment-facet angle, 30
segment-segment angle, 30, 60
input angle assumption, 34
relaxed, 60
input segment assumption, 47
insertion order, 54
insertion radius, 30
interior
of a polytope, 16
of a PLS, 19
of a polyhedron, 17
of a simplex, 15
internal boundary, 18
floated, 18
invisible, 9, 84
isotropic, 68

local degeneracy, 104
local degeneracy removal algorithm, 104
local feature size, 31, 68

M-matrix, 3
manifold, 14
maximum principle, 2
local, 3
mesh (of PLS), 22
mesh adaptation, 7
mesh adaption problem, 7

mesh conformity criterion, 68
mesh generation, 1
mesh quality, 5
mesh size
    bounded, 5
mesh sizing function, 7, 68
minimum dihedral angle, 5
missing region, 96
missing segments, 89

neighborhood, 14

object function, 5
open set, 13
optimal mesh, 5
orientation test, 121

parent, 30, 72
partial differential equation, 1
partition, 4
    Voronoi partition, 4
PDE, *see* partial differential equation
periodic point set, 56
piecewise linear system, 18
planar straight line graph, 8
PLS, *see* piecewise linear system
PLSG, *see* planar straight line graph
point, 13
point accepting rule, 70
point generating rules, 28, 61, 69
point-in-sphere test, 121
poly file format, 116
polyhedral complex, 16
polyhedron, 17
polytopal complex, 16
polytope, *see* polyhedron
proper face, 14, 16, 17
protecting ball, 69
pure CDT, 85
Pyramid, 9

radius edge ratio, 26
reference point, 90
refinement (of a PLS), 89
reflex edges, 8
relative interior, *see* interior
ridge, 17

Schönhardt polyhedron, 8, 20
segment, 27
segment recovery algorithm, 92
segment splitting rules, 90
separation, 14
sequence of parents, 35
sharp segment, 61, 74
Shewchuk's CDT Theorem, 88
simplex, 4, 14
simplicial complex, 15
skeleton, 15, 18
sliver, 27, 63
SoP, *see* sequence of parents
sparse ball, 69
sphere, 14
spread, 56
star, 124
Steiner CDT, 85
Steiner points, 7, 20
subcomplex, 15
subdivision, *see* partition
subface, 28
subsegment, 27
subspace, 14
subsystem, 18

TetGen, 9, 78, 105, 113
tetrahedron shape measure, 26
tetrahedron-based data structure, 119
topological space, 13
topologically equivalent, *see* homeomorphic
topology, 13
    disjoint union topology, 15, 19
    induced topology, 14
    natural topology, 13
    quotient topology, 15, 19
Triangle, 122
triangle-edge data structure, 118
triangle-edge pair, 119
triangulation

of PLS, 21
of space, 21
type-0 segment, 89
type-1 segment, 89

underlying space, 15, 19

vertex, 13
vertex degree, 52
vertex set, 17, 19
of a simplicial complex, 15
vertex, source, 47
visibility, 84
visible, 84
Voronoi cell, 24
Voronoi diagram, 24
Voronoi face, 24

well-graded, 35

Curriculum Vitae

Hang Si

born on January 13, 1971 in Hangzhou,
Zhejiang Province, China

1978 – 1983 :	Elementary school in Hangzhou
1983 – 1989 :	High school in Hangzhou
1989 – 1990 :	High school in Xiamen
1990 – 1994 :	Diploma study in Hangzhou University*
	in the Department of Electrical Engineering,
July 1994 :	graduated with Bachelor's degree
1994 – 1997 :	Electrical Engineer in Hangzhou
1997 – 1999 :	Software Engineer in Hangzhou
1998 – 2002 :	Graduate study in Zhejiang University
	in the Department of Computer Science,
March 2002 :	graduated with Master's degree
2002– :	Scientific staff member in Weierstrass Institute
	for Applied Analysis and Stochastics (WIAS) Berlin,

∗ Zhejiang Universität, Hangzhou Universität, Zhejiang Agricultral Universität und Zhejiang Medical Universität wurden als Zhejiang Universität am 15. September 1998 vereinigt.