Delaunay Refinement Mesh Generation of Curve-bounded Domains

by

Serge Gosselin

B.Eng., Université Laval, 2002
M.A.Sc., The University of British Columbia, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate Studies

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

October, 2009

Abstract

Delaunay refinement is a mesh generation paradigm noted for offering theoretical guarantees regarding the quality of its output. As such, the meshes it produces are a good choice for numerical methods. This thesis studies the practical application of Delaunay refinement mesh generation to geometric domains whose boundaries are curved, in both two and three dimensions. It is subdivided into three manuscripts, each of them addressing a specific problem or a previous limitation of the method. The first manuscript is concerned with the problem in two dimensions. It proposes a technique to sample the boundary with the objective of improving its recoverability. The treatment of small input angles is also improved. The quality guarantees offered by previous algorithms are shown to apply in the presence of curves. The second manuscript presents an algorithm to construct constrained Delaunay tetrahedralizations of domains bounded by piecewise smooth surfaces. The boundary conforming meshes thus obtained are typically coarser than those output by other algorithms. These meshes are a perfect starting point for the experimental study presented in the final manuscript. Therein, Delaunay refinement is shown to eliminate slivers when run using non-standard quality measures, albeit without termination guarantees. The combined results of the last two manuscripts are a major stepping stone towards combining Delaunay refinement mesh generation with CAD modelers. Some algorithms presented in this thesis have already found application in high-order finite-volume methods. These algorithms have the potential to dramatically reduce the computational time needed for numerical simulations.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost I would like to thank all of the faculty who have guided me during these years as a graduate student. Special thanks go to my supervisor, Dr Carl Ollivier-Gooch whose valuable insights and constant availability for discussions and help made this work possible. I have learned plenty under his supervision.

I would also like to thank my labmates and student colleagues who shared good moments with me over the years. I must especially thank my good friend Chris who was alongside in the lab for this entire journey. Thanks for your constant enthusiasm, for sharing your brilliant ideas and for introducing me to the backcountry. This adventure would not have been the same without you.

To all the friends I made in Vancouver and with whom I have spent time on the slopes, on the rink or around a good bottle of wine, I thank you. You have kept me grounded and made my life outside of work enjoyable. Merci aussi à mes amis du Québec, en particulier à Tommy pour ses précieux conseils.

Merci à mes parents d'avoir cru en moi et de m'avoir toujours encouragé à suivre ma voie. Merci surtout pour votre amour. Votre support dans les bons moments, tout comme dans les moins bons, a toujours été inconditionnel. Je ne pourrai jamais suffisamment vous en remercier. Merci à mon oncle Serge et à sa partenaire Luce pour l'aide qu'ils m'ont fourni. Sans eux, mes études auraient été plus difficiles.

Finalement, merci à Marie-Josée avec qui j'ai le plaisir de partager ma vie depuis plusieurs années. Cette thèse, je te la dédie. Sans ton amour et tes encouragements, je n'y serais jamais parvenu.

Co-authorship Statement

The research ideas and methods explored in the three co-authored manuscripts of this thesis are the fruits of a close working relationship between Dr Carl Ollivier-Gooch and Serge Gosselin. The implementation of the methods, the data analysis, and the manuscript preparation were done by Gosselin with invaluable guidance from Ollivier-Gooch throughout the process.

Chapter 1

Introduction

Many physical systems relevant in scientific and engineering applications can be modeled with partial differential equations (PDE). Unfortunately, obtaining an analytical solution to these equations is often impossible, especially for a system defined over a complex domain. One classic example is that of the Navier-Stokes equations which describe the motion of viscous fluids. Although exact solutions are available for simplified flows, such as Couette or Poiseuille flows [23], the existence of a solution to these equations in their complete form remains to be proven.

As an alternative, numerical simulations can be used to compute approximate solutions to PDEs. Numerical methods are now commonly used in the design and validation of systems involving such phenomena as heat transfer, fluid flows or electromagnetics. Prior to conducting the simulation itself, the continuous region under study must be decomposed into simpler entities, thereby forming a *mesh*.

This thesis focuses on the generation of triangular and tetrahedral meshes destined for numerical simulations, more specifically on the application of mesh generation techniques based on the Delaunay refinement paradigm to computational domains bounded by curves and curved surfaces. The thesis is composed of three manuscripts, each presented in a separate chapter. Chapter 2 is devoted to two-dimensional Delaunay refinement. Chapter 3 describes an algorithm to construct tetrahedralizations constrained to geometries bounded by curved surfaces and containing sharp features. The tetrahedralizations thus obtained are then used in the experimental study presented in Chapter 4, where Delaunay refinement and non-standard quality measures are used in conjunction to generate meshes with good quality properties.

This introductory chapter presents background information relevant to the definition of the problems solved in this thesis. The relationship between meshes and numerical simulations is first described, emphasizing on the properties a mesh destined for numerical methods should exhibit. The components of a canonical mesh generation method satisfying these properties are then defined. This canonical method is the basis for the algorithms developed in the thesis. The chapter is concluded by a presentation of the objectives pursued by this research and by results showing how these objectives have been attained.

1.1 Meshes and Numerical Methods

There are essentially three components involved in the numerical simulation of physical phenomena: modeling, discretization and solution. During the modeling part, the equations governing the problem are defined. These equations are then discretized over the domain of interest, transforming the continuous problem into an approximating system of algebraic equations. Finally, this system of equations is solved to obtain a discrete solution.

The finite-volume and the finite-element methods are two of the most commonly used approaches for spatial discretization. Application of these methods require the computational domain to first be partitioned into simpler, boundary conforming entities; these are commonly referred to as *control volumes* in the finite-volume method or as *elements* in the finite-element method. The union of these entities form a *mesh*.

1.1.1 Structured and Unstructured Meshes

Meshes used in numerical methods can typically be categorized as structured or unstructured, according to their connectivity. A structured mesh, also commonly termed a grid, has a fixed and predictable topology where each vertex can be identified by indices. For example a vertex with indices (i, j) has vertex $(i-1, j)$ as left neighbor and vertex $(i+1, j)$ as right neighbor. Unstructured meshes on the other hand lack this topological structure and

as a result, do not have a predictable connectivity. The mesh adjacency data must therefore be stored to know the neighbors of a given vertex.

Although the previous categorization is not influenced by the geometric nature of the mesh entities, structured meshes are generally composed of quadrilaterals in two dimensions and hexahedra in three dimensions. It is nevertheless conceivable for a mesh to have a structured connectivity and yet be composed of entities of a different shape. As for unstructured meshes, they are most commonly made of triangles or quadrilaterals in two dimensions and of tetrahedra and hexahedra in three dimensions. Prisms and pyramids can also be encountered. An unstructured mesh may also contain an amalgam of different types of entities in which case it is referred to as a *mixed* mesh.

From a numerical solver's perspective, structured meshes are attractive. Faster solution schemes with reduced memory usage can be developed by capitalizing on their predictable topology. In the finite-volume method, a fixed template is sufficient to compute the fluxes through a structured control volume's boundaries. To evaluate these same fluxes on an unstructured mesh, a polynomial reconstruction of the solution over each control volume must first be performed. While the generation of both structured and unstructured meshes poses its own challenges, obtaining a structured mesh of a complex domain often involves time-consuming human interventions that can easily offset the savings realized by the solver itself [28]. Unstructured meshes lend themselves better to automation, even when arbitrarily complex geometries are involved. Furthermore, their local density can more easily be tailored to fit the needs of a given problem, something that is very difficult to achieve with structured grids. A solution computed on a coarse but judiciously sized unstructured mesh can conceivably be as accurate as a solution obtained on a much finer structured mesh, although multiblock structured grids reduce some of this advantage.

This thesis is only concerned with the generation of triangular and tetrahedral unstructured meshes from geometric domains having piecewise smooth boundaries.

1.1.2 Properties of Meshes for Numerical Methods

A simplicial mesh destined for numerical simulations must meet specific requirements not necessarily encountered in other applications. A detailed presentation of these desirable properties is made by Shewchuk [40]; this section summarizes them.

The primary objective when generating a mesh is to construct a good geometric approximation of the computational domain. Most numerical methods require a proper discrete representation of the domain's boundaries to define the problem's boundary conditions. The mesh entities must therefore conform to these boundaries, both external and internal. External boundaries separate the meshed domain from the unmeshed space surrounding it. Internal boundaries appear inside the meshed domain and are useful for separating regions with different physical properties. They can also be used to track the interface between two fluids. There does exist consistent numerical methods that do not require conforming boundaries, implicitly capturing the correct boundary conditions instead. The immersed interface method is one example. These however suffer in accuracy relative to conforming methods.

The problem of boundary conformity is a mild issue in two dimensions: a triangulation conforming to a set of non-intersecting segments can always be obtained through edge flips [26]. In three dimensions however, some simple polyhedra cannot be tetrahedralized without the addition of Steiner vertices [5, 41], the classic example being the Schönhardt polyhedron. The construction of boundary conforming tetrahedralizations is therefore more difficult.

Numerical solutions to isotropic problems are known to be adversely affected by meshes exhibiting poor quality properties. It was demonstrated that for the finite-element method to compute an accurate solution, the mesh must not contain angles approaching 180° [3, 25]. Small angles on the other hand lead to ill-conditioned stiffness matrices [21]. Iterative solvers used on the resulting system of algebraic equations will therefore converge slowly, if at all. Furthermore, regardless of the type of solver used, a high condition

number will likely lead to an inaccurate floating-point solution of the system of equations. These same observations apply to dihedral angles in three dimensions, but not to solid angles which affect neither the accuracy nor the efficiency of a simulation [43]. There is experimental evidence showing that finite-volume discretizations also suffer from poor quality meshes [19]. As only a handful of poorly-shaped entities are sufficient to affect a simulation, generating meshes in which the worst entities are as good as possible is an important consideration.

Finally, the density of mesh points should be easily controllable and allowed to vary quickly over a short distance. This property, known as *grading*, is useful to make the mesh finer in certain regions where solution details must be captured, while keeping it coarse everywhere else. Since the computational effort required to obtain a numerical solution directly depends on the number of elements or control volumes in the mesh, grading prevents time from being wasted in regions that would otherwise be over-resolved. Smooth grading is also important when solving hyperbolic PDEs. It mitigates the numerical reflection or refraction of waves with wavelengths comparable to the mesh spacing in regions they may travel through.

In summary, a mesh generator should aim at achieving the best possible quality properties using as few mesh points as possible. It is always easier to refine a mesh to match a prescribed sizing function than it is to coarsen it. In other words, the mesh entities should remain as large as the input domain allows, while still being nicely shaped.

1.1.3 Meshes for High-order Accurate Simulations

For consistent numerical methods, the discretization of partial differential equations should in general produce an error of $\mathcal{O}(h^p)$ where h is the mesh length scale and p is the order of accuracy. When the mesh density is doubled, one can expect the error for second- and fourth-order accurate discretizations to be reduced by factors of four and sixteen respectively. A high-order accurate method can therefore use a coarser mesh to achieve a given level of accuracy. As such, high-order methods have the potential to

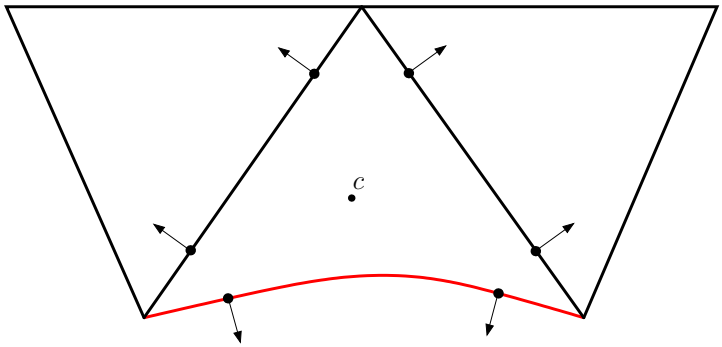


Figure 1.1: A high-order boundary control volume. The curvature must be accounted for when computing the location of the Gauss points and the direction of the normals for boundary flux integration. The location of the control volume's centroid c must also be adjusted.

reduce the computational time needed for numerical simulations [29].

High-order accurate finite volume simulations place additional requirements on mesh generation. As is illustrated in Figure 1.1, flux integration for third- and fourth-order schemes require two Gauss points per control volume face. When integrating fluxes along curved boundaries, the Gauss points and the boundary normals must accurately reflect the shape of the boundary, otherwise the schemes will fail to achieve the discretization's nominal order of accuracy. One straightforward way to obtain the information required for flux integration is to query the boundary representation directly, which must therefore be available along with the mesh.

For the flux integration data to remain consistent, the discrete approximation of the boundary must approach the true underlying boundary shape with the same order of accuracy as the discretization scheme [32, 33]. However, the curved boundary and its piecewise linear representation are separated by a distance that is $\mathcal{O}\left(h^2\right)$ for edge length h. In other words, it is necessary to place new vertices directly on boundary curves when refining a mesh, not on the mesh boundary edges. A mesh generator designed for high-order finite volume simulations must therefore have access to the exact curved boundary representation.

1.2 Overview of the Method

The Delaunay triangulation and its use in mesh generation have been studied extensively over the years. Mesh generation methods based on this structure have indeed been in use for at least the past twenty five years [47].

This section defines a canonical mesh generation method based on the Delaunay refinement paradigm. It is the foundation upon which the algorithms presented in subsequent chapters are developed. Basic concepts concerning Delaunay triangulations and their construction are first presented. More specific details about boundary conforming Delaunay triangulations, interior point placement and mesh post-processing follow. These pieces are finally assembled to form the quality mesh generation method used in this thesis.

1.2.1 The Delaunay Triangulation

The Delaunay triangulation is a geometric structure originally proposed in 1934 by Boris Delaunay. Many mesh generation algorithms have since then been developed around it. This choice is easily justifiable by the fact that Delaunay triangulations maximize the minimum angle among all possible triangulations of a given point set [26]. The concept extends straightforwardly to three dimensions, but unfortunately, the minimum angle maximization property does not. A class of undesirable tetrahedra, commonly known as *slivers*, can indeed appear in Delaunay tetrahedralizations. A canonical sliver tetrahedron is formed by four nearly coplanar vertices, equally spaced around its circumsphere's equator. With dihedral angles arbitrarily close to 0° and 180°, they are obviously detrimental to numerical simulations.

Delaunay triangulations are characterized by the *empty ball* property: Given a set of points P, the Delaunay triangulation of P, noted $\mathrm{DT}(P)$, is a triangulation such that no point in P is located strictly inside the circumcircle of any triangle in $\mathrm{DT}(P)$ (see Figure 1.2b). Points are however allowed to lie directly on the circumcircles. The empty circle property also apply to the edges of the Delaunay triangulation: an edge will appear in $\mathrm{DT}(P)$ if there exists a circle circumscribing this edge that does not contain any points in P. If the points are in general position, that is no four points are located on

a common circle, then the Delaunay triangulation is unique.

The Voronoi Diagram

The Voronoi diagram is the dual graph of the Delaunay triangulation. Every point in set P is a site in the Voronoi diagram. Every site is associated with a Voronoi cell consisting of all points closer to its own site than to any other site. As is illustrated in Figure 1.3, these cells are obtained by connecting the circumcenters of the triangles in $\mathrm{DT}(P)$.

Voronoi diagrams play a subtle, but important role in the definition of meshing algorithms based on the Delaunay refinement paradigm. These algorithms incrementally insert vertices in a Delaunay triangulation with the objective of improving its quality. This is achieved by inserting new vertices as far as possible from existing vertices. This implies that insertions coincide with Voronoi vertices.

1.2.2 Delaunay Triangulation Construction

There exist at least three types of algorithms to compute the Delaunay triangulation of a vertex set in two dimensions. The most simple is undoubtedly the incremental insertion method. A typical implementation starts with the Delaunay triangulation of a rectangular bounding box enclosing the space where the vertices are located. The vertices are then inserted one by one in the triangulation, while maintaining the empty circle property.

The earliest incremental algorithm is due to Lawson [26]. As is shown in Figure 1.4, it relies on an insert-and-flip strategy. When a new vertex is inserted, it generally splits an existing triangle into three new ones (it is possible to insert directly on an edge, in which case the edge is split into two and its two adjacent triangles are split into four). To maintain the Delaunay property, every edge that is no longer locally Delaunay must be flipped. The edge in question is the diagonal of a quadrilateral formed by the two triangles adjacent to it. During the flipping procedure, the diagonal of this quadrilateral is swapped to connect the vertices that previously opposed the edge. The swap can only occur if the quadrilateral is convex. This action is

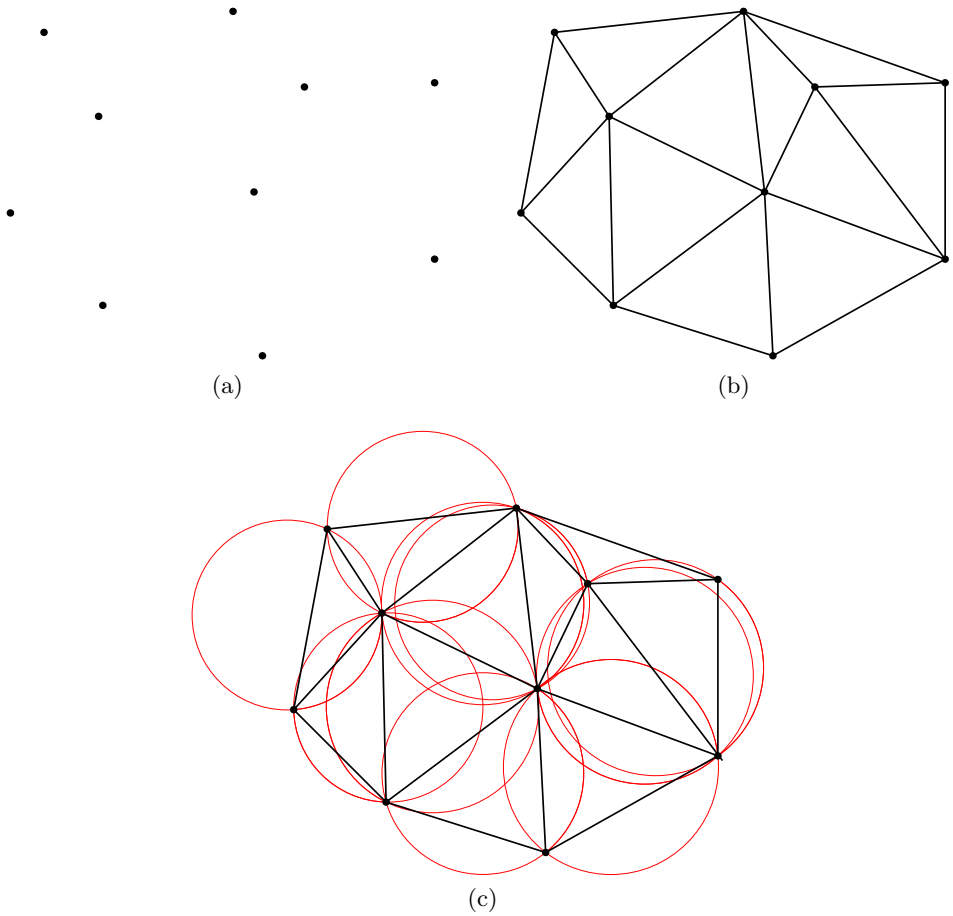
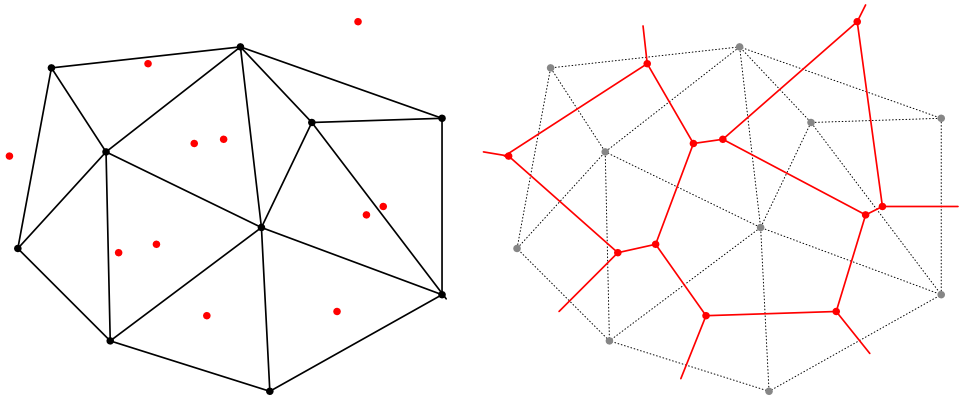


Figure 1.2: The Delaunay triangulation of a set of random points. (a) A set of points in general position. (b) The Delaunay triangulation of this set of points. (c) The triangulation respects the empty circumcircle property.



(a) (b)

Figure 1.3: The Voronoi diagram of a set of random points. (a) The circumcenters of Delaunay triangles are vertices of the Voronoi diagram. (b) The Voronoi cells, drawn in red, are superimposed on the Delaunay triangulation. Every point in a Voronoi cell is closer to its site than to any other site.

repeated until a Delaunay triangulation is obtained.

An alternative incremental algorithm is illustrated in Figure 1.5. It was concurrently proposed by Bowyer [9] and Watson [46]. When a new vertex is added to the triangulation, every triangle whose circumcircle contain the vertex are deleted, forming an empty *cavity*. All other triangles are unaltered. The edges located on the perimeter of the cavity are then reconnected to the inserted vertex and a Delaunay triangulation is obtained.

Both incremental algorithms can be adapted to build Delaunay tetrahedralizations. The extension of Bowyer-Watson algorithm is straightforward while that of the flip algorithm is trickier. Depending on the configuration of the tetrahedra adjoining a face, multiple flip operations are possible (see Figure 1.9), which can make the procedure more complicated.

The Bowyer-Watson algorithm is more sensitive to floating point round-off errors, especially in three dimensions. The insphere test to identify which tetrahedra must be deleted can sometimes return erroneous answers. As a result, the cavity could fail to be star-shaped (a vertex on the cavity's

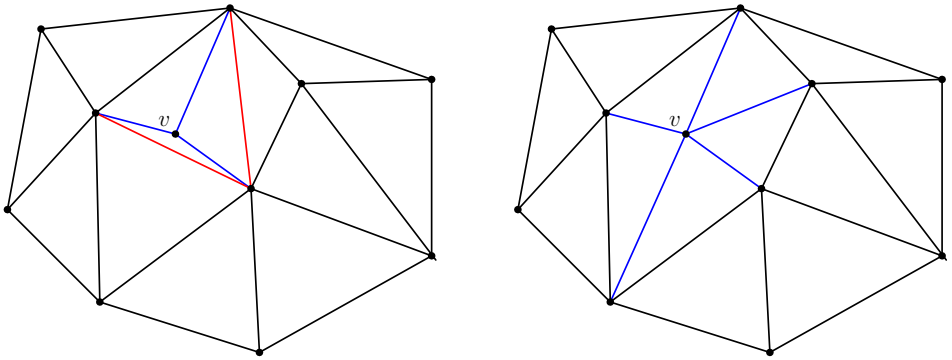


Figure 1.4: Illustration of Lawson's incremental flip algorithm. (a) Vertex v is inserted in the Delaunay triangulation of Figure 1.2b. One triangle is split into three by the blue edges. The red edges are no longer Delaunay and must be flipped. (b) The resulting Delaunay triangulation.

periphery is invisible from the vertex being inserted) or could fail to be simply connected. In both cases, a valid reconnection of the peripheral entities to the new vertex is impossible. Fortunately, these situations can be avoided with careful implementation. For instance, the first problem can be resolved by post-processing the cavity to recursively remove tetrahedra attached to invisible vertices [4]. The second is avoided by constructing the cavity using a depth-first search starting from the tetrahedron in which the new vertex is inserted. For reasons that will become clear in Section 1.2.4, almost all insertion operations in this thesis use the Bowyer-Watson algorithm.

Better algorithms are available to construct Delaunay triangulations of a known vertex set. According to results by Su and Drysdale [45] and by Shewchuk [39], the divide and conquer algorithm of Guibas and Stolfi [22] and the sweepline algorithm of Fortune [17] are significantly faster than incremental insertion algorithms. Unfortunately, the meshing algorithms used in this thesis have no a priori knowledge of the vertex locations. The faster algorithms could perhaps be used in the initialization phase, where a Delaunay triangulation of input vertices is constructed. But given the relatively small number of input vertices, the time gains would be marginal.

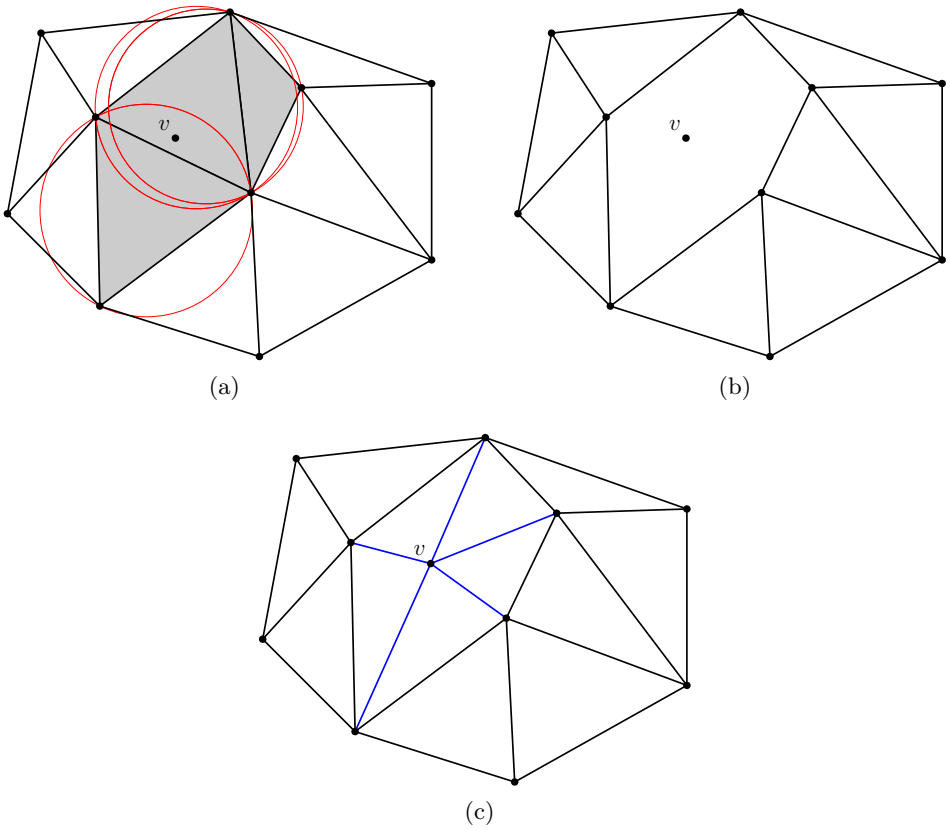


Figure 1.5: Illustration of the Bowyer-Watson algorithm. (a) Vertex v is inserted in the Delaunay triangulation of Figure 1.2b. v is located inside the circumcircle of three triangles which must be deleted from the triangulation. (b) An empty cavity is formed after the deletion. (c) The Delaunay triangulation is obtained by connecting the external edges of the cavity to v.

The extra implementation effort is therefore not warranted.

1.2.3 Boundary Conformity

One of the major challenges faced by Delaunay-based meshing algorithms is the construction of an initial triangulation conforming to the geometric domain to be meshed. A discrete representation of the domain's boundary in the form of a set of edges in two dimensions or a set of triangular faces in three dimensions is supplied as input to the mesh generator. These entities provide constraints that must be matched by the mesh. In other words, the constraining edges must appear as edges of the initial triangulation and the constraining faces must appear as faces of the initial tetrahedralization before the mesh can be refined. Most algorithms require the construction of a boundary conforming triangulation before any vertex can be inserted in the interior. There are exceptions, for instance an algorithm by Mavriplis [27] which recovers the boundary a posteriori. In this case, because the recovery procedure is performed last, it can introduce poor quality entities in the output mesh.

The Delaunay triangulation of the vertex set coming from the boundary discretization will typically fail to match all the constraining entities, as is shown in Figure 1.6. The missing entities therefore need to be recovered. The simplest way to accomplish this is by a technique commonly referred to as stitching. As it was mentioned in Section 1.2.1, edges appearing in the Delaunay triangulation are circumscribed by a circle whose interior does not contain any vertex of the triangulation. Stitching splits these missing edges by inserting additional vertices on the boundary. Each split creates two shorter edges, with smaller circumscribing circles. The edges must eventually become short enough to satisfy the empty circle property and therefore appear in the *conforming* Delaunay triangulation. The same applies for constraining faces in three dimensions. Conforming Delaunay tetrahedralization have recently been used to mesh domains bounded by piecewise smooth surfaces [11, 36]. The major problem is that recovery by stitching might require a prohibitively large number of vertices [16].

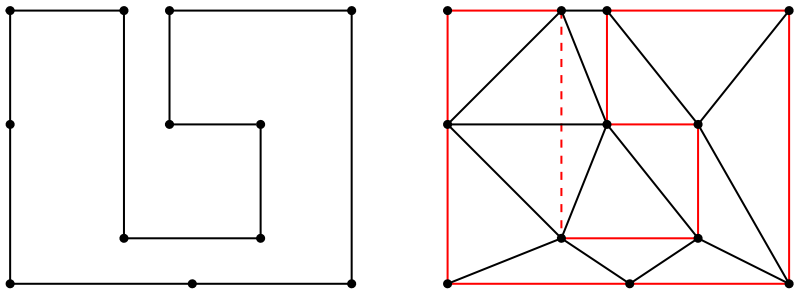


Figure 1.6: A Delaunay triangulation not conforming to the input. (a) The input domain (b) The Delaunay triangulation of the input vertices. Edges in red are part of the input. The dashed edge fails to appear in the Delaunay triangulation.

Remember from Section 1.1.2 that a good mesh generator should output meshes that are as coarse as the input will allow. This objective is defeated by the use of conforming Delaunay triangulations. A better method to enforce boundary conformity is warranted. The solution comes in the form of *constrained* Delaunay triangulations (CDT) [12]. Despite their name, CDTs are not Delaunay triangulations. By definition, a triangle in a CDT can have vertices located inside its circumcircle, as long as these vertices are not *visible* from the triangle's interior. Visibility is blocked by input edges in two dimensions and input facets in three. Furthermore, the edges of a constrained Delaunay triangle must not cut through an edge of the input domain; all triangles must respect the domain's boundary. CDTs maintain most of the Delaunay triangulation's favorable properties and interact well with Delaunay-based mesh generation algorithms [41]. Figure 1.7 illustrates the difference between boundary enforcement methods.

Enforcing boundary conformity with CDTs is fairly straightforward in two dimensions, but is much more difficult in three. The root of the problem lies in the fact that a CDT might not exists for a given set of constraining edges and faces. While it is guaranteed that a set a non-intersecting constraining edges can be recovered in a triangulation using edge flips only [26], the same does not apply for tetrahedralizations. Conditions on the existence of a CDT have been demonstrated for domains bounded by piecewise planar

surfaces by Shewchuk [41] and Si [44]. These conditions can always be satisfied by judiciously adding vertices to the input domain's boundary prior to constructing the CDT.

Alternatively, the boundary can be recovered using an approach based on sound heuristics. An attempt is first made at recovering the constraining edges and triangles using face flips. Vertices are only inserted as a last resort, when flipping fails to make further progress towards a CDT. This procedure is bound to terminate because in the worst case scenario, a conforming Delaunay tetrahedralization will eventually be obtained. In practice, fewer insertions are necessary to recover the boundary. The method offers the added bonus of inherently fixing potential intersections between the constraining faces from coarsely discretized neighboring curved surfaces. It is therefore well-suited for the applications in this thesis.

1.2.4 Delaunay Refinement Mesh Generation

With a Delaunay triangulation conforming to the boundary, vertices can now be inserted in the domain's interior. Many vertex placement strategies are available: placing the vertices on a uniform background grid, bisecting the longest edges in the triangulation [37] or using a locally optimal vertex placement as is done in advancing front methods [31] are just a few examples. However, in terms of mesh quality improvement, no strategy is more effective than inserting at a triangle's circumcenter, as was first proposed by Frey [20]. This idea later lead to the development of the Delaunay refinement paradigm [13].

Delaunay refinement algorithms take as input a Delaunay — or possibly a constrained Delaunay — triangulation and incrementally add vertices at the circumcenter of poorly-shaped triangles until they all satisfy a prescribed quality criterion. Practically relevant Delaunay refinement algorithms were first published by Chew [14] and Ruppert [38]. The method was later extended to three dimensions by Shewchuk [40].

The increasing popularity enjoyed by Delaunay refinement can partly be attributed to the guarantees it offers regarding mesh quality. Provably-good

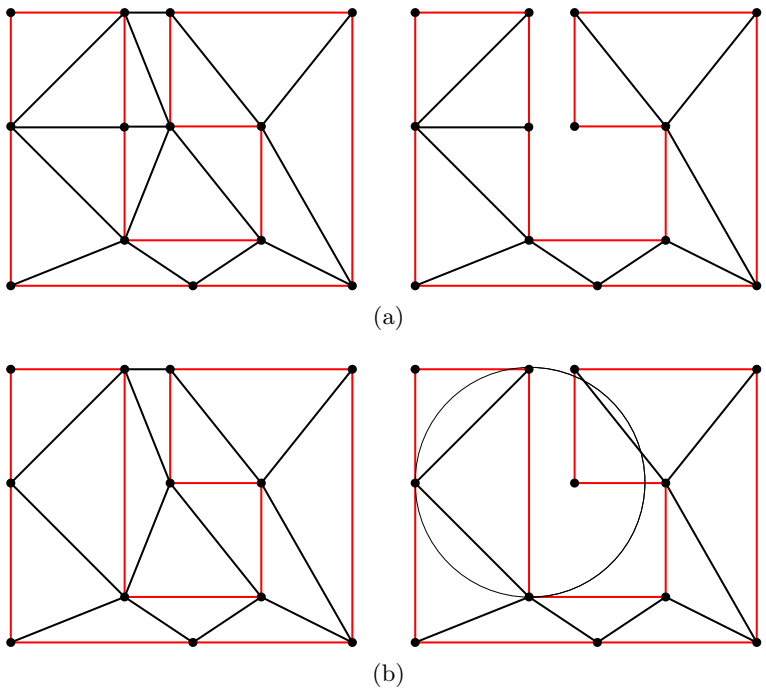


Figure 1.7: A comparison between conforming and constrained Delaunay triangulations is illustrated. In both cases, once the boundary is recovered, the triangles exterior to the domain are deleted. (a) Recovery by stitching. One additional vertex was added to the domain's boundary. (b) Boundary enforced by *constrained* Delaunay triangulation. The missing edge was recovered by flipping. A triangle contains a vertex in its circumcircle, but this vertex is occluded from the triangle by a constraining edge.

Delaunay refinement algorithms target triangles and tetrahedra having a large *circumradius-to-shortest-edge ratio* with circumcenter insertions. Repeated insertions can decrease this ratio down to a certain provable bound. As this quality measure is directly related to a triangle's minimum angle, reducing its value equates to increasing the minimum angle in the mesh. Indirectly, the maximum angle is also reduced. Unfortunately a similar relationship does not exist between this measure and a tetrahedron's dihedral angles. As is depicted in Figure 1.8, a sliver tetrahedron has a small circumradius-to-shortest-edge ratio despite having dihedral angles arbitrarily close to 0°.

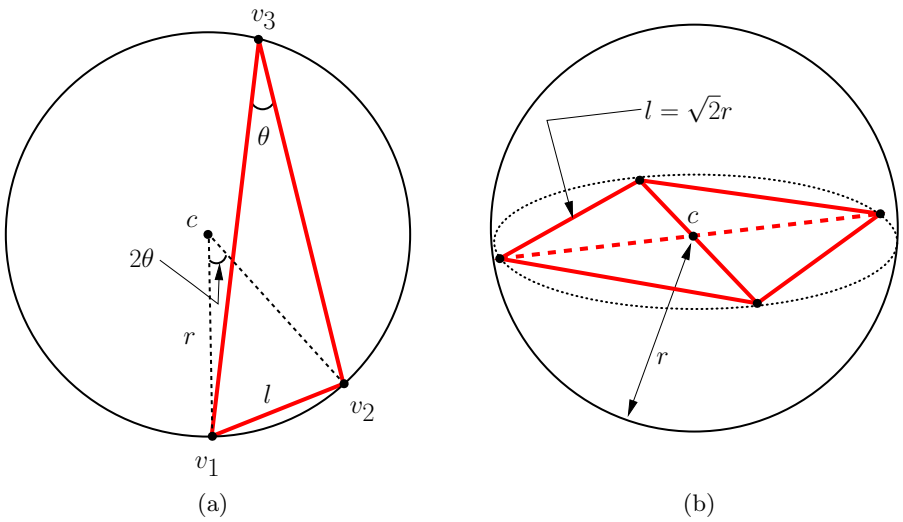


Figure 1.8: (a) Triangle $\triangle v_1 v_2 v_3$ with its circumcenter at c has a minimum angle θ. From the fact that $\angle v_1 c v_2 = 2\theta$, it is easy to show that θ is related to the circumradius-to-shortest-edge ratio r/l by the formula $\theta = \arcsin\left(\frac{l}{2r}\right)$. (b) The circumradius-to-shortest-edge ratio of a canonical sliver is $\sqrt{2}/2$.

Consequently, slivers cannot be provably eliminated from the mesh since they are never targeted with insertions. As Chapter 4 will demonstrate, driving the refinement with different quality measures is an effective way of eliminating slivers in practice.

Delaunay refinement algorithms are very easy to state because they rely on a few well-defined high-level insertion operations. Vertices keep being inserted at the circumcenters of poorly-shaped mesh elements, unless the proposed insertion is located inside the smallest circumscribing circle (or sphere) of a lower-dimensionality boundary entity. The location is then said to be *encroaching* on the boundary, in which case the proposed insertion is rejected. Instead, a vertex is inserted on the boundary to split the entity that is encroached upon. This form of boundary protection prevents unnecessarily small edges from being created and precludes vertices from being inserted outside the domain.

In this context, the choice of the Bowyer-Watson algorithm to perform

vertex insertions becomes clearer. When a circumcenter insertion is proposed, the cavity of elements to be deleted is constructed by performing a depth-first search, starting from the element owning the proposed circumcenter. When the search hits a boundary entity, a test for encroachment is immediately performed. Both the cavity construction and encroachment checks are therefore efficient local operations. Furthermore, the search is never allowed to cross a boundary entity. This is ideal when refining CDTs. Elements whose visibility is blocked from the new vertex by a boundary entity will never be considered in the search. Enforcing the constrained Delaunay property is therefore easy.

Termination of Delaunay refinement algorithms is conditional to the absence of small angles in the input. If small angles are present, modifications to the algorithm are necessary. The treatment of small input angles is still an active area of research, especially for three dimensional domains [35].

1.2.5 Mesh Improvement

Once a mesh is generated, its quality can potentially be improved during a post-processing step. As Delaunay refinement is known to produce meshes containing sliver tetrahedra, a clean-up algorithm can be used to eliminate these unwanted entities.

Mesh improvement algorithms typically utilize two operations to achieve their goal: topological transformations and smoothing. Topological transformations locally reconfigure the mesh. The mesh connectivity is changed with the objective of replacing undesirable entities by a different set of entities occupying the same space. As such, topological transformations only modify a very small portion of the mesh. Figure 1.9 illustrates the face flipping operations: the 2-3 flip, the 3-2 flip and the 4-4 flip. A 2-2 flip is also possible for boundary tetrahedra, which amounts to flipping a surface edge. Edge swaps replacing n tetrahedra with $2n - 4$ tetrahedra can also be used. The 3-2 and 4-4 flips are actually the simplest examples of edge swaps.

Smoothing on the other hand keeps the connectivity constant, but moves vertices about to improve the quality of entities adjoining them. Laplacian

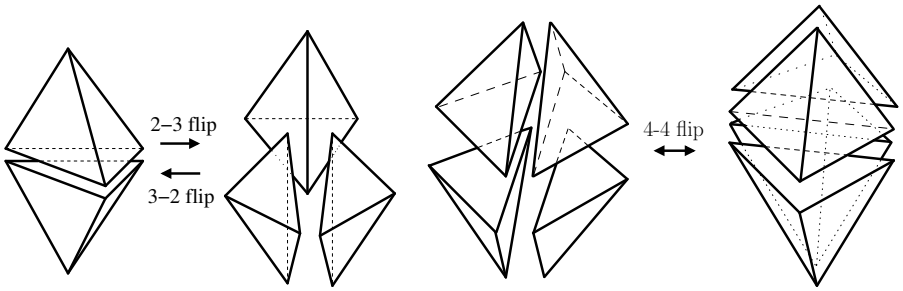


Figure 1.9: Some face flipping operations potentially used by tetrahedral mesh improvement algorithms.

smoothing is perhaps the best known and the most commonly used smoothing technique. The idea is to calculate the new position of a vertex based on the location of its neighbors. The centroid of the vertex's neighbors is typically chosen as the new location. The method is extremely easy to implement and is fairly effective in two dimensions, but less effective in three. A far better alternative for tetrahedral meshes is to use optimization-based smoothing. An objective function relating the position of each vertex to the quality of its adjacent tetrahedra is defined. A gradient-based algorithm is then used to find the vertex positions by locally maximizing the objective function.

Freitag and Ollivier-Gooch report that topological transformations and optimization-based smoothing are most effective when used concurrently [18]. They achieve their best results when maximizing the minimum sine of a tetrahedron's dihedral angle. However, since the method attempts to optimize non-convex functionals using a gradient-based approach, it tends to quickly get stuck at local optima. This issue is addressed in more recent research. Alliez et al. [1] define a global energy measure which, when minimized, translates into a quality tetrahedralization. On the other hand, Klingner and Shewchuk [24] use what they refer to as compound operations, a combination of several mesh improvement operations, in the hopes of escaping one "valley" and finding a deeper one. The example meshes presented in both publications are of spectacular quality, but this quality comes at a price. Despite the fact that Klingner and Shewchuk probably present the best quality tetrahedral

meshes ever published, their algorithm takes between one and a half to four hours to optimize a mesh of one hundred thousand tetrahedra, depending on the scheduling and the objective function used. Given that it is not uncommon for numerical simulations to use meshes with millions of tetrahedra, spending so much time on mesh improvement, regardless of how good the results are, is impractical.

1.2.6 Canonical Mesh Generation Algorithm

Based on the background information presented in Sections 1.2.1 to 1.2.5 a canonical meshing method can now be stated. This approach is the basis for the algorithms developed in this thesis. The procedure consists of five phases. The output is a quality mesh conforming to the boundary of an input geometric domain.

1. *Initialization*: The domain to be meshed is defined; domains bounded by piecewise smooth curves and surfaces are acceptable. When curved boundaries are present, the initialization phase includes a boundary discretization operation. Boundary discretization is fairly straightforward in two dimensions, but much more difficult in three. The majority of Chapter 3 is devoted to the discretization of curved surfaces. Small angles are then protected to allow for termination. Optionally, a size distribution function which will be matched by the mesh can also be defined at this stage.

2. *Mesh construction*: Since an incremental Delaunay method is used, an initial Delaunay triangulation (tetrahedralization) must be constructed. To this end, a bounding box enclosing the input domain is defined. In 2D, a rectangular box is meshed with two triangles. In 3D, a rectangular prism is meshed with five tetrahedra. Vertices coming from the input and the boundary discretization are then incrementally inserted in the mesh while maintaining the Delaunay property.

3. *Boundary recovery*: Boundary recovery operations are arguably the most critical to the success of Delaunay-based mesh generation. In

most instances, the mesh constructed in stage 2 will not conform to the domain's boundary. As opposed to advancing front algorithms, where boundary conformity is inherent to the method, Delaunay-based schemes require the boundary to be explicitly recovered before any interior point are inserted.

4. *Interior insertions*: A point placement strategy based on the Delaunay refinement paradigm is used to improve mesh quality. Vertices are inserted at the circumcenter of poorly-shaped elements until a prescribed quality criterion is achieved by all elements. Insertions can optionally be performed such that the mesh matches a size distribution function.

5. *Mesh improvement*: Once Delaunay refinement terminates, the mesh can be further improved in a post-processing step. In two dimensions, given the strong quality guarantees offered by Delaunay refinement, only marginal quality gains are generally possible. However, in three dimensions, post-processing can be used to remove slivers. Note that the application of mesh improvement algorithms is typically the costliest stage of the entire procedure.

1.3 Objectives and Outline

A mesh generator based on the Delaunay refinement paradigm can offer guarantees regarding the quality and size optimality of the meshes it outputs [14, 38, 40], making it a good choice to create meshes for numerical simulations. As theoretical meshing analysis is often based on the assumption that the input geometry is piecewise linear or piecewise planar, it is not surprising that Delaunay refinement algorithms were originally developed under this assumption. However, in the context of high-order accurate finite-volume methods, this restriction is unacceptable. In fact, proper treatment of curved boundaries is paramount to achieving the scheme's nominal order of accuracy. For high-order methods to benefit from the desirable properties offered by Delaunay refined meshes, the meshes must be constructed directly from curved geometries. In this context, the central objective of this thesis is

to study the application of Delaunay refinement mesh generation algorithms to domains bounded by piecewise smooth curves and surfaces. This broad objective is attained by considering a number of specific, previously unresolved practical issues. Ultimately, this study should allow generation of meshes directly from clean CAD models. Although this long term objective has not yet been reached, the meshing algorithms herein proposed are designed for this purpose. Proper access to CAD geometry via the mesh generator remains an unresolved issue.

The core chapters of this thesis are composed of three manuscripts submitted for journal publication. Further motivation and relevant background information is therefore addressed in each individual chapter.

In Chapter 2, the problem of triangular mesh generation is addressed by applying Shewchuk's hybrid algorithm [42] to geometries bounded by curves. This work can be seen as an extension of a previously published algorithm by Boivin and Ollivier-Gooch [8]. The major improvement comes in the form of an initial curve discretization procedure leading to a priori enforcement of the constrained Delaunay property on the constraining edges. As a result, these edges are all unencroached and are recoverable by edge flips only. The launch of the refinement phase is therefore simplified. The procedure is guaranteed to terminate without creating unnecessarily short boundary edges, which would inevitably result in over-refined regions in the mesh. If the input does not contain curves meeting at angles of less than 60°, the algorithm is guaranteed to terminate producing a mesh where all angles are above 25.7°. In practice, termination is always possible with an angle bound as high as 30°. When the input contains angles smaller than 60°, modifications are necessary for the algorithm to terminate. Pav and Walkington [34] propose an algorithm with termination guarantees in the presence of small angles subtended by curves. Although this is a major achievement, their implementation suffers from flaws that makes it less attractive for practical applications. In this work, small angles are handled by extending a protection scheme devised by Miller et al. [30] to curved input. The method works well in practice even though the argument at the basis of the termination proof does not apply to curves. No failure to terminate has been observed in our experiments.

The meshing algorithm presented in this chapter has already been used by Michalak [29] to perform high-order accurate aerodynamics simulations.

Chapter 3 is devoted to the construction of constrained Delaunay tetrahedralizations from piecewise smooth geometries. This is undoubtedly the most difficult part of the entire meshing process. The chosen approach is closely related to that originally proposed by Boissonnat and Oudot [6, 7]. It combines Chew's furthest point strategy [14] for surface refinement with the Voronoi filtering technique coming from sampling theory [2]. The Delaunay tetrahedralization of a set of sample vertices located on the model's surface is maintained. The faces of the tetrahedralization whose dual Voronoi edge intersect form a subset known as the *restricted Delaunay triangulation*. The main result provided by Boissonnat and Oudot is that when the sample becomes dense enough, the restricted Delaunay triangulation becomes homeomorphic to the model's surface. Their algorithm requires the knowledge of a local feature size, which is computed using the medial axis transform. A similar algorithm by Cheng et al. [11] relies on topological violations to build the sample. In both cases, the entire model is sampled at once, which inevitably lead to a conforming Delaunay tetrahedralization. The approach presented in this thesis is also driven by topological violations, but samples each individual surface patch independently. It offers three main advantages over previously published methods: it only requires a limited set of topological tests, it generates coarser meshes for many cases of practical interest and it obviates the need to protect small input dihedral angles, which is the main implementation difficulty in Cheng et al.'s algorithm. On the flipside, a tetrahedralization combining the samples from every surfaces patches must be constructed once the sampling phase is completed. Its boundary must then be recovered.

The CDTs constructed in Chapter 3 are an ideal starting point to apply Delaunay refinement to generate tetrahedralization with good quality properties. Chapter 4 studies the practical capabilities of Delaunay refinement for tetrahedral meshes. In three dimensions, Shewchuk's algorithm [40] is able to generate almost-good meshes. Because the refinement uses the circumradius-to-shortest-edge ratio to evaluate a tetrahedron's quality, the al-

gorithm makes no attempt at removing slivers. They therefore survive in the mesh. Post-processing is then necessary to remove these slivers, with various levels of success [10, 15, 18]. We show that by using non-standard quality measures, tetrahedralizations with dihedral angles between 18° and 154° can consistently be obtained. The number of insertions needed to achieve this level of quality is comparable to that required using the circumradius-to-shortest-edge ratio. This chapter also explores the use of mesh improvement techniques on Delaunay refined meshes. Our results indicate that the quality properties of the improved meshes are independent of the quality measure used during refinement. The Delaunay refinement algorithm presented in Chapter 4 is the first one that is capable of handling CDTs of domains bounded by piecewise smooth surfaces. Finally, in Chapter 5, the significance of the results presented in the manuscript chapters are discussed. A number of possible improvements and extensions to the method are also suggested.

1.4 Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617 – 625, 2005.

[2] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22(4):481–504, 1999.

[3] Ivo Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.

[4] Timothy J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, 5(3-4):161–175, 1989.

[5] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In Ding-Zhu Du and Frank Hwang, editors, *Computing in Euclidean geometry*, pages 23–90. World Scientific, 1992.

[6] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.

[7] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of Lipschitz surfaces. In *Proceedings of the Twenty-Second Annual Symposium on Computatinal Geometry*, pages 337–346, 2006.

[8] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1203, 2002.

[9] Adrian Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[10] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver exudation. *Journal of the ACM*, 47:883–904, 2000.

[11] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 477–494, 2007.

[12] L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[13] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell Univeristy, 1989.

[14] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280, 1993.

[15] Herbert Edelsbrunner and Damrong Guoy. An experimental study of sliver exudation. *Engineering with Computers*, 18:229–204, 2002.

[16] Herbert Edelsbrunner and Tiow Seng Tan. An upper bound for conforming Delaunay triangulations. *Discrete and Computational Geometry*, 10(1):197–213, 1993.

[17] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987.

[18] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[19] Lori A. Freitag and Carl Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *International Journal of Computational Geometry and Applications*, 10(4):361–382, 2000.

[20] William H. Frey. Selective refinement: A new strategy for automatic node placement in graded triangular meshes. *International Journal for Numerical Methods in Engineering*, 24:2183–2200, 1987.

[21] Isaac Fried. Condition of finite element matrices generated from nonuniform meshes. *AIAA Journal*, 10:219–221, 1972.

[22] Leonidas J. Guibas and Jorge Stolfi. Manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[23] C. R. Illingworth and Samuel J. Goldstein. Some solutions of the equations of flow of a viscous compressible fluid. *Mathematical Proceedings of the Cambridge Philosophical Society*, 46(3):469–478, 1950.

[24] Bryan M. Klingner and Jonathan R. Shewchuk. Agressive tetrahedral mesh improvement. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 3–23, 2007.

[25] Michal Křížek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.

[26] Charles L. Lawson. Software for C^1 surface interpolation. In John R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.

[27] Dimitri J. Mavriplis. Unstructured mesh generation and adaptivity. Technical Report 95-26, ICASE, 1995.

[28] Fotis Mavriplis. CFD in aerospace in the new millenium. *Canadian Aeronautics and Space Journal*, 46(4):167–176, 2000.

[29] Christopher Michalak. *Effcient High-Order Accurate Unstructured Finite-Volume Algorithms for Viscous and Inviscid Compressible Flows*. PhD thesis, Department of Mechanical Engineering, The University of British Columbia, 2009.

[30] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the Twelfth International Meshing Roundtable*, pages 91–102, 2003.

[31] Jens-Dominik Müller, Philip L. Roe, and Herman Deconinck. A frontal approach for internal node generation in Delaunay triangulations. *International Journal for Numerical Methods in Fluids*, 17(3):241–255, 1993.

[32] Carl Ollivier-Gooch, Amir Nejat, and Christopher Michalak. On obtaining and verifying high-order finite-volume solutions to the Euler equations on unstructured meshes. *American Institute of Aeronautics and Astronautics Journal*. To appear.

[33] Carl F. Ollivier-Gooch and Micheal Van Altena. A high-order accurate unstructured mesh finite-volume scheme for the advection-diffusion equation. *Journal of Computational Physics*, 181(2):729–752, 2002.

[34] Steven E. Pav and Noel J. Walkington. Delaunay refinement by corner lopping. In *Proceedings of the Fourteenth International Meshing Roundtable*, pages 165–181, 2005.

[35] Alexander Rand and Noel Walkington. 3D Delaunay refinement of sharp domains without a local feature size oracle. In *Proceedings of the Seventeenth International Meshing Roundtable*, pages 37–54, 2008.

[36] Laurent Rineau and Mariette Yvinec. Meshing 3D domains bounded by piecewise smooth surfaces. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 443–460, 2007.

[37] Maria-Cecilia Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21(3):604–613, 1984.

[38] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[39] Jonathan R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry*

Towards Geometric Engineering, pages 203–222. Springer Berlin / Heidelberg, 1996.

[40] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.

[41] Jonathan R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the Eleventh Meshing Roundtable*, pages 193–204, 2002.

[42] Jonathan R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):27–74, 2002.

[43] Jonathan R. Shewchuk. What is a good linear element? Interpolation, conditioning, anisotropy, and quality measures. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 115–126, September 2002.

[44] Hang Si and Klaus Gärtner. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the Fourteenth International Meshing Roundtable*, pages 147–163, 2005.

[45] Peter Su and Robert L. Drysdale. A comparison of sequential Delaunay triangulation algorithms. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, pages 61–70, 1995.

[46] David F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

[47] Nigel P. Weatherhill. The generation of unstructured grids using Dirichlet tesselation. Technical Report MAE 1715, Princeton University, 1985.

Chapter 2

Triangular Mesh Generation *

2.1 Introduction

The mesh generation process, during which a geometry is divided into subdomains suitable for computation, is an integral part of numerical simulations. In fact, meshing often forms a bottleneck in the simulation cycle. While many unstructured meshing tools are known for their ability to automatically discretize complex geometries, effectively widening the bottleneck, they may not satisfy a numerical solver's need for mesh quality. Triangular meshes containing large or small angles are known to adversely affect finite element discretization schemes [1, 7]. Experimental evidence shows that finite volume schemes also suffer from poor quality triangles [6]. The problem is compounded by the fact that only a handful of bad triangles are sufficient to compromise a simulation's performance [18].

In this context, being able to automatically generate high quality meshes can potentially decrease the time devoted to simulation preprocessing. The need for an algorithm capable of achieving this objective has lead to the appearance of the Delaunay refinement paradigm in mesh generation. By inserting vertices at the circumcenter of poorly-shaped triangles, Delaunay refinement algorithms eliminate small angles from a triangulation, thus improving its quality. The first practically relevant algorithms are due to Ruppert [14] and Chew [4]. Under certain input conditions, these two

*A version of this chapter will be submitted for publication. Gosselin, S. and Ollivier-Gooch, C. Revisiting Delaunay Refinement Triangular Mesh Generation on Curve-bounded Domains.

algorithms are guaranteed to terminate, producing graded and size-optimal meshes with bounded smallest angle. Ruppert's and Chew's schemes are sufficiently similar for Shewchuk to analyze them in a common framework [15, 17]. His analysis lead to a new hybrid algorithm and relaxed restrictions on the input domain. Shewchuk's algorithm is outlined in Section 2.3.

The aforementioned algorithms have the common shortcoming of only accepting input geometries whose boundaries are piecewise linear. If the domain to be meshed is bounded by piecewise smooth curves — a collection of parametric curves for instance — a set of approximating segments must first be created, more or less arbitrarily if no sizing field is provided, before being fed to the meshing algorithm. The mesh to be generated then becomes dependent on the density of the initial sampling: too dense and the mesh will contain far too many triangles, too coarse and the boundaries might not be correctly resolved. Such an approach also has the obvious drawback of discarding all curvature information, which is needed if one wants to conduct a high-order accurate finite volume simulation [11]. A far better alternative is to build the mesh directly from the curves defining the geometric domain, as was first done by Boivin and Ollivier-Gooch [2]. Their strategy involves the creation of a protective region in the curves' neighborhood. This is achieved by bounding the tangent variation between two successive vertices along boundary curves. The presence of the protective region forbids new vertices from being inserted too close to the domain's boundary, thus preventing arbitrarily small edges from being created during the refinement process. Termination can then be proved, with the same quality guarantees offered by previous algorithms. However, the authors recognize that their curve sampling procedure is, in general, not sufficient to create a Delaunay triangulation constrained to the discrete representation of the boundary; the initial condition needed to launch Delaunay refinement cannot be met. They propose a series of heuristics to overcome this problem. These can be tedious to implement and sometimes achieve boundary recovery only through the addition of a large number of vertices. In Section 2.4, we revisit the boundary conformality problem and propose a pre-refinement technique allowing us to obtain a Delaunay triangulation constrained to a curve-bounded domain

using only edge flips.

Pav and Walkington [13] also extended the application of Delaunay refinement to curves. They argue that the algorithm presented by Boivin and Ollivier-Gooch is not shown to work when small angles are present in the input. Using Ruppert's corner lopping heuristic [14], they show termination without requiring a minimum angle between adjoining input curves. Although this is a major theoretical achievement, their implementation suffers from flaws that makes it less desirable for practical applications. Nonetheless, they clearly highlight the fact that small angles are the primary mechanism through which Delaunay refinement may fail. We handle small angles slightly differently, in a way similar to that proposed by Miller et al. [8], as is presented in Section 2.5.

In this work, we present a method allowing Shewchuk's hybrid algorithm to be applied to curve-bounded domains. Our main contribution is the boundary recovery strategy mentioned above, which does not create arbitrarily short boundary edges. As such, the final refined mesh does not contain unnecessarily small triangles due to an over-refined boundary. Our implementation is available within the GRUMMP meshing libraries [10] and uses the Common Geometry Module [19] as its geometric back-end. Shewchuk's algorithm terminates using an angle bound of 30°, although the guaranteed bound is lower at 25.7°. As is shown in the results presented in Section 2.6, our algorithm achieves the same practical bound.

2.2 Preliminaries

We begin by defining the input to the meshing algorithm, which we name Planar Curve Complex (PCC).

Definition 2.1. PCC. A *Planar Curve Complex* is a set of points $\mathcal{P}$ and a set of piecewise smooth curves $\mathcal{C}$ embedded in $\mathbb{R}^2$ such that

- A curve $c \in \mathcal{C}$ is either non-closed and bounded by two endpoints p_1, $p_2 \in \mathcal{P}$, or is closed and bounded by a single endpoint $p \in \mathcal{P}$.

- Two curves c_1, $c_2 \in \mathcal{C}$ can only intersect each other at an endpoint common to both curves.
- Points in $\mathcal{P}$ can only intersect curves in $\mathcal{C}$ at an endpoint.

We require that the PCC be curve-bounded — the boundary separating the domain to be meshed from its exterior must entirely be covered by curves. Note that if all curves in $\mathcal{C}$ are straight line segments, then Definition 2.1 is equivalent to that of a Planar Straight Line Graph (PSLG), which is the input accepted by the Delaunay Refinement algorithms of Ruppert [14], Chew [4] and Shewchuk [17].

During mesh generation, boundary curves are approximated by piecewise linear segments. Given v_1 and v_2, two successive vertices along curve $c \in \mathcal{C}$, we name the segment of c bounded by v_1 and v_2 *subcurve*. We use *boundary edge* to describe the line segment $\overline{v_1 v_2}$.

The mesh generation algorithm outputs a triangulation conforming to the PCC's boundaries: all points in $\mathcal{P}$ are vertices in the triangulation, all curves in $\mathcal{C}$ are approximated by boundary edges, appearing as edges of the final triangulation. Boundary conformality is enforced by constrained Delaunay triangulations (CDT). In general, CDTs achieve boundary conformality using less boundary vertices than conforming Delaunay triangulations [5, 16], which in turns leads to coarser output triangulations. According to Shewchuk [16] and as illustrated in Figure 2.1, a simplex is said to be constrained Delaunay if it conforms to the boundary of the PCC (it does not cross any boundary edges) and if there exists a circle circumscribing the simplex such that the circle does not contain any vertex visible from the simplex's relative interior. Once the algorithm terminates, the triangulation it outputs only contains edges and triangles respecting the constrained Delaunay property. Most of these triangles are of good quality, except those located in the neighborhood of small input angles. In these regions, ill-shaped triangles are tolerated.

To prove that his algorithm terminates and outputs a mesh where vertex spacing is within a constant factor of optimal spacing, Ruppert introduces a function called the local feature size [14]. We adapt his classical definition, noting that it applies to PCCs whose curves have been discretized.

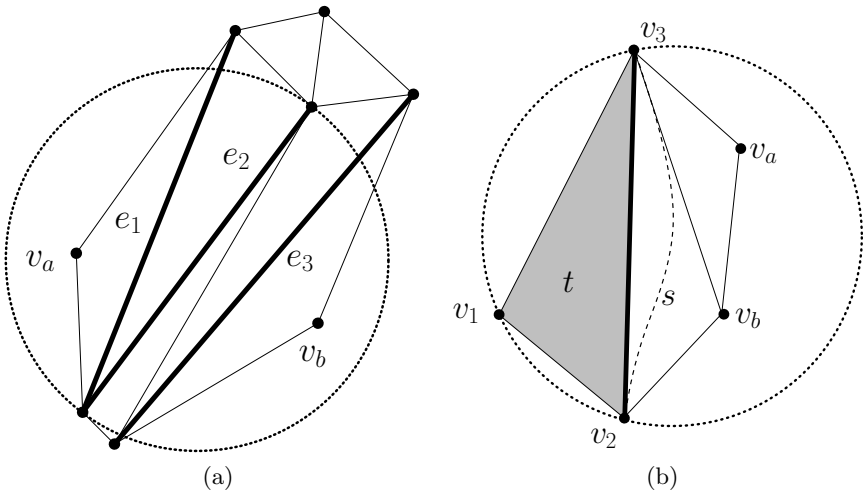


Figure 2.1: Constrained Delaunay simplices: (a) Edges e_1, e_2 and e_3 are linear approximations of some underlying boundary curves, e_2 is constrained Delaunay; vertices v_a and v_b are contained in its circumscribing circle, but are not visible due to the presence of e_1 and e_3. (b) Triangle t is constrained Delaunay; visibility of vertices v_a and v_b from its interior is blocked by boundary edge $\overline{v_2v_3}$, approximating subcurve s.

Definition 2.2. Local feature size. Given a discretized PCC X, the *local feature size* at a point p, noted $\mathrm{lfs}\,(p)$, is the radius of the smallest disk centered at p that intersects two non-incident points or subcurves of X.

According to Definition 2.1, a curve in the input PCC can be closed: the curve forming the mock airfoil depicted in Figure 2.2 is a good example. By strictly applying Ruppert's original local feature size definition to a point p located on the airfoil's upper portion, then $\mathrm{lfs}(p)$ would neither depend on the distance to the lower portion of c, nor on the distance to point q, which is incident on c. However, after splitting c into a number subcurves, $\mathrm{lfs}(p)$ becomes a function of the distance to the lower portion of c, a better approximation of the optimal vertex spacing at p for a triangulation of the airfoil's interior. Since we do not make explicit use of the local feature size until a boundary-constrained triangulation has been obtained, a quantitative description of how the curves must be discretized to achieve a satisfactory

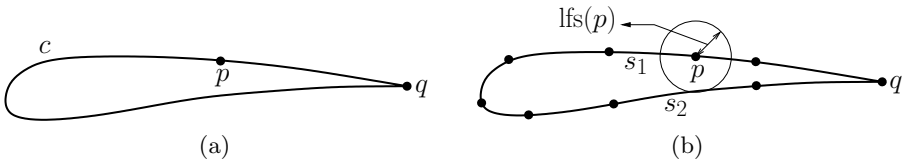


Figure 2.2: (a) A mock airfoil made of one closed curve. lfs(p) does not depend on the distance to the lower surface or the distance to q. (b) After c has been discretized, lfs(p) is the distance to s_2, a subcurve non-incident on subcurve s_1.

approximation of lfs is not necessary. A qualitative measure, forbidding a subcurve to turn too much over its span for instance, is sufficient to apply the definition.

Before being able to build a triangulation of the input PCC, the meshing code needs to generate an initial set of straight segments approximating the curves of the PCC. The resulting subcurves will then possibly be split during a refinement operation. Initial discretization and subcurve splits are both governed by the total variation of the tangent angle, as proposed by Boivin and Ollivier-Gooch [2].

Definition 2.3. Total variation of the tangent. The total variation of the tangent angle along a curve c is $TV \equiv \int_c |d\theta|$, where θ is the angle subtended by the tangent of the curve.

2.3 Delaunay Refinement

Delaunay refinement describes a family of algorithms known for their ability to generate meshes that are particularly well-suited for numerical methods. These algorithms all proceed in the same general fashion: they improve Delaunay (or possibly constrained Delaunay) triangulations by incrementally inserting vertices at carefully chosen locations, while maintaining the (constrained) Delaunay property. In this work, we adapt an algorithm originally proposed by Shewchuk [15, 17] so it can accept curve-bounded domains as input.

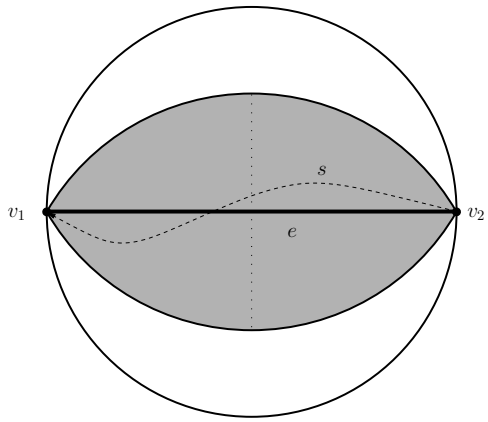


Figure 2.3: The diametral lens (shaded) and diametral circle of a boundary edge, e, approximating subcurve s. The lens is the interior of two circular arcs intersecting at v_1 and v_2. Each arc has its center on the other arc, along the bisector of e. e is encroached if a vertex is located inside its diametral lens.

Given an initial constrained Delaunay triangulation, Shewchuk's algorithm eliminates bad quality triangles by inserting a new vertex at their circumcenter. To avoid adding a vertex either outside the domain or too close to the boundary, which would create unnecessarily short edges, inserting a vertex encroaching on a subcurve is forbidden. A subcurve is said to be encroached upon if the diametral lens of its corresponding boundary edge contains a *visible* vertex, as is illustrated in Figure 2.3.

The algorithm is simple to state. It relies on only two high-level operations that it keeps performing until it can no longer do so:

- If a triangle is of bad quality or too large with respect to some sizing field, a vertex is inserted at its circumcenter, UNLESS

- the proposed insertion location encroaches on one or more subcurves. In this case, the vertex is not inserted; instead, encroached subcurves are split into two subcurves. If the split point encroaches on other subcurves, those must also be split. Before splitting a subcurve, all visible *interior* vertices located inside its boundary edge's diametral

circle must be deleted from the mesh.

Note that subcurve splitting always takes priority over circumcenter insertion.

The quality of a triangle is taken to be inversely proportional to its minimum angle. By bounding the minimum angle of a triangle, $\theta_{\min}$, one also bounds its maximum angle such that $\theta_{\max} \leq \pi - 2\theta_{\min}$. The minimum angle of a triangle can be related to its circumcenter-to-shortest edge ratio r/l, by the formula $r/l = 1/2 \sin \theta_{\min}$. This ratio is often used in the analysis of Delaunay refinement algorithms because it is naturally improved by circumcenter insertions. Shewchuk is able to demonstrate that by using an upper bound of $\frac{2\sqrt{3}}{3}$ on the circumradius-to-shortest-edge ratio (implying a minimum angle $\theta_{\min} \approx 25.7°$), his algorithm terminates and outputs a graded triangulation. This termination guarantee is conditional to the fact that the input does not contain any segments subtending an angle of less than 60°. As is discussed in Section 2.5, special care must be taken to achieve termination in the presence of small input angles.

Delaunay refinement is famous for its ability to outperform its theoretical guarantees. In practice, Shewchuk's algorithm terminates using a ratio $r/l = 1$ ($\theta_{\min} = 30°$) while still generating a nicely graded mesh. Miller et al. [8] even state that it would be difficult to imagine a geometry for which Delaunay refinement would fail to terminate when using an angle bound of 30°. The grading property however deteriorates when the angle bound is pushed higher. The algorithm will eventually fall into an infinite refinement loop and will fail to terminate.

2.4 Application to Curves

In this section, we present how we use Shewchuk's Delaunay refinement algorithm to mesh curve-bounded inputs. One of the main challenges of this process is to achieve the algorithm's starting condition, which requires a constrained Delaunay triangulation of the input geometry. To construct this triangulation, a piecewise linear approximation of the boundary curves is necessary. Obtaining a good approximation, one from which a satisfactory

CDT can be generated, is not straightforward.

A coarse initial sample of the boundary curves is first created. The sample vertices are then connected with boundary edges, which need to match edges of the CDT. Unfortunately, some of these boundary edges could be intersecting. Obviously, since a valid triangulation cannot have intersecting edges, constructing a CDT directly from the initial sample might prove impossible.

The solution is to enrich the sample a priori, thereby improving the boundary approximation. Before a triangulation is built, the discrete boundary is refined until all its boundary edges respect the constrained Delaunay property. From a Delaunay triangulation of the augmented sample, the boundary edges can be recovered, and a CDT obtained, simply by using edge flips.

This initial CDT is the perfect search structure to compute the local feature size at boundary vertices. The lfs is used to define an automatic sizing field, allowing control over the size of triangles output by the meshing algorithm. Based on this sizing field, the triangulation can then be refined until both size and quality targets are met.

2.4.1 Initial Curve Sampling

To launch Shewchuk's Delaunay refinement algorithm, a CDT whose boundary approximates that of the input geometry is necessary. Since the input geometry is a PCC, its boundary typically includes piecewise smooth curves. A discrete approximation of these curves must first be computed and later be matched with edges of the CDT. An initial sample is easily created by placing some vertices along each curves. Each two successive vertices are then connected by a boundary edge (each of which having a corresponding subcurve) to obtain the approximation. The question is how to select the sampling vertices such that a satisfactory CDT may eventually be constructed?

The number of triangles in the final mesh will closely depend on the density of the boundary sample. On the one hand, we would like to generate a triangulation that is as coarse as possible since it is generally easier to

refine a mesh, while maintaining its quality properties, than it is to coarsen it. On the other hand, the discrete boundary must not be so coarse as to cause vertices to be inserted too close to a curve, or worst, to be inserted outside the PCC while applying Delaunay refinement.

Boivin and Ollivier-Gooch [2] proposed a sampling strategy based on the total variation of the tangent angle, TV (see Definition 2.3). All curves of the input PCC not satisfying a desired bound on TV are split into two subcurves. Subcurves are then split recursively, until TV measured over their span becomes small enough for them to be contained inside their boundary edge's diametral lens. By the insertion rules presented in Section 2.3, all attempts at placing vertices inside one of these lenses will be rejected. The union of diametral lenses therefore creates a protection zone in the neighborhood of the PCC's boundary. Under these conditions, inserting a vertex inside the triangulation but outside the PCC is impossible.

Sampling curves using TV has some interesting benefits. Notably, this measure naturally adds more discretizing vertices where curvature is high; geometric resolution is better where it is needed. In Section 2.4.3, we show how the local feature size can be used to control triangle sizes. Since lfs evaluation is based on a discretized domain, TV-based sampling inherently leads to triangles whose size is proportional to the boundary's local curvature. TV along curves is also relatively easy to calculate. If the curve's equation is available, it can be computed exactly, simply by finding curve points between which the change in TV is monotone [2]. If however the curve definition is only accessible through point queries, a good enough approximation can be obtained numerically.

All subcurves are guaranteed to be lens protected if $TV < \pi/3$ along their span. This can easily be shown by lumping all of the change in tangent angle at a single point — this is equivalent to having a subcurve made of two straight segments joining at an elbow where local curvature approaches infinity (see Figure 2.4).

Our adaptation of Shewchuk's algorithm runs well even if all subcurves are not lens protected, as long as TV along their span remains smaller than $\pi/2$. Part of a subcurve with $\pi/3 \leq TV < \pi/2$ can be outside its boundary

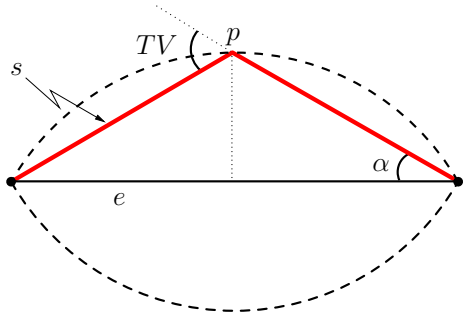


Figure 2.4: Lens protection of a subcurve: subcurve s is drawn in red, its boundary edge e in black, its diametral lens is dashed. When the subcurve's change in tangent is lumped at any single point p on e's diametral lens, then $\pi/3 \leq TV < \pi/2$ (here $\alpha = \pi/6$ and $TV = \pi/3$). A subcurve having $TV < \pi/3$ will be contained inside e's diametral lens.

edge's diametral lens, but still be fully contained inside the diametral circle (Figure 2.5). In these conditions, a vertex can be inserted between a subcurve and its boundary edge; the vertex is located outside the PCC, but remains inside the triangulation. If the subcurve ever needs to be split, the vertices located inside its boundary edge's diametral circle will first be removed from the mesh. No vertex located outside the PCC can survive the subcurve split and therefore, no vertex will ever find itself to be outside the triangulation. This bound on TV is greater than the one required by Boivin and Ollivier-Gooch and as such, our algorithm should generate coarser meshes on similar inputs.

2.4.2 Constrained Delaunay Triangulation Construction

The boundary edges obtained during initial discretization form a piecewise linear approximation of the input PCC's boundary. Remember that a CDT matching this approximated boundary is needed to launch the refinement algorithm.

Unfortunately, such a triangulation might not exist at this point. In fact, the current approximation of the PCC might be so coarse that some boundary edges may be intersecting. When this situation occurs, edge flips

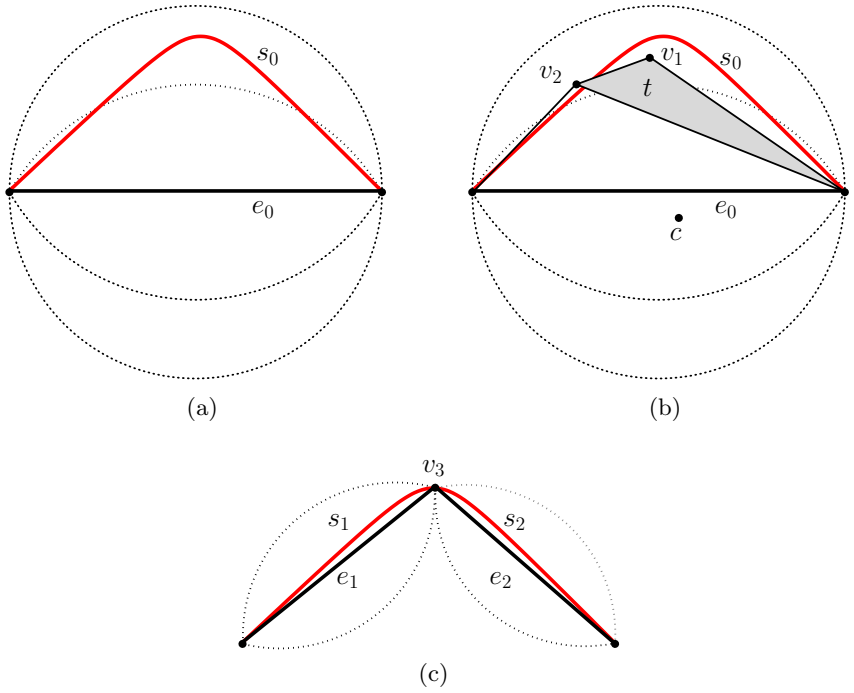


Figure 2.5: (a) Boundary edge e_0 is illustrated with its diametral lens and circle, subcurve s_0 has $\pi/3 \leq TV < \pi/2$ and is not lens protected. (b) Vertex v_1 is allowed to between e_0 and s_0. It is outside the input PCC, but still inside the triangulation. Also, v_2 is arbitrarily close to s_0. Circumcenter c of poorly-shaped triangle t is the next scheduled insertion. (c) c encroaches on e_0, so s_0 must be split: v_3 is inserted on s_0, but only after v_1 and v_2 have been removed from the triangulation.

are powerless at forcing all boundary edges into the triangulation: at some point, a flip to recover a missing edge will inevitably knock a recovered one out of the triangulation. The boundary is then unrecoverable.

Instead of constructing the triangulation from the initial sample, we enrich the boundary sample beforehand, to allow for easier recoverability. Obviously, this is done while remaining aware that the boundary discretization must be kept as coarse as possible. As mentioned in Section 2.2, that all simplices in a constrained Delaunay triangulation must themselves be constrained Delaunay. Consequently, the initial condition required to start our refinement

algorithm is achieved when all boundary edges are constrained Delaunay — not only must all boundary edges appear in the CDT, but they must also be unencroached before any circumcenter insertion is attempted. Every subcurve whose boundary edge does not satisfy this property must be split.

To determine which subcurves to split, boundary edges whose diametral circle are not vertex-free first need to be identified. When a vertex is found inside a diametral circle, its visibility from the corresponding boundary edge must then be assessed. If the vertex is visible, a subcurve split is performed. Visibility testing is done following Algorithm 2.2. Upon termination of the boundary enrichment algorithm, all boundary edges respect the constrained Delaunay property. This process is summarized in Algorithm 2.1.

Vertices located inside diametral circles must be detected as efficiently as possible. Unfortunately, we do not yet have the luxury of a triangulation to act as a search structure. Instead, we rely on a spatial indexing tree τ which indexes the axis-aligned bounding boxes of all boundary edges' circumcircles. For the boundary enrichment to terminate, the input PCC must not contain any small angles. This restriction is lifted in Section 2.5.

Determining if a vertex is visible from a boundary edge is key to the success of Algorithm 2.1. Assume that vertex v is inside boundary edge e's diametral circle. Visibility of v from e is considered to be blocked if either of the following two conditions hold:

1. There is an unmeshed region — a hole in the geometric domain — located between v and e.

2. One, or many, boundary edges are located between v and e, such that it is impossible to define a segment from any point on e to v without intersecting another boundary edge.

To verify v's visibility against Condition 1, e must know which curve of the PCC it approximates (its parent curve). This curve must be able to determine whether v lies on its right side or on its left side and whether or not it bounds an unmeshed region on either side (see Figure 2.6a). The geometric modeler beneath the meshing algorithm must be able to answer these queries correctly.

Algorithm 2.1 Boundary enrichment algorithm

BoundaryEnrichment($X, \mathcal{V}, \mathcal{E}$)
(X is the input PCC. $\mathcal{V}$ is the set of sampling vertices.
$\mathcal{E}$ is the set of boundary edges.)

```
Construct τ from E (spatial index of boundary edges)
Q_v ← empty queue of vertices
for each vertex v ∈ V do
  Enqueue(v, Q_v)
while Q_v is not empty do
  v ← Dequeue(Q_v)
  E' ← intersection of v and τ
  for each boundary edge e ∈ E' do
    if v in diametral circle of e and Visibility(v, e, τ) then
      s ← subcurve of e in X
      SplitSubcurve(s)
      Enqueue(v, Q_v)
return V and E such that all e ∈ E are constrained Delaunay
```

SplitSubcurve(s)

```
v_1, v_2 ← endpoints of s
e ← v_1v_2
v_3 ← new vertex on s such that TV_s|_{v_1}^{v_3} = TV_s|_{v_3}^{v_2}
e_1 ← v_1v_3, e_2 ← v_3v_2
V ← V ∪ v_3
E ← {E ∪ {e_1, e_2}} \ e
add e_1 and e_2 to τ
mark e Deleted and remove from τ
Enqueue(v_3, Q_v)
```

Algorithm 2.2 Vertex visibility from a boundary edge

VISIBILITY(v, e, τ)

Require: v inside diametral circle of e
  (Testing Condition 1)
  $c \leftarrow$ parent curve of e
  $\text{side}_c \leftarrow$ side of c where v is located
  $\text{side}_e \leftarrow$ side of e where v is located
  if side_e is undertermined (v is directly on e) **then**
    return true (the endpoint of a boundary edge is on e)
  else if $\text{side}_e \neq \text{side}_c$ **then**
    return true (e intersects another boundary edge)
  else if c bounds an unmeshed region on side_c **then**
    return false
  (Testing Condition 2)
  $v_1, v_2 \leftarrow$ endpoints of e
  $e_1 \leftarrow \overline{v_1 v}$
  $e_2 \leftarrow \overline{v_2 v}$
  $\text{int}_1 \leftarrow$ **false**
  $\text{int}_2 \leftarrow$ **false**
  $B \leftarrow$ axis-aligned bounding rectangle of $\Delta v v_1 v_2$
  $\mathcal{E}' \leftarrow$ intersection of B and τ
  for each edge $e' \in \mathcal{E}'$ disjoint from v **do**
    if e' intersects e_1 **then**
      $\text{int}_1 \leftarrow$ **true**
    if e' intersects e_1 **then**
      $\text{int}_2 \leftarrow$ **true**
    if int_1 and int_2 **then**
      return false
  return true

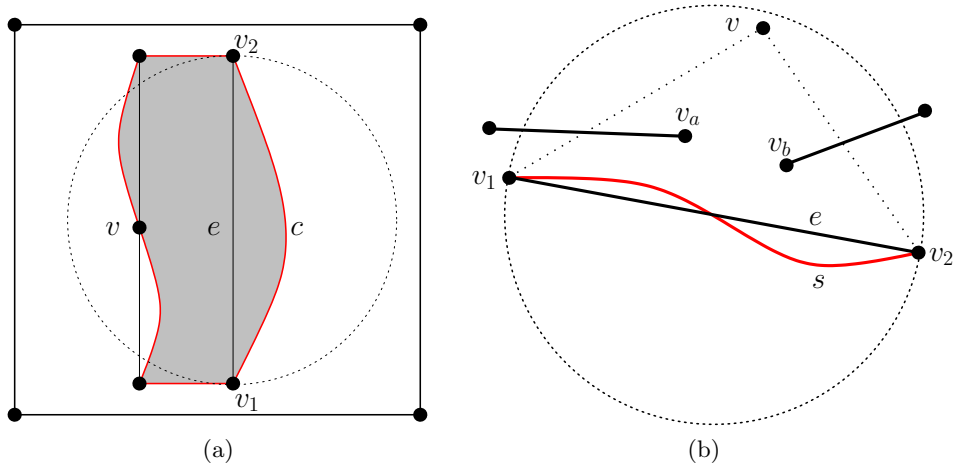


Figure 2.6: Two examples where vertex v is inside boundary edge e's diametral circle, but is not considered visible from e. (a) The interior of the rectangle is to be triangulated, except for the hole drawn in gray. To determine that v is invisible, curve c (the parent of e) must be able to determine that it bounds a hole and that v is located across this hole (b) v is not visible from e's endpoints, v_1 and v_2. This is sufficient to declare v invisible from e, even if v is actually visible from e's interior. This situation can only occur if other vertices, here v_a and v_b, are also inside e's diametral circle. They will eventually force a split of subcurve s; the same end result is achieved.

Note that v is automatically considered visible if it is the endpoint of a boundary edge that intersects e; e's subcurve is therefore split immediately. Boundary edge intersections are detected while carrying out the visibility test for condition 1. Because $TV < \pi/2$ over any subcurve and because curves in the input PCC cannot cross each other, v, the endpoint of a boundary edge intersecting e must be located inside e's diametral circle (see Figure 2.7a). As such, a visibility test is inevitably conducted. As is depicted in Figure 2.7b, if e is indeed intersected by other boundary edges, then v must either be directly on e or it must be located on different sides of e and of e's parent curve. Figure 2.7c shows that there cannot be an intersection of boundary edges if v is on the same side as both e and e's parent curve; this would imply

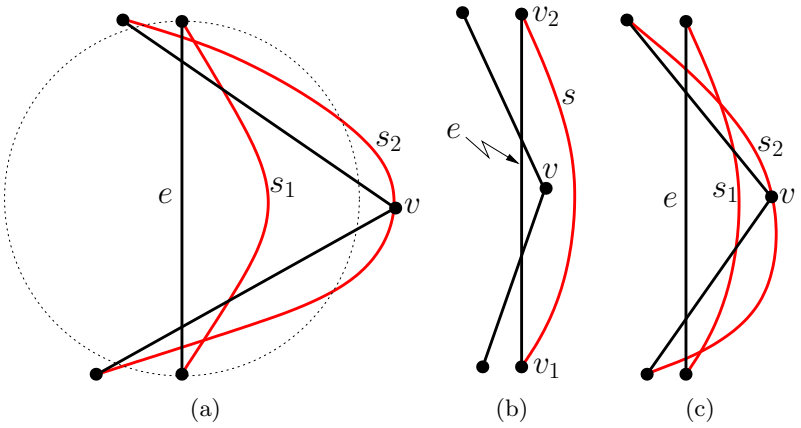


Figure 2.7: Boundary edge intersections. (a) Since $TV_{s_2} < \pi/2$, for v to be outside e's diametral circle, subcurve s_2 would need to cross subcurve s_1, a violation of Definition 2.1. (b) An intersection is detected because v is located on the right side of boundary edge e, but on the left side of its parent subcurve s. (c) A boundary edge intersection where v is on the same side of e and of e's parent subcurve (s_1) would imply an invalid PCC where curves cross each other.

curves crossing each other in the input PCC, a violation of Definition 2.1.

If visibility is not found to be occluded according to Condition 1 and if no boundary edge intersections are detected, Condition 2 is then tested. As can be seen in Figure 2.6b, given vertex v inside boundary edge e's diametral circle, it is sufficient to verify that the line of sight from v to both end vertices of e is blocked by other boundary edges to declare v invisible from e. It is obviously possible for visibility to be blocked along these two lines, but for v to still be visible from e's interior. However, this situation is only achievable if other vertices are located inside e's diametral circle. These vertices will eventually cause the subcurve of e to be split, the same end result as if v had originally been declared visible.

Once all the boundary edges satisfy the constrained Delaunay property, a Delaunay triangulation containing all the sampling vertices is constructed. The boundary edges are non-intersecting and as such form a valid PSLG approximating the input geometry. In these conditions, the boundary of the

PCC is guaranteed to be recoverable using edge flips only [15, Lemma 9]. By removing all triangles located outside the triangulation domain — triangles covering holes in the domain, for instance — a constrained Delaunay triangulation whose boundary edges are all unencroached is obtained. Delaunay refinement can then be run on this triangulation to improve its quality.

2.4.3 Sizing Field

The constrained Delaunay triangulation built in the previous section currently only has boundary vertices. It can thus be used as a search structure to quickly approximate the local feature size (lfs) at these vertices. Knowledge of the lfs over the input then allows definition of a sizing field to automatically control the size of triangles in the output. Ollivier-Gooch and Boivin [9] introduced a sizing field depending on the lfs and on two user-defined parameters. The quality guarantees offered by previous algorithms apply to theirs because the sizing field is a Lipschitz continuous function, as is Ruppert's definition of the lfs. In this section, we modify their approach for a more independent control of boundary and of interior vertex spacing.

The process begins by computing the lfs at the vertices of the CDT. This is a fairly straightforward procedure. Given a boundary vertex v, evaluating $\text{lfs}(v)$ only requires information about the neighborhood of v. Triangles adjacent to v have edges that are either connected to v or are opposing v. $\text{lfs}(v)$ is taken as the minimum of:

- the length of the shortest edge connected to v, or
- the shortest distance from v to a subcurve discretized by an edge opposing v.

Once the lfs is known at every boundary vertex, the same function as the one used in Ollivier-Gooch and Boivin's original scheme allows to compute these vertices' length scale, $LS\,(v)$:

$$\widetilde{LS}\,(v) \;=\; \min_i \left(LS\,(w_i) + \frac{|vw_i|}{G} \right),\ w_i \in \text{neighbors of } v \qquad (2.1)$$

$$LS(v) \quad = \quad \min\left(\frac{\text{lfs}(v)}{R},\, \widetilde{LS}\,(v)\right), \tag{2.2}$$

where $|\cdot|$ is the Euclidean distance, R is a boundary refinement factor and G is a grading constant, controlling how fast triangle size can change over a certain distance. A boundary edge is considered too long and its subcurve is therefore split if its length is greater than the sum of the length scales at its vertices. A triangle is deemed to be too large whenever the ratio of its circumradius to the average length scale at its vertices is greater than $\frac{\sqrt{2}}{2}$.

As opposed to the original scheme, when we insert a new vertex in the interior of the triangulation, its length scale is computed using only Equation 2.1. Therefore, the interior vertex spacing exclusively depends on the grading constant and on the density of the boundary discretization. This can clearly be observed in Figure 2.8. To compute the length scale of a vertex resulting from a subcurve split, we proceed slightly differently. A ceiling value is first set by linear interpolation: using the length scale of the subcurve's vertices, the maximum possible length scale for the new vertex is interpolated along the subcurve's arc length. The length scale is then taken to be the minimum between the interpolated value and the graded length scale, as computed by Equation 2.1.

The new sizing field scheme is simpler and faster than the original. Furthermore, because we do not use the refinement factor for the interior vertices, there is no need for them to store information about nearby boundary entities as is done in the original method. This results in memory savings and simplified bookkeeping.

Triangle sizing can also be manually controlled by the user. In this case, a sizing field is defined using a background mesh or a spatial indexing tree covering all of the domain to be meshed, or part thereof. This approach obviously requires a priori knowledge of the desired length scale, which can for instance be computed from the error indicators of a numerical solution. An example mesh generated with a user-defined length scale and used to solve Euler's equation is shown in Figure 2.9.

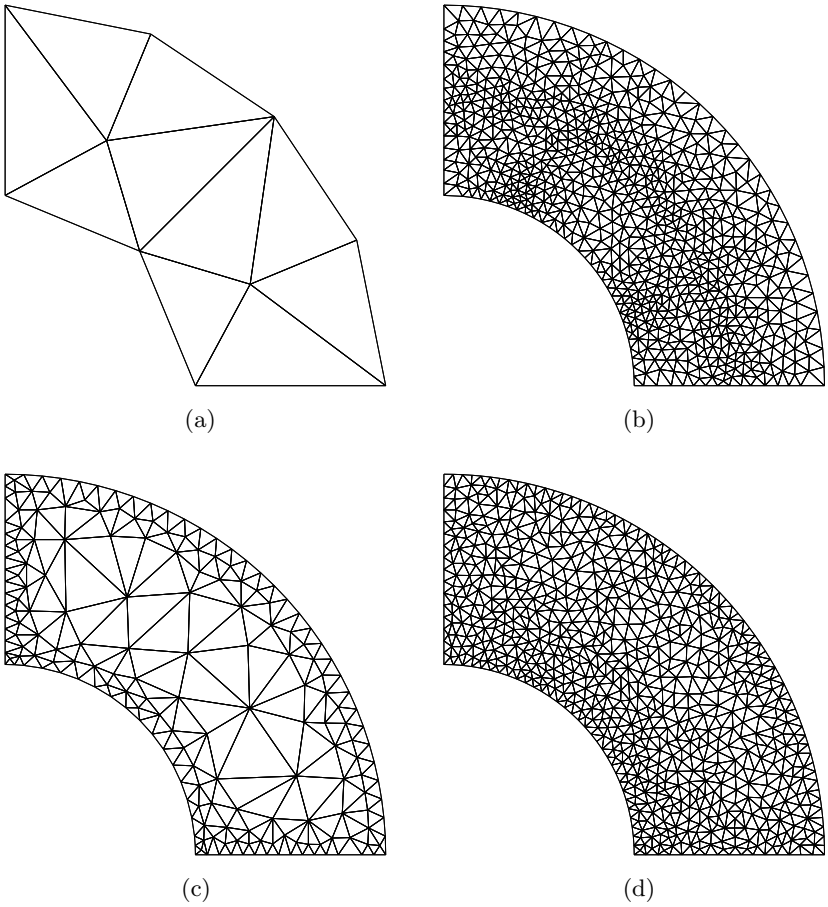


Figure 2.8: Effects of the sizing field on the triangulation of an annulus. (a) Coarsest possible mesh, $R = G = 1$. (b) Original sizing field, $R = 10$, $G = 1$. Note that smaller triangles are observed along the center of the channel. (c) New sizing field, $R = 10$, $G = 1$. The refinement factor only affects the region close to the boundary. (d) New sizing field, $R = 10$, $G = 10$. Increasing the grading factor increases mesh density in the interior.

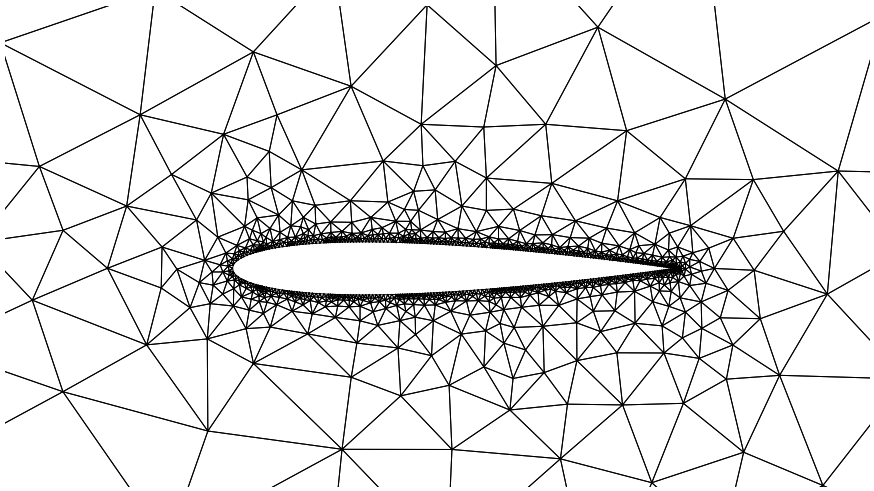


Figure 2.9: Close-up of a mesh around a NACA-0012 airfoil, generated using a user-defined sizing field. Density is very high close to the airfoil surface and decreases quickly moving towards the farfield.

2.4.4 Triangulation Refinement

The CDT constructed in Section 2.4.2 is not yet suitable for numerical simulations. The triangles are likely to be too big to properly capture solution features, or are of such poor quality that conducting an efficient and accurate simulation is impossible.

At this point, the CDT can be improved by Shewchuk's Delaunay refinement algorithm. A triangle that is too large with respect to the sizing field or that does not achieve the quality target, as evaluated by its circumradius-to-shortest-edge ratio, will be attacked by the algorithm: an attempt at inserting a new vertex at its circumcenter will be made. Although the order in which the algorithm targets triangles does not affect the quality guarantees, Chew notes that always splitting the worst quality triangle first leads to more visually appealing meshes [4]. More importantly, Shewchuk also shows that this strategy generates meshes containing fewer triangles [17]. Based on these observations, we schedule circumcenter insertions with a priority queue. All insertion operations (interior and boundary) are implemented on a Bowyer-Watson framework [3, 20], coupled with Shewchuk's robust

adaptive precision geometric predicates [15].

Prior to launching the refinement algorithm, elements of the CDT that need to be split are identified and queued. Subcurves that are too long are inserted in a double-ended queue (remember that by using Algorithm 2.1, all the subcurves in the initial CDT are initially unencroached). Triangles that are too large or that do not meet the quality criterion are inserted in the priority queue: large triangles are given higher priorities than poor quality ones. Subcurve splits always have priority over circumcenter insertions; no triangle is popped from the priority queue unless the subcurve queue is empty. The algorithm terminates when both queues are empty. Note that we refine the boundary for size first. Experience has shown that proceeding this way leads to a better evaluation of the sizing field by Equations 2.1 and 2.2, especially for interior vertices.

The fact that the meshed domain has curved boundaries adds a few complications to the refinement algorithm. Of course, there is the matter of determining the split location for subcurves. The midpoint splits typically used for straight segments does not apply. As was mentioned earlier in Algorithm 2.1, the subcurve split point is chosen such that TV is equal over the two new subcurves.

We identify two special insertion cases directly resulting from the presence of curved boundaries. The first one is a direct consequence of the fact that it is still possible for the convex hull of two or more subcurves to intersect each other. In this condition, despite our efforts to eliminate intersecting boundary edges prior to building the CDT, a subcurve split can still create new intersections, as is illustrated in Figure 2.10. Such a split cannot be correctly handled by the Bowyer-Watson algorithm and will result in failure of the whole algorithm. One obvious solution would be to modify Algorithm 2.1 to require, a priori, that no two subcurve convex hulls intersect. However, our experiments demonstrate that in many cases, this strategy results in an unnecessarily dense boundary discretization. Instead, we prefer to resolve boundary intersections on an as needed basis, especially since they are easily detectable and avoidable. Prior to inserting a new vertex v to split a subcurve approximated by boundary edge e_1, a depth-first search over the

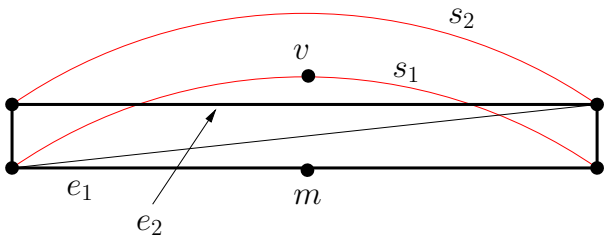


Figure 2.10: Boundary edge intersections will be created if subcurve s_1 is split before subcurve s_2. The potential intersection can be detected by acknowledging that the midpoint m of boundary edge e_1 and the split vertex v of s_1 are on different sides of e_2.

mesh is performed, starting from e_1, to identify all triangles containing v in their circumcircle. Every time the walk hits another boundary edge, e_2, a verification is made to determine if inserting v will create an intersection. If the midpoint m of e_1 and the vertex v are both located on the same side of e_2, then the subcurve can be safely split. If however the points are on different sides of e_2, splitting the subcurve will result in an intersection. In this case, e_2 is pushed to the front of the subcurve queue — this is why a double-ended queue is used — so that it is split before e_1. The process is repeated until all scheduled subcurve splits can be performed.

The second special insertion happens when a subcurve split vertex, v, is located outside all circumcircles of the CDT. Figure 2.11 shows an example where this situation happens. Because v is not inside any circumcircle, no triangle will be deleted from the mesh as a result of its insertion. Instead, one new triangle is attached "outside" the triangulation. One of the new triangle's edges, the boundary edge of the subcurve getting split, becomes an interior edge. Its two other edges are the new boundary edges created by the split.

Once the refinement algorithm terminates, all triangles in the mesh have a circumradius-to-shortest-edge ratio smaller than a prescribed value. In practice, Shewchuk's algorithm terminates and outputs nicely graded meshes with a radius-edge ratio as low as 1, although the provable value is slightly higher. As presented in Appendix A, our algorithm offers the same

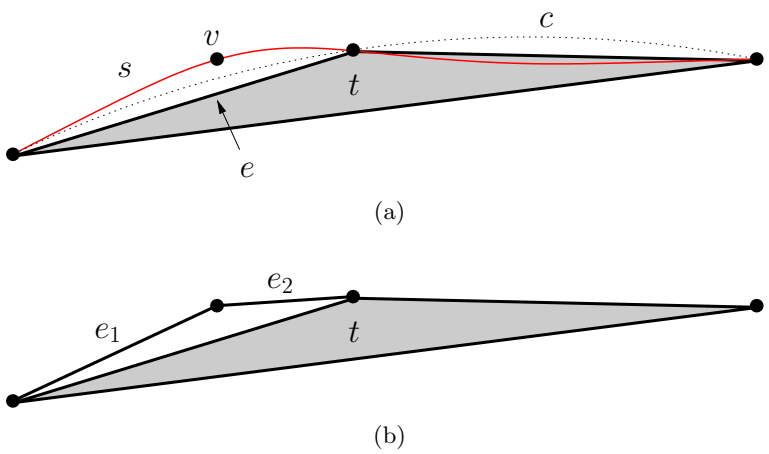


Figure 2.11: (a) Vertex v to split subcurve s is outside the circumcircle c of boundary triangle t. (b) Inserting v creates a new triangle outside the existing triangulation. Boundary edge e becomes an interior edge when boundary edges e_1 and e_2 are created.

termination and quality guarantees. All the meshes presented herein are obtained using this same value. This result only holds if no small angles are present in the input PCC. The complete algorithm is analyzed in Appendix A. The method used to handle small input angles is presented in Section 2.5.

2.4.5 Triangulation Post-processing

It is possible that some boundary edges intersect after applying the Delaunay refinement algorithm. Fortunately, this only occurs for contrived geometries, such as the one illustrated in Figure 2.12. Nonetheless, these boundary intersections must be fixed in a post-processing operation.

The intersections are a direct result of using the Bowyer-Watson insertion scheme. The new vertex v inserted to perform the split of subcurve s can possibly be located across a boundary edge which is separated from s by an unmeshed region. Again, a depth-first search over the mesh, starting from the boundary edge of s, is performed to identify the triangles containing v inside their circumcircle. The "walk" is local, typically covers a handful of triangles in the neighborhood of s, but never crosses an unmeshed region.

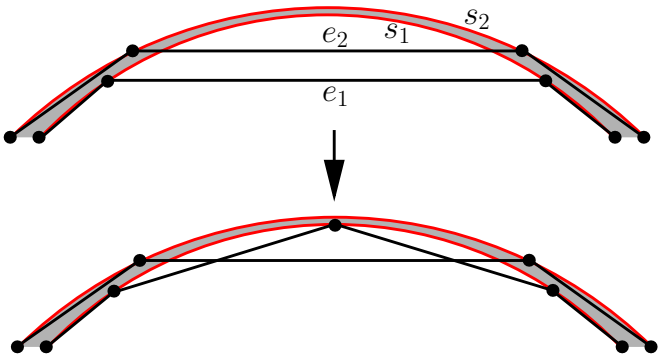


Figure 2.12: Boundary intersection appearing after running Delaunay refinement. A split of s_1 does not warrant a split of s_2 because of an unmeshed region (drawn in gray) located between both subcurves.

As a consequence, intersections such as the one in Figure 2.12 cannot be detected.

Once Delaunay refinement terminates, a tree indexing the boundary edges can be used to efficiently identify possible intersections. All subcurves whose boundary edge is found to be intersected are split. Inserting new boundary vertices can potentially create triangles not satisfying the quality bound. Therefore, Delaunay refinement must be applied to eliminate these poor triangles. If quality refinement leads to new intersections, the process is repeated until no boundary intersections or any poor quality triangles remain in the mesh.

2.5 Handling Small Input Angles

Delaunay refinement algorithms are notorious for failing to terminate when the input geometry contains small angles. Figure 2.13 illustrates one of the failure mechanisms, where successive attempts to fix boundary edge encroachment leads to cascading refinement. A few more failure examples are given by Pav [12]. Small angles act as edge length reducers, creating edges that get shorter and shorter as the algorithm tries to eliminate poor quality triangles adjoining them. The argument that vertex spacing remains

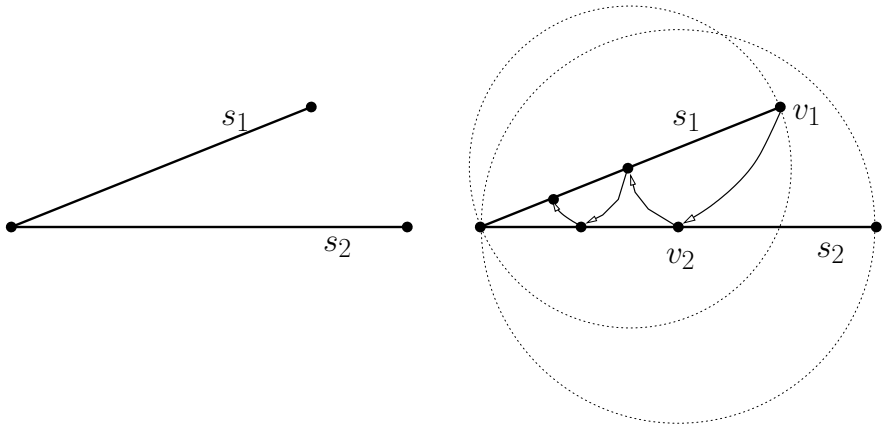


Figure 2.13: An input causing failure of the Delaunay refinement algorithm (using diametral circle encroachment for simplicity). (a) Two adjoining linear subcurves of different lengths, s_1 and s_2, form a small angle. (b) The endpoint of s_1 is contained in the diametral circle of s_2, so vertex v_1 is inserted. v_1 is now in s_2's circumcircle, so v_2 is inserted. The process cascades and leads to the creation of arbitrarily short edges.

within a constant factor of optimal no longer holds in the neighborhood of small angles and as such, termination can no longer be guaranteed.

To "block" runaway encroachment splits, Ruppert originally suggested modifying how boundary edges are split around small angles [14]. Instead of inserting at the edge's midpoint, new vertices are placed at the intersection of the edge and concentric shells centered at the apex of small angles (see Figure 2.14a). Miller et al. [8] show that with a simple alteration to the concentric shell method, termination can be guaranteed for domains whose boundaries are piecewise linear. Given a triangulation with small angles protected by concentric shells, their algorithm skips attempts to remove a bad quality triangle whose edge opposite its smallest angle spans a small angle in the input.

The termination proof provided by Miller et al. relies on the assumption that the length ratios between any two adjoining input segments is a power

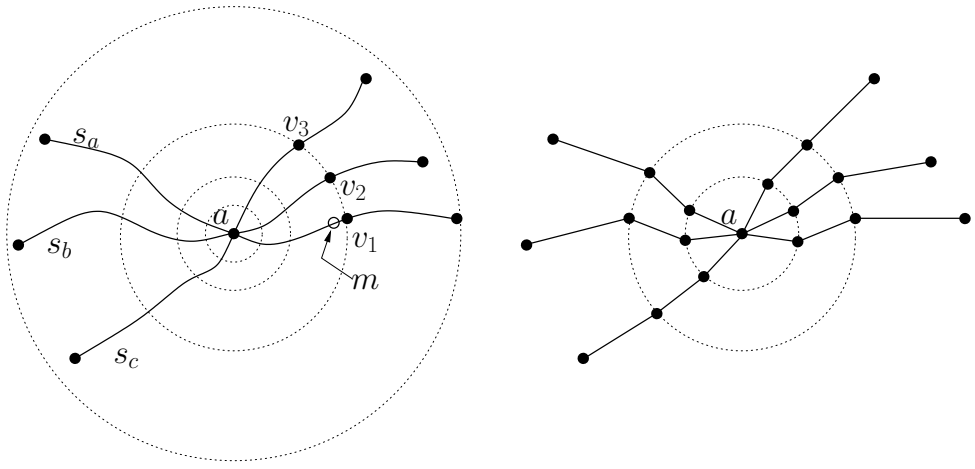


Figure 2.14: (a) Concentric shell splitting applied to curves. When subcurves sharing a vertex a at the apex of a small angle need to be split, the new vertices (v_1 to v_3) are inserted at the intersection between the subcurves and concentric shells centered at a, not at the mid-TV point, m. The shells' radii are powers of two. (b) Typically, concentric shell splitting of subcurves does not yield adjoining boundary edges whose length ratio is a power of two.

of two. As is shown in Figure 2.14b, this cannot be satisfied, in general, for curved inputs. Therefore, their proof cannot be extended directly. Nevertheless, we argue that the method still yields a terminating algorithm when applied to curved inputs. When boundary edges get shorter around a small angle, the subcurves they approximate will increasingly resemble straight segments. In the limit, the subcurves are straight and their boundary edges can therefore satisfy the power of two requirement of the proof. Interestingly, the proven size optimality constant is on the order of 10^7 for a small angle of 1° [12]; the algorithm will not generate an edge shorter than lfs/10^7. Subcurve can probably be treated as straight segments long before they become this short. In practice, both the original algorithm and our adaption outperform this theoretical bound.

Small angles must be protected before executing Algorithm 2.1, otherwise it will fail to terminate. Any input angle of less than 60° is considered small

and requires protection. Let a *small angle complex* be a set of subcurves, all having a common apex vertex as an endpoint and forming contiguous small angles at this apex vertex. For example, subcurves s_a, s_b and s_c of Figure 2.14a form a small angle complex. Note that a single apex vertex can harbor many small angle complexes.

The small angle protection scheme is applied immediately after the curves' initial discretization. It starts by identifying all small angle complexes. Subcurves appearing in two different complexes are bounded by two apex vertices. They are first split at their regular split point. Then, given a small angle complex Σ, define the maximum split radius $r_{\max}$ to be two thirds of the length of the shortest boundary edge, e, approximating a subcurve of Σ. The biggest power of two less than $r_{\max}$ is chosen to be the radius of the shell onto which the new vertices will be placed:

$$k = \left\lfloor \frac{\ln(r_{\max})}{\ln(2)} \right\rfloor$$
$$r_{\text{shell}} = 2^k$$

This ensures a split between 1/3 and 2/3 of the length of e. Σ can then be protected by splitting all of its subcurves at their intersections with a circle of radius r_{shell} centered at the apex vertex. All subsequent splits of a subcurve that is part of Σ and connected to the apex vertex is done using the next smaller power of two shell.

At this point, to extend the algorithm of Miller et al. to curve inputs, we only need to flag triangles in the neighborhood of small angle complexes as "untouchable"; Delaunay refinement will never attempt to insert at these triangles' circumcenters, even though they do not meet quality targets or are too big with respect to the sizing field. As such, our algorithm never tries to remove a poor quality triangle if the edge opposing its smallest angle is bounded by two vertices on a shell. This is shown in Figure 2.15.

Although we cannot extend the termination proof of this protection strategy to curved inputs, its practical performance is superior to Ruppert's method originally implemented by Boivin and Ollivier-Gooch [2]. In our experiments, mesh refinement never failed in the presence of small input

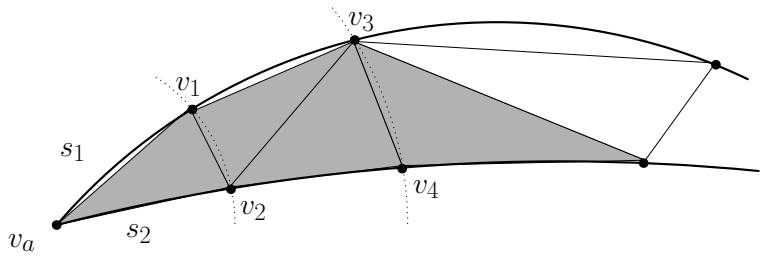


Figure 2.15: Small angle protection. Subcurves s_1 and s_2 are separated by a small angle. Vertices v_1 through v_4 are on concentric shells centered at a. An insertion to improve the shaded triangles' quality will never be attempted.

angles.

2.6 Results

This section presents meshes generated by our algorithm. Unless otherwise stated, all the meshes are generated using $TV = \frac{\pi}{2}$ for the initial discretization and with an angle bound of 30°.

In Figure 2.16, we present a comparison of our algorithm with results previously published by Boivin and Ollivier-Gooch [2] to illustrate how we are able to generate coarser meshes on a similar geometry. For the comparison to be as fair as possible, the sizing field was disabled; triangles were refined solely based on quality. By running our algorithm with $TV = \frac{\pi}{2}$, we obtain a triangulation with 40% fewer vertices. When initial sampling is executed with $TV = \frac{\pi}{6}$, the same bound on TV required by the original algorithm, we still generate a coarser mesh. The difference is partly due to the boundary recovery procedure. In the regions where two airfoil elements are very close together, our new boundary enrichment algorithm inserts fewer vertices than the original strategy based on stitching and flipping.

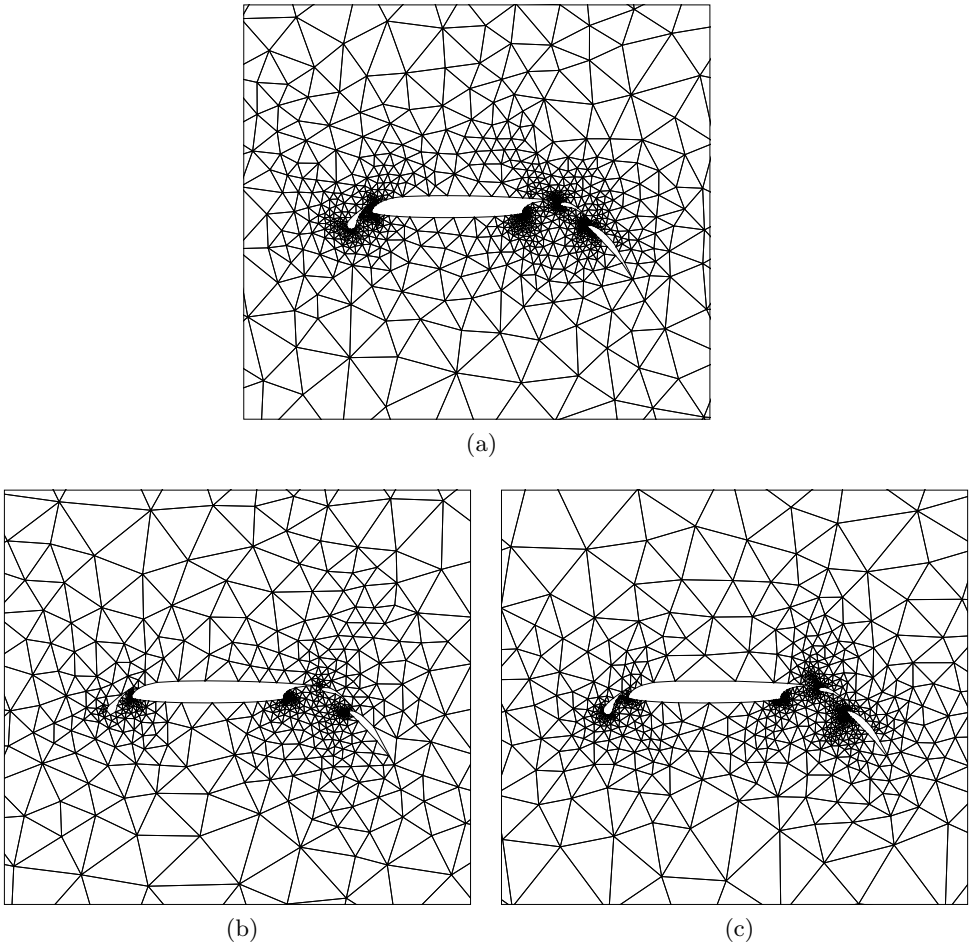
Boundary enrichment is generally very efficient: in most cases, it requires less than 5% of the total algorithm's execution time. Obviously, this number is strongly dependent on the ratio of initial boundary vertices to the number of vertices inserted during refinement. For instance, boundary enrichment

makes up about 25% of the — very short — execution time needed for the generation of a mesh such as the one in Figure 2.8c. This ratio is much lower for meshes generated using a sizing field defined with high values of R and G.

Figure 2.17 shows the mesh of a geometry made of multiple disjoint regions. The boundary is a collection of splines and polylines. The mesh illustrates the benefits of using constrained Delaunay triangulation to enforce boundary conformality. For this domain, vertices located in a given region cannot be visible from vertices in all other regions. The careful observer will notice that many boundary edges contain at least one vertex in their diametral circle which did not get split because they are separated by unmeshed regions. The mesh therefore remains coarser than if all edges were forced to be strongly Delaunay. These long boundary edges remaining in the final mesh demonstrate the effectiveness of the visibility algorithm (Algorithm 2.2) used during the boundary enrichment process.

Figure 2.18 and 2.19 show two meshes inside of a domain bounded by multiple splines, generated with different levels of refinement. Again, it is possible to observe that a few boundary edges containing at least one vertex in their circumcircle remain after refinement. As opposed to the mesh of Figure 2.17, the triangulation covers a single region. This implies that the visibility algorithm correctly detects holes and gaps in the domain. By comparing the histograms of angle distribution in Figures 2.18b and 2.19b, it is possible to see that more angles are clustered around 60° in the finer mesh. This indicates that quality and cardinality are two opposing problems. In other words, the more vertices in a mesh, the easier it is to obtain high-quality triangles.

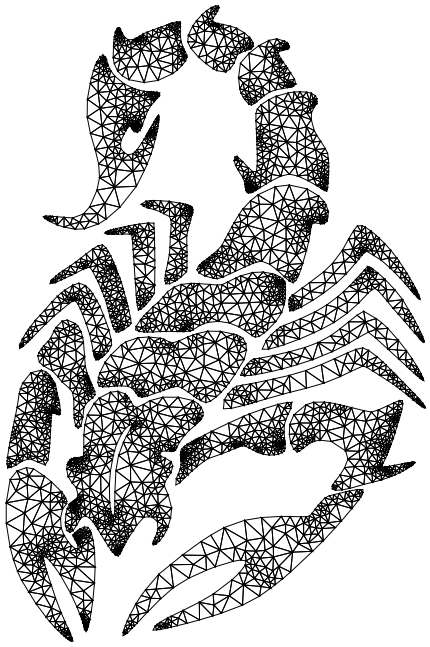
Finally, in Figure 2.20, the mesh of a domain containing a large number of small angles is depicted. This test case is specifically designed to demonstrate that our adaptation of the method by Miller et al. [8] to curved inputs works well in practice, despite lacking a strong proof of termination. In this example, as well as many more not presented herein, not only do we achieve termination, but the maximal angle remains under 137°, which is consistent with the theoretical bound proven by Miller et al. The quality histogram



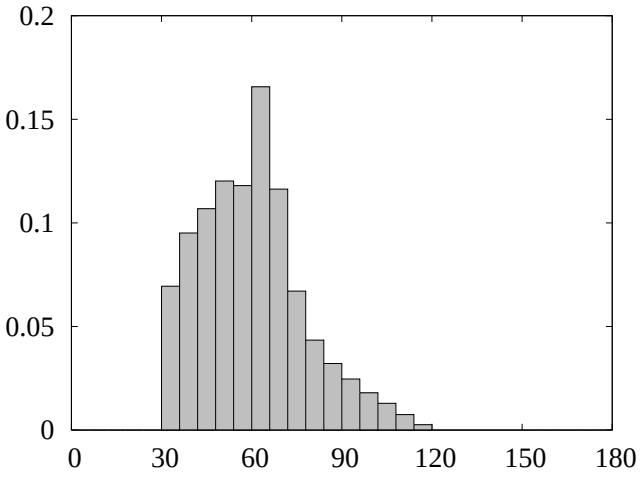
(a)

(b) (c)

Figure 2.16: Close-up of meshes around a multi-element airfoil. All meshes have a minimum angle of 30° and are generated without a sizing field (a) Algorithm of Boivin and Ollivier-Gooch. 1341 vertices and 2506 triangles (b) Our algorithm, initial sampling with $TV = \frac{\pi}{2}$. 775 vertices and 1440 triangles. (c) Our algorithm, initial sampling with $TV = \frac{\pi}{6}$. 1117 vertices and 2068 triangles.

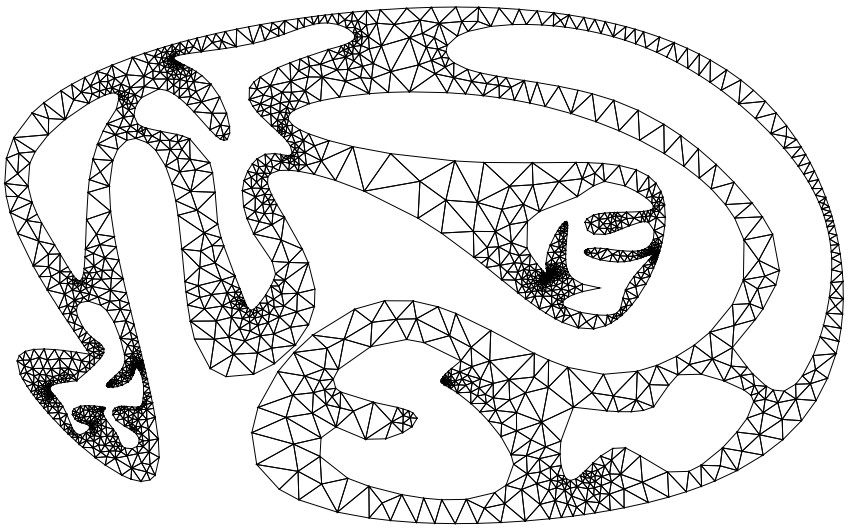


(a)

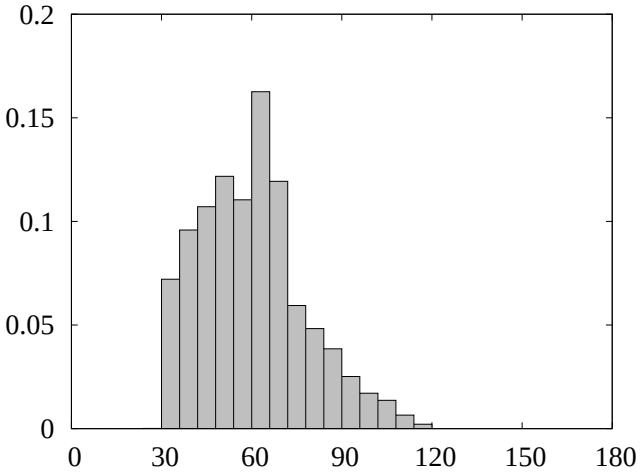


(b)

Figure 2.17: (a) Mesh of a scorpion obtained with $R = G = 1$. 3750 vertices and 5855 triangles. (b) Histogram of the angle distribution in the mesh. All angles are between 30° and 120°.

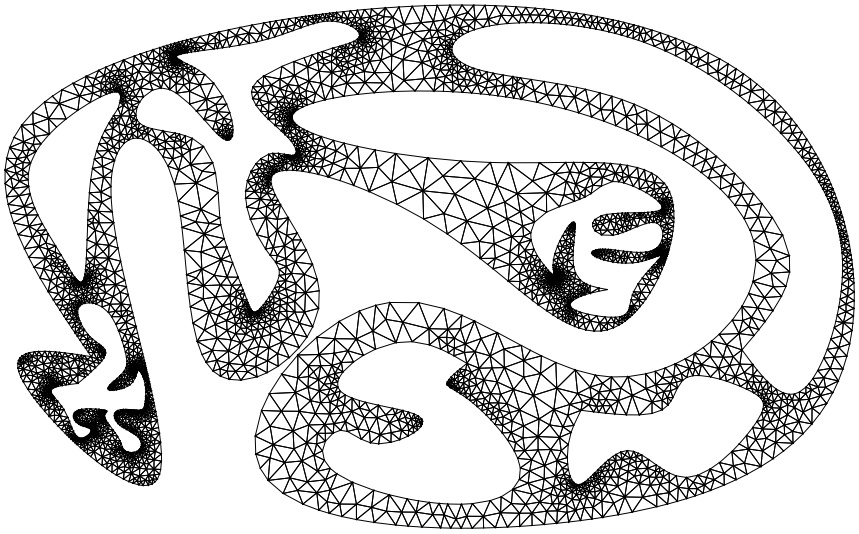


(a)

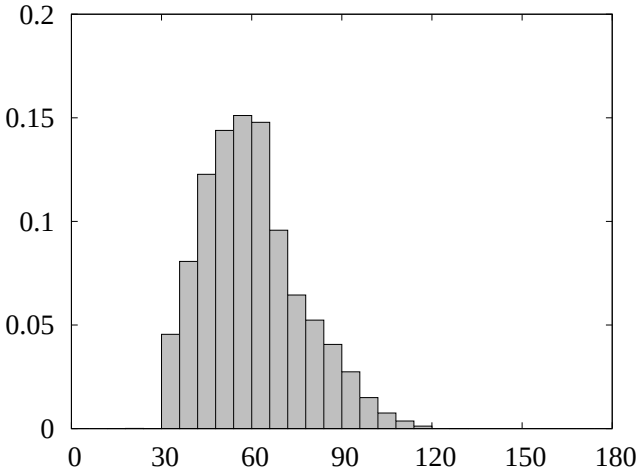


(b)

Figure 2.18: (a) Coarse mesh of a multi-spline domain, where the sizing field is computed with $R = G = 1$. The mesh contains 2131 vertices and 3315 triangles. (b) Histogram of the angle distribution in the mesh. The minimum angle is 24° due to the presence of small angles in the input. The maximum angle is 120°.



(a)



(b)

Figure 2.19: (a) Fine mesh of a multi-spline domain, where the sizing field is computed with $R = 2$, $G = 5$. The mesh contains 4686 vertices and 7741 triangles. (b) Histogram of the angle distribution in the mesh. The minimum angle is 17° due to the presence of small angles in the input. The maximum angle is 130°.

also demonstrates that most angles in the final mesh are between 30° and 120°. Smaller and larger angles left in the triangulation are located in the neighborhood of small input angles.

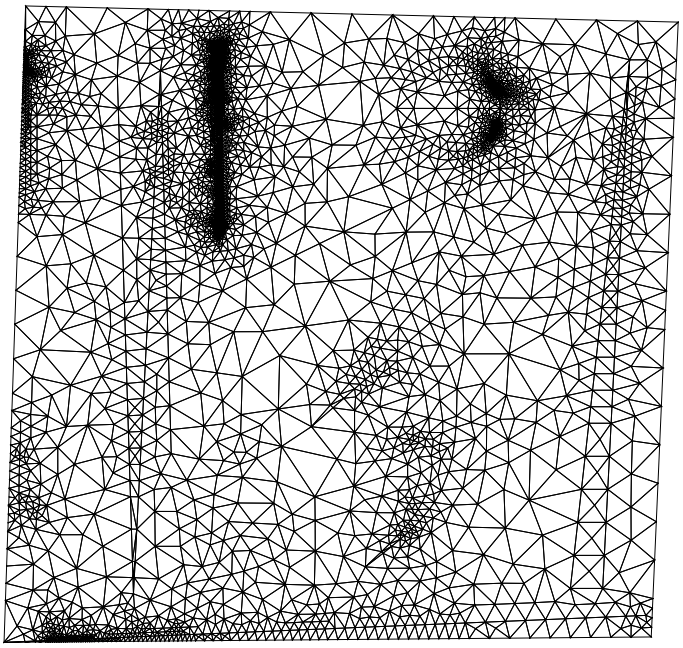
A final note on observed wall clock time. Our implementation of the Bowyer-Watson algorithm inserts about 25000 interior vertices per second when run on a single core of an Intel Q6600 processor. Boundary insertions are very slightly slower, notably because computing the insertion location on a curve is a bit more expensive and because more data manipulations are necessary.

2.7 Conclusions

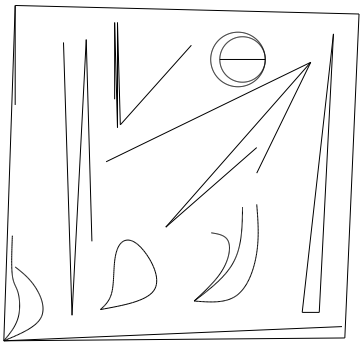
We have presented a mesh generation algorithm based on the Delaunay refinement paradigm. The algorithm accepts input domains whose boundaries are curved and outputs constrained Delaunay triangulations (CDT) of these domains. All triangles in the output meshes have bounded circumradius-to-shortest-edge ratio and as such satisfy a minimum angle bound. In practice, the algorithm can be run with an angle bound of 30°, although the provable bound is slightly lower at about 25.7°. These results are achievable as long as no small angles are present in the input.

Our algorithm requires a Delaunay triangulation constrained to the input as a starting condition. To construct this triangulation, the curves must first be approximated with straight segments. The initial curve sampling is based on the total variation of the tangent angle. We were able to show that lens protection, as required by a previous algorithm [2], is not necessary. As a result, the curves can be approximated with fewer segments, which inevitably leads to coarser triangulations, as demonstrated by our results.

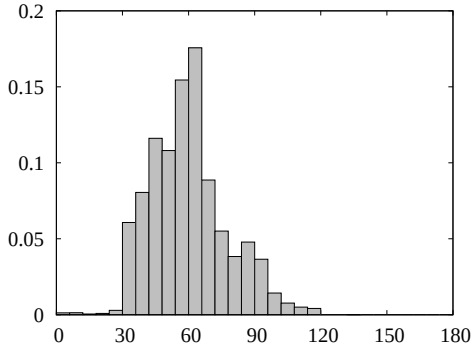
Due to the coarseness of the initial discretization, it is possible for the boundary to be unrecoverable. We avoid this pitfall by using a new strategy which consists of enriching the boundary discretization until all of its segments respect the constrained Delaunay property. Once this is done, the boundary can be recovered without resorting to stitching, but rather by using edge flips only.



(a)



(b)



(c)

Figure 2.20: Triangulation of a geometry containing multiple small input angles. (a) The mesh contains 4452 vertices and 8685 triangles. The minimum and maximum angles are 0.5° and 135° respectively. (b) The input domain. (c) Histogram of the angle distribution in the mesh.

The initial CDT can be used to evaluate the local feature size (lfs) over the meshed domain. A sizing field can then be defined to allow control over the size of triangles output by the meshing algorithm. We slightly modified an existing technique such that it creates a sizing field that can be computed with less computational effort and at reduced memory cost. The new method offers a better decoupling between vertex density in the interior and on the boundary of the mesh. It also avoids a higher vertex density in the middle of channels.

All quality guarantees offered by our algorithm assume that no small angles are present in the input. To avoid the failure of the mesh generation process due to these small angles, we adapt a known protection strategy to curved inputs [8]. We cannot provide a bona fide proof that the extended method produces a working algorithm due to its failure to, a priori, satisfy an assumption of the original strategy. We can however argue that with sufficient refinement in the neighborhood of small angles, the assumption will be satisfied and as such, our adapted protection scheme should work. This has been demonstrated at least experimentally.

This work must be seen as a stepping stone to applying Delaunay refinement to three-dimensional domains bounded by piecewise smooth surfaces. Given a geometric model, the biggest challenge will be to obtain a quality surface triangulation faithfully representing the model's topology and geometry. Building a Delaunay tetrahedralization constrained to the surface triangulation will also prove challenging since the existence of this tetrahedralization is not guaranteed for a given set of constraining edges and facets. Therefore, a boundary enrichment strategy, such as the one used herein, might prove insufficient to obtain a recoverable boundary. An improved strategy will need to be devised.

2.8 Bibliography

[1] Ivo Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.

[2] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1203, 2002.

[3] Adrian Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[4] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280, 1993.

[5] Herbert Edelsbrunner and Tiow Seng Tan. An upper bound for conforming Delaunay triangulations. *Discrete and Computational Geometry*, 10(1):197–213, 1993.

[6] Lori A. Freitag and Carl Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *International Journal of Computational Geometry and Applications*, 10(4):361–382, 2000.

[7] Isaac Fried. Condition of finite element matrices generated from nonuniform meshes. *AIAA Journal*, 10:219–221, 1972.

[8] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the Twelfth International Meshing Roundtable*, pages 91–102, 2003.

[9] Carl Olliver-Gooch and Charles Boivin. Guaranteed-quality simplicial mesh generation with cell size and grading control. *Engineering with Computers*, 17(3):269–286, 2001.

[10] Carl Ollivier-Gooch. *GRUMMP Version 0.3.0 User's Guide*. Department of Mechanical Engineering, The University of British Columbia, 2005. Available from http://tetra.mech.ubc.ca/GRUMMP/.

[11] Carl Ollivier-Gooch, Amir Nejat, and Christopher Michalak. On obtaining and verifying high-order finite-volume solutions to the Euler equations on unstructured meshes. *American Institute of Aeronautics and Astronautics Journal*. To appear.

[12] Steven E. Pav. *Delaunay Refinement Algorithms*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 2003.

[13] Steven E. Pav and Noel J. Walkington. Delaunay refinement by corner lopping. In *Proceedings of the Fourteenth International Meshing Roundtable*, pages 165–181, 2005.

[14] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[15] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.

[16] Jonathan R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the Eleventh Meshing Roundtable*, pages 193–204, 2002.

[17] Jonathan R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):27–74, 2002.

[18] Jonathan R. Shewchuk. What is a good linear element? Interpolation, conditioning, anisotropy, and quality measures. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 115–126, September 2002.

[19] Timothy J. Tautges. The common geometry module (CGM). Technical Report SAND2004-6252, Sandia National Laboratories, December 2004.

[20] David F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

Chapter 3

Constrained Delaunay Tetrahedralization Construction *

3.1 Introduction

Many applications in the field of scientific computing require the generation of a mesh whose boundaries respect those of a complex geometric domain. This is notably the case when one wants to compute the numerical solution to boundary value problems, commonly encountered when simulating physical phenomena such as fluid flows, heat transfer or solid deformations. Indeed, these numerical simulations, usually based on finite-element or finite-volume spatial discretization schemes, require a priori decomposition of the computational domain into simpler entities, for example tetrahedra. To correctly enforce the problem's boundary conditions, these entities must generally match the domain's boundaries.

While many unstructured meshing tools allow for complex geometries to be discretized automatically, generating meshes that are well-suited for numerical simulations can be quite challenging. It is a well-known fact that these meshes must exhibit good quality properties; tetrahedral meshes containing small or large dihedral angles can be at the root of inefficient and inaccurate simulations [13, 19, 34]. These requirements from numerical solvers have motivated the recent development of mesh generation [20, 24, 32]

*A version of this chapter will be submitted for publication. Gosselin, S. and Ollivier-Gooch, C. Constructing Constrained Delaunay Tetrahedralizations of Volumes Bounded by Piecewise Smooth Surfaces.

and mesh improvement [1, 12, 17] algorithms whose objective are to produce tetrahedral meshes devoid of poor quality elements. Among these are the algorithms based on the Delaunay refinement paradigm, which have received much attention since having been introduced by Ruppert [31] and Chew [11]. Extension to three dimensions soon followed: Shewchuk's seminal work extends Delaunay refinement's application to tetrahedral mesh generation. His algorithm is capable of producing boundary conforming, provably-good meshes of domains bounded by planar surfaces [32].

There are typically two approaches used to enforce boundary conformity when generating meshes by Delaunay refinement: conforming Delaunay tetrahedralizations and constrained Delaunay tetrahedralizations. In general, the latter — which we denote CDT throughout this article — requires fewer vertices to achieve conformity. Although its tetrahedra not necessarily respect the empty sphere property, a CDT still retains many of the Delaunay tetrahedralization's favorable properties. For instance, it favors rounder tetrahedra over high aspect ratio ones, something that is desirable if, for example, the mesh is destined to be used with interpolation schemes [33]. The fact that CDTs usually lead to meshes of smaller size is particularly interesting in the context of adaptive refinement. Adaptively refining a coarse mesh is much simpler than using a fine mesh as a starting point and then having to coarsen it. More complete definitions of these concepts are presented in Section 3.2.

Cavalanti and Mello [7], Shewchuk [33] and Si and Gärtner [35] all proposed methods to build CDTs from input domains commonly known as Piecewise Linear Complexes (PLC) [23]. In all cases, a set of constraining triangles is required to launch the algorithm. These can either be created by triangulating the planar facets of the PLC, or can be directly given as input to the algorithm in the form of a surface triangulation of a model. The latter approach is generally used when meshing a piecewise planar approximation of a model whose boundaries are curved. There are notable inconveniences to building a CDT from a surface triangulation. If the input triangulation contains poorly-shaped triangles, these will also appear in the CDT. In this context, the CDT can be seen as a slave to the surface triangulation since

unnecessarily small features and skinny triangles will be carried throughout the mesh generation process. Obviously, if a good quality triangulation of the model's surface is available, this becomes a non-issue. The use of a piecewise planar approximation also causes exact geometric information about the curved surfaces to be lost. This is of particular concern if the mesh is destined to be used in high-order accurate numerical simulations. For example, when using a finite-volume scheme, to obtain a truly higher-order accurate solution, the boundary flux integration must be carried out on the curved boundary itself, not on its planar approximation [26].

In this paper, we propose an algorithm to generate CDTs directly from models bounded by piecewise smooth surfaces, without requiring any a priori discretization. The CDTs thus obtained can later be used to initiate Shewchuk's algorithm, to generate quality tetrahedralizations. As is the case for geometries bounded by planar facets, a set of constraining triangles approximating the curved surfaces is required to construct the CDT. Many options are available to construct the constraining triangles. A popular approach is to generate a triangular mesh in parametric space — if a parametrization is available — and then project it back on the three-dimensional surface. If it is not done carefully, this procedure can easily lead to poorly-shaped or even inverted triangles once the projection is performed. We argue that better control over the triangles' quality and size is possible when the mesh is created in direct space. Although this approach requires the use of more complicated and potentially less robust three-dimensional meshing algorithms, it is more versatile and circumvents some of the problems encountered when meshing in parametric space.

Triangulating a curved surface in direct space by Delaunay refinement can be tricky, mostly because of the challenges associated with correct topology preservation. The problem can be seen as the opposite of surface reconstruction, where one wants to recover a given surface's geometry given a known point sample [2]. Borrowing ideas from reconstruction theory, Boissonnat and Oudot proposed an algorithm to sample and mesh smooth surfaces that guarantees topology recovery [3]. They later extended their algorithm to mesh models containing sharp features, but devoid of small

input dihedral angles [4]. The surface mesh thus generated is a subset of the Delaunay tetrahedralization (DT) of the vertex sample. In other words, the faces of the DT best representing the model's surfaces are chosen to form the desired surface triangulation. As such, once a smooth surface is correctly triangulated, a conforming DT of the volume it bounds is readily available and can then be refined to obtain a quality mesh [27]. This last algorithm was later extended by Rineau and Yvinec to generate tetrahedral meshes of geometries bounded by piecewise smooth surfaces, albeit without input dihedral angles of less than 90° [30].

The aforementioned algorithms all use the medial axis transform to compute the local feature size at any location on the surface, which ultimately controls the density of the mesh. Although Rineau and Yvinec do not explicitly require the computation of the local feature size, their algorithm still demands a sizing field that must be smaller or equal to the local feature size at any point over the input domain. When the triangles become small enough with respect to the local feature size, the piecewise smooth surface and its triangulation are guaranteed to be homeomorphic. The procedure can however result in unnecessary over refinement for some specific geometries. A particularly striking example is that of two rectangular surfaces parallel to each other, separated by a small distance. Although each surface would be correctly discretized with only two triangles, the surfaces' proximity implies a small local feature size. Consequently, more triangles than what is really needed will be generated. Nevertheless, an important insight can be gained from Boissonat and Oudot's main result: inserting enough points on the surface will eventually allow the model's topology to be recovered correctly.

This observation is exploited by Cheng et al. [10]. Their algorithm is conceptually very similar to that of Boissonat and Oudot, but avoids the difficulties and cost associated with the medial axis computation. Instead of driving the refinement using the local feature size, they instead define a series of tests to identify topological violations. They then proceed to eliminate these violations by judiciously inserting vertices into the sample. Two extensions of the method were later published, allowing application to domains bounded by piecewise smooth surfaces [8, 9]. The former offers

stronger topological guarantees, but is self-admittedly not practically implementable. The latter is simplified by lumping all topological verification into one single test, but unfortunately does not allow complete control over triangle sizing. To handle small dihedral angles in the input, both algorithms use a protection scheme requiring the evaluation of a length scale along every curve in the model, a procedure that can only be done through a computationally expensive, brute force approach. Ultimately, a conforming DT of the input domain is generated.

The algorithm we present in Section 3.3 also uses a surface sampling strategy to construct the surface triangulation of the model. We borrow the idea of violation-driven refinement from Cheng et al. However, we introduce the idea of sampling each of the model's surfaces separately, while still matching the samples at curves bounding two or more surfaces. Each triangulation is therefore unaware of any neighboring geometric entities, thus eliminating the possibility of triangles incorrectly connecting two separate surfaces. Consequently, fewer topological violations need to be fixed, resulting in less insertions and inevitably coarser meshes. Furthermore, our algorithm does not require the expensive computation of the local feature size. However, as opposed to any of the aforementioned algorithms, a tetrahedralization conforming to the model's boundaries is not obtained for "free", as a by-product of the surface sampling. The tetrahedralization is instead generated separately from the vertex sample. Since this sample is typically coarser than that observed when building a conforming DT, the tetrahedralization might not be boundary conforming. A recovery stage is therefore necessary. After recovery, the resulting mesh exclusively contains constrained Delaunay tetrahedra matching a discrete representation of the boundary surfaces and is ideally suited for subsequent application of Delaunay refinement in its interior.

Since every surface patch is sampled independently, the proposed algorithm obviates the need for small angle protection. No costly length scale computation is therefore required prior to sampling. The algorithm is implemented using the GRUMMP meshing libraries [25]. All geometric functionality is handled by the Common Geometry Module (CGM) [36]. Selected

implementation issues are presented in Section 3.4, followed by results in Section 3.5. Our long term objective is to apply Delaunay refinement directly to geometries created by CAD modelers. We believe that our algorithm is particularly well-suited for that task.

3.2 Background

The meshing algorithm presented in this paper generates CDTs of geometric domains bounded by piecewise smooth surfaces. The objective is to create a tetrahedral mesh which will act as a starting point to launch Shewchuk's Delaunay refinement algorithm. Eventually, this will lead to a quality mesh usable in numerical simulations. In this section, we define the concepts necessary to the algorithm's presentation.

Following Cheng et al. [9], we define the input domain $\mathcal{D}$ to be a Piecewise Smooth Complex (PSC), the union of $d-$manifolds, $0 \leq d \leq 3$, embedded in $\mathbb{R}^3$; we denote by $\mathcal{D}_p$ the set of points, $\mathcal{D}_c$ the set of curves, $\mathcal{D}_s$ the set of surfaces and $\mathcal{D}_v$ the set of volumes. A curve can either be bounded by two points or can be closed with both ends connecting to a single point. Two curves can only intersect each other at a shared endpoint. A surface patch can either be smooth and closed or be entirely bounded by curves. In both cases, scratch curves or isolated points can be embedded in the surface. A surface may only intersect a curve at one (or more) shared point, unless the curve is embedded in the surface or is part of the surface's boundary. A surface may only intersect another surface along shared curves and points. We require the input PSC to be surface-bounded — the boundary separating the domain to be meshed from its exterior must be entirely covered by surface patches. This class of geometries notably includes solids produced by CAD modelers, which are of primary interest in the context of mesh generation directed towards simulations of engineering-relevant phenomena, as well as non-manifold models. If all elements in $\mathcal{D}_c \cup \mathcal{D}_s$ have zero curvature, then this definition reduces to that of a Piecewise Linear Complex (PLC), the input typically accepted by most previously published Delaunay refinement algorithms.

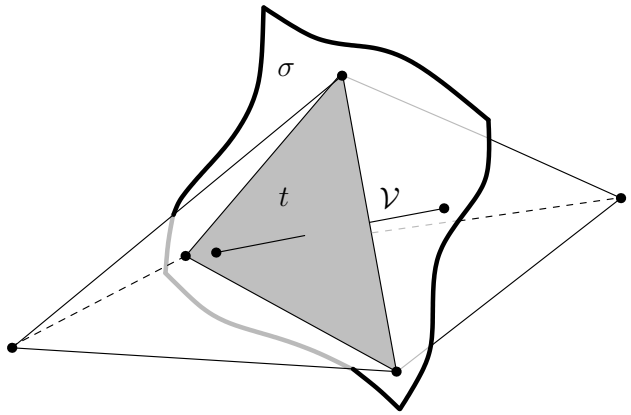


Figure 3.1: A Delaunay triangle part of the restricted Delaunay triangulation of a surface is illustrated. The Voronoi edge $\mathcal{V}$ dualing Delaunay triangle t intersects surface σ, therefore t is part of the restricted Delaunay triangulation.

Before a tetrahedralization of the PSC can be built, geometric entities in $\mathcal{D}_c$ and $\mathcal{D}_s$ must first be discretized. As a result, the set of segments and triangles required to constrain the tetrahedralization is obtained. We call a segment discretizing a curve *boundary edge* and a triangle discretizing a surface *boundary facet*. Extending the nomenclature of Shewchuk [33], every piece of curve discretized by a boundary edge is called a *subcurve*. The geodesic triangle corresponding to a boundary facet is called a *subsurface*.

As will be explained in Section 3.3.2, the boundary facets are created using a surface sampling algorithm. For a given surface σ, vertices are incrementally added to a sample v_σ, while maintaining these vertices' Delaunay tetrahedralization, $T(v_\sigma)$. The Delaunay triangulation *restricted* to σ, denoted $T(v_\sigma)|_\sigma$, is the subset of $T(v_\sigma)$ made of the triangular faces whose dual Voronoi edges intersect σ, as is illustrated in Figure 3.1. Boissonnat and Oudot [3] demonstrate that when the sample becomes dense enough, $T(v_\sigma)|_\sigma$ is homeomorphic to σ. The constraining boundary facets can then be extracted directly from the restricted Delaunay triangulation.

The mesh output by the algorithm is a simplicial complex $\mathcal{S}$, formed by the union of sets of vertices, edges, triangles and tetrahedra. It conforms to

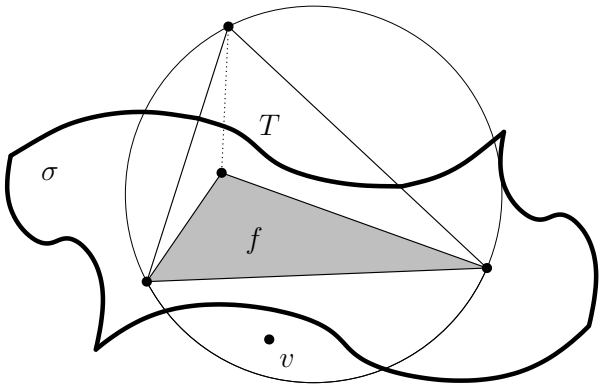


Figure 3.2: Tetrahedron T is constrained Delaunay. It contains vertex v inside its circumsphere, but boundary facet f discretizing surface σ occludes it from the interior of T.

$\partial\mathcal{D}$, the boundary of $\mathcal{D}$: all the points in $\mathcal{D}_p$ appear as vertices in $\mathcal{S}$, all the curves in $\mathcal{D}_c$ are entirely covered by edges in $\mathcal{S}$ and all surfaces in $\mathcal{D}_s$ are entirely covered by triangles in $\mathcal{S}$. The problem of boundary conformity is solved by the use of CDTs, thus the tetrahedra of $\mathcal{S}$ filling the volumes in $\mathcal{D}_v$ must themselves be constrained Delaunay. As illustrated in Figure 3.2, a tetrahedron is constrained Delaunay if it respects the boundary of the PSC and if there exists a sphere circumscribing it such that the sphere does not contain any vertex visible from the tetrahedron's interior [33]. A tetrahedron respects the PSC's boundary if none of its edges crosses a boundary facet and none of its faces cuts through a boundary edge.

3.3 Meshing Algorithm

This section describes the algorithm to construct CDTs from models bounded by piecewise smooth surfaces. The mesh is built from the bottom up, following a three part process. During the first part, the curves given as input are discretized. The vertices thus created form the basis from which the surface sampling phase is launched. Once this process is completed, the samples from each surface provide a discrete representation of the input domain's boundaries, from which the constraining edges and facets are obtained. A

Delaunay tetrahedralization of the sample vertices is then built. A boundary recovery phase follows, during which edges and faces of the tetrahedralization are matched with the constraining entities. A CDT is ultimately obtained.

3.3.1 Curve Discretization

The algorithm is initiated by creating a set of vertices on the curves in $\mathcal{D}_c$. A set of boundary edges, along with their corresponding subcurves, are built by connecting any two consecutive vertices along a given curve. A piecewise linear approximation of each curve is thus created.

The choice of the curve discretization technique is not critical to the success of the algorithm. Before being able to launch the surface sampling phase, it is likely that the initial sample will be refined. In fact, the subcurves of encroached boundary edges — boundary edges whose diametral sphere contain a vertex other than its two end vertices — must be split before any surface vertex can be safely inserted. Choosing a discretization scheme yielding a dense sample will obviously lead to a CDT containing more vertices and vice versa.

As was previously done in two dimensions [5, 16], the initial curve samples are built by bounding the total variation of the tangent angle, $TV(\theta) = \int_c |d\theta|$, where θ is the angle subtended by the tangent to the curve, over any subcurve. This method is very simple to implement and inherently places more vertices where the curvature is higher. For the meshes presented in this paper, we set the bound to $\pi/4$. Any value under $\pi/2$ guarantees that the subcurve will be entirely contained inside its boundary edge's diametral sphere.

Once the initial discretization is done, the neighborhood of vertices at the apex of acute angles subtended by input curves must be protected to prevent infinitely recursive refinement. Encroached boundary edges are usually absent from the restricted Delaunay triangulation of a given surface and as a result, subcurve splits are performed to recover them. If the missing boundary edge is adjacent to a small input angle, the split may cause another boundary edge adjoining this same small angle to go missing. The cycle repeats infinitely

and cascading subcurve splits ensue.

This behavior can be avoided by forcing subcurve splits to coincide with concentric balls centered at the apex of the small angles. This method's two-dimensional equivalent was originally proposed by Ruppert [31]. We illustrate the procedure in Figure 3.3.

Since we use diametral spheres to define boundary edge encroachment, input angles of less than 90° need to be protected. During the curve discretization phase, only a first layer of protection is built. Every subcurve adjoining a small angle and having *both* its vertices corresponding to points in $\mathcal{D}_p$ are first split at their regular split point — under normal circumstances, we split a subcurve such that TV is equal over the two newly created subcurves. Then, for vertex a at the apex of a small angle, we define $r_{\max}$, the maximum split radius, to be two thirds of the length of the shortest boundary edge e connected to a and adjoining the small angle. The first ball's radius r is then chosen to be the biggest power of two less than $r_{\max}$. All subcurves connected to a and adjoining the small angle are split at distance r from a. This ensures a split between one third and two thirds of the length of e. Subcurves are then protected against cascading encroachment splits.

As illustrated in Figure 3.3d, every subsequent small angle subcurve splits that may occur during the following phases of the algorithm will have to be on the next smaller concentric ball whose radius is a power of two.

Vertex connectivity data along every curve must be maintained throughout subsequent phases. As will be seen in the next section, this data is essential to verify if any given boundary edge appears in the restricted Delaunay triangulation.

3.3.2 Surface Sampling

Our meshing algorithm enforces boundary conformity by requiring the tetrahedral mesh to match a set of constraining facets, in the form of a surface triangulation. To obtain this triangulation, we use a surface sampling method devised from previous results by Boissonnat and Oudot [3] and Cheng et al. [8].

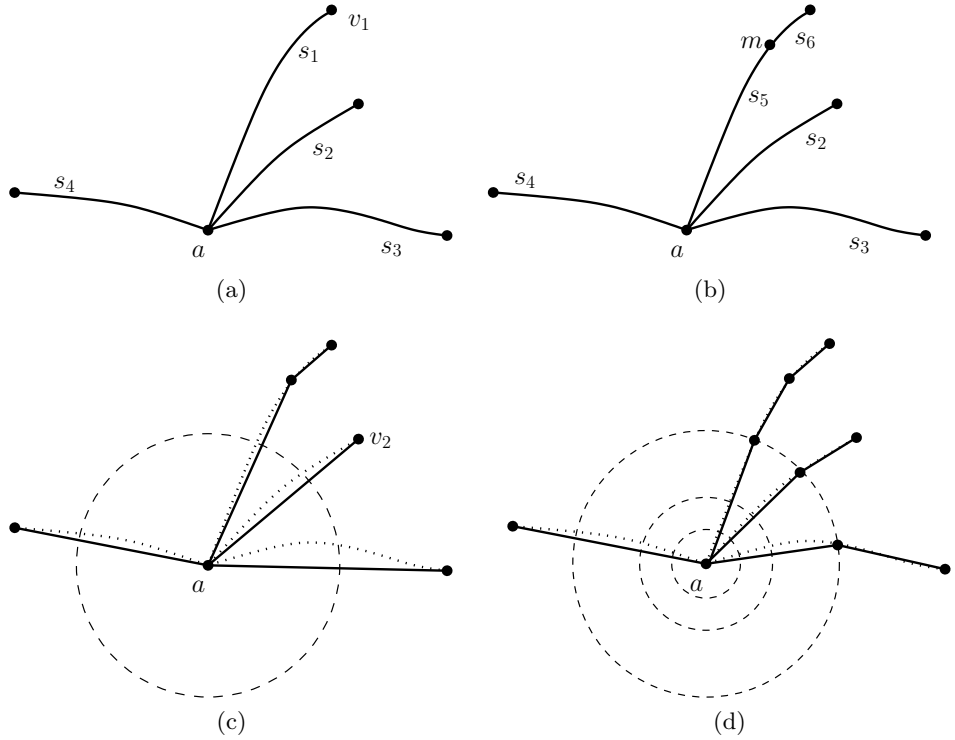


Figure 3.3: Protecting the neighborhood of a vertex at the apex of a small input angle. (a) Subcurves s_1, s_2 and s_3 adjoin a small input angle at vertex a. s_4 is connected to a but does not form a small angle with another subcurve. a and v_1 both correspond to points in $\mathcal{D}_p$. (b) Subcurve s_1 is split at its mid-TV point m, thereby forming s_5 and s_6. (c) $\overline{av_2}$ is the shortest boundary edge adjoining a small angle. Its length defines the initial protection ball's radius. (d) Subcurves are split at their intersection with the protection ball. Subsequent splits will occur on smaller concentric balls.

The constraining facets are extracted from the restricted Delaunay triangulation of a surface sample constructed incrementally; every vertex insertion is an attempt at removing a topological violation. Optionally, poorly-shaped triangles or triangles considered too large according to a given sizing function can also be targeted with vertex insertions. The main difference between previous algorithms and ours is that we sample every surface patch independently. Patches are only aware of neighboring geometric entities through the vertices and subcurves they share with them. The remainder of this section is devoted to the presentation of the sampling procedure.

3.3.2.1 Initialization

To launch the sampling algorithm, a Delaunay triangulation restricted to each individual surface $\sigma \in \mathcal{D}_s$ must be created. To this end, the Delaunay tetrahedralization T of a rectangular prism large enough to contain σ is built. The vertices used to discretize the elements in $\mathcal{D}_p \cup \mathcal{D}_c$ *adjacent* to σ are then inserted into T. The Delaunay empty sphere property is maintained in T at all time. This can either be done using an incremental face flip algorithm or the Bowyer-Watson insertion algorithm [6, 37]; our implementation uses the latter. The initial restricted triangulation $T|_\sigma$ is finally extracted from T by identifying the faces whose Voronoi edge intersects σ. The process is then repeated as to obtain a restricted Delaunay triangulation for each surface patch in $\mathcal{D}_s$.

At this point we might be faced with either of the following two situations:

- $T|_\sigma$ contains at least one triangle. In this case, the topological violation-driven sampling algorithm can be launched immediately.

- $T|_\sigma$ contains no triangle. This can only happen if σ is a smooth surface, most probably without embedded entities, or if all initial boundary edges are encroached. In the former case, the surface must be seeded by inserting arbitrary vertices, ideally as far as possible from each other, until restricted faces are obtained. The latter case is very unlikely, but splitting the encroached boundary edges' subcurves will eventually create some restricted faces.

3.3.2.2 Sampling for Topological Correctness

The objective of surface sampling is to produce a triangulation that is a good approximation, both topologically and geometrically, of every surface $\sigma \in \mathcal{D}_s$. At this stage, vertices are added to the sample to recover σ's topology. Every vertex insertion creates and deletes restricted faces in $T|_\sigma$. Consequently, an insertion might repair some topological violations, but can also introduce new violations. A series of tests are performed on new vertices and their neighboring restricted faces to assess whether or not $T|_\sigma$ locally respects the topology of σ. We identify three canonical violations that our algorithm detects and attempts to fix:

- Missing or encroached boundary edges.
- Voronoi edges intersecting the surface more than once.
- Invalid topological disk around a sample vertex.

The number of potential violations is small due to the fact that every surface is sampled independently. It is for instance impossible for a restricted triangulation to connect two neighboring surfaces since a given triangulation is not "aware" of any other nearby geometric entities. The sampling algorithm is incremental and topological testing is only performed on newly inserted vertices. Every time a new vertex v is inserted into the sample, it is tested against the canonical violations. If v fails any of the tests, a new vertex $\tilde{v}$ is added to the sample to fix the violation. Both v and $\tilde{v}$ are then queued to be submitted to a new series of tests. Once v passes all tests successfully, it will not need to be checked again.

Insertions to recover missing boundary edges are given priority over those triggered by multiple Voronoi edge intersections, which in turn have priority over those to fix an incorrect topological disk around a vertex. Generally, the multiple intersection violation occurs sporadically, only at the very beginning of a sampling run.

Missing or Encroached Boundary Edges

When launching the sampling algorithm, many boundary edges are likely to be missing from $T|_\sigma$. As illustrated in Figure 3.4, this situation occurs when a boundary edge is encroached upon by another vertex (a vertex lies inside the boundary edge's diametral sphere). It is also possible for a boundary edge previously present in $T|_\sigma$ to go missing following the split of a nearby subcurve. The detection of missing boundary edges is the reason why the connectivity data along curves must be maintained throughout the sampling procedure.

Before being able to start sampling the surface, missing boundary edges must first be recovered. Vertices created during the curve discretization phase of Section 3.3.1 are all checked to verify if their adjoining boundary edges correctly appear in $T|_\sigma$. Whenever a boundary edge is found to be missing, its subcurve is immediately split. Obviously, if surface σ shares this subcurve with neighboring surface $\tilde{\sigma}$, the split vertex must be inserted in both σ and $\tilde{\sigma}$ samples to keep both curve discretizations matching at all time.

Boundary edge encroachment can also lead to a face of T being declared restricted when, in fact, it should not. Such a situation is depicted in Figure 3.5. These unwanted faces can be eliminated by splitting the encroached boundary edge's subcurve. To detect this violation, every time a vertex v is inserted on a curve, any boundary edge on the perimeter of its restricted 1-ring neighborhood must be tested against encroachment. This must be done in addition to ensuring that v's adjoining boundary edges appear in $T|_\sigma$. The subcurve of any encroached boundary edge must immediately be split. Note that since the algorithm refuses to insert any surface vertex encroaching on boundary edges, this test can be skipped for all surfaces vertices.

Multiple Voronoi Edge Intersections

The surface sampler's next highest priority is to eliminate restricted faces whose dual Voronoi edges intersect the surface at more than one location. As was previously mentioned, this type of violation is very rare and generally

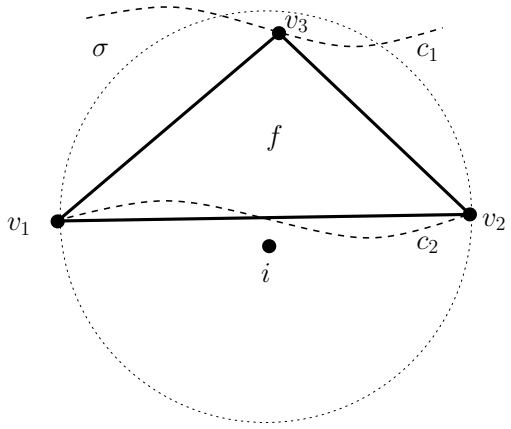


Figure 3.4: The case of a boundary edge missing from the restricted Delaunay triangulation is illustrated. Curves c_1 and c_2 bound surface σ. f is a face of the Delaunay tetrahedralization T and its Voronoi edge intersects the plane of f at i. The Voronoi edge never intersects surface σ and so f is not in $T|_\sigma$. Consequently, boundary edge $\overline{v_1v_2}$, which is encroached upon by v_3, is missing from $T|_\sigma$, a topological violation.

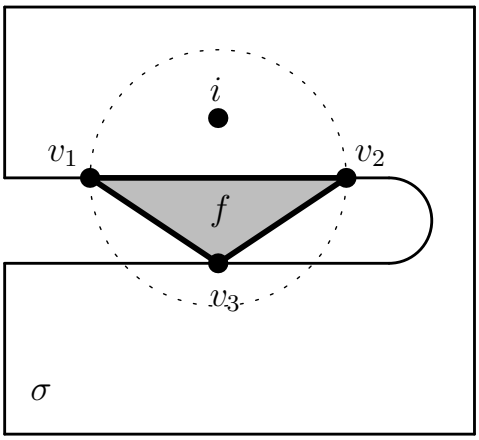


Figure 3.5: Boundary edge encroachment can lead to faces being incorrectly declared restricted. Surface σ has a gap in the middle. Face f covers this gap and should not be part of surface σ's triangulation. Because v_3 encroaches upon $\overline{v_1v_2}$, f's Voronoi edge intersects σ at i. As such, f is recognized as being part of $T|_\sigma$, a topological violation.

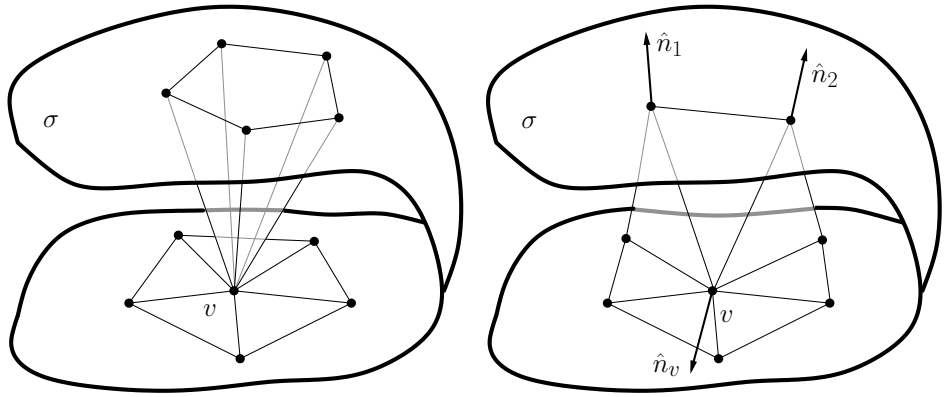
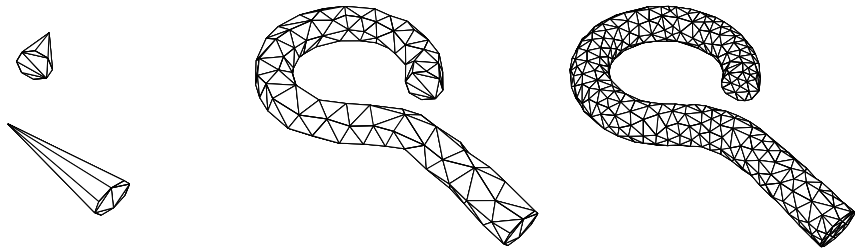


Figure 3.6: Two situations where the restricted 1-ring neighborhood around surface vertex v is considered incorrect. (a) The restricted neighborhood of v does not form a valid topological disk. It connects to two distinct locations on σ. (b) A valid topological disk is present around v, but the angles between the surface normals are considered too large.

occurs when the surface sample is still very coarse. In the vast majority of cases, the potentially guilty faces are gone by the time boundary edge encroachment is fixed. Nonetheless, since this type of violation is very easy to detect, we still implement it. Given a restricted face f with a Voronoi edge intersecting surface σ at multiple locations, a new sample vertex is inserted at the intersection point that is the farthest from f's circumcenter.

Invalid Topological Disk

The last of the topological checks involves the inspection of the sample vertices' restricted 1-ring faces. These faces must form a valid topological disk — or part thereof in the case of a curve vertex — otherwise $T|_\sigma$ is non-manifold around the vertex. In addition, we enforce a normal deviation condition over the restricted 1-ring neighborhood, which we lump into the topological disk verification. Figure 3.6 shows examples where either the topological disk or the normal deviations conditions are violated around a surface vertex.



(a) (b) (c)

Figure 3.7: Justification for the normal deviation condition on the *hook* model. (a) All vertices satisfy the topological disk condition, but the model's topology is not correctly recovered. (b) Adding the normal deviation condition fixes the problem. (c) The surface mesh obtained by adding size and quality criteria (see Section 3.3.2.3).

The restriction imposed on normal deviation is necessary to detect situations such as the one depicted in Figure 3.7. The hook model shown on the figure is made of three surface patches. The largest patch's topology is obviously incorrect in Figure 3.7a, despite the fact that the topological disk condition is respected at all its vertices. The problem is fixed by forcing the angle between the surface normals at a sample vertex v and at any other vertex in v's restricted neighborhood to be smaller than some threshold value; our implementation uses a maximum allowed angle of $\pi/2$.

The same problem could also be fixed without imposing a normal deviation condition. Instead, the algorithm would need to recognize that that the hook's main surface is represented by two disjoint sets of triangles, a topological violation in itself. However, implementing such a test in an incremental vertex insertion setting is difficult and relatively expensive, especially if it must be repeated frequently. Indeed, all other tests only perform checks on a vertex's immediate neighborhood, whereas this test would require repeated walks over the entire surface triangulation.

When v is a vertex is located on a curve bounding σ, the topological disk test has to be slightly modified. Asking for the restricted neighborhood to form a disk is obviously inadequate in this situation. Instead, we reduce the disk condition to requiring a walkable restricted neighborhood around v. In

other words, knowing that a curve vertex always adjoins two boundary edges, it must be possible to start from one of these boundary edge, walk across the restricted faces connected to v and attain the other boundary edge. The walk must traverse *all* restricted faces adjoining v once, otherwise a violation is detected.

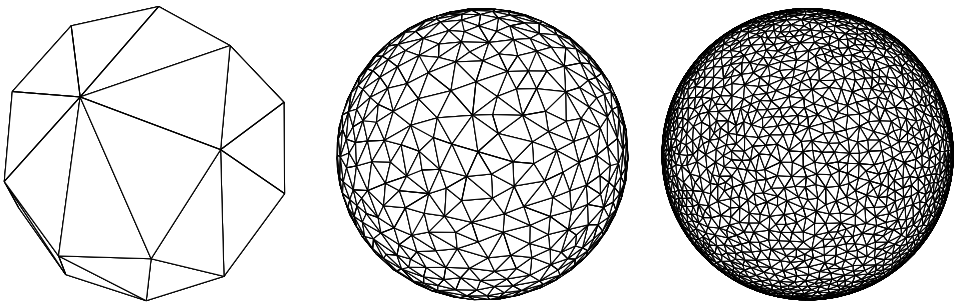
A failure to satisfy either the disk or the normal deviation condition leads to the split of the largest face in the restricted 1-ring neighborhood of the vertex under scrutiny. A new sample vertex is inserted at the intersection between the largest face's Voronoi edge and the surface. This is Chew's furthest point strategy [11]. The procedure is repeated until no more violations can be found.

3.3.2.3 Sampling for Size and Quality

Once all topological violations are fixed, sampling can optionally be continued to satisfy size and quality criteria. In both cases, restricted triangles are split according to Chew's furthest point strategy.

The choice of a sizing function to control the density of a surface triangulation destined for numerical simulations is very much application dependent. As a proof of concept, we chose to implement a sizing field based on the surface curvature. A restricted face gets split every time its circumradius is greater than a user-defined constant ρ times the minimum radius of curvature at its vertices. The effect of our approach on a simple sphere model is illustrated in Figure 3.8. This demonstrates that mesh density can be easily controlled in practice.

Additional insertions can also improve the quality of the surface triangulation. Boissonnat and Oudot [3] demonstrate that the same angle bound of 20.7° originally proven by Ruppert for two-dimensional domains can also be achieved when sampling smooth surfaces. A minimum angle of 30° can be reached in practice. Restricted triangles are therefore split until their circumradius-to-shortest-edge ratio becomes less than or equal to 1. The algorithm's behavior must however be modified in the neighborhood of small angles subtended by input curves. Following Miller et al. [22],



(a) (b) (c)

Figure 3.8: Surface meshes of a *sphere* model with three levels of curvature-based refinement. (a) $\rho = 1$. (b) $\rho = 5$. (c) $\rho = 10$.

we refuse to split restricted triangles whose shortest edge spans a small input angle when this edge's vertices are both on a concentric ball. This method was successfully adapted to work with curve-bounded inputs in two dimensions [16].

3.3.2.4 Limitations

The choice to sample each surface individually is partly justified by the non-necessity to protect small input dihedral angles, at least for most geometries. There are unfortunately two situations where the algorithm is likely to fail to terminate: at the apex of a cone forming a small angle and along a curve where the same surface patch attaches twice, forming a dihedral angle of less than 90°. An example of the latter is shown in Figure 3.9. The small dihedral angle causes vertex v_b, located on the bottom of surface σ, to be incorrectly connected to the top part of σ by restricted faces. An insertion to fix the problem should create a similar violation nearby. As a result, edge length will dwindle in this neighborhood and the algorithm will fail to terminate. The situation is analogous to that of the runaway encroachment splits discussed earlier.

The most obvious fix to the situation in Figure 3.9 is to partition σ into two surfaces by using for instance virtual geometry operations [18]. The problem with cone-like geometries could conceivably be solved with the same

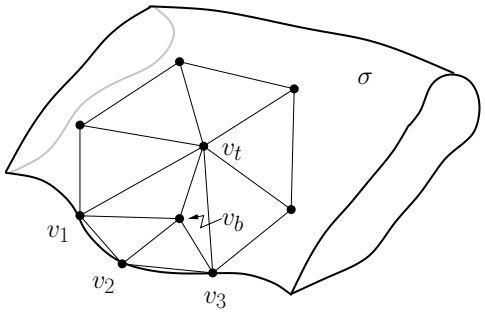


Figure 3.9: A case where the sampling method is likely to fail to terminate. Vertex v_b is on the bottom part of σ, v_t is on the top part. Assume v_b to be located inside the equatorial spheres of $\triangle v_1 v_2 v_t$ and $\triangle v_2 v_3 v_t$. As such, these triangles might not be in $T|_\sigma$. The normal deviation condition is violated at v_t, but attempting to remove this violation by sampling will result in cascading refinement.

approach. The case of the warped surface attaching along a curve is easily identifiable by topological adjacency queries. A protection technique could therefore be applied along the problematic curves [8, 29], while leaving other curves untouched. More investigation into the matter is necessary.

3.3.3 Boundary Recovery and CDT Construction

At the end of the sampling operation, the triangulations restricted to every surface patch provide the boundary edges and facets needed to constrain the CDT. This section presents how this CDT is generated using the sample vertices and the constraining entities.

The process begins by building a Delaunay tetrahedralization T of all the surface patches' sample vertices. Some constraining entities might appear in T, but others will probably be missing and will need to be recovered. To identify the missing entities, the restricted triangulations $T_R = \bigcup_{\sigma \in \mathcal{D}_s} T|_\sigma$ must be maintained throughout the process, alongside a dictionary of the one-to-one correspondence between vertices in T and T_R. The boundary recovery method we propose relies on topological transformations and if necessary, on vertex insertion. The topological transformations include the

face flips and edge swaps described by Freitag and Ollivier-Gooch [12].

Edge Recovery by Swapping

The recovery algorithm starts by identifying which boundary edges and surface edges are absent from T. An attempt is first made to recover the missing edges exclusively with topological transformations. Assume e to be either a boundary or a surface edge in T_R bounded by vertices v_0 and v_1. If v_0 and v_1 are not connected in T, then e must be recovered. The following operations are performed until either e appears in T or e is deemed impossible to recover by swapping alone, in which case it will have to be recovered by insertion.

1. Compute the line segment $\overline{v_0 v_1}$. If e is absent from T, then this segment must either intersect a face f of T or be within machine precision of hitting another edge $\tilde{e}$ of T squarely. In the latter case, go to step 2, otherwise go to step 3. If no face is intersected and no edge is hit by $\overline{v_0 v_1}$, then e appears in T.

2. An edge swap to remove the offending edge $\tilde{e}$ from T is attempted; if this fails, then e is not recoverable by swapping alone, exit and try to recover e by vertex insertion. If $\tilde{e}$ is successfully eliminated, then go back to step 1

3. If a face f is in a flippable configuration, then attempt the flip to remove f from T. If successful, go back to step 1, otherwise go to step 4.

4. Search for $\tilde{f}$, the next face after f to be intersected by $\overline{v_0 v_1}$. If such face is found, set $f \leftarrow \tilde{f}$ and go back to step 3. If an edge $\tilde{e}$ is hit before finding $\tilde{f}$, go back to step 2. If v_1 is reached without the path from v_0 to v_1 being completely cleared, then exit and try to recover e by insertion.

Edge Recovery by Insertion

The swapping procedure alone often fails to force all constraining edges into T; a tetrahedralization containing all constraining edges might not even exists [33]. At this point, edges and faces of T are not locked in place. It is therefore likely that a swap to recover one given edge will knock a previously recovered edge out of T. When topological transformations are unsuccessful, there is no other choice but to resort to vertex insertion.

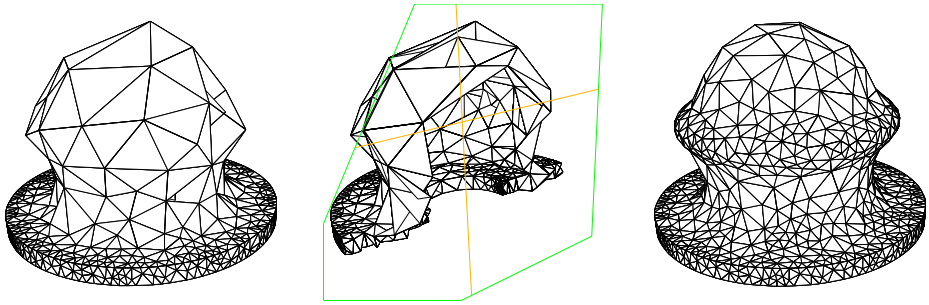
The unrecoverable constraining edge e is either a surface edge in $T|_\sigma \in T_R$ or a boundary edge shared by one or many members of T_R:

- If e is a surface edge, split e in $T|_\sigma$ by inserting a sample vertex v at the projection of e's midpoint onto σ.
- If e is a boundary edge, split e's subcurve in the members of T_R sharing e. Vertex v is inserted at the subcurve's regular split point, unless a small angle is involved, in which case v is placed on a concentric ball. It should be clear that even though v is inserted in many members of T_R, it corresponds to *one* single vertex in T. This is done to keep the subcurves matching along any curve shared by two or more surfaces.

In both cases, inserting v can lead to the insertion of other sample vertices, whether it is to fix a newly created topological violation or to satisfy the quality criterion. The set of constraining entities must be updated to reflect the changes in T_R.

The new sample vertices are then inserted in T. Because of the topological transformations previously performed, T is no longer Delaunay and consequently, Bowyer-Watson insertion obviously cannot be used. Any insertion scheme maintaining a valid tetrahedralization works since we do not aim at maintaining the empty sphere property.

After the insertions, if edges are still missing from T, then a new attempt at recovering them by topological transformations is made. The cycle is repeated until all constraining edges appear in T.



(a) (b) (c)

Figure 3.10: *Snapfit* model. (a) Coarse surface meshes obtained without curvature-based sampling. The triangulations from two surface patches are intersecting. (b) Cutaway of the boundary constrained tetrahedralization, some insertions were needed to recover the intersecting boundary facets. (c) Surface mesh obtained using curvature-based refinement with $\rho = 1$.

Facet Recovery

Now that all constraining edges appear in T, the focus shifts towards recovering boundary facets. Again, recovery through swapping is first attempted. Vertex insertion is typically only necessary to fix self-intersections that can appear in T as a result of sampling surfaces independently. If two or more constraining facets are intersecting, it will be impossible to recover them both by swapping; recovering one of them will inevitably knock the other out of T. Vertex insertions need to be used until all intersections disappear. This is the case for the model presented in Figure 3.10.

Although the boundary recovery procedure relies entirely on heuristics, we make the claim that it is bound to terminate. Our claim is based on the fact that every time a constraining entity cannot be recovered by topological transformations, at least one new sampling vertex is inserted. If recovery through transformations keeps failing, every constraining entity must eventually become small enough to have a vertex free circumscribing sphere in T. They then all respect the Delaunay property. At this point a DT conforming to the constraining entities exists and is recoverable by face and edge swaps. Therefore, in the worst case scenario, the tetrahedralization

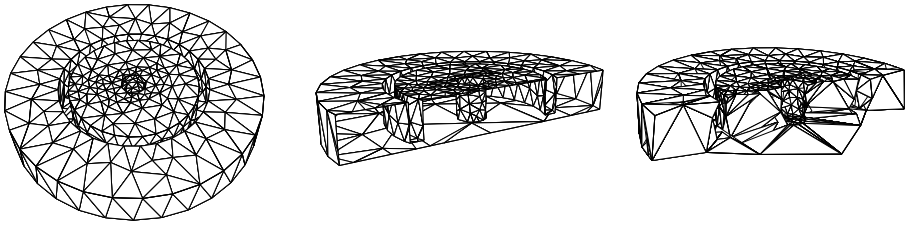
generated by our algorithm will be a conforming DT. In practice, T becomes boundary constrained before that. Previously published algorithms [10, 27, 30] are designed to generate conforming DTs, which will typically be finer than our CDTs on similar geometries.

Obtaining a CDT

Once the constraining edges and facets have been recovered in T, they are locked in place and can no longer be swapped away. From this point, the CDT is obtained by flipping interior faces until all tetrahedra's circumsphere are vertex free. Theory tells us that this CDT might not exist. If interior flips fail, constraining entities can be unlocked and the surface samples can be further refined. However this situation has never occurred in practice — a CDT is always obtained once T is boundary constrained.

One major benefit of sampling every surface independently and then recovering the boundary by topological transformations is illustrated in Figure 3.11. The model's bottom surface remains coarse even though another surface comes very close to it. If the entire model had been sampled at once, the size of the triangles on the bottom surface would have been much smaller, within a constant factor of the distance between the surfaces.

This observation is significant if Delaunay refinement is going to be used to generate quality tetrahedralizations from the CDTs produced by our algorithm. For instance instead of tetrahedralizing the model's interior as is depicted in Figure 3.11c, imagine that only the tetrahedra exterior to the model, extending to some bounding box, are kept after boundary recovery. Since the CDT's interior is not meshed, the top and bottom surfaces are invisible from each other and are therefore unaffected by encroachment rules. Consequently, the local feature size on the bottom surface does not depend on the distance to the top surface. Since the size of the elements in a Delaunay refined mesh are guaranteed to be within a constant factor of this local feature size, we can expect a quality mesh of the domain's exterior to be coarser with our approach than if conforming DTs had been used.



(a) (b) (c)

Figure 3.11: The main benefit of sampling every surface independently and then applying boundary recovery is illustrated on the *wheel* model.(a) Top view of the surface mesh. (b) Cutaway showing that triangles on the bottom surface are larger than the local feature size. (c) Boundary constrained tetrahedral mesh. No additional vertex is necessary to recover the boundary and the bottom surface remains coarse.

3.4 Implementation Notes

The algorithm presented in this paper was implemented using the GRUMMP meshing libraries [25]. The Common Geometry Module [36] is used as a geometric back-end. Results presented in this paper were obtained using CGM's facet-based modeler [28]. Starting from a faceted geometry, for instance a surface triangulation in STL format, the model's topology is inferred by analyzing the dihedral angle between facets; if the dihedral angle subtended by two facets is larger than some user-specified value, then the edge shared by the facets must be part of a curve. Using the list of curves thus obtained, a boundary representation of the model is then constructed. Curved surfaces can then be interpolated from a collection of facets using Bézier patches. The construction of a topologically correct boundary representation is not an easy problem, especially when the model's complexity increases or when the quality of the surface triangulation is questionable. It is fairly common for curves to fail to be detected. When this happens, the meshing algorithm has little to no chance of success. Proper access to geometry currently limits our implementation's range of application and is the reason why the models presented in this paper are fairly simple.

The OpenCASCADE interface recently added to CGM will be integrated in our mesher in the near future to solve this problem.

As we are using faceted models, surface curvature data is not readily available. To evaluate curvature, we rely on the discrete differential geometry operators proposed by Meyer et al. [21]. Since their curvature operator is non-convergent, the curvature can only be calculated on the input triangulation, not on the mesh being refined. An accurate evaluation therefore depends on the quality of the faceted surface. A bad input mesh can affect the curvature-based refinement, as can be seen in Figure 3.14. A small spot of over refinement is present on the blade's surface as a consequence of poorly-shaped facets in the input triangulation. Other possible approaches to evaluate curvature of a triangulated surface are described in the survey by Garimella and Swartz [14].

As a final note, we mention that insertions during the sampling phase are performed using a Bowyer-Watson framework. This approach is particularly well-suited to the sampling algorithm; every time a new vertex is inserted, an empty hull is formed around the new vertex which is later reconnected with tetrahedra. New entities in the mesh are thus easily identifiable. Faces which must be added to the restricted triangulation can quickly be computed among a small set of new faces. For increased robustness, the insertion scheme is used in conjunction with Shewchuk's adaptive precision geometric predicates [32].

3.5 Results

This section presents cardinality and timing data for the various meshes presented in this paper. These results are presented in Table 3.5. All the meshes were obtained using a curvature sampling constant of $\rho = 1$, except for the sphere model which uses $\rho = 10$. Figures 3.12 to 3.14 show additional test geometries. Wall clock speeds are obtained on a single core of an Intel Q6600 processor.

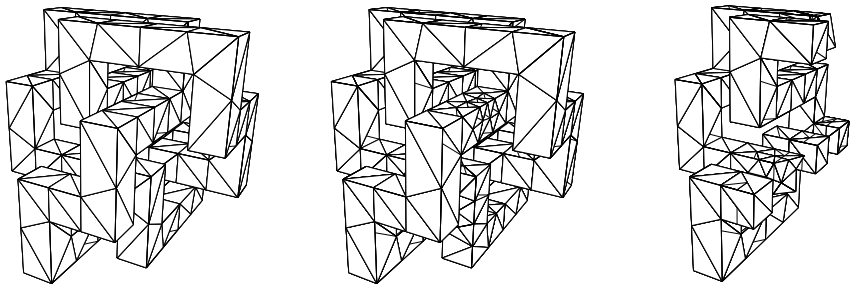
For most test models, the Delaunay tetrahedralization built from the surface samples is not initially boundary constrained. The results of Table 3.1

	Number of vertices			Time (s)	
Model	Curve	Sample	Recovery	Sample	Recovery
Hook	16	484	0	0.17	0.32
Sphere	0	3173	0	0.84	1.0
Snapfit	40	1700	1	0.60	1.2
Wheel	64	387	0	0.15	0.13
Sculpture	96	108	42	0.18	0.79
Torus	114	1830	0	1.2	1.5
Twist	84	421	0	0.27	1.1
Rings	137	1099	0	0.87	1.0
Blade	83	2017	0	0.63	0.89

Table 3.1: Sampling and CDT construction data. The number of vertices needed to discretize the curves, to sample the surfaces and to recover the boundary are given. The time (in seconds) required for sampling and recovering the tetrahedralization's boundary is also given.

show that the recovery procedure based on topological transformations typically adds very few vertices, if any, to achieve its objective. This evidence allows us to conclude that the CDT we generate are coarser than a conforming Delaunay tetrahedralization of the same model.

When compared with the published results of Cheng et al. [10], our implementation is about one order of magnitude quicker in terms of the number of vertices inserted every second. We can insert about 1500 vertices per second during sampling, while Cheng et al.'s insertion rate is anywhere between 100 and 200, depending on the test case. Furthermore, their timing includes insertions in the interior of the tetrahedralization, which are typically much faster than surface sampling insertions. For example, our CDT refinement algorithm inserts about 4000 vertices per second [15]. We believe that this fairly large difference in timing is the result of two factors: we do not evaluate the local feature size by brute force prior to starting surface sampling and we use a sampling strategy relying on simpler topological tests. Nonetheless, Cheng et al. have the merit of presenting test cases of higher complexity than ours. Efficiency comparison with the implementation of Rineau and



(a) (b) (c)

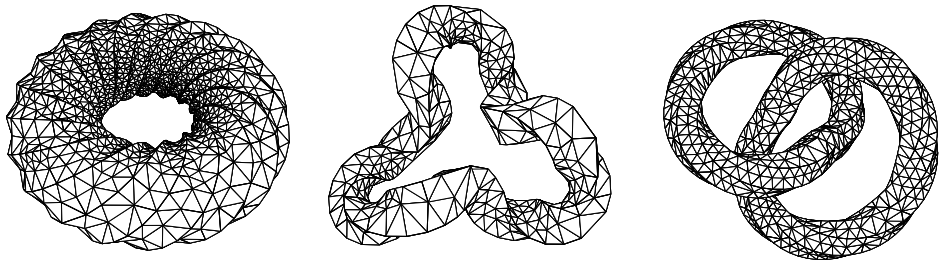
Figure 3.12: The sampling algorithm performs well on models bounded by planar facets as shown by this *sculpture* model. (a) Surface triangulation after sampling. (b) Surface triangulation after recovery. (c) Cutaway view showing the tetrahedralization.

Yvinec [30] is impossible since they are silent regarding timing.

3.6 Conclusion

We have presented a practical algorithm which meshes volumes bounded by piecewise smooth surfaces with constrained Delaunay tetrahedra. The constraining surface triangulation is obtained by sampling all surfaces independently, until topological violations are eliminated. The independent sampling approach allows for a simpler set of tests that are easier to implement and faster to execute. Optionally, sampling can also improve the quality of the surface triangulation and adapt it to a sizing field. Because each surface is sampled independently and since the boundary is recovered mostly through swapping, the meshes generated by our algorithm should typically be coarser than a conforming Delaunay tetrahedralization of the same model. We believe that these meshes are an ideal starting point to produce quality tetrahedralizations with Delaunay refinement algorithms.

Although we demonstrate the benefits of independent sampling in terms of mesh cardinality, some might argue that these are outweighed by the need to double the amount of work to build a boundary constrained tetrahedralization.

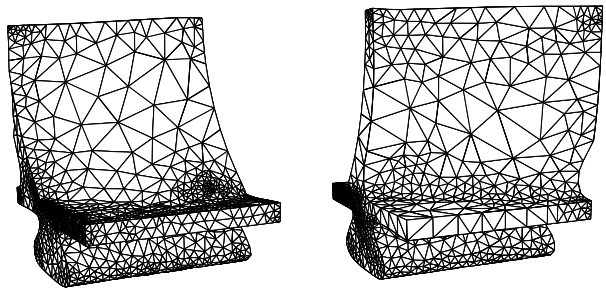


(a) (b) (c)

Figure 3.13: Surface meshes of various models. (a) *Torus*. (b) *Twist*. (c) *Rings*.

After all, insertion operations must be performed at least twice: once in the surface sampler (perhaps more than once if the vertex is inserted on a curve shared by many surfaces) and once in the final tetrahedralization. Maintaining many meshes will undoubtedly consume more memory. However, we believe the potential increase in computational time is a non-issue with today's omnipresence of multi-core processors. Insertions on surfaces and in the final tetrahedralization are two completely independent processes and can easily be run in separate threads, on two different cores. Conceptually, the independent sampling procedure also seems well-suited to be multi-threaded, although achieving proper load balancing could be a challenge.

We are aware that our algorithm will fail in the presence of a few, well identified, topological configurations. Future work will attempt to address these flaws. We will also focus our attention on applying the algorithm to real CAD geometries, through the recent OpenCASCADE interface in CGM. The difficulties encountered with boundary representation construction from a faceted model will thus be avoided. This will allow us to verify how our approach scales when used with an increase in complexity and mesh sizes.



(a) (b)

Figure 3.14: Surface mesh of the *blade* model. (a) Front view. Notice the overrefined region due to an inaccurate curvature estimation resulting from a faceted model with poor quality triangles. (b) Back view.

3.7 Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617 – 625, 2005.

[2] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22(4):481–504, 1999.

[3] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.

[4] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of Lipschitz surfaces. In *Proceedings of the Twenty-Second Annual Symposium on Computatinal Geometry*, pages 337–346, 2006.

[5] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1203, 2002.

[6] Adrian Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[7] Paulo Roma Cavalcanti and Ulisses T. Mello. Three-dimensional constrained Delaunay triangulation: A minimalist approach. In *Proceedings of the Eighth International Meshing Roundtable*, pages 119–129, 1999.

[8] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 477–494, 2007.

[9] Siu-Wing Cheng, Tamal K. Dey, and Edgar A. Ramos. Delaunay refinement for piecewise smooth complexes. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–1105, 2007.

[10] Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, and Tathagata Ray. Sampling and meshing a surface with guaranteed topology and geometry. *SIAM Journal on Computing*, 37(4):1199–1227, 2007.

[11] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280, 1993.

[12] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[13] Lori A. Freitag and Carl Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *International Journal of Computational Geometry and Applications*, 10(4):361–382, 2000.

[14] Rao V. Garimella and Blair K. Swartz. Curvature estimation for unstructured triangulations of surfaces. Technical Report LA-UR-03-8240, Los Alamos National Laboratory, 2003.

[15] Serge Gosselin and Carl Ollivier-Gooch. Constructing constrained Delaunay tetrahedralizations of volumes bounded by piecewise smooth surfaces. Manuscript, 2009.

[16] Serge Gosselin and Carl Ollivier-Gooch. Revisiting Delaunay refinement triangular mesh generation on curve-bounded domains. Manuscript, 2009.

[17] Bryan M. Klingner and Jonathan R. Shewchuk. Agressive tetrahedral mesh improvement. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 3–23, 2007.

[18] Jason Kraftchek. Virtual Geometry: A mechanism for modification of CAD model topology for improved meshability. Master's thesis, Department of Mechanical Engineering, University of Wisconsin, 2000.

[19] Michal Křížek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.

[20] François Labelle and Jonathan R. Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transations on Graphics*, 26(3):57.1–57.10, July 2007.

[21] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential geometry operators for triangulated 2-manifolds. In *Proceedings of Visualization and Mathematics*, 2002.

[22] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the Twelfth International Meshing Roundtable*, pages 91–102, 2003.

[23] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Proceedings of the Fifth International Meshing Roundtable*, pages 47–61, 1996.

[24] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in three dimensions. In *Proceedings of the Eight Annual Symposium on Computational Geometry*, pages 212–221. ACM, 1992.

[25] Carl Ollivier-Gooch. *GRUMMP Version 0.3.0 User's Guide*. Department of Mechanical Engineering, The University of British Columbia, 2005. Available from http://tetra.mech.ubc.ca/GRUMMP/.

[26] Carl Ollivier-Gooch, Amir Nejat, and Christopher Michalak. On obtaining and verifying high-order finite-volume solutions to the Euler equations on unstructured meshes. *American Institute of Aeronautics and Astronautics Journal*. To appear.

[27] Steve Oudot, Laurent Rineau, and Mariette Yvinec. Meshing volumes bounded by smooth surfaces. Technical Report 5626, INRIA, July 2005.

[28] Steven J. Owen, David R. White, and Timothy J. Tautges. Facet-based surfaces for 3D mesh generation. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 297–311, 2002.

[29] Alexander Rand and Noel Walkington. 3D Delaunay refinement of sharp domains without a local feature size oracle. In *Proceedings of the Seventeenth International Meshing Roundtable*, pages 37–54, 2008.

[30] Laurent Rineau and Mariette Yvinec. Meshing 3D domains bounded by piecewise smooth surfaces. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 443–460, 2007.

[31] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[32] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.

[33] Jonathan R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the Eleventh Meshing Roundtable*, pages 193–204, 2002.

[34] Jonathan R. Shewchuk. What is a good linear element? Interpolation, conditioning, anisotropy, and quality measures. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 115–126, September 2002.

[35] Hang Si and Klaus Gärtner. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the Fourteenth International Meshing Roundtable*, pages 147–163, 2005.

[36] Timothy J. Tautges. The common geometry module (CGM). Technical Report SAND2004-6252, Sandia National Laboratories, December 2004.

[37] David F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

Chapter 4

Tetrahedral Mesh Refinement *

4.1 Introduction

The finite volume and finite element methods are two of the most popular techniques used to carry numerical simulations of complex physical phenomena. Regardless of the technique employed, the first step toward obtaining a numerical solution consists of decomposing the computational domain into simpler entities, such as tetrahedra. For the solution schemes to be efficient and accurate, the tetrahedra forming this decomposition must be well-shaped: their dihedral angles must neither be too small nor too large [16, 31]. As a result, tetrahedral mesh generation techniques producing quality elements have been the subject of much scrutiny over recent years.

Generating quality tetrahedral meshes is a challenging task, especially when compared with the equivalent two-dimensional problem. The large number of algorithms that only partially solve the problem are witness to its difficulty. The added complexity comes from various sources. From a practical perspective, one can identify numerical computations, such as those involved in geometric predicates, as being partly responsible. These predicates tend to be less robust in three dimensions and are therefore more prone to returning erroneous results, which can easily cause the meshing algorithm to fail. Fundamentally, the problem of boundary conformity is more involved in 3D. As opposed to 2D, where a triangulation conforming to any

*A version of this chapter will be submitted for publication. Gosselin, S. and Ollivier-Gooch, C. Tetrahedral Mesh Generation with Delaunay Refinement and Non-standard Quality Measures.

set of non-intersecting constraining edges can be constructed, it is possible for a three dimensional geometry to be untetrahedralizable without the addition of Steiner points, one classic example being the Schönhardt polyhedron [30]. Another fundamental problem is the fact that the tetrahedralization of a well-spaced vertex set can contain nearly degenerate elements, known as slivers, which have both very small and very large dihedral angles. These slivers have terrible quality properties and are detrimental to numerical simulations.

The octree-based algorithm of Mitchell and Vavasis is perhaps the earliest three-dimensional meshing method offering a guarantee on the shape of the elements it generates [21, 22]. It produces tetrahedra with bounded aspect ratio, as measured by the ratio of a tetrahedron's longest edge to its smallest altitude. However this bound is not strong enough to be completely satisfactory in practice.

Lattice-based methods are another solution to the problem posed by quality tetrahedral mesh generation. They work by tiling the space in and around a geometric model with tetrahedra, according to predefined stencils. The model's boundaries are then matched by modifying some of the tetrahedra in their neighborhood. Molino et al. [23] proposed an algorithm in which smooth surface models are meshed with tetrahedra having dihedral angles between 13° and 156°. More recently, Labelle and Shewchuk [18] presented what is, to this day, the only algorithm offering a provable bound on dihedral angles that is practically relevant. It generates meshes where all dihedral angles are guaranteed to be between 10.7° and 164.7°. It also has the advantage of being very fast when compared to most other meshing techniques. However, since it has a tendency to smooth sharp features, it is better suited for applications such as physically-based animation than to engineering applications for which boundary conformity is paramount.

Algorithms using the Delaunay refinement paradigm also offer mesh quality guarantees. They have gathered a large following after the pioneering work of Shewchuk [29], in which he demonstrates that Delaunay refinement can generate tetrahedral meshes with bounded circumradius-to-shortest-edge ratios. Unfortunately, according to this quality measure, slivers have

good quality, despite their poor dihedral angles. Consequently, they are not targeted by refinement and many of them remain in the final mesh. Sliver removal has received a lot of attention, with mostly lukewarm results. Chew [9] introduced an algorithm based on randomized vertex insertion coming with a provable bound on dihedral angles. However, this bound is minute and since no implementation that we know of exists, evaluating the method's practical performance is impossible. Sliver exudation, originally proposed by Cheng et al. [6], is another technique used to rid a Delaunay mesh of its slivers. It has successfully been applied to domains bounded by planar surfaces [5, 8], by smooth surfaces [26] and by piecewise smooth surfaces [7]. The guarantee on the smallest dihedral angle proven by Cheng et al. [6] is not practically relevant. However, experiments show that the method outperforms its theoretical guarantees [10]. Nevertheless, a few dihedral angles of less than 5° can remain in the mesh, generally close to its boundary. Labelle provides perhaps the only Delaunay-based algorithm with a strong angle guarantee, eliminating all slivers and leaving behind dihedral angles between 30° and 135° [17]. However, it does not produce a boundary conforming mesh which again, makes the method impractical for simulations involving engineered objects.

In this article, we are partly concerned with exploring the practical capabilities of three-dimensional Delaunay refinement algorithms. After some background information given in Section 4.2, we present, in Section 4.3, an algorithm applying the Delaunay refinement technique to constrained Delaunay tetrahedralizations (CDT) of volumes bounded by piecewise smooth surfaces. To our knowledge, it is the first published algorithm capable of this. Our approach is similar to Si's [32] in the sense that we do not need to maintain a surface triangulation along with our CDT during refinement. The mesh refinement algorithm we propose is able to output tetrahedra matching an automatically-generated sizing function, whose definition is presented in Section 4.4. This sizing field is also used to block infinitely recursive insertions in the neighborhood of small input angles.

In Section 4.5, we demonstrate that, in practice, our algorithm is able to consistently eliminate slivers from CDTs, solely by relying on refinement.

The idea is based on the use of non-standard quality measures, other than the typical circumradius-to-shortest-edge ratio, to drive mesh refinement. Shewchuk hinted that this was indeed possible [29], but an extensive study on the subject has yet to be published. We show that by setting a conservative stopping criterion for a given quality measure, termination is achieved and a mesh comparable in size to one generated with the circumradius-to-shortest-edge ratio is obtained. By refining to improve the minimum sine of a tetrahedron's dihedral angles, we were able to consistently generate meshes with dihedral angles between 18° and 154°. By using a fairer measure, one that detects every class of bad tetrahedra, meshes with dihedral angles between 14° and 154° are obtained. These results are of course conditional to the absence of small angles in the input domain. The dihedral angles we achieve are comparable to those obtained by Labelle and Shewchuk [18], but our results do not come with a theoretical guarantee.

The use of heuristic improvement methods to post-process our Delaunay-refined meshes is also studied in Section 4.6. The algorithm of Freitag and Ollivier-Gooch [12], which combines topological transformations and optimization-based smoothing, is used in the study. The meshes benefit from post-processing to varying degrees. In most cases, we are able to increase the minimum dihedral angle above 20°. We observe that the quality of the post-processed meshes is somewhat independent of the quality measure employed during Delaunay refinement. This is consistent with Ollivier-Gooch and Boivin who concluded that the vertex distribution in Delaunay-refined meshes lends itself well to improvement methods [24]. Our post-processed meshes are not quite as good as those obtained by Alliez et al. [1] or by Klingner and Shewchuk [15], but the algorithm we use has the advantage of being faster.

We conclude the article by describing, in Section 4.7, some implementation issues regarding the geometry interface and vertex insertion. We also give some timing data for our algorithm.

4.2 Preliminaries

The mesh refinement algorithm presented herein accepts a constrained Delaunay tetrahedralization (CDT) as input. The CDT is accompanied by its underlying geometry which describes the domain to be meshed. The geometry is supplied in the form of a boundary representation, allowing for full topological adjacency queries. The input domain is a Piecewise Smooth Complex (PSC), as defined by Cheng et al. [7].

Definition 4.1. A *Piecewise Smooth Complex* is the union of a set of points $\mathcal{P}$, a set of curves $\mathcal{C}$ and a set of orientable surfaces $\mathcal{S}$ embedded in $\mathbb{R}^3$ such that:

- Each element forming the PSC is a smooth manifold.
- A point in $\mathcal{P}$ can only intersect a curve in $\mathcal{C}$ at one of its endpoints. A point can intersect a surface in $\mathcal{S}$ on its boundary or be embedded into the surface.
- A curve in $\mathcal{C}$ is either non-closed and bounded by two endpoints in $\mathcal{P}$, or is closed having both ends connected to a single endpoint in $\mathcal{P}$. Furthermore, two curves in $\mathcal{C}$ can only intersect each other at an endpoint common to both curves.
- A surface in $\mathcal{S}$ is either closed or entirely bounded by curves in $\mathcal{C}$. In both cases, the surface can contain embedded points in $\mathcal{P}$ and embedded curves in $\mathcal{C}$, that are not part of its boundary. A surface bounded by a curve c in $\mathcal{C}$ must also contain the endpoints of c on its boundary (that is a surface cannot be bounded by a curve that extends beyond the boundary of the surface). Two surfaces in $\mathcal{S}$ can intersect each other along one or more common curve in $\mathcal{C}$ or common points in $\mathcal{P}$.

We require the input geometry to be surface-bounded — the boundary separating the domain to be meshed from its exterior must be entirely covered by surface patches. Note that if all curves in the PSC are straight

line segments and all surfaces are planar, then the domain is a Piecewise Linear Complex (PLC) [20], the input accepted by previous constrained Delaunay refinement algorithms [29, 32]. In our implementation, the Common Geometry Module library [33] is used as back end for constructing, modifying and querying the geometric models.

During mesh refinement operations, boundary curves are approximated by piecewise linear segments while boundary surfaces are represented by triangular facets. Given v_1 and v_2, two successive vertices along a curve c of the PSC, we name the segment of c bounded by v_1 and v_2 a *subcurve*. We use *boundary edge* to describe the line segment $\overline{v_1 v_2}$. The term *boundary facet* designates a planar triangle approximating a part of surface patch s; the corresponding geodesic triangle is called *subsurface*.

Correctly solving the problem of boundary conformity is paramount when generating meshes to be used for simulations on engineered objects. The mesh must not only conform to the input's boundary, but must also capture all of its features. One way to solve this problem is through the use of CDTs. When compared with other methods, CDTs generally have the advantage of achieving conformity without requiring as many vertices. Yet, they still allow for the application of Delaunay refinement algorithms. As illustrated in Figure 4.1, a simplex is said to be constrained Delaunay if it respects the boundary of the PSC and if there exists a sphere circumscribing the simplex such that the sphere does not contain any vertex visible from the simplex's interior [30]. We consider a tetrahedron to be respecting the PSC's boundary if none of its edges crosses a boundary facet and none of its faces cuts through a boundary edge. A CDT is a tetrahedralization where every tetrahedron satisfies the constrained Delaunay property. The problem of constructing a CDT from a PSC has been the subject of previous work [13] and is not covered here.

Delaunay refinement algorithms are known to output provably good triangulations whose vertex spacing is guaranteed to be within a constant factor of optimal. To prove this classical result, Ruppert introduced a function to measure this desired spacing called the *local feature size* [28]. Although explicit computation of the local feature size is not necessary to

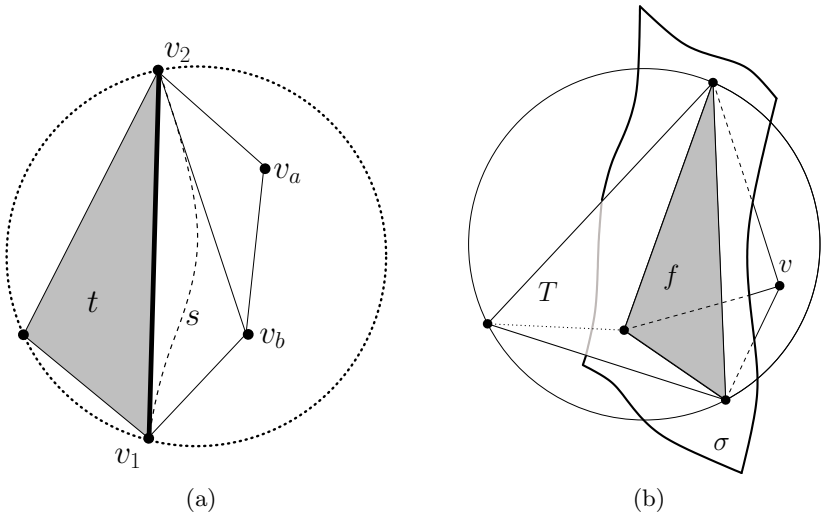


Figure 4.1: Constrained Delaunay simplices. (a) A two-dimensional example: The circumcircle of triangle t contains vertices v_a and v_b, but they are occluded from t by boundary edge $\overline{v_1 v_2}$, a discrete approximation of subcurve s. (b) Tetrahedron T is constrained Delaunay. v is inside T's circumsphere, but is not visible from T's interior. Visibility is occluded by boundary facet f on surface σ. Note that in 3D, boundary edges do not occlude visibility.

the execution of the algorithm, its knowledge provides the means to define an automatic sizing field which allows for better control over mesh element size and grading; such a sizing field is presented in Section 4.4.

For geometries bounded by planar surfaces, the optimal vertex spacing is entirely dependent on the input domain itself. However, when dealing with curved boundaries, the spacing also depends on the PSC's initial discretization i.e. its initial CDT. Since the initial CDT is in fact a piecewise linear approximation of the input domain, where every boundary edge can be seen as an input segment and every boundary facet as an input facet, we use the following definition of the local feature:

Definition 4.2. Local feature size. Given a PSC X and its *initial* CDT D, the *local feature size* at a point p, noted $\mathrm{lfs}(p)$, is the radius of the smallest ball centered at p that intersects two points that lie on non-adjacent vertices

of D, or on any subcurves or subsurfaces of X.

4.3 Delaunay Refinement Algorithm

Delaunay refinement describes a family of algorithms capable of generating meshes whose properties are favorable for use in numerical simulations. Starting from a Delaunay (or possibly a constrained Delaunay) tetrahedralization, they incrementally insert new vertices to it, with the objective of improving its quality and possibly matching a specified sizing field. The (constrained) Delaunay property is maintained after each insertion.

The algorithm we use operates on constrained Delaunay tetrahedralizations (CDT). Three high level operations are repeated until none of them apply. These operations are essentially the same as the ones proposed by Shewchuk [29], with slight modifications, notably to account for curved boundaries:

1. *Subcurve split*: if a subcurve is encroached or is too long with respect to the sizing field, it must be split, thereby creating two new subcurves. The new vertex location is chosen such that the total variation of the tangent angle over the new subcurves is equal. If the newly-inserted vertex encroaches on any other subcurves, those should be split as well.

2. *Subsurface split*: if a vertex encroaches on a subsurface or if the subsurface is too large with respect to the sizing field, then split the subsurface. A subsurface is split by inserting a vertex at the projection of its corresponding boundary facet's circumcenter onto the curved surface. If the proposed insertion location encroaches upon any subcurves, then the vertex is not inserted, instead encroached subcurves are split. If the subsurface still exists after subcurve splitting, the subsurface is split afterwards.

3. *Tetrahedron split*: if a tetrahedron is of poor quality or is too large with respect to the sizing field, then insert a vertex at its circumcenter *unless* that location would encroach on any subsurfaces or subcurves,

in which case the circumcenter is not inserted. Instead the encroached boundary entities are split. If the tetrahedron survives the boundary entity splits, then the tetrahedron is split.

Subcurve split operations have priority over subsurface splits, which in turn have priority over inserting at the circumcenter of a tetrahedron. We define a subcurve's diametral sphere to be the unique, smallest sphere that contains the subcurve's underlying boundary edge. A subcurve is encroached if a vertex lies inside or on this diametral sphere. A subsurface is encroached when a vertex, other than its endpoints, is located inside or on its equatorial sphere, the smallest possible sphere passing through three vertices. We prefer protecting subsurfaces with equatorial spheres over the more involved equatorial lenses. Although lens protection provides a better provable bound on circumradius-to-shortest-edge ratio, it is more difficult to implement. When a lens-protected subsurface is split, the interior vertices located inside the subsurface's equatorial sphere must first be deleted. We have observed that it is sometimes impossible to maintain a valid CDT after deleting those vertices. Our choice is further justified by the fact that the practically reachable radius-edge ratio is the same for either definition of encroachment. As is the case for the two-dimensional Delaunay refinement, the practical bound is well below the theoretical bound. The only possible drawback is perhaps a slight increase in mesh size when using sphere protection.

Small input angles pose a threat to the refinement algorithm's termination. Dihedral angles of less than 90° tend to create edges that get shorter and shorter as the algorithm tries to eliminate the poor quality tetrahedra adjoining them. As such, they can be seen as edge length reducers; the argument that vertex spacing remains within a constant factor of optimal no longer holds in their neighborhood. As a result termination can no longer be guaranteed.

To prevent infinite refinement from occurring close to small input angles, we rely on a sizing function $LS(p)$, defined at every point p of the geometric domain (details on the computation of this sizing function are given in the next section). A new vertex v will only be inserted in the mesh if the local

spacing is sparse with respect to the sizing function. More specifically, v is only inserted if

$$\frac{1}{n}\sum_{i=0}^{n} LS\left(v_i\right) > \frac{2\lambda}{\sqrt{n-1}}r \tag{4.1}$$

where n is the number of vertices the mesh entity has (2 for a subcurve, 3 for a subsurface and 4 for a tetrahedron), $LS\left(v_i\right)$ is the value of the sizing function at each of the entity's vertex, λ is a user-defined constant and r is a measure of length specific to each entity. It is either the length of a subcurve's underlying boundary edge, the circumradius of a subsurface's boundary facet or the circumradius of a tetrahedron.

4.4 Sizing Field

The initial CDT given as input to the Delaunay refinement algorithm exclusively contains boundary vertices (vertices located on curves or on surfaces). The input tetrahedralization can therefore be used as a search structure to quickly evaluate the local feature size at each of these vertices. Knowledge of the lfs over the domain allows defining a sizing field, which is used to automatically control the size of the elements in the mesh output by the algorithm. Ollivier-Gooch and Boivin [24] introduced a sizing field definition depending on lfs and on two parameters, defined at run-time by the user. Since their sizing field is a Lipschitz continuous function, it can be shown that the quality guarantees offered by previous algorithms also apply to theirs. In this section, we show how their approach can be modified for the boundary and interior vertex spacing to be controlled more independently.

As is hinted above, definition of the sizing field begins by computing lfs at every vertex of the initial CDT. This is a fairly straightforward procedure. Given a boundary vertex v, evaluating lfs(v) only requires information on the elements adjoining v in the CDT. Therefore, all search operations are therefore local and computationally inexpensive. According to Definition 4.2, lfs (v) is chosen to be the minimum between the following:

- The length of the shortest mesh edge (including boundary edges)

connected to v.

- Among the boundary facets adjoining v, find the edges $\mathcal{E}$ *not* connected to v. Pick the shortest distance to an edge $e \in \mathcal{E}$. If e is a boundary edge, use the shortest distance to the underlying subcurve instead.
- Among the tetrahedra adjoining v, find the boundary facets $\mathcal{F}$ *not* connected to v. Pick the shortest distance to a subsurface underlying a boundary facet $f \in \mathcal{F}$.

Once lfs is known at every boundary vertex, the same function as the one used in Ollivier-Gooch and Boivin's original scheme is used to compute the value of a length scale at these vertices, noted $LS(v)$:

$$\widetilde{LS}(v) = \min_i \left(LS(w_i) + \frac{|vw_i|}{G} \right), \; w_i \in \text{neighbors of } v \tag{4.2}$$

$$LS(v) = \min \left(\frac{\text{lfs}(v)}{R}, \widetilde{LS}(v) \right), \tag{4.3}$$

where $|\cdot|$ is the Euclidean distance, R is a boundary refinement factor and G is a grading constant, controlling how fast the size of tetrahedra can change over a certain distance. The vertices' length scales define the sizing field.

When applying Delaunay refinement, a boundary edge is considered too long — its subcurve therefore needs to be split — if its length is greater than the sum of the length scales at its vertices. A boundary facet is deemed to be too large whenever the ratio of its circumradius to the average length scale at its vertices is greater than $\frac{\sqrt{2}}{2}$, thereby triggering a subsurface split. Finally, a tetrahedron is too large and a vertex is inserted at its circumcenter when the ratio of its circumradius to the average length scale at its vertices is greater than $\frac{\sqrt{3}}{2}$. The effects of the sizing on mesh refinement are depicted in Figure 4.2.

When a new vertex is inserted in the interior of the CDT, its length scale is computed using only Equation 4.2. The interior vertex spacing therefore depends exclusively on the grading constant and on the density of the boundary discretization. To compute the length scale of a vertex resulting from a subcurve or a subsurface split, the procedure is slightly

different. For a subcurve, a ceiling value is first set by linear interpolation: using the length scale of the subcurve's vertices, the maximum possible length scale for the new vertex is interpolated along the subcurve's arc length. In the case of a subsurface split, the insertion will create a set of new boundary facets adjoining the new vertex. A weighted average of the length scale at these boundary facets' vertices is used to set the ceiling value. The weights are chosen as the inverse of the distance for the new vertex to the boundary facets' vertices. In both cases, the length scale is taken to be the minimum between the ceiling value and the graded length scale, as computed by Equation 4.2.

The proposed sizing field scheme is simpler to compute and consequently faster than the original. Because we do not use the refinement factor for interior vertices, there is no need for them to store information about nearby boundary entities as is done in the original method. This results in memory savings and simplified bookkeeping.

4.5 Non-standard Quality Measures

Delaunay refinement algorithms are notorious for generating meshes containing sliver tetrahedra (Figure 4.3). Indeed, based on the taxonomies of Bern et al. [3] and Cheng et al. [6] (the latter being reproduced in Figure B.1 of Appendix B), every type of poorly-shaped tetrahedra, except slivers, can provably be eliminated from a mesh by Delaunay refinement. With four dihedral angles close to 0° and two approaching 180°, slivers are clearly detrimental to numerical simulations. While extensive work has been devoted to the design of post-processing techniques for removing slivers and other types of bad tetrahedra from meshes [6, 12, 15], very few attempts have been made to generate sliver-free meshes by solely relying on circumcenter insertions.

The provable quality bound accompanying Delaunay refinement is typically measured in terms of a tetrahedron's circumradius-to-shortest-edge ratio. This measure is naturally improved by Delaunay refinement and is at the basis of the compaction argument used to prove the algorithm's termination. While this ratio is high for most types of bad tetrahedra, it can also be

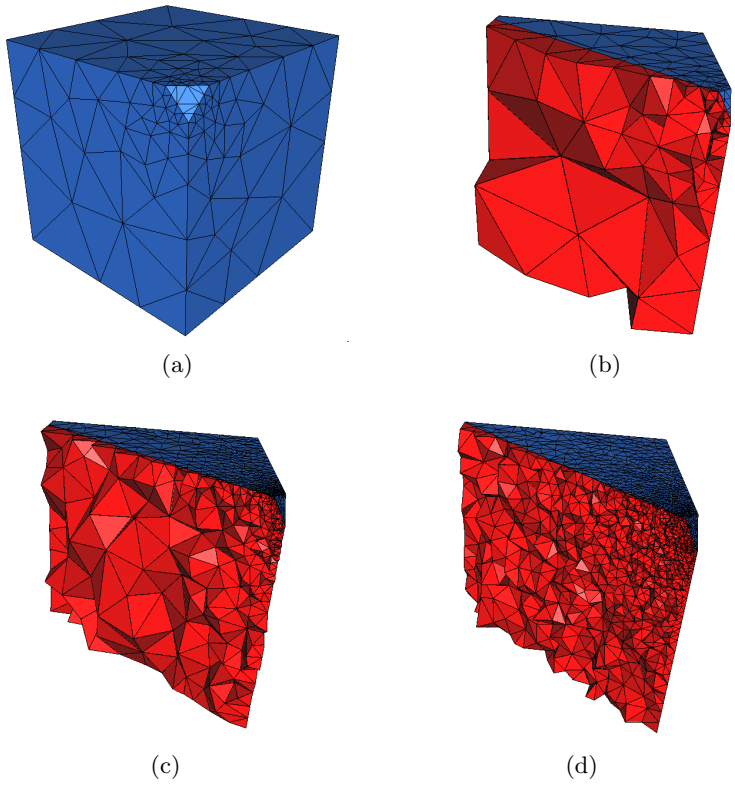


Figure 4.2: Illustration of the effects of the sizing field on the refined mesh. (a) Surface triangulation of the input CDT. (b) Refined mesh with $R = G = 1$. (c) Refined mesh with $R = 3$ and $G = 1$. (d) Refined mesh with $R = 3$ and $G = 15$.

very low for slivers, well below theoretical — or even practically reachable — bounds. Since the algorithm only targets those tetrahedra it considers bad, slivers are skipped because of their good circumradius-to-shortest edge ratio. As a result, many of them remain in the refined mesh. Nevertheless, the ability of Delaunay refinement to eliminate tetrahedra from a mesh begs the following question: would Delaunay refinement be capable of eliminating all poor-quality tetrahedra from a mesh if a quality measure encompassing all of them was utilized?

Some preliminary results presented by Shewchuk [29] indicate that re-

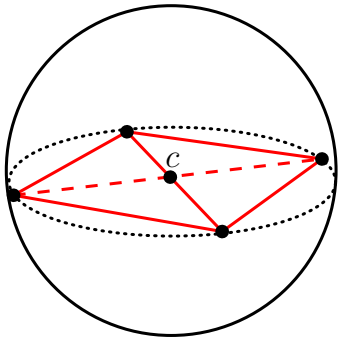


Figure 4.3: A *sliver* tetrahedron is formed by four nearly coplanar vertices, equally spaced around the circumsphere's equator. Its circumradius-to-shortest edge ratio can therefore be as low as $\frac{\sqrt{2}}{2}$. However, since its volume can be arbitrarily close to zero, it is considered appalling by many other measures.

finement can potentially be driven by quality measures other than the circumradius-to-shortest-edge ratio — he uses the minimum dihedral angle of a tetrahedron — and still terminate, provided the quality target is set reasonably. In this section, we push this idea further by testing various non-standard measures, some of them detecting all types of bad tetrahedra, in an attempt to determine how mesh quality can be improved by circumcenter refinement only. The aim is to establish the practical capabilities of Delaunay refinement algorithms, theoretical considerations aside.

4.5.1 Quality Measures Definitions

During a typical refinement run, every tetrahedron τ, either present in the initial mesh or created later by vertex insertion, is assigned a score, $q(\tau)$, according to some quality measure. As can be seen in Figures B.2 to B.9, the eight different measures we study give a higher score to well-shaped tetrahedra and are normalized so that the possible scores range from zero to one. When τ scores below a certain threshold, it is queued to be split so that an attempt will be made to eliminate it from the mesh by inserting a vertex at its circumcenter. The quality measures we use are:

- *Shortest-edge-to-circumradius ratio*: $q_1 = \sqrt{\frac{3}{8}}\frac{L_{\min}}{R}$. The normalized inverse of the quality measure traditionally used by Delaunay refinement [29]. Figure B.2 clearly shows that slivers obtain a high score according to this measure. Despite this fact, we know that the refined mesh obtained by using this measure will be coarse and nicely graded. It is therefore a good baseline case for mesh cardinality comparison.

- *Minimum sine of a tetrahedron's dihedral angles*: $q_2 = \min_{0\leq i\leq 5}(\sin\phi_i)$, where ϕ_i are the six dihedral angles of a tetrahedron. This measure has the advantage of targeting tetrahedra having either small or large dihedral angles. However, as Figure B.3 shows, it does not detect needle-shaped tetrahedra with good dihedral angles (spires and to a lesser extent, spindles). These might not be harmful to numerical methods and can be useful for anisotropic simulations. Freitag and Ollivier-Gooch [12] report their mesh improvement algorithm to perform best when using this measure.

- *Volume-length measure*: $q_3 = 6\sqrt{2}\frac{V}{L_{\text{rms}}^3}$, where V is the volume of the tetrahedron and L_{rms} is the l^2-norm of its six edge lengths. Klingner and Shewchuk [15] find this measure to be very effective at improving meshes. Using this measure, Delaunay refinement will target every type of bad tetrahedra.

- *Inradius-to-circumradius ratio*: $q_4 = 3\frac{r}{R}$. This measure detects all types of bad tetrahedra, but can also be unstable numerically. Computation of a tetrahedron's circumradius requires the evaluation of the determinant of a matrix whose condition number increases as the tetrahedron becomes flatter.

- *Minimum solid angle of a tetrahedron*: $q_5 = \frac{\min_{0\leq i\leq 3}(\Omega_i)}{3\arccos(1/3)-\pi}$, where Ω_i are the four solid angles of a tetrahedron. This measure also detects all types of bad tetrahedra. Note that the denominator is a scaling factor.

- *Mean ratio*: $q_6 = \frac{12(3V)^{\frac{2}{3}}}{\sum_{0\leq i<j\leq 3} L_{ij}^2}$. A quality measure proposed by Joe [14], it also detects all types of bad tetrahedra.

- *Inradius-to-maximum-altitude ratio*: $q_7 = 4\frac{r}{H}$. One of numerous measures studied by Field [11]. According to Figure B.8, the measure only recognizes a few types of poorly-shaped tetrahedra. It is nonetheless included in this study for comparison purposes, to see if the meshes are worse than with the shortest-edge-to-circumradius ratio for instance.

- *Minimum altitude-to-longest-edge ratio*: $q_8 = \sqrt{\frac{3}{2}}\frac{L_{\max}}{h}$. The inverse of the aspect ratio measure by which Mitchell and Vavasis [22] evaluate the quality bound of their octree-based algorithm. Another measure targeting all classes of bad tetrahedra.

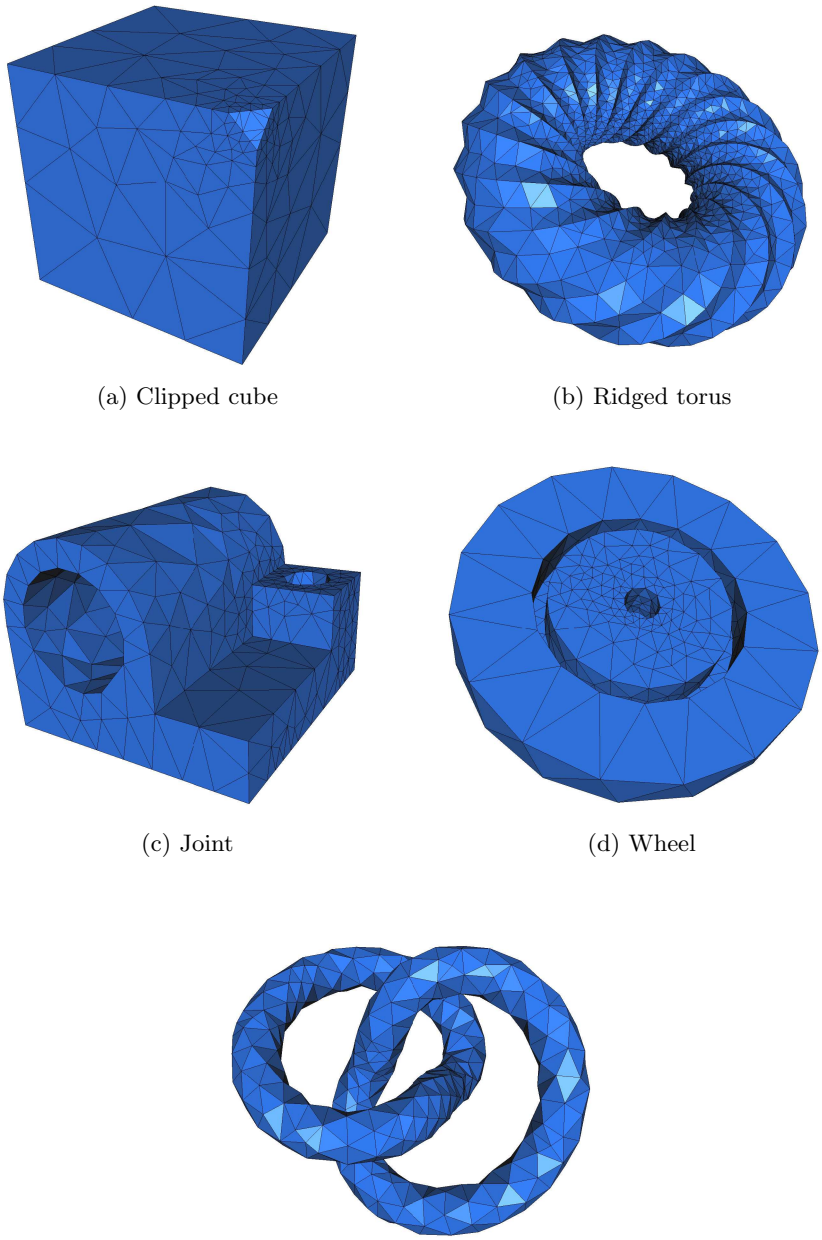
According to the plots of Figures B.2 to B.9, five of those eight measures are equivalent in the sense that they will assign a score approaching zero to every possible type of bad tetrahedra while they will give a score of one to a regular tetrahedron. Even though they do not necessarily approach zero or one at the same rate for every class of bad tetrahedra, they are still expected to produce meshes exhibiting similar quality properties.

4.5.2 Establishing Quality Thresholds

Before being able to use any of the aforementioned quality measures to schedule vertex insertion, threshold scores must first be established for each of them. A tetrahedron assigned a score higher than this threshold will never be queued for circumcenter insertion. If on the other hand the tetrahedron receives a score below the threshold, an attempt will be made at splitting it.

The threshold values are determined experimentally, by running the Delaunay refinement algorithm of Section 4.3 for an extended period. The procedure starts from the initial CDT of the five geometries shown in Figure 4.4. Vertices are then incrementally inserted, always targeting the tetrahedron with the worst quality score still remaining in the mesh. The sizing field is turned off so as to not affect the insertion order or to block any attempted insertion if some of the mesh elements become too small.

The experiment is repeated for each of the five geometries, using every proposed quality measure. After each insertion, the score of the tetrahedron



(a) Clipped cube

(b) Ridged torus

(c) Joint

(d) Wheel

(e) Rings

Figure 4.4: The five geometric models used to determine the quality score thresholds. The surface triangulations of the initial CDTs are depicted shown.

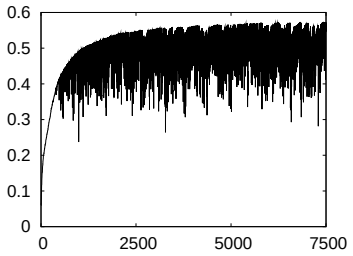
Model	q_1	q_2	q_3	q_4
Clipped cube	0.591	0.351	0.399	0.444
Ridged torus	0.572	**0.349**	**0.347**	**0.393**
Joint	0.583	0.377	0.377	0.443
Wheel	0.590	0.359	0.378	0.429
Rings	**0.570**	0.383	0.392	0.442

Model	q_5	q_6	q_7	q_8
Clipped cube	0.243	0.542	0.637	0.298
Ridged torus	**0.227**	**0.494**	**0.617**	**0.262**
Joint	0.244	0.522	0.625	0.281
Wheel	0.233	0.523	0.622	0.271
Rings	0.242	0.536	0.623	0.295

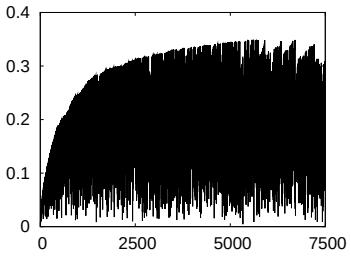
Table 4.1: Maximum quality scores achieved for each model during the experiments to establish the refinement thresholds. The values in bold are the lowest maxima for each quality measure.

having the worst quality is recorded. Plots similar to those presented in Figure 4.5 are obtained for each geometry — for the sake of brevity, we limit ourselves to presenting results for the ridged torus model. Table 4.1 summarizes the maximum scores achieved over each run.

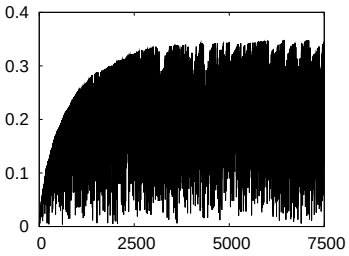
Something that is immediately apparent from the plots of Figure 4.5 is that vertex insertion is prone to creating new tetrahedra having particularly bad quality, mostly slivers as is hinted by Figures 4.5a and 4.5g. According to these two specific plots, an insertion does not appear to generate new tetrahedra that are as poor as those created in any other case. However, the quality measures used in these two plots simply do not perceive slivers as having poor quality — a large number of new slivers are generated, they are just not considered of poor quality according to the quality measures. As Delaunay tetrahedralizations have an inherent tendency to harbor slivers, this behavior is not surprising. On the other hand, the results also indicate that only a few additional insertions are generally sufficient to remove the slivers and to return the worst element quality score back to the nominal value. This observation is crucial regarding mesh cardinality: assuming the quality threshold is set reasonably, it shows that in practice, a quality mesh



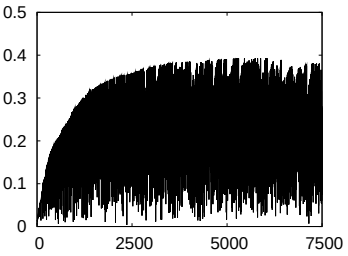
(a) Shortest edge-to-circumradius



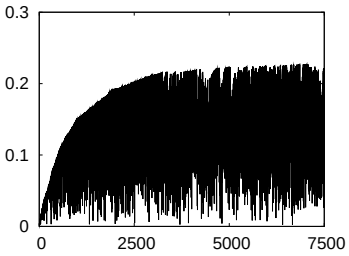
(b) Min. sine of dihedral angles



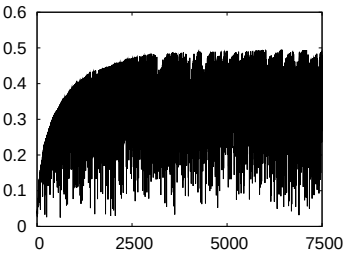
(c) Volume-length measure



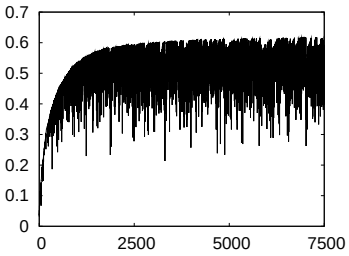
(d) Inradius-to-circumradius ratio



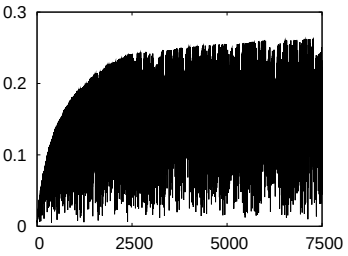
(e) Minimum solid angle



(f) Mean ratio

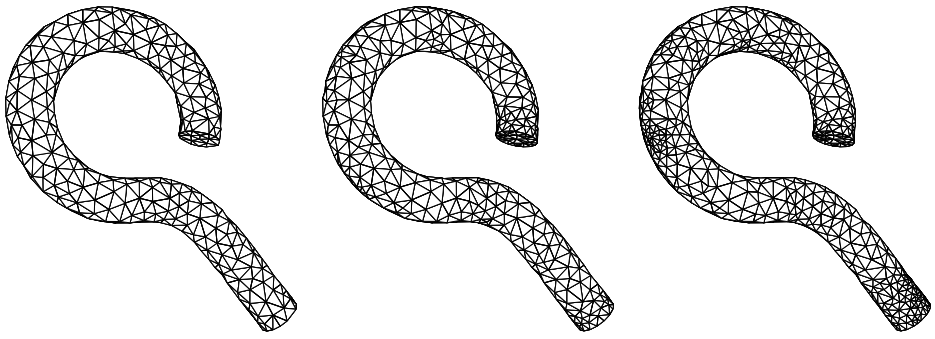


(g) Inradius-to-max. altitude ratio



(h) Long edge-to-min. altitude ratio

Figure 4.5: The evolution of the worst quality score according to various measures, recorded over after each of the first 7500 vertex insertions inside the ridged torus model.



(a) (b) (c)

Figure 4.6: Illustration of the effects of setting the quality threshold too high. The surface triangulation of the model is shown after refinement, using using different values of q_2^t. (a) The initial CDT, before refinement. (b) The final mesh, refined with $q_2^t = 0.3$ (minimum dihedral angle $\approx 18°$). (c) The final mesh, refined with $q_2^t = 0.37$. Some overrefined regions are clearly apparent on the surface. The algorithm does not terminate if q_2^t is set to 0.39 or higher.

can be generated using non-standard measures without overrefined regions.

The data presented in Figure 4.5 and in Table 4.1 allow us to establish threshold values for each studied measure. The thresholds, which we note q_i^t, must be chosen carefully, remembering that our aim remains to achieve the best possible quality while keeping the mesh as coarse as possible. These are two competing requirements: the higher the quality targets are, the more vertices will need to be inserted to reach them. If the bar is set too high, the mesh grading will start to suffer and as a result, unnecessarily small edges will start appearing in the mesh. When the target reaches a certain tipping point, termination will become impossible; refinement alone cannot improve mesh quality beyond that point. As can be observed in Figure 4.6, when the threshold is set conservatively, the surface triangles roughly remain uniformly sized as in the input CDT. However, when approaching the critical value past which termination becomes impossible, clusters of smaller surface triangles begin to appear, indicating that certain parts of the mesh are overrefined.

The plots in Figure 4.5 show that, for all measures studied, rapid gains

q_1^t	q_2^t	q_3^t	q_4^t	q_5^t	q_6^t	q_7^t	q_8^t
0.52	0.32	0.32	0.36	0.21	0.47	0.59	0.24

Table 4.2: The refinement thresholds chosen for each quality measure.

in quality are realized with the first few insertions. The rate of improvement then tapers off. A plateau is finally reached at a value approximately equal to the maximum achievable quality score. Based on this observation and keeping in mind that we aim for a trade-off between mesh quality and coarseness, we choose our refinement thresholds so that they fall in between the rapid improvement region and the quality plateau. We consider this choice to be conservative enough to preserve good grading properties and yet to allow for the generation of high-quality meshes. The chosen values are summarized in Table 4.2.

4.5.3 Results

With the various quality thresholds properly established, the Delaunay refinement algorithm can now be executed on the test geometries of Figure 4.4. Each of the input CDTs is refined, successively using the eight quality measures to schedule vertex insertion. The stopping criterion is reached once the worst tetrahedron in the mesh achieves a quality score equal to or higher than the threshold value of the measure employed. Tables summarizing the refined meshes' quality data are presented in Appendix B. The results are compared using the dihedral angles (both minimum and maximum), the inradius-to-circumradius ratio and the volume-length measure.

Our main observation is that we are able, for every test run, to reach the established quality target using a number of vertices comparable to the number required when refining based on the shortest-edge-to-circumradius ratio. This allows us to answer the question we asked at the beginning of this section: Delaunay refinement can indeed be used in conjunction with non-standard quality measures to improve meshes and to eliminate slivers. The results show that we can consistently generate meshes containing tetrahedra having dihedral angles between 18° and 154° when the refinement directly

targets elements with bad dihedral angles. Note that on most models, better extremal angles can be achieved (the minimum dihedral angle can be as high as 22°). However, these higher values cannot be achieved consistently, without resorting to inserting a prohibitively large number of vertices. We must also point out that the good dihedral angles observed are conditional to an input geometry not containing very small dihedral angles or curves subtending small face angles. Obviously, the refined mesh will contain angles at least as small as the smallest input angle in such cases.

As was mentioned earlier, five of the studied measures detect all possible classes of bad tetrahedra. As such, we would expect these measures to perform similarly. In practice, we observe that three of these measures seem to have a slight edge, namely the volume-length measure (q_3), the inradius-to-circumradius ratio (q_4) and the mean ratio (q_6). The volume-length measure and the mean ratio are two very similar quality measures as they are both ratios of a tetrahedron's volume to its edge lengths, treated slightly differently. Our results indicate the mean ratio has a slight advantage over other measures. However, the chosen refinement threshold for the mean ratio leads to meshes having slightly more vertices. This can explain the better perceived performance. In all case, meshes generated using these measures have dihedral angles between 14° and 154°. Although the observed dihedral angles are not as good as when refining to directly remove tetrahedra with bad dihedral angles, the tetrahedra left in the mesh are "rounder" as indicated by a worst radius ratio of 0.28 and a worst volume-length measure of 0.26 (compared to 0.18 and 0.22 respectively for dihedral angle refinement). For practical applications, we recommend the use of the volume-length measure or the mean ratio over the inradius-to-circumradius ratio since the latter is numerically unstable for nearly-flat tetrahedra.

The minimum altitude-to-longest-edge ratio (q_8) slightly underperforms relative to the three measures mentioned above. The dihedral angles are between 13° and 157° while the minimum radius ratio and volume-length measure are 0.23 and 0.24 respectively. Refining based on the minimum solid angle (q_5) yields deceiving results. Although the measure targets every possible type of bad tetrahedra, it does not perform as well as other measures

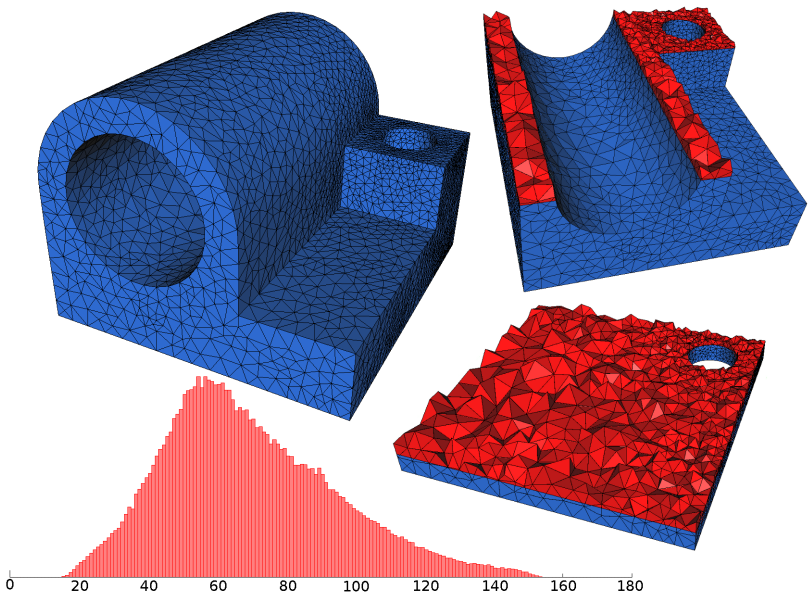


Figure 4.7: Views of the joint model refined using the volume-length measure, with $R = 2$ and $G = 10$, accompanied by the dihedral angle distribution. The dihedral angles are between 15° and 154°. The mesh has 9992 vertices and 44397 tetrahedra.

as evidenced by the dihedral angles ranging from 10° and 163° and a worst observed volume-length measure of 0.17.

As expected, the shortest-edge-to-circumradius and inradius-to-maximum-altitude ratios perform poorly, clearly leaving slivers in the final mesh. Consequently, the worst observed radius ratio and volume-length measure are also very low. Even though the inradius-to-maximum-altitude ratio only targets a few types of poorly-shaped tetrahedra, the quality of the meshes obtained using this measure is comparable with those generated by the shortest edge-to-circumradius ratio, at least when judged based solely on extremal dihedral angle values.

Figures 4.7 and 4.8 present quality meshes generated by Delaunay refinement. As opposed to the results in Section B.2, the meshes in these figures were obtained with a refinement factor different from 1. Despite this, the

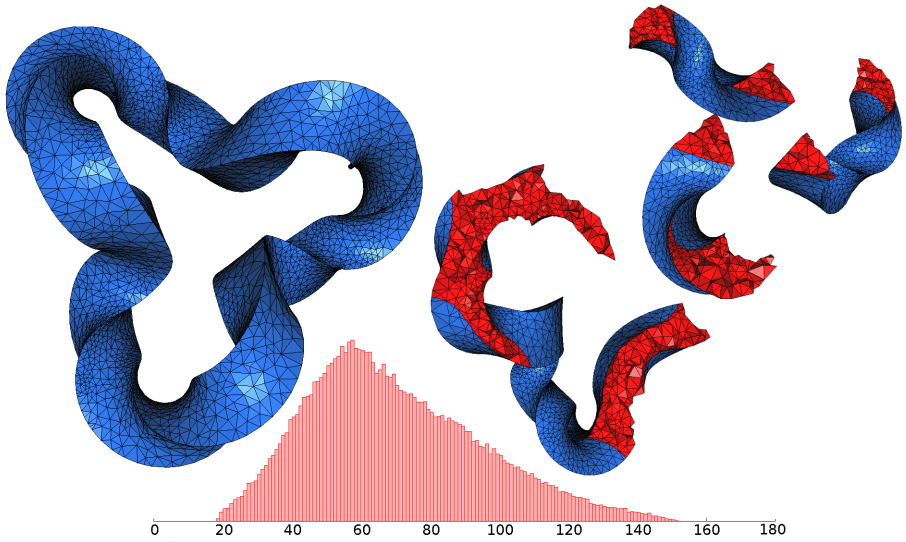


Figure 4.8: Views of the twist model refined using the minimum sine of the dihedral angles, with $R = 2$ and $G = 1$ and $\lambda = 5$, accompanied by the dihedral angle distribution. The minimum input dihedral is about 40°. The dihedral angles are between 19°and 154°. The mesh has 6737 vertices and 25720 tetrahedra.

extremal values of dihedral angles are still within the same limits as those observed using $R = G = 1$. However, we should note that, for the twist model of Figure 4.8, this result is a consequence of the value of λ. In fact, the input geometry contains small dihedral angles, the smallest being around 40°. To prevent infinitely recursive refinement, the sizing field had to stop insertions, according to equation 4.1 (for this particular example, we used $\lambda = 5$). The last allowed insertion in the neighborhood of the small angle, before the tetrahedra size became too small with respect to the sizing field, might possibly create a sliver or another type of bad tetrahedron. Since no more insertions are permitted in this region, this tetrahedron violating the quality threshold will survive in the refined mesh. For example, when λ is set to 6, the smallest dihedral angle left in the refined mesh is 11°. The mesh will need to be post-processed to remove the small dihedral angles left close to small input angles.

Our algorithm outperforms Si's TetGen [32] when extremal dihedral angles are compared. To make the fairest possible comparison, we give the surface triangulations of our initial CDTs as input to TetGen 1.4.2 (all models of Figure 4.4 plus the twist model). On all tested models, the meshes generated by TetGen have dihedral angles between 5.4°and 166.7°, both extremal values occurring with the ridged torus models. The best results are unsurprisingly achieved on the clipped cube, with dihedral angles ranging from 9.7°and 159.6°.

4.6 Post-processing

In the previous section, we demonstrated that it is possible, in practice, to generate meshes with good dihedral angles solely by relying on Delaunay refinement. These meshes could potentially benefit from the application of mesh improvement methods in a post-processing step. Some very successful algorithms have already been proposed. In our opinion, the best are from Freitag and Ollivier-Gooch [12], Alliez et al. [1] and Klingner and Shewchuk [15]. The algorithm of Freitag and Ollivier-Gooch is already integrated within the GRUMMP libraries. It is therefore an obvious choice to verify how improvement algorithms fare when operating on Delaunay-refined meshes and whether or not the choice of quality measure used during refinement has an impact on the resulting mesh.

Freitag and Ollivier-Gooch's implementation relies on a combination of topological transformations and vertex smoothing to achieve its goal. Topological transformations locally reconfigure the mesh with the objective of finding the connectivity offering the best possible quality. Operations include the face flips depicted in Figure 4.9 as well as edge swaps, which replace n tetrahedra with $2n-4$ tetrahedra. Only configurations up to $n = 7$ are currently implemented — it is the authors' opinion that only marginal gains can be made with edge swaps involving more tetrahedra, this opinion not being shared by Klingner and Shewchuk. Smoothing, on the other hand, maintains constant connectivity but moves vertices about, with the objective of improving the adjacent tetrahedra's quality. Gradient-based optimization

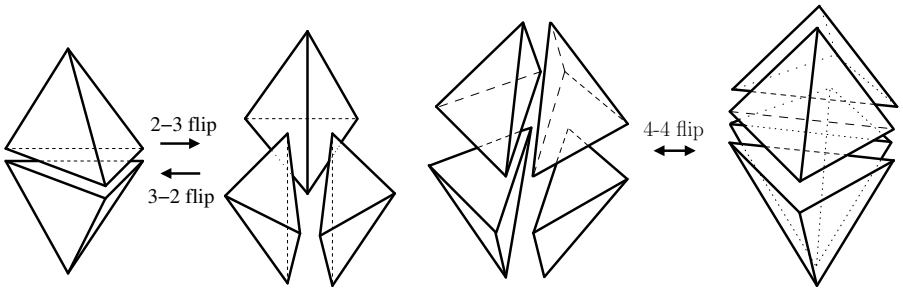


Figure 4.9: The face flipping operations used by the mesh improvement algorithm. The topological transformations employed also include edge swapping not illustrated here.

is used to determine in which direction and by how much each vertex should be moved to get the best possible improvement in quality. The authors have demonstrated that smoothing and swapping perform best when used concurrently. Note that in the current implementation, mesh improvement operations are only utilized in the domain's interior; boundary faces and vertices remain untouched.

Among the three aforementioned algorithms, Freitag and Ollivier-Gooch's has the distinct advantage of being the fastest, at least on the basis of published timed runs. As opposed to the method of Alliez et al., it keeps boundary vertices exactly on the boundary, a property essential to the applications we are targeting our meshes for. It can also respect a sizing field, if one is specified. Klingner and Shewchuk's approach is currently incapable of that. However, since the algorithm relies on local optimization for vertex smoothing, it has a tendency to remain stuck at local minima, the set of topological transformations not being rich enough to get it unstuck. As a result, the quality levels are not as good as those obtained by the other two methods.

In our experiments, we use, as a starting point, meshes refined with three different quality measures: the usual shortest-edge-to-circumradius ratio, the minimum sine of the dihedral angles and the volume-length measure. The set of topological transformations is then applied, with the objective of maximizing the minimum sine of the dihedral angles. Two passes are

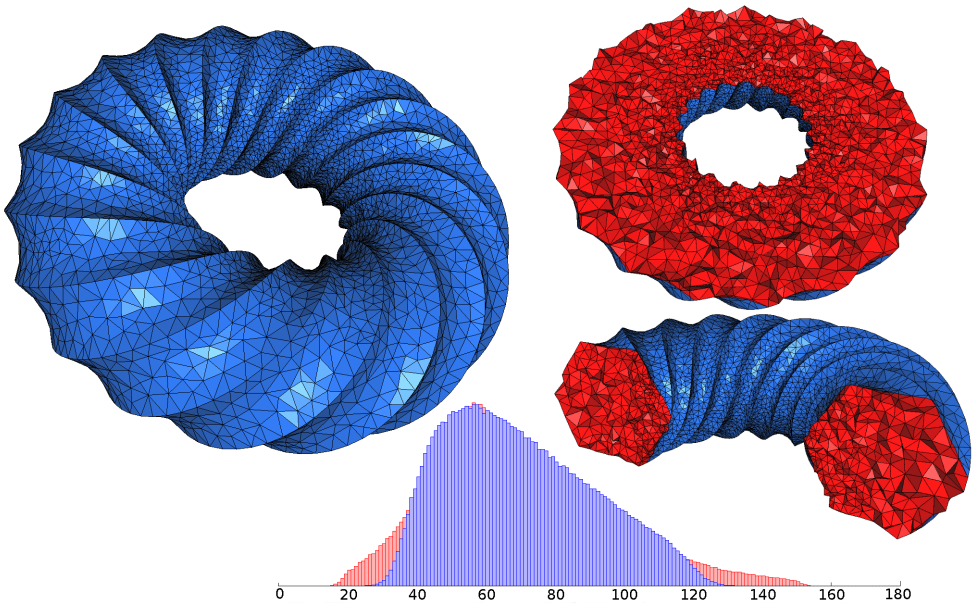


Figure 4.10: Views of the ridged torus model after applying the mesh improvement algorithm with the objective of maximizing the volume-length measure. Dihedral angles distributions before (red) and after (blue) post-processing are also presented. The refined mesh was obtained using $R = 1$ and $G = 15$. After improvements, the dihedral angles are between 22° and 143° and the minimum volume-length measure is 0.34.

generally sufficient to achieve the best possible configuration. This is followed by two passes of smoothing — most of the improvements occur during the first pass. Two objective functions to be maximized by smoothing are considered: again the minimum sine of the dihedral angles and the volume-length measure, which has the added advantage of removing spire-type tetrahedra. This entire cycle can be repeated, although the improvements observed after the first cycle are generally very small. The results from our post-processing experiments are presented in the histograms of Figures B.10 to B.19 in Appendix B. Figure 4.10 shows an example of a mesh on which the improvement algorithm has been applied.

The results unsurprisingly show that even the already good meshes we

used as a starting point benefit from the application of post-processing, some obviously more than others. More surprising is the fact that regardless of what quality measure is used during refinement, the improved meshes have similar extremal dihedral angles. In some rare cases, we even observe that meshes refined using the shortest-edge-to-circumradius ratio achieve better dihedral angles once post-processing is applied. For every tested geometry, we are able to improve meshes generated with this quality measure to levels equal or better than what is possible by refinement alone. This indicates that the vertex distribution in a Delaunay-refined mesh lends itself well to the post-processing techniques we used. This observation is in line with the conclusions of Ollivier-Gooch and Boivin [24].

From our results, it is impossible to determine which combination of measure for refinement and improvement is the best. No measure is consistently better than any other across all tested geometries. Nonetheless, by lumping together the results from all post-processed meshes, we find the best minimal dihedral to be slightly under 20°, when refinement and smoothing both aim at maximizing the minimum sine of the dihedral angles. The best maximal dihedral angle of 145°is achieved using the same refinement approach, but then by smoothing based on the volume-length measure.

Something that becomes apparent when comparing the observed improvements on meshes of the same geometry, but generated with different levels of refinement, is that the lack of boundary operations ultimately has an impact on mesh quality. For instance, take the two meshes of the joint model, one obtained using $R = G = 1$, the other with $R = 2$, $G = 15$. When comparing the improved meshes, the finer one clearly has better extremal dihedral angles. The worst tetrahedron remaining in the coarser mesh is located in the thin section above the hole with the largest diameter (see Figure 4.7). Very few interior vertices need to inserted in this region to reach the refinement threshold. Because the boundary remains untouched during post-processing, very few topological transformations and almost no vertex smoothing are possible in this neighborhood and therefore, no improvements are observed.

As expected, we are unable to reach the same quality levels obtained

Klingner and Shewchuk. However, their algorithm attains its best possible results at very high computational cost, to the point of being impractical. Comparisons with the by Alliez et al.'s algorithm are made in terms of the minimum and average inradius-to-circumradius ratio. They present results for only five different meshes, which have minimum radius ratios between 0.23 and 0.42, while the averages are between 0.86 and 0.89. The algorithm requires four minutes to produce a quality mesh of 275K tetrahedra. This result is already a few years old, so the same results would undoubtedly be obtained more quickly today. Our results show that we can beat their worst value by refinement alone (we can refine the ridged torus mesh until it contains around 330K tetrahedra in roughly 16 seconds) but we cannot compete with their averages, even when applying post-processing. It is however generally accepted that the accuracy and efficiency of a simulation is dictated by the worst element present in a mesh [31]. As such, we believe that the focus of mesh improvement should be directed at improving the extremal quality values, not the average.

4.7 Implementation Details

4.7.1 Geometry Interface

The constrained Delaunay refinement algorithm presented in Section 4.3 has been implemented in the GRUMMP meshing libraries [25]. As was mentioned earlier, we rely on the Common Geometry Module CGM for all geometry management operations [33]. We currently only use the facet-based modeler included in CGM [27]. Starting from a faceted geometry, for instance a surface triangulation in STL format, the model's topology is inferred by analyzing the dihedral angle between facets; if the dihedral angle subtended by two facets is larger than some user-specified value, then the edge shared by the facets is considered to be part of a curve. From the obtained list of curves, a boundary representation of the model is then constructed. Optionally, the facet-based modeler can also interpolate curved surfaces from a collection of facets with Bézier patches.

Unfortunately, constructing a topologically correct boundary representation from a surface triangulation is a difficult problem prone to errors. For example, as the model's complexity increases, not all features will be properly detected. An error in the model's topology almost always spells failure for our meshing algorithm. This fact is currently the main limiting factor to our implementation's range of application. An interface to OpenCASCADE has recently been developed for CGM and we plan to integrate its use into our mesh generator in the near future.

4.7.2 Vertex Insertion

The vertex insertion operations used in our refinement algorithm are all implemented on a Bowyer-Watson framework [4, 34]. Every time a new vertex v is inserted, all tetrahedra whose circumsphere contains v and who are visible from v are deleted from the mesh, thereby creating an empty hull. The faces forming the boundary of this hull are then reconnected to the new vertex, effectively retetrahedralizing the emptied cavity. For insertions in the domain's interior, there are no further actions to take. However, if v is inserted on the domain's boundary, either to split a subcurve or a subsurface, boundary edges and boundary facets will also need to be removed from the mesh.

For a subsurface split, the procedure also begins by determining all tetrahedra no longer respecting the constrained Delaunay property as a result of v's insertion. The faces of these tetrahedra form a set $\mathcal{F}$, among which the boundary facets to be deleted are found. They are identified by walking out along the surface from the boundary facet getting split. The walk is stopped either when the next contiguous boundary facet is not found in $\mathcal{F}$, or when a boundary edge is hit — the walk is never allowed to cross a boundary edge. The perimeter of the boundary facets to be removed form a polygon, the edges of which are then reconnected to v, creating the set of new boundary facets. To perform a subcurve split, we follow the exact same approach, although this time, many polygons are formed, one on each geometric surface adjoining the subcurve.

The refinement algorithm in Section 4.3 is run in various phases which might not necessarily respect the insertion priority therein presented: favoring a subcurve split over a subsurface, which is favored over a circumcenter insertion.

During the first phase of insertion, every sink tetrahedra not meeting the quality or size requirements and whose circumcenter does not encroach on the boundary is split. Since a sink is a tetrahedron containing its own circumcenter, there is no risk of inserting outside the meshed domain, even though its boundaries might still be encroached. By initiating our refinement algorithm in this manner, we avoid the creation of reflex edges that could occur following a subsurface split. As illustrated in Figure 4.11, these reflex edges are a consequence of the preferred orientation of the tetrahedra in the CDT, due to the surface curvature being high in one direction and null in an orthogonal direction. Consequently, when using Bowyer-Watson insertion to split a subsurface, the polygon defining the new boundary facets' construction will be oblong, its longer axis aligned with the tetrahedra. Some surface edges resulting from its reconnection therefore may be reflex. Inserting at sinks first completely avoids this problem in practice.

Once all sinks have been inserted, boundary encroachment can safely be fixed. Subcurves and subsurfaces are then refined to match the prescribed sizing function. Finally, in the last phase, we run the Delaunay refinement algorithm with its usual insertion priority, until all tetrahedra match the sizing function and satisfy the desired quality criterion.

To perform these insertions, orientation and insphere tests are required. We use Shewchuk's adaptive precision predicates [29]. Although these predicates are consistent and give the correct answer most of the time, they are not a silver bullet against the numerical robustness issues inherent to refining CDTs; it is not uncommon for them to return an erroneous answer. We experienced problems with the insphere predicate when constructing the hull of tetrahedra to be removed from the mesh as a result of an insertion. We observed that this situation occurs when the vertex being inserted is both very close to a tetrahedron's circumsphere and almost co-circular with the three vertices of a face of this same tetrahedron. As illustrated in Figure 4.12,

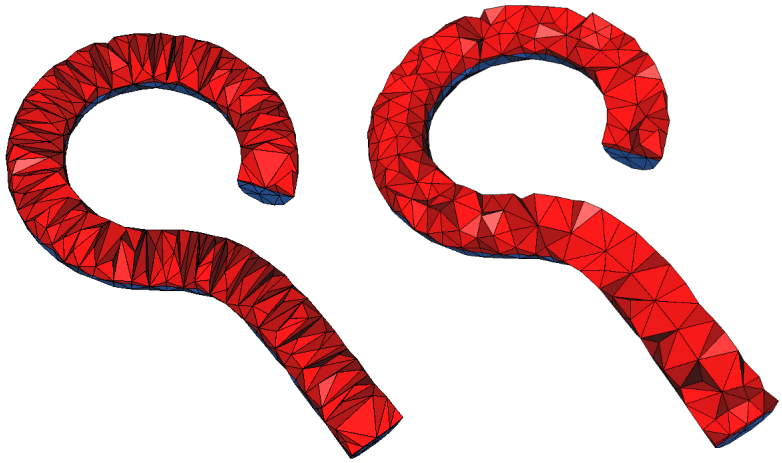


Figure 4.11: The effect of inserting at sinks are illustrated. On the left is depicted the initial CDT while the tetrahedralization refined with sinks appear on the right. It is clearly apparent that after sink insertions, tetrahedra no longer have a preferred orientation.

if the new vertex is located under the plane formed by the cocircular vertices and if the insphere predicate finds the vertex to be inside the circumsphere, then it will be impossible to reconnect the hull without creating an inverted tetrahedron.

Fortunately, the problem can easily be corrected by post-processing the hull. For every hull face, we must ensure that the new vertex is located on the same side of the face's plane as the vertex opposing the face i.e. the fourth vertex of the hull tetrahedron sharing the face. For example, in Figure 4.12, vertex v would need to be on the same side of f_1's plane as v_4, a vertex of hull tetrahedron T. When this is not the case, then the culprit tetrahedron needs to be removed from the hull, and the hull faces adjusted consequently. Proceeding this way does not affect the behavior of the Bowyer-Watson insertion scheme. In fact, Baker [2] demonstrates this approach to be correct.

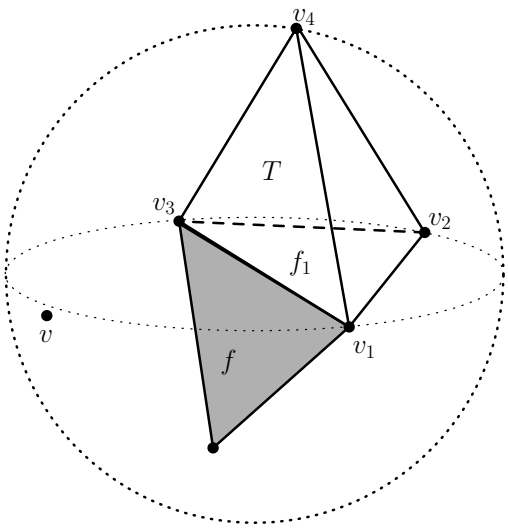


Figure 4.12: A situation where the insphere predicate might lead to the construction of a hull that cannot be correctly reconnected. Vertex v is close to T's circumsphere and is nearly cocircular with v_1, v_2 and v_3, but is slightly under the plane formed by the three vertices. If according to the insphere predicate, v is inside T's circumsphere, then T must be removed from the mesh. An edge connecting v to v_2 would go through f, a face part of the hull.

4.7.3 Timing

We present herein timing data for our Delaunay refinement implementation. The results are obtained on a single core of an Intel Q6600 processor.

Creating a tetrahedralization from a point cloud is conceivably the simplest problem our algorithm can solve. Such a test allows us to disable any boundary interaction checks, scheduling routines and queries to the geometry interface. The fastest possible insertion rates can thus be observed. making this a good test to evaluate the efficiency of our mesh data structures. Using the 35947 vertices of the Stanford bunny, a Delaunay tetrahedralization is generated in 3.6 seconds; the algorithm therefore inserts roughly 10000 vertices per second.

Correct boundary treatment adds complexity to the mesh generation

problem and as such the insertion rates are expected to drop considerably when applying Delaunay refinement on curve-bounded domains. A mesh of the ridged torus using $R = 2$ and $G = 10$ yields around 65K vertices and 330K tetrahedra. The refinement phase requires 16.3 seconds, therefore inserting slightly less than 4000 vertices every second. The same insertion rates are observed for the meshes of Figures 4.7, 4.8 and 4.10. These timing indicate that we can generate a mesh containing about one million tetrahedra in under one minute.

Mesh improvement operations are more computationally expensive. Every cycle, which includes 2 passes of topological transformations and 2 passes of smoothing, requires 40 seconds on the ridged torus model described above, 85% being spent on smoothing alone. These numbers are still under those published by Alliez et al. [1] and Klingner and Shewchuk [15] for similarly-sized meshes.

4.8 Conclusion

A constrained Delaunay refinement algorithm capable of meshing volumes bounded by curved surfaces was presented. It is an extension of well-known work by Shewchuk [29]. As opposed to Shewchuk's, our algorithm does not need to maintain a separate surface triangulation during refinement. In this regard, it is more similar to the work of Ollivier-Gooch and Boivin [24] or to TetGen's implementation [32]. To our knowledge, it is the first time an algorithm capable of refining constrained Delaunay tetrahedralization of domains bounded by curves is published.

We demonstrated that our algorithm is able to generate meshes devoid of slivers. Our experiments indicate that by choosing a proper quality measure, a high quality mesh can be generated by Delaunay refinement alone. The results presented in Appendix B show that we can produce tetrahedra with dihedral angles between 18° and 154°, which is better than any previous methods offering a provable bound on this measure [6, 9, 18, 19]. Note that the practical performances of these methods can be better than their theoretical guarantees. The meshes can be further improved, at a cost, by applying

post-processing. Only expensive mesh improvement techniques [1, 15] or mesh generation algorithms not exactly respecting the input's boundary [17] achieve better results. Although we do not provide any theoretical guarantees, our results are consistent enough across the test geometries for us to be confident that they can be repeated on a wider range of models. The fact that our implementation does not come with any provable bounds might nevertheless be seen as a limitation by some.

Currently, our algorithm's range of application is restricted by an underdeveloped access to geometry. As such, the most pressing work facing us is to integrate the interface to OpenCASCADE offered by CGM into our mesher.

Acknowledgements

The twist, rings and ridged torus models were obtained from Herbert Edelsbrunner's 180 wrapped tubes website (http://www.cs.duke.edu/~edels/Tubes/). The joint model is from the Aim@Shape repository (http://shapes.aim-at-shape.net/). All the other models were either created by us or are of unknown origin.

4.9 Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617 – 625, 2005.

[2] Timothy J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, 5(3-4):161–175, 1989.

[3] Marshall Bern, L. Paul Chew, David Eppstein, and Jim Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 189–196, 1995.

[4] Adrian Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[5] Siu-Wing Cheng and Tamal K. Dey. Quality meshing with weighted Delaunay refinement. In *Proceedings of the Thirteenth ACM-SIAM Symposium on Discrete Algorithms*, pages 137–146, 2002.

[6] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver exudation. *Journal of the ACM*, 47:883–904, 2000.

[7] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 477–494, 2007.

[8] Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, and Tathagata Ray. Quality meshing for polyhedra with small angles. *International Journal on Computational Geometry and Applications*, 15:421–461, 2005.

[9] L. Paul Chew. Guaranteed-quality Delaunay meshing in 3D (short version). In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393. ACM, June 1997.

[10] Herbert Edelsbrunner and Damrong Guoy. An experimental study of sliver exudation. *Engineering with Computers*, 18:229–204, 2002.

[11] David A. Field. Qualitative measures for initial meshes. *International Journal for Numerical Methods in Engineering*, 47:887–906, 2000.

[12] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[13] Serge Gosselin and Carl Ollivier-Gooch. Constructing constrained Delaunay tetrahedralizations of volumes bounded by piecewise smooth surfaces. Manuscript, 2009.

[14] Barry Joe. On the shape of tetrahedra from bisection. *Mathematics of Computation*, 63:141–154, 1994.

[15] Bryan M. Klingner and Jonathan R. Shewchuk. Agressive tetrahedral mesh improvement. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 3–23, 2007.

[16] Michal Křížek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.

[17] François Labelle. Sliver removal by lattice refinement. In *Proceedings of the Twenty-Second Annual Symposium on Discrete Algorithms*, pages 347–356, 2006.

[18] François Labelle and Jonathan R. Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transations on Graphics*, 26(3):57.1–57.10, July 2007.

[19] Xiang-Yang Li and Shang-Hua Teng. Generating well-shaped Delaunay meshes in 3D. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–37, 2001.

[20] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. Control volume meshes using sphere packing: Generation, refinement

and coarsening. In *Proceedings of the Fifth International Meshing Roundtable*, pages 47–61, 1996.

[21] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in three dimensions. In *Proceedings of the Eight Annual Symposium on Computational Geometry*, pages 212–221. ACM, 1992.

[22] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM Journal on Computing*, 29(4):1334–1370, 2000.

[23] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. A crystalline red green strategy for meshing highly deformable objects with tetrahedra. In *Proceedings of the Twelfth International Meshing Roundtable*, pages 103–114, 2003.

[24] Carl Olliver-Gooch and Charles Boivin. Guaranteed-quality simplicial mesh generation with cell size and grading control. *Engineering with Computers*, 17(3):269–286, 2001.

[25] Carl Ollivier-Gooch. *GRUMMP Version 0.3.0 User's Guide*. Department of Mechanical Engineering, The University of British Columbia, 2005. Available from http://tetra.mech.ubc.ca/GRUMMP/.

[26] Steve Oudot, Laurent Rineau, and Mariette Yvinec. Meshing volumes bounded by smooth surfaces. Technical Report 5626, INRIA, July 2005.

[27] Steven J. Owen, David R. White, and Timothy J. Tautges. Facet-based surfaces for 3D mesh generation. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 297–311, 2002.

[28] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[29] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.

[30] Jonathan R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the Eleventh Meshing Roundtable*, pages 193–204, 2002.

[31] Jonathan R. Shewchuk. What is a good linear element? Interpolation, conditioning, anisotropy, and quality measures. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 115–126, September 2002.

[32] Hang Si. On refinement of constrained Delaunay tetrahedralizations. In *Proceedings of the Fifteenth Internation Meshing Roundtable*, pages 509–528, 2006.

[33] Timothy J. Tautges. The common geometry module (CGM). Technical Report SAND2004-6252, Sandia National Laboratories, December 2004.

[34] David F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, 24(2):167–172, 1981.

Chapter 5

Conclusion

In the previous chapters, a study on the application of Delaunay refinement mesh generation algorithms to geometric domains bounded by piecewise smooth curves and surfaces was presented. This work is part of the development of a collection of research tools to study the benefits of high-order accurate finite-volume simulations in terms of solution accuracy and efficiency.

In Chapter 2, a number of specific improvements to an existing algorithm [4] were introduced, notably regarding the pre-discretization of curves and the treatment of small input angles. These proposed changes lead to the implementation of a provably-good triangular mesh generator which has been shown to construct smaller-sized meshes when compared to previous approaches. The same quality bounds offered by the original algorithm for piecewise linear domains [20] are also demonstrated.

Chapters 3 and 4 are concerned with the more complex three-dimensional problem. In particular, Chapter 3 presents an algorithm to construct tetrahedralizations conforming to domains bounded by piecewise smooth surfaces. Previous research solved this problem using conforming Delaunay tetrahedralizations [3, 5, 19]. The implementation presented in this manuscript is based on the more involved constrained Delaunay tetrahedralizations (CDTs). The difficulty of this approach resides in the fact that not every set of constraining edges and faces admits a CDT. This complicates the ensuing boundary recovery algorithm. The extra effort is however rewarded with a mesh devoid of unnecessarily small features providing an ideal starting point for the study presented in Chapter 4. As was mentioned in the thesis, Delaunay-refined tetrahedralizations are plagued with the presence of slivers. These cannot be provably eliminated by Delaunay refinement alone. Nevertheless, the results in the manuscript show that if non-standard quality measures are

used to drive the refinement procedure, the slivers can be easily be removed in practice. In terms of mesh quality, the mesh generator implemented for the study outperforms other methods for sliver removal [8, 21]. To the author's knowledge, this is the first CDT-based Delaunay refinement algorithm to generate quality meshes of domains bounded by curved surfaces.

In this chapter, the significance of the three manuscripts is discussed. The strengths and weaknesses of the method are also covered. This is followed by an outlook on future research.

5.1 Significance of the Results

The central objective of the three manuscript chapters at the core of this thesis was to study and implement practically relevant Delaunay refinement algorithms. The main focus was on the extension of this mesh generation paradigm to domains whose boundaries are curved. The method was originally developed under the idealized assumption that all input geometries are piecewise linear or piecewise planar. Although this simplification helped to establish the theoretical guarantees the method is now noted for, it also makes it less appealing for use in a design loop involving numerical simulations. Today, CAD modelers are an integral part of the engineering design process and the inherent incapacity of Delaunay refinement to create meshes directly from geometries produced by these modelers is an important practical limitation. While the results presented in the manuscripts only include meshes of mock-up CAD models, the algorithms developed for this thesis are an important stepping stone towards the coupling of Delaunay refinement mesh generators to CAD applications.

5.1.1 Triangular Mesh Generation

The manuscript presented in Chapter 2 addresses unresolved issues from a previously published Delaunay refinement algorithm accepting domains bounded by curves [4]. Two new theoretically relevant results have been established. The first one concerns the sampling criterion used during the

initial sampling of curves. It was shown that much coarser discretizations are possible while still guaranteeing that no invalid insertions will be performed near the domain's boundary. As a result, triangulations of smaller size can be generated, an important considerations for practitioners using solution adaptive meshes. The second result is a consequence of a pre-refinement algorithm forcing every boundary edge to respect the constrained Delaunay property prior to building the triangulation. Applying the algorithm allows the domain's boundary to be recovered using flips only while not creating unnecessarily small features. Furthermore, upon recovery, all boundary edges are unencroached. Interior refinement can therefore be launched immediately, without performing additional verifications. The new approach replaces a more complicated boundary recovery algorithm based on heuristics [4]. With access to proper data structures, the presented method is arguably simpler to implement.

Another important contribution of the manuscript concerns the treatment of small input angles. A simple extension of a provably-good protection scheme for piecewise linear inputs [14] is used to prevent infinitely recursive refinement. Interestingly, our solution to the problem posed by small angles illustrates the different points of view applied mathematicians and engineers have regarding mesh generation. The original scheme's termination proof is based on an assumption that does not extend to curved inputs. Nonetheless, if recursive refinement was to happen, something we failed to observe in practice, the curve segments subtending a small angle would eventually become short enough to be considered straight. Consequently, the assumption needed to prove termination is bound to be valid at some point. For an engineer, this argument is sufficient to warrant an implementation. Despite the lack of a termination proof, the resulting algorithm exhibits very good practical performance. It was tested with a large number of pathological cases that were specifically designed to cause a failure, but no failures occurred. At the other end of the spectrum, Pav and Walkington's answer was to design a new algorithm also dealing with small angles between input curves and admitting a termination proof [17]. This is a very important achievement in itself. However, from a practical standpoint, their implementation has

a tendency to create triangles with angles approaching 180° opposing the small input angles, something that should be avoided in meshes for numerical solutions of partial differential equations. Different fields of research do have different objectives.

The triangular mesh generation algorithm presented in the manuscript is shown to achieve the same quality bounds as Shewchuk's Ruppert-Chew hybrid [20], upon which it is based. In practice, it creates meshes where all angles are between 30° and 120° and whose triangle size can be controlled, either automatically or by a user-defined size distribution function. The implementation is mature and robust. Very few improvements to the method can be foreseen, although extensions allowing parallel or anisotropic mesh refinement would be worthwhile problems to explore. The mesh generator which produced the results in the manuscript has already been used successfully to make significant contributions to the development of efficient high-order accurate finite-volume simulations [13].

5.1.2 CDT Construction

The manuscript in Chapter 3 presents an algorithm to generate CDTs out of geometries bounded by piecewise smooth surfaces. This is the first step towards constructing quality meshes in three dimensions, which is a significant contribution to the future integration of Delaunay refinement to CAD-based design. The generation of an initial boundary conforming mesh is perhaps the most crucial and difficult part of any Delaunay-based mesh generation algorithm. The whole process can be subdivided into two major phases: boundary discretization and boundary recovery.

The boundary discretization phase essentially consists of creating a surface triangulation to which the tetrahedralization will be constrained. This triangulation must form a good approximation of the surface, both topologically and geometrically. Some authors have already shown that by starting from an initially coarse sample and by combining Chew's furthest point strategy [7] with results from sampling theory [1], Delaunay refinement can be used to create a topologically correct surface triangulation of the model in

direct space [3, 5]. The approach proposed in the manuscript is similar, with the exception that the model's surfaces are sampled individually. Although simple, this idea can make a significant difference regarding the density of sample needed to correctly recover the topology. This is especially true in situations where two bounding surfaces are close together, but separated by an unmeshed region. A coarser sample will eventually lead to a coarser tetrahedralization, thus satisfying one of the objectives stated in Chapter 1. Furthermore, individual sampling obviates the need for small angle protection, an operation which typically require brute-force evaluation of the local feature size along curves. As a result, the proposed implementation is about ten times faster than the timings reported by Cheng et al. [5].

Sampling in direct space offers some advantages over the creation of a surface mesh in parametric space. Better control over the triangle's size and quality is possible. The creation of inverted elements resulting from the projection of the parametric triangulation onto the surface is also avoided. Versatility is another benefit: the proposed method can accept various types of surface representations: parametric, implicit or faceted surfaces for instance. On the flipside, meshing in parametric space is simpler and faster because it relies on two-dimensional mesh generation algorithms.

Once the surface triangulation is obtained, a boundary recovery phase is necessary to construct the CDT. The proposed approach is based on heuristics and combines topological transformations with vertex insertions. Some might argue that the lack of a termination guarantee is a weakness of the method. However, when these transformations fail, vertices are then inserted on the surface. After a certain number of insertions, a boundary conforming Delaunay tetrahedralization must eventually be obtained. In this worst case scenario, the tetrahedralization will have a cardinality comparable to meshes constructed by competing methods [5, 19]. In practice however, very few additional vertices are generally needed. Consequently, coarser boundary conforming tetrahedralizations are possible, a result which is quite significant in regards to the generation of coarse quality meshes in three dimensions.

5.1.3 Tetrahedral Mesh Refinement

The manuscript of Chapter 4 presents an experimental study on sliver removal by Delaunay refinement. The intuition behind this experiment is based on two observations. Firstly, a canonical sliver is formed by four nearly coplanar vertices equally distributed around the equator of its circumsphere. The four vertices are close to being cocircular and therefore in a degenerate configuration. Inserting a vertex at the center of this circle removes this degeneracy and with it, the sliver. However, every insertion can potentially create new slivers, which is the reason why Delaunay refinement does not come with a guarantee on dihedral angles. The algorithm would fail to terminate if new slivers keep being created every time a removal insertion is attempted. In practice, slivers are generally sparsely distributed throughout a Delaunay-refined tetrahedralization. Based on this observation, the failure mechanism just described is unlikely, as confirmed by the experimental results presented in the manuscript.

The most significant result in the manuscript is that Delaunay refinement can be driven with non-standard quality measures and produce meshes devoid of slivers. In our experiments, tetrahedralizations with dihedral angles between 18° and 154° are consistently obtained using a refinement strategy based on the minimum sine of a tetrahedron's dihedral angles. A fairer measure detecting every possible type of poorly-shaped tetrahedra yields dihedral angles between 14° and 154°. To the author's knowledge, no other tetrahedral mesh generation algorithm strictly enforcing boundary conformity has been shown to produce meshes with such good dihedral angles, at least without any post-processing being applied. The quality surface triangulations produced by the algorithms in Chapter 3 contribute to these good results.

The main weakness of the approach is evidently its lack of theoretical guarantee. A contrived vertex configuration leading to a failure of the algorithm might be conceivable. Such a situation is yet to be observed in practice. The number of test cases studied and their complexity is however limited, which is a consequence of insufficient access to geometry directly

from the mesh generator. The most pressing future work is to improve the geometry interface to demonstrate the performance of the mesh generator with a larger test suite.

The manuscript also explores the application of mesh improvement algorithms to Delaunay-refined tetrahedralizations. The objective here was to establish whether a mesh refined with a given quality measure would lend itself better to improvements. The improvement schedules were fairly simple and relatively inexpensive in terms of computational cost. Unfortunately, the results obtained are inconclusive. Post-processing does generally improve mesh quality (typically, the minimum dihedral angle can be pushed over 20°, sometimes higher), but no correlation could be established between the quality measure used during refinement and the mesh quality after improvements.

5.2 Outlook

Delaunay refinement is a mesh generation technique that is gaining in maturity. As demonstrated in this thesis, its practical capabilities often exceed its theoretical guarantees. Based on its ability to produce quality simplicial meshes both in two and three dimensions, it is bound to emerge out of the realms of academic research software and make its appearance in commercial applications in a near future.

In this section are listed a few desirable improvements from which the current implementation would benefit. As was mentioned in the manuscripts, the most pressing issue is to improve access to geometry in three dimensions. The ability to create anisotropic meshes for high-order finite-volume simulations and to generate such meshes in parallel would also be interesting features.

5.2.1 Geometry Interface

One of the major limitations of the implemented tetrahedral mesh generator is its insufficient access to geometry. The CGM library currently offers

geometry support [23]. It offers a facet-based modeler and an interface to the ACIS modeler, not available to us despite our request. Support for OpenCASCADE has recently been added. It is not exploited yet.

At the moment, meshes can only be generated with the facet-based modeler. Unfortunately its capabilities are limited. It is an excellent basis upon which ideas can be tested and developed. However, when models become more complex, the facet-based modeler often fails to provide a correct boundary representations. This inevitably spells failure for the mesh generation algorithm.

One of the main implementation problems resides in the fact that the mesh generator is not de-coupled from geometry queries. Calls to CGM are embedded deeply in the mesh generation routines. Even though GRUMMP offers a simple native geometry support, building it now requires linkage to CGM: a problematic situation from a code distribution perspective. A better alternative would be to construct a native geometry interface using polymorphic calls to access geometric information. The mesh generator could therefore be built with options to link to CGM, OpenCASCADE or other modelers. Geometric routines specific to particular models could then be considered in each derived class. For instance, the intersection between line segments and surfaces required by the surface sampling algorithm of Chapter 3 could be implemented differently for a faceted or for a parametric surface, exploiting the characteristics of each to produce more efficient algorithms.

When this native geometry interface is completed, the range of application of our tetrahedral mesh generator will be broadened. Testing on more complex geometries will help validate some of the findings presented in this thesis' manuscripts.

5.2.2 Anisotropy

The algorithms in this thesis generate meshes which are good for isotropic problems. However, many physical phenomena relevant to engineers have anisotropic solutions. For instance, the boundary layer, encountered in viscous fluid flows, is a region located near wall boundaries where strong

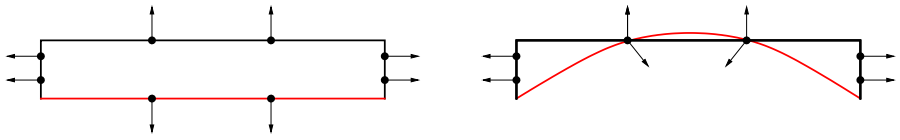
velocity gradients develop in a direction perpendicular to the flow. To correctly resolve it in a computational aerodynamics simulation, one could use a large number of very small isotropic triangles clustered at the boundary, as is done in the mesh depicted in Figure 2.9. Such an approach will correctly resolve the strong gradients in the perpendicular direction, but will also over-resolve the flow in the tangential direction. As a result, the simulation will be inefficient. A better approach is to use anisotropic entities aligned with the flow.

Delaunay-based methods have already been used to generate anisotropic triangulations [11, 16]. For numerical methods, a better approach would perhaps be to use mixed meshes relying on stretched quadrilaterals in anisotropic regions and Delaunay-refined triangles for the rest of the domain. A few options could be explored to generate such meshes: constructing a structured mesh by stacking quadrilaterals at wall boundaries and completing the mesh by Delaunay refinement and "quadifying" an anisotropic triangulation are two of the possible avenues. More investigation on the subject will be necessary. Surely, three-dimensional anisotropic mesh generation will be even more challenging.

In the context of high-order simulations, a subtle but nonetheless important problem might arise with anisotropic control volumes and curved boundaries. The situation is illustrated in Figure 5.1. To obtain high-order accuracy, it is essential to account for the curved boundary when locating flux integration points. A highly anisotropic mesh could lead to ill-defined control volumes where boundary faces extends past their opposing faces. To avoid this problem, the interior faces near the boundary also need to be curved. Solutions to this problem have been proposed by Luo [12] and Persson and Preraire [18].

5.2.3 Parallelization

The Delaunay refinement algorithms implemented for this thesis are all sequential. They are sufficiently fast for adaptively refining meshes of relatively small problems, say for solutions requiring on the order of a few million



(a) (b)

Figure 5.1: The effects of curved boundaries on anisotropic quadrilaterals in a high-order finite-volume simulations. The Gauss points and normals are depicted. (a) Control volume neighboring a straight boundary. (b) Control volume neighboring a curved boundary. The Gauss points on the curve coincide with those on its opposing face.

elements. Generating and updating solution-adapted meshes for larger scale problems however calls for the meshing process to be parallelized. Delaunay refinement is conceptually difficult to run in parallel because it is sequential in nature. The decision to insert a new vertex is based on the current state of the mesh. Furthermore, every time a vertex is inserted, the mesh is modified locally, making it hard to maintain entity-based partitions.

Parallel Delaunay refinement is a subject of active research. Analytical work [10, 22] as well as a few implementations have already been proposed for two-dimensional domains [6, 9], three-dimensional points clouds [2] and simple polyhedral domains [15]. These algorithms have demonstrated an increase in performance when compared with their sequential counterparts. Hudson et al. [10] however complain that these all lack strong theoretical guarantees.

A parallel Delaunay refinement algorithm accepting curved inputs would be faced with the same challenges as other parallel implementations, but would also require a geometry interface able to run in parallel. As was previously mentioned, the current objective is to improve the range of application of our implemented interface. Designing a parallel interface is not planned in the foreseeable future, but is nonetheless an interesting consideration for the long term.

5.3 Bibliography

[1] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22(4):481–504, 1999.

[2] Daniel K. Blandford, Guy E. Blelloch, and Clemens Kadow. Engineering a compact parallel Delaunay algorithm in 3D. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, pages 292–300, 2006.

[3] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.

[4] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1203, 2002.

[5] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 477–494, 2007.

[6] Andrey N. Chernikov and Nikos P. Chrisochoides. Generalized Delaunay mesh refinement: From scalar to parallel. In *Proceedings of the Fifteenth International Meshing Roundtable*, pages 563–580, 2006.

[7] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280, 1993.

[8] Herbert Edelsbrunner and Damrong Guoy. An experimental study of sliver exudation. *Engineering with Computers*, 18:229–204, 2002.

[9] Sariel Har-Peled and Alper Ün̈gör. A time-optimal delaunay refinement algorithm in two dimensions. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 228–236, 2005.

[10] Benoît Hudson, Gary L. Miller, and Todd Phillips. Sparse parallel Delaunay mesh refinement. In *Proceedings of the Nineteenth ACM*

Symposium on Parallelism in Algorithms and Architectures, pages 339–347, 2007.

[11] François Labelle and Jonathan R. Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 191–200, 2003.

[12] Xiaojuan Luo. *An Automatic Adaptive Directional Variable p-version Method in 3D Curved Domains.* PhD thesis, Rensselaer Polytechnic Institute, 2005.

[13] Christopher Michalak. *Effcient High-Order Accurate Unstructured Finite-Volume Algorithms for Viscous and Inviscid Compressible Flows.* PhD thesis, Department of Mechanical Engineering, The University of British Columbia, 2009.

[14] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the Twelfth International Meshing Roundtable*, pages 91–102, 2003.

[15] Démian Nave, Nikos Chrisochoides, and L. Paul Chew. Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28(2-3):191–215, 2004.

[16] Doug Pagnutti and Carl Ollivier-Gooch. Delaunay-based anisotropic mesh adaptation. In *Proceedings of the Seventeenth International Meshing Roundtable*, pages 141–157, 2008.

[17] Steven E. Pav and Noel J. Walkington. Delaunay refinement by corner lopping. In *Proceedings of the Fourteenth International Meshing Roundtable*, pages 165–181, 2005.

[18] Per-Olof Persson and Jaime Preraire. *Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics.* 47th AIAA Aerospace

Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, paper AIAA 2009-949, 2009.

[19] Laurent Rineau and Mariette Yvinec. Meshing 3D domains bounded by piecewise smooth surfaces. In *Proceedings of the Sixteenth International Meshing Roundtable*, pages 443–460, 2007.

[20] Jonathan R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):27–74, 2002.

[21] Hang Si. On refinement of constrained Delaunay tetrahedralizations. In *Proceedings of the Fifteenth Internation Meshing Roundtable*, pages 509–528, 2006.

[22] Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings of the Eleventh Internation Meshing Roundtable*, pages 205–218, 2002.

[23] Timothy J. Tautges. The common geometry module (CGM). Technical Report SAND2004-6252, Sandia National Laboratories, December 2004.

Appendix A

Triangular Mesh Generation Analysis

A.1 Mesh Quality

The proof presented in this appendix closely follows the method used by Ruppert [3]. We assume that diametral lenses protect boundary edges and that input curves are separated by angles of at least 60°. The same angle bound as the one obtained by Shewchuk [4] for diametral lens encroachment will be showed. The constants in the proof are tighter for a grading factor $G > 1$, which involves slower change in triangle size over some distance.

Lemma A.1. For fixed constants C_0, C_1, C_2 and C_3 specified below, the following statements hold:

1. During boundary enrichment, for each vertex v inserted to split a subcurve, the distance to its nearest visible neighbor vertex is at least $\frac{\text{lfs}(v)}{C_0}$.

2. At initialization of the Delaunay refinement algorithm, for each input vertex v, the distance to its nearest visible neighbor vertex is at least $R \cdot LS(v)$.

3. When a vertex v is chosen as the circumcenter of a large triangle, the distance to the nearest visible vertex is at least $\frac{LS(v)}{C_1}$. v can be added to the triangulation or may be rejected because of encroachment rules.

4. When a vertex v is chosen as the circumcenter of a poor quality triangle, the distance to the nearest visible vertex is at least $\frac{LS(v)}{C_2}$. v can be

added to the triangulation or may be rejected because of encroachment rules.

5. When a vertex v is inserted to split a subcurve, the distance to its nearest visible neighbor vertex is at least $\frac{LS(v)}{C_3}$.

Proof. Every possible insertion case is treated separately:

Case 1. Vertex v is inserted to split a subcurve s bounded by vertices v_1 and v_2 during boundary enrichment. This split is performed because vertex v_3 is inside the diametral circle of boundary edge $e = \overline{v_1 v_2}$ and is visible from e. Assume $\text{lfs}_{\min}$ is the smallest local feature size over the entire domain.

Subcase 1a. v_3 is the endpoint of a boundary edge e' (with associated subcurve s') non-disjoint from e. Since we have assumed that all curves are separated by angles of at least 60°, this situation can only happen if subcurve s' turns on itself such that v_3 is located in e's diametral circle (this is possible since TV can be as high as $\frac{\pi}{2}$ over a subcurve). Subcurves s and s' will be recursively split until all their boundary edges become unencroached. *Subcases* 1b or 1c then apply.

Subcase 1b. v_3 is the endpoint of a boundary edge e' disjoint from e. By definition, the local feature size is the smallest distance between two disjoint boundary entities, therefore $\text{lfs}(v)$ cannot be greater than $|vv_3|$. Lemma A.1 holds for any bounded value of $C_0 \geq 1$.

Subcase 1c. v_3 is the endpoint of a boundary edge e' intersecting e. e and e' must be disjoint for them to intersect, so any bounded value of $C_0 \geq 1$ suffices.

This shows that the boundary enrichment algorithm does not create boundary edges shorter than $\frac{\text{lfs}_{\min}}{C_0}$. Therefore, each boundary edge has finite size and only a finite number of them will be required. The boundary enrichment algorithm must terminate.

Case 2. v is an input vertex: a boundary vertex which is either the endpoint of a curve, or was inserted to discretize a curve. By definition of the length scale $LS(v)$, (Equation 2.2) if $\widetilde{LS} < \frac{\text{lfs}(v)}{R}$, then $\text{lfs}(v) < R \cdot LS(v)$,

otherwise $\mathrm{lfs}(v) = R \cdot LS(v)$. Therefore, Statement 2 of this lemma holds provided $R \geq 1$.

Having established the lemma's truth for the initial mesh, we now analyze every insertion possibility to prove that it must be true for all meshes generated by the algorithm. As such, we assume that the lemma holds for every vertex previously inserted and determines bounds on C_1, C_2, C_3 and G that are required for it to hold for newly inserted vertices.

Case 3. v is considered for insertion at the circumcenter of a large triangle t. The insertion was scheduled because

$$r \geq \frac{\sqrt{2}}{2}\overline{LS}_t, \tag{A.1}$$

where r is the circumradius of t and $\overline{LS}_t$ is the average length scale at the vertices of t. By definition of the length scale:

$$LS(v) \leq \widetilde{LS}(v) \leq LS(v_i) + \frac{|vv_i|}{G} = LS(v_i) + \frac{r}{G},$$

where v_i are the vertices of t. Summing both sides of the equation for each vertex of t:

$$\widetilde{LS}(v) \leq \frac{\sum_{i=1}^{3} LS(v_i)}{3} + \frac{r}{G} = \overline{LS}_t + \frac{r}{G}. \tag{A.2}$$

Combining Equations A.1 and A.2:

$$r \geq \frac{\sqrt{2}}{2}\left(\widetilde{LS}(v) - \frac{r}{G}\right) = \frac{\widetilde{LS}(v)}{\sqrt{2} + \frac{1}{G}} \geq \frac{LS(v)}{\sqrt{2} + \frac{1}{G}}.$$

This inequality places a lower bound on vertex spacing for the insertion at the circumcenter of a large triangle. Statement 3 of Lemma A.1 holds for

$$C_1 \geq \sqrt{2} + \frac{1}{G}. \tag{A.3}$$

The lower bound on C_1 decreases as G increases, which confirms the expected slower change in size between neighboring triangles.

Case 4. v is considered for insertion at the circumcenter of a triangle t

(with vertices v_1, v_2 and v_3) not meeting the required angle bound. Without loss of generality, we assume vertices v_1 and v_2 to be connected by the shortest edge of t, having length l. We also assume that v_1 was inserted after v_2 (or that both were input vertices). The radius of the circle free of visible vertices around v_1 is ρ. We are first seeking a lower bound on l.

Subcase 4a. v_1 is an input vertex, then so is v_2. The result obtained from *Case* 2 suffices: $l \geq R \cdot LS(v_1)$.

Subcase 4b. v_1 was inserted at the circumcenter of a large triangle t of circumradius ρ. ρ cannot be larger than l since v_2 cannot be inside the circumcircle of t. Then, the result obtained from *Case* 3 suffices: $l \geq \rho \geq \frac{LS(v_1)}{C_1}$.

Subcase 4c. v_1 was inserted at the circumcenter of a triangle t not meeting the quality target. The argument of *Subcase* 4b holds, so applying Statement 4 of this lemma to v_1, we obtain $l \geq \rho \geq \frac{LS(v_1)}{C_2}$.

Subcase 4d. v_1 was inserted to split a subcurve s. We know that v_2 is not inside the diametral circle of the boundary edge of s, otherwise v_2 would have been deleted prior to the subcurve split. Applying Statement 5 of this lemma to v_1, we obtain $l \geq \rho \geq \frac{LS(v_1)}{C_3}$.

Assuming $C_3 \geq C_2 = C_1 \geq 1$ (which we will show is possible), then Statement 4 of this lemma will be satisfied if $LS(v_1) \leq C_3 \cdot l$. Knowing that the circumradius-to-shortest-edge ratio of t relates l to t's smallest angle θ by the formula $\frac{r}{l} = \frac{1}{2\sin\theta}$ and using the definition of the length scale, we obtain the following inequality:

$$LS(v) \leq \widetilde{LS}(v) \leq LS(v_1) + \frac{|vv_1|}{G} = LS(v_1) + \frac{r}{G} \leq C_3 \cdot l + \frac{r}{G} = 2C_3 r \sin\theta + \frac{r}{G},$$

The attempt to insert a new vertex at the circumcenter of t is made because θ is smaller than the required angle bound α. As $\sin\alpha > \sin\theta$, the following bound is obtained:

$$r \geq \frac{LS(v)}{\frac{1}{G} + 2C_3 \sin\alpha}.$$

Statement 4 of this lemma requires that $r \geq \frac{LS(v)}{C_2}$, which is satisfied when

$$C_2 \geq \frac{1}{G} + 2C_3 \sin\alpha. \tag{A.4}$$

Case 5. v is inserted to split subcurve s associated with boundary edge e and bounded by vertices v_1 and v_2.

Subcase 5a. v_3, an endpoint of boundary edge e', is located inside the diametral lens of e. Because we assume that input curves meet at an angle of at least 60° and since any initial encroachment was fixed during the boundary enrichment phase, then e and e' must be disjoint. $LS(v) \leq \frac{\text{lfs}(v)}{R} \leq \frac{|vv_3|}{R}$. *Case* 2 requires $R \geq 1$, therefore this lemma is satisfied for bounded values of $C_3 \geq 1$.

Subcase 5b. v_3 is the rejected circumcenter of a large triangle t, located inside e's diametral lens. Let r be the radius of e's diametral circle, $r = \frac{|v_1v_2|}{2}$. The circumcircle of t must be vertex-free, therefore its radius r' must not be larger than the shorter of $|v_1v_3|$ and $|v_2v_3|$; r' is maximal when v_3 is at the apex of the diametral lens, so $r' \leq \frac{2\sqrt{3}}{3}\frac{|v_1v_2|}{2} = \frac{2\sqrt{3}}{3}r$. We also know from this lemma that $\frac{LS(v_3)}{C_1} \leq r$. Once again, applying the definition of the length scale:

$$LS(v) \leq \widetilde{LS}(v) \leq LS(v_3) + \frac{|vv_3|}{G}.$$

Since v and v_3 are both inside e's diametral circle, then $|vv_3| \leq 2r$:

$$\begin{aligned} LS(v) &\leq LS(v_3) + \tfrac{2r}{G} \\ &\leq r'C_1 + \tfrac{2r}{G} \\ &\leq \tfrac{2\sqrt{3}}{3}rC_1 + \tfrac{2r}{G} \end{aligned}$$

All vertices inside e's diametral circle have been removed before inserting v; v has no vertex remaining in the triangulation closer than some distance d. d is minimal when v is as close as possible to the diametral circle of e. This occurs when the change in tangent over s, TV_s, is lumped at a single point. In these conditions, v is on a circular arc having its center along the bisector

of e and passing through v_1 and v_2. The angle formed by connecting v to the endpoints of e is a function of TV_s: $\angle v_1 v v_2 \geq \pi - TV_s$, with $TV_s < \frac{\pi}{2}$. If v is along the bisector of e, then $d \geq r\left(1 - \tan\left(\frac{TV_s}{2}\right)\right)$. If v is anywhere else on the arc, this bound is reduced by a factor λ, where $0 < \lambda \leq 1$. Therefore, the length scale inequality can be rewritten as:

$$LS\left(v\right) \leq \frac{2d\left(\frac{\sqrt{3}}{3}C_1 + \frac{1}{G}\right)}{\lambda\left(1 - \tan\left(\frac{TV_s}{2}\right)\right)}.$$

This inequality satisfies Lemma A.1 provided that

$$C_3 \geq \frac{2\left(\frac{\sqrt{3}}{3}C_1 + \frac{1}{G}\right)}{\lambda\left(1 - \tan\left(\frac{TV_s}{2}\right)\right)}. \tag{A.5}$$

Subcase 5c. v_3 is the rejected circumcenter of a poor quality triangle, located inside e's diametral lens. The same approach as in *Subcase* 5c can be applied and therefore

$$C_3 \geq \frac{2\left(\frac{\sqrt{3}}{3}C_2 + \frac{1}{G}\right)}{\lambda\left(1 - \tan\left(\frac{TV_s}{2}\right)\right)}. \tag{A.6}$$

Subcase 5d. v is inserted because e is too long with respect to the sizing field:

$$l \geq LS\left(v_1\right) + LS\left(v_2\right),$$

where l is the length of e. Since the algorithm always splits long boundary edges before inserting any other vertices in the mesh, then the vertex closest from v must either be v_1, v_2 or v_3, the endpoint of a boundary edge disjoint from e. *Subcase* 5a applies and this lemma is satisfied for bounded values of $C_3 \geq 1$.

To establish the truth of Lemma A.1, we must find values of C_1, C_2 and C_3 that simultaneously satisfy Inequalities A.3, A.4, A.5 and A.6. We will

establish tight bounds on each constant by requiring equality in each case. Using only Inequalities A.4, A.5 and A.6, we find that

$$\begin{aligned} & & C_3 &= \frac{1}{G}\frac{2\left(\sqrt{3}+1\right)}{\sqrt{3}\delta-4\sin\alpha} \\ C_1 &= C_2 &= \frac{1}{G}\frac{\sqrt{3}\delta+4\sqrt{3}\sin\alpha}{\sqrt{3}\delta-4\sin\alpha} \end{aligned}$$

where $\delta = \lambda\left(1-\tan\left(\frac{TV_s}{2}\right)\right)$. The constants are bounded for any angle $\alpha < \arcsin\left(\frac{\sqrt{3}}{4}\delta\right)$. Note that as more points are inserted to split subcurves, $TV_s \to 0$ and $\lambda \to 1$. Therefore, as we strive to achieve the angle bound, subcurves will be split such that $\delta \to 1$. As a result, the angle bound $\alpha \approx 25.7°$ proved by Shewchuk for a segment bounded domain also applies for curved inputs. In these conditions, the constants are nearly identical to Ruppert's, with the addition of a constant factor $\frac{1}{G}$.

We can treat Equation A.3 as establishing a lower bound on the grading rate $G \geq \frac{1}{C_1-\sqrt{2}}$. So long as G is finite, the mesh will be non-uniform. Relating G to the angle bound α, we find:

$$G \leq \frac{2\sqrt{2}\left(\sqrt{3}+1\right)\sin\alpha}{\sqrt{3}\delta-4\sin\alpha}.$$

The minimum theoretical grading rate G remains finite up to the previously established angle bound. If the lower bound for G is used, we have

$$\begin{aligned} & & C_3 &= \frac{\sqrt{2}}{2}\sin\alpha \\ C_1 &= C_2 &= \frac{\sqrt{6}(\delta+4\sin\alpha)}{2\left(\sqrt{3}+1\right)\sin\alpha} \end{aligned}$$

In summary, we have established that for $R \geq 1$, our algorithm generates meshes with the same angle bound as Shewchuk's algorithm for triangular mesh generation. In the process, we have placed bounds on the grading G and the length of the shortest edge in the mesh relative to the local length scale (the constants C_1, C_2 and C_3 give this information). □

A.2 Termination and Size Optimality

We use Lemma A.1 to prove the following theorem about finite mesh size and mesh size optimality.

Theorem A.2. *Let $LS_{\min}$ be the shortest length scale in the input PCC. Then, given a vertex v in the output mesh, its nearest neighbor vertex w is at a distance at least $LS(v)/(C_3 + 1/G)$. This implies mesh size optimality.*

Proof. Lemma A.1 handles the case where v is inserted after w. If w is inserted last, then we apply the lemma to w:

$$|vw| \geq \frac{LS\left(w\right)}{C_3}.$$

But $LS\left(v\right) \geq \widetilde{LS}\left(v\right) \geq LS\left(w\right) + \frac{|vw|}{G}$, so

$$|vw| \geq \frac{LS\left(v\right) - \frac{|vw|}{G}}{C_3} = \frac{LS\left(v\right)}{C_3 + \frac{1}{G}}.$$

Because the shortest edge in the mesh must be longer than $\frac{LS_{\min}}{C_3 + \frac{1}{G}}$, each triangle has finite size and only a finite number of them will be required. Therefore, the algorithm will always terminate.

We can say more than that. Because the shortest possible edge is within a constant factor of the length scale locally, the smallest possible triangle is within the square of that same factor of the size of a triangle whose edges all match the length scale. This implies that the size of the mesh must be within a constant factor of the size of the smallest possible mesh whose triangles meet the quality bound and whose edges have length within a constant factor of the length scale locally. □

Appendix B

Tetrahedral Mesh Refinement Supporting Data

B.1 Quality Measures Analysis

In this section, we analyze the quality measures we propose to use with our Delaunay refinement algorithm. The goal is to establish how well each measure performs in the presence of the different types of poorly-shaped tetrahedra. We classify bad tetrahedra using the taxonomy proposed by Cheng et al. [1], which we reproduce in Figure B.1 for completeness.

To determine how each measure behaves as a tetrahedron's quality degrades, a parametrization of every possible bad tetrahedra is utilized. The vertex coordinates are defined as a function of variable u, $0 < u \leq 1$. When $u = 1$, every parametrization yields a regular tetrahedron while a poorly-shaped tetrahedron is obtained as $u \to 0$. The approach is inspired by Liu and Joe [2] and is adapted to include all classes of tetrahedron described in Cheng et al.'s taxonomy. The vertex coordinates are defined as follows:

Spire:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (u,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}, \tfrac{\sqrt{6}}{3}\right) \\ v_3 &= \left(\tfrac{1}{2}u, \tfrac{\sqrt{3}}{2}u, 0\right) \end{aligned}$$

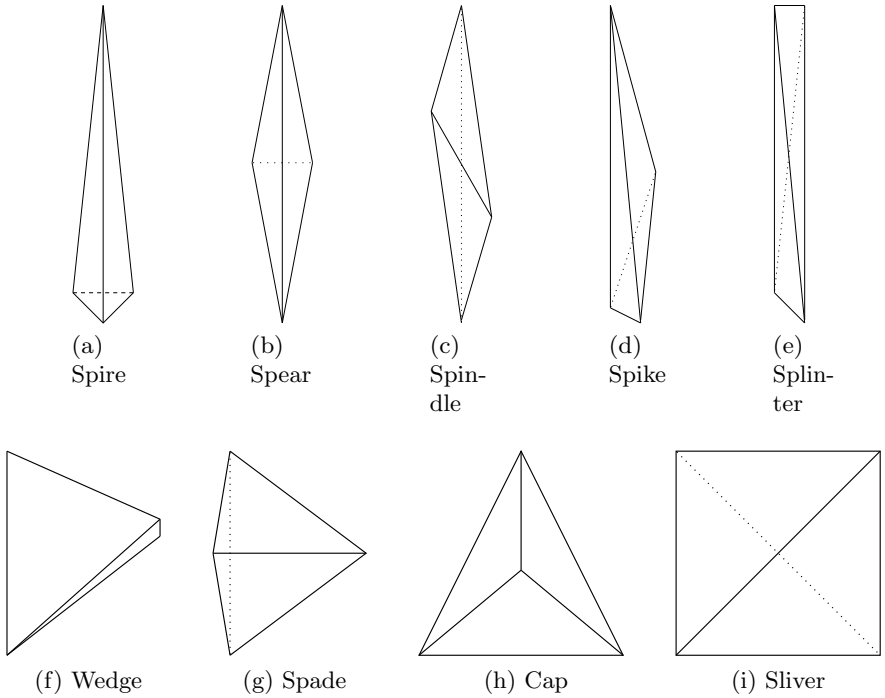


Figure B.1: Bad tetrahedra taxonomy borrowed from Cheng et al. [1]

Spear:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}u, \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{2}u, 0\right) \end{aligned}$$

Spindle:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2} + \tfrac{1-u}{6}, \tfrac{\sqrt{3}}{6}u, \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2} - \tfrac{1-u}{6}, \tfrac{\sqrt{3}}{2}u, 0\right) \end{aligned}$$

Spike:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (u,0,0) \\ v_2 &= \left(\tfrac{1}{2}u, \tfrac{1}{6}\left[3-\left(3-\sqrt{3}\right)u\right], \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2}u, \tfrac{1}{2}\left[2-\left(2-\sqrt{3}\right)u\right], 0\right) \end{aligned}$$

Splinter:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}, \tfrac{\sqrt{6}}{3}\right)u \\ v_3 &= \left(\tfrac{1}{2}(2-u), \tfrac{\sqrt{3}}{2}u, 0\right) \end{aligned}$$

Wedge:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}, \tfrac{\sqrt{6}}{3}\right)u \\ v_3 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{2}, 0\right) \end{aligned}$$

Spade:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}u, \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{2}, 0\right) \end{aligned}$$

Cap:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{6}, \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2}, \tfrac{\sqrt{3}}{2}, 0\right) \end{aligned}$$

Sliver:

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (1,0,0) \\ v_2 &= \left(\tfrac{1}{2}(2-u), 1-\left(1-\tfrac{\sqrt{3}}{6}\right)u, \tfrac{\sqrt{6}}{3}u\right) \\ v_3 &= \left(\tfrac{1}{2}u, 1-\left(1-\tfrac{\sqrt{3}}{2}\right)u, 0\right) \end{aligned}$$

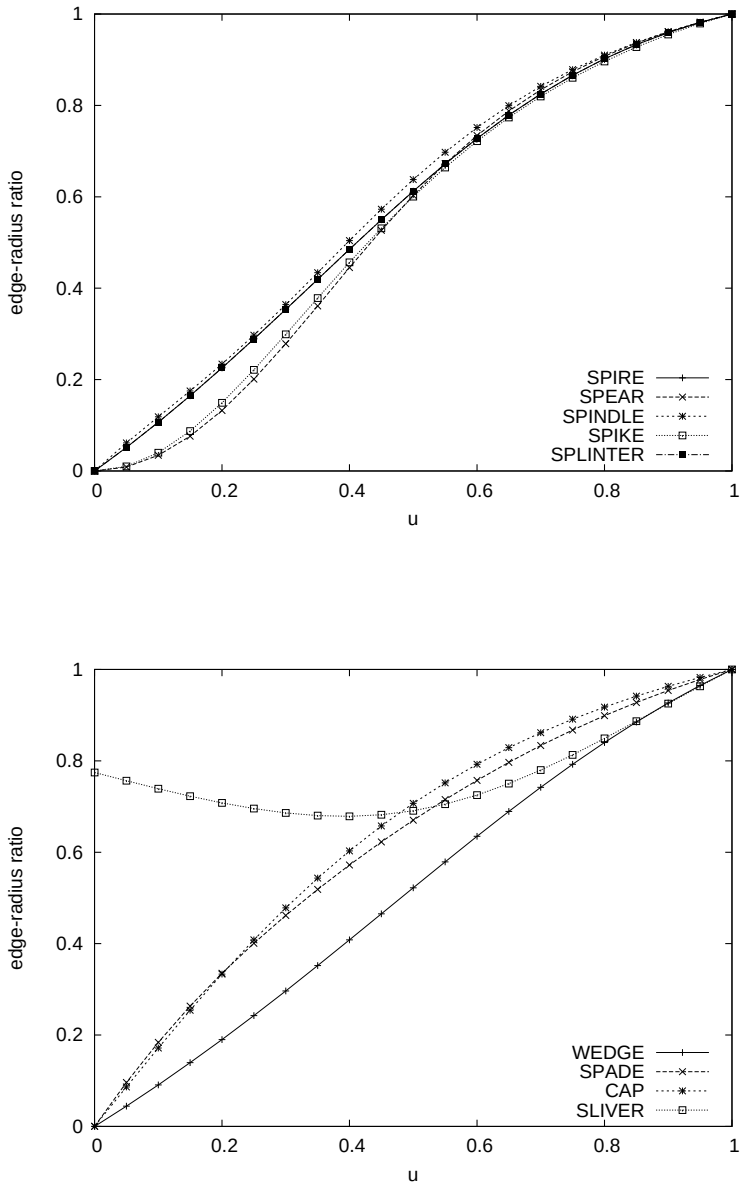


Figure B.2: Evolution of the *shortest edge-to-circumradius ratio* value as a function of parameter u

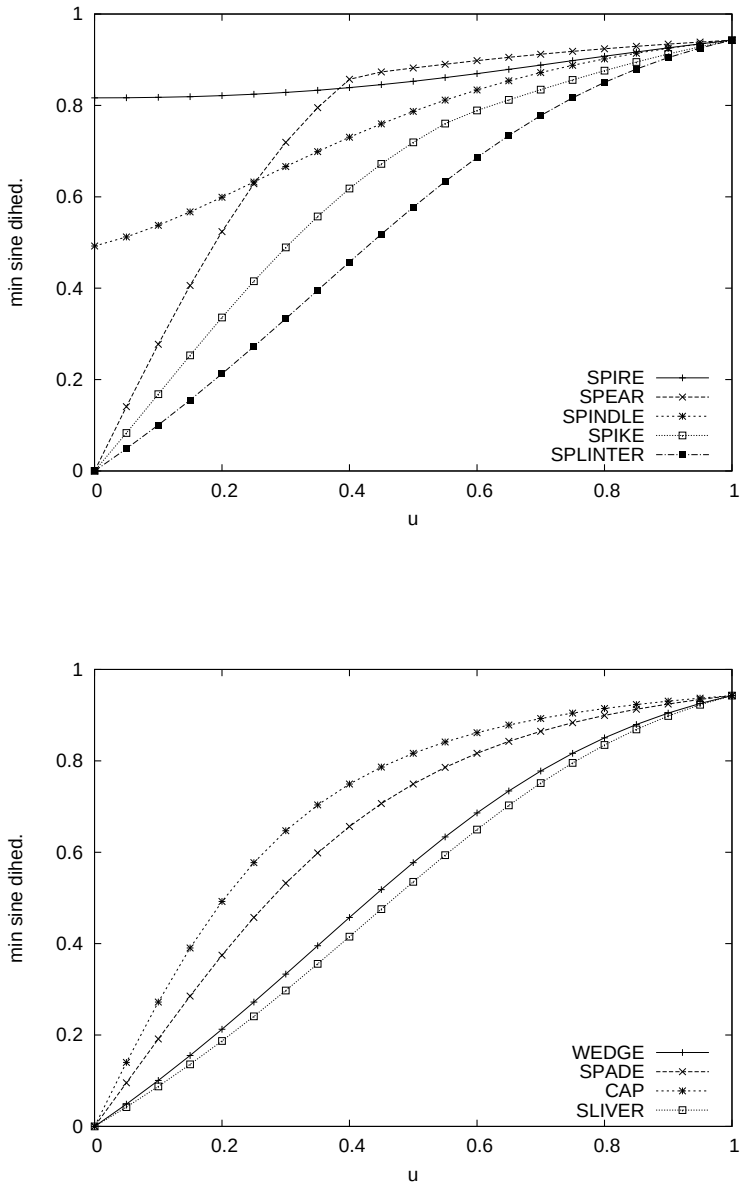


Figure B.3: Evolution of the *minimum sine of a tetrahedron's dihedral angles* value as a function of parameter u

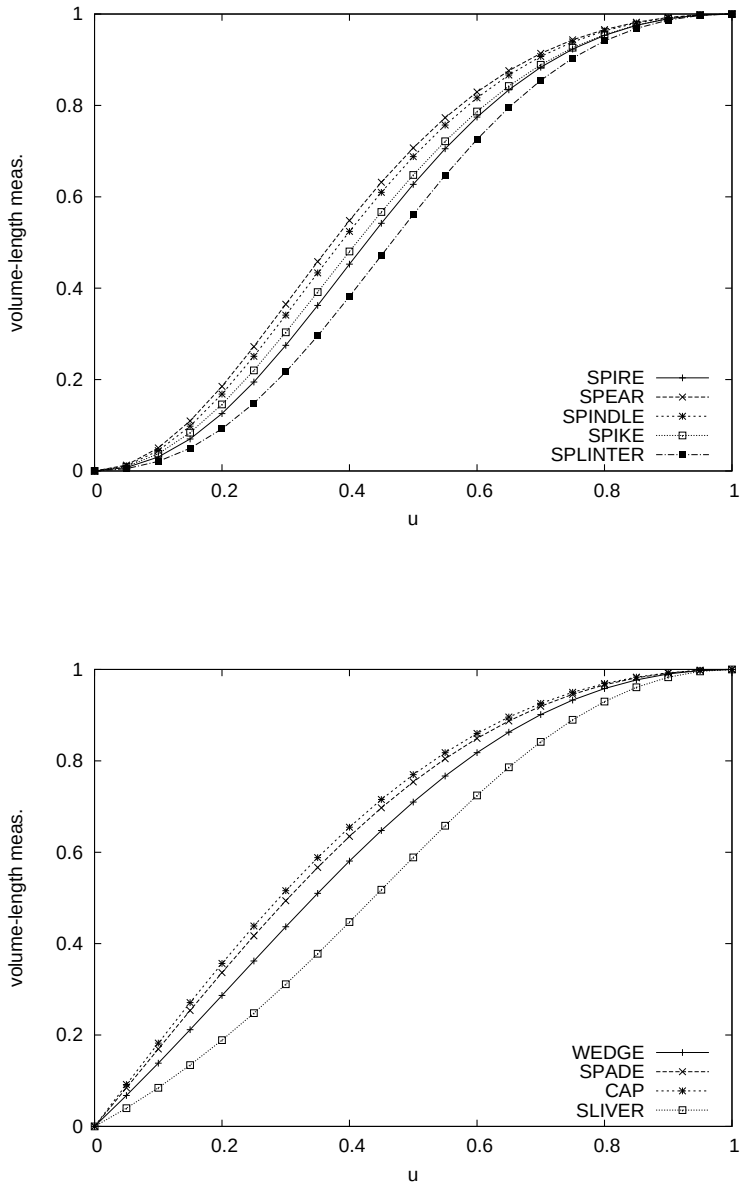


Figure B.4: Evolution of the *volume-length measure* value as a function of parameter u

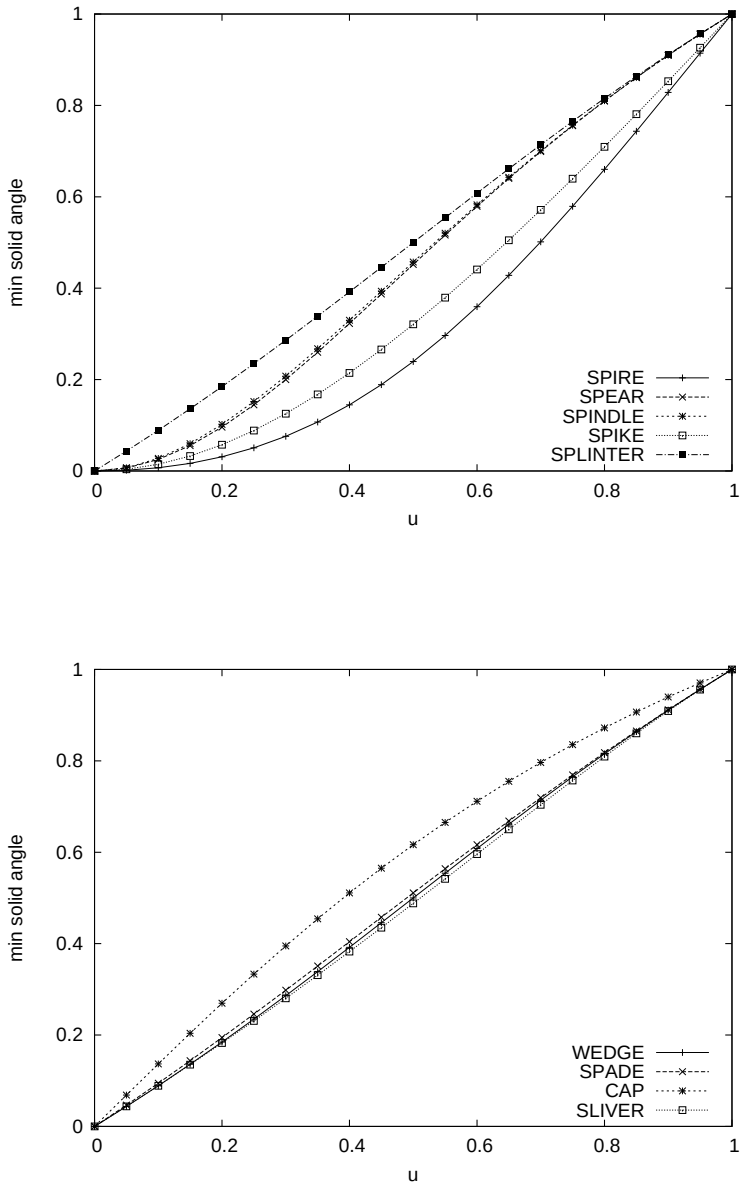


Figure B.5: Evolution of the *minimum solid angle* value as a function of parameter u

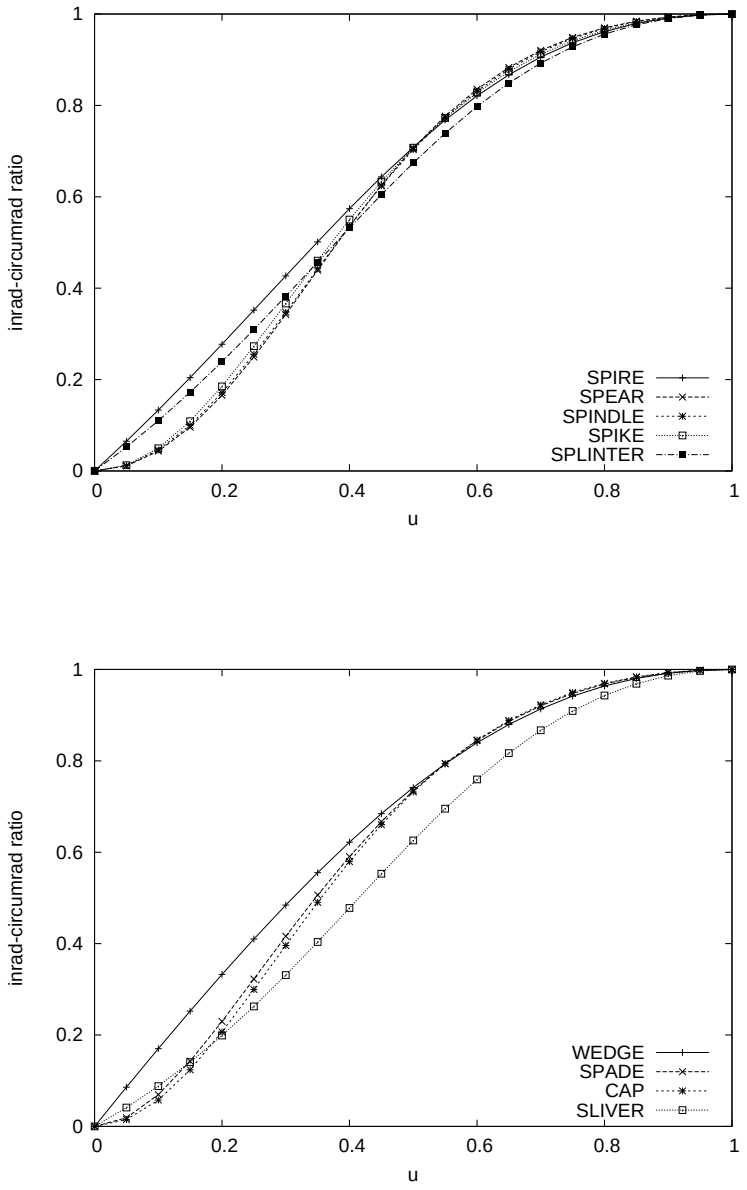


Figure B.6: Evolution of the *inradius-to-circumradius ratio* value as a function of parameter u

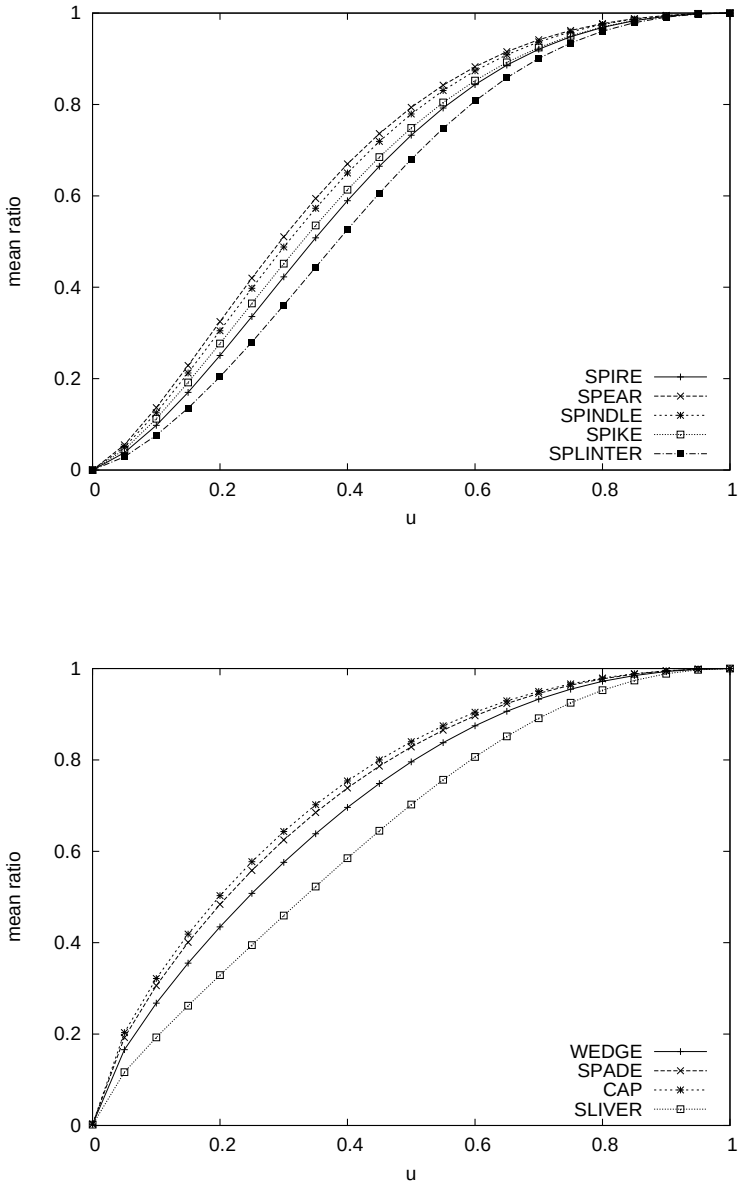


Figure B.7: Evolution of the *mean ratio* value as a function of parameter u

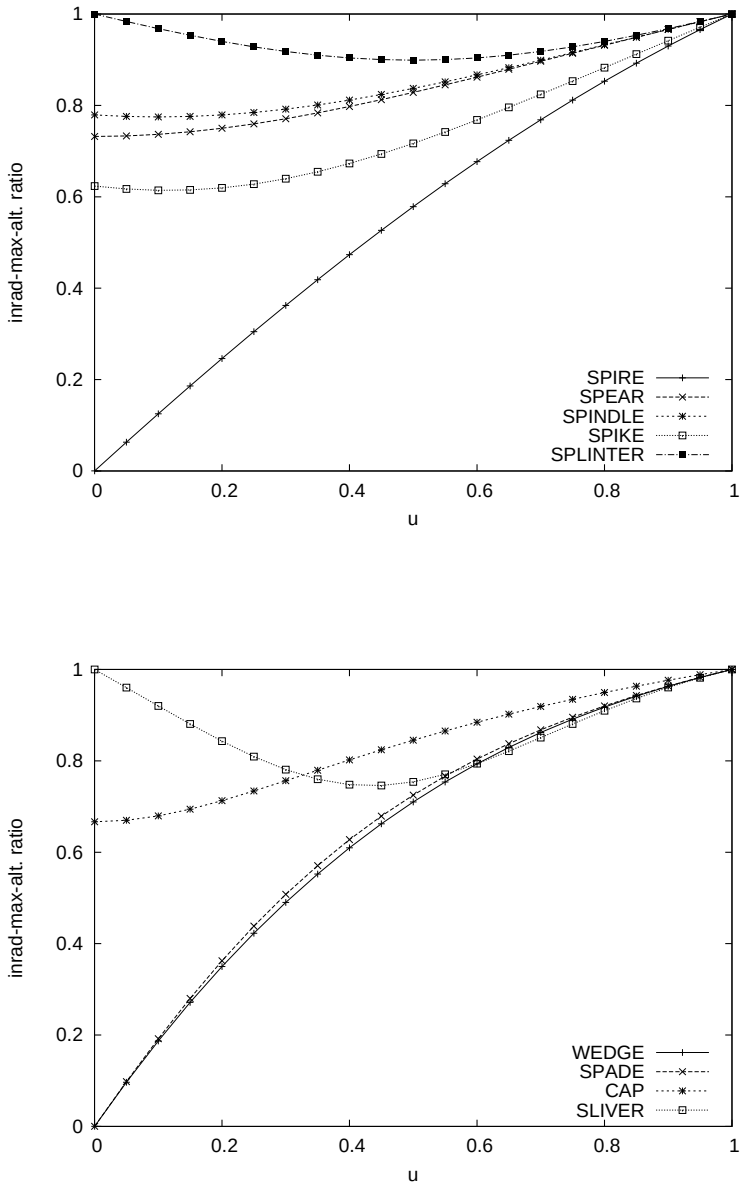


Figure B.8: Evolution of the *inradius-to-maximum altitude ratio* value as a function of parameter u

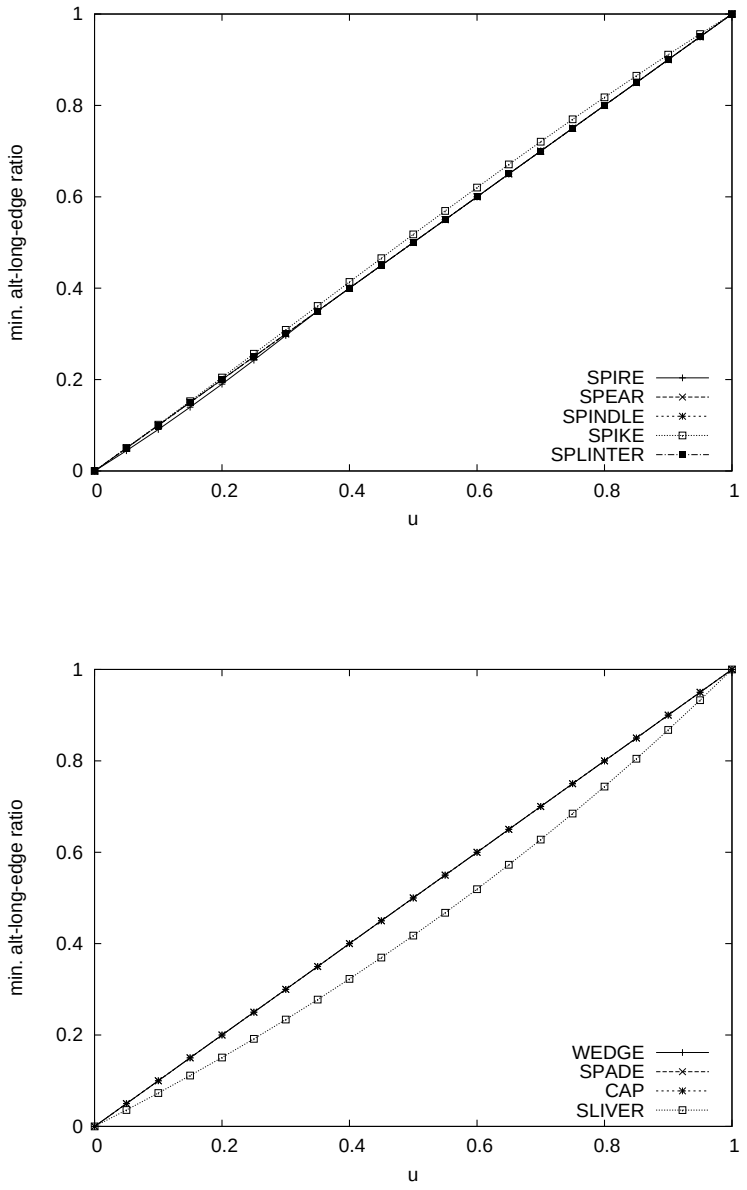


Figure B.9: Evolution of the *minimum altitude-to-longest-edge ratio* value as a function of parameter u

Measure	Num verts	Num tets	Dihedral ∠ Min	Dihedral ∠ Max	Radius ratio	Volume ratio
q_1	342	1269	1.63°	177.21°	0.03	0.03
q_2	291	999	18.74°	149.59°	0.30	0.24
q_3	277	911	17.66°	149.31°	0.32	0.30
q_4	281	934	15.32°	144.73°	0.36	0.30
q_5	394	1540	11.88°	160.16°	0.22	0.22
q_6	307	1079	15.85°	144.88°	0.38	0.33
q_7	415	1598	4.79°	171.89°	0.10	0.08
q_8	333	1193	15.48°	154.04°	0.33	0.28

Table B.1: Mesh data for the *clipped cube* model, refined using different quality measures. In all cases, the average dihedral angle was between 68.6°and 69.9°; the average radius ratio was between 0.72 and 0.76; and the average volume ratio was between 0.69 and 0.73.

The plots presented in Figures B.2 to B.9 show the evolution of the quality scores as poorly-shaped tetrahedra are transformed into regular tetrahedra following their parametrization.

B.2 Refined Mesh Quality Data

Tables B.1 to B.5 present the mesh quality data obtained by applying our Delaunay refinement algorithm to the five models of Figure 4.4. The algorithm is run successively using one of the eight quality measures defined in Section 4.5, until the worst element remaining in the mesh achieves a score equal or above the threshold corresponding to the measure in use. The following tables presents the number of vertices and tetrahedra generated with each refinement run. Quality data includes the extremal values of the dihedral angles, the minimum inradius-to-circumradius ratio times 3 (noted radius ratio) and the minimum ratio of a tetrahedron's volume to the l^2-norm of its edge lengths, times $6\sqrt{2}$ (noted volume ratio).

Measure	Num verts	Num tets	Dihedral ∠ Min	Dihedral ∠ Max	Radius ratio	Volume ratio
q_1	4934	21211	0.88°	178.36°	0.02	0.02
q_2	6122	27576	18.66°	154.11°	0.25	0.23
q_3	5064	21559	14.16°	155.13°	0.29	0.30
q_4	5393	23574	13.54°	152.45°	0.35	0.26
q_5	6497	29867	9.67°	162.92°	0.21	0.17
q_6	5724	25475	15.49°	152.54°	0.30	0.32
q_7	6534	29848	0.66°	178.58°	0.01	0.01
q_8	5975	26939	14.65°	155.70°	0.23	0.27

Table B.2: Mesh data for the *ridged torus* model, refined using different quality measures. In all cases, the average dihedral angle was between 69.5°and 69.8°; the average solid angle was between 28.6°and 29.4°; the average radius ratio was between 0.74 and 0.76; and the average volume ratio was between 0.71 and 0.73.

Measure	Num verts	Num tets	Dihedral ∠ Min	Dihedral ∠ Max	Radius ratio	Volume ratio
q_1	1343	4826	3.03°	174.98°	0.06	0.06
q_2	1336	4651	18.88°	151.32°	0.23	0.24
q_3	1212	4149	15.48°	153.14°	0.28	0.30
q_4	1179	3948	14.78°	150.52°	0.36	0.27
q_5	1294	4535	11.34°	144.93°	0.24	0.21
q_6	1295	4540	16.60°	150.78°	0.32	0.32
q_7	1959	7539	0.42°	179.30°	0.01	0.01
q_8	1307	4592	13.41°	154.37°	0.31	0.24

Table B.3: Mesh data for the *joint* model, refined using different quality measures. In all cases, the average dihedral angle was between 68.5°and 69.8°; the average solid angle was between 28.3°and 29.7°; the average radius ratio was between 0.74 and 0.75; and the average volume ratio was between 0.70 and 0.72.

Measure	Num verts	Num tets	Dihedral ∠ Min	Dihedral ∠ Max	Radius ratio	Volume ratio
q_1	2066	8432	1.03°	178.42°	0.02	0.02
q_2	2227	9110	18.71°	153.45°	0.18	0.22
q_3	1795	6936	14.54°	153.47°	0.28	0.30
q_4	1877	7294	13.53°	151.72°	0.35	0.27
q_5	2576	10785	9.83°	161.96°	0.23	0.18
q_6	2000	7958	14.63°	151.02°	0.31	0.32
q_7	2868	12294	1.03°	178.42°	0.02	0.02
q_8	1946	7640	14.07°	156.60°	0.27	0.24

Table B.4: Mesh data for the *wheel* model, refined using different quality measures. In all cases, the average dihedral angle was between 69.4°and 69.8°; the average solid angle was between 28.3°and 29.4°; the average radius ratio was between 0.73 and 0.76; and the average volume ratio was between 0.69 and 0.75.

Measure	Num verts	Num tets	Dihedral ∠ Min	Dihedral ∠ Max	Radius ratio	Volume ratio
q_1	3231	12276	1.67°	177.27°	0.03	0.03
q_2	2956	10985	18.67°	152.17°	0.30	0.23
q_3	2767	10094	15.01°	154.02°	0.31	0.30
q_4	2789	10244	14.21°	152.71°	0.35	0.29
q_5	3072	11598	10.91°	162.96°	0.22	0.19
q_6	2904	10768	15.05°	151.68°	0.32	0.32
q_7	3897	15443	1.34°	177.83°	0.02	0.02
q_8	2838	10366	15.52°	155.52°	0.29	0.27

Table B.5: Mesh data for the *rings* model, refined using different quality measures. In all cases, the average dihedral angle was between 69.5°and 69.8°; the average solid angle was between 28.5°and 29.3°; the average radius ratio was between 0.75 and 0.78; and the average volume ratio was between 0.72 and 0.77.

B.3 Dihedral Angle Distributions

In this section, we present the dihedral angle distributions found in the meshes shown in the paper. Every figure presents nine histograms arranged as follows:

- The first line shows the dihedral angle distributions in meshes obtained by applying Delaunay refinement only. The first column uses the circumradius-to-shortest-edge ratio (q_1) as quality measure, the second uses the minimum sine of a tetrahedron's dihedral angles (q_2) and the third, the volume-length measure (q_3).

- The second line shows data for improved meshes, where smoothing tries to maximize the minimum sine of dihedral angles. The improved mesh is obtained using the refined mesh (first row) of its corresponding column as a starting point.

- The third line shows data for improved meshes, where smoothing tries to maximize the volume-length measure. The improved mesh is obtained using the refined mesh (first row) of its corresponding column as a starting point.

All refined meshes are generated with $R = G = 1$ unless otherwise stated.

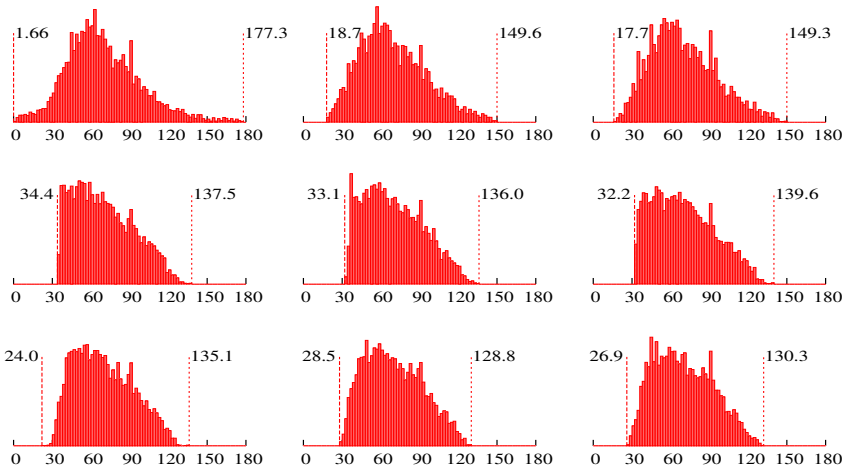


Figure B.10: Dihedral angle distributions for the *clipped cube* model.

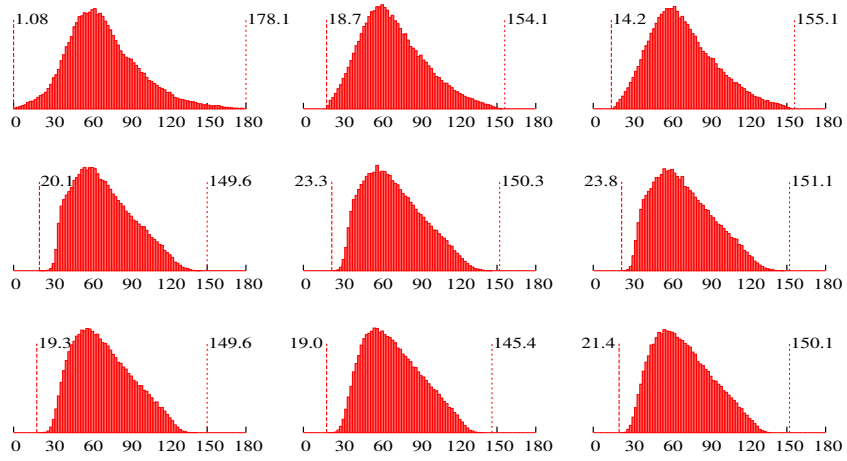


Figure B.11: Dihedral angle distributions for the *ridged torus* model.

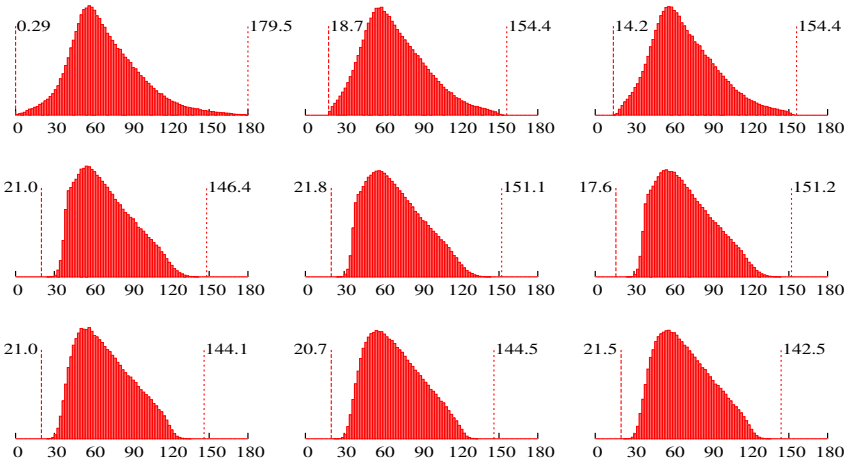


Figure B.12: Dihedral angle distributions for the *ridged torus* model, $R = 1$, $G = 15$.

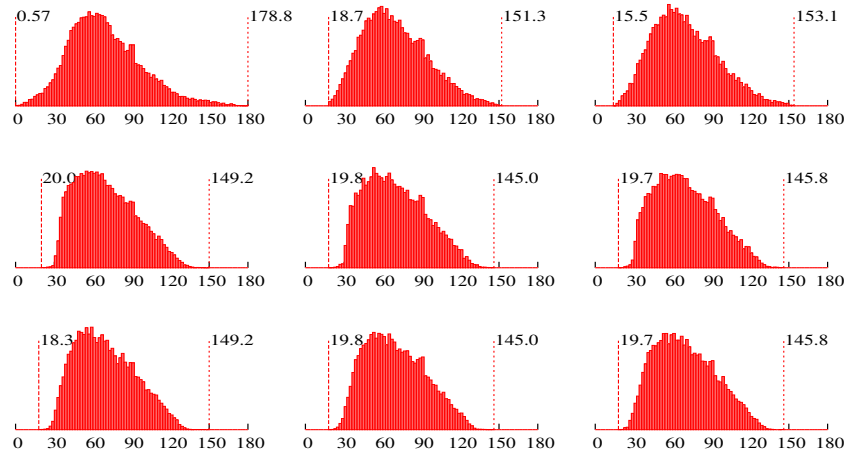


Figure B.13: Dihedral angle distributions for the *joint* model.

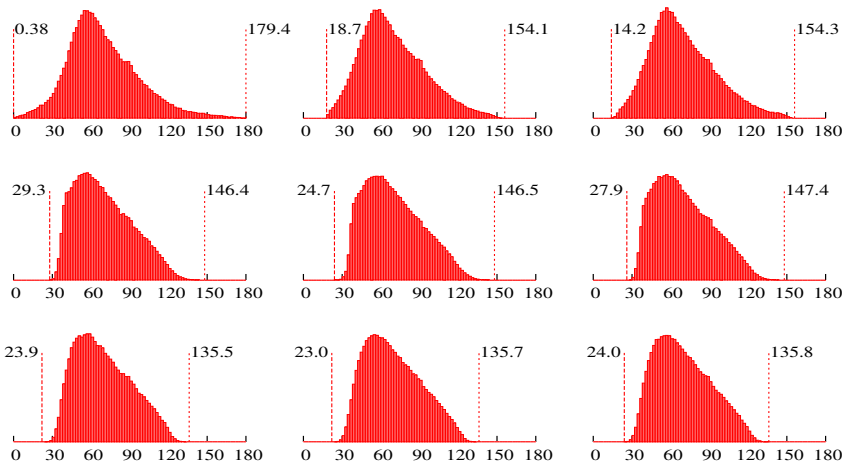


Figure B.14: Dihedral angle distributions for the *joint* model, $R = 2$, $G = 10$.

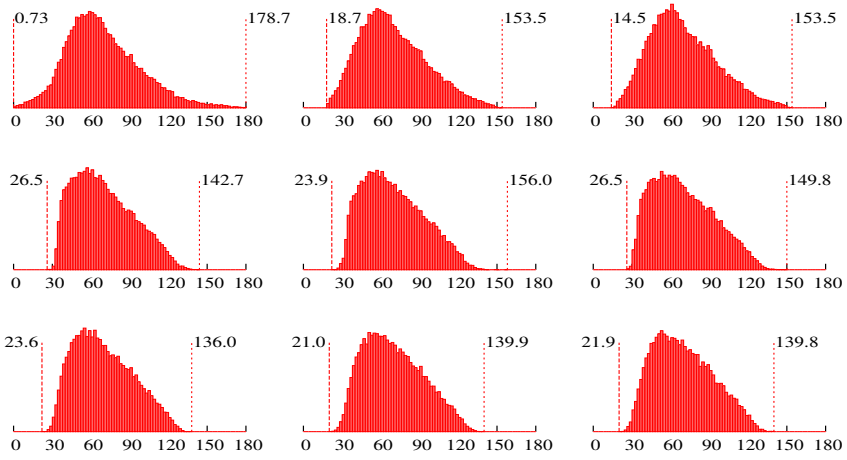


Figure B.15: Dihedral angle distributions for the *wheel* model.

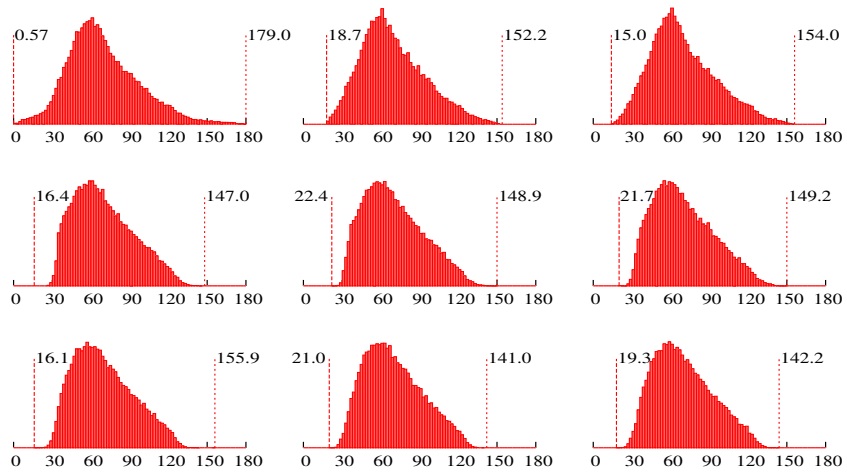


Figure B.16: Dihedral angle distributions for the *rings* model.

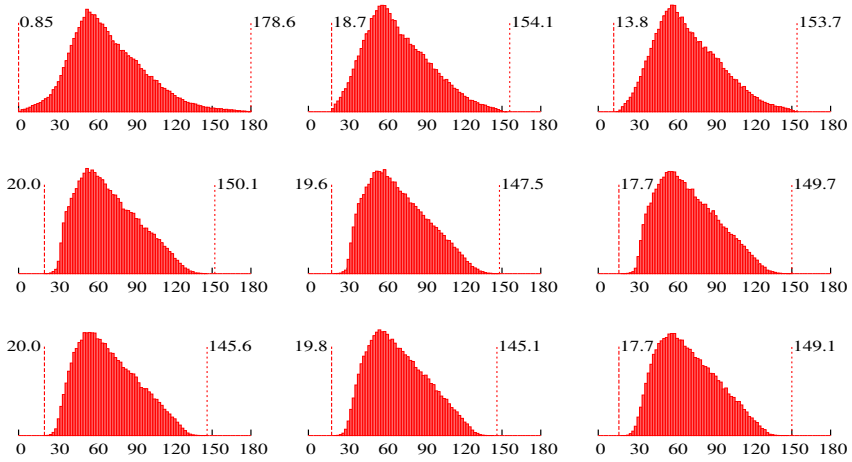


Figure B.17: Dihedral angle distributions for the *twist* model, $R = 2$, $G = 1$, $\lambda = 5$.

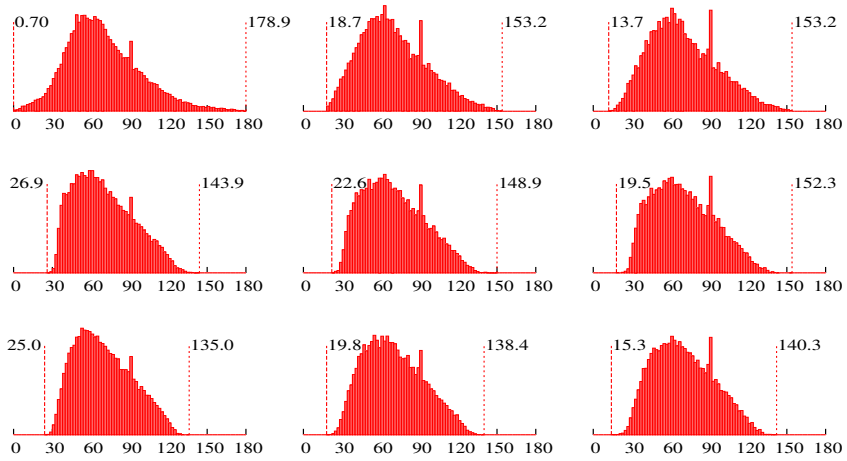


Figure B.18: Dihedral angle distributions for the *tire incinerator* model.

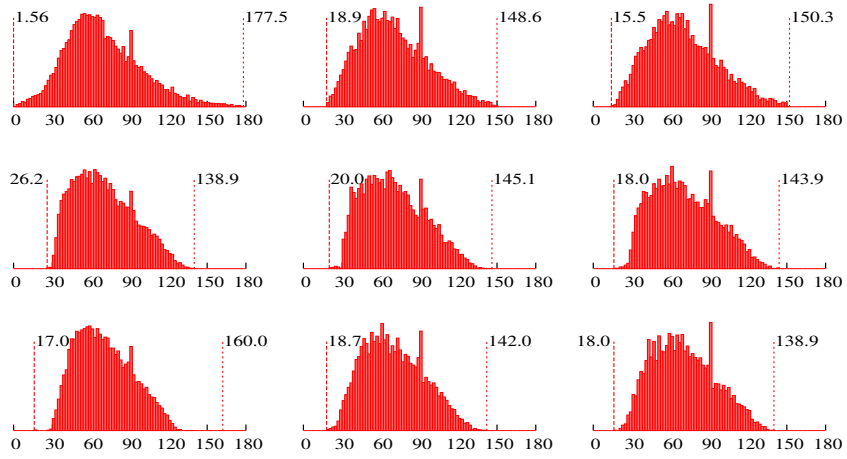


Figure B.19: Dihedral angle distributions for the *tangentially-fired boiler* model.

B.4 Bibliography

[1] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver exudation. *Journal of the ACM*, 47:883–904, 2000.

[2] Anwei Liu and Barry Joe. Relationship between tetrahedron shape measures. *BIT Numerical Mathematics*, 34(2):268–287, 1994.

[3] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[4] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.