Lecture Notes on Delaunay Mesh Generation

Jonathan Richard Shewchuk

September 20, 1999

Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720

Effort sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF under agreement number F30602-96-1-0287, in part by the National Science Foundation under Grant CMS-9318163, in part by the Natural Sciences and Engineering Research Council of Canada under a 1967 Science and Engineering Scholarship, and in part by a grant from the Intel Corporation. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Keywords: triangular mesh generation, tetrahedral mesh generation, Delaunay triangulation, Delaunay refinement, computational geometry

Contents

Chapter 1

Introduction

Meshes composed of triangles or tetrahedra are used in applications such as computer graphics, interpolation, surveying, and terrain databases. Although the algorithms described in this document have been used successfully to generate meshes for these and other purposes, the most challenges application of meshes are numerical methods for the solution of partial differential equations. Numerical methods such as the finite element method and the finite volume method are an irreplaceable means of simulating a wide variety of physical phenomena in scientific computing. They place particularly difficult demands on mesh generation. If one can generate meshes that are completely satisfying for numerical techniques like the finite element method, the other applications fall easily in line.

Delaunay refinement, the main topic of these notes, is a mesh generation technique that has theoretical guarantees to back up its good performance in practice. The bulk of these notes is an extensive exploration of the theory of Delaunay refinement in two and three dimensions, found in Chapters 3 and 4. Delaunay refinement is based upon a well-known geometric structure called the *Delaunay triangulation*, reviewed in Chapter 2.

This introductory chapter is devoted to explaining the problem that the remaining chapters undertake to solve. Unfortunately, the problem is not entirely well-defined. In a nutshell, one wishes to create a mesh that conforms to the geometry of the physical problem one wishes to model. This mesh must be composed of triangles or tetrahedra of appropriate sizes—possibly varying throughout the mesh—and these triangles or tetrahedra must be nicely shaped. Reconciling these constraints is not easy. Historically, the automation of mesh generation has proven to be more challenging than the entire remainder of the simulation process.

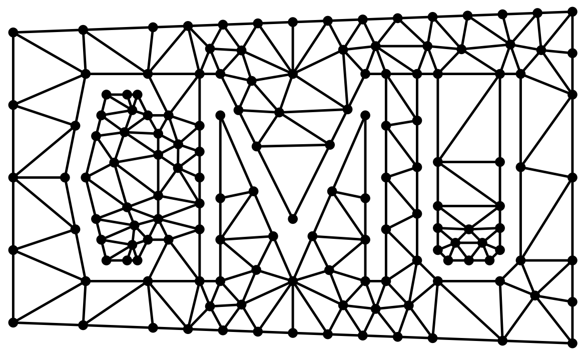
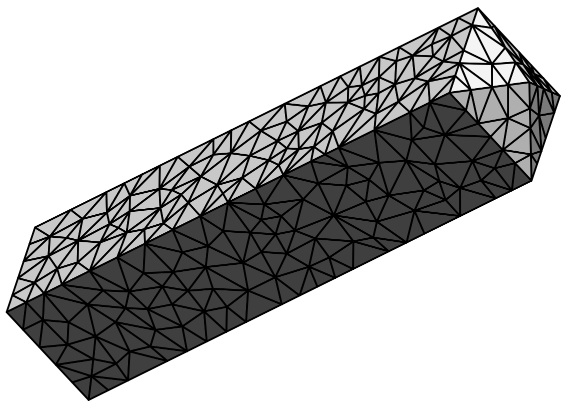



Figure 1.1: Two and three-dimensional finite element meshes. At left, each triangle is an element. At right, each tetrahedron is an element.

1.1 Meshes and Numerical Methods

Many physical phenomena in science and engineering can be modeled by partial differential equations (PDEs). When these equations have complicated boundary conditions or are posed on irregularly shaped objects or domains, they usually do not admit closed-form solutions. A numerical approximation of the solution is thus necessary.

Numerical methods for solving PDEs include the *finite element method* (FEM), the *finite volume method* (FVM, also known as the *control volume method*), and the *boundary element method* (BEM). They are used to model disparate phenomena such as mechanical deformation, heat transfer, fluid flow, electromagnetic wave propagation, and quantum mechanics. These methods numerically approximate the solution of a linear or nonlinear PDE by replacing the continuous system with a finite number of coupled linear or nonlinear algebraic equations. This process of *discretization* associates a variable with each of a finite number of points in the problem domain. For instance, to simulate heat conduction through an electrical component, the temperature is recorded at a number of points, called *nodes*, on the surface and in the interior of the component.

It is not enough to choose a set of points to act as nodes; the problem domain (or in the BEM, the boundary of the problem domain) must be partitioned into small pieces of simple shape. In the FEM, these pieces are called *elements*, and are usually triangles or quadrilaterals (in two dimensions), or tetrahedra or hexahedral bricks (in three dimensions). The FEM employs a node at every element vertex (and sometimes at other locations); each node is typically shared among several elements. The collection of nodes and elements is called a *finite element mesh*. Two and three-dimensional finite element meshes are illustrated in Figure 1.1. Because elements have simple shapes, it is easy to approximate the behavior of a PDE, such as the heat equation, on each element. By accumulating these effects over all the elements, one derives a system of equations whose solution approximates a set of physical quantities such as the temperature at each node.

The FVM and the BEM also use meshes, albeit with differences in terminology and differences in the meshes themselves. Finite volume meshes are composed of *control volumes*, which sometimes are clusters of triangles or tetrahedra, and sometimes are the cells of a geometric structure known as the *Voronoi diagram*. In either case, an underlying simplicial mesh is typically used to interpolate the nodal values and to generate the control volumes. Boundary element meshes do not partition an object; only its boundaries are partitioned. Hence, a two-dimensional domain would have boundaries divided into straight-line elements,

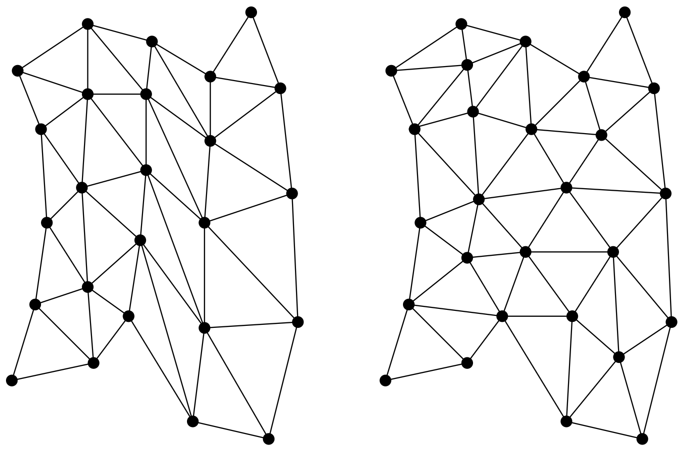


Figure 1.2: Structured (left) and unstructured (right) meshes. The structured mesh has the same topology as a square grid of triangles, although it is deformed enough that one might fail to notice its structure.

and a three-dimensional domain would have boundaries partitioned into polygonal (typically triangular) elements.

Meshes can (usually) be categorized as structured or unstructured. Figure 1.2 illustrates an example of each. Structured meshes exhibit a uniform topological structure that unstructured meshes lack. A functional definition is that in a structured mesh, the indices of the neighbors of any node can be calculated using simple addition, whereas an unstructured mesh necessitates the storage of a list of each node's neighbors.

The generation of both structured and unstructured meshes can be surprisingly difficult, each posing challenges of their own. This document considers only the task of generating unstructured meshes, and furthermore considers only simplicial meshes, composed of triangles or tetrahedra. Meshes with quadrilateral, hexahedral, or other non-simplicial elements are passed over, although they comprise an interesting field of study in their own right.

1.2 Desirable Properties of Meshes and Mesh Generators

Unfortunately, discretizing one's object of simulation is a more difficult problem than it appears at first glance. A useful mesh satisfies constraints that sometimes seem almost contradictory. A mesh must conform to the object or domain being modeled, and ideally should meet constraints on both the size and shape of its elements.

Consider first the goal of correctly modeling the shape of a problem domain. Scientists and engineers often wish to model objects or domains with complex shapes, and possibly with curved surfaces. Boundaries may appear in the interior of a region as well as on its exterior surfaces. *Exterior boundaries* separate meshed and unmeshed portions of space, and are found on the outer surface and in internal holes of a mesh. *Interior boundaries* appear within meshed portions of space, and enforce the constraint that elements may not pierce them. These boundaries are typically used to separate regions that have different physical properties; for example, at the contact plane between two materials of different conductivities in a heat propagation problem. An interior boundary is represented by a collection of edges (in two dimensions) or faces (in three dimensions) of the mesh.

In practice, curved boundaries can often be approximated by piecewise linear boundaries, so theoretical mesh generation algorithms are often based upon the idealized assumption that the input geometry is piecewise linear—composed without curves. This assumption is maintained throughout this document, and curved surfaces will not be given further consideration. This is not to say that the problem of handling curves is so easily waved aside; it surely deserves study. However, the simplified problem is difficult enough to provide ample gristle for the grinder.

Given an arbitrary straight-line two-dimensional region, it is not difficult to generate a triangulation that conforms to the shape of the region. It is trickier to find a tetrahedralization that conforms to an arbitrary linear three-dimensional region; some of the fundamental difficulties of doing so are described in Section 2.1.3. Nevertheless, the problem is reasonably well understood, and a thorough survey of the pertinent techniques, in both two and three dimensions, is offered by Bern and Eppstein [6].

A second goal of mesh generation is to offer as much control as possible over the sizes of elements in the mesh. Ideally, this control includes the ability to grade from small to large elements over a relatively short distance. The reason for this requirement is that element size has two effects on a finite element simulation. Small, densely packed elements offer more accuracy than larger, sparsely packed elements; but the computation time required to solve a problem is proportional to the number of elements. Hence, choosing an element size entails trading off speed and accuracy. Furthermore, the element size required to attain a given amount of accuracy depends upon the behavior of the physical phenomena being modeled, and may vary throughout the problem domain. For instance, a fluid flow simulation requires smaller elements amid turbulence than in areas of relative quiescence; in three dimensions, the ideal element in one part of the mesh may vary in volume by a factor of a million or more from the ideal element in another part of the mesh. If elements of uniform size are used throughout the mesh, one must choose a size small enough to guarantee sufficient accuracy in the most demanding portion of the problem domain, and thereby possibly incur excessively large computational demands. To avoid this pitfall, a mesh generator should offer rapid gradation from small to large sizes.

Given a *coarse* mesh—one with relatively few elements—it is not difficult to *refine* it to produce another mesh having a larger number of smaller elements. The reverse process is not so easy. Hence, mesh generation algorithms often set themselves the goal of being able, in principle, to generate as small a mesh as possible. (By "small", I mean one with as few elements as possible.) They typically offer the option to refine portions of the mesh whose elements are not small enough to yield the required accuracy.

A third goal of mesh generation, and the real difficulty, is that the elements should be relatively "round" in shape, because elements with large or small angles can degrade the quality of the numerical solution.

Elements with large angles can cause a large *discretization error*; the solution yielded by a numerical method such as the finite element method may be far less accurate than the method would normally promise. In principle, the computed discrete solution should approach the exact solution of the PDE as the size of the largest element approaches zero. However, Babuška and Aziz [2] show that if mesh angles approach 180° as the element size decreases, convergence to the exact solution may fail to occur.

Another problem caused by large angles is large errors in derivatives of the solution, which arise as an artifact of interpolation over the mesh. Figure 1.3 demonstrates the problem. The element illustrated has values associated with its nodes that represent an approximation of some physical quantity. If linear interpolation is used to estimate the solution at non-nodal points, the interpolated value at the center of the bottom edge is 51, as illustrated. This interpolated value depends only on the values associated with the bottom two nodes, and is independent of the value associated with the upper node. As the angle at the upper node approaches 180°, the interpolated point (with value 51) becomes arbitrarily close to the upper node (with value 48). Hence, the directional derivative of the estimated solution in the vertical direction may become arbitrarily large, and is clearly specious, even though the nodal values may themselves be perfectly accurate. This effect occurs because a linearly interpolated value is necessarily in error if the true solution is

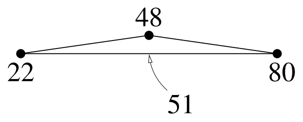


Figure 1.3: The nodal values depicted may represent an accurate estimate of the correct solution. Nevertheless, as the large angle of this element approaches $180°$, the vertical directional derivative, estimated via linear interpolation, becomes arbitrarily large.

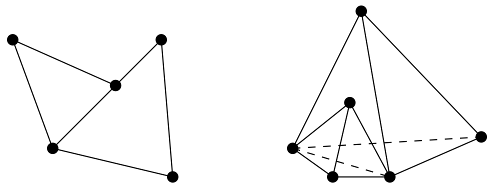


Figure 1.4: Elements are not permitted to meet in the manner depicted here.

not linear, and any error in the derivative computation is magnified because of the large angle. This problem can afflict any application that uses meshes for interpolation, and not just PDE solvers. However, it is of particular concern in simulations of mechanical deformation, in which the derivatives of a solution (the strains) are of interest, and not the solution itself (the displacements). The problem is partly, but not entirely, ameliorated by the use of a higher-degree (rather than piecewise linear) interpolating surface.

Small angles are also feared, because they can cause the coupled systems of algebraic equations that numerical methods yield to be ill-conditioned [10]. If a system of equations is ill-conditioned, roundoff error degrades the accuracy of the solution if the system is solved by direct methods, and convergence is slow if the system is solved by iterative methods.

By placing a lower bound on the smallest angle of a triangulation, one is also bounding the largest angle; for instance, in two dimensions, if no angle is smaller than θ, then no angle is larger than $180° - 2\theta$. Hence, many mesh generation algorithms take the approach of attempting to bound the smallest angle.

Despite this discussion, the effects of element shape on numerical methods such as the finite element method are still being investigated. Our understanding of the relative merit of different metrics for measuring element quality, or the effects of small numbers of poor quality elements on numerical solutions, is based as much on engineering experience and rumor as it is on mathematical foundations. Furthermore, the notion of a nicely shaped element varies depending on the numerical method, the type of problem being solved, and the polynomial degree of the piecewise functions used to interpolate the solution over the mesh. For physical phenomena that have anisotropic behavior, the ideal element may be long and thin, despite the claim that small angles are usually bad. Hence, the designer of algorithms for mesh generation is shooting at an ill-defined target.

The constraints of element size and element shape are difficult to reconcile because elements must meet squarely along the full extent of their shared edges or faces. Figure 1.4 illustrates illegal meetings between adjacent elements. For instance, at left, the edge of one triangular element is a portion of an edge of an adjoining element. There are variants of methods like the finite element method that permit such *nonconforming elements*. However, such elements are not preferred, as they may degrade or ruin the convergence of the method. Although nonconforming elements make it easier to create a mesh with seemingly nicely shaped elements, the problems of numerical error may still persist.

For an example of how element quality and mesh size are traded off, look ahead to Figure 3.18 on Page 69.

1.3 Why Unstructured Meshes?

Is it really worth the trouble to use unstructured meshes? The process of solving the linear or nonlinear systems of equations yielded by the finite element method and its brethren is simpler and faster on structured meshes, because of the ease of determining each node's neighbors. Because unstructured meshes necessitate the storage of pointers to each node's neighbors, their demands on storage space and memory traffic are greater. Furthermore, the regularity of structured meshes makes it straightforward to parallelize computations upon them, whereas unstructured meshes engender the need for sophisticated partitioning algorithms and parallel unstructured solvers.

Nonetheless, there are cases in which unstructured meshes are preferable or even indispensable. Many problems are defined on irregularly shaped domains, and resist structured discretization. Several more subtle advantages of unstructured meshes are visible in Figures 1.6 and 1.7, which depict meshes used to model a cross-section of the Los Angeles Basin, itself illustrated in Figure 1.5.

A numerical method is used to predict the surface ground motion due to a strong earthquake. The mesh of Figure 1.7 is finer in the top layers of the valley, reflecting the much smaller wavelength of seismic waves in the softer upper soil, and becomes coarser with increasing depth, as the soil becomes stiffer and the corresponding seismic wavelength increases by a factor of twenty. Whereas an unstructured mesh can be flexibly tailored to the physics of this problem, the structured mesh must employ a uniform horizontal distribution of nodes, the density being dictated by the uppermost layer. As a result, it has five times as many nodes as the unstructured mesh, and the solution time and memory requirements of the simulation are correspondingly larger. The disparity is even more pronounced in three-dimensional domains and in simulations where the scales of the physical phenomena vary more.

Another important difference is that the mesh of Figure 1.7 conforms to the interior boundaries of the basin in a way that the mesh of Figure 1.6 cannot, and hence may better model reflections of waves from the interfaces between layers of soil with differing densities. This difference in accuracy only manifests itself if the unstructured and structured meshes under comparison are relatively coarse.

Unstructured meshes, far better than structured meshes, can provide multiscale resolution and conformity to complex geometries.

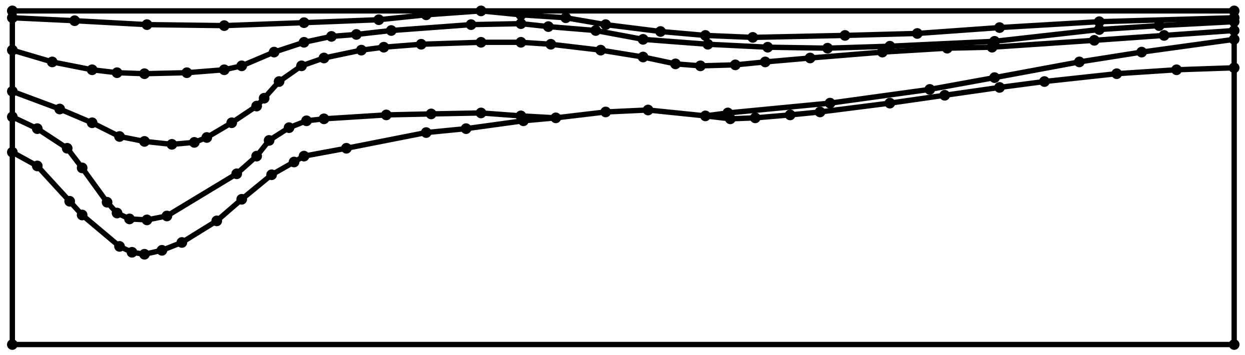


Figure 1.5: Los Angeles Basin.

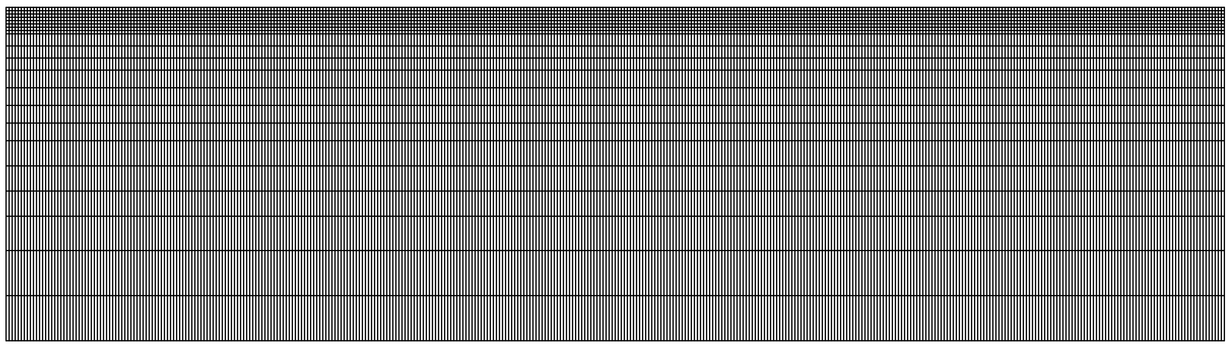


Figure 1.6: Structured mesh of Los Angeles Basin.

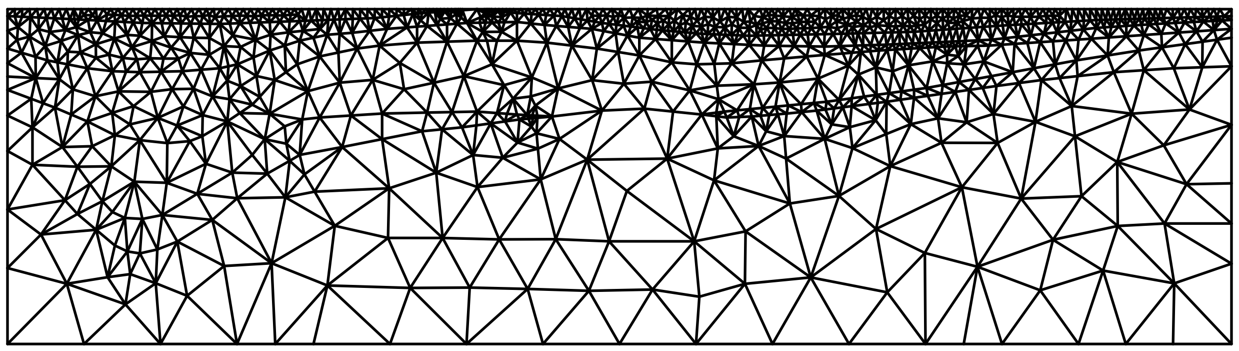


Figure 1.7: Unstructured mesh of Los Angeles Basin.

Chapter 2

The Delaunay Triangulation and Unstructured Mesh Generation

The Delaunay triangulation is a geometric structure that has enjoyed great popularity in mesh generation since mesh generation was in its infancy. In two dimensions, it is not hard to understand why: the Delaunay triangulation of a vertex set maximizes the minimum angle among all possible triangulations of that vertex set. If one is concerned with element quality, it seems almost silly to consider using a triangulation that is not Delaunay.

This chapter surveys Delaunay triangulations in two or more dimensions, their properties, and several algorithms for constructing them. Delaunay tetrahedralizations are not quite so effective as two-dimensional Delaunay triangulations at producing elements of good quality, but they have nevertheless enjoyed nearly as much popularity in the mesh generation literature as their two-dimensional cousins. I focus only on details relevant to mesh generation; for more general surveys, Aurenhammer [1] and Fortune [25] are recommended. I also discuss constrained Delaunay triangulations, which ensure that specified segments and facets are present in the mesh.

Also found in this chapter is a brief survey of research in mesh generation, with special attention given to methods based on Delaunay triangulations and tetrahedralizations, and methods that generate meshes that are guaranteed to have favorable qualities. These algorithms are part of the history that led to the discovery of the provably good Delaunay refinement algorithms studied in Chapters 3 and 4.

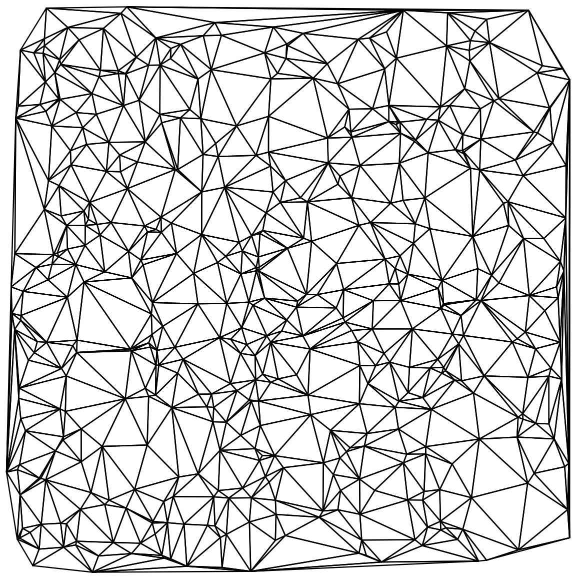


Figure 2.1: A Delaunay triangulation.

2.1 Delaunay Triangulations

2.1.1 The Delaunay Triangulation in Two Dimensions

In two dimensions, a *triangulation* of a set V of vertices is a set T of triangles whose vertices collectively are V, whose interiors do not intersect each other, and whose union is the convex hull of V, if every triangle intersects V only at the triangle's vertices.

The *Delaunay triangulation* D of V, introduced by Delaunay [20] in 1934, is the graph defined as follows. Any circle in the plane is said to be *empty* if it encloses no vertex of V. (Vertices are permitted on the circle.) Let u and v be any two vertices of V. A *circumcircle* (circumscribing circle) of the edge uv is any circle that passes through u and v. Any edge has an infinite number of circumcircles. The edge uv is in D if and only if there exists an empty circumcircle of uv. An edge satisfying this property is said to be *Delaunay*. Figure 2.1 illustrates a Delaunay triangulation.

As defined above, the Delaunay triangulation of a vertex set is clearly unique, because the definition specifies an unambiguous test for the presence or absence of an edge in the triangulation. Every edge that lies on the boundary of the convex hull of a vertex set and has no vertex in its interior is Delaunay. Figure 2.2 illustrates the reason why. For any convex hull edge e, it is always possible to find an empty circumcircle of e by starting with the smallest circumcircle of e and "growing" it away from the triangulation.

Every edge connecting a vertex to its nearest neighbor is Delaunay. If w is the vertex nearest v, the smallest circle passing through v and w does not enclose any vertices.

It's not at all obvious that the set of Delaunay edges of a vertex set collectively forms a triangulation. For the definition I have given above, the Delaunay triangulation is guaranteed to be a triangulation only if the vertices of V are in *general position*, here meaning that no four vertices of V lie on a common circle. As a first step to proving this guarantee, I describe the notion of a Delaunay triangle. The *circumcircle* of a triangle is the unique circle that passes through all three of its vertices. A triangle is said to be *Delaunay*

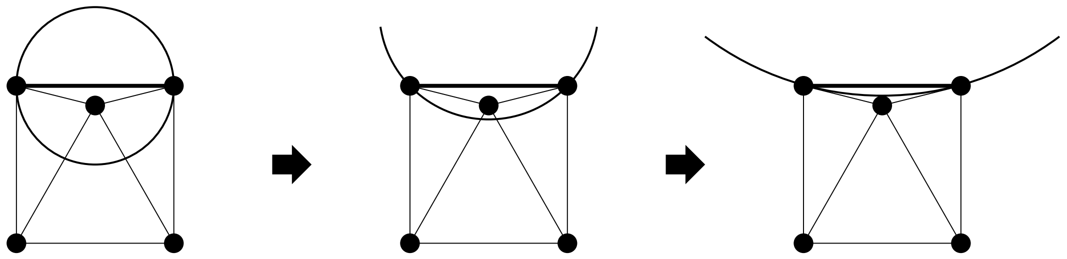


Figure 2.2: Each edge on the convex hull is Delaunay, because it is always possible to find an empty circle that passes through its endpoints.

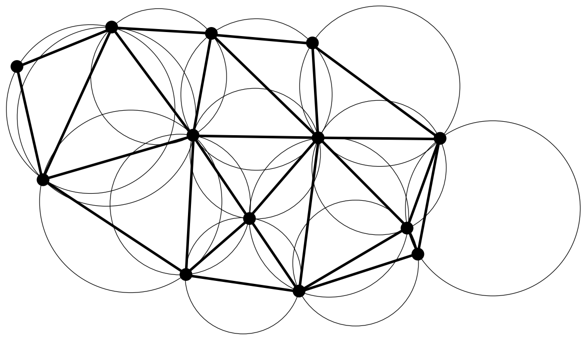


Figure 2.3: Every triangle of a Delaunay triangulation has an empty circumcircle.

if and only if its circumcircle is empty. This defining characteristic of Delaunay triangles, illustrated in Figure 2.3, is called the *empty circumcircle property*.

Lemma 1 *Let T be a triangulation. If all the triangles of T are Delaunay, then all the edges of T are Delaunay, and vice versa.*

Proof: If all the triangles of T are Delaunay, then the circumcircle of every triangle is empty. Because every edge of T belongs to a triangle of T, every edge has an empty circumcircle, and is thus Delaunay.

If all the edges of T are Delaunay, suppose for the sake of contradiction that some triangle t of T is not Delaunay. Because T is a triangulation, t cannot contain any vertices (except its corners), so some vertex v of T lies inside the circumcircle of t, but outside t itself. Let e be the edge of t that separates v from the interior of t, and let w be the vertex of t opposite e, as illustrated in Figure 2.4. One cannot draw a circumcircle of e that encloses neither v nor w, so e is not Delaunay. The result follows by contradiction. ■

The method by which I prove that the Delaunay triangulation is a triangulation is somewhat nonintuitive. I will describe a well-known algorithm called the *flip algorithm*, and show that all the edges of the triangulation produced by the flip algorithm are Delaunay. Then I will show that no other edges are Delaunay.

The flip algorithm begins with an arbitrary triangulation of V, and searches for an edge that is not *locally Delaunay*. All edges on the boundary (convex hull) of the triangulation are considered to be locally Delaunay. For any edge e not on the boundary, the condition of being locally Delaunay is similar to the condition of being Delaunay, but only the two triangles that contain e are considered. For instance, Figure 2.5 demonstrates two different ways to triangulate a subset of four vertices. In the triangulation at left, the edge e is locally Delaunay, because the depicted circumcircle of e does not enclose either of the vertices opposite

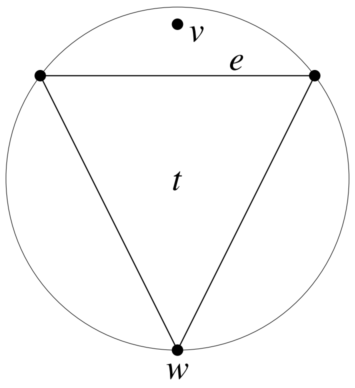


Figure 2.4: If the triangle t is not Delaunay, then at least one of its edges (in this case, e) is not Delaunay.

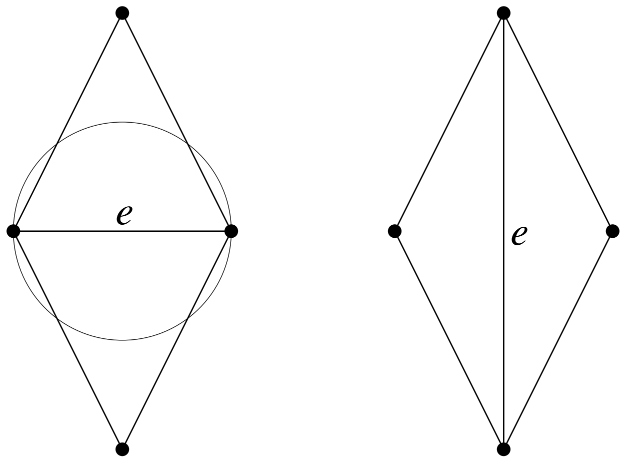


Figure 2.5: Two triangulations of a vertex set. At left, e is locally Delaunay; at right, e is not.

e in the two triangles that contain e. In the triangulation at right, e is not locally Delaunay, because the two vertices opposite e preclude the possibility that e has an empty circumcircle. Observe that if the triangles at left are part of a larger triangulation, e might not be Delaunay, because vertices that lie in neither triangle may lie in e's circumcircle. However, such vertices have no bearing on whether or not e is locally Delaunay.

Whenever the flip algorithm identifies an edge that is not locally Delaunay, the edge is *flipped*. To flip an edge is to delete it, thereby combining the two containing triangles into a single *containing quadrilateral*, and then to insert the crossing edge of the quadrilateral. Hence, an edge flip could convert the triangulation at left in Figure 2.5 into the triangulation at right, or vice versa. (The flip algorithm would perform only the latter flip.) Not all triangulation edges are flippable, as Figure 2.6 shows, because the containing quadrilateral of an edge might not be convex.

Lemma 2 *Let e be an edge of a triangulation of V. Either e is locally Delaunay, or e is flippable and the edge created by flipping e is locally Delaunay.*

Proof: Let v and w be the vertices opposite e, which together with e define the containing quadrilateral of e, illustrated in Figure 2.7. Let C be the circle that passes through v and the endpoints of e. Either w is strictly inside C, or w lies on or outside C.

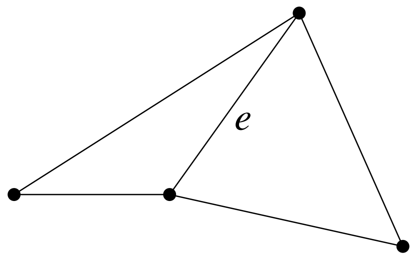


Figure 2.6: In this concave quadrilateral, e cannot be flipped.

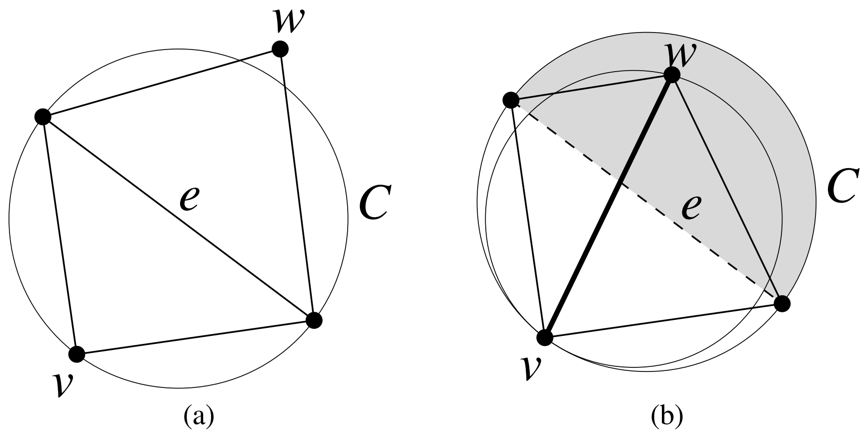


Figure 2.7: (a) Case where e is locally Delaunay. (b) Case where e is not locally Delaunay. The edge created if e is flipped is locally Delaunay.

If w is on or outside C, as in Figure 2.7(a), then the empty circle C demonstrates that e is locally Delaunay.

If w is inside C, then w is contained in the section of C bounded by e and opposite v; this section is shaded in Figure 2.7(b). The containing quadrilateral of e is thus constrained to be strictly convex, so the edge e is flippable. Furthermore, the circle that passes through v and w, and is tangent to C at v, does not enclose the endpoints of e, as Figure 2.7(b) demonstrates; hence the edge vw is locally Delaunay. ■

The success of the flip algorithm relies on the fact, proven below, that if any edge of the triangulation is not Delaunay, then there is an edge that is not locally Delaunay, and can thus be flipped.

Lemma 3 *Let T be a triangulation whose edges are all locally Delaunay. Then every edge of T is (globally) Delaunay.*

Proof: Suppose for the sake of contradiction that all edges of T are locally Delaunay, but some edge of T is not Delaunay. By Lemma 1, the latter assertion implies that some triangle t of T is not Delaunay. Let v be a vertex inside the circumcircle of t, and let e_1 be the edge of t that separates v from the interior of t, as illustrated in Figure 2.8(a). Without loss of generality, assume that e_1 is oriented horizontally, with t below e_1.

Draw a line segment from the midpoint of e_1 to v (see the dashed line in Figure 2.8(a)). Let e_1, e_2, e_3, …, e_m be the sequence of triangulation edges (from bottom to top) whose interiors this line segment intersects. (If the line segment intersects some vertex other than v, replace v with the first such vertex.)

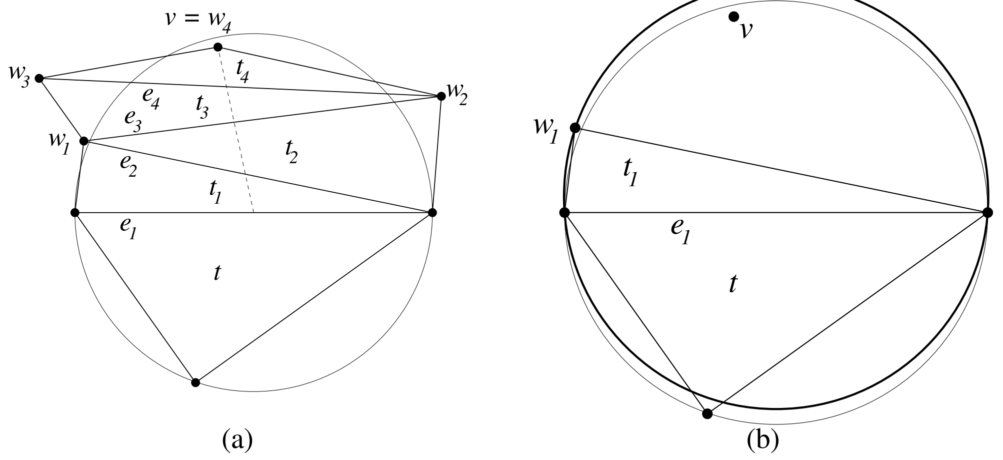


Figure 2.8: (a) If v lies inside the circumcircle of t, there must be an edge between v and t that is not locally Delaunay. (b) Because v lies above e_1 and inside the circumcircle of t, and because w_1 lies outside the circumcircle of t, v must lie inside the circumcircle of t_1.

Let w_i be the vertex above e_i that forms a triangle t_i in conjunction with e_i. Because T is a triangulation, $w_m = v$.

By assumption, e_1 is locally Delaunay, so w_1 lies on or outside the circumcircle of t. As Figure 2.8(b) shows, it follows that the circumcircle of t_1 encloses every point above e_1 in the circumcircle of t, and hence encloses v. Repeating this argument inductively, one finds that the circumcircle of t_m encloses v. But $w_m = v$ is a vertex of t_m, which contradicts the claim that v is inside the circumcircle of t_m. ■

An immediate consequence of Lemma 3 is that if a triangulation contains an edge that is not Delaunay, then it contains an edge that is not locally Delaunay, and thus the flip algorithm may proceed. The following lemma shows that the flip algorithm cannot become trapped in an endless loop.

Lemma 4 *Given a triangulation of n vertices, the flip algorithm terminates after $\mathcal{O}(n^2)$ edge flips, yielding a triangulation whose edges are all Delaunay.*

Proof: Let $\Phi(T)$ be a function defined over all triangulations, equal to the number of vertex-triangle pairs (v, t) such that v is a vertex of T, t is a triangle of T, and v lies inside the circumcircle of t. Because T has n vertices and $\mathcal{O}(n)$ triangles, $\Phi(T) \in \mathcal{O}(n^2)$. Clearly, $\Phi(T) = 0$ if and only if T is Delaunay.

Suppose an edge e of T is flipped, forming a new triangulation T'. Let t_1 and t_2 be the triangles containing e (before the flip), and let v_1 and v_2 be the apices of t_1 and t_2. Because e is not locally Delaunay, v_1 is inside the circumcircle of t_2, and v_2 is inside the circumcircle of t_1. Let t_1' and t_2' be the triangles that replace t_1 and t_2 after the edge flip. Let C_1, C_2, C_1', and C_2' be the open circumdisks of t_1, t_2, t_1', and t_2' respectively, as illustrated in Figure 2.9(a). (An open circumdisk is the set of points inside a circumcircle, omitting points on the circumcircle itself.)

It is not difficult to show that $C_1 \cup C_2 \supset C_1' \cup C_2'$ (Figure 2.9(b)) and $C_1 \cap C_2 \supset C_1' \cap C_2'$ (Figure 2.9(c)). Therefore, if a vertex w lies inside n_w circumcircles of triangles of T, and hence contributes n_w to the count $\Phi(T)$, then w lies inside no more than n_w circumcircles of triangles of T', and contributes at most n_w to the count $\Phi(T')$. If, after the edge flip, a vertex is counted because it lies in C_1' or C_2', then it must have lain in C_1 or C_2 before the edge flip; and if it lies in both C_1' and C_2', then it must have lain in both C_1 and C_2.

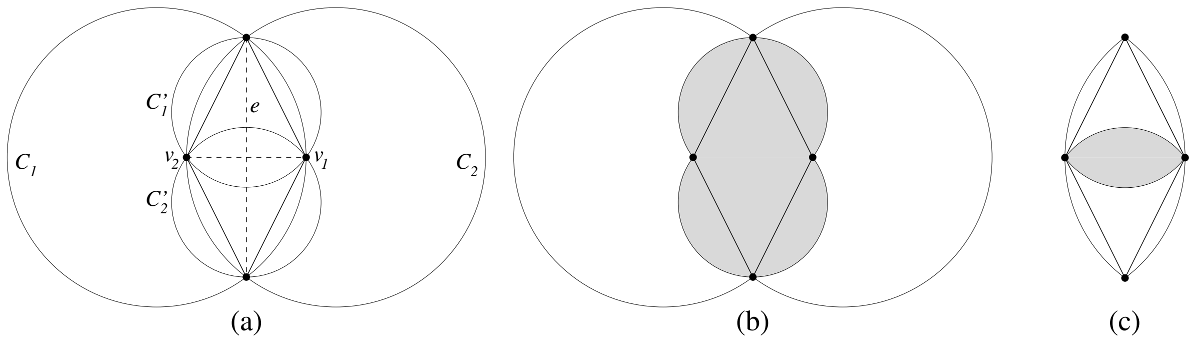


Figure 2.9: (a) Circumdisks before and after an edge flip. (b) The union of the circumdisks afterward (shaded) is contained in the union of the prior circumdisks. (c) The intersection of the circumdisks afterward (shaded) is contained in the intersection of the prior circumdisks.

However, the vertices v_1 and v_2 each lie inside one less circumcircle than before the edge flip. For instance, v_1 lay in C_2, but lies in neither C'_1 nor C'_2. Hence, $\Phi(T') \leq \Phi(T) - 2$.

The flip algorithm terminates after $\mathcal{O}(n^2)$ edge flips because $\Phi \in \mathcal{O}(n^2)$, every edge flip reduces Φ by at least two, and Φ cannot fall below zero. The flip algorithm terminates only when every edge is locally Delaunay; thus, by Lemma 3, every edge is Delaunay. ■

Theorem 5 *Let V be a set of three or more vertices in the plane that are not all collinear. If no four vertices of V are cocircular, the Delaunay triangulation of V is a triangulation, and is produced by the flip algorithm.*

Proof: Because the vertices of V are not all collinear, there exists a triangulation of V. By Lemma 4, the application of the flip algorithm to any triangulation of V produces a triangulation D whose edges are all Delaunay.

I shall show that no other edge is Delaunay. Consider any edge $v_1v_2 \notin D$, with $v_1, v_2 \in V$. Because D is a triangulation, v_1v_2 must cross some edge $w_1w_2 \in D$. Because w_1w_2 is in D, it is Delaunay, and there is a circle C passing through w_1 and w_2 that encloses neither v_1 nor v_2. Because no four vertices are cocircular, at least one of v_1 and v_2 lies strictly outside C. It follows that no empty circle passes through v_1 and v_2, hence v_1v_2 is not Delaunay (see Figure 2.10).

Therefore, D is the Delaunay triangulation of V. ■

What if V contains cocircular vertices? In this circumstance, the Delaunay triangulation may have crossing edges, as illustrated in Figure 2.11(a). Because an arbitrarily small perturbation of the input vertices can change the topology of the triangulation, V and its Delaunay triangulation are said to be *degenerate*.

The definition of "Delaunay triangulation" is usually modified to prevent edges from crossing. Occasionally, one sees in the literature a definition wherein all such crossing edges are omitted; polygons with more than three sides may appear in the Delaunay diagram, as Figure 2.11(b) shows. (The usefulness of this definition follows in part because the graph thus defined is the geometric dual of the well-known Voronoi diagram.) For most applications, however, it is desirable to have a true triangulation, and some of the Delaunay edges (and thus, some of the Delaunay triangles) are omitted to achieve this, as in Figure 2.11(c). In this case, the Delaunay triangulation is no longer unique. By Lemma 4, the flip algorithm will find one of

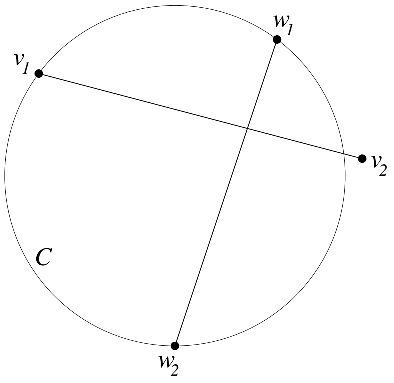


Figure 2.10: If no four vertices are cocircular, two crossing edges cannot both be Delaunay.

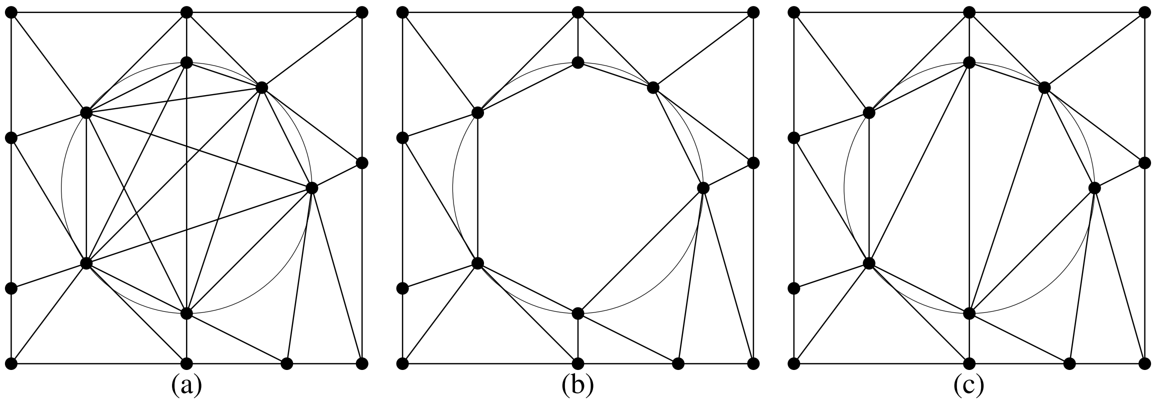


Figure 2.11: Three ways to define the Delaunay diagram in the presence of cocircular vertices. (a) Include all Delaunay edges, even if they cross. (b) Exclude all crossing Delaunay edges. (c) Choose a subset of Delaunay edges that forms a triangulation.

the Delaunay triangulations; the choice of omitted Delaunay edges depends upon the starting triangulation. Because numerical methods like the finite element method generally require a true triangulation, I will use this latter definition of "Delaunay triangulation" throughout the rest of these notes.

Say that an edge or triangle is *strongly Delaunay* if it has a circumcircle such that no vertex lies inside *or on* the circumcircle, except the vertices of the edge or triangle itself. A strongly Delaunay edge or triangle is guaranteed to appear in any Delaunay triangulation of the vertex set, regardless of which definition of Delaunay triangulation is used.

Delaunay triangulations are valuable in part because they have the following optimality properties.

Theorem 6 *Among all triangulations of a vertex set, the Delaunay triangulation maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle, where the* min-containment circle *of a triangle is the smallest circle that contains it (and is not necessarily its circumcircle).*

Proof: It can be shown that each of these properties is locally improved when an edge that is not locally Delaunay is flipped. The optimal triangulation cannot be improved, and thus has only locally Delaunay edges. By Theorem 5, a triangulation whose edges are all locally Delaunay is the Delaunay triangulation. ■

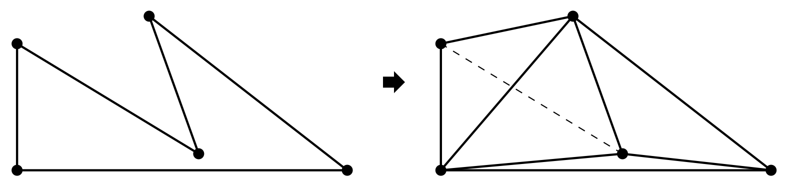


Figure 2.12: The Delaunay triangulation of a set of vertices does not usually solve the mesh generation problem, because it may contain poor quality triangles and omit some domain boundaries.

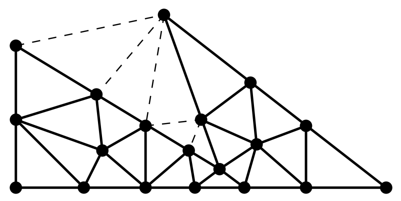


Figure 2.13: By inserting additional vertices into the triangulation, boundaries can be recovered and poor quality elements can be eliminated.

The property of max-min angle optimality was first noted by Lawson [43], and helps to account for the popularity of Delaunay triangulations in mesh generation. Unfortunately, neither this property nor the min-max circumcircle property generalizes to Delaunay triangulations in dimensions higher than two. The property of minimizing the largest min-containment circle was first noted by D'Azevedo and Simpson [19], and has been shown by Rajan [55] to hold for higher-dimensional Delaunay triangulations.

2.1.2 Planar Straight Line Graphs and Constrained Delaunay Triangulations

Given that the Delaunay triangulation of a set of vertices maximizes the minimum angle (in two dimensions), why isn't the problem of mesh generation solved? There are two reasons, both illustrated in Figure 2.12, which depicts an input object and a Delaunay triangulation of the object's vertices. The first reason is that Delaunay triangulations are oblivious to the boundaries that define an object or problem domain, and these boundaries may or may not appear in a triangulation. The second reason is that maximizing the minimum angle usually isn't good enough; for instance, the bottommost triangle of the triangulation of Figure 2.12 is quite poor.

Both of these problems can be solved by inserting additional vertices into the triangulation, as illustrated in Figure 2.13. Chapters 3 and 4 will discuss this solution in detail. Here, however, I review a different solution to the first problem that requires no additional vertices. Unfortunately, it is only applicable in two dimensions.

The usual input for two-dimensional mesh generation is not merely a set of vertices. Most theoretical treatments of meshing take as their input a *planar straight line graph* (PSLG). A PSLG is a set of vertices and segments that satisfies two constraints. First, for each segment contained in a PSLG, the PSLG must also contain the two vertices that serve as endpoints for that segment. Second, segments are permitted to intersect only at their endpoints. (A set of segments that does not satisfy this condition can be converted into a set of segments that does. Run a segment intersection algorithm [18, 63], then divide each segment into smaller segments at the points where it intersects other segments.)

The *constrained Delaunay triangulation* (CDT) of a PSLG X is similar to the Delaunay triangulation, but every input segment appears as an edge of the triangulation. An edge or triangle is said to be *constrained*

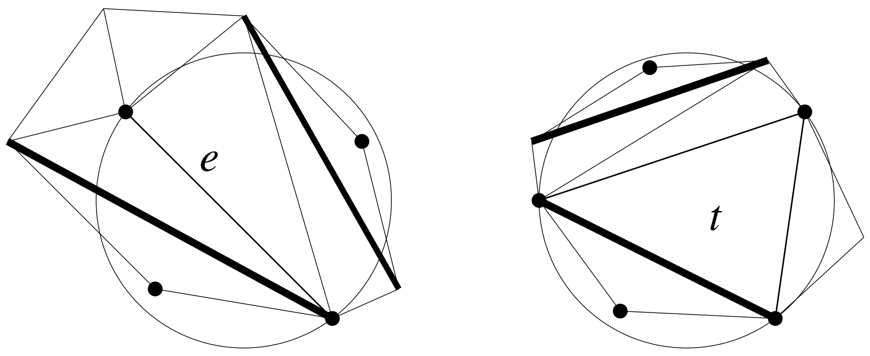


Figure 2.14: The edge e and triangle t are each constrained Delaunay. Bold lines represent segments.

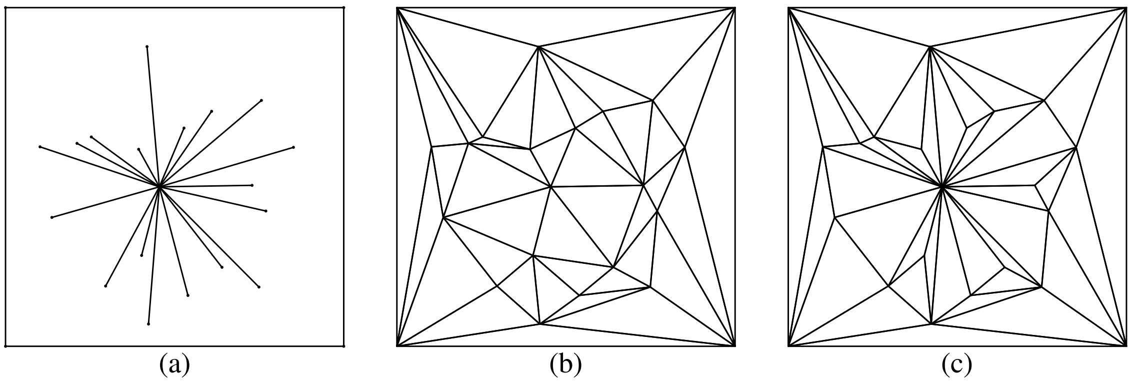


Figure 2.15: (a) A planar straight line graph. (b) Delaunay triangulation of the vertices of the PSLG. (c) Constrained Delaunay triangulation of the PSLG.

Delaunay if it satisfies the following two conditions. First, its interior does not intersect an input segment (unless it *is* an input segment). Second, it has a circumcircle that encloses no vertex of X that is *visible* from the interior of the edge or triangle. Here, visibility between two points is deemed to be occluded if a segment of X lies between them. Two points can *see* each other if the line segment connecting them intersects no segment of X, except perhaps at the two points.

Figure 2.14 demonstrates examples of a constrained Delaunay edge e and a constrained Delaunay triangle t. Input segments appear as bold lines. Although e has no empty circumcircle, the depicted circumcircle of e encloses no vertex that is visible from the interior of e. There are two vertices inside the circle, but both are hidden behind segments. Hence, e is constrained Delaunay. Similarly, the circumcircle of t encloses two vertices, but both are hidden from the interior of t by segments, so t is constrained Delaunay.

If no four vertices of X lie on a common circle, the CDT of X is the set of constrained Delaunay triangles of X. This CDT includes all the segments in X, and all the constrained Delaunay edges of X. If X has cocircularities, then some constrained Delaunay triangles and edges may need to be omitted to obtain a true triangulation, as with ordinary Delaunay triangulations.

Figure 2.15 illustrates a PSLG, a Delaunay triangulation of its vertices, and a constrained Delaunay triangulation of the PSLG. Some of the edges of the CDT are constrained Delaunay but not Delaunay. Don't be fooled by the name: constrained Delaunay triangulations are not necessarily Delaunay triangulations.

Like Delaunay triangulations, constrained Delaunay triangulations can be constructed by the flip algorithm. However, the flip algorithm must begin with a triangulation whose edges include all the segments

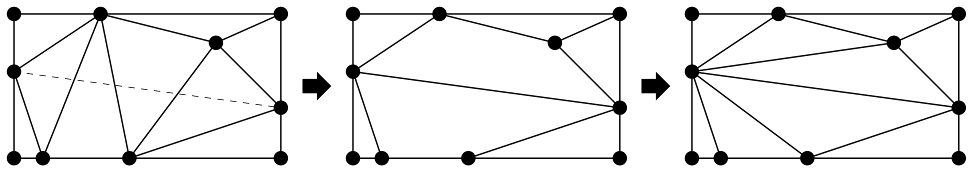


Figure 2.16: Inserting a segment into a triangulation.

of the input PSLG. To show that such a triangulation always exists (assuming the input vertices are not all collinear), begin with an arbitrary triangulation of the vertices of the PSLG. Examine each input segment in turn to see if it is missing from the triangulation. Each missing segment is forced into the triangulation by deleting all the edges it crosses, inserting the new segment, and retriangulating the two resulting polygons (one on each side of the segment), as illustrated in Figure 2.16. (For a proof that any polygon can be triangulated, see Bern and Eppstein [6].)

Once a triangulation containing all the input segments is found, the flip algorithm may be applied, with the provision that segments cannot be flipped. The following results may be proven analogously to the proofs in Section 2.1.1. The only changes that need be made in the proofs is to ignore the presence of vertices that are hidden behind input segments.

Lemma 7 *Let T be a triangulation of a PSLG. If every triangle of T is constrained Delaunay, then every edge of T is either constrained Delaunay or an input segment, and vice versa.* ■

Lemma 8 *Let T be a triangulation whose nonconstraining edges (those that do not represent input segments) are all locally Delaunay. Then every nonconstraining edge of T is constrained Delaunay.* ■

Lemma 9 *Given a triangulation of n vertices in which all input segments are represented as edges, the flip algorithm terminates after $\mathcal{O}(n^2)$ edge flips, yielding a triangulation whose nonconstraining edges are all constrained Delaunay.* ■

Theorem 10 *Let X be a PSLG containing three or more vertices that are not all collinear. If no four vertices of X are cocircular, the constrained Delaunay triangulation of X is a triangulation, and is produced by the flip algorithm.* ■

Theorem 11 *Among all constrained triangulations of a PSLG, the constrained Delaunay triangulation maximizes the minimum angle, minimizes the largest circumcircle, and minimizes the largest min-containment circle.* ■

2.1.3 Higher-Dimensional Delaunay Triangulations

The Delaunay triangulation of a vertex set V in E^d, for $d \geq 2$, is a straightforward generalization of the two-dimensional Delaunay triangulation. A *triangulation* of V is a set T of d-simplices (for example, tetrahedra in three dimensions) whose vertices collectively are V, whose interiors do not intersect each other, and whose union is the convex hull of V, if every d-simplex intersects V only at its vertices.

Let s be a k-simplex (for any k) whose vertices are in V. Let S be a (full-dimensional) sphere in E^d; S is a *circumsphere* of s if S passes through all the vertices of s. If $k = d$, then s has a unique

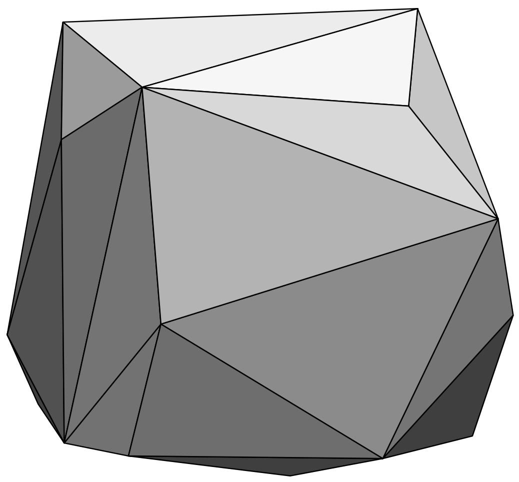


Figure 2.17: A Delaunay tetrahedralization.

circumsphere; otherwise, s has infinitely many circumspheres. The simplex s is *Delaunay* if there exists an empty circumsphere of s. The simplex s is *strongly Delaunay* if there exists a circumsphere S of s such that no vertex of V lies inside *or on* S, except the vertices of s. Every 0-simplex (vertex) of V is trivially strongly Delaunay.

If no $d + 2$ vertices are cospherical, the Delaunay triangulation is a triangulation and is unique. Figure 2.17 illustrates a Delaunay tetrahedralization. If cospherical vertices are present, it is customary to define the Delaunay triangulation to be a true triangulation. As with degenerate Delaunay triangulations, a subset of the Delaunay d-simplices (as well as edges, 2-simplices, and so forth) may have to be omitted to achieve this, thus sacrificing uniqueness.

I have mentioned that the max-min angle optimality of the two-dimensional Delaunay triangulation, first shown by Lawson [43], does not generalize to higher dimensions. Figure 2.18 illustrates this unfortunate fact with a three-dimensional counterexample. A hexahedron is illustrated at top. Its Delaunay tetrahedralization, which appears at lower left, includes a thin tetrahedron known as a *sliver* or *kite*, which may have dihedral angles arbitrarily close to 0° and 180°. A better quality tetrahedralization of the hexahedron appears at lower right.

Edge flips, discussed in Section 2.1.1, have higher-dimensional analogues. The three-dimensional analogue toggles between these two tetrahedralizations. There are two basic flips in three dimensions, both illustrated in Figure 2.19. A *2-3 flip* transforms the two-tetrahedron configuration on the left into the three-tetrahedron configuration on the right, eliminating the face $\triangle cde$ and inserting the edge ab and three triangular faces connecting ab to c, d, and e. A *3-2 flip* is the reverse transformation, which deletes the edge ab and inserts the face $\triangle cde$.

Recall from Figure 2.6 that a two-dimensional edge flip is not possible if the containing quadrilateral of an edge is not strictly convex. Similarly, a three-dimensional flip is not possible if the containing hexahedron of the edge or face being considered for elimination is not strictly convex. A 2-3 flip is prevented if the line ab does not pass through the interior of the face $\triangle cde$. A 3-2 flip is prevented if $\triangle cde$ does not pass through the interior of the edge ab (Figure 2.19, bottom).

Although the idea of a flip generalizes to three or more dimensions, the flip algorithm does not. Joe [38] gives an example that demonstrates that if the flip algorithm starts from an arbitrary tetrahedralization, it may become stuck in a local optimum, producing a tetrahedralization that is not Delaunay. The tetrahedralization

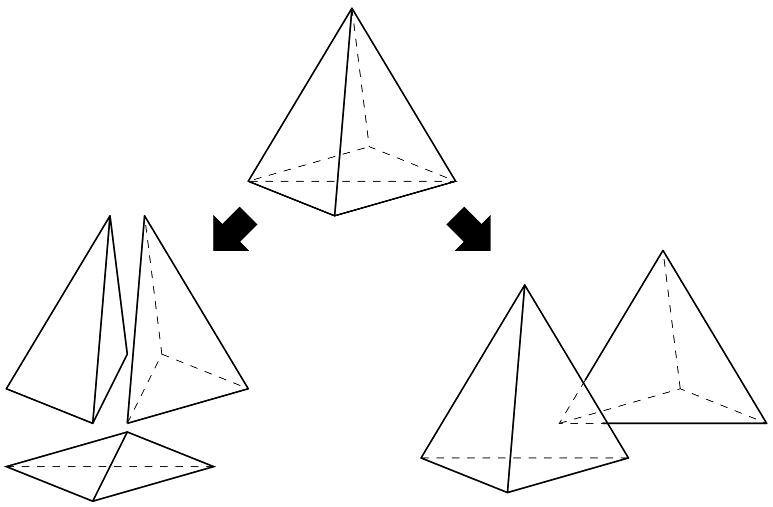


Figure 2.18: This hexahedron can be tetrahedralized in two ways. The Delaunay tetrahedralization (left) includes an arbitrarily thin tetrahedron known as a *sliver*, which could compromise the accuracy of a finite element simulation. The non-Delaunay tetrahedralization on the right consists of two nicely shaped elements.

may contain a locally non-Delaunay face that cannot be flipped because its containing hexahedron is not convex, or a locally non-Delaunay edge that cannot be flipped because it is contained in more than three tetrahedra.

It is not known whether an arbitrary tetrahedralization can always be transformed into another arbitrary tetrahedralization of the same vertex set through a sequence of flips. Nevertheless, Delaunay tetrahedralizations can be constructed by an incremental insertion algorithm based on flips.

Any algorithm based on flips in dimensions greater than two must give some consideration to the possibility of coplanar vertices. For instance, a three-dimensional flip-based incremental Delaunay tetrahedralization algorithm must be able to perform the *4-4 flip* demonstrated in Figure 2.20. This transformation is handy when the vertices c, d, e, and f are coplanar. The 4-4 flip is directly analogous to the two-dimensional edge flip, wherein the edge df is replaced by the edge ce. 4-4 flips are used often in cases where c, d, e, and f lie on an interior boundary facet of an object being meshed. Be aware of a special case called a *2-2 flip*, where c, d, e, and f lie on an exterior boundary, and the top two tetrahedra, as well as the vertex a, are not present.

Section 2.1.1 proved that the two-dimensional Delaunay triangulation is a triangulation by using the flip algorithm, but the flip algorithm does not work in higher dimensions. Fortune [25] describes a different proof of the existence of a Delaunay triangulation, which applies in any dimension. This proof also relies on an algorithm for constructing the Delaunay triangulation, called *gift-wrapping*. Throughout the proof, the terms “simplex” and “convex hull” refer to closed, convex sets of points; hence, they include all the points on their boundaries and in their interiors.

Let V be a set of vertices in E^d, and suppose that some subset of $d+1$ vertices in V is affinely independent. The set of Delaunay d-simplices defined on V forms a triangulation of V if V contains no $d+2$ vertices that lie on a common sphere. This fact may be proven in two steps: first, by showing that every point in the convex hull of V is contained in some Delaunay d-simplex of V; second, by showing that any two Delaunay d-simplices have disjoint interiors, and intersect only at a shared face (if at all). Hence, the union of the Delaunay d-simplices is the convex hull of V, and the Delaunay d-simplices (and their faces) form a

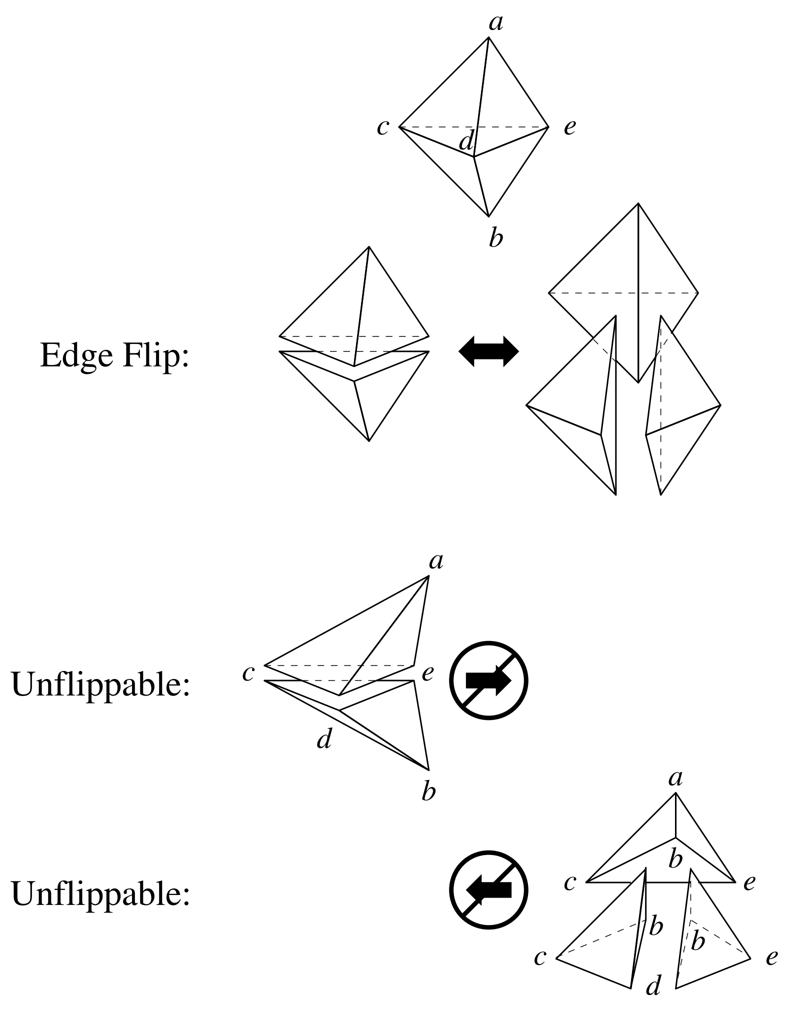


Figure 2.19: Flips in three dimensions. The two-tetrahedron configuration (left) can be transformed into the three-tetrahedron configuration (right) only if the line ab passes through the interior of the triangular face $\triangle cde$. The three-tetrahedron configuration can be transformed into the two-tetrahedron configuration only if the plane containing $\triangle cde$ passes through the interior of the edge ab.

simplicial complex. Only the second step requires the assumption that no $d + 2$ vertices are cospherical.

The first step is based on a straightforward procedure for "growing" a simplex, vertex by vertex. The procedure is presumed to be in possession of a Delaunay k-simplex s_k, whose vertices are $v_0, \ldots, v_k$, and produces a Delaunay $(k + 1)$-simplex s_{k+1} that possesses a face s_k and an additional vertex v_{k+1}.

Because s_k is Delaunay, it has an empty circumsphere S. The growth procedure expands S like a bubble, so that its center O_S moves in a direction orthogonal to the k-dimensional hyperplane h_k that contains s_k, as illustrated in Figure 2.21. Because O_S is moving in a direction orthogonal to h_k, O_S remains equidistant from the vertices of s_k, and hence it is always possible to choose a radius for S that ensures that S continues to pass through all the vertices of s_k. The expansion ends when S contacts an additional vertex v_{k+1}. The simplex s_{k+1} defined by v_{k+1} and the vertices of s_k is Delaunay, because S is still empty.

The motion of the sphere center O_S is governed by a *direction vector* c_k, which is constrained to be orthogonal to h_k, but may otherwise be specified freely. In the limit as O_S moves infinitely far away, S will approach the $(d - 1)$-dimensional hyperplane that contains s_k and h_k and is orthogonal to c_k. The region

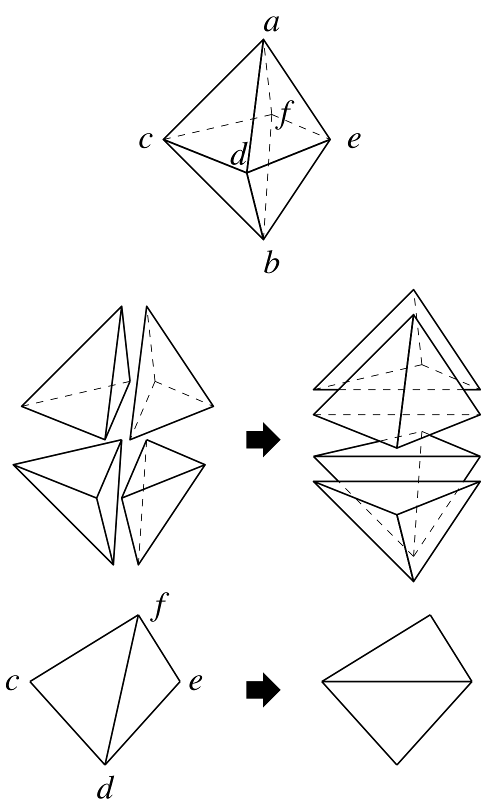


Figure 2.20: A 4-4 flip. The vertices c, d, e, and f are coplanar. This transformation is analogous to the two-dimensional edge flip (bottom).

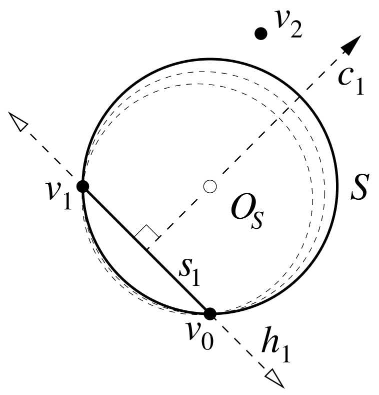


Figure 2.21: An empty sphere S that circumscribes s_1, expanding in search of another vertex v_2.

enclosed by S will approach an open half-space bounded by this hyperplane. A point is said to be *above* h_k if it lies in this open half-space. Any vertex outside S that S comes in contact with while expanding must lie above h_k; the portion of S below h_k is shrinking toward the inside of S (as Figure 2.21 shows).

A special case occurs if the sphere S already contacts a vertex u not in s_k before S begins expanding. If u is affinely independent of s_k, then it is immediately accepted as v_{k+1}. Otherwise, u is ignored. (The affinely dependent case can only occur for $k \geq 2$; for instance, when a triangle grows to become a tetrahedron, as illustrated in Figure 2.22, there may be ignored vertices on the triangle's planar circumcircle.)

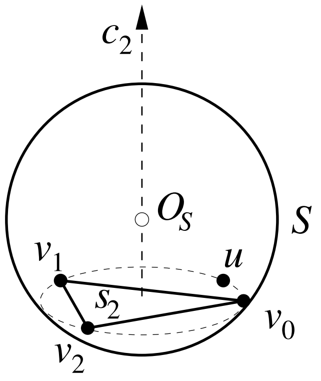


Figure 2.22: The point u lies on S, but is ignored because it is not affinely independent from $\{v_0, v_1, v_2\}$.

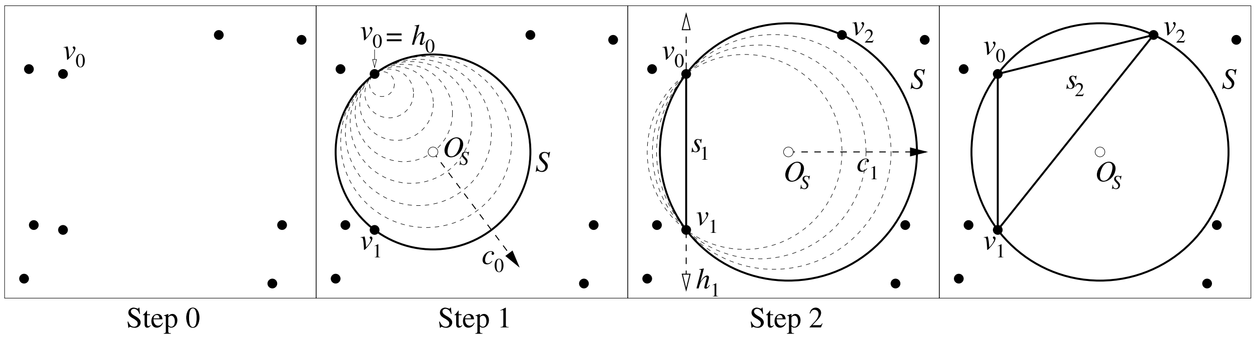


Figure 2.23: Growing a Delaunay simplex. If $d > 2$, c_2 will point directly out of the page.

If the procedure is to succeed, c_k must satisfy the following *visibility hypothesis*: the open half-space above h_k must contain a vertex of V.

This growth procedure is the basis for the following proof.

Theorem 12 *Let p be any point in the convex hull of V. Some Delaunay d-simplex of V contains p.*

Proof: The proof is based on a two-stage constructive procedure. The first stage finds an arbitrary Delaunay d-simplex, and involves $d+1$ steps, numbered zero through d. For step zero, define a sphere S whose center is an arbitrary vertex v_0 in V, and whose radius is zero. The vertex v_0 seeds the starting simplex $s_0 = \{v_0\}$.

During the remaining steps, illustrated in Figure 2.23, S expands according to the growth procedure described above. At step $k+1$, the direction vector c_k is chosen so that some vertex of V lies above h_k, and thus the visibility hypothesis is established. Such a choice of c_k is always possible because V contains $d+1$ affinely independent vertices, and thus some vertex is not in h_k. If there are several vertices above h_k, let v_{k+1} be the first such vertex contacted by the expanding sphere, so that no vertex is inside the sphere. The new simplex s_{k+1} is Delaunay.

After step d, s_d is a Delaunay d-simplex. If s_d contains p, the procedure is finished; otherwise, the second stage begins.

Consider the directed line segment qp, where q is an arbitrary point in the interior of s_d chosen so that qp does not intersect any simplices (defined on the vertices of V) of dimension $d-2$ or smaller, except

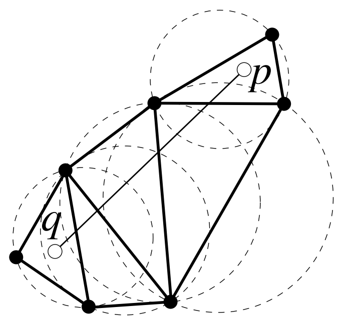


Figure 2.24: Walking to the d-simplex that contains p.

possibly at the point p. (Such a choice of q is always possible; the set of points in s_d that *don't* satisfy this condition has measure zero.) The second stage "walks" along the segment *qp*, traversing a sequence of Delaunay d-simplices that intersect *qp* (illustrated in Figure 2.24), until it finds one that contains p.

Wherever *qp* exits a d-simplex, the next d-simplex may be constructed as follows. Let s be the face through which *qp* exits the current d-simplex. The $(d-1)$-simplex s is Delaunay, and forms the base from which another Delaunay d-simplex is grown, with the direction vector chosen to be orthogonal to s and directed out of the previous d-simplex. Is the visibility hypothesis satisfied? Clearly, the point p is above s. Some vertex of V must lie above s, because if none did, the convex hull of V would not intersect the half-space above s and thus would not contain p. Hence, a new Delaunay d-simplex may be formed as in the first stage.

Because the new d-simplex is above s, it is distinct from the previous d-simplex. Because each successive d-simplex intersects a subsegment of *qp* having nonzero length, each successive $(d-1)$-face intersects *qp* closer to p, and thus no simplex is visited twice. Since only a finite number of simplices can be defined (over a finite set of vertices), the procedure must terminate; and since the procedure will not terminate unless the current d-simplex contains p, there exists a Delaunay d-simplex that contains p. ■

This procedure is the basis for a well-known algorithm, called *gift-wrapping*, *graph traversal*, *pivoting*, or *incremental search*, for constructing Delaunay triangulations [23]. Gift-wrapping begins by finding a single Delaunay d-simplex, which is used as a seed upon which the remaining Delaunay d-simplices crystallize one by one. Each $(d-1)$-face of a Delaunay d-simplex is used as a base from which to search for the vertex that serves as the apex of an adjacent d-simplex.

As every point in the convex hull of V is contained in a Delaunay d-simplex, it remains only to show that the set of Delaunay d-simplices do not occupy common volume or fail to intersect neatly. It is only here that the assumption that no $d+2$ vertices are cospherical is needed.

Theorem 13 *Suppose that no $d+2$ vertices of V lie on a common sphere. Let s and t be two distinct Delaunay d-simplices of V. Then s and t have disjoint interiors, and if they intersect at all, their intersection is a face of both s and t.*

Proof: Let S_s and S_t be the circumspheres of s and t. S_s and S_t cannot be identical, because s and t each have at least one vertex not shared by the other; if S_s and S_t were the same, at least $d+2$ vertices would lie on S_s. S_s cannot enclose S_t, nor the converse, because S_s and S_t enclose no vertices. Hence, either S_s and S_t are entirely disjoint (and thus so are s and t), or their intersection is a lower-dimensional sphere or point

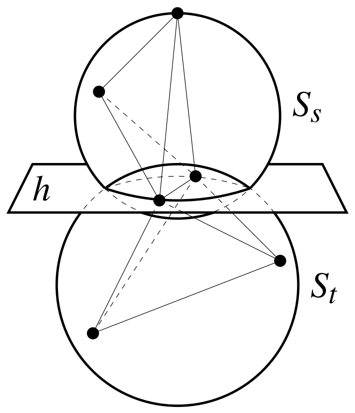


Figure 2.25: The Delaunay d-simplices s and t intersect at a lower-dimensional shared face (in this illustration, an edge).

and is contained in a $(d-1)$-dimensional hyperplane h, as Figure 2.25 illustrates. (If S_s and S_t intersect at a single point, h is chosen to be tangent to both spheres.)

Because every vertex of s lies on S_s, and no vertex is inside S_t, every vertex of s must lie on or above h. Similarly, every vertex of t must lie on or below h. Hence, the interiors of s and t do not intersect. Furthermore, as no $d+2$ vertices lie on a common sphere, every vertex of s or t that lies in h—and hence in the intersection of S_s and S_t—must be a vertex of both s and t. Hence, s and t intersect in a shared face. ■

2.1.4 Piecewise Linear Complexes and Higher-Dimensional Constrained Delaunay Triangulations

Although Delaunay triangulations in three dimensions or higher are invaluable for mesh generation, they are in several ways more limited than their two-dimensional brethren. One difficulty is that, whereas every polygon can be triangulated (without creating additional vertices), there are polyhedra that cannot be tetrahedralized. Schönhardt [62] furnishes an example depicted in Figure 2.26 (right). The easiest way to envision this polyhedron is to begin with a triangular prism. Imagine grasping the prism so that one of its two triangular faces cannot move, while the opposite triangular face is rotated slightly about its center without moving out of its plane. As a result, each of the three square faces is broken along a diagonal *reflex edge* (an edge at which the polyhedron is locally nonconvex) into two triangular faces. After this transformation, the upper left corner and lower right corner of each (former) square face are separated by a reflex edge and are no longer visible to each other within the polyhedron. Any four vertices of the polyhedron include two separated by a reflex edge; thus, any tetrahedron whose vertices are vertices of the polyhedron will not lie entirely within the polyhedron. Therefore, Schönhardt's polyhedron cannot be tetrahedralized without additional vertices. (However, it can be tetrahedralized with the addition of one vertex at its center.)

Hence, constrained tetrahedralizations do not always exist. What if we forbid constraining facets, but permit constraining segments? Figure 2.27 illustrates a set of vertices and segments for which a constrained tetrahedralization does not exist. (The convex hull, a cube, is illustrated for clarity, but no constraining facets are present in the input.) Three orthogonal segments pass by each other near the center of the cube, but do not intersect. If any one of these segments is omitted, a tetrahedralization is possible. This example shows that, unlike in the two-dimensional case, it is not always possible to insert a new segment into a constrained tetrahedralization.

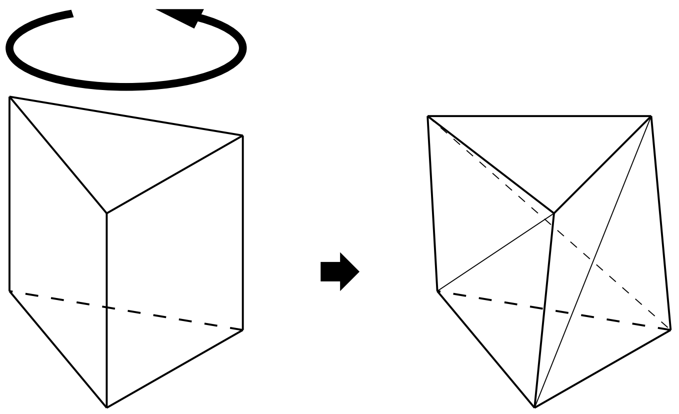


Figure 2.26: Schönhardt's untetrahedralizable polyhedron (right) is formed by rotating one end of a triangular prism (left), thereby creating three diagonal reflex edges.

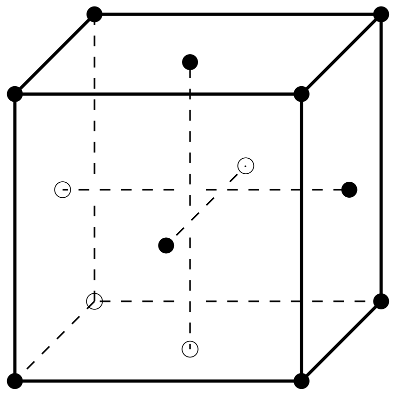


Figure 2.27: A set of vertices and segments for which there is no constrained tetrahedralization.

Ruppert and Seidel [60] add to the difficulty by proving that it is NP-hard to determine whether a polyhedron is tetrahedralizable. The mesh generation algorithm discussed in Chapter 4 recovers boundaries by strategically inserting additional vertices. Unfortunately, Ruppert and Seidel also show that the problem of determining whether a polyhedron can be tetrahedralized with only k additional vertices is NP-hard. On the bright side, Bern and Eppstein [6] show that any polyhedron can be tetrahedralized with the insertion of $\mathcal{O}(n^2)$ additional vertices, so the demands of tetrahedralization are not limitless.

Nevertheless, there is an easily tested condition that guarantees that a CDT exists, for a conservative definition of CDT. Furthermore, there are useful consequences to this guarantee, because the condition is naturally enforced by the meshing algorithm of Chapter 4.

Before exploring higher-dimensional CDTs, we must ask: what is the input to a higher-dimensional mesh generation algorithm? At the least, we would like to tetrahedralize any polyhedron, but polyhedra are not general enough for many applications. Ideally, we would like to be able to specify regions of space bounded by arbitrary curved surfaces, but mesh generation theory (and practice) is not yet up to the challenge. To strike a balance between mathematical abstraction, tractability, and flexibility, I use a generalization of a planar straight line graph called a *piecewise linear complex* (PLC), following Miller, Talmor, Teng, Walkington, and Wang [49].

Before considering PLCs in their full generality, consider the special case in which a PLC X is a set of vertices and *constraining simplices* in E^d. Each constraining simplex is a k-dimensional simplex (henceforth, k-simplex) having $k + 1$ affinely independent vertices, where $1 \leq k \leq d - 1$. Each constraining simplex must appear as a face of the final triangulation. (Observe that a PSLG is a two-dimensional PLC.)

PLCs have restrictions similar to those of PSLGs. If X contains a simplex s, then X must contain every lower-dimensional face of s, including its vertices. Any two simplices of a PLC, if one is not a face of the other, may intersect only at a shared lower-dimensional face or vertex.

Say that the visibility between two points p and q in E^d is *occluded* if there is a $(d-1)$-simplex s of X such that p and q lie on opposite sides of the hyperplane that contains s, and the line segment pq intersects s (either in the boundary or in the relative interior of s). If either p or q lies in the hyperplane containing s, then s does not occlude the visibility between them. Simplices in X of dimension less than $d - 1$ do not occlude visibility. The points p and q can *see* each other if there is no occluding $(d-1)$-simplex of X.

Let s be a simplex (of any dimension) whose vertices are in X (but s is not necessarily a constraining simplex of X). The simplex s is *constrained Delaunay* if no constraining simplex of X intersects the interior of s unless it contains s in its entirety, and there is a circumsphere S of s such that no vertex of X inside S is visible from any point in the relative interior of s.

A PLC X is said to be *ridge-protected* if each constraining simplex in X of dimension $d - 2$ or less is strongly Delaunay.

One can prove that if X is ridge-protected, and if no $d + 2$ vertices of X lie on a common sphere, then the constrained Delaunay d-simplices defined on the vertices of X collectively form a triangulation of X [66]. This triangulation is called the *constrained Delaunay triangulation* of X. The condition that X be ridge-protected holds trivially in two dimensions (hence the success of two-dimensional CDTs), but not in three or more. In three dimensions, a CDT exists if all the segments of X are strongly Delaunay.

Note that it is not sufficient for each lower-dimensional constraining simplex to be Delaunay; if Schönhardt's polyhedron is specified so that all six of its vertices lie on a common sphere, then all of its edges (and its faces as well) are Delaunay, but it still does not have a tetrahedralization. It is not possible to place the vertices of Schönhardt's polyhedron so that all three of its reflex edges are strongly Delaunay (though any two may be).

If some $d + 2$ vertices of X are cospherical, the existence of a CDT can still be established (through perturbation arguments), but the CDT is not necessarily unique. There may be constrained Delaunay d-simplices whose interiors are not disjoint. Some of these simplices must be omitted to yield a proper triangulation.

It is appropriate, now, to consider a more general definition of PLC. To illustrate the limitation of the definition given heretofore, consider finding a tetrahedralization of a three-dimensional cube. If each square face is represented by two triangular constraining simplices, then the definition given above requires that the diagonal edge in each face be a constraining edge of X. However, the existence of a CDT can be proven even if these edges are not strongly Delaunay. Hence, a PLC may contain constraining *facets*, which are more general than constraining simplices. Each facet is a polytope of any dimension from one to $d - 1$, possibly with holes and lower-dimensional facets inside it. A k-simplex s is said to be a *constraining simplex in* X if s appears in the k-dimensional CDT of some k-facet in X. If X is ridge-protected, then the CDT of a facet is just its Delaunay triangulation, because the boundary simplices of each facet are strongly Delaunay.

Figure 2.28 illustrates a three-dimensional PLC. As the figure illustrates, a facet may have any number of sides, may be nonconvex, and may have holes, slits, or vertices inside it. In the three-dimensional case, every facet must be planar. Figure 2.29 illustrates the CDT of the region bounded by the PLC.

A three-dimensional PLC X is required to have the following properties.

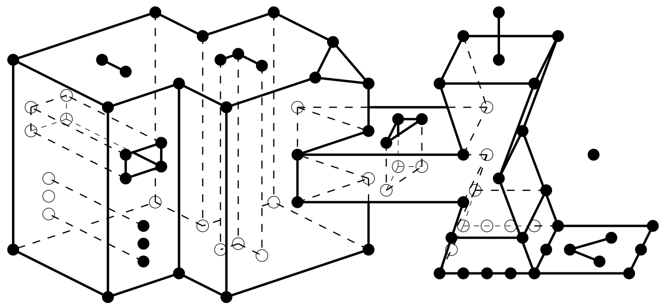


Figure 2.28: Any facet of a PLC may contain holes, slits, and vertices; these may support intersections with other facets, or simply enforce the presence of specific faces so a user of the finite element method may apply boundary conditions there.

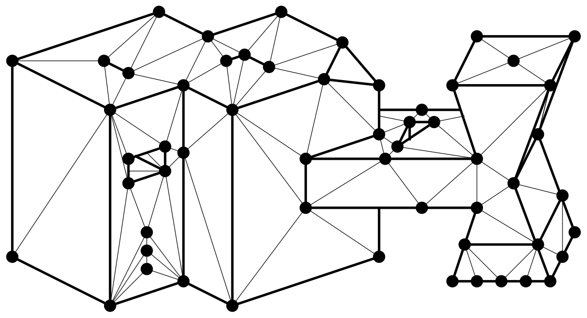


Figure 2.29: A constrained Delaunay tetrahedralization.

- X contains both endpoints of each segment of X.
- For any facet in X, every edge and vertex of the facet must appear as a segment or vertex of X. Hence, all facets are *segment-bounded*: the boundary of each facet is the union of selected segments of X.
- X is (loosely speaking) closed under intersection. Hence, if two facets of X intersect at a line segment, that line segment must be represented by a segment of X. If the intersection of two facets is several line segments, each of those line segments must be in X. If a segment or facet of X intersects another segment or facet of X at one or more individual points, those points must be vertices in X.
- If a segment of X intersects a facet of X at an infinite number of points, then the segment must be entirely contained in the facet. This rule ensures that facets "line up" with their boundaries. A facet cannot be bounded by a segment that extends beyond the boundary of the facet.

See Miller et al. [49] for an equivalent list of conditions for the arbitrary-dimensional case.

The advantage of employing whole facets, rather than individual constraining simplices, is that only the lower-dimensional boundary facets of a $(d-1)$-facet need be included in the PLC (including interior boundaries, at holes and slits). The lower-dimensional faces that are introduced in the relative interior of a $(d-1)$-facet when it is triangulated do not need to be strongly Delaunay for a CDT to exist.

This advantage does not extend to lower-dimensional facets, because a ridge-protected k-facet, for $k \leq d-2$, must be composed of strongly Delaunay k-simplices, and it is easy to show that any lower-dimensional face of a strongly Delaunay simplex is strongly Delaunay. Hence, in a ridge-protected PLC, all simplicial faces in the Delaunay triangulation of a facet are required to be strongly Delaunay, except for the faces in the triangulation of a $(d-1)$-facet that are not in the facet's boundary.

Testing whether a PLC is ridge-protected is straightforward. Form the Delaunay triangulation of the vertices of the PLC. If a constraining simplex s is missing from the triangulation, then s is not strongly Delaunay. Otherwise, s is Delaunay; testing whether s is strongly Delaunay is a local operation equivalent to determining whether the dual face of s in the corresponding Voronoi diagram is nondegenerate.

Why is it useful to know that ridge-protected PLCs have CDTs? Although a given PLC X may not be ridge-protected, it can be made ridge-protected by splitting simplices that are not strongly Delaunay into smaller simplices, with the insertion of additional vertices. The result is a new ridge-protected PLC Y, which has a CDT. The CDT of Y is not a CDT of X, because it has vertices that X lacks, but it is what I call a *conforming constrained Delaunay triangulation* (CCDT) of X: conforming because of the additional vertices, and constrained because its simplices are constrained Delaunay (rather than Delaunay).

One advantage of a CCDT over an ordinary Delaunay triangulation is that the number of vertices that must be added to recover all the missing segments and facets is generally smaller. Once a PLC X has been augmented to form a ridge-protected PLC Y, the Delaunay triangulation of the vertices of Y contains all the constraining simplices of Y of dimension $d-2$ or smaller (because they are strongly Delaunay), but does not necessarily respect the $(d-1)$-simplices of Y. The present result implies that the CDT of Y may be formed without adding any more vertices to Y (Figure 2.30, bottom). This idea stands in apposition to the most common method of recovering unrepresented facets in three-dimensional Delaunay-based mesh generation algorithms [72, 35, 73, 56], wherein additional vertices are inserted within the facet (for instance, where an edge of the tetrahedralization intersects the missing facet) to help recover the facet (Figure 2.30, top).

The proof that ridge-protected PLCs have CDTs is quite complicated, and is omitted here. A CDT can be constructed by gift-wrapping, as described in Section 2.1.3. The only modification needed to adapt gift-wrapping to CDTs is to ensure that each gift-wrapping step considers only those vertices that are visible from the interior of the simplex being "grown."

2.1.5 Incremental Insertion Algorithms for Constructing Delaunay Triangulations

Three types of algorithms are in common use for constructing Delaunay triangulations. The simplest are incremental insertion algorithms, which have the advantage of generalizing to arbitrary dimensionality, and will be discussed in some depth here. In two dimensions, there are faster algorithms based upon divide-and-conquer and sweepline techniques, which will be discussed here only briefly. Refer to Su and Drysdale [68, 67] for an overview of these and other two-dimensional Delaunay triangulation algorithms.

Incremental insertion algorithms operate by maintaining a Delaunay triangulation, into which vertices are inserted one at a time. The earliest such algorithm, introduced by Lawson [43], is based upon edge flips. An incremental algorithm that does not use edge flips, and has the advantage of generalizing to arbitrary dimensionality, was introduced simultaneously by Bowyer [8] and Watson [70]. These two articles appear side-by-side in a single issue of The Computer Journal[1]. I will examine the Bowyer/Watson algorithm first, and then return to the algorithm of Lawson.

In the Bowyer/Watson algorithm, when a new vertex is inserted, each triangle whose circumcircle encloses the new vertex is no longer Delaunay, and is thus deleted. All other triangles remain Delaunay, and

[1]The two algorithms are similar in all essential details, but Bowyer reports a better asymptotic running time than Watson, which on close inspection turns out to be nothing more than an artifact of his more optimistic assumptions about the speed of point location.

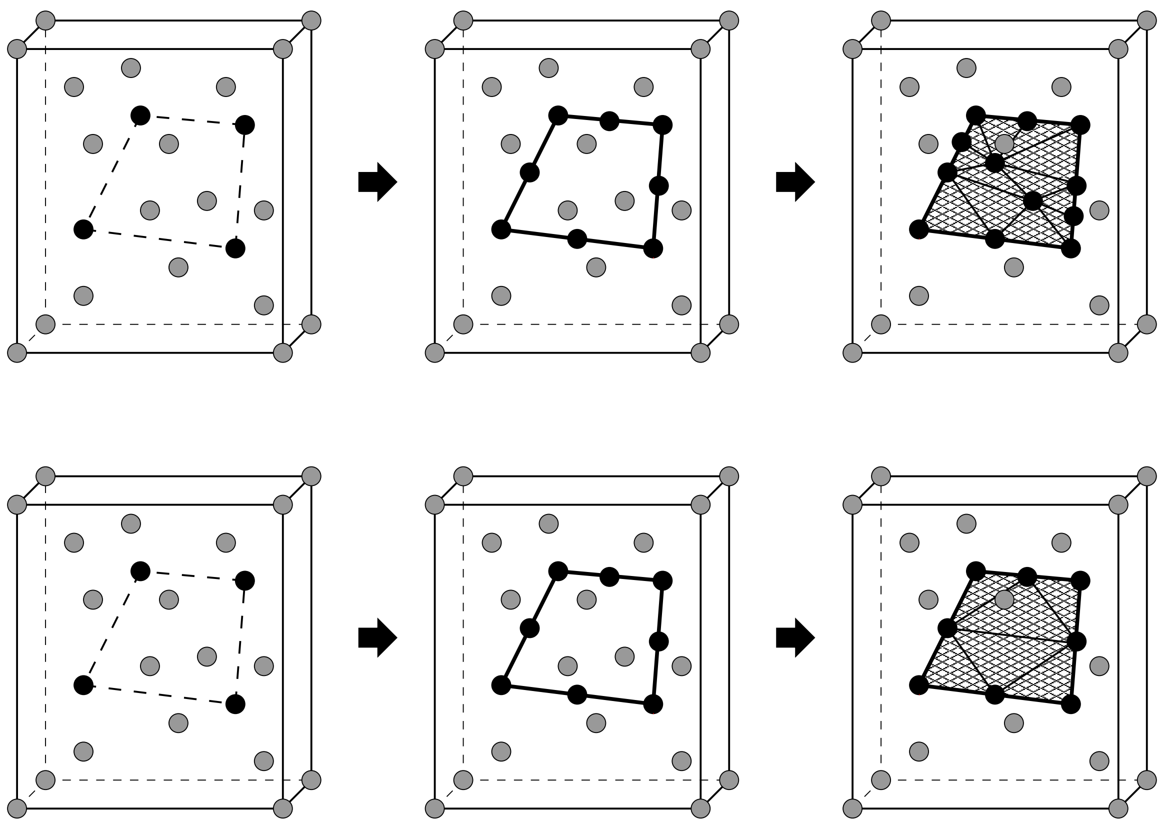


Figure 2.30: Comparison of facet recovery by vertex insertion (top), which is the standard practice in many mesh generation algorithms, with facet recovery by forming a CCDT (bottom). For clarity, vertices inside each box are shown, but tetrahedra are not. The dashed lines at left indicate input segments, bounding an input facet, that are missing from the Delaunay tetrahedralization of the vertices. In both cases, missing segments must be recovered by vertex insertion. However, the CCDT requires no additional vertices to recover missing facets.

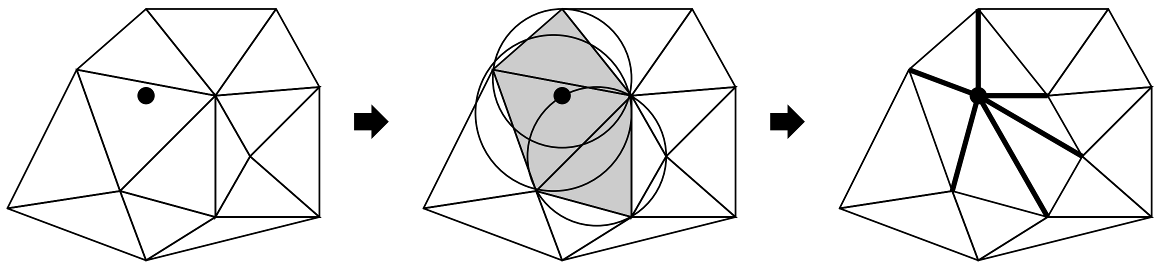


Figure 2.31: The Bowyer/Watson algorithm in two dimensions. When a new vertex is inserted into a triangulation (left), all triangles whose circumcircles enclose the new vertex are deleted (center; deleted triangles are shaded). Edges are created connecting the new vertex to the vertices of the insertion polyhedron (right).

are left undisturbed. The set of deleted triangles collectively form an *insertion polygon*, which is left vacant by the deletion of these triangles, as illustrated in Figure 2.31. The Bowyer/Watson algorithm connects each vertex of the insertion polyhedron to the new vertex with a new edge. These new edges are Delaunay due to the following simple lemma.

Theorem 14 *Let v be a newly inserted vertex, let t be a triangle that is deleted because its circumcircle*

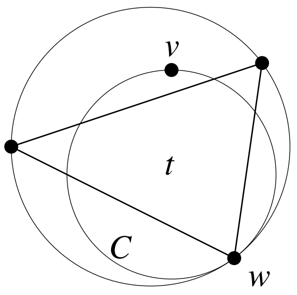


Figure 2.32: If v is a newly inserted vertex, and w is a vertex of a triangle t whose circumcircle encloses only v, then vw is Delaunay.

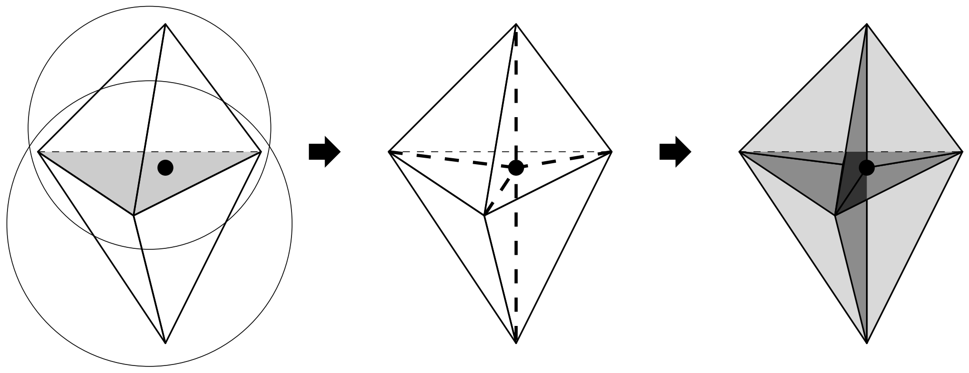


Figure 2.33: The Bowyer/Watson algorithm in three dimensions. At left, a new vertex falls inside the circumspheres of the two tetrahedra illustrated. (These tetrahedra may be surrounded by other tetrahedra, which for clarity are not shown.) These tetrahedra are deleted, along with the face (shaded) between them. At center, the five new Delaunay edges (bold dashed lines). At right, the nine new Delaunay faces (one for each edge of the insertion polyhedron) appear translucent. Six new tetrahedra are formed.

encloses v, and let w be a vertex of t. Then vw is Delaunay.

Proof: See Figure 2.32. The circumcircle of t encloses no vertex but v. Let C be the circle that passes through v and w, and is tangent to the circumcircle of t at w. C is empty, so vw is Delaunay. ■

All new edges created by the insertion of a vertex v have v as an endpoint. This must be true of any correct incremental insertion algorithm, because if an edge (not having v as an endpoint) is not Delaunay before v is inserted, it will not be Delaunay after v is inserted.

The Bowyer/Watson algorithm extends in a straightforward way to three (or more) dimensions. When a new vertex is inserted, every tetrahedron whose circumsphere encloses the new vertex is deleted, as illustrated in Figure 2.33. The new vertex then floats inside a hollow *insertion polyhedron*, which is the union of the deleted tetrahedra. Each vertex of the insertion polyhedron is connected to the new vertex with a new edge. Each edge of the insertion polyhedron is connected to the new vertex with a new triangular face.

In two dimensions, vertices may be incrementally inserted into (and deleted from) a CDT just like an ordinary Delaunay triangulation. This is true in higher dimensions as well, so long as the underlying PLC (which changes incrementally with the triangulation) remains ridge-protected. When a vertex is inserted, the simplices that are no longer constrained Delaunay are deleted. When a vertex is deleted, the simplices that contain the vertex are deleted. In either case, the resulting polytopal hole is retriangulated to complete

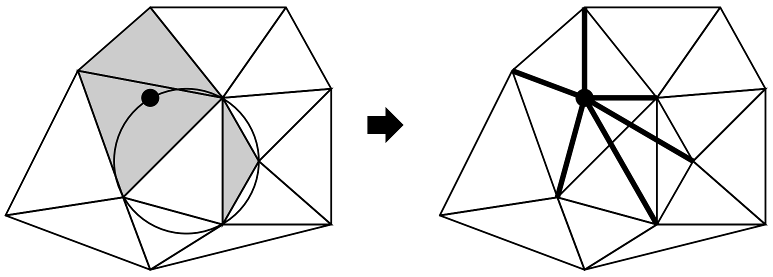


Figure 2.34: The Bowyer/Watson algorithm may behave nonsensically under the influence of floating-point roundoff error.

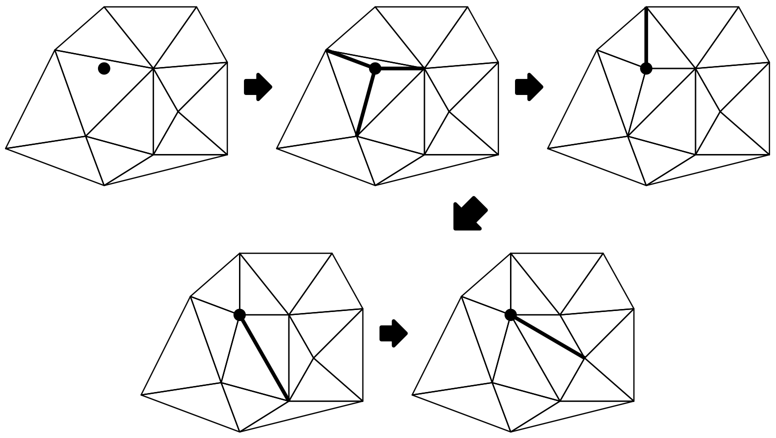


Figure 2.35: Lawson's incremental insertion algorithm uses edge flipping to achieve the same result as the Bowyer/Watson algorithm.

the new CDT. The existence of a CDT of the entire underlying PLC ensures that a CDT of the hole can be produced.

In its simplest form, the Bowyer/Watson algorithm is not robust against floating-point roundoff error. Figure 2.34 illustrates a degenerate example in which two triangles have the same circumcircle, but due to roundoff error only one of them is deleted, and the triangle that remains stands between the new vertex and the other triangle. The insertion polyhedron is not simple, and the triangulation that results after the new triangles are added is nonsensical.

In two dimensions, this problem may be avoided by returning to Lawson's algorithm [43], which is based upon edge flips. Lawson's algorithm is illustrated in Figure 2.35.

When a vertex is inserted, the triangle that contains it is found, and three new edges are inserted to attach the new vertex to the vertices of the containing triangle. (If the new vertex falls upon an edge of the triangulation, that edge is deleted, and four new edges are inserted to attach the new vertex to the vertices of the containing quadrilateral.) Next, a recursive procedure tests whether the new vertex lies within the circumcircles of any neighboring triangles; each affirmative test triggers an edge flip that removes a locally non-Delaunay edge. Each edge flip reveals two additional edges that must be tested. When there are no longer any locally non-Delaunay edges opposite the new vertex, the triangulation is globally Delaunay.

Disregarding roundoff error, Lawson's algorithm achieves exactly the same result as the Bowyer/Watson algorithm. In the presence of roundoff error, Lawson's algorithm avoids the catastrophic circumstance illustrated in Figure 2.34. Lawson's algorithm is not absolutely robust against roundoff error, but failures

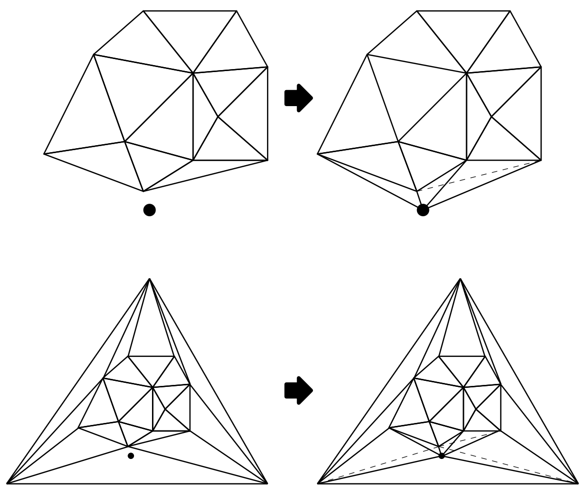


Figure 2.36: Top: A vertex is inserted outside the convex hull of the previously existing vertices. Bottom: A triangular bounding box is used so that all vertex insertions are internal. In practice, the bounding box should be much larger than the one depicted here. Even then, problems may arise.

are rare compared to the most naïve form of the Bowyer/Watson algorithm. However, the Bowyer/Watson algorithm can be implemented to behave equally robustly; for instance, the insertion polygon may be found by depth-first search from the initial triangle.

A better reason for noting Lawson's algorithm is that it is slightly easier to implement, in part because the topological structure maintained by the algorithm remains a triangulation at all times. Guibas and Stolfi [34] provide a particularly elegant implementation. Another reason is because edge flipping can help simplify the process of determining where each new vertex should be inserted, as the next section will show.

Joe [39, 40] and Rajan [55] have generalized Lawson's flip-based algorithm to arbitrary dimensionality. Of course, these algorithms achieve the same outcome as the Bowyer/Watson algorithm, but may present the same advantages for implementation that Lawson's algorithm offers in two dimensions.

This discussion of incremental insertion algorithms in two and three dimensions has assumed that all new vertices fall within the existing triangulation. What if a vertex falls outside the convex hull of the previous vertices? One solution is to handle this circumstance as a special case. New triangles or tetrahedra are created to connect the new vertex to all the edges or faces of the convex hull visible from that vertex. Then, flipping may proceed as usual, as illustrated in Figure 2.36 (top).

An alternative solution that simplifies programming is to bootstrap incremental insertion with a very large triangular or tetrahedral bounding box that contains all the input vertices, as illustrated in Figure 2.36 (bottom). After all vertices have been inserted, the bounding box is removed as a postprocessing step. There is no need to handle vertices outside the triangulation as a special case. This approach is not recommended, because one must be careful to choose the vertices of the bounding box so that they do not cause triangles or tetrahedra to be missing from the final Delaunay triangulation. I am not aware of any straightforward way to determine how far away the bounding box vertices must be to ensure correctness.

2.1.6 The Analysis of Randomized Incremental Insertion in Two Dimensions

If one knows the triangle or tetrahedron in which a new vertex is to be inserted, the amount of work required to insert the vertex is proportional to the number of edge flips, which is typically small. Pathological cases

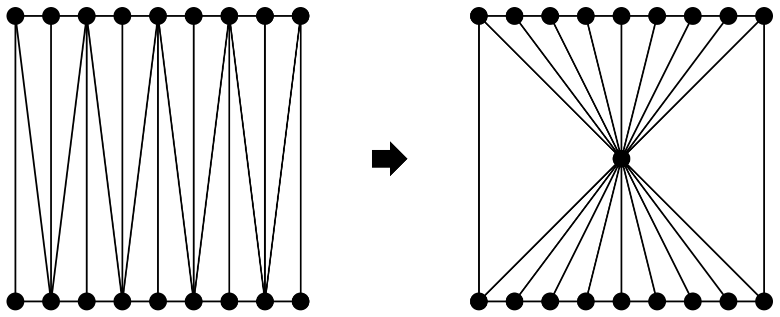


Figure 2.37: A single vertex insertion can cause $\Theta(n)$ edge flips, replacing $\Theta(n)$ triangles with $\Theta(n)$ different triangles.

can occur in which a single vertex insertion causes $\Theta(n)$ flips in two dimensions (Figure 2.37), or $\Theta(n^2)$ in three, where n is the number of vertices in the final triangulation. However, such cases occur only occasionally in mesh generation, and it is common to observe that the average number of flips per insertion is a small constant.

In two dimensions, this observation is given some support by a simple theoretical result. Suppose one wishes to construct, using Lawson's algorithm, the Delaunay triangulation of a set of vertices that is entirely known at the outset. If the input vertices are inserted in a random order, chosen uniformly from all possible permutations, then the expected (average) number of edge flips per vertex insertion is bounded below three. This expectation is the average, over all possible permutations and over all vertex insertions within a permutation, of the number of flips per insertion. There are some pathological permutations that cause the incremental insertion algorithm to perform $\Theta(n^2)$ flips (during the course of n vertex insertions), but these permutations are few enough that the *average* permutation requires fewer than $3n$ flips.

This elegant result seems to originate with Chew [15], albeit in the slightly simpler context of Delaunay triangulations of convex polygons. The result was proven more generally by Guibas, Knuth, and Sharir [33], albeit with a much more complicated proof than Chew's. The result is based on the observation that when a vertex is inserted, each edge flip increases by one the degree of the new vertex. Hence, if the insertion of a vertex causes three edge flips, there will be six edges incident to that vertex. As Figure 2.35 illustrates, the first three edges connect the new vertex to the vertices of the triangle in which it falls, and the latter three are created through edge flips. If the new vertex fell upon an edge of the original triangulation, there would be seven incident edges.

The three-flip bound is derived through the technique of *backward analysis*. The main principle of backward analysis is that after an algorithm terminates, one imagines reversing time and examining the algorithm's behavior as it runs backward to its starting state. In the case of Lawson's algorithm, imagine beginning with a complete Delaunay triangulation of all the input vertices, and removing one vertex at a time.

The power of backward analysis stems from the fact that a uniformly chosen random permutation read backward is still a uniformly chosen random permutation. Hence, imagine that triangulation vertices are randomly selected, one at a time from a uniform distribution, and removed from the triangulation. With time running backward, the amount of time it takes to remove a vertex from the triangulation is proportional to the degree of the vertex. Because the average degree of vertices in a planar graph is bounded below six, the expected number of edge flips observed when a randomly chosen vertex is removed is bounded below three. Because any triangulation of a six-sided polygon has four triangles, the expected number of new triangles created when a randomly chosen vertex is removed is bounded below four.

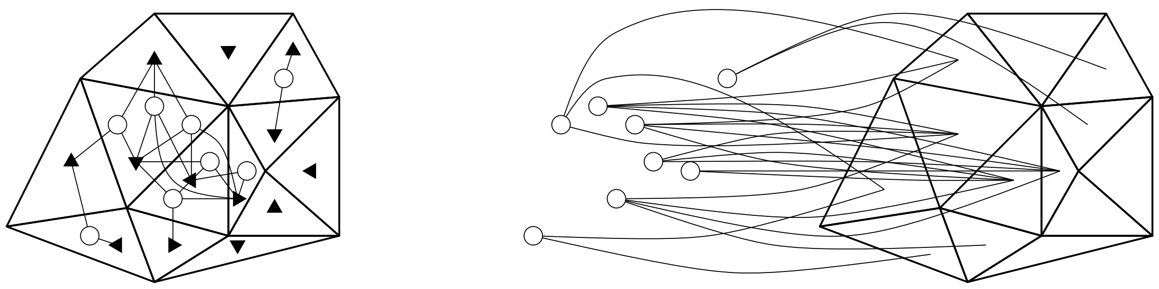


Figure 2.38: Two depictions of a single conflict graph.

Hence, when Lawson's algorithm is running forward in time, the expected number of edge flips required to insert a vertex is at most three, and the expected number of triangles deleted is at most four. Unfortunately, this result is not strictly relevant to most Delaunay-based mesh generation algorithms, because these algorithms usually do not know the entire set of vertices in advance, the vertex insertion order cannot be randomized. Nevertheless, the result gives useful intuition for why constant-time vertex insertion is so commonly observed in mesh generation.

During the incremental construction of the Delaunay triangulation of an arbitrary set of vertices, edge flips are not the only cost. In many circumstances, the dominant cost is the time required for *point location*: finding the triangle or tetrahedron in which a vertex lies, so that the vertex may be inserted. Fortunately, most Delaunay-based mesh generation algorithms insert vertices in places that have already been identified as needing refinement, and thus the location of each new vertex is already approximately known. However, in a general-purpose Delaunay triangulator, point location can be expensive.

I describe next a point location scheme that makes it possible for randomized incremental Delaunay triangulation to run in expected $\mathcal{O}(n \log n)$ time in two dimensions. All vertices must be known from the beginning of the run, so the bound on the expected running time is not valid for most Delaunay-based mesh generators, but the point location algorithm can still be used profitably in practice.

The bound was first obtained by Clarkson and Shor [18], who perform point location by maintaining a *conflict graph*. Imagine that, midway through the incremental algorithm, p vertices (of the n input vertices) have been inserted, yielding a Delaunay triangulation D_p; and $n - p$ vertices remain uninserted. Let t be a triangle of D_p, and let w be one of the $n - p$ vertices not yet inserted. If w is inside the circumcircle of t, then t are w are *in conflict*; clearly, t cannot be a triangle of the final triangulation D_n. Note that t cannot be one of the triangles that briefly appears and disappears during edge flipping; only triangles that exist in one of the triangulations D_i, $0 \leq i \leq n$, are considered to be in conflict with uninserted vertices.

The conflict graph is a bipartite graph in which each node represents a triangle or uninserted vertex, and each edge represents a conflict between a triangle and a vertex. Figure 2.38 depicts a conflict graph in two different ways. Each vertex insertion changes the conflict graph. When a vertex is inserted, the newly created triangles (incident to that vertex) may conflict with uninserted vertices, so conflicts are added to the conflict graph. The vertex insertion causes the deletion of each triangle the vertex conflicts with, and all conflicts involving those triangles are deleted from the conflict graph. Recall that the expected number of triangles deleted during a vertex insertion is less than four. Hence, the expected number of triangles that an uninserted vertex conflicts with is less than four. (The expectation is over all possible permutations of the vertex insertion order.)

Clarkson and Shor's original algorithm maintains the entire conflict graph, but the authors also suggest a simpler and more practical variant in which only one conflict is maintained for each uninserted vertex;

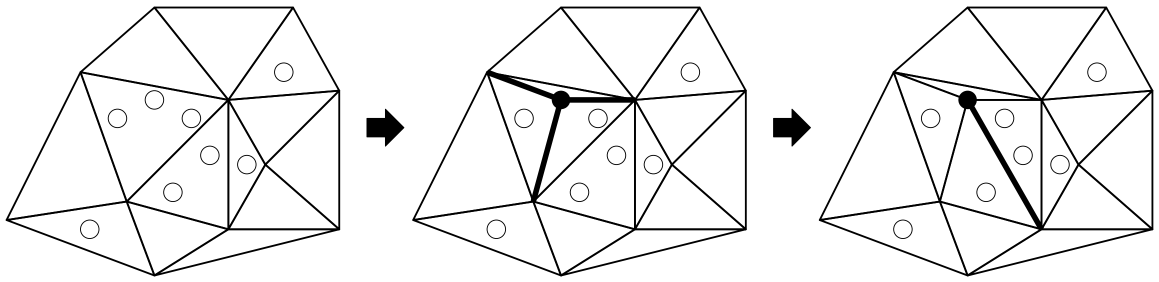


Figure 2.39: The simplified conflict graph maintains a mapping between uninserted vertices (open circles) and the triangles that contain them. The bounding box vertices and the edges incident to them are not shown.

the remaining conflicts are omitted from the *simplified conflict graph* (although they are still considered to be conflicts). For simplicity, assume that a large bounding box is used to contain the input vertices. In the simplified conflict graph, each uninserted vertex is associated with the triangle that contains it (Figure 2.39, left). If a vertex lies on an edge, one of the two containing triangles is chosen arbitrarily. For each uninserted vertex, the algorithm maintains a pointer to its containing triangle; and for each triangle, the algorithm maintains a linked list of uninserted vertices in that triangle. Thus, point location is easy: when a vertex is inserted, the simplified conflict graph identifies the triangle that contains the vertex in constant time.

When a triangle is subdivided into three triangles (Figure 2.39, center), two triangles are subdivided into four (because the newly inserted vertex lies on an edge of the triangulation), or an edge is flipped (Figure 2.39, right), the vertices in the deleted triangle(s) are redistributed among the new triangles as dictated by their positions.

Although Clarkson and Shor [18] and Guibas, Knuth, and Sharir [33] both provide ways to analyze this algorithm, the simplest analysis originates with Kenneth Clarkson and is published in a report by Seidel [63]. The proof relies on backward analysis.

The dominant cost of the incremental Delaunay triangulation algorithm is the cost of redistributing uninserted vertices to their new containing triangles each time a vertex is inserted. Each redistribution of an uninserted vertex to a new triangle, when a triangle is divided or an edge is flipped, takes $\mathcal{O}(1)$ time. How many such redistributions occur? The following lemma gives us a clue.

Lemma 15 *Each time an uninserted vertex is redistributed, a conflict is eliminated.*

This doesn't necessarily mean that the *number* of conflicts is reduced; new conflicts may be created.

Proof: Let w be an uninserted vertex, and let t be the triangle that w is associated with in the simplified conflict graph. Because w lies in t and is not a vertex of t, w is inside the circumcircle of t and conflicts with t.

When a new vertex v is inserted, there are two ways w might be redistributed. If t is eliminated by a subdivision or an edge flip, w is redistributed, and the conflict between w and t is eliminated (Figure 2.40, left and center). On the other hand, if t has already been eliminated, an edge flip may cause w to be redistributed again, as illustrated in Figure 2.40 (center and right). (This can occur any number of times.) Prior to the edge flip, two triangles adjoin the edge: a triangle that has v for a vertex and contains w, and another triangle s. (The first triangle exists only briefly during the insertion of v, and thus does not participate in the conflict graph.) The edge is being flipped because v is inside the circumcircle of s. It follows that w must also be

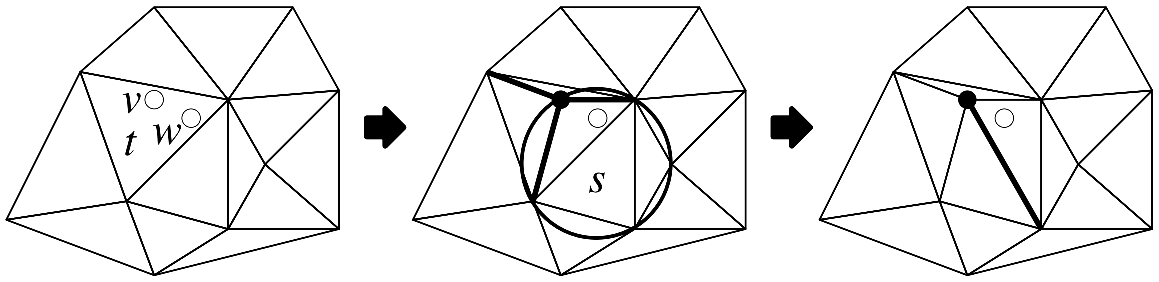


Figure 2.40: Each time w is redistributed, a conflict is eliminated: first the conflict between w and t, then the conflict between w and s.

inside the circumcircle of s. When the edge is flipped, s is deleted, and the conflict between w and s is eliminated. (It is irrelevant that this latter conflict was not represented in the simplified conflict graph.) ■

From Lemma 15 it follows that the time spent maintaining the simplified conflict graph is no greater than the number of conflicts eliminated during the incremental algorithm's run. A conflict must exist before it can be eliminated, so the number of conflicts eliminated is no greater than the number of conflicts created during the run. Therefore, the analysis strategy is to bound the number of conflicts that come into existence. This can be accomplished through backward analysis, which asks how many conflicts disappear as the algorithm runs backward.

Theorem 16 *Clarkson and Shor's two-dimensional randomized incremental Delaunay triangulation algorithm runs in $\mathcal{O}(n \log n)$ time.*

Proof: Consider the step wherein a $(p-1)$-vertex triangulation D_{p-1} is converted into a p-vertex triangulation D_p by inserting a randomly chosen vertex; but consider running the step in reverse. In the backward step, a random vertex v of D_p is chosen for deletion. What is the expected number of conflicts that disappear? There are $n-p$ uninserted vertices. Recall that the expected number of triangles that each uninserted vertex conflicts with is less than four. (This expectation is over all possible values of D_p; that is, considering every possible choice of p of the n input vertices.) By the linearity of expectation, the expected number of conflicts between uninserted vertices and triangles of D_p is less than $4(n-p)$. Each triangle of D_p has three vertices, so the probability that any given triangle is deleted when v is deleted is $\frac{3}{p}$. (The probability is actually slightly smaller, because some triangles have vertices of the bounding box, but $\frac{3}{p}$ is an upper bound. This bound applies regardless of which p vertices form D_p.) Hence, the expected number of conflicts that disappear when v is deleted is less than $4(n-p) \times \frac{3}{p} = 12\frac{n-p}{p}$.

Now, consider the algorithm running forward in time. When the initial bounding box is created, there are n conflicts. When the pth vertex is inserted, the expected number of conflicts created is less than $12\frac{n-p}{p}$. Hence, the expected total time required to maintain the simplified conflict graph, and the running time of the incremental algorithm, is $\mathcal{O}(n + \sum_{p=1}^{n} 12\frac{n-p}{p}) = \mathcal{O}(12n \sum_{p=1}^{n} \frac{1}{p}) = \mathcal{O}(n \log n)$.

The same analysis technique can be used, albeit with complications, to show that incremental Delaunay triangulation in higher dimensions can run in randomized $\mathcal{O}(n^{\lceil d/2 \rceil})$ time, where d is the dimension. Consult Seidel [63] for details.

Guibas, Knuth, and Sharir [33] point out that this point location method may be used even if the vertices are not known in advance (although the insertion order cannot be randomized and the $\mathcal{O}(n \log n)$ expected-time analysis does not hold). Guibas et al. maintain a data structure for point location called the *history dag*.

The history dag is a directed acyclic graph whose nodes represent every triangle that has been created during the course of incremental insertion, however briefly they may have existed. The root node represents the initial triangular bounding box. Whenever one or two triangles are replaced because of a vertex insertion, the history dag is augmented with pointers from the nodes that represent the eliminated triangles to the nodes that represent the triangles that replaced them.

When the history dag is used to locate a new vertex, the vertex passes through the same set of triangles as if it had been present in the simplified conflict graph from the beginning.

2.1.7 Gift-Wrapping Algorithms for Constructing Delaunay Triangulations

As Section 2.1.3 explained, gift-wrapping begins by finding a single Delaunay d-simplex, upon which the remaining Delaunay d-simplices crystallize one by one. Say that a $(d-1)$-face of a Delaunay d-simplex is *unfinished* if the algorithm has not yet identified the other Delaunay d-simplex that shares the face. To *finish* the face is to find the other simplex. (If the $(d-1)$-face lies in the boundary of the convex hull of the input, the face becomes *finished* when the algorithm recognizes that there is no other adjoining d-simplex.)

The gift-wrapping algorithm maintains a dictionary of unfinished faces, which initially contains the $d+1$ faces of the first Delaunay d-simplex. Repeat the following step: remove an arbitrary unfinished face f from the dictionary, and search for a vertex that finishes f. This vertex must be above f, and have the property that the new d-simplex s that results has an empty circumsphere. Naïvely, this search takes $\Theta(n)$ time. If no vertex is above f, then f lies on the boundary of the convex hull. Otherwise, s becomes part of the growing triangulation. Check each $(d-1)$-face of s, except f, against the dictionary. If a face is already present in the dictionary, then the face is now finished, so remove it from the dictionary. Otherwise, the face is new, so insert it into the dictionary.

The running time is thus $\mathcal{O}(nn_s)$, where n is the number of input vertices and n_s is the number of d-simplices in the output. However, under some circumstances the time to finish a face can be improved. Dwyer [23] presents a sophisticated search algorithm based on *bucketing*. He partitions space into a grid of cubical buckets, and distributes the input vertices appropriately among the buckets. If the vertices are uniformly randomly distributed in a sphere, Dwyer offers techniques for searching the buckets that make it possible to finish each face in $\mathcal{O}(1)$ expected time, regardless of the dimension! Hence, the entire Delaunay triangulation may be constructed in $\mathcal{O}(n_s)$ expected time.

Of course, not all real-world point sets are so nicely distributed. Bucketing is a poor search technique if there are large variations in the spacing of vertices. It is easy to construct a vertex set for which most of the vertices fall into one bucket.

Gift-wrapping can construct constrained Delaunay triangulations as well. To finish a face, one must test the visibility of each vertex from that face, which is most easily done by testing each vertex against each constraining facet. The running time is thus $\mathcal{O}(nn_f n_s)$, where n_f is the number of $(d-1)$-simplices in the triangulations of the input $(d-1)$-facets. This leaves much room for improvement, and Dwyer's technique probably cannot be applied so successfully as it is to ordinary Delaunay triangulations. Here, I offer a few practical suggestions without asymptotic guarantees.

For most applications, the fastest way to form the CDT of a ridge-protected PLC (albeit not in the worst case) is to use the best available algorithm to find an unconstrained Delaunay triangulation of the input vertices, then recover the $(d-1)$-facets one by one. Each $(d-1)$-facet f may be recovered by deleting the d-simplices whose interiors it intersects, then retriangulating the polytopes now left empty on each side of f (recall Figure 2.16). It is easy to show that all of the simplices not thus deleted are still constrained

Delaunay. Since a CDT of the new configuration exists, each empty polytope can be triangulated with constrained Delaunay simplices. If these polytopes are typically small, the performance of the algorithm used to triangulate them is not critical, and gift-wrapping will suffice.

The best approach to triangulating a PLC might be to start with a Delaunay triangulation of the vertices of the $(d-1)$-facets, then recover the $(d-1)$-facets themselves, and then finally insert the remaining vertices incrementally.

2.1.8 Other Algorithms for Constructing Delaunay Triangulations

The first $\mathcal{O}(n \log n)$ algorithm for two-dimensional Delaunay triangulation was not an incremental algorithm, but a divide-and-conquer algorithm. Shamos and Hoey [64] developed an algorithm for the construction of a Voronoi diagram, which may be easily dualized to form a Delaunay triangulation. In programming practice, Shamos and Hoey's algorithm is unnecessarily complicated, because forming a Delaunay triangulation directly is much easier, and is in fact the easiest way to construct a Voronoi diagram. Lee and Schachter [44] were the first to publish a divide-and-conquer algorithm that directly constructs a Delaunay triangulation. The algorithm is nonetheless intricate, and Guibas and Stolfi [34] provide an important aid to programmers by filling out many tricky implementation details. Dwyer [22] offers an interesting modification to divide-and-conquer Delaunay triangulation that achieves better asymptotic performance on some vertex sets, and offers improved speed in practice as well. There is also an $\mathcal{O}(n \log n)$ divide-and-conquer algorithm for constrained Delaunay triangulations, thanks to Chew [13].

Another well-known $\mathcal{O}(n \log n)$ two-dimensional Delaunay triangulation algorithm is Fortune's sweepline algorithm [24]. Fortune's algorithm builds the triangulation by sweeping a horizontal line vertically up the plane. The growing triangulation accretes below the sweepline. The upper surface of the growing triangulation is called the *front*. When the sweepline collides with a vertex, a new edge is created that connects the vertex to the front. When the sweepline reaches the top of the circumcircle of a Delaunay triangle, the algorithm is confident that no vertex can lie inside that circumcircle, so the triangle is created.

2.2 Research in Unstructured Mesh Generation

The discussion in this chapter has heretofore been concerned with triangulations of complete vertex sets. Of course, a mesh generator rarely knows all the vertices of the final mesh prior to triangulation, and the real problem of meshing is deciding where to place vertices to ensure that the mesh has elements of good quality and proper sizes.

I attempt here only the briefest of surveys of mesh generation algorithms. Detailed surveys of the mesh generation literature have been supplied by Thompson and Weatherill [69] and Bern and Eppstein [6]. I focus my attention on algorithms that make use of Delaunay triangulations, and on algorithms that achieve provable bounds. I postpone algorithms due to L. Paul Chew and Jim Ruppert that share both these characteristics. They are described in detail in Chapter 3.

Only simplicial mesh generation algorithms are discussed here; algorithms for generating quadrilateral, hexahedral, or other non-simplicial meshes are omitted. The most popular approaches to triangular and tetrahedral mesh generation can be divided into three classes: Delaunay triangulation methods, advancing front methods, and methods based on grids, quadtrees, or octrees.

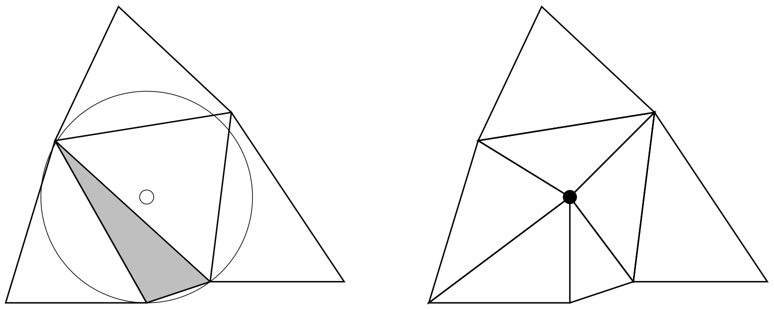


Figure 2.41: Each poor triangle is removed by inserting a vertex at its circumcenter and maintaining the Delaunay property.

2.2.1 Delaunay Mesh Generation

It is difficult to trace who first used Delaunay triangulations for finite element meshing, and equally difficult to tell where the suggestion arose to use the triangulation to guide vertex creation. These ideas have been intensively studied in the engineering community since the mid-1980s, and began to attract interest from the computational geometry community in the early 1990s.

I will name only a few scattered references from the voluminous literature. Many of the earliest papers suggest performing vertex placement as a separate step, typically using structured grid techniques, prior to Delaunay triangulation. For instance, Cavendish, Field, and Frey [11] generate grids of vertices from cross-sections of a three-dimensional object, then form their Delaunay tetrahedralization. The idea of using the triangulation itself as a guide for vertex placement followed quickly; for instance, Frey [29] removes poor quality elements from a triangulation by inserting new vertices at their *circumcenters*—the centers of their circumcircles—while maintaining the Delaunay property of the triangulation (Figure 2.41). This idea went on to bear vital theoretical fruit, as Chapters 3 and 4 will demonstrate.

I have mentioned that the Delaunay triangulation of a vertex set may be unsatisfactory for two reasons: elements of poor quality may appear, and input boundaries may fail to appear. Both these problems have been treated in the literature. The former problem is typically treated by inserting new vertices at the circumcenters [29] or centroids [71] of poor quality elements. It is sometimes also treated with an advancing front approach, discussed briefly in Section 2.2.2.

The problem of recovering missing boundaries may be treated in several ways. These approaches have in common that boundaries may have to be broken up into smaller pieces. For instance, each input segment is divided into a sequence of triangulation edges which I call *subsegments*, with a vertex inserted at each division point. In three dimensions, each facet of an object to be meshed is divided into triangular faces which I call *subfacets*. Vertices of the tetrahedralization lie at the corners of these subfacets.

In the earliest publications, boundary integrity was assured simply by spacing vertices sufficiently closely together on the boundary prior to forming a triangulation [29]—surely an error-prone approach. A better way to ensure the presence of input segments is to first form the triangulation, and then check whether any input segments are missing.

Missing segments can be recovered by one of several methods, which work in two or three dimensions. One method inserts a new vertex (while maintaining the Delaunay property of the mesh) at the midpoint of any missing segment, splitting it into two subsegments [71]. Sometimes, the two subsegments appear as edges of the resulting triangulation. If not, the subsegments are recursively split in turn. This method, sometimes called *stitching*, is described in more detail in Section 3.4.1. Although it is not obvious how this

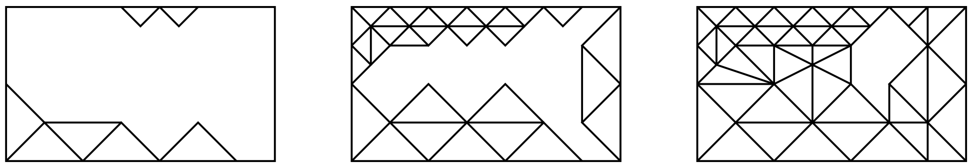


Figure 2.42: Several stages in the progression of an advancing front algorithm.

method might generalize to three-dimensional facet recovery, I will demonstrate in Section 4.2.1 that this generalization is possible and has some advantages over the next method I describe.

Another method, usually only used in three dimensions, can be applied to recover both missing segments and missing facets. This method inserts a new vertex wherever a face of the triangulation intersects a missing segment or an edge of the triangulation intersects a missing facet [72, 35, 73, 56]. The method is often coupled with flips [31, 72], which are used to reduce the number of vertices that must be inserted. The pessimistic results on constrained tetrahedralizations in Section 2.1.3 imply that, in three dimensions, flips cannot always achieve boundary recovery on their own; in some cases, new vertices must inevitably be inserted to fully recover a boundary.

Boundary recovery methods will be discussed further in Sections 3.4.1, 3.5.1, and 4.2.1.

2.2.2 Advancing Front Methods

Advancing front methods [45, 4, 37, 47] begin by dividing the boundaries of the mesh into edges (in two dimensions) or triangular faces (in three). These discretized boundaries form the initial *front*. Triangles or tetrahedra are generated one-by-one, starting from the boundary edges or faces and working toward the center of the domain, as illustrated in Figure 2.42. The exposed inner faces of these elements collectively form an *advancing front*.

Advancing front methods require a good deal of second-guessing, first to ensure that the initial division of the boundaries is prudent, and second to ensure that when the advancing walls of elements collide at the center of the mesh, they are merged together in a manner that does not compromise the quality of the elements. A poor choice of element sizes may result in disaster, as when a front of small elements collides with a front of large elements, making it impossible to fill the space between with nicely shaped elements. These problems are sufficiently difficult that there are, to my knowledge, no provably good advancing front algorithms. Advancing front methods typically create astonishingly good triangles or tetrahedra near the boundaries of the mesh, but are less effective where fronts collide.

In three dimensions, generating the surface mesh may be a difficult problem itself. Ironically, the mesh generator described by Marcum and Weatherill [46] uses a Delaunay-based mesh generator to create a complete tetrahedralization, then throws away the tetrahedralization except for the surface mesh, which is used to seed their advancing front algorithm.

Mavriplis [47] combines the Delaunay triangulation and advancing front methods. The combination makes a good deal of sense, because a Delaunay triangulation in the interior of the mesh is a useful search structure for determining how close different fronts are to each other. (Some researchers use background grids for this task.) Conversely, the advancing front method may be used as a vertex placement method for Delaunay meshing. A sensible strategy might be to abandon the advancing front shortly before fronts collide, and use a different vertex placement strategy (such as inserting vertices at circumcenters or centroids of poor quality elements) in the center of the mesh, where such strategies tend to be most effective.

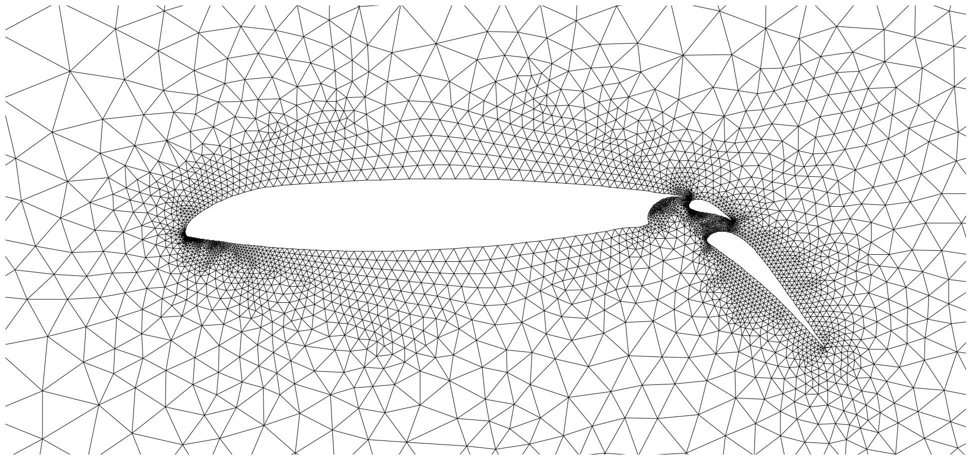


Figure 2.43: Mesh produced by an advancing front, moving outward from an airfoil.

Figure 2.43 depicts one of the world's most famous meshes, generated by an advancing front method of Barth and Jesperson [5]. The mesh is the Delaunay triangulation of vertices placed along an advancing front moving outward from an airfoil. The problems associated with colliding fronts are reduced in circumstances like this, where one is meshing the exterior, rather than the interior, of an object.

2.2.3 Grid, Quadtree, and Octree Methods

The last decade has seen the emergence of mesh generation algorithms with provably good bounds.

Baker, Grosse, and Rafferty [3] gave the first algorithm to triangulate PSLGs with guaranteed upper and lower bounds on element angle. By placing a fine uniform grid over a PSLG, warping the edges of the grid to fit the input segments and vertices, and triangulating the warped grid, they are able to construct a triangular mesh whose angles are bounded between 13° and 90° (except where the input PSLG has angles smaller than 13°; these cannot be improved). The elements of the mesh are of uniform size.

To produce graded meshes, some researchers have turned to *quadtrees*. A quadtree is a recursive data structure used to efficiently manipulate multiscale geometric objects in the plane. Quadtrees recursively partition a region into axis-aligned squares. A top-level square called the *root* encloses the entire input PSLG. Each quadtree square can be divided into four *child* squares, which can be divided in turn, as illustrated in Figure 2.44(a). *Octrees* are the generalization of quadtrees to three dimensions; each cube in an octree can be subdivided into eight cubes. See Samet [61] for a survey of quadtree data structures.

Meshing algorithms based on quadtrees and octrees have been used extensively in the engineering community for over a decade [74, 75, 65]. Their first role in mesh generation with provable bounds appears in a paper by Bern, Eppstein, and Gilbert [7]. The Bern et al. algorithm triangulates a polygon with guaranteed bounds on both element quality and the number of elements produced. All angles (except small input angles) are greater than roughly 18.4°, and the mesh is *size-optimal*: the number of elements in the final mesh is at most a constant factor larger than the number in the best possible mesh that meets the same bound on minimum angle.

The angle bound applies to triangulations of polygons with polygonal holes, but cannot be extended to general PSLGs, as Section 3.7 will show. Figure 2.44(b) depicts a mesh generated by one variant of the Bern et al. algorithm. For this illustration, a set of input vertices was specified (with no constraining segments),

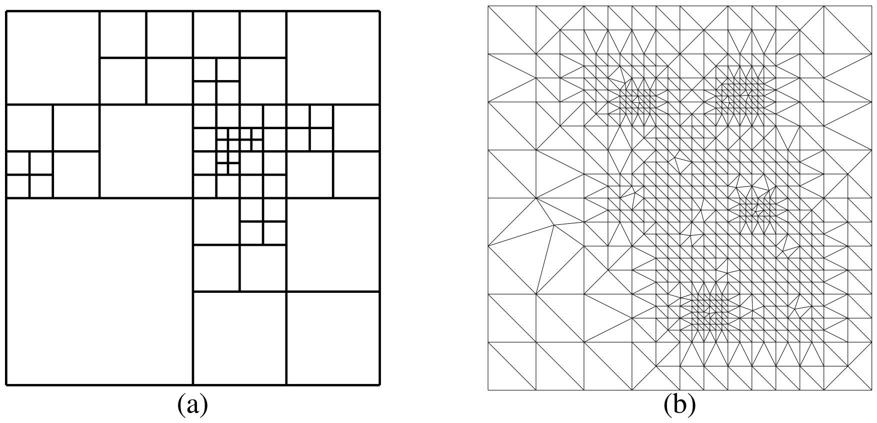


Figure 2.44: (a) A quadtree. (b) A quadtree-based triangulation of a vertex set, with no angle smaller than 20° (courtesy Marshall Bern).

and a mesh was generated (adding a great many additional vertices) that accommodates the input vertices and has no angle smaller than 20°. Figure 3.7 (top) on Page 55 depicts a mesh of a polygon with holes.

The algorithm of Bern et al. works by constructing a quadtree that is dense enough to isolate each input feature (vertex or segment) from other features. Next, the quadtree is warped to coincide with input vertices and segments. (Warping changes the shape of the quadtree, but not its topology.) Finally, the squares are triangulated.

Neugebauer and Diekmann [53] have improved the results of Bern et al., replacing the square quadtree with a rhomboid quadtree so that the triangles of the final mesh tend to be nearly equilateral. Assuming there are no small input angles, polygonal domains with polygonal holes and isolated interior points can be triangulated with all angles between 30° and 90°.

Remarkably, provably good quadtree meshing has been extended to polyhedra of arbitrary dimensionality. Mitchell and Vavasis [51, 52] have developed an algorithm based on octrees (and their higher-dimensional brethren) that triangulates polyhedra, producing size-optimal meshes with guaranteed bounds on element aspect ratios. The generalization to more than two dimensions is quite intricate, and the theoretical bounds on element quality are not strong enough to be entirely satisfying in practice. Figure 2.45 depicts two meshes generated by Vavasis' QMG mesh generator. The mesh at left is quite good, whereas the mesh at right contains some tetrahedra of marginal quality, with many small angles visible on the surface.

In practice, the theoretically good mesh generation algorithms of Bern, Eppstein, and Gilbert [7] and Mitchell and Vavasis [51] often create an undesirably large number of elements. Although both algorithms are size-optimal, the constant hidden in the definition of size-optimality is large, and although both algorithms rarely create as many elements as their theoretical worst-case bounds suggest, they typically create too many nonetheless. In contrast, the Finite Octree mesh generator of Shephard and Georges [65] generates fewer tetrahedra, but offers no guarantee. Shephard and Georges eliminate poor elements, wherever possible, through mesh smoothing, described below.

2.2.4 Smoothing and Topological Transformations

All the algorithms discussed thus far have the property that once they have decided to insert a vertex, the vertex is rooted permanently in place. In this section, I discuss techniques that violate this permanence.

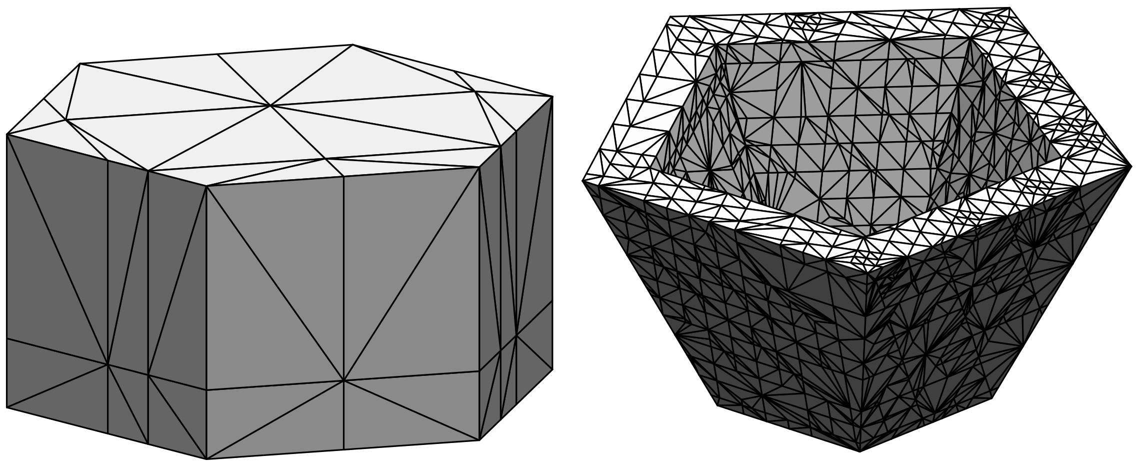


Figure 2.45: Two meshes generated by Stephen Vavasis' QMG package, an octree-based mesh generator with provable bounds. (Meshes courtesy Stephen Vavasis.)

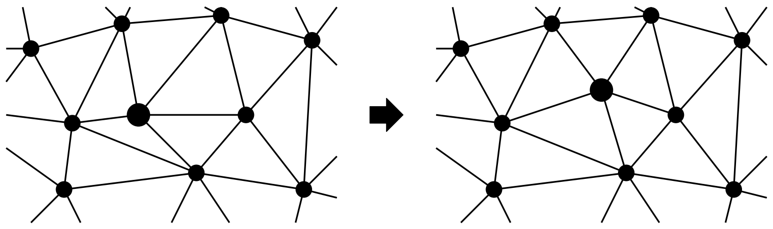


Figure 2.46: Laplacian smoothing.

These are not mesh generation methods; rather, they are mesh improvement procedures, which may be applied to a mesh generated by any of the methods discussed heretofore.

Smoothing is a technique wherein mesh vertices are moved to improve the quality of the adjoining elements. No changes are made to the topology of the mesh. Of course, vertices that lie in mesh boundaries might be constrained so that they can only move within a segment or facet, or they might be unable to move at all.

The most famous smoothing technique is *Laplacian smoothing*, in which a vertex is moved to the centroid of the vertices to which it is connected [36] (Figure 2.46), if such a move does not create collapsed or inverted elements. Typically, a smoothing algorithm will run through the entire set of mesh vertices several times, smoothing each vertex in turn. Laplacian smoothing is reasonably effective in two dimensions, but performs poorly in three.

Were Laplacian smoothing not so easy to implement and so fast to execute, it would be completely obsolete. Much better smoothing algorithms are available, based on constrained optimization techniques [54]. The current state of the art is probably the nonsmooth optimization algorithm discussed by Freitag, Jones, and Plassman [26] and Freitag and Ollivier-Gooch [27, 28]. The latter authors report considerable success with a procedure that maximizes the minimum sine of the dihedral angles of the tetrahedra adjoining the vertex being smoothed.

Another approach to mesh improvement is to use the *topological transformations* outlined by Canann [9], which are similar to ideas proposed by Frey and Field [30]. Examples of some transformations are illustrated in Figure 2.47. The familiar edge flip is included, but the other transformations have the effect

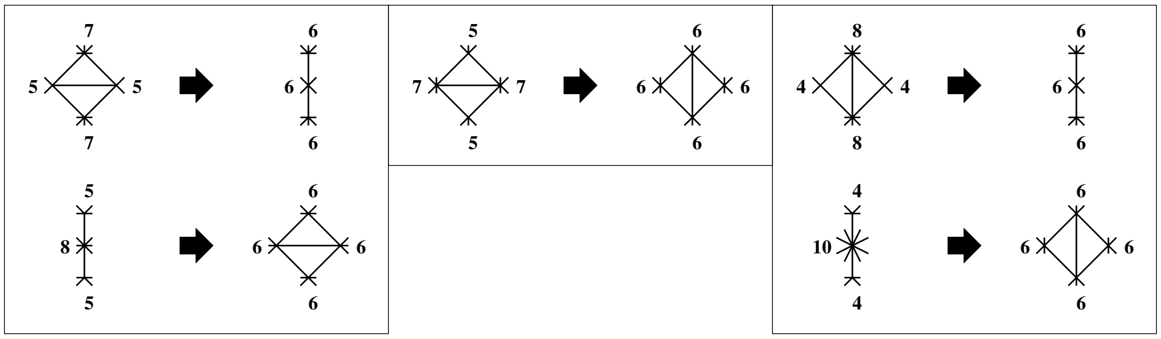


Figure 2.47: A selection of topological local transformations. Each node is labeled with its degree. These labels represent ideal cases, and are not the only cases in which these transformations would occur.

of inserting or deleting a vertex. An unusual aspect of Canann's approach is that he applies transformations based on the topology, rather than the geometry, of a mesh. In two dimensions, the ideal degree of a vertex is presumed to be six (to echo the structure of a lattice of equilateral triangles), and transformations are applied in an attempt to bring the vertices of the mesh as close to this ideal as possible. Canann claims that his method is fast because it avoids geometric calculations and makes decisions based on simple topological measures. The method relies upon smoothing to iron out any geometric irregularities after the transformations are complete. The research is notable because of the unusually large number of transformations under consideration.

Other researchers have considered mixing smoothing with topological transformations, but typically consider only a limited set of transformations, often restricted to 2-3 and 3-2 flips. For instance, Golias and Tsiboukis [32] report obtaining good results in tetrahedral mesh improvement by alternating between Laplacian smoothing and flipping.

A more sophisticated approach is taken by Freitag and Ollivier-Gooch [27, 28], who combine optimization-based smoothing with several transformations, including 2-3 and 3-2 flips, as well as another set of transformations they refer to as "edge swapping." Figure 2.48 demonstrates the results obtained by these techniques. In these before-and-after images, tetrahedra with poor dihedral angles are shaded. Before mesh improvement, the dihedral angles range from 0.66° to 178.88°. Afterward, they range from 13.67° to 159.82°.

As Delaunay tetrahedralizations lack the optimality properties of their two-dimensional counterparts, it is natural to ask whether one should forgo the Delaunay criterion, and instead use flips to directly maximize the minimum solid angle. Joe [38] studies this question experimentally, and concludes that a procedure that performs local flips to locally optimize the minimum solid angle is notably inferior to the Delaunay tetrahedralization. However, if one first constructs the Delaunay tetrahedralization, and then applies additional flips to locally improve the minimum solid angle, one does better than the Delaunay tetrahedralization alone. Joe's results indicate that a tetrahedralization that is locally optimal with respect to solid angle may be far from globally optimal. Although the Delaunay tetrahedralization does not maximize the minimum solid angle, it certainly seems to optimize something useful for mesh generation. Marcum and Weatherill [46] suggest that alternating between the Delaunay criterion and a min-max criterion (minimize the maximum dihedral angle) works even better.

Later research by Joe [41] indicates that local improvements can often be made by considering the effect of two consecutive flips (even though the first of the two flips may worsen element quality). Joe identifies several dual transformations that are frequently effective in practice, and several that rarely prove to be useful.

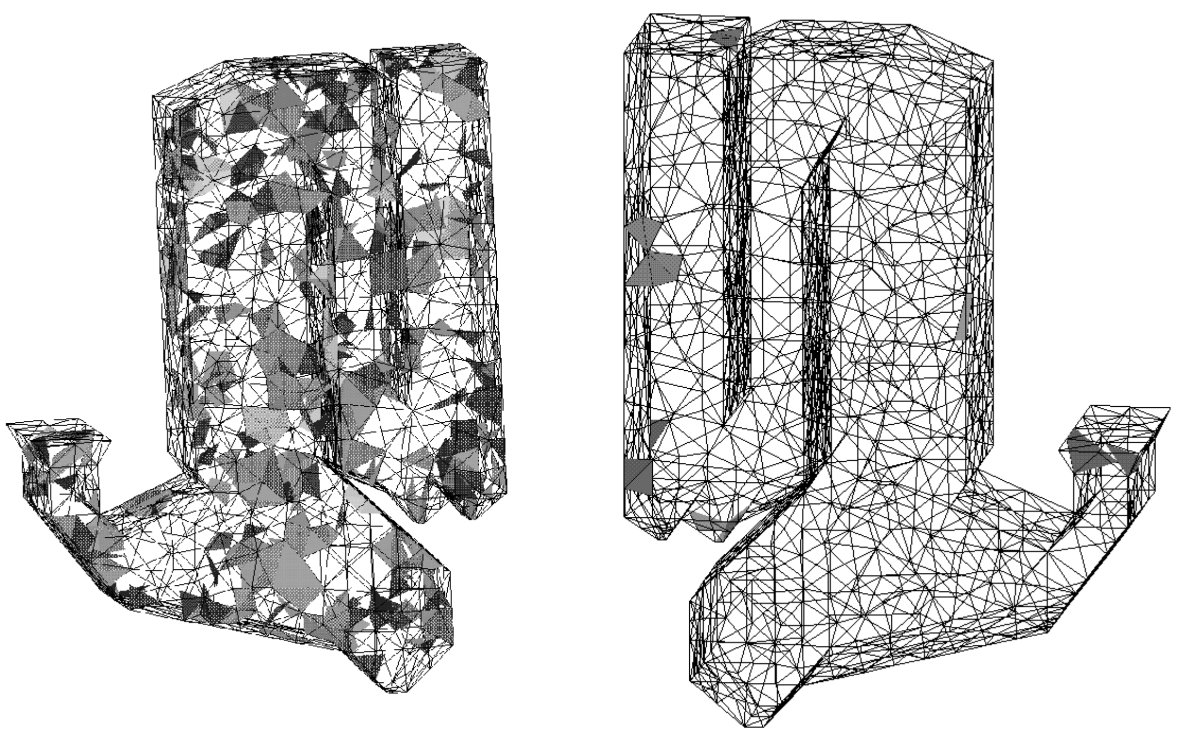


Figure 2.48: Tire incinerator mesh before and after mesh improvement. Shaded tetrahedra have dihedral angles smaller than 18° or greater than 162°. (Courtesy Lori Freitag and Carl Ollivier-Gooch.)

All of these mesh improvement techniques are applicable to meshes generated by the algorithms described in Chapters 3 and 4. However, I will not explore them further in this document.

Chapter 3

Two-Dimensional Delaunay Refinement Algorithms for Quality Mesh Generation

Delaunay refinement algorithms for mesh generation operate by maintaining a Delaunay or constrained Delaunay triangulation, which is refined by inserting carefully placed vertices until the mesh meets constraints on element quality and size.

These algorithms are successful because they exploit several favorable characteristics of Delaunay triangulations. One such characteristic, already mentioned in Chapter 2, is Lawson's result that a Delaunay triangulation maximizes the minimum angle among all possible triangulations of a point set. Another feature is that inserting a vertex is a local operation, and hence is inexpensive except in unusual cases. The act of inserting a vertex to improve poor quality elements in one part of a mesh will not unnecessarily perturb a distant part of the mesh that has no bad elements. Furthermore, Delaunay triangulations have been extensively studied, and good algorithms are available.

The greatest advantage of Delaunay triangulations is less obvious. The central question of any Delaunay refinement algorithm is, "Where should the next vertex be inserted?" As this chapter will demonstrate, a reasonable answer is, "As far from other vertices as possible." If a new vertex is inserted too close to another vertex, the resulting small edge will engender thin triangles.

Because a Delaunay triangle has no vertices in its circumcircle, a Delaunay triangulation is an ideal search structure for finding points that are far from other vertices. (It's no coincidence that the circumcenter of each triangle of a Delaunay triangulation is a vertex of the corresponding Voronoi diagram.)

This chapter begins with a review of Delaunay refinement algorithms introduced by L. Paul Chew and Jim Ruppert. Ruppert [58] proves that his algorithm produces nicely graded, size-optimal meshes with no angles smaller than about 20.7°. I also discuss theoretical and practical issues in triangulating regions with small angles. The foundations built here undergird the three-dimensional Delaunay refinement algorithms examined in the next chapter.

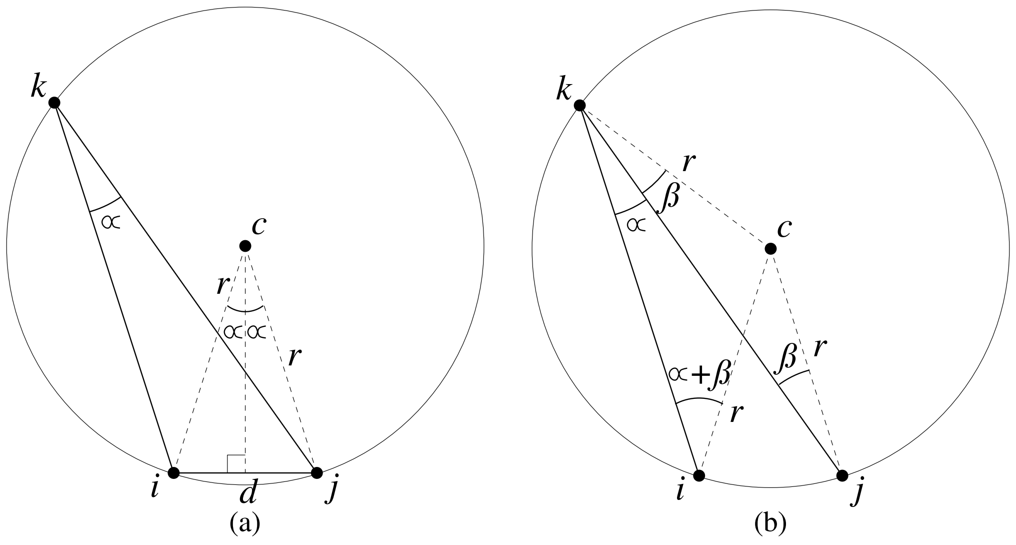


Figure 3.1: (a) Diagram for proof that $d = 2r \sin \alpha$. (b) Diagram for proof that $\angle icj = 2\angle ikj$.

3.1 A Quality Measure for Simplices

In the finite element community, there are a wide variety of measures in use for the quality of an element, the most obvious being the smallest and largest angles associated with the element. Miller, Talmor, Teng, and Walkington [48] have pointed out that the most natural and elegant measure for analyzing Delaunay refinement algorithms is the *circumradius-to-shortest edge ratio* of a simplex: the radius of the circumsphere of the simplex divided by the length of the shortest edge of the simplex. This ratio is the metric that is naturally optimized by Delaunay refinement algorithms. One would like this ratio to be as small as possible.

But does optimizing this metric aid practical applications? In two dimensions, the answer is yes. A triangle's circumradius-to-shortest edge ratio r/d is related to its smallest angle α by the formula $r/d = 1/(2 \sin \alpha)$. The smaller a triangle's ratio, the larger its smallest angle.

To see this, let $\triangle ijk$ have circumcenter c and circumradius r, as illustrated in Figure 3.1(a). Suppose the length of edge ij is d, and the angle opposite this edge is $\alpha = \angle ikj$.

It is a well-known geometric fact that $\angle icj = 2\alpha$. See Figure 3.1(b) for a derivation. Let $\beta = \angle jkc$. Because $\triangle kci$ and $\triangle kcj$ are isosceles, $\angle kci = 180° - 2(\alpha + \beta)$ and $\angle kcj = 180° - 2\beta$. Subtracting the former from the latter, $\angle icj = 2\alpha$. (This derivation holds even if β is negative.)

Returning to Figure 3.1(a), it is apparent that $\sin \alpha = d/(2r)$. It follows that if the triangle's shortest edge has length d, then α is its smallest angle. Hence, if B is an upper bound on the circumradius-to-shortest edge ratio of all triangles in a mesh, then there is no angle smaller than $\arcsin \frac{1}{2B}$ (and vice versa). A triangular mesh generator is wise to make B as small as possible.

In this chapter, I shall describe two-dimensional Delaunay refinement algorithms due to Paul Chew and Jim Ruppert that act to bound the maximum circumradius-to-shortest edge ratio, and hence bound the minimum angle of a triangular mesh. Chew's algorithms produce meshes whose triangles' circumradius-to-shortest edge ratios are bounded below one, and hence their angles are bounded between 30° and 120°. Ruppert's algorithm can produce meshes whose triangles' ratios are bounded below $\sqrt{2}$, and hence their angles range between 20.7° and 138.6°.

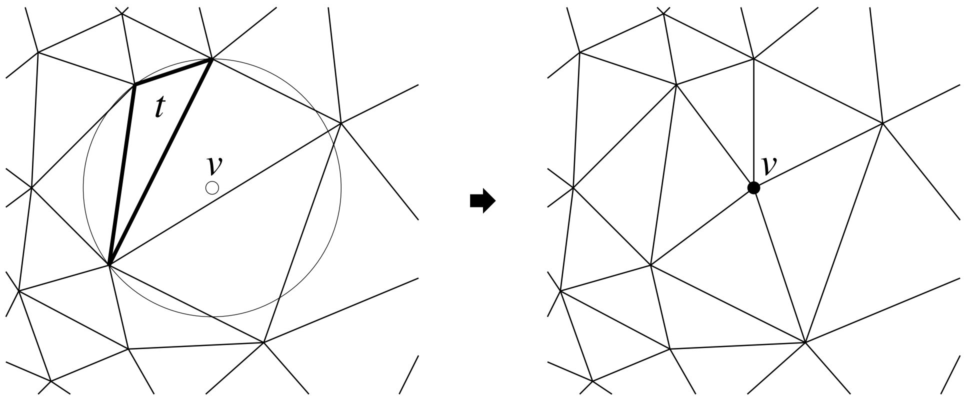


Figure 3.2: Any triangle whose circumradius-to-shortest edge ratio is larger than some bound B is split by inserting a vertex at its circumcenter. The Delaunay property is maintained, and the triangle is thus eliminated. Every new edge has length at least B times that of shortest edge of the poor triangle.

Unfortunately, a bound on circumradius-to-shortest edge ratio does not imply an angle bound in dimensions higher than two. Nevertheless, the ratio is a useful measure for understanding Delaunay refinement in higher dimensions.

3.2 The Key Idea Behind Delaunay Refinement

The central operation of Chew's and Ruppert's Delaunay refinement algorithms, as well as the three-dimensional algorithms described in the next chapter, is the insertion of a vertex at the circumcenter of a triangle or tetrahedron of poor quality. The Delaunay property is maintained, perhaps by Lawson's algorithm or the Bowyer/Watson algorithm for the incremental update of Delaunay triangulations. The poor-quality triangle cannot survive, because its circumcircle is no longer empty. For brevity, I refer to the act of inserting a vertex at a triangle's circumcenter as *splitting* a triangle. The idea dates back at least to the engineering literature of the mid-1980s [29]. If poor triangles are split one by one, either all will eventually be eliminated, or the algorithm will run forever.

The main insight behind all the Delaunay refinement algorithms is that the refinement loop is guaranteed to terminate if the notion of "poor quality" includes only triangles that have a circumradius-to-shortest edge ratio larger than some appropriate bound B. Recall that the only new edges created by the Delaunay insertion of a vertex v are edges connected to v (see Figure 3.2). Because v is the circumcenter of some Delaunay triangle t, and there were no vertices inside the circumcircle of t before v was inserted, no new edge can be shorter than the circumradius of t. Because t has a circumradius-to-shortest edge ratio larger than B, every new edge has length at least B times that of the shortest edge of t.

Ruppert's Delaunay refinement algorithm [59] employs a bound of $B = \sqrt{2}$, and Chew's second Delaunay refinement algorithm [16] employs a bound of $B = 1$. Chew's first Delaunay refinement algorithm [14] splits any triangle whose circumradius is greater than the length of the shortest edge in the entire mesh, thus achieving a bound of $B = 1$, but forcing all triangles to have uniform size. With these bounds, every new edge created is at least as long as some other edge already in the mesh. Hence, no vertex is ever inserted closer to another vertex than the length of the shortest edge in the initial triangulation. Delaunay refinement must eventually terminate, because the augmented triangulation will run out of places to put vertices.

Henceforth, a triangle whose circumradius-to-shortest edge ratio is greater than B is said to be *skinny*.

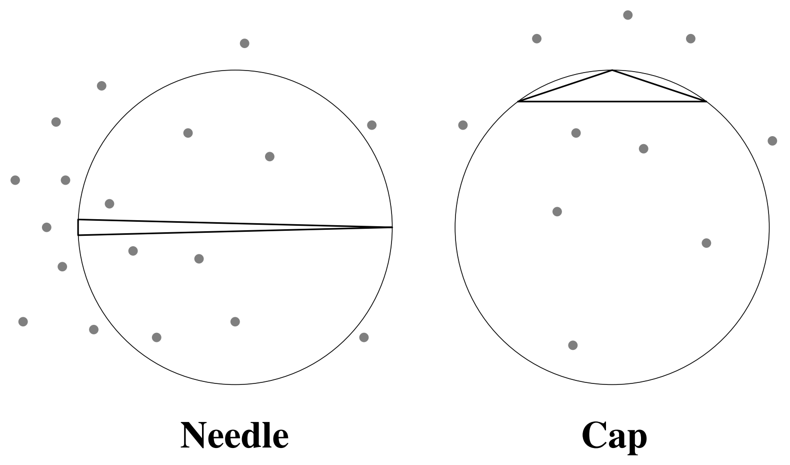


Figure 3.3: Skinny triangles have circumcircles larger than their smallest edges. Each skinny triangle may be classified as a needle, whose longest edge is much longer than its shortest edge, or a cap, which has an angle close to 180°. (The classifications are not mutually exclusive.)

Figure 3.3 provides a different intuition for why all skinny triangles are eventually eliminated by Delaunay refinement. The new vertices that are inserted into a triangulation (grey dots) are spaced roughly according to the length of the shortest nearby edge. Because skinny triangles have relatively large circumradii, their circumcircles are inevitably popped. When enough vertices are introduced that the spacing of vertices is somewhat uniform, large empty circumcircles cannot adjoin small edges, and no skinny triangles can remain in the Delaunay triangulation. Fortunately, the spacing of vertices does not need to be so uniform that the mesh is poorly graded; this fact is formalized in Section 3.4.4.

These ideas generalize without change to higher dimensions. Imagine a triangulation that has no boundaries—perhaps it has infinite extent, or perhaps it lies in a space that "wraps around" at the boundaries (like the videogame *Asteroids*). Regardless of the dimensionality, Delaunay refinement can eliminate all simplices having a circumradius-to-shortest edge ratio greater than one, without creating any edge smaller than the smallest edge already present.

Unfortunately, my description of Delaunay refinement thus far has a gaping hole: mesh boundaries have not been accounted for. The flaw in the procedure presented above is that the circumcenter of a poor triangle might not lie in the mesh at all. Delaunay refinement algorithms, including the two-dimensional algorithms of Chew and Ruppert and the three-dimensional algorithm described in the next chapter, are distinguished primarily by how they handle boundaries. Boundaries complicate mesh generation immensely, and the difficulty of coping with boundaries increases in higher dimensions.

3.3 Chew's First Delaunay Refinement Algorithm

Paul Chew has published at least two Delaunay refinement algorithms of great interest. The first, described here, produces triangulations of uniform density [14]. The second [16] is related to Ruppert's algorithm, but is not reviewed in these notes.

Given a triangulation whose shortest edge has length $h_{\min}$, Chew's first algorithm splits any triangle whose circumradius is greater than $h_{\min}$, and hence creates a uniform mesh. The algorithm clearly never introduces an edge shorter than $h_{\min}$, so any two vertices are separated by a distance of at least $h_{\min}$. The augmented triangulation will eventually run out of places to put vertices, and termination is inevitable.

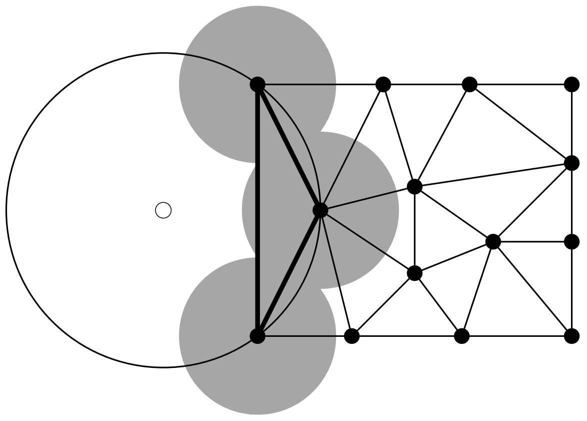


Figure 3.4: The bold triangle could be eliminated by inserting a vertex in its circumcircle. However, a vertex cannot be placed outside the triangulation, and it is forbidden to place a vertex within a distance of $h_{\min}$ from any other vertex. The forbidden region includes the shaded disks, which entirely cover the bold triangle.

Of course, the boundaries of the mesh may prevent some skinny triangles from being eliminated. Figure 3.4 illustrates an example in which there is a poor quality triangle, but no vertex can be placed inside its circumcircle without creating an edge smaller than $h_{\min}$, which would compromise the termination guarantee. Chew avoids this problem by subdividing the boundaries (possibly with a smaller value of $h_{\min}$) before commencing Delaunay refinement.

The input to Chew's algorithm is a PSLG that is presumed to be *segment-bounded*, meaning that the region to be triangulated is entirely enclosed within segments. (Any PSLG may be converted to a segment-bounded PSLG by any two-dimensional convex hull algorithm, if a convex triangulation is desired.) Untriangulated holes in the PSLG are permitted, but these must also be bounded by segments. A segment must lie anywhere a triangulated region of the plane meets an untriangulated region.

For some parameter h chosen by the user, all segments are divided into subsegments whose lengths are in the range $[h, \sqrt{3}h]$. New vertices are placed at the division points. The parameter h must be chosen small enough that some such partition is possible. Furthermore, h may be no larger than the smallest distance between any two vertices of the resulting partition. (If a vertex is close to a segment, this latter restriction may necessitate a smaller value of h than would be indicated by the input vertices alone.)

Chew constructs the constrained Delaunay triangulation of this modified PSLG, then applies Delaunay refinement. Circumcenters of triangles whose circumradii are larger than h are inserted, one at a time. When no such triangle remains, the algorithm terminates.

Because no subsegment has length greater than $\sqrt{3}h$, and specifically because no boundary subsegment has such length, the circumcenter of any triangle whose circumradius exceeds h falls within the mesh, at a distance of at least $h/2$ from any subsegment. Why? If a circumcenter is a distance less than $h/2$ from a subsegment whose length is no greater than $\sqrt{3}h$, then the circumcenter is a distance less than h from one of the subsegment's endpoints.

Upon termination, no circumradius is longer than h, and no edge is shorter than h, so no triangle has a circumradius-to-shortest edge ratio larger than one, and the mesh contains no angle smaller than 30°. Furthermore, no edge is longer than $2h$. (If the length of a Delaunay edge is greater than $2h$, then at least one of the two Delaunay triangles that contain it has a circumradius larger than h and is eligible for splitting.)

Chew's first algorithm handles boundaries in a simple and elegant manner, at the cost that it only produces meshes of uniform density, as illustrated in Figure 3.5.

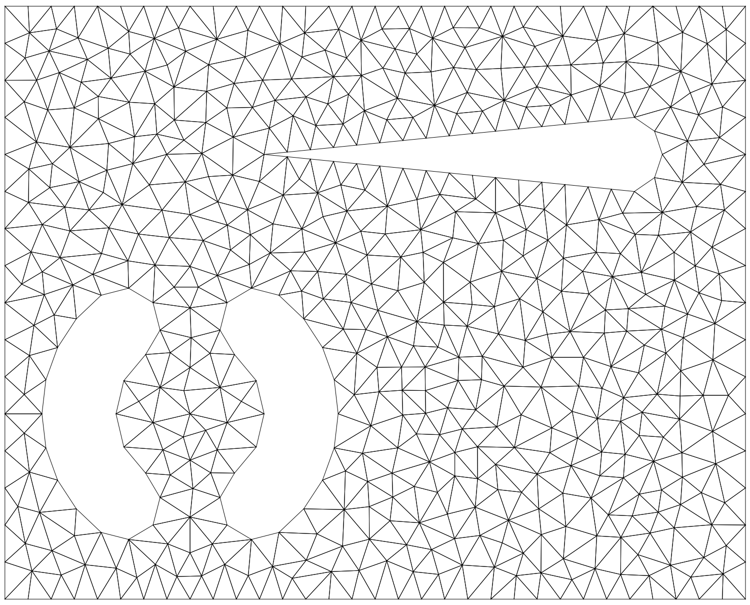


Figure 3.5: A mesh generated by Chew's first Delaunay refinement algorithm. (Courtesy Paul Chew).

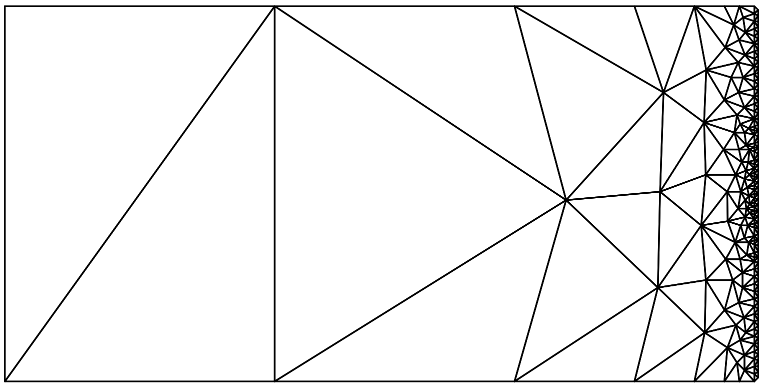


Figure 3.6: A demonstration of the ability of the Delaunay refinement algorithm to achieve large gradations in triangle size while constraining angles. No angle is smaller than 24°.

3.4 Ruppert's Delaunay Refinement Algorithm

Jim Ruppert's algorithm for two-dimensional quality mesh generation [59] is perhaps the first theoretically guaranteed meshing algorithm to be truly satisfactory in practice. It extends Chew's first algorithm by allowing the density of triangles to vary quickly over short distances, as illustrated in Figure 3.6. The number of triangles produced is typically smaller than the number produced either by Chew's algorithm or the Bern-Eppstein-Gilbert quadtree algorithm [7] (discussed in Section 2.2.3), as Figure 3.7 shows.

I have already mentioned that Chew independently developed a similar algorithm [16]. It may be worth noting that Ruppert's earliest publications of his results [57, 58] slightly predate Chew's. I present Ruppert's algorithm because it is accompanied by a theoretical framework with which he proves its ability to produce meshes that are both nicely graded and *size-optimal*. Size optimality means that, for a given bound on the minimum angle, the algorithm produces a mesh whose size (number of elements) is at most a constant factor

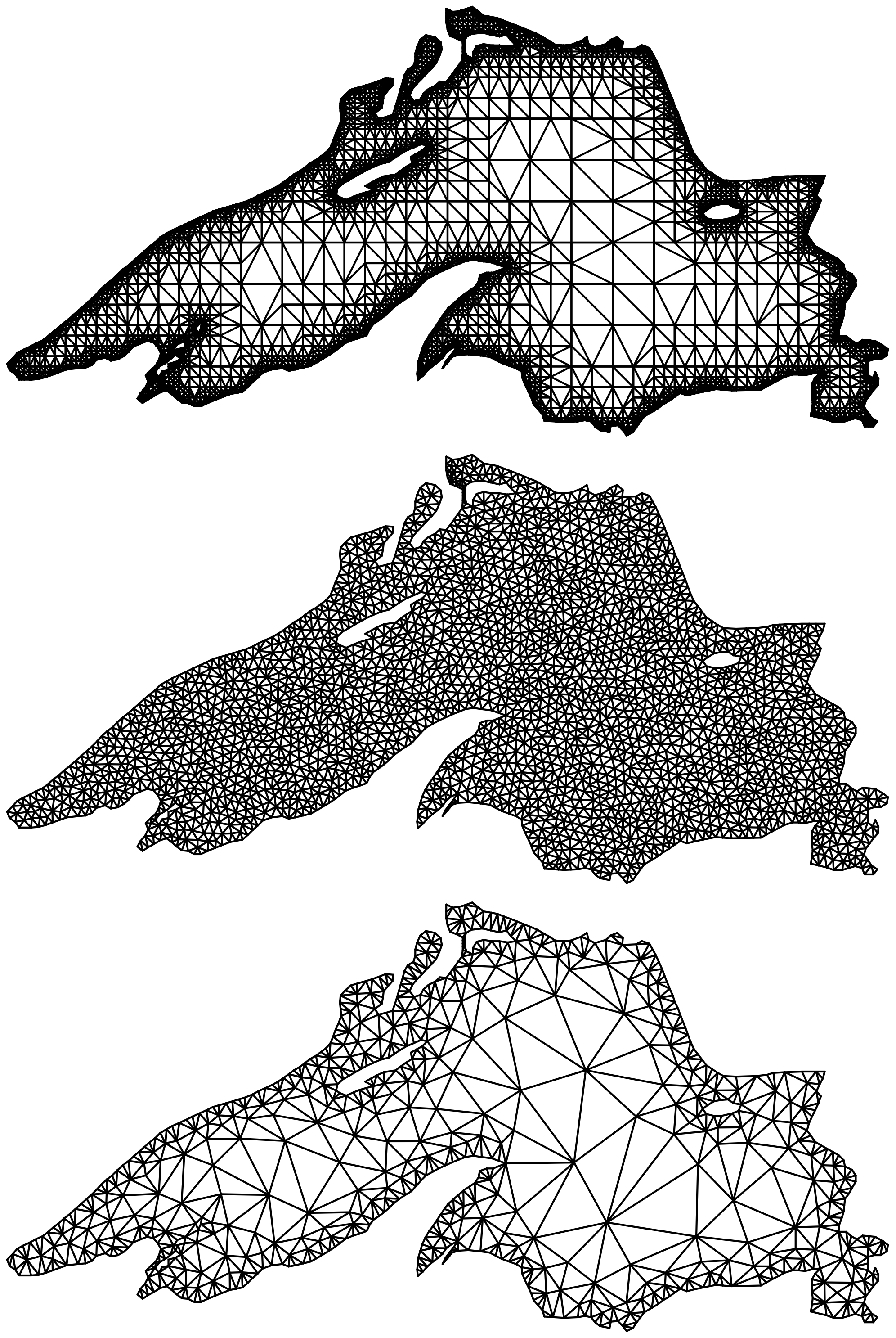


Figure 3.7: Meshes generated by the Bern-Eppstein-Gilbert quadtree-based algorithm (top), Chew's first Delaunay refinement algorithm (center), and Ruppert's Delaunay refinement algorithm (bottom). (The first mesh was produced by the program `tripoint`, courtesy Scott Mitchell.)

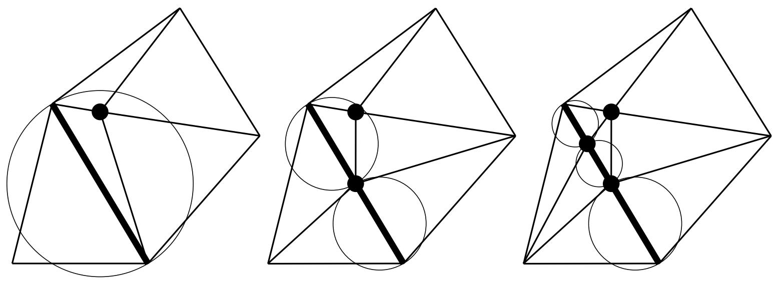


Figure 3.8: Segments are split recursively (while maintaining the Delaunay property) until no subsegments are encroached.

larger than the size of the smallest possible mesh that meets the same angle bound. (The constant depends upon the angle bound, but is independent of the input PSLG. The constant can be explicitly calculated for any specific angle bound, but it is too large to be useful as a practical bound.)

3.4.1 Description of the Algorithm

Like Chew's algorithms, Ruppert's algorithm takes a segment-bounded PSLG as its input. Unlike Chew's algorithms, which maintain a constrained Delaunay triangulation, Ruppert's algorithm may refine either a constrained or an unconstrained Delaunay triangulation. Ruppert's presentation of his algorithm is based on unconstrained triangulations, and it is interesting to see how the algorithm responds to missing segments, so assume that the algorithm begins with the Delaunay triangulation of the input vertices, ignoring the input segments. Segments that are missing from the triangulation will be inserted as a natural consequence of the algorithm.

Ruppert's algorithm refines the mesh by inserting additional vertices (while maintaining the Delaunay property) until all triangles satisfy the quality constraint. Like Chew's algorithm, Ruppert's may divide each segment into subsegments—but not as the first step of the algorithm; rather, segment subdivision is interleaved with triangle splitting. Initially, each segment comprises one subsegment (whether it appears as an edge of the triangulation or not). Vertex insertion is governed by two rules.

- The *diametral circle* of a subsegment is the (unique) smallest circle that encloses the subsegment. A subsegment is said to be *encroached* if a vertex lies strictly inside its diametral circle, or if the subsegment does not appear as an edge of the triangulation. (Recall that the latter case usually implies the former, the only exceptions being degenerate examples where several vertices lie precisely on the diametral circle.) Any encroached subsegment that arises is immediately split into two subsegments by inserting a vertex at its midpoint, as illustrated in Figure 3.8. These subsegments have smaller diametral circles, and may or may not be encroached themselves; splitting continues until no subsegment is encroached.

- Each skinny triangle (having a circumradius-to-shortest edge ratio greater than some bound B) is normally split by inserting a vertex at its circumcenter, thus eliminating the triangle. However, if the new vertex would encroach upon any subsegment, then it is not inserted; instead, all the subsegments it would encroach upon are split.

Encroached subsegments are given priority over skinny triangles.

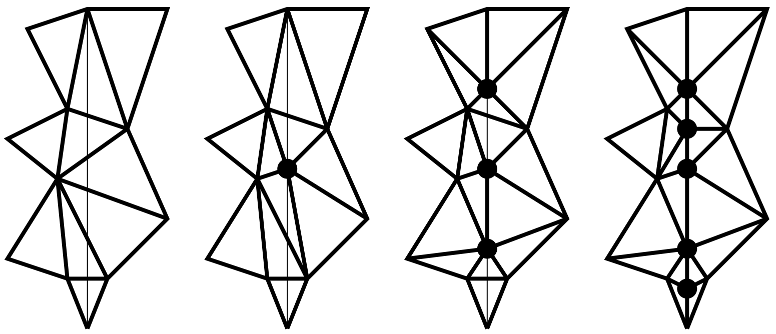


Figure 3.9: Missing segments are recovered by the same recursive splitting procedure used for encroached subsegments that are in the mesh. In this sequence of illustrations, the thin line represents a segment missing from the triangulation.

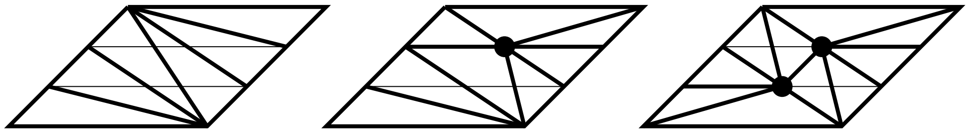


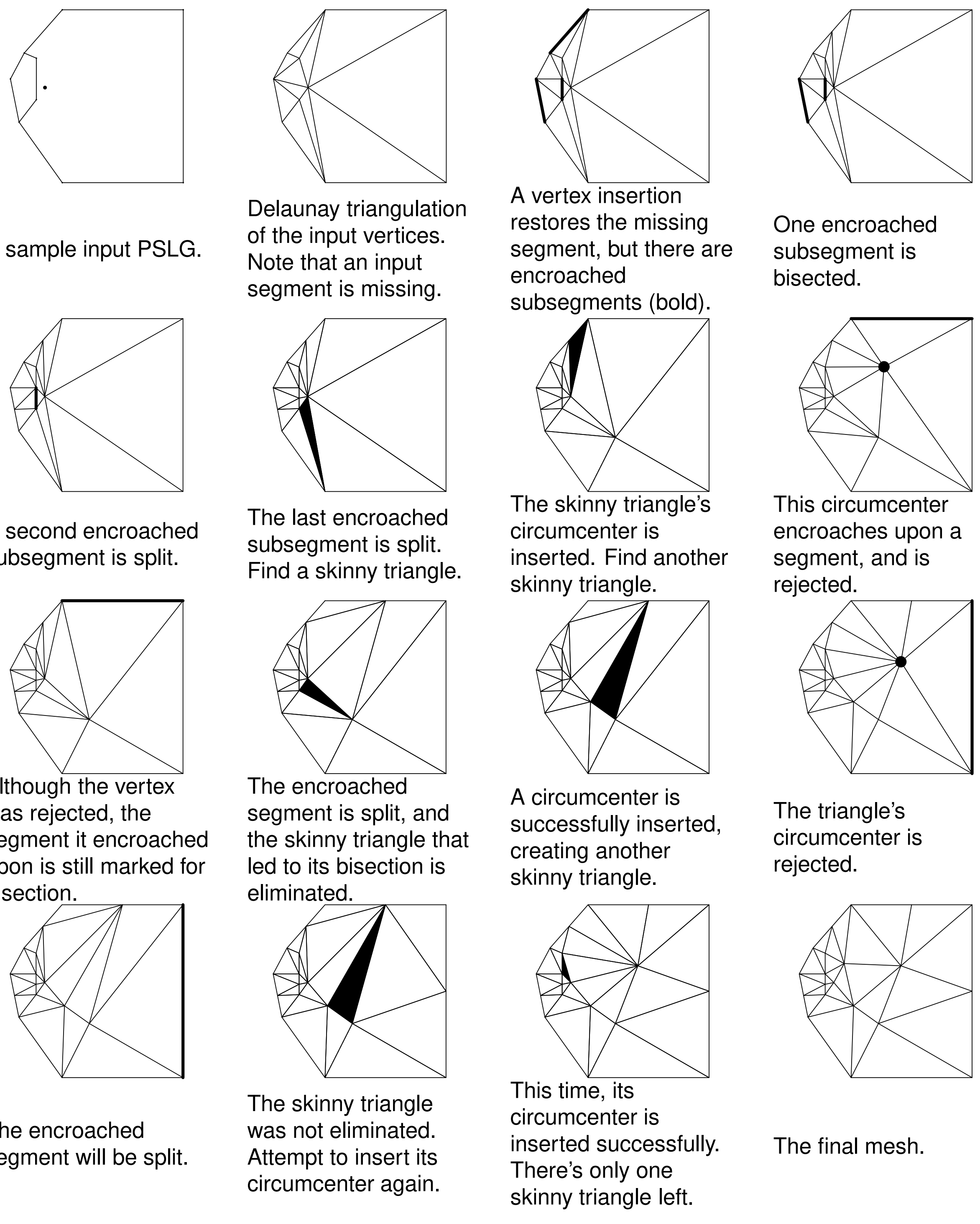
Figure 3.10: In this example, two segments (thin lines) must be recovered. The first is successfully recovered with a single vertex insertion, but the attempt to recover the second eliminates a subsegment of the first.

If a subsegment is missing from a Delaunay triangulation, then the subsegment is not strongly Delaunay, and there must be a vertex (other than its endpoints) on or inside its diametral circle. This observation is important because it unifies the theoretical treatment of missing subsegments and encroached subsegments that are not missing.

An implementation may give encroached subsegments that are not present in the mesh priority over encroached subsegments that are present (though it isn't necessary). If this option is chosen, the algorithm begins by recovering all the segments that are missing from the initial triangulation. Each missing segment is bisected by inserting a vertex into the mesh at the midpoint of the segment (more aptly, at the midpoint of the place where the segment should be). After the mesh is adjusted to maintain the Delaunay property, the two resulting subsegments may appear in the mesh. If not, the procedure is repeated recursively for each missing subsegment until the original segment is represented by a contiguous linear sequence of edges of the mesh, as illustrated in Figure 3.9. We are assured of eventual success because the Delaunay triangulation always connects a vertex to its nearest neighbor; once the spacing of vertices along a segment is sufficiently small, its entire length will be represented. In the engineering literature, this process is sometimes called *stitching*.

Unfortunately, the insertion of a vertex to recover a segment may eliminate a subsegment of some other segment (Figure 3.10). The subsegment thus eliminated is encroached, and must be split further. To avoid eliminating subsegments, one could maintain a constrained Delaunay triangulation (although it was not part of Ruppert's original specification). From an implementor's point of view, one may *lock* subsegments of the mesh by marking the edges that represent them to indicate that they are constrained. Flipping of such constrained edges is forbidden. This strategy can be implemented in two different ways: either the algorithm begins with a CDT with all segments locked, or the algorithm begins with a Delaunay triangulation and locks each subsegment when it first appears in the mesh.

It makes little material difference to Ruppert's algorithm whether one uses a CDT or not, because sub-



A sample input PSLG.

Delaunay triangulation of the input vertices. Note that an input segment is missing.

A vertex insertion restores the missing segment, but there are encroached subsegments (bold).

One encroached subsegment is bisected.

A second encroached subsegment is split.

The last encroached subsegment is split. Find a skinny triangle.

The skinny triangle's circumcenter is inserted. Find another skinny triangle.

This circumcenter encroaches upon a segment, and is rejected.

Although the vertex was rejected, the segment it encroached upon is still marked for bisection.

The encroached segment is split, and the skinny triangle that led to its bisection is eliminated.

A circumcenter is successfully inserted, creating another skinny triangle.

The triangle's circumcenter is rejected.

The encroached segment will be split.

The skinny triangle was not eliminated. Attempt to insert its circumcenter again.

This time, its circumcenter is inserted successfully. There's only one skinny triangle left.

The final mesh.

Figure 3.11: A complete run of Ruppert's algorithm with the quality bound $B = \sqrt{2}$. The first two images are the input PSLG and the (unconstrained) Delaunay triangulation of its vertices. In each image, highlighted subsegments or triangles are about to be split, and bold vertices are rejected because they encroach upon a subsegment.

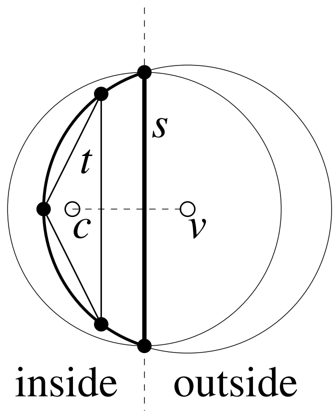


Figure 3.12: If the circumcenter v of a triangle t lies outside the triangulation, then some subsegment s is encroached.

segments whose diametral circles are not empty are considered encroached, and will be split eventually. When all encroached subsegments have been recursively bisected, and no encroached subsegments remain, all edges (including subsegments) of the triangulation are Delaunay. A mesh produced by Ruppert's algorithm is truly Delaunay, and not merely constrained Delaunay. However, locked subsegments may yield a faster implementation. Furthermore, CDTs have important advantages for triangulating nonconvex regions; see Section 3.5.1 for details. Chew's second algorithm, which is similar to Ruppert's, requires the use of a CDT throughout, and does not necessarily produce a mesh whose subsegments are all Delaunay.

Figure 3.11 illustrates the generation of a mesh by Ruppert's algorithm from start to finish. Several characteristics of the algorithm are worth noting. First, if the circumcenter of a skinny triangle is considered for insertion and rejected, it may still be successfully inserted later, after the subsegments it encroaches upon have been split. On the other hand, the act of splitting those subsegments is sometimes enough to eliminate the skinny triangle. Second, the smaller features at the left end of the mesh lead to the insertion of some vertices toward the right, but the size of the elements to the right remains larger than the size of the elements to the left. The smallest angle in the final mesh is 21.8°.

There is a loose end to tie up. One might ask what should happen if the circumcenter of a skinny triangle falls outside the triangulation. Fortunately, the following lemma shows that the question is moot.

Lemma 17 *Let T be a segment-bounded Delaunay triangulation (hence, any edge of T that belongs to only one triangle is a subsegment). Suppose that T has no encroached subsegments. Let v be the circumcenter of some triangle t of T. Then v lies in T.*

Proof: Suppose for the sake of contradiction that v lies outside T. Let c be the centroid of t; c clearly lies inside T. Because the triangulation is segment-bounded, the line segment cv must cross some subsegment s, as Figure 3.12 illustrates. Because cv is entirely contained in the interior of the circumcircle of t, the circumcircle must contain a portion of s; but the Delaunay property requires that the circumcircle be empty, so the circumcircle cannot contain the endpoints of s.

Say that a point is *inside* s if it is on the same side of s as c, and *outside* s if it is on the same side of s as v. Because the center v of the circumcircle of t is outside s, the portion of the circumcircle that lies strictly inside s (the bold arc in the illustration) is entirely enclosed by the diametral circle of s. The vertices of t lie upon t's circumcircle and are (not strictly) inside s. Up to two of the vertices of t may be the endpoints of s, but at least one vertex of t must lie strictly inside the diametral circle of s. But T has no encroached subsegments by assumption; the result follows by contradiction. ■

Lemma 17 offers another reason why encroached subsegments are given priority over skinny triangles. Because a circumcenter is inserted only when there are no encroached subsegments, one is assured that the circumcenter will be within the triangulation. The act of splitting encroached subsegments rids the mesh of triangles whose circumcircles lie outside it. The lemma also explains why the triangulation should be completely bounded by segments before applying the refinement algorithm.

In addition to being required to satisfy a quality criterion, triangles can also be required to satisfy a maximum size criterion. A finite element simulation may require that the triangles be small enough to model a phenomenon within some error bound. An implementation of Ruppert's algorithm can allow the user to specify an upper bound on allowable triangle areas or edge lengths, and the bound may be a function of each triangle's location in the mesh. Triangles that exceed the local upper bound are split, whether they are skinny or not. So long as the function bounding the sizes of triangles is itself bounded everywhere above some positive constant, there is no threat to the algorithm's termination guarantee.

3.4.2 Local Feature Sizes of Planar Straight Line Graphs

The claim that Ruppert's algorithm produces nicely graded meshes is based on the fact that the spacing of vertices at any location in the mesh is within a constant factor of the sparsest possible spacing. To formalize the idea of "sparsest possible spacing," Ruppert introduces a function called the *local feature size*, which is defined over E^2, relative to a specific PSLG.

Given a PSLG X, the local feature size $\text{lfs}(p)$ at any point p is the radius of the smallest disk centered at p that intersects two nonincident vertices or segments of X. (Two distinct features, each a vertex or segment, are said to be *incident* if they intersect.) Figure 3.13 illustrates the notion by giving examples of such disks for a variety of points.

The local feature size of a point is proportional to the sparsest possible spacing of vertices in the neighborhood of that point in any conforming triangulation whose triangles are of good quality. The function $\text{lfs}(\cdot)$ is continuous and has the property that its directional derivatives (where they exist) are bounded in the range $[-1, 1]$. This property sets a lower bound (within a constant factor to be derived shortly) on the rate at which edge lengths grade from small to large as one moves away from a small feature.

Lemma 18 (Ruppert [59]) *For any PSLG X, and any two points u and v in the plane,*

$$\text{lfs}(v) \leq \text{lfs}(u) + |uv|.$$

Proof: The disk having radius $\text{lfs}(u)$ centered at u intersects two nonincident features of X. The disk having radius $\text{lfs}(u) + |uv|$ centered at v contains the prior disk, and thus also intersects the same two features. Hence, the smallest disk centered at v that intersects two nonincident features of X has radius no larger than $\text{lfs}(u) + |uv|$. ■

This lemma generalizes without change to higher dimensions, so long as the answer to the question "Which pairs of points are said to lie on nonincident features?" is independent of u and v. In essence, the proof relies only on the triangle inequality: if u is within a distance of $\text{lfs}(u)$ of each of two nonincident features, then v is within a distance of $\text{lfs}(u) + |uv|$ of each of those same two features.

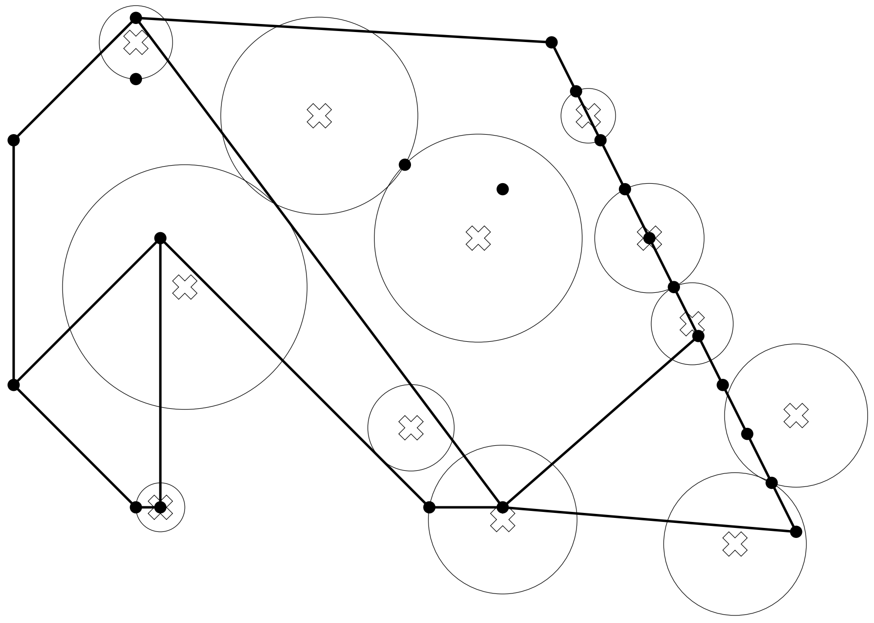


Figure 3.13: The radius of each disk illustrated is the local feature size of the point at its center.

3.4.3 Proof of Termination

In this section and the next, I present two proofs of the termination of Ruppert's algorithm. The first is similar to the proof that Chew's first algorithm terminates, and is included largely for its intuitive value; it also offers better bounds on the lengths of the shortest edges. The second proof is taken from Ruppert, but is rewritten in a more intuitive form that emphasizes the algorithm's natural tendency to optimize the circumradius-to-shortest edge ratio, which figures prominently in several extensions to the algorithm. The second proof offer better bounds on the lengths of the longer edges of a nonuniform mesh, and shows that the algorithm produces meshes that are nicely graded and size-optimal.

Both proofs require that $B \geq \sqrt{2}$, and that any two incident segments (segments that share an endpoint) in the input PSLG are separated by an angle of 60° or greater. (Ruppert asks for angles of at least 90°, but an improvement to the original proof is made here.) For the second proof, these inequalities must be strict.

A *mesh vertex* is any vertex that has been successfully inserted into the mesh (including the input vertices). A *rejected vertex* is any vertex that is considered for insertion but rejected because it encroaches upon a subsegment. With each mesh vertex or rejected vertex v, associate an *insertion radius* r_v, equal to the length of the shortest edge connected to v immediately after v is introduced into the triangulation. Consider what this means in three different cases.

- If v is an input vertex, then r_v is the Euclidean distance between v and the input vertex nearest v; see Figure 3.14(a).

- If v is a vertex inserted at the midpoint of an encroached subsegment, then r_v is the distance between v and the nearest encroaching mesh vertex; see Figure 3.14(b). If there is no encroaching mesh vertex (some triangle's circumcenter was considered for insertion but rejected as encroaching), then r_v is the radius of the diametral circle of the encroached subsegment, and hence the length of each of the two subsegments thus produced; see Figure 3.14(c).

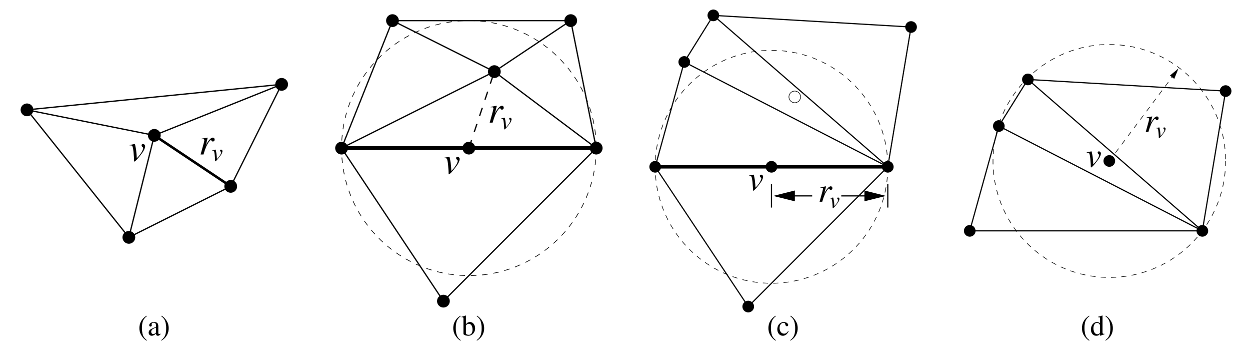


Figure 3.14: The insertion radius of a vertex v is the distance to the nearest vertex when v first appears in the mesh. (a) If v is an input vertex, r_v is the distance to the nearest other input vertex. (b) If v is the midpoint of a subsegment encroached upon by a mesh vertex, r_v is the distance to that vertex. (c) If v is the midpoint of a subsegment encroached upon only by a rejected vertex, r_v is the radius of the subsegment's diametral circle. (d) If v is the circumcenter of a skinny triangle, r_v is the radius of the circumcircle.

- If v is a vertex inserted at the circumcenter of a skinny triangle, then r_v is the circumradius of the triangle; see Figure 3.14(d).

If a vertex is considered for insertion but rejected because of an encroachment, its insertion radius is defined the same way—as if it had been inserted, even though it is not actually inserted.

Each vertex v, including any rejected vertex, has a *parent* vertex $p(v)$, unless v is an input vertex. Intuitively, $p(v)$ is the vertex that is "responsible" for the insertion of v. The parent $p(v)$ is defined as follows.

- If v is an input vertex, it has no parent.
- If v is a vertex inserted at the midpoint of an encroached subsegment, then $p(v)$ is the encroaching vertex. (Note that $p(v)$ might be a rejected vertex; a parent need not be a mesh vertex.) If there are several encroaching vertices, choose the one nearest v.
- If v is a vertex inserted (or rejected) at the circumcenter of a skinny triangle, then $p(v)$ is the most recently inserted endpoint of the shortest edge of that triangle. If both endpoints of the shortest edge are input vertices, choose one arbitrarily.

Each input vertex is the root of a tree of vertices. However, what is interesting is not each tree as a whole, but the sequence of ancestors of any given vertex, which forms a sort of history of the events leading to the insertion of that vertex. Figure 3.15 illustrates the parents of all vertices inserted or considered for insertion during the sample execution of Ruppert's algorithm in Figure 3.11.

I will use these definitions to show why Ruppert's algorithm terminates. The key insight is that no descendant of a mesh vertex has an insertion radius smaller than the vertex's own insertion radius. Certainly, no edge will ever appear that is shorter than the smallest feature in the input PSLG. To prove these facts, consider the relationship between a vertex's insertion radius and the insertion radius of its parent.

Lemma 19 *Let v be a vertex, and let $p = p(v)$ be its parent, if one exists. Then either $r_v \geq \mathrm{lfs}(v)$, or $r_v \geq Cr_p$, where*

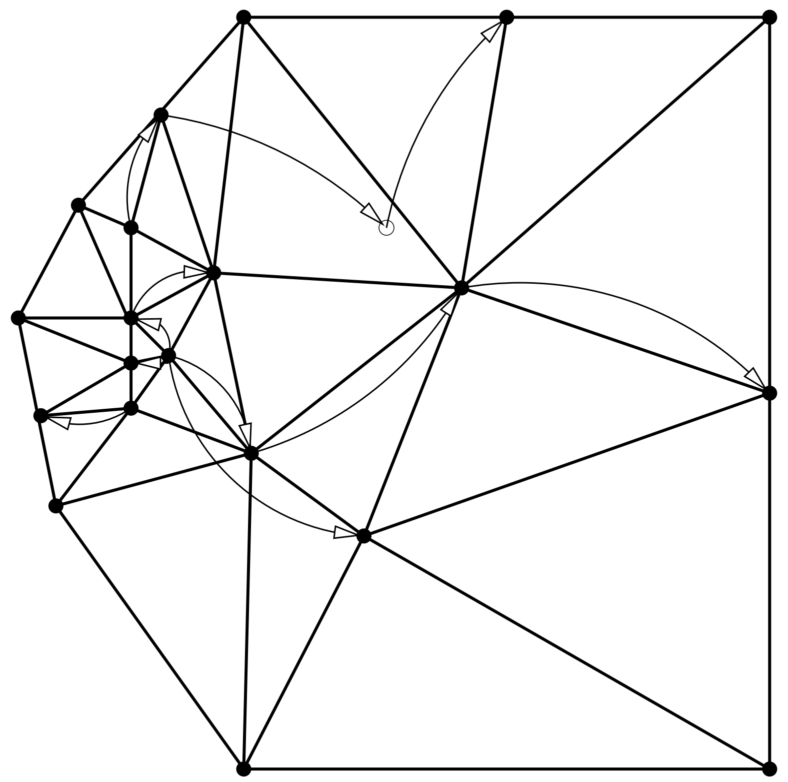


Figure 3.15: Trees of vertices for the example of Figure 3.11. Arrows are directed from parents to their children. Children include all inserted vertices and one rejected vertex.

- $C = B$ *if* v *is the circumcenter of a skinny triangle;*
- $C = \frac{1}{\sqrt{2}}$ *if* v *is the midpoint of an encroached subsegment and* p *is the circumcenter of a skinny triangle;*
- $C = \frac{1}{2\cos\alpha}$ *if* v *and* p *lie on incident segments separated by an angle of* α *(with* p *encroaching upon the subsegment whose midpoint is* v*), where* $45° \leq \alpha < 90°$*; and*
- $C = \sin\alpha$ *if* v *and* p *lie on incident segments separated by an angle of* $\alpha \leq 45°$.

Proof: If v is an input vertex, there is another input vertex a distance of r_v from v, so $\mathrm{lfs}(v) \leq r_v$, and the theorem holds.

If v is inserted at the circumcenter of a skinny triangle, then its parent $p = p(v)$ is the most recently inserted endpoint of the shortest edge of the triangle; see Figure 3.16(a). Hence, the length of the shortest edge of the triangle is at least r_p. Because the triangle is skinny, its circumradius-to-shortest edge ratio is at least B, so its circumradius is $r_v \geq Br_p$.

If v is inserted at the midpoint of an encroached subsegment s, there are four cases to consider. The first two are all that is needed to prove termination of Ruppert's algorithm if no angles smaller than $90°$ are present in the input. The last two cases consider the effects of acute angles.

- If the parent p is an input vertex, or was inserted in a segment not incident to the segment containing s, then by definition, $\mathrm{lfs}(v) \leq r_v$.

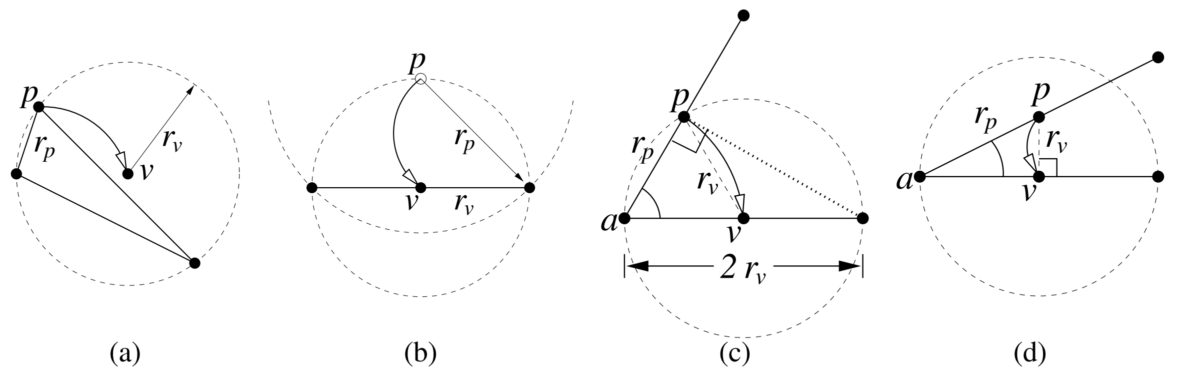


Figure 3.16: The relationship between the insertion radii of a child and its parent. (a) When a skinny triangle is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by the circumcenter of a skinny triangle, the child's insertion radius may be (arbitrarily close to) a factor of $\sqrt{2}$ smaller than the parent's, as this worst-case example shows. (c, d) When a subsegment is encroached upon by the midpoint of an incident subsegment, the relationship depends upon the angle α separating the two segments.

- If p is a circumcenter that was considered for insertion but rejected because it encroaches upon s, then p lies strictly inside the diametral circle of s. By the Delaunay property, the circumcircle centered at p contains no vertices, so its radius is limited by the nearest endpoint of s. Hence, $r_v > \frac{r_p}{\sqrt{2}}$; see Figure 3.16(b) for an example where the relation is nearly equality.

- If v and p lie on incident segments separated by an angle α where $45° \leq \alpha < 90°$, the vertex a (for "apex") where the two segments meet obviously cannot lie inside the diametral circle of s; see Figure 3.16(c). Because s is encroached upon by p, p lies inside its diametral circle. (If s is not present in the triangulation, p might lie on its diametral circle in a degenerate case.) To find the worst-case value of $\frac{r_v}{r_p}$, imagine that r_p and α are fixed; then $r_v = |vp|$ is minimized by making the subsegment s as short as possible, subject to the constraint that p cannot fall outside its diametral circle. The minimum is achieved when $|s| = 2r_v$; if s were shorter, its diametral circle would not contain p. Basic trigonometry shows that $|s| = 2r_v \geq \frac{r_p}{\cos \alpha}$.

- If v and p lie on incident segments separated by an angle α where $\alpha \leq 45°$, then $\frac{r_v}{r_p}$ is minimized not when p lies on the diametral circle, but when v is the orthogonal projection of p onto s, as illustrated in Figure 3.16(d). Hence, $r_v \geq r_p \sin \alpha$. ∎

Lemma 19 limits how quickly the insertion radius can decrease as one traverses a sequence of descendants of a vertex. If vertices with ever-smaller insertion radii cannot be generated, then edges shorter than existing ones cannot be introduced, and Delaunay refinement is guaranteed to terminate.

Figure 3.17 expresses this notion as a dataflow graph. Vertices are divided into three classes: input vertices (which are omitted from the figure because they cannot participate in cycles), *free vertices* inserted at circumcenters of triangles, and *segment vertices* interted at midpoints of subsegments. Labeled arrows indicate how a vertex can cause the insertion of a child whose insertion radius is some factor times that of its parent. If the graph contains no cycle whose product is less than one, termination is guaranteed. This goal is achieved by choosing B to be at least $\sqrt{2}$, and ensuring that the minimum angle between input segments is at least 60°. The following theorem formalizes these ideas.

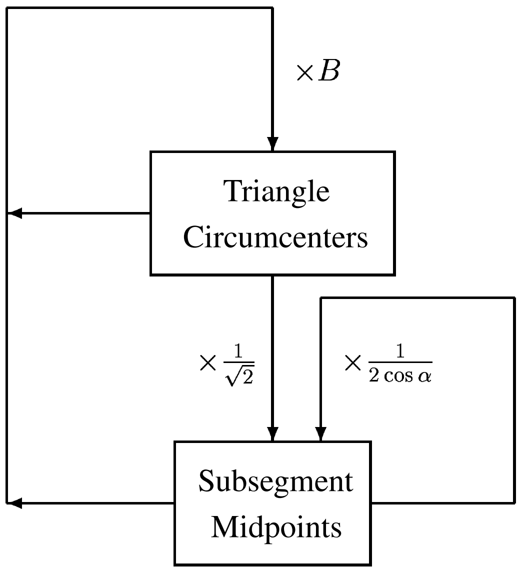


Figure 3.17: Dataflow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycles have a product smaller than one, Ruppert's Delaunay refinement algorithm will terminate. Input vertices are omitted from the diagram because they cannot contribute to cycles.

Theorem 20 *Let* $\text{lfs}_{\min}$ *be the shortest distance between two nonincident entities (vertices or segments) of the input PSLG*[1].

Suppose that any two incident segments are separated by an angle of at least $60°$, *and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is larger than* B, *where* $B \geq \sqrt{2}$. *Ruppert's algorithm will terminate, with no triangulation edge shorter than* $\text{lfs}_{\min}$.

Proof: Suppose for the sake of contradiction that the algorithm introduces an edge shorter than $\text{lfs}_{\min}$ into the mesh. Let e be the first such edge introduced. Clearly, the endpoints of e cannot both be input vertices, nor can they lie on nonincident segments. Let v be the most recently inserted endpoint of e.

By assumption, no edge shorter than $\text{lfs}_{\min}$ existed before v was inserted. Hence, for any ancestor a of v that is a mesh vertex, $r_a \geq \text{lfs}_{\min}$. Let $p = p(v)$ be the parent of v, and let $g = p(p)$ be the grandparent of v (if one exists). Consider the following cases.

- If v is the circumcenter of a skinny triangle, then by Lemma 19, $r_v \geq Br_p \geq \sqrt{2}r_p$.
- If v is the midpoint of an encroached subsegment and p is the circumcenter of a skinny triangle, then by Lemma 19, $r_v \geq \frac{1}{\sqrt{2}}r_p \geq \frac{B}{\sqrt{2}}r_g \geq r_g$. (Recall that p is rejected.)
- If v and p lie on incident segments, then by Lemma 19, $r_v \geq \frac{r_p}{2\cos\alpha}$. Because $\alpha \geq 60°$, $r_v \geq r_p$.

[1]Equivalently, $\text{lfs}_{\min} = \min_u \text{lfs}(u)$, where u is chosen from among the input vertices. The proof that both definitions are equivalent is omitted, but it relies on the recognition that if two points lying on nonincident segments are separated by a distance d, then at least one of the endpoints of one of the two segments is separated from the other segment by a distance of d or less. Note that $\text{lfs}_{\min}$ is not a lower bound for $\text{lfs}(\cdot)$ over the entire domain; for instance, a segment may have length $\text{lfs}_{\min}$, in which case the local feature size at its midpoint is $\text{lfs}_{\min}/2$.

In all three cases, $r_p \geq r_a$ for some ancestor a of p in the mesh. It follows that $r_p \geq \mathrm{lfs}_{\min}$, contradicting the assumption that e has length less than $\mathrm{lfs}_{\min}$. It also follows that no edge shorter than $\mathrm{lfs}_{\min}$ is ever introduced, and the algorithm must terminate. ■

Ruppert's algorithm terminates only when all triangles in the mesh have a circumradius-to-shortest edge ratio of B or better; hence, at termination, there are no angles smaller than $\arcsin \frac{1}{2B}$. If $B = \sqrt{2}$, the smallest value for which termination is guaranteed, no angle is smaller than 20.7°. Later, I will describe several modifications to the algorithm that improve this bound.

3.4.4 Proof of Good Grading and Size-Optimality

Theorem 20 guarantees that no edge of the final mesh is smaller than $\mathrm{lfs}_{\min}$. This guarantee may be satisfying for a user who desires a uniform mesh, but is not satisfying for a user who requires a spatially graded mesh. What follows is a proof that each edge of the output mesh has length proportional to the local feature sizes of its endpoints. Hence, edge lengths are determined by local considerations; features lying outside the disk that defines the local feature size of a point can only weakly influence the sizes of edges that contain that point. Triangle sizes vary quickly over short distances where such variation is desirable to help reduce the number of triangles in the mesh.

Lemma 19 was concerned with the relationship between the insertion radii of a child and its parent; the next lemma is concerned with the relationship between $\frac{\mathrm{lfs}(v)}{r_v}$ and $\frac{\mathrm{lfs}(p)}{r_p}$. For any vertex v, define $D_v = \frac{\mathrm{lfs}(v)}{r_v}$. Think of D_v as the one-dimensional density of vertices near v when v is inserted, weighted by the local feature size. One would like this density to be as small as possible. $D_v \leq 1$ for any input vertex, but D_v tends to be larger for a vertex inserted late.

Lemma 21 *Let v be a vertex with parent $p = p(v)$. Suppose that $r_v \geq C r_p$ (following Lemma 19). Then $D_v \leq 1 + \frac{D_p}{C}$.*

Proof: By Lemma 18, $\mathrm{lfs}(v) \leq \mathrm{lfs}(p) + |vp|$. By definition, the insertion radius r_v is $|vp|$ if p is a mesh vertex, whereas if p is rejected, then $r_v \geq |vp|$. Hence, we have

$$\begin{aligned} \mathrm{lfs}(v) &\leq \mathrm{lfs}(p) + r_v \\ &= D_p r_p + r_v \\ &\leq \frac{D_p}{C} r_v + r_v. \end{aligned}$$

The result follows by dividing both sides by r_v. ■

Lemma 21 generalizes to any dimension, because it relies only upon Lemma 18. Ruppert's first main result follows.

Lemma 22 (Ruppert [59]) *Suppose the quality bound B is strictly larger than $\sqrt{2}$, and the smallest angle between two incident segments in the input PSLG is strictly greater than 60°. There exist fixed constants $D_T \geq 1$ and $D_S \geq 1$ such that, for any vertex v inserted (or considered for insertion and rejected) at the circumcenter of a skinny triangle, $D_v \leq D_T$, and for any vertex v inserted at the midpoint of an encroached subsegment, $D_v \leq D_S$. Hence, the insertion radius of every vertex has a lower bound proportional to its local feature size.*

Proof: Consider any non-input vertex v with parent $p = p(v)$. If p is an input vertex, then $D_p = \frac{\text{lfs}(p)}{r_p} \leq 1$. Otherwise, assume for the sake of induction that the lemma is true for p, so that $D_p \leq D_T$ if p is a circumcenter, and $D_p \leq D_S$ if p is a midpoint. Hence, $D_p \leq \max\{D_T, D_S\}$.

First, suppose v is inserted or considered for insertion at the circumcenter of a skinny triangle. By Lemma 19, $r_v \geq Br_p$. Thus, by Lemma 21, $D_v \leq 1 + \frac{\max\{D_T, D_S\}}{B}$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen so that

$$1 + \frac{\max\{D_T, D_S\}}{B} \leq D_T. \tag{3.1}$$

Second, suppose v is inserted at the midpoint of a subsegment s. If its parent p is an input vertex or lies on a segment not incident to s, then $\text{lfs}(v) \leq r_v$, and the theorem holds. If p is the circumcenter of a skinny triangle (considered for insertion but rejected because it encroaches upon s), $r_v \geq \frac{r_p}{\sqrt{2}}$ by Lemma 19, so by Lemma 21, $D_v \leq 1 + \sqrt{2}D_T$.

Alternatively, if p, like v, is a subsegment midpoint, and p and v lie on incident segments, then $r_v \geq \frac{r_p}{2\cos\alpha}$ by Lemma 19, and thus by Lemma 21, $D_v \leq 1 + 2D_S \cos\alpha$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen so that

$$\begin{aligned} 1 + \sqrt{2}D_T &\leq D_S, \qquad \text{and} & (3.2) \\ 1 + 2D_S \cos\alpha &\leq D_S. & (3.3) \end{aligned}$$

If the quality bound B is strictly larger than $\sqrt{2}$, Inequalities 3.1 and 3.2 are simultaneously satisfied by choosing

$$D_T = \frac{B+1}{B-\sqrt{2}}, \qquad D_S = \frac{(1+\sqrt{2})B}{B-\sqrt{2}}.$$

If the smallest input angle $\alpha_{\min}$ is strictly greater than 60°, Inequalities 3.3 and 3.1 are satisfied by choosing

$$D_S = \frac{1}{1 - 2\cos\alpha_{\min}}, \qquad D_T = 1 + \frac{D_S}{B}.$$

One of these choices will dominate, depending on the values of B and $\alpha_{\min}$. However, if $B > \sqrt{2}$ and $\alpha_{\min} > 60°$, there are values of D_T and D_S that satisfy the lemma. ■

Note that as B approaches $\sqrt{2}$ or α approaches 60°, D_T and D_S approach infinity. In practice, the algorithm is better behaved than the theoretical bound suggests; the vertex spacing approaches zero only after B drops below one.

Theorem 23 (Ruppert [59]) *For any vertex v of the output mesh, the distance to its nearest neighbor w is at least $\frac{\text{lfs}(v)}{D_S+1}$.*

Proof: Inequality 3.2 indicates that $D_S > D_T$, so Lemma 22 shows that $\frac{\text{lfs}(v)}{r_v} \leq D_S$ for any vertex v. If v was added after w, then the distance between the two vertices is $r_v \geq \frac{\text{lfs}(v)}{D_S}$, and the theorem holds. If w was added after v, apply the lemma to w, yielding

$$|vw| \geq r_w \geq \frac{\text{lfs}(w)}{D_S}.$$

By Lemma 18, $\text{lfs}(w) + |vw| \geq \text{lfs}(v)$, so

$$|vw| \geq \frac{\text{lfs}(v) - |vw|}{D_S}.$$

It follows that $|vw| \geq \frac{\text{lfs}(v)}{D_S+1}$. ■

To give a specific example, consider triangulating a PSLG (having no acute input angles) so that no angle of the output mesh is smaller than $15°$; hence $B \doteq 1.93$. For this choice of B, $D_T \doteq 5.66$ and $D_S \doteq 9.01$. Hence, the spacing of vertices is at worst about ten times smaller than the local feature size. Away from boundaries, the spacing of vertices is at worst 6.66 [42] times smaller than the local feature size.

Figure 3.18 shows the algorithm's performance for a variety of angle bounds. Ruppert's algorithm typically terminates for angle bounds much higher than the theoretically guaranteed $20.7°$, and typically exhibits much better vertex spacing than the provable worst-case bounds imply.

Ruppert [59] uses Theorem 23 to prove the size-optimality of the meshes his algorithm generates, and his result has been improved by Scott Mitchell. Mitchell's theorem is stated below, but the proof, which is rather involved, is omitted. The *cardinality* of a triangulation is the number of triangles in the triangulation.

Theorem 24 (Mitchell [50]) *Let* $\text{lfs}_T(p)$ *be the local feature size at* p *with respect to a triangulation* T *(treating* T *as a PSLG), whereas* $\text{lfs}(p)$ *remains the local feature size at* p *with respect to the input PSLG. Suppose a triangulation* T *with smallest angle* θ *has the property that there is some constant* $k_1 \geq 1$ *such that for every point* p*,* $k_1 \text{lfs}_T(p) \geq \text{lfs}(p)$*. Then the cardinality of* T *is less than* k_2 *times the cardinality of any other triangulation of the input PSLG with smallest angle* θ*, where* $k_2 \in \mathcal{O}(k_1^2/\theta)$*.* ■

Theorem 23 can be used to show that the precondition of Theorem 24 is satisfied by meshes generated by Ruppert's algorithm. Hence, a mesh generated by Ruppert's algorithm has cardinality within a constant factor of the best possible mesh satisfying the angle bound.

3.5 Algorithm and Implementation Details

3.5.1 Segment Recovery and Nonconvex Triangulations

Although Ruppert's algorithm does not require the use of a constrained Delaunay triangulation, there is a strong argument for using a CDT to mesh a nonconvex region. By starting with a CDT of the input PSLG, an implementation may immediately remove all triangles that are not part of the region prior to applying Delaunay refinement. The sequence of events is illustrated in Figures 3.19 through 3.22. The first step is to construct the CDT of the input (Figure 3.20). The second step, which diverges from Ruppert's original algorithm, is to remove triangles from concavities and holes (Figure 3.21). The third step is to apply a Delaunay refinement algorithm (Figure 3.22).

An advantage of removing triangles before refinement is that computation is not wasted refining triangles that will eventually be deleted. A more important advantage is illustrated in Figure 3.23. If extraneous triangles remain during the refinement stage, overrefinement can occur if very small features outside the object being meshed cause the creation of small triangles inside the mesh. Ruppert [59] himself suggests solving this problem by using the constrained Delaunay triangulation and ignoring interactions that take

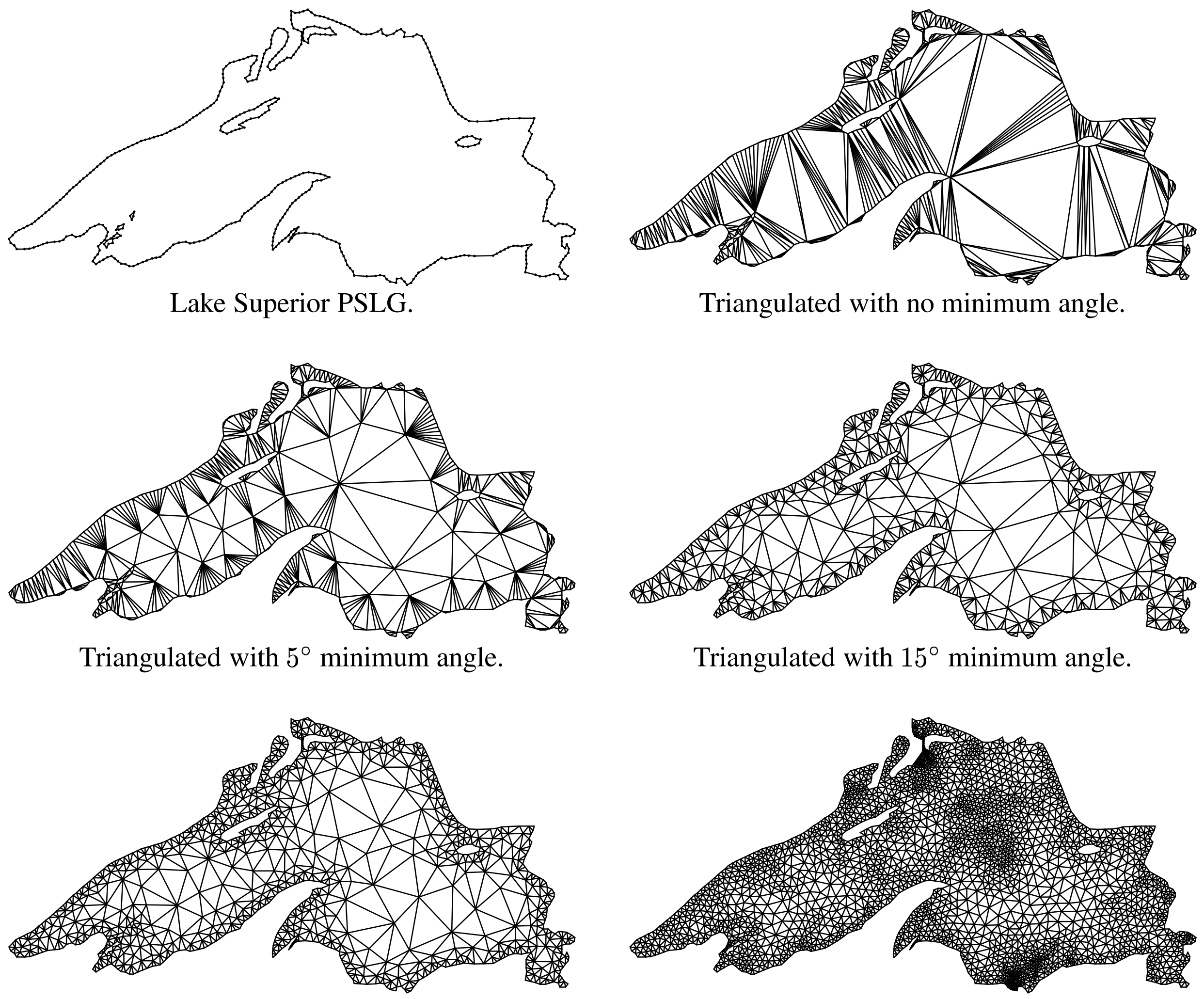

Lake Superior PSLG.
Triangulated with no minimum angle.
Triangulated with 5° minimum angle.
Triangulated with 15° minimum angle.
Triangulated with 25° minimum angle.
Triangulated with 34.2° minimum angle.

Figure 3.18: Meshes generated with Ruppert's algorithm for several different quality bounds. The algorithm does not terminate for angle bounds of 34.3° or higher on this PSLG.

place outside the region being triangulated. Early removal of triangles provides a nearly effortless way to accomplish this effect. Subsegments and subfacets that would normally be considered encroached are ignored (Figure 3.23, right), because encroached subsegments are diagnosed by noticing that they occur opposite an obtuse angle in a triangle. (See the next section for details.)

This advantage can be formalized by redefining the notion of local feature size. Let the *geodesic distance* between two points be the length of the shortest path between them that does not pass through an untriangulated region of the plane. In other words, any geodesic path must go around holes and concavities. Given a PSLG X and a point p in the triangulated region of X, define the local feature size $\text{lfs}(p)$ to be the smallest value such that there exist two points u and v within a geodesic distance of $\text{lfs}(p)$ from p that lie on nonincident features of X. This is essentially the same as the definition of local feature size given in Section 3.4.2, except that Euclidean distances are replaced with geodesic distances.

All of the proofs in this chapter can be made to work with geodesic distances, because Lemma 18 depends only upon the triangle inequality, which holds for geodesic distances as well as Euclidean distances,

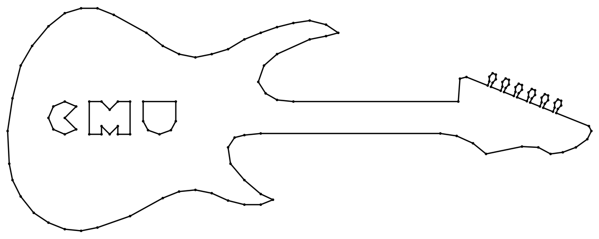


Figure 3.19: Electric guitar PSLG.

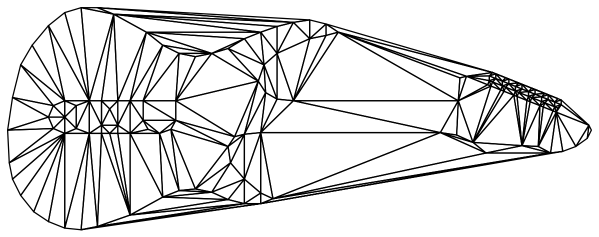


Figure 3.20: Constrained Delaunay triangulation of PSLG.

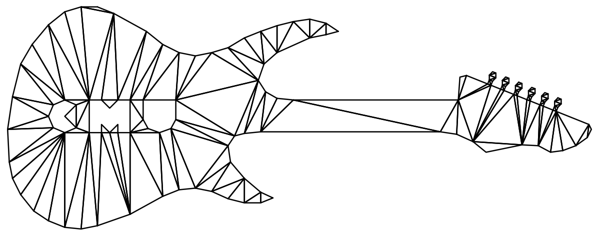


Figure 3.21: Triangles are removed from concavities and holes.

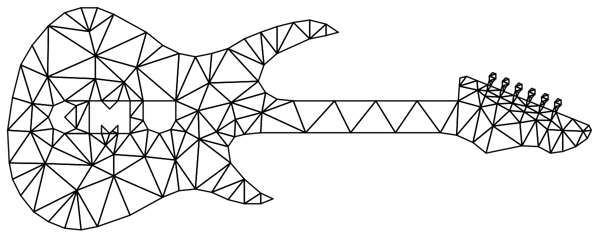


Figure 3.22: Conforming Delaunay triangulation with $20°$ minimum angle.

and because a child and its parent are never separated by an untriangulated region of the plane. The use of geodesic distances can yield improved bounds on edge lengths in some regions of the mesh, because small exterior features separations like those in Figure 3.23 do not affect local feature size within the triangulation.

These observations about geodesic distance can also be used to mesh joined two-dimensional planar surfaces embedded in three dimensions. Where surfaces meet at shared segments, small feature sizes in one surface may propagate into another. In this case, the geodesic distance between two points is the length of the shortest path between them that lies entirely within the triangulated surfaces. Two-dimensional Delaunay refinement algorithms may be applied in this context with virtually no change.

Another source of spurious small features is the edges in the boundary of the convex hull of the input,

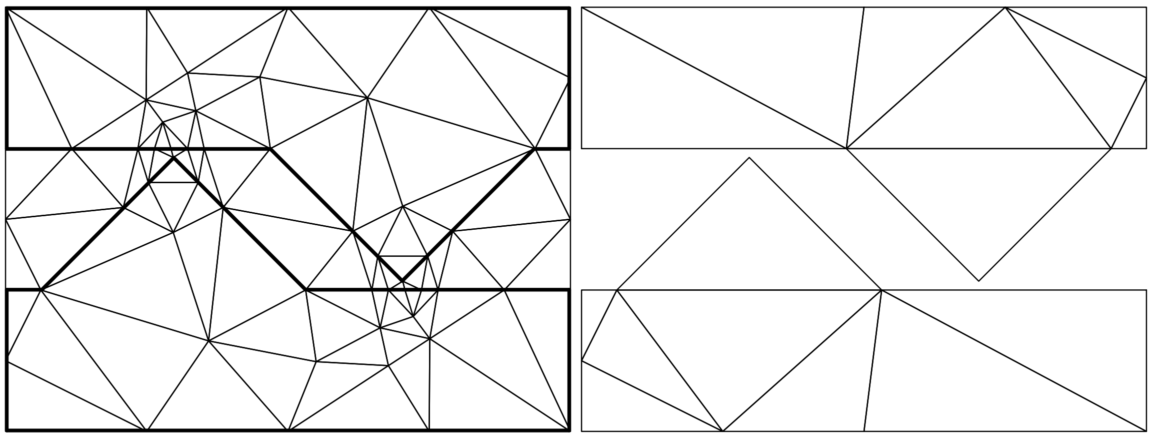


Figure 3.23: Two variations of Ruppert's Delaunay refinement algorithm with a 20° minimum angle. Left: If exterior triangles are not removed until after Delaunay refinement, overrefinement occurs. Right: Mesh created using the constrained Delaunay triangulation and early removal of triangles.

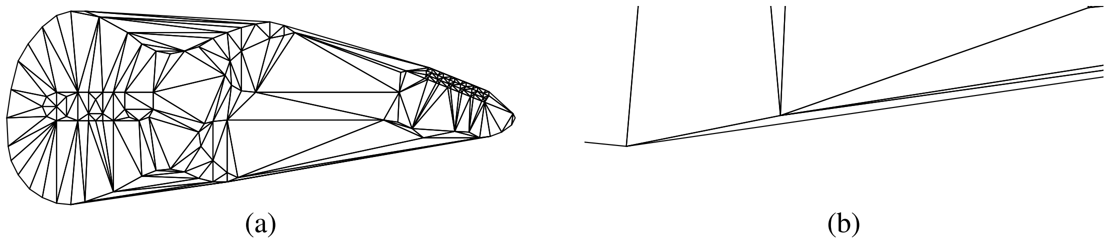


Figure 3.24: (a) Constrained Delaunay triangulation of the guitar PSLG and its convex hull. (b) Close-up of a small angle formed at the bottom of PSLG because of the convex hull.

which appear in the initial Delaunay triangulation or CDT. The Delaunay refinement algorithm requires that the triangulation it operates on be segment-bounded, so if extraneous triangles are not removed before Delaunay refinement, the edges that bound the convex hull must be treated as segments. If an input vertex lies just inside the convex hull, then the local feature size near the vertex may be artificially reduced to an arbitrarily small length. Hence, it is important that convex hull edges not be treated as subsegments if they are not part of the input PSLG. Early removal of extraneous triangles solves this problem.

Small angles present another motivation for removing triangles from holes and concavities prior to Delaunay refinement. If a small angle is present within a hole or concavity (rather than in the triangulated portion of the PSLG), the small angle should have no effect on meshing. However, if the mesh were refined prior to removing triangles from concavities and holes, unnecessarily small triangles might be produced, or Delaunay refinement might fail to terminate. This problem can appear with dastardly stealth when meshing certain nonconvex objects that do not appear to have small angles. A very small angle may be unexpectedly formed between an input segment and an edge of the convex hull, as illustrated in Figure 3.24. The user, unaware of the effect of the convex hull edge, would be mystified why the Delaunay refinement algorithm fails to terminate on what appears to be an easy PSLG. (In fact, this is how the negative result described in Section 3.7 first became evident to me.)

The main concern with removing triangles before Delaunay refinement is that general point location is difficult in a nonconvex triangulation. Fortunately, Delaunay refinement does not use the most general form of point location. Point location is used only to find the circumcenter of a triangle, which may be accomplished by walking from the centroid of the triangle toward the circumcenter. Lemma 17 guarantees

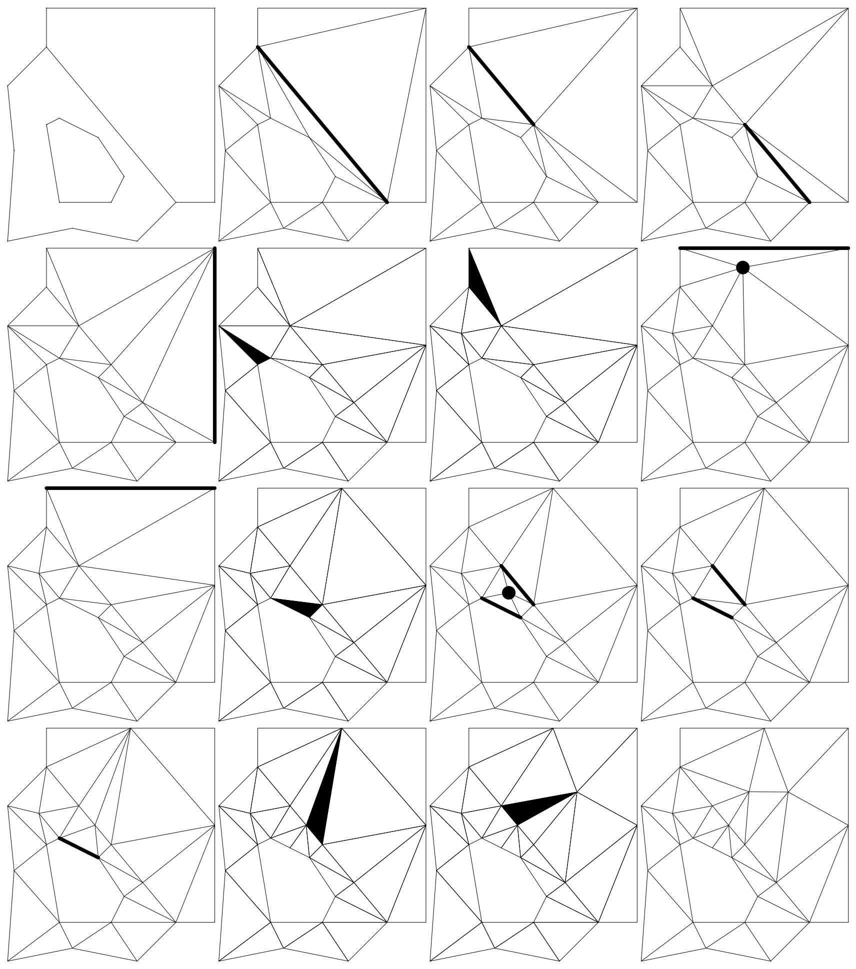


Figure 3.25: Demonstration of the refinement step. The first two images are the input PSLG and its constrained Delaunay triangulation. In each image, highlighted segments or triangles are about to be split, and highlighted vertices are rejected for insertion. The algorithm easily accommodates interior boundaries and holes.

that any circumcenter considered for insertion lies inside the mesh, but roundoff error might perturb it to just outside the mesh. Because the mesh is segment-bounded, the walk must either succeed or be foiled by an encroached entity. If the path is blocked by a subsegment, the culprit is marked as encroached, and the search may be abandoned.

3.5.2 Implementing Delaunay Refinement

The refinement step is illustrated in Figure 3.25.

A refinement algorithm maintains a queue of encroached subsegments and a queue of skinny triangles, each of which are initialized at the beginning of the refinement step and maintained throughout. Every vertex insertion may add new members to either queue. The queue of encroached subsegments rarely contains more than a few items, except at the beginning of the refinement step, when it may contain many.

Each queue is initialized by traversing a list of all subsegments or triangles in the initial triangulation. Detection of encroached subsegments is a local operation. A subsegment may be tested for encroachment

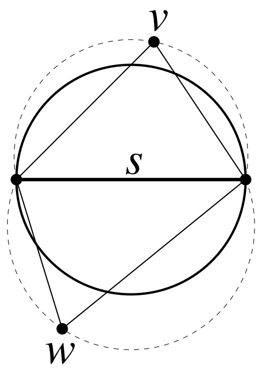


Figure 3.26: If the apices (here, v and w) of the triangles that contain a subsegment s are outside the diametral circle of s, then no vertex lies in the diametral circle of s, because the triangles are Delaunay.

by inspecting only those vertices that appear directly opposite the subsegment in a triangle. To see why, consider Figure 3.26. Both of the vertices (v and w) opposite the segment s lie outside the diametral circle of s. Because the mesh is constrained Delaunay, each triangle's circumcircle is empty (on its side of s), and therefore the diametral circle of s is empty.

After the queues are initialized, the Delaunay refinement process may cause other subsegments to become encroached. The most obvious way to test whether a new vertex encroaches upon some subsegment or subfacet is to insert it into the triangulation, then test each of the edges that appear opposite the vertex in some triangle. If a subsegment is encroached, it is inserted into the queue; if the new vertex is a triangle circumcircle, the vertex is deleted from the mesh. As the CDT vertex deletion operation can be tricky to implement efficiently, a programmer may want to write code that tests the effect of an insertion without actually changing the original mesh, or that records the sequence of edge flips so they may easily be reversed.

The subsegment encroachment test is quite simple. Let t be a triangle formed by a subsegment s and a vertex w opposite s. If the angle at w is greater than 90°, then w encroaches upon s; if the endpoints of s are u and v, this test simply checks the sign of the dot product $(u - w) \cdot (v - w)$.

I turn from the topic of detecting encroachment to the topic of managing the queue of skinny triangles (which also holds triangles that are too large). Each time a vertex is inserted or deleted, each new triangle that appears is tested, and is inserted into the queue if its quality is too poor, or its area or edge lengths too large. The number of triangles in the final mesh is determined in part by the order in which skinny triangles are split, especially when a strong quality bound is used. Figure 3.27 demonstrates how sensitive Ruppert's algorithm is to the order. For this example with a 33° minimum angle, a heap of skinny triangles indexed by their smallest angle confers a 35% reduction in mesh size over a first-in first-out (FIFO) queue. (This difference is typical for strong angle bounds, but thankfully seems to disappear for small angle bounds.) The discrepancy probably occurs because circumcenters of very skinny triangles are likely to eliminate more skinny triangles than circumcenters of mildly skinny triangles. Unfortunately, a heap is slow for large meshes, especially when bounds on the maximum triangle size force all of the triangles into the heap. Delaunay refinement usually takes $\mathcal{O}(n)$ time in practice, but the use of a heap increases the complexity to $\mathcal{O}(n \log n)$.

A solution that seems to work well in practice is to use 64 FIFO queues, each representing a different interval of circumradius-to-shortest edge ratios. Oddly, it is counterproductive in practice to order well-shaped triangles, so one queue is used for well-shaped but too-large triangles whose quality ratios are all roughly smaller than 0.8, corresponding to a minimum angle of about 39°. Triangles with larger quality ratios are partitioned among the remaining queues. When a skinny triangle is chosen for splitting, it is taken

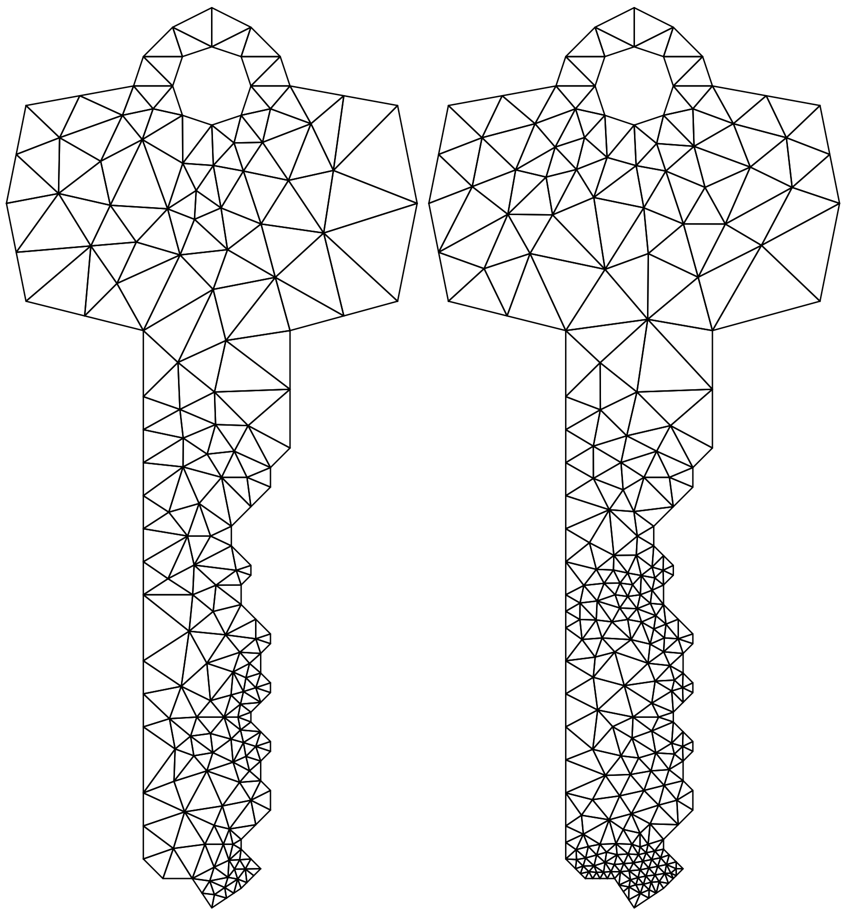


Figure 3.27: Two meshes with a 33° minimum angle. The left mesh, with 290 triangles, was formed by always splitting the worst existing triangle. The right mesh, with 450 triangles, was formed by using a first-come first-split queue of skinny triangles.

from the "worst" nonempty queue. An ordered queue of nonempty queues is maintained so that a skinny triangle may be chosen quickly. This method yields meshes comparable with those generated using a heap, but is only slightly slower than using a single queue.

3.6 Improving the Quality Bound in the Interior of the Mesh

Here, I describe a modification to Delaunay refinement that improves the quality of triangles away from the boundary of the mesh. The improvement arises easily from the discussion in Section 3.4.3. So long as no cycle having a product less than one appears in the insertion radius dataflow graph (Figure 3.17), termination is assured. The barrier to reducing the quality bound B below $\sqrt{2}$ is the fact that, when an encroached segment is split, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than its parent's. However, not every segment bisection is a worst-case example, and it is easy to explicitly measure the insertion radii of a parent and its potential progeny before deciding to take action. One can take advantage of these facts with any one of the following strategies.

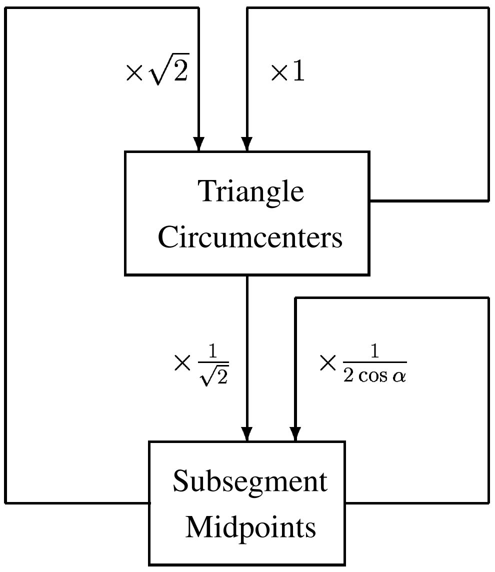


Figure 3.28: This dataflow diagram demonstrates how a simple modification to Ruppert's algorithm can improve the quality of elements away from mesh boundaries.

- Use a quality bound of $B = 1$ for triangles that are not in contact with segment interiors, and a quality bound of $B = \sqrt{2}$ for any triangle having a vertex that lies in the interior of a segment.
- Attempt to insert the circumcenter of any triangle whose circumradius-to-shortest edge ratio is larger than one. If any subsegments would be encroached, the circumcenter is rejected as usual, but the encroached subsegments are split only if the triangle's circumradius-to-shortest edge ratio is greater than $\sqrt{2}$.
- Attempt to insert the circumcenter of any triangle whose circumradius-to-shortest edge ratio is larger than one. If any subsegments would be encroached, the circumcenter is rejected as usual, and each encroached subsegment is checked to determine the insertion radius of the new vertex that might be inserted at its midpoint. The only midpoints inserted are those whose insertion radii are at least as large as the length of the shortest edge of the skinny triangle.

The first strategy is easily understood from Figure 3.28. Because segment vertices may have smaller insertion radii than free vertices, segment vertices are only allowed to father free vertices whose insertion radii are larger than their own by a factor of $\sqrt{2}$. Hence, no diminishing cycles are possible.

The other two strategies work for an even more straightforward reason: all vertices (except rejected vertices) are expressly forbidden from creating descendants having insertion radii smaller than their own. The third strategy is more aggressive than the second, as it always chooses to insert a vertex if the second strategy would do so.

The first strategy differs from the other two in its tendency to space segment vertices more closely than free vertices. The other two strategies tend to space segment vertices and free vertices somewhat more equally. The first strategy interrupts the propagation of reduced insertion radii from segment vertices to the free vertices, whereas the other two interrupt the process by which free vertices create segment vertices with smaller insertion radii. The effect of the first strategy is easily stated: upon termination, all angles are better than 20.7°, and all triangles whose vertices do not lie in segment interiors have angles of 30° or better. For

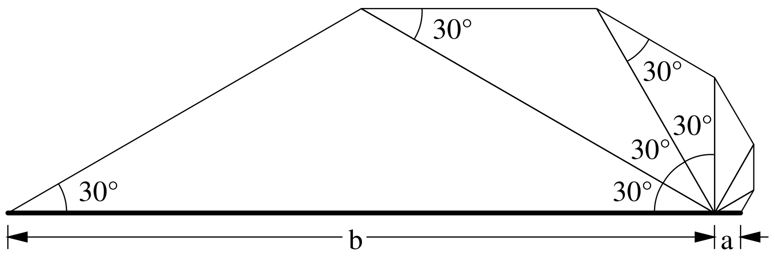


Figure 3.29: In any triangulation with no angles smaller than $30°$, the ratio b/a cannot exceed 27.

the other two strategies, the delineation between 20.7° triangles and 30° triangles is not so clear, although the former only occur near boundaries.

None of these strategies compromises good grading or size-optimality, although the bounds may be weaker. Assume that a quality bound B is applied to all triangles, and a stronger quality bound $B_I > 1$ applies in the interior of the mesh. Then Inequality 3.1 is accompanied by the inequality

$$1 + \frac{D_T}{B_I} \leq D_T,$$

which holds true when

$$\frac{B_I}{B_I - 1} \leq D_T.$$

If this bound is stricter than the bounds given for D_T in the proof of Lemma 22, then choose D_T to be $\frac{B_I}{B_I-1}$. If the first strategy is used, recalculate D_S using Inequality 3.2. If $B > \sqrt{2}$, $B_I > 1$, and $\alpha_{\min} > 60°$, there are values of D_T and D_S that satisfy the lemma.

If the second or third strategy is used, then Inequalities 3.1 and 3.2 do not apply, and a slightly more complicated analysis is needed.

3.7 A Negative Result on Quality Triangulations of PSLGs that Have Small Angles

For any angle bound $\theta > 0$, there exists a PSLG $\mathcal{P}$ such that it is not possible to triangulate $\mathcal{P}$ without creating a new corner (not present in $\mathcal{P}$) whose angle is smaller than θ. This statement applies to any triangulation algorithm, and not just those discussed in these notes. Here, I discuss why this is true.

The result holds for certain PSLGs that have an angle much smaller than θ. Of course, one must respect the PSLG; small input angles cannot be removed. However, one would like to believe that it is possible to triangulate a PSLG without creating any small angles that aren't already present in the input. Unfortunately, no algorithm can make this guarantee for all PSLGs.

The reasoning behind the result is as follows. Suppose a segment in a conforming triangulation has been split into two subsegments of lengths a and b, as illustrated in Figure 3.29. Mitchell [50] proves that if the triangulation has no angles smaller than θ, then the ratio b/a has an upper bound of $(2\cos\theta)^{180°/\theta}$. (This bound is tight if $180°/\theta$ is an integer; Figure 3.29 offers an example where the bound is obtained.) Hence any bound on the smallest angle of a triangulation imposes a limit on the gradation of triangle sizes along a segment (or anywhere in the mesh).

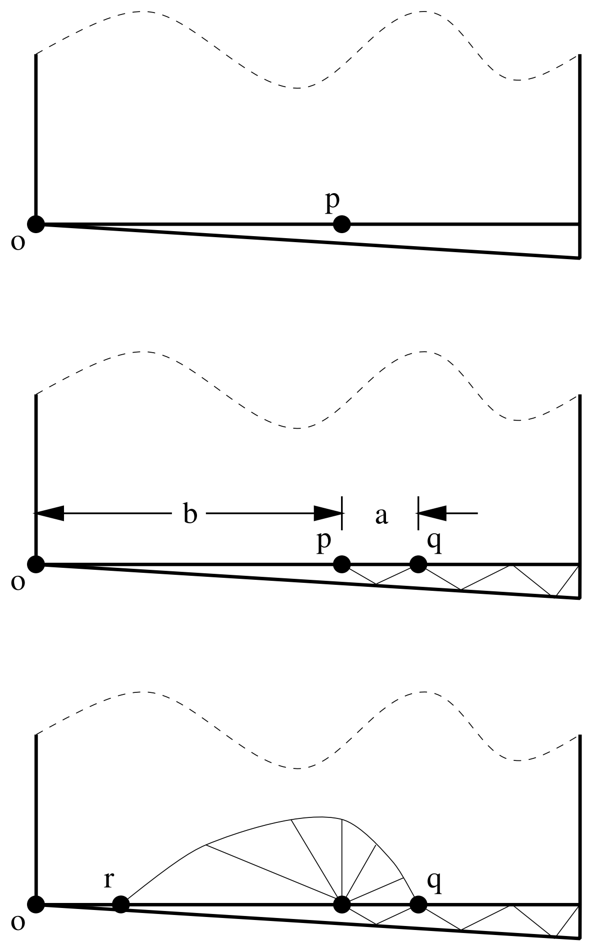


Figure 3.30: Top: A difficult PSLG with a small interior angle ϕ. Center: The vertex p and the angle constraint necessitate the insertion of the vertex q. Bottom: The vertex q and the angle constraint necessitate the insertion of the vertex r. The process repeats eternally.

A problem can arise if a small angle ϕ occurs at the intersection vertex o of two segments of a PSLG, and one of these segments is separated by a much larger angle from a third segment incident at o. Figure 3.30 (top) illustrates this circumstance. Assume that the middle segment of the three is split by a vertex p, which may be present in the input or may have been inserted to help achieve the angle constraint elsewhere in the triangulation. The insertion of p forces the narrow wedge between the first two segments to be triangulated (Figure 3.30, center), which may necessitate the insertion of a new vertex q on the segment containing p. Let $a = |pq|$ and $b = |op|$ as illustrated. If the angle bound is respected, the length a cannot be large; the ratio a/b cannot exceed

$$\frac{\sin\phi}{\sin\theta}\left(\cos(\theta+\phi)+\frac{\sin(\theta+\phi)}{\tan\theta}\right).$$

If the upper region (above the wedge) is part of the interior of the PSLG, the fan effect demonstrated in Figure 3.29 may necessitate the insertion of another vertex r between o and p (Figure 3.30, bottom); this circumstance is unavoidable if the product of the bounds on b/a and a/b given above is less than one. For an angle constraint of $\theta = 30°$, this condition occurs when ϕ is about six tenths of a degree. Unfortunately, the new vertex r creates the same conditions as the vertex p, but is closer to o; the process will cascade, eternally necessitating smaller and smaller triangles to satisfy the angle constraint. No algorithm can produce a finite

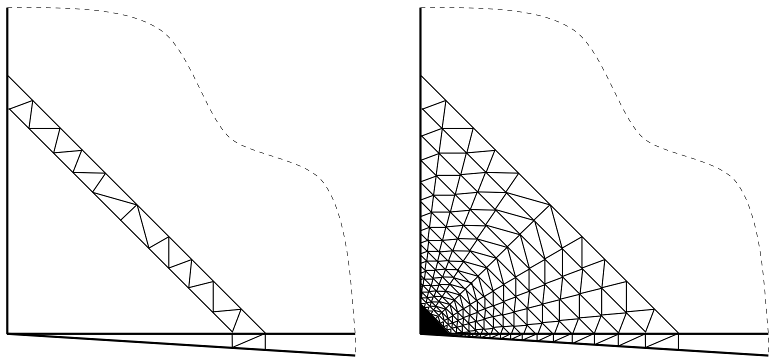


Figure 3.31: How to create a quality triangulation of infinite cardinality around the apex of a very small angle. The method employs a thin strip of well-shaped triangles about the vertex (left). Ever-smaller copies of the strip fill the gap between the vertex and the outer strip (right).

triangulation of such a PSLG without violating the angle constraint.

This bound is probably not strict. It would not be surprising if a 30° angle bound is not obtainable by any algorithm for $\phi = 1^\circ$, and Delaunay refinement often fails in practice to achieve a 30° angle bound for $\phi = 5^\circ$.

Oddly, it appears to be straightforward to triangulate this PSLG using an infinite number of well-shaped triangles. A vertex at the apex of a small angle can be shielded with a thin strip of well-shaped triangles, as Figure 3.31 illustrates. (This idea is related to Ruppert's technique of using *shield edges* [59]. However, Ruppert mistakenly claims that the region concealed behind shield edges always has a finite good-quality triangulation.) The strip is narrow enough to admit a quality triangulation at the smallest input angle. Its shape is chosen so that the angles it forms with the segments outside the shield are obtuse, and the region outside the shield can be triangulated by Delaunay refinement. The region inside the shield is triangulated by an infinite sequence of similar strips, with each successive strip smaller than the previous strip by a constant factor close to one.

3.8 Practical Handling of Small Input Angles

A practical mesh generator should not respond to small input angles by failing to terminate, even if the only alternative is to leave bad angles behind. The result of the previous section quashes all hope of finding a magic pill that will make it possible to triangulate any PSLG without introducing additional small angles. The Delaunay refinement algorithms discussed thus far will fail to terminate on PSLGs like that of Figure 3.30. Of course, any Delaunay refinement algorithm should be modified so that it does not try to split any skinny triangle that bears a small input angle. However, even this change does not help with the bad PSLGs described in the previous section, because such PSLGs always have a small angle that is removable, but another small angle invariably takes its place. How can one detect this circumstance, and ensure termination of the algorithm without unnecessarily leaving many bad angles behind?

Figure 3.32 demonstrates one of the difficulties caused by small input angles. If two incident segments have unmatched lengths, a endless cycle of mutual encroachment may produce ever-smaller subsegments incident to the apex of the small angle. This phenomenon is only observed with angles smaller than 45°.

To solve this problem, Ruppert [59] suggests "modified segment splitting using concentric circular shells." Imagine that each input vertex is encircled by concentric circles whose radii are all the powers

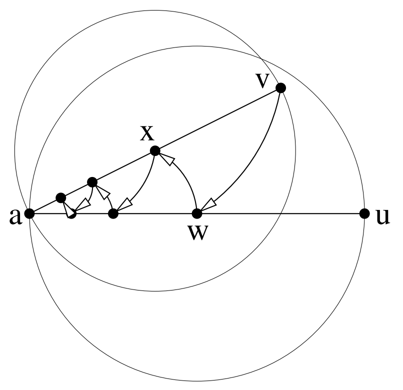


Figure 3.32: A problem caused by a small input angle. Vertex v encroaches upon au, which is split at w. Vertex w encroaches upon av, which is split at x. Vertex x encroaches upon aw, and so on.

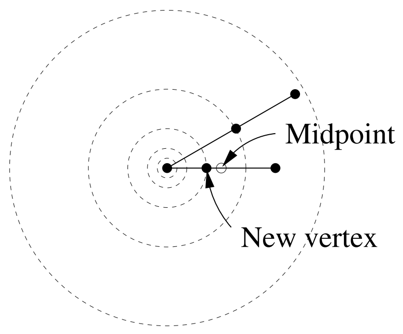


Figure 3.33: If an encroached subsegment has a shared input vertex for an endpoint, the subsegment is split at its intersection with a circular shell whose radius is a power of two.

of two, as illustrated in Figure 3.33. When an encroached subsegment has an endpoint that is an input vertex shared with another segment, the subsegment is split not at its midpoint, but at one of the circular shells, so that one of resulting subsegments has a power-of-two length. The shell that gives the best-balanced split is chosen; in the worst case, the smaller resulting subsegment is one-third the length of the split subsegment. If both endpoints are shared input vertices, choose one endpoint's shells arbitrarily. Each input segment may undergo up to two unbalanced splits, which create power-of-two subsegments at the ends of the segment. All other subsegment splits are bisections.

Concentric shell segment splitting prevents the runaway cycle of ever-smaller subsegments portrayed in Figure 3.32, because incident subsegments of equal length do not encroach upon each other. Again, it is important to modify the algorithm so that it does not attempt to split a skinny triangle that bears a small input angle, and cannot be improved.

Modified segment splitting using concentric circular shells is generally effective in practice for PSLGs that have small angles greater than $10°$, and often for smaller angles. It is always effective for polygons with holes (for reasons to be discussed shortly). As the previous section hints, difficulties are only likely to occur when a small angle is adjacent to a much larger angle. The negative result of the previous section arises not because subsegment midpoints can cause incident subsegments to be split, but because the free edge opposite a small angle is shorter than the subsegments that define the angle, as Figure 3.34 illustrates.

The two subsegments of Figure 3.34 are coupled, in the sense that if one is bisected then so is the other, because the midpoint of one encroaches upon the other. This holds true for any two segments of equal length

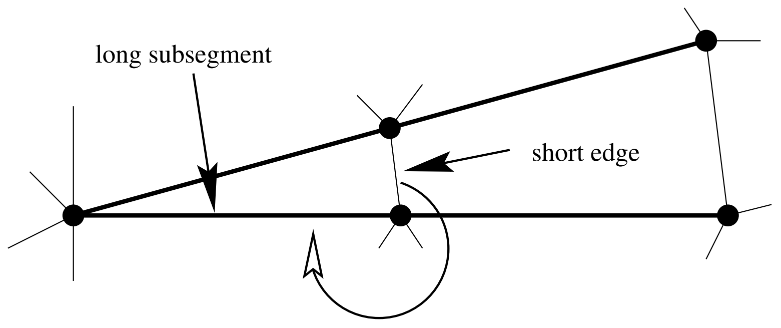


Figure 3.34: The short edge opposite a small angle can cause other short edges to be created as the algorithm attempts to remove skinny triangles. If the small insertion radii propagate around an endpoint and cause the supporting subsegments to be split, a shorter edge is created, and the cycle may repeat.

separated by less than 60°. Each time such a dual bisection occurs, a new edge is created that is shorter than the subsegments produced by the bisection; the free edge can be arbitrarily short if the angle is arbitrarily small. One of the endpoints of the free edge has a small insertion radius, though that endpoint's parent (typically the other endpoint) might have a large insertion radius. Hence, a small angle acts as an "insertion radius reducer." The new short edge will likely engender other short edges as the algorithm attempts to remove skinny triangles. If small insertion radii propagate around an endpoint of the short edge, the incident subsegments may be split again, commencing an infinite sequence of shorter and shorter edges.

If the PSLG is a polygon (possibly with polygonal holes), small insertion radii cannot propagate around the small edge, because the small edge partitions the polygon into a skinny triangle (which the algorithm does not attempt to split) and everything else. The small edge is itself flipped or penetrated only if there is an even smaller feature elsewhere in the mesh. If the small edge is thus removed, the algorithm will attempt to fix the two skinny triangles that result, thereby causing the subsegments to be split again, thus creating a new smaller edge (Figure 3.35).

For general PSLGs, how may one diagnose and cure diminishing cycles of edges? A sure-fire way to guarantee termination was hinted at in Section 3.6: never insert a vertex whose insertion radius is smaller than the insertion radius of its most recently inserted ancestor (its parent if the parent was inserted; its grandparent if the parent was rejected), unless the parent is an input vertex or lies on a nonincident segment.

This restriction is undesirably conservative for two reasons. First, if a Delaunay triangulation is desired, the restriction might prevent us from obtaining one, because segments may be left encroached. A second, more serious problem is demonstrated in Figure 3.36. Two subsegments are separated by a small input angle, and one of the two is bisected. The other subsegment is encroached, but is not bisected because the midpoint would have a small insertion radius. One unfortunate result is that the triangle bearing the small input angle also bears a large angle of almost 180°. Recall that large angles can be worse than small angles, because they jeopardize convergence and interpolation accuracy in a way that small angles do not. Another unfortunate result is that many skinny triangles may form. The triangles in the figure cannot be improved without splitting the upper subsegment.

As an alternative, I suggest the following scheme.

The Quitter: A Delaunay refinement algorithm that knows when to give up. Guaranteed to terminate.

The Quitter is based on Delaunay refinement with concentric circular shells. When a subsegment s is encroached upon by the circumcenter of a skinny triangle, a decision is made whether to split s with a

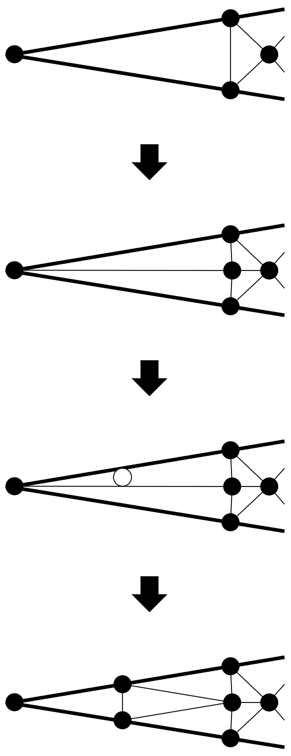


Figure 3.35: Concentric shell segment splitting ensures that polygons (with holes) can be triangulated, because it causes small angles to be clipped off. This sequence of illustrations demonstrate that if a clipped triangle's interior edge is breached, a smaller clipped triangle will result.

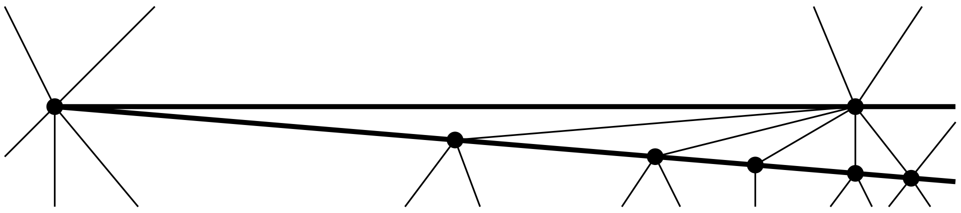


Figure 3.36: The simplest method of ensuring termination when small input angles are present has undesirable properties, including the production of large angles and many small angles.

vertex v, or to leave s whole. (In either case, the circumcenter is rejected.) The decision process is somewhat elaborate.

If neither endpoint of s bears a small input angle (less than 60°), or if both endpoints do, then s is split. Otherwise, let a be the apex of the small angle. Define the *subsegment cluster* of s to be the set of subsegments incident to a that are separated from s, or from some other member of the subsegment cluster of s, by less than 60°. Once all the subsegments of a cluster have been split to power-of-two lengths, they must all be the same length to avoid encroaching upon each other. If one is bisected, the others follow suit, as illustrated in Figure 3.37(a). If v is inserted in s it is called a *trigger vertex*, because it may trigger the splitting of all the subsegments in a cluster. (A trigger vertex is any vertex inserted into a segment that belongs to a subsegment cluster and is encroached upon by a triangle circumcenter.)

The definition of subsegment cluster does not imply that all subsegments incident to an input vertex are

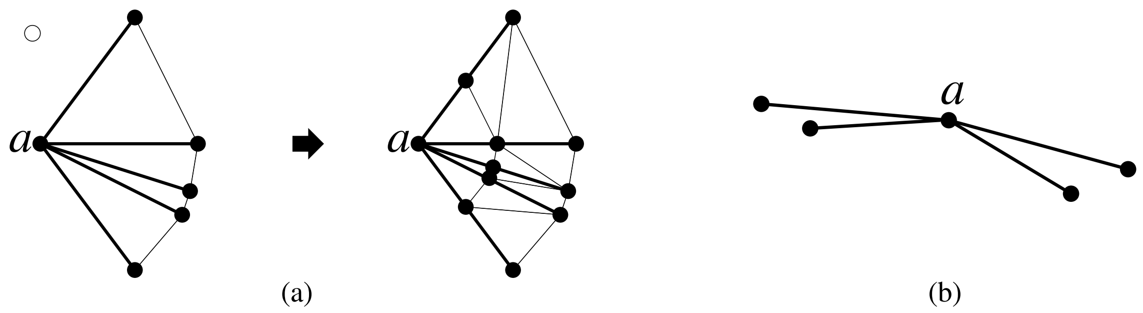


Figure 3.37: (a) Example of a subsegment cluster. If all the subsegments of a cluster have power-of-two lengths, then they all have the same length and are effectively split as a unit because of mutual encroachment. (b) Several independent subsegment clusters may share the same apex.

part of the same cluster. For instance, Figure 3.37(b) shows two independent subsegment clusters sharing one apex, separated from each other by angles of at least 60°.

To decide whether s should be split with a vertex v, the Quitter determines the insertion radius r_g of v's grandparent g (which is the parent of the encroaching circumcenter), and the minimum insertion radius $r_{\min}$ of all the midpoint vertices (including v) that will be introduced into the subsegment cluster of s if all the subsegments in the cluster having length $|s|$ or greater are split. If all the subsegments in the cluster have the same length, then $r_{\min}$ depends upon the smallest angle in the subsegment cluster.

The vertex v is inserted, splitting s, only if one or more of the following three conditions hold.

- If $r_{\min} \geq r_g$, then v is inserted.
- If one of the segments in the subsegment cluster of s has a length that is not a power of two, then v is inserted.
- If no ancestor of v also lies in the interior of the segment containing s, then v is inserted. (Endpoints of the segment do not count.)

End of description of the Quitter.

If there are no input angles smaller than 60°, the Quitter acts no differently from Ruppert's algorithm by the following reasoning. Any encroached subsegment s is the only subsegment in its cluster, and $r_{\min} = r_v$ (where v is the new vertex inserted in s). If s is precisely bisected, Theorem 20 states that the first condition ($r_{\min} \geq r_g$) always holds. If the length of s is not a power of two, s may be split unevenly, and hence the condition $r_{\min} \geq r_g$ may not be true, but the second condition above ensures that such splits are not prevented.

On the other hand, if small angles are present, and the first condition fails for some encroached segment, the third condition identifies situations in which the mesh can be improved without threatening the guarantee of termination. The third condition attempts to distinguish between the case where a segment is encroached because of small input features, and the case where a segment is encroached because it bears a small angle.

Theorem 25 *The Quitter always terminates.*

Proof sketch: Suppose for the sake of contradiction that the Quitter fails to terminate. Then there must be an infinite sequence of vertices V with the property that each vertex of V (except the first) is the child of its predecessor, and for any positive real value γ, some vertex in V has insertion radius smaller than γ. (If there is no such sequence of descendants, then there is a lower bound on the length of an edge, and the algorithm must terminate.)

Say that a vertex v has the *diminishing property* if its insertion radius r_v is less than that of all its ancestors. The sequence V contains an infinite number of vertices that have the diminishing property.

If a vertex v has the diminishing property, then v must have been inserted in a subsegment s under one of the following circumstances:

- v is not the midpoint of s, because s was split using concentric circular shells.
- s is encroached upon by an input vertex or a vertex lying on a segment not incident to s.
- s is encroached upon by a vertex p that lies on an segment incident to s at an angle less than 60°.

Only a finite number of vertices can be inserted under the first two circumstances. The first circumstance can occur only twice for each input segment: once for each end of the segment. Only a finite number of vertex insertions of the second type are possible as well, because any subsegment shorter than lfs_{min} cannot be encroached upon by a nonincident feature. Hence, V must contain an infinite number of vertices inserted under the third circumstance.

It follows that V must contain an infinite number of trigger vertices. Why? V cannot have arbitrarily long contiguous sequences of vertices inserted under the third circumstance above. Power-of-two segment splitting prevents a cluster of incident segments from engaging in a chain reaction of ever-diminishing mutual encroachment. Specifically, let 2^x be the largest power of two less than or equal to the length of the shortest subsegment in the cluster. No subsegment of the cluster can be split to a length shorter than 2^{x-1} through the mechanism of encroachment alone. The edges opposite the apex of the cluster may be much shorter than 2^{x-1}, but some other mechanism is needed to explain how the sequence V can contain insertion radii even shorter than these edges. The only such mechanism that can be employed an infinite number of times is the attempted splitting of a skinny triangle. Therefore, V contains an infinite number of trigger vertices.

One of the Quitter's rules is that a trigger vertex may only be inserted if it has no ancestor in the interior of the same segment. Hence, V may only contain one trigger vertex for each input segment. It follows that the number of trigger vertices in V is finite, a contradiction. ■

The Quitter eliminates all encroached subsegments, so if diametral circles are used, there is no danger that a segment will fail to appear in the final mesh (if subsegments are not locked), or that the final mesh will not be Delaunay (if subsegments are locked). Because subsegments are not encroached, an angle near 180° cannot appear immediately opposite a subsegment (as in Figure 3.36), although large angles can appear near subsegment clusters. Skinny triangles in the final mesh occur only near input angles less than (in practice, *much* less than) 60°.

The Quitter has the unfortunate characteristic that it demands more memory than would otherwise be necessary, because each mesh vertex must store its insertion radius and a pointer to its parent (or, if its parent was rejected, its grandparent). Hence, I suggest possible modifications to avoid these requirements.

The Quitter needs to know the insertion radius of a vertex only when a trigger vertex v is being considered for insertion. It is straightforward to compute the insertion radii of v and the other vertices that will be inserted into the cluster. However, the insertion radius of the grandparent of the trigger vertex is used

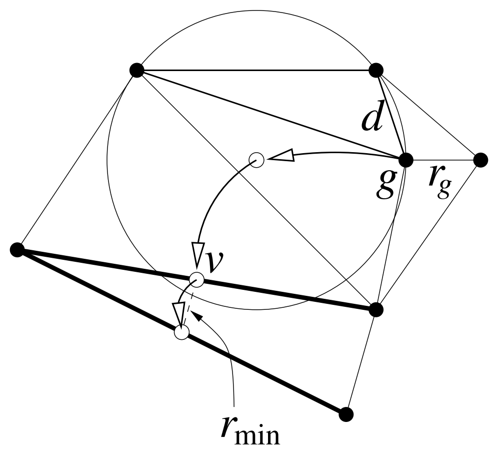


Figure 3.38: The length d of the shortest edge of a skinny triangle is an upper bound on the insertion radius r_g of the most recently inserted endpoint of that edge.

for comparison, and may not be directly computable from the mesh, because other vertices may have been inserted near g since g was inserted. Nevertheless, it is reasonable to approximate r_g by using the length d of the shortest edge of the skinny triangle whose circumcenter is v's parent, illustrated in Figure 3.38. The length d is an upper bound on r_g, so its use will not jeopardize the Quitter's termination guarantee; the modified algorithm is strictly more conservative in its decision of whether to insert v. With this modification, there is no need to store the insertion radius of each vertex.

The only apparent way to avoid storing a pointer from each vertex to its nearest inserted ancestor is to eliminate the provision that a trigger vertex may be inserted if none of its ancestors lies in the same segment. The possible disadvantage is that a small nearby input feature might fail to cause the segment to be split even though it ought to have the privilege, and thus skinny triangles will unnecessarily remain in the mesh.

Chapter 4

Three-Dimensional Delaunay Refinement Algorithms

Herein I discuss Delaunay refinement algorithms for generating tetrahedral meshes. The generalization of Chew's and Ruppert's ideas to three dimensions is relatively straightforward, albeit not without complications. The basic operation is still the Delaunay insertion of a vertex at the circumcenter of a simplex, and the result is still a mesh whose elements have bounded circumradius-to-shortest edge ratios.

In three dimensions, however, such a mesh is not entirely adequate for the needs of interpolation or finite element methods. As Dey, Bajaj, and Sugihara [21] illustrate, most tetrahedra with poor angles have circumcircles much larger than their shortest edges, including the needle, wedge, and cap illustrated in Figure 4.1. But there is one type called a *sliver* or *kite* tetrahedron that does not.

The canonical sliver is formed by arranging four vertices, equally spaced, around the equator of a sphere, then perturbing one of the vertices slightly off the equator, as Figure 4.2 illustrates. A sliver can have

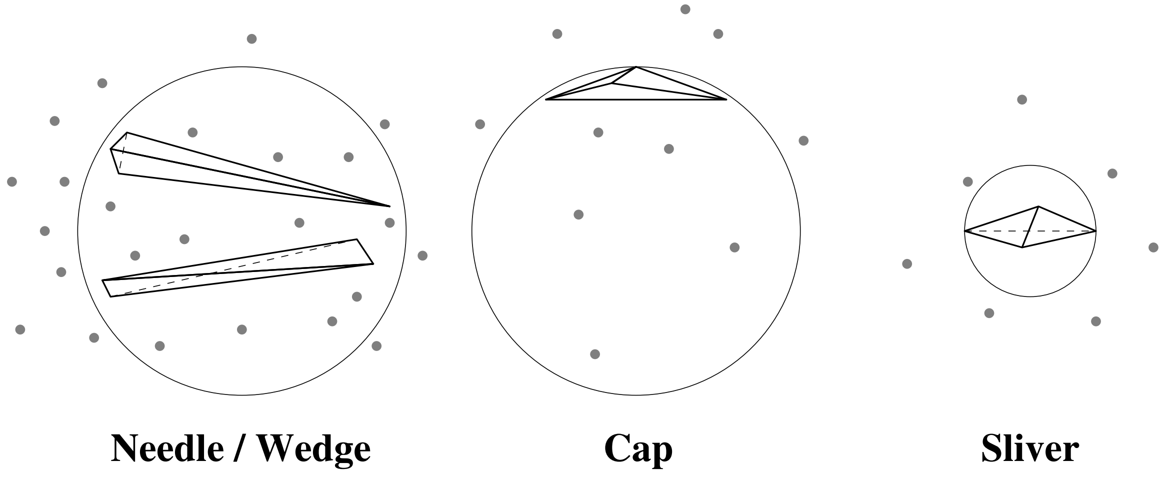


Figure 4.1: Tetrahedra with poor angles. Needles and wedges have edges of greatly disparate length; caps have a large solid angle; slivers have neither, and can have good circumradius-to-shortest edge ratios. Needles, wedges, and caps have circumspheres significantly larger than their shortest edges, and are thus eliminated when additional vertices are inserted with a spacing proportional to the shortest edge. A sliver can easily survive in a Delaunay tetrahedralization of uniformly spaced vertices.

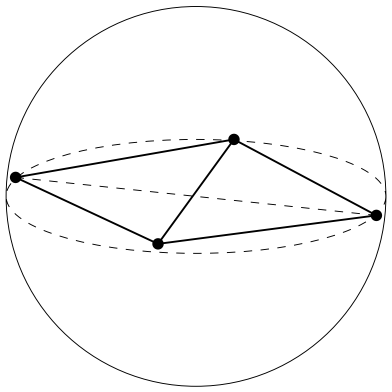


Figure 4.2: A sliver tetrahedron.

an admirable circumradius-to-shortest edge ratio (as low as $\frac{1}{\sqrt{2}}$!) yet be considered awful by most other measures, because its volume and its shortest altitude can be arbitrarily close to zero, and its dihedral angles can be arbitrarily close to 0° and 180°. Slivers have no two-dimensional analogue; any triangle with a small circumradius-to-shortest edge ratio is considered "well-shaped" by the usual standards of finite element methods and interpolation.

Slivers often survive Delaunay-based tetrahedral mesh generation methods because their small circumradii minimize the likelihood of vertices being inserted in their circumspheres (as Figure 4.1 illustrates). A perfectly flat sliver whose edge lengths are $\mathrm{lfs}_{\mathrm{min}}$ about the equator and $\sqrt{2}\mathrm{lfs}_{\mathrm{min}}$ across the diagonals is guaranteed to survive any Delaunay refinement method that does not introduce edges smaller than $\mathrm{lfs}_{\mathrm{min}}$, because every point in the interior of its circumsphere is a distance less than $\mathrm{lfs}_{\mathrm{min}}$ from one of its vertices; no vertex can be inserted inside the sphere.

Despite slivers, Delaunay refinement methods are valuable for generating three-dimensional meshes. Slivers having good circumradius-to-shortest edge ratios typically arise in small numbers in practice. As Section 4.4 will demonstrate, the worst slivers can often be removed by Delaunay refinement, even if there is no theoretical guarantee. Meshes with bounds on the circumradius-to-shortest edge ratios of their tetrahedra are an excellent starting point for mesh smoothing and optimization methods that remove slivers and improve the quality of an existing mesh (see Section 2.2.4). The most notable of these is the sliver exudation algorithm of Cheng, Dey, Edelsbrunner, Facello, and Teng [12], which is based on weighted Delaunay triangulations. Even if slivers are not removed, the Voronoi dual of a tetrahedralization with bounded circumradius-to-shortest edge ratios has nicely rounded cells, and is sometimes ideal for use in the control volume method [48].

In this chapter, I present a three-dimensional generalization of Ruppert's algorithm that generates tetrahedralizations whose tetrahedra have circumradius-to-shortest edge ratios no greater than the bound $B = \sqrt{2} \doteq 1.41$. If B is relaxed to be greater than two, then good grading can also be proven. Size-optimality, however, cannot be proven.

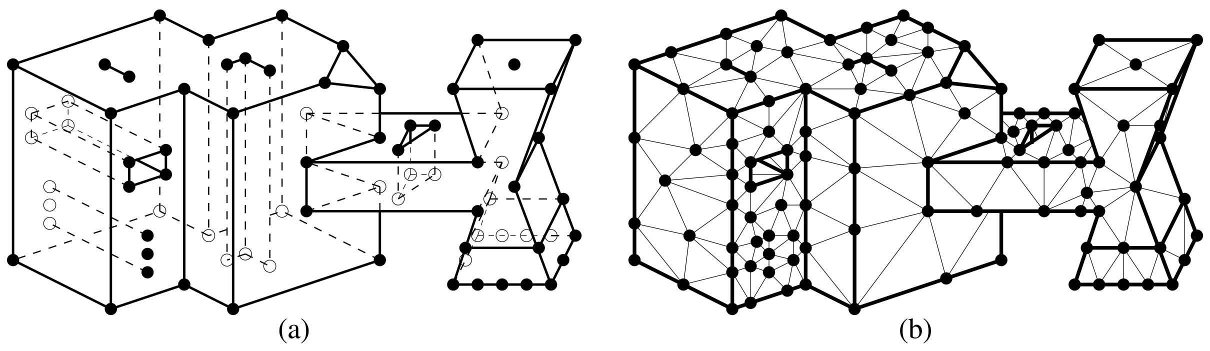


Figure 4.3: (a) Any facet of a PLC may contain holes, slits, and vertices. (b) When a PLC is tetrahedralized, each facet of the PLC is partitioned into triangular subfacets, which respect the holes, slits, and vertices.

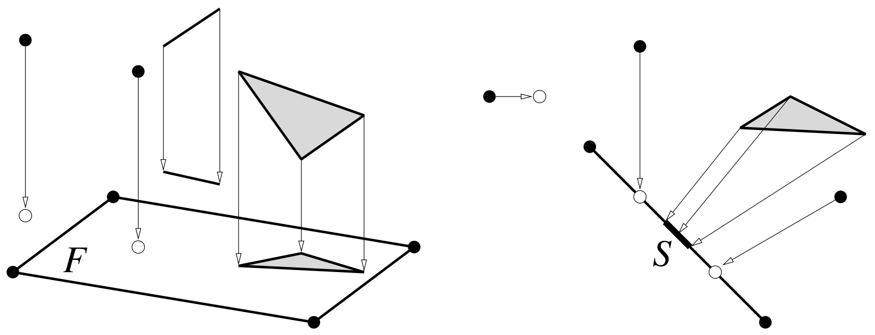


Figure 4.4: The orthogonal projections of points and sets of points onto facets and segments.

4.1 Definitions

Tetrahedral mesh generation necessarily divides each facet of a PLC, like that depicted in Figure 4.3(a), into triangular faces, as illustrated in Figure 4.3(b). Just as the triangulation edges that comprise a segment are called subsegments, the triangular faces that comprise a facet are called *subfacets*. The bold edges in Figure 4.3(b) are subsegments; other edges are not. All of the triangular faces visible in Figure 4.3(b) are subfacets, but most of the faces in the interior of the tetrahedralization are not.

Frequently in this chapter, I use the notion of the *orthogonal projection* of a geometric entity onto a line or plane. Given a facet or subfacet F and a point p, the orthogonal projection $\text{proj}_F(p)$ of p onto F is the point that is coplanar with F and lies in the line that passes through p orthogonally to F, as illustrated in Figure 4.4. The projection exists whether or not it falls in F.

Similarly, the orthogonal projection $\text{proj}_S(p)$ of p onto a segment or subsegment S is the point that is collinear with S and lies in the plane through p orthogonal to S.

Sets of points, as well as points, may be projected. If F and G are facets, then $\text{proj}_F(G)$ is the set $\{\text{proj}_F(p) : p \in G\}$.

4.2 A Three-Dimensional Delaunay Refinement Algorithm

In this section, I describe a three-dimensional Delaunay refinement algorithm that produces well-graded tetrahedral meshes satisfying any circumradius-to-shortest edge ratio bound greater than two. Miller, Talmor, Teng, Walkington, and Wang [49] have developed a related (but less effective) algorithm.

4.2.1 Description of the Algorithm

Three-dimensional Delaunay refinement takes a *facet-bounded* PLC as its input. Tetrahedralized and untetrahedralized regions of space must be separated by facets so that, in the final mesh, any triangular face not shared by two tetrahedra is a subfacet.

Many approaches to tetrahedral mesh generation permanently triangulate the input facets as a separate step prior to tetrahedralizing the interior of a region. The problem with this approach is that these independent facet triangulations may not be collectively ideal for forming a good tetrahedralization. For instance, a feature that lies near a facet (but not necessarily in the plane of the facet) may necessitate the use of smaller subfacets near that feature. The present algorithm uses another approach, wherein facet triangulations are refined in conjunction with the tetrahedralization. The tetrahedralization process is not beholden to poor decisions made earlier.

Any vertex inserted into a segment or facet during Delaunay refinement remains there permanently. However, keep in mind that the edges that partition a facet into subfacets are *not* permanent, are *not* treated like subsegments, and are subject to flipping (within the facet) according to the Delaunay criterion.

The algorithm's first step is to construct a Delaunay tetrahedralization of the input vertices. Some input segments and facets might be missing (or partly missing) from this mesh. As in two dimensions, the tetrahedralization is refined by inserting additional vertices into the mesh, using an incremental Delaunay tetrahedralization algorithm such as the Bowyer/Watson algorithm [8, 70] or three-dimensional flipping [38, 55], until all segments and facets are recovered and all constraints on tetrahedron quality and size are met. Vertex insertion is governed by three rules.

- The *diametral sphere* of a subsegment is the (unique) smallest sphere that contains the subsegment. A subsegment is encroached if a vertex other than its endpoints lies inside or on its diametral sphere. (This definition of encroachment is slightly stronger than that used by Ruppert's algorithm, to ensure that all unencroached subsegments are strongly Delaunay. This makes it possible to form a CDT, and also strengthens an upcoming result called the Projection Lemma.) A subsegment may be encroached whether or not it actually appears as an edge of the tetrahedralization. If a subsegment is missing from the tetrahedralization, it is not strongly Delaunay and thus must be encroached. Any encroached subsegment that arises is immediately split into two subsegments by inserting a vertex at its midpoint. See Figure 4.5(a).

- The *equatorial sphere* of a triangular subfacet is the (unique) smallest sphere that passes through the three vertices of the subfacet. (The *equator* of an equatorial sphere is the unique circle that passes through the same three vertices.) A subfacet is encroached if a non-coplanar vertex lies inside or on its equatorial sphere. If a subfacet is missing from the tetrahedralization, and it is not covered by other faces that share the same circumcircle, then it is encroached. (The question of what subfacets should not be missing from the tetrahedralization will be considered shortly.) Each encroached subfacet is normally split by inserting a vertex at its circumcenter; see Figure 4.5(b). However, if the new vertex would encroach upon any subsegment, it is not inserted; instead, all the subsegments it would encroach upon are split.

- A tetrahedron is said to be *skinny* if its circumradius-to-shortest edge ratio is larger than some bound B. (By this definition, not all slivers are considered skinny.) Each skinny tetrahedron is normally split

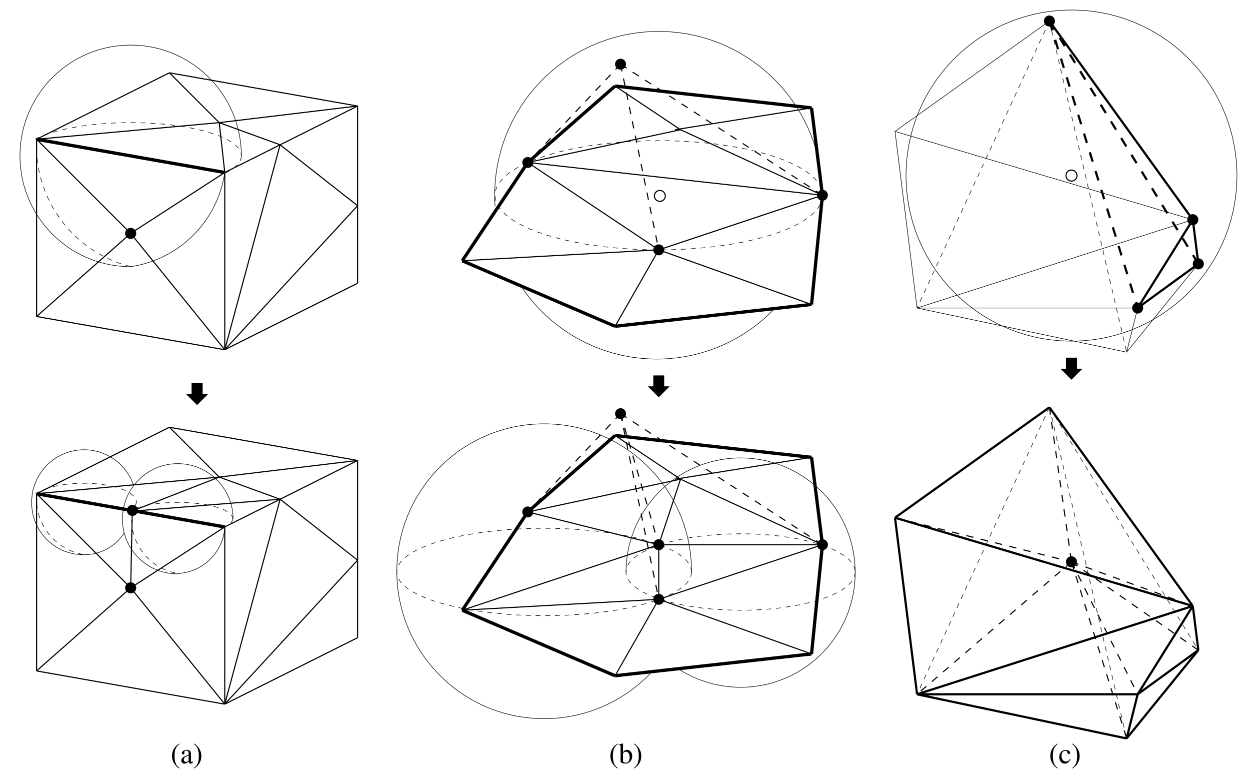


Figure 4.5: Three operations for three-dimensional Delaunay refinement. (a) Splitting an encroached subsegment. Dotted arcs indicate where diametral spheres intersect faces. The original subsegment is encroached because there is a vertex in its diametral sphere. In this example, the two subsegments created by bisecting the original subsegment are not encroached. (b) Splitting an encroached subfacet. The triangular faces shown are subfacets of a larger facet, with tetrahedra (not shown) atop them. A vertex in the equatorial sphere of a subfacet causes a vertex to be inserted at its circumcenter. In this example, all equatorial spheres (included the two illustrated) are empty after the split. (c) Splitting a skinny tetrahedron. A vertex is inserted at its circumcenter.

by inserting a vertex at its circumcenter, thus eliminating the tetrahedron; see Figure 4.5(c). However, if the new vertex would encroach upon any subsegment or subfacet, then it is not inserted; instead, all the subsegments it would encroach upon are split. If the skinny tetrahedron is not eliminated as a result, then all the subfacets its circumcenter would encroach upon are split. (A subtle point is that, if the tetrahedron is eliminated by subsegment splitting, the algorithm should not split any subfacets that appear during subsegment splitting, or the bounds proven later will not be valid. Lazy programmers beware.)

Encroached subsegments are given priority over encroached subfacets, which have priority over skinny tetrahedra. These encroachment rules are intended to recover missing segments and facets, and to ensure that all vertex insertions are valid. Because all facets are segment-bounded, Lemma 17 shows that if there are no encroached subsegments, then each subfacet circumcenter lies in the containing facet. One can also show (with a similar proof) that if there are no encroached subfacets, then each tetrahedron circumcenter lies in the mesh.

Missing subsegments are recovered by stitching, just as in the two-dimensional algorithm. When no encroached subsegment remains, missing facets are recovered in an analogous manner. The main complication is that if a facet is missing from the mesh, it is difficult to say what its subfacets are. With segments

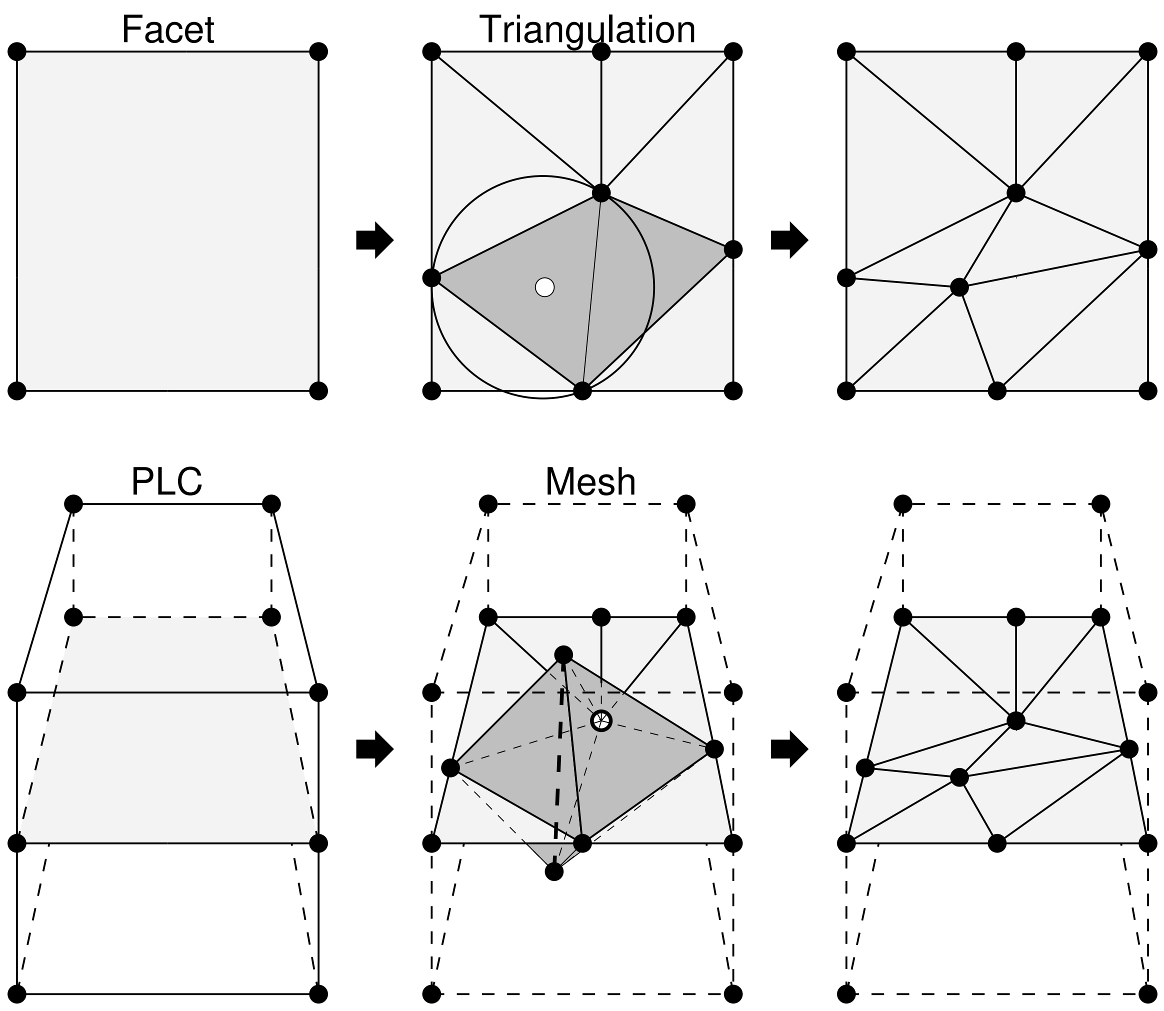


Figure 4.6: The top illustrations depict a rectangular facet and its triangulation. The bottom illustrations depict the facet's position as an interior boundary of a PLC, and its progress as it is recovered. Most of the vertices and tetrahedra of the mesh are omitted for clarity. The facet triangulation and the tetrahedralization are maintained separately. Shaded triangular subfacets in the facet triangulation (top center) are missing from the tetrahedralization (bottom center). The bold dashed line (bottom center) represents a tetrahedralization edge that passes through the facet. A missing subfacet is recovered by inserting a vertex at its circumcenter (top right and bottom right). The vertex is independently inserted into both the triangulation and the tetrahedralization.

there is no such problem; if a segment is missing from the mesh, and a vertex is inserted at its midpoint, one knows unambiguously where the two resulting subsegments are. But how may we identify subfacets that do not yet exist?

The solution is straightforward. For each facet, it is necessary to maintain a two-dimensional Delaunay triangulation of its vertices, independently from the tetrahedralization in which we hope its subfacets will eventually appear. By comparing the triangles of a facet's triangulation against the faces of the tetrahedralization, one can identify subfacets that need to be recovered. For each triangular subfacet in a facet triangulation, look for a matching face in the tetrahedralization; if the latter is missing, insert a vertex at the circumcenter of the subfacet (subject to rejection if subsegments are encroached), as illustrated in Figure 4.6. The new vertex is independently inserted into both the facet triangulation and the tetrahedralization.

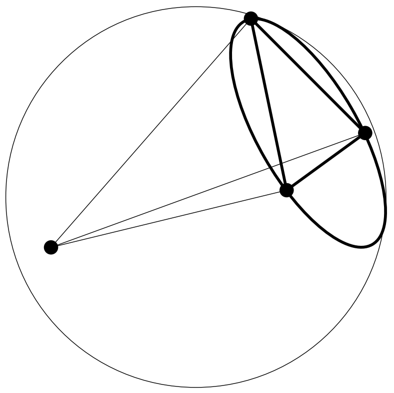


Figure 4.7: If a tetrahedron is Delaunay, the circumcircle of each of its faces is empty, because each face's circumcircle is a cross-section of the tetrahedron's circumsphere.

Similarly, the midpoint of an encroached subsegment is independently inserted into the tetrahedralization and into *each* facet triangulation that contains the subsegment.

In essence, Ruppert's algorithm (and the present algorithm) uses the same procedure to recover segments. However, the process of forming a one-dimensional triangulation is so simple that it passes unnoticed.

Which vertices of the tetrahedralization need to be considered in a facet triangulation? It is a fact, albeit somewhat nonintuitive, that if a facet appears in a Delaunay tetrahedralization as a union of faces, then the triangulation of the facet is determined solely by the vertices of the tetrahedralization that lie in the plane of the facet. If a vertex lies near a facet, but is not coplanar with the facet, it may cause a subfacet to be missing (as in Figure 4.6, bottom center), but it cannot otherwise affect the shape of the triangulation. Why? Suppose a subfacet of a facet appears in the tetrahedralization. Then the subfacet must be a face of a Delaunay tetrahedron. The subfacet's circumcircle is empty, because its circumcircle is a cross-section of the tetrahedron's empty circumsphere, as illustrated in Figure 4.7. Therefore, if a facet appears as a union of faces in a Delaunay tetrahedralization, then those faces form a two-dimensional Delaunay triangulation of the facet. Because the Delaunay triangulation is unique (except in nondegenerate cases), vertices that do not lie in the plane of the facet have no effect on how the facet is triangulated.

Furthermore, because each facet is segment-bounded, and segments are recovered (in the tetrahedralization) before facets, each facet triangulation can safely ignore vertices that lie outside the facet (coplanar though they may be). A triangulation need only take into account the segments and vertices in the facet. The requirements set forth in Section 2.1.4 ensure that all of the vertices and segments of a facet must be explicitly identified in the input PLC. The only additional vertices to be considered are those that were inserted in segments to help recover those segments and other facets. The algorithm maintains a list of the vertices on each segment, ready to be called upon when a facet triangulation is initially formed.

Unfortunately, if a facet's Delaunay triangulation is not unique because of cocircularity degeneracies, then the facet might be represented in the tetrahedralization by faces that do not match the independent facet triangulation, as Figure 4.8 illustrates. (If exact arithmetic is not used, nearly-degenerate cases may team up with floating-point roundoff error to make this circumstance more common.) An implementation must detect these cases and correct the triangulation so that it matches the tetrahedralization. (It is not always possible to force the tetrahedralization to match the triangulation.)

To appreciate the advantages of this facet recovery method, compare it with the most popular method [35, 73, 56]. In many tetrahedral mesh generators, facets are inserted by identifying points where the edges of the

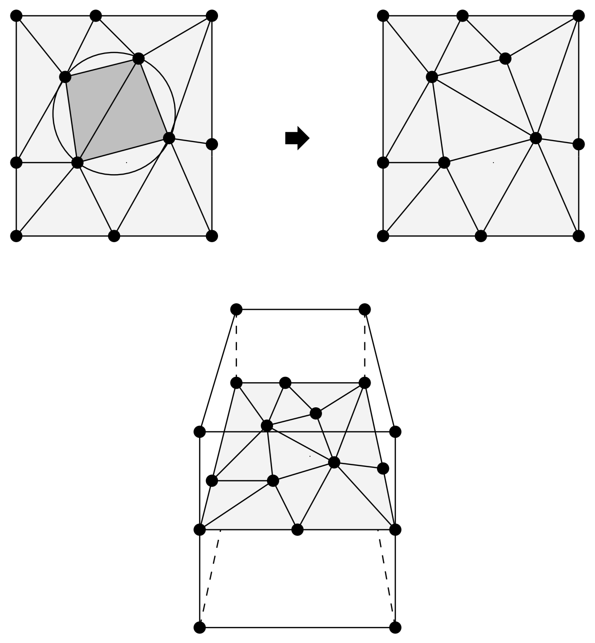


Figure 4.8: A facet triangulation and a tetrahedralization may disagree due to cocircular vertices. This occurrence should be diagnosed and fixed as shown here.

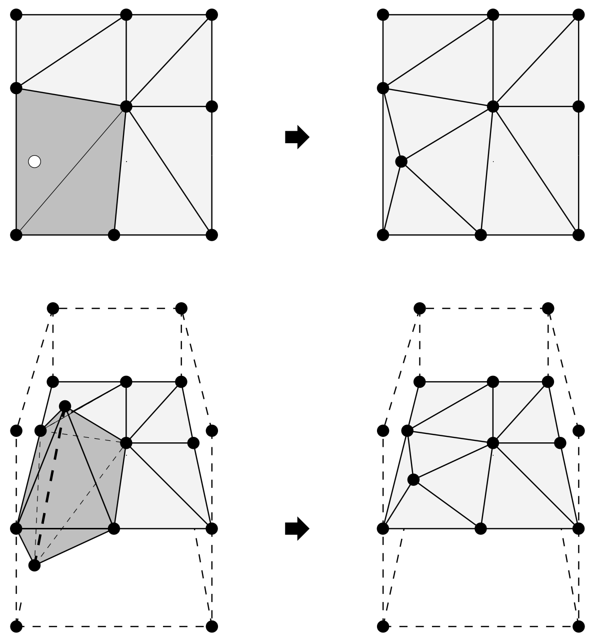


Figure 4.9: One may recover a missing facet by inserting vertices at the intersections of the facet with edges of the tetrahedralization, but this method might create arbitrarily small features by placing vertices close to segments.

tetrahedralization intersect a missing facet, and inserting vertices at these points. The perils of so doing are illustrated in Figure 4.9. In the illustration, a vertex is inserted where a tetrahedralization edge (bold dashed line) intersects the facet. Unfortunately, the edge intersects the facet near one of the bounding segments of the facet, and the new vertex creates a feature that may be arbitrarily small. Afterward, the only alternatives

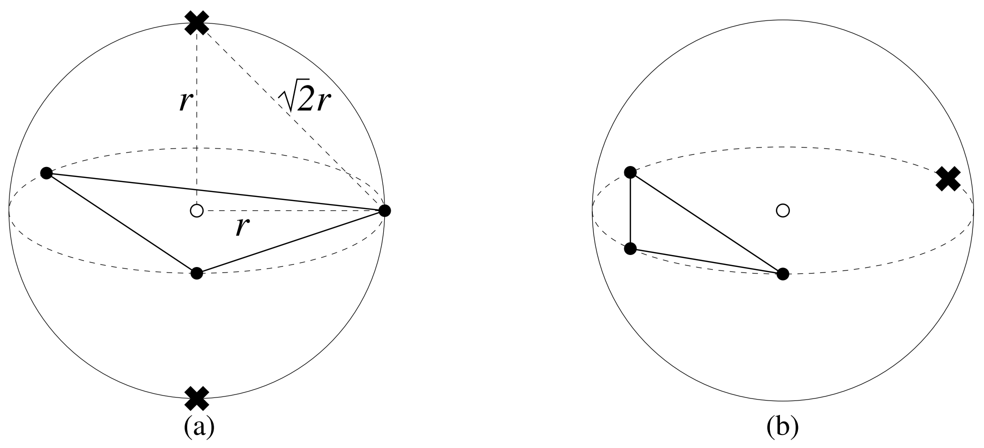


Figure 4.10: The relationship between the insertion radii of the circumcenter of an encroached subfacet and the encroaching vertex. Crosses identify the location of an encroaching vertex having maximum distance from the nearest subfacet vertex. (a) If the encroached subfacet contains its own circumcenter, the encroaching vertex is no further from the nearest vertex of the subfacet than $\sqrt{2}$ times the circumradius of the subfacet. (b) If the encroached subfacet does not contain its own circumcenter, the encroaching vertex may be further away.

are to refine the tetrahedra near the new vertex to a small size, or to move or remove the vertex. Some mesh generators cope with this problem by smoothing the vertices on each facet after the facet is competely inserted.

The encroachment-based facet recovery method does not insert such vertices at all. A vertex considered for insertion too close to a segment is rejected, and a subsegment is split instead. This would not necessarily be true if edge-facet intersections were considered for insertion, because such an intersection may be near a vertex lying on the segment, and thus fail to encroach upon any subsegments. Subfacet circumcenters are better choices because they are far from the nearest vertices, and cannot create a new small feature without encroaching upon a subsegment.

Of course, an even better idea is to form a conforming CDT of the input PLC as soon as all the segments have been recovered by stitching, thereby recovering the facets without inserting additional vertices. This measure helps to mitigate (but not eliminate) the unwanted effects of small exterior feature sizes, discussed in Section 3.5.1. For the purposes of analysis, however, it is instructive to consider the variant of the algorithm that uses unconstrained Delaunay triangulations.

When no encroached subsegment or subfacet remains, every input segment and facet is represented by a union of edges or faces of the mesh. The first time the mesh reaches this state, all exterior tetrahedra (lying in the convex hull of the input vertices, but outside the region enclosed by the facet-bounded PLC) should be removed prior to splitting any skinny tetrahedra. This measure prevents the problems described in Section 3.5.1 that can arise if superfluous skinny tetrahedra are split, such as overrefinement and failure to terminate because of exterior small angles and spurious small angles formed between the PLC and its convex hull.

One further amendment to the algorithm is necessary to obtain the best possible bound on the circumradius-to-shortest edge ratios of the tetrahedra. It would be nice to prove, in the manner of Lemma 19, that whenever an encroached subfacet is split at its circumcenter, the insertion radius of the newly inserted vertex is no worse than $\sqrt{2}$ times smaller than the insertion radius of its parent. Unfortunately, this is not true for the algorithm described above.

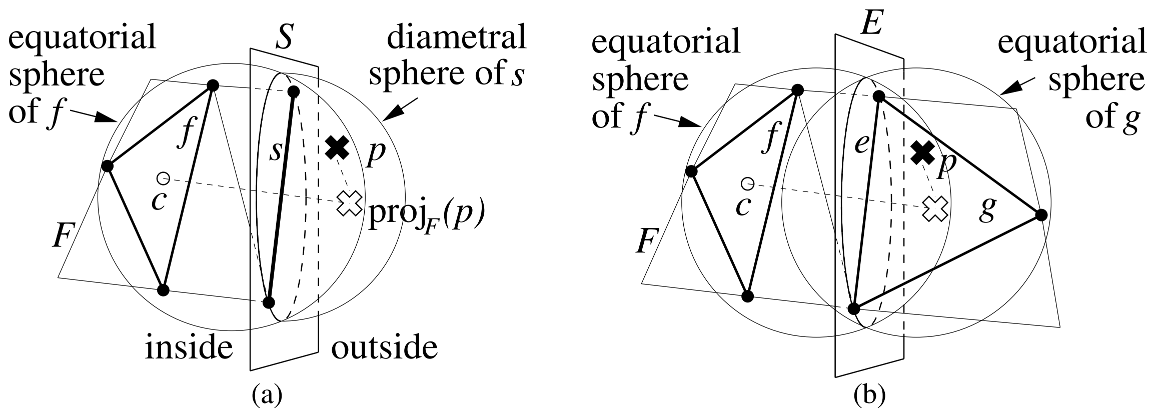


Figure 4.11: Two properties of encroached Delaunay subfacets. (a) If a vertex p encroaches upon a Delaunay subfacet f of a facet F, but its projection into the plane containing F lies outside F, then p encroaches upon some subsegment s of F as well. (b) If a vertex p encroaches upon a subfacet f of a Delaunay triangulated facet F, but does not encroach upon any subsegment of F, then p encroaches upon the subfacet(s) g of F that contains $\text{proj}_F(p)$.

Consider the two examples of Figure 4.10. If a subfacet that contains its own circumcenter is encroached, then the distance between the encroaching vertex and the nearest vertex of the subfacet is no more than $\sqrt{2}$ times the circumradius of the subfacet. This distance is maximized if the encroaching vertex lies at a pole of the equatorial sphere (where the *poles* are the two points of the sphere furthest from its equator), as illustrated in Figure 4.10(a). However, if a subfacet that does not contain its own circumcenter is encroached, the distance is maximized if the encroaching vertex lies on the equator, equidistant from the two vertices of the longest edge of the subfacet, as in Figure 4.10(b). Even if the encroaching vertex is well away from the equator, its distance from the nearest vertex of the subfacet can still be larger than $\sqrt{2}$ times the radius of the equatorial sphere. (I have confirmed through my implementation that such cases do arise in practice.)

Rather than settle for a looser guarantee on quality, one can make a small change to the algorithm that will yield a $\sqrt{2}$ bound. When several encroached subfacets exist, they should not be split in arbitrary order. If a vertex p encroaches upon a subfacet f of a facet F, but the projected point $\text{proj}_F(p)$ does not lie in f, then splitting f is not the best choice. One can show (with the following lemma) that there is some other subfacet g of F that is encroached upon by p and contains $\text{proj}_F(p)$. (The lemma assumes that there are no encroached subsegments in the mesh, as they have priority.) A better bound is achieved if the algorithm splits g first and delays the splitting of f indefinitely.

Lemma 26 (Projection Lemma) *Let f be a subfacet of the Delaunay triangulated facet F. Suppose that f is encroached upon by some vertex p, but p does not encroach upon any subsegment of F. Then* $\text{proj}_F(p)$ *lies in the facet F, and p encroaches upon a subfacet of F that contains* $\text{proj}_F(p)$.

Proof: First, I prove that $\text{proj}_F(p)$ lies in F, using similar reasoning to that employed in Lemma 17. Suppose for the sake of contradiction that $\text{proj}_F(p)$ lies outside the facet F. Let c be the centroid of f; c clearly lies inside F. Because all facets are segment-bounded, the line segment connecting c to $\text{proj}_F(p)$ must intersect some subsegment s in the boundary of F. Let $\mathcal{S}$ be the plane that contains s and is orthogonal to F, as illustrated in Figure 4.11(a).

Because f is a Delaunay subfacet of F, its circumcircle (in the plane containing F) encloses no vertex of F. However, its equatorial sphere may enclose vertices—including p—and f might not appear in the tetrahedralization.

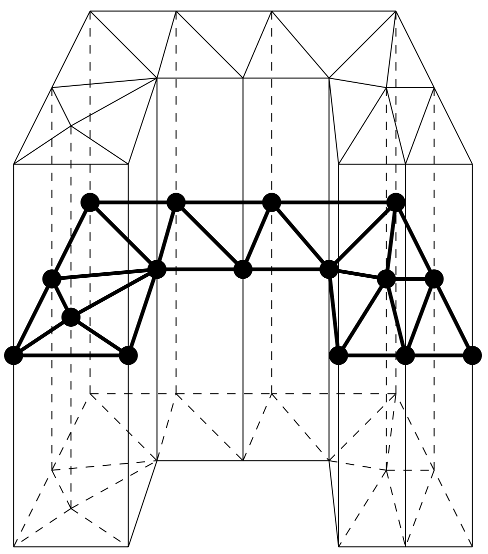


Figure 4.12: Each subfacet's equatorial sphere dominates the triangular prism defined by extending the subfacet orthogonally.

It is apparent that p and $\text{proj}_F(p)$ lie on the same side of $\mathcal{S}$, because the projection is orthogonal to F. Say that a point is *inside* $\mathcal{S}$ if it is on the same side of $\mathcal{S}$ as c, and *outside* $\mathcal{S}$ if it is on the same side as p and $\text{proj}_F(p)$. The circumcircle of f cannot enclose the endpoints of s, because f is Delaunay in F. Furthermore, the circumcenter of f lies in F by Lemma 17. It follows that the portion of f's equatorial sphere outside $\mathcal{S}$ lies entirely inside or on the diametral sphere of s (as the figure demonstrates). Because p is inside or on the equatorial sphere of f, p also lies inside or on the diametral sphere of s, contradicting the assumption that p encroaches upon no subsegment of F.

It follows that $\text{proj}_F(p)$ must be contained in some subfacet g of F. (The containment is not necessarily strict; $\text{proj}_F(p)$ may fall on an edge interior to F, and be contained in two subfacets.) To complete the proof of the lemma, I shall show that p encroaches upon g. If $f = g$ the result follows immediately, so assume that $f \neq g$.

Again, let c be the centroid of f. The line segment connecting c to $\text{proj}_F(p)$ must intersect some edge e of the subfacet g, as illustrated in Figure 4.11(b). Let $\mathcal{E}$ be the plane that contains e and is orthogonal to F. Say that a point is on the *g-side* if it is on the same side of $\mathcal{E}$ as g. Because the triangulation of F is Delaunay, the portion of f's equatorial sphere on the g-side lies entirely inside or on the equatorial sphere of g. The point p lies on the g-side or in $\mathcal{E}$ (because $\text{proj}_F(p)$ lies in g), and p lies inside or on the equatorial sphere of f, so it must also lie inside or on the equatorial sphere of g, and hence encroaches upon g. ■

One way to interpret the Projection Lemma is to imagine that the facet F is orthogonally extended to infinity, so that each subfacet of F defines an infinitely long triangular prism (Figure 4.12). Each subfacet's equatorial sphere dominates its prism, in the sense that the sphere contains any point in the prism that lies within the equatorial sphere of any other subfacet of F. If a vertex p encroaches upon any subfacet of F, then p encroaches upon the subfacet in whose prism p is contained. If p encroaches upon some subfacet of F but is contained in none of the prisms, then p also encroaches upon some boundary subsegment of F.

In the latter case, because encroached subsegments have priority, subsegments encroached upon by p are split until none remains. The Projection Lemma guarantees that any subfacets of F that were encroached upon by p are eliminated in the process.

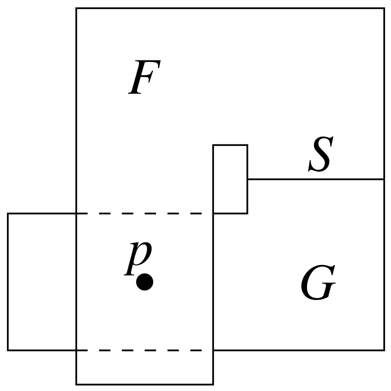


Figure 4.13: Two incident facets separated by a dihedral angle of nearly 180°. What is the local feature size at p?

On the other hand, if p lies in the prism of a subfacet g, and no subsegment is encroached, then splitting g is a good choice. As a result, several new subfacets will appear, at least one of which contains $\text{proj}_F(p)$; if this subfacet is encroached, then it is split as well, and so forth until the subfacet containing $\text{proj}_F(p)$ is not encroached. The Projection Lemma guarantees that any other subfacets of F that were encroached upon by p are eliminated in the process.

4.2.2 Local Feature Sizes of Piecewise Linear Complexes

Because the shape of a facet is versatile, the definition of local feature size does not generalize straightforwardly. Figure 4.13 demonstrates the difficulty. Two facets F and G are incident at a segment S, separated by a dihedral angle of almost 180°. The facets are not convex, and they may pass arbitrarily close to each other in a region far from S. What is the local feature size at the point p? Because F and G are incident, a ball (centered at p) large enough to intersect two nonincident features must have diameter as large as the width of the prongs. However, the size of tetrahedra near p is determined by the distance separating F and G, which could be arbitrarily small. The straightforward generalization of local feature size does not account for this peccadillo of nonconvex facets.

To develop a more appropriate metric, I define a *facet region* to be any region of a facet visible from a single point on its boundary. Visibility is defined solely within the facet in question; the vertices p and q are *visible* to each other if the line segment pq lies entirely in the facet. Two facet regions on two different facets are said to be *incident* if they are defined by the same boundary point. Figure 4.14 illustrates two incident facet regions, and the point that defines them. Two points, one lying in F and one lying in G, are said to lie in incident facet regions if there is any point on the shared boundary of F and G that is visible from both points. They are said to be nonincident feature points (formally defined below) if no such point exists.

Similarly, if a segment S is incident to a facet F at a single vertex q, then S is said to be incident to the facet region of F visible from q. If a vertex v is incident to a facet F, then v is said to be incident to the facet region of F visible from v.

Two distinct points x and y are *nonincident feature points* if x lies on a feature (vertex, segment, or facet) f_x of X, y lies on a feature f_y of X, and there is no point $q \in f_x \cap f_y$ such that the segment xq is entirely contained in f_x and the segment yq is entirely contained in f_y. (Note that q may be x or y.) If such a point q does exist, then x and y lie in incident vertices, segments, or facet regions of X. However, each

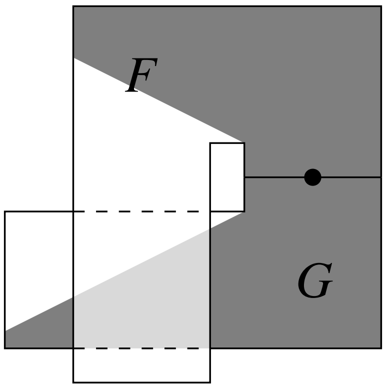


Figure 4.14: Shaded areas are two incident facet regions. Both regions are visible from the indicated point.

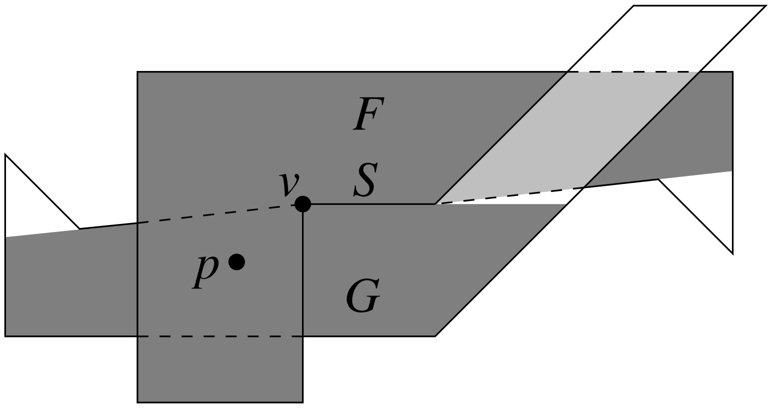


Figure 4.15: Two incident facets separated by a dihedral angle of nearly 180°. The definition of local feature size should not approach zero near v, but it is nonetheless difficult to mesh the region between F and G near v.

point may lie in several features simultaneously; so even if x and y lie in incident facet regions, they may still be nonincident feature points (if they lie in nonincident segments, for instance).

Given a piecewise linear complex X, I define the local feature size $\text{lfs}(p)$ at a point p to be the radius of the smallest ball centered at p that intersects a pair of nonincident feature points.

Unfortunately, careful specification of which portions of facets are incident doesn't solve all the problems attributable to nonconvex facets. Figure 4.15 demonstrates another difficulty. Again, two facets F and G are incident at a segment S, separated by a dihedral angle slightly less than 180°. One endpoint v of S is a reflex vertex of F. The incident facet regions defined by the vertex v have the same problem we encountered in Figure 4.13: the local feature size at point p may be much larger than the distance between facets F and G at point p.

In this case, however, the problem is unavoidable. Suppose one chooses a definition of local feature size that reflects the distance between F and G at p. As p moves toward v, $\text{lfs}(p)$ approaches zero, suggesting that infinitesimally small tetrahedra are needed to mesh the region near v. Intuitively and practically, a useful definition of local feature size must have a positive lower bound. Therefore, $\text{lfs}(p)$ cannot be proportional to the distance between F and G at p.

The mismatch between the definition of local feature size proposed here and the small distance between

F and G at p reflects a fundamental difficulty in meshing the facets of Figure 4.15—a difficulty that is not present in Figure 4.13. In Figure 4.15, it is not possible to mesh the region between F and G at v without resorting to poorly shaped tetrahedra. The facets of Figure 4.13 can be meshed entirely with well-shaped tetrahedra. The three-dimensional Delaunay refinement algorithm discussed here outlaws inputs like Figure 4.15, at least for the purposes of analysis.

Lemma 18, which states that $\text{lfs}(v) \leq \text{lfs}(u) + |uv|$ for any two points u and v, applies to this definition of local feature size just as it applies in two dimensions. The only prerequisite for the correctness of Lemma 18, besides the triangle inequality, is that there be a consistent definition of which pairs of points are nonincident feature points.

4.2.3 Proof of Termination

The proof of termination for three-dimensional Delaunay refinement is similar to that of Ruppert's two-dimensional algorithm. Assume that in the input PLC, any two incident segments are separated by an angle of 60° or greater. If a segment meets a facet at one vertex v, and the orthogonal projection of the segment onto the facet intersects the interior of the facet region defined by v, then the angle separating the segment from the facet must be no less than $\arccos \frac{1}{2\sqrt{2}} \doteq 69.3^\circ$. If the projection of the segment does not intersect the interior of the facet, the Projection Lemma implies that no vertex on the segment can encroach upon any subfacet of the facet without also encroaching upon a boundary segment of the facet, so the 69.3° separation angle is unnecessary. However, there still must be a 60° separation between the segment and the segments incident on v that bound the facet.

The condition for two incident facets is more complicated. If both facets are convex and meet at a segment, then it is sufficient for the facets to be separated by a dihedral angle of 90° or greater. In general, the two facets must satisfy the following *projection condition*.

For any point p where two facets F and G meet, let $\text{vis}_p(F)$ be the facet region of F visible from p, and define $\text{vis}_p(G)$ likewise. By definition, $\text{vis}_p(F)$ and $\text{vis}_p(G)$ are incident facet regions. No point of the orthogonal projection of $\text{vis}_p(F)$ onto G may intersect the interior of $\text{vis}_p(G)$. (Here, "interior" is defined to exclude all boundaries, including isolated slits and input vertices in the interior of the facet.) Formally, for any point p on $F \cap G$, the projection condition requires that $\text{proj}_G(\text{vis}_p(F)) \cap \text{interior}(\text{vis}_p(G)) = \emptyset$, or equivalently, that $\text{proj}_F(\text{vis}_p(G)) \cap \text{interior}(\text{vis}_p(F)) = \emptyset$.

The payoff of this restriction is that, by Lemma 26, no vertex in $\text{vis}_p(F)$ may encroach upon a subfacet contained entirely in $\text{vis}_p(G)$ without also encroaching upon a subsegment of G or a subfacet of G not entirely in $\text{vis}_p(G)$. The converse is also true. The purpose of this restriction is so that no vertex can split a subfacet in a facet region incident to a facet region containing that vertex. Otherwise, subfacets might be split to arbitrarily small sizes through mutual encroachment in regions arbitrarily close to p.

The projection condition just defined is always satisfied by two facets separated by a dihedral angle of exactly 90°. It is also satisfied by facets separated by a dihedral angle greater than 90° if the facets meet each other only at segments whose endpoints are not reflex vertices of either facet. (Recall Figure 4.15, which depicts two facets that are separated by a dihedral angle greater than 90° but fail the projection condition because v is a reflex vertex of F.)

The following lemma extends Lemma 19 to three dimensions. It is true if the algorithm never splits any encroached subfacet f that does not contain the projection $\text{proj}_f(p)$ of the encroaching vertex p. (Even more liberally, an implementation can easily measure the insertion radii of the parent p and its potential progeny, and may split f if the latter is no less than $\frac{1}{\sqrt{2}}$ times the former.)

The insertion radius is defined as before: r_v is the length of the shortest edge incident to v immediately after v is inserted. The parent of a vertex is defined as before, with the following amendments. If v is the circumcenter of a skinny tetrahedron, its parent $p(v)$ is the most recently inserted endpoint of the shortest edge of that tetrahedron. If v is the circumcenter of an encroached subfacet, its parent is the encroaching vertex closest to v (whether that vertex is inserted or rejected).

Lemma 27 *Let v be a vertex, and let $p = p(v)$ be its parent, if one exists. Then either $r_v \geq \text{lfs}(v)$, or $r_v \geq Cr_p$, where*

- *$C = B$ if v is the circumcenter of a skinny tetrahedron;*
- *$C = \frac{1}{\sqrt{2}}$ if v is the midpoint of an encroached subsegment or the circumcenter of an encroached subfacet;*
- *$C = \frac{1}{2\cos\alpha}$ if v and p lie on incident segments separated by an angle of α, or if v lies in the interior of a facet incident to a segment containing p at an angle α, where $45^\circ \leq \alpha < 90^\circ$.*

Proof: If v is an input vertex, the circumcenter of a tetrahedron (Figure 4.16(a)), or the midpoint of an encroached subsegment, then it may be treated exactly as in Lemma 19. One case from that lemma is worth briefly revisiting to show that nothing essential has changed.

If v is inserted at the midpoint of an encroached subsegment s, and its parent $p = p(v)$ is a circumcenter (of a tetrahedron or subfacet) that was considered for insertion but rejected because it encroaches upon s, then p lies inside or on the diametral sphere of s. Because the tetrahedralization/facet triangulation is Delaunay, the circumsphere/circumcircle centered at p encloses no vertices, and in particular does not enclose the endpoints of s. Hence, $r_p \leq \sqrt{2}r_v$; see Figure 4.16(b) for an example where the relation is equality. Note that the change from circles (in the two-dimensional analysis) to spheres makes little difference. Perhaps the clearest way to see this is to observe that if one takes a two-dimensional cross-section that passes through s and p, the cross-section is indistinguishable from the two-dimensional case. (The same argument can be made for the case where p and v lie on incident segments.)

Only the circumstance where v is the circumcenter of an encroached subfacet f remains. Let F be the facet that contains f. There are four cases to consider.

- If the parent p is an input vertex, or if v and p are nonincident feature points, then $\text{lfs}(v) \leq r_v$.
- If p is a tetrahedron circumcenter that was considered for insertion but rejected because it encroaches upon f, then p lies strictly inside the equatorial sphere of f. Because the tetrahedralization is Delaunay, the circumsphere centered at p contains no vertices, including the vertices of f. The subfacet f contains $\text{proj}_f(p)$; otherwise, the algorithm would choose a different encroached subfacet to split first. The height of p above $\text{proj}_f(p)$ is no greater than r_v, and the distance between $\text{proj}_f(p)$ and the nearest vertex of f is no greater than r_v (because $\text{proj}_f(p)$ lies in f), so $r_p \leq \sqrt{2}r_v$. See Figure 4.16(c) for an example where the relation is equality.
- If p was inserted on a segment that is incident to F at one vertex a, separated by an angle of $\alpha \geq 45^\circ$ (Figure 4.16(d)), the shared vertex a cannot lie inside the equatorial sphere of f because the facet F is Delaunay triangulated. (This is true even if f does not appear in the tetrahedralization.) Because the segment and facet are separated by an angle of α, the angle $\angle pav$ is at least α. Because f is encroached upon by p, p lies inside its equatorial sphere. (If f is not present in the tetrahedralization,

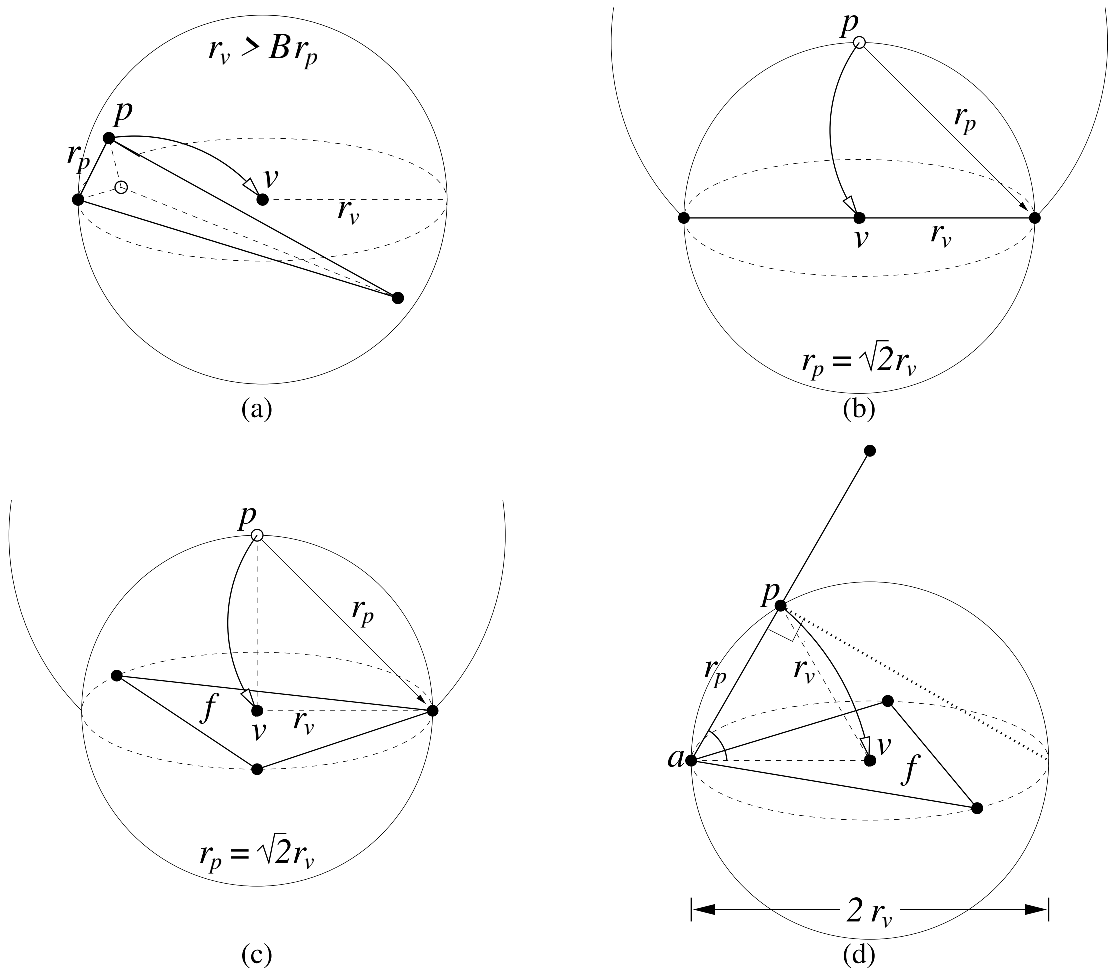


Figure 4.16: The relationship between the insertion radii of a child and its parent. (a) When a skinny tetrahedron is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by a circumcenter, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than its parent's. (c) When a subfacet is encroached upon by the circumcenter of a skinny tetrahedron, and the subfacet contains the orthogonal projection of the encroaching circumcenter, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than its parent's. (d) When a subfacet is encroached upon by the midpoint of a subsegment, and the corresponding facet and segment are incident at one vertex, the analysis differs little from the case of two incident segments.

p might lie on its equatorial sphere in a degenerate case.) Analogously to the case of two incident segments (see Lemma 19), if $\alpha \geq 45°$, then $\frac{r_v}{r_p}$ is minimized when the radius of the equatorial sphere is $r_v = |vp|$, and p lies on the sphere. (If the equatorial sphere were any smaller, it could not contain p.) Therefore, $r_v \geq \frac{r_p}{2\cos\alpha}$. ■

Lemma 27 provides the information one needs to ensure that Delaunay refinement will terminate. As with the two dimensional algorithms, the key is to prevent any vertex from begetting a sequence of descendants with ever-smaller insertion radii.

Figure 4.17 depicts a dataflow graph corresponding to Lemma 27. Mesh vertices are divided into four classes: input vertices (which cannot contribute to cycles), segment vertices (inserted into segments), facet

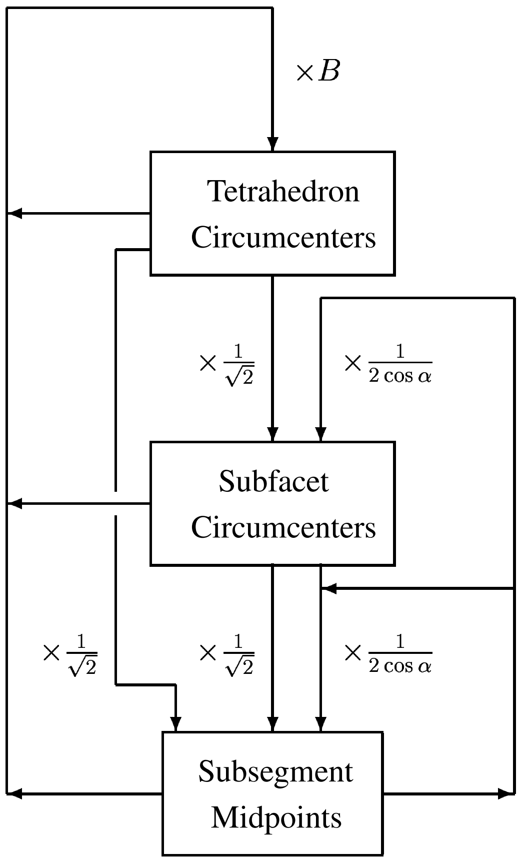

$\times B$
Tetrahedron Circumcenters
$\times \frac{1}{\sqrt{2}}$
$\times \frac{1}{2\cos\alpha}$
Subfacet Circumcenters
$\times \frac{1}{\sqrt{2}}$
$\times \frac{1}{\sqrt{2}}$
$\times \frac{1}{2\cos\alpha}$
Subsegment Midpoints

Figure 4.17: Dataflow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycle has a product smaller than one, the three dimensional Delaunay refinement algorithm will terminate.

vertices (inserted into facet interiors), and free vertices (inserted at circumcenters of tetrahedra). As we have seen, free vertices can father facet vertices whose insertion radii are smaller by a factor of $\sqrt{2}$, and these facet vertices in turn can father segment vertices whose insertion radii are smaller by another factor of $\sqrt{2}$. Hence, to avoid spiralling into the abyss, it is important that segment vertices can only father free vertices whose insertion radii are at least twice as large. This constraint fixes the best guaranteed circumradius-to-shortest edge ratio at $B = 2$.

The need to prevent diminishing cycles also engenders the requirement that incident segments be separated by angles of 60° or more, just as it did in the two-dimensional case. A segment incident to a facet must be separated by an angle of at least $\arccos \frac{1}{2\sqrt{2}} \doteq 69.3^\circ$ so that if a vertex on the segment encroaches upon a subfacet of the facet, the child that results will have an insertion radius at least $\sqrt{2}$ larger than that of its parent. (Recall from Lemma 27 that $r_v \geq \frac{r_p}{2\cos\alpha}$.)

Theorem 28 *Let* $\text{lfs}_{\min}$ *be the shortest distance between two nonincident feature points of the input PLC. Suppose that any two incident segments are separated by an angle of at least* 60°*, any two incident facet regions satisfy the projection condition, and any segment incident to a facet region at one vertex is separated from it by an angle of at least* $\arccos \frac{1}{2\sqrt{2}}$ *or satisfies the projection condition.*

Suppose a tetrahedron is considered to be skinny if its circumradius-to-shortest edge ratio is larger than

B, where $B \geq 2$. The three-dimensional Delaunay refinement algorithm described above will terminate, with no tetrahedralization edge shorter than $\mathrm{lfs}_{\min}$.

Proof: Suppose for the sake of contradiction that the algorithm introduces one or more edges shorter than $\mathrm{lfs}_{\min}$ into the mesh. Let e be the first such edge introduced. Clearly, the endpoints of e cannot both be input vertices, nor can they lie on nonincident feature points. Let v be the most recently inserted endpoint of e.

By assumption, no edge shorter than $\mathrm{lfs}_{\min}$ existed before v was inserted. Hence, for any ancestor a of v that is a mesh vertex, $r_a \geq \mathrm{lfs}_{\min}$. Let $p = p(v)$ be the parent of v, let $g = p(p)$ be the grandparent of v (if one exists), and let $h = p(g)$ be the great-grandparent of v (if one exists). Because of the projection condition, v and p cannot lie in incident facet regions. Consider the following cases.

- If v is the circumcenter of a skinny tetrahedron, then by Lemma 27, $r_v \geq Br_p \geq 2r_p$.
- If v is the midpoint of an encroached subsegment or the circumcenter of an encroached subfacet, and p is the circumcenter of a skinny tetrahedron, then by Lemma 27, $r_v \geq \frac{1}{\sqrt{2}}r_p \geq \frac{B}{\sqrt{2}}r_g \geq \sqrt{2}r_g$.
- If v is the midpoint of an encroached subsegment, p is the circumcenter of an encroached subfacet, and g is the circumcenter of a skinny tetrahedron, then by Lemma 27, $r_v \geq \frac{1}{\sqrt{2}}r_p \geq \frac{1}{2}r_g \geq \frac{B}{2}r_h \geq r_h$.
- If v and p lie on incident segments, then by Lemma 27, $r_v \geq \frac{r_p}{2\cos\alpha}$. Because $\alpha \geq 60^\circ$, $r_v \geq r_p$.
- If v is the circumcenter of an encroached subfacet and p lies on a segment incident (at a single vertex) to the facet containing v, then by Lemma 27, $r_v \geq \frac{r_p}{2\cos\alpha}$. Because $\alpha \geq \arccos\frac{1}{2\sqrt{2}}$, $r_v \geq \sqrt{2}r_p$.
- If v is the midpoint of an encroached subsegment, p is the (rejected) circumcenter of an encroached subfacet, and g lies on a segment incident (at a single vertex) to the facet containing p, then by Lemma 27, $r_v \geq \frac{1}{\sqrt{2}}r_p \geq \frac{1}{2\sqrt{2}\cos\alpha}r_g$. Because $\alpha \geq \arccos\frac{1}{2\sqrt{2}}$, $r_v \geq r_g$.
- If v is the midpoint of an encroached subsegment, and p has been inserted on a nonincident segment or facet region, then by the definition of parent, pv is the shortest edge introduced by the insertion of v. Because p and v lie on nonincident entities, p and v are separated by a distance of at least $\mathrm{lfs}_{\min}$, contradicting the assumption that e has length less than $\mathrm{lfs}_{\min}$.

In the first six cases, $r_p \geq r_a$ for some mesh vertex a that is an ancestor of p. It follows that $r_p \geq \mathrm{lfs}_{\min}$, contradicting the assumption that e has length less than $\mathrm{lfs}_{\min}$. Because no edge shorter than $\mathrm{lfs}_{\min}$ is ever introduced, the algorithm must terminate. ■

4.2.4 Proof of Good Grading

As with the two-dimensional algorithm, a stronger termination proof is possible, showing that each edge of the output mesh has length proportional to the local feature sizes of its endpoints, and thus guaranteeing nicely graded meshes. The proof makes use of Lemma 21, which generalizes unchanged to three or more dimensions. Recall that the lemma states that if $r_v \geq Cr_p$ for some vertex v with parent p, then their lfs-weighted vertex densities are related by the formula $D_v \leq 1 + \frac{D_p}{C}$, where $D_v = \frac{\mathrm{lfs}(v)}{r_v}$ and $D_p = \frac{\mathrm{lfs}(p)}{r_p}$.

Lemma 29 *Suppose the quality bound B is strictly larger than 2, and all angles between segments and facets satisfy the conditions listed in Theorem 28, with all inequalities replaced by strict inequalities.*

Then there exist fixed constants $D_T \geq 1$, $D_F \geq 1$, and $D_S \geq 1$ such that, for any vertex v inserted (or rejected) at the circumcenter of a skinny tetrahedron, $D_v \leq D_T$; for any vertex v inserted (or rejected) at the circumcenter of an encroached subfacet, $D_v \leq D_F$; and for any vertex v inserted at the midpoint of an encroached subsegment, $D_v \leq D_S$. Hence, the insertion radius of every vertex has a lower bound proportional to its local feature size.

Proof: Consider any non-input vertex v with parent $p = p(v)$. If p is an input vertex, then $D_p = \frac{\text{lfs}(p)}{r_p} \leq 1$ by Lemma 27. Otherwise, assume for the sake of induction that the lemma is true for p. In either case, $D_p \leq \max\{D_T, D_F, D_S\}$.

First, suppose v is inserted or considered for insertion at the circumcenter of a skinny tetrahedron. By Lemma 27, $r_v \geq B r_p$. Therefore, by Lemma 21, $D_v \leq 1 + \frac{\max\{D_T, D_F, D_S\}}{B}$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen sufficiently large that

$$1 + \frac{\max\{D_T, D_F, D_S\}}{B} \leq D_T. \tag{4.1}$$

Second, suppose v is inserted or considered for insertion at the circumcenter of a subfacet f. If its parent p is an input vertex or if v and p are nonincident feature points, then $\text{lfs}(v) \leq r_v$, and the theorem holds. If p is the circumcenter of a skinny tetrahedron (rejected because it encroaches upon f), $r_v \geq \frac{r_p}{\sqrt{2}}$ by Lemma 27, so by Lemma 21, $D_v \leq 1 + \sqrt{2} D_T$.

Alternatively, if p lies on a segment incident to the facet containing f, then $r_v \geq \frac{r_p}{2\cos\alpha}$ by Lemma 27, and thus by Lemma 21, $D_v \leq 1 + 2D_S \cos\alpha$. It follows that one can prove that $D_v \leq D_F$ if D_F is chosen sufficiently large that

$$1 + \sqrt{2} D_T \leq D_F, \quad \text{and} \tag{4.2}$$

$$1 + 2D_S \cos\alpha \leq D_F. \tag{4.3}$$

Third, suppose v is inserted at the midpoint of a subsegment s. If its parent p is an input vertex or if v and p are nonincident feature points, then $\text{lfs}(v) \leq r_v$, and the theorem holds. If p is the circumcenter of a skinny tetrahedron or encroached subfacet (rejected because it encroaches upon s), $r_v \geq \frac{r_p}{\sqrt{2}}$ by Lemma 27, so by Lemma 21, $D_v \leq 1 + \sqrt{2} \max\{D_T, D_F\}$.

Alternatively, if p and v lie on incident segments, then $r_v \geq \frac{r_p}{2\cos\alpha}$ by Lemma 27, and thus by Lemma 21, $D_v \leq 1 + 2D_S \cos\alpha$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen sufficiently large that

$$1 + \sqrt{2} \max\{D_T, D_F\} \leq D_S \quad \text{and} \tag{4.4}$$

$$1 + 2D_S \cos\alpha \leq D_S. \tag{4.5}$$

If the quality bound B is strictly larger than 2, Inequalities 4.1, 4.2, and 4.4 are simultaneously satisfied by choosing

$$D_T = \frac{B + 1 + \sqrt{2}}{B - 2}, \qquad D_F = \frac{(1 + \sqrt{2})B + \sqrt{2}}{B - 2}, \qquad D_S = \frac{(3 + \sqrt{2})B}{B - 2}.$$

If the smallest angle α_{FS} between any facet and any segment is strictly greater than $\arccos \frac{1}{2\sqrt{2}} \doteq 69.3^\circ$, Inequalities 4.3 and 4.4 may be satisfied by choosing

$$D_F = \frac{1 + 2\cos\alpha_{FS}}{1 - 2\sqrt{2}\cos\alpha_{FS}}, \qquad D_S = \frac{1+\sqrt{2}}{1 - 2\sqrt{2}\cos\alpha_{FS}},$$

if these values exceed those specified above. In this case, adjust D_T upward if necessary to satisfy Inequality 4.1.

If the smallest angle α_{SS} between two segments is strictly greater than 60°, Inequality 4.5 may be satisfied by choosing

$$D_S = \frac{1}{1 - 2\cos\alpha_{SS}},$$

if this value exceeds those specified above. In this case, adjust D_T and D_F upward if necessary to satisfy Inequalities 4.1 and 4.2. ■

Theorem 30 *For any vertex v of the output mesh, the distance to its nearest neighbor is at least $\frac{\mathrm{lfs}(v)}{D_S+1}$.*

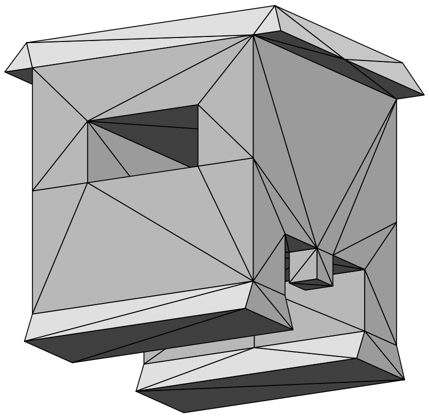
Proof: Inequality 4.4 indicates that D_S is larger than D_T and D_F. The remainder of the proof is identical to that of Theorem 23. ■

To provide an example, suppose $B = 2.5$ and the input PLC has no acute angles. Then $D_T \doteq 9.8$, $D_F \doteq 14.9$, and $D_S \doteq 22.1$. Hence, the spacing of vertices is at worst about 23 times smaller than the local feature size. Note that as B approaches 2, α_{SS} approaches 60°, or α_{FS} approaches $\arccos \frac{1}{2\sqrt{2}}$, the values of D_T, D_F, and D_S approach infinity.

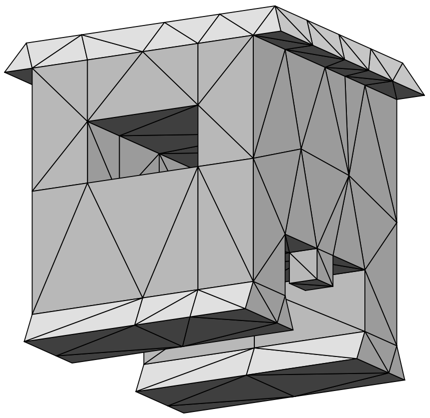
As Figure 4.18 shows, the algorithm performs much better in practice. The upper mesh is the initial tetrahedralization after all segments and facets are inserted and unwanted tetrahedra have been removed from the holes. (Some subsegments remain encroached because during the segment and facet recovery stages, my implementation only splits an encroached subsegment if it is missing or it is encroached within the facet currently being recovered.) In this example, as soon as all encroached subsegments and subfacets have been eliminated (middle left), the largest circumradius-to-shortest edge ratio is already less than 2.1. The shortest edge length is 1, and $\mathrm{lfs}_{\min} = \sqrt{5}$, so the spectre of edge lengths 23 times smaller than the local feature size has not materialized. As the quality bound B decreases, the number of elements in the final mesh increases gracefully until B drops below 1.05. With $B = 1.04$, the algorithm fails to terminate.

Figure 4.19 offers a demonstration of the grading of a tetrahedralization generated by Delaunay refinement. A cube has been truncated at one corner, cutting off a portion whose width is one-millionth that of the cube. Although this mesh satisfies a bound on circumradius-to-shortest edge ratio of $B = 1.2$, reasonably good grading is apparent. For this bound there is no theoretical guarantee of good grading, but the worst edge is 73 times shorter than the local feature size at one of its endpoints. If a bound of $B = 2.5$ is applied, the worst edge is 9 (rather than 23) times smaller than the local feature size at one of its endpoints.

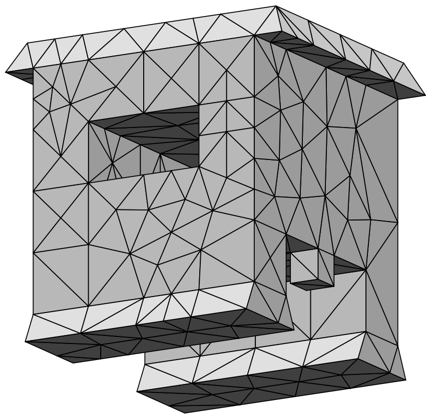
Unfortunately, the proof of good grading does not yield a size-optimality proof as it does in the two-dimensional case. Gary Miller and Dafna Talmor (private communication) have pointed out the counterexample depicted in Figure 4.20. Inside this PLC, two segments pass very close to each other without intersecting. The PLC might reasonably be tetrahedralized with a few dozen tetrahedra having bounded circumradius-to-shortest edge ratios, if these tetrahedra include a sliver tetrahedron whose four vertices are



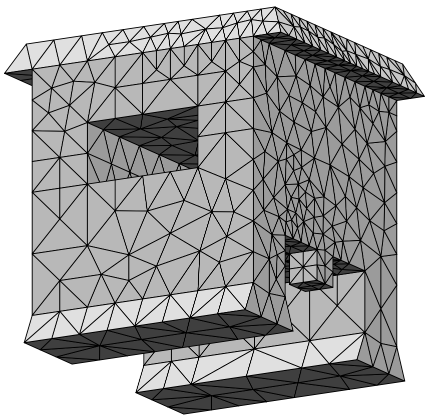
Initial tetrahedralization after segment and facet recovery. 71 vertices, 146 tetrahedra.



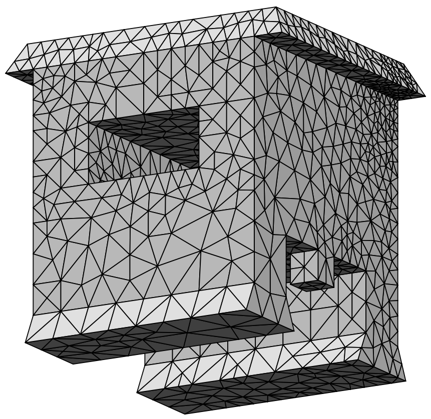
$B = 2.095$, $\theta_{\min} = 1.96°$, $\theta_{\max} = 176.02°$, $h_{\min} = 1$, 143 vertices, 346 tetrahedra.



$B = 1.2$, $\theta_{\min} = 1.20°$, $\theta_{\max} = 178.01°$, $h_{\min} = 0.743$, 334 vertices, 1009 tetrahedra.



$B = 1.07$, $\theta_{\min} = 1.90°$, $\theta_{\max} = 177.11°$, $h_{\min} = 0.369$, 1397 vertices, 5596 tetrahedra.



$B = 1.041$, $\theta_{\min} = 0.93°$, $\theta_{\max} = 178.40°$, $h_{\min} = 0.192$, 3144 vertices, 13969 tetrahedra.

Figure 4.18: Several meshes of a $10 \times 10 \times 10$ PLC generated with different bounds (B) on circumradius-to-shortest edge ratio. Below each mesh is listed the smallest dihedral angle $\theta_{\min}$, the largest dihedral angle $\theta_{\max}$, and the shortest edge length $h_{\min}$. The algorithm does not terminate on this PLC for the bound $B = 1.04$.

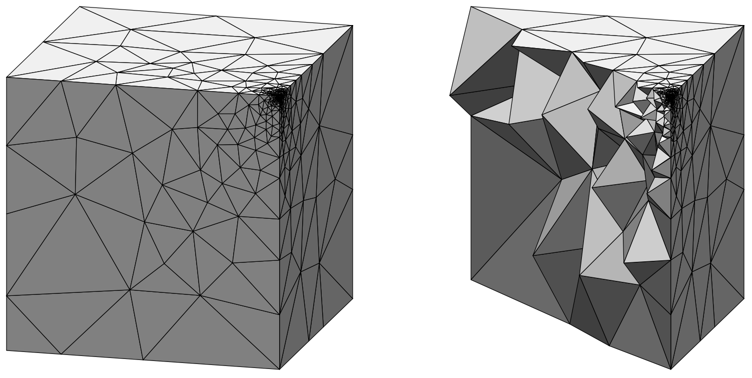


Figure 4.19: At left, a mesh of a truncated cube. At right, a cross-section through a diagonal of the top face.

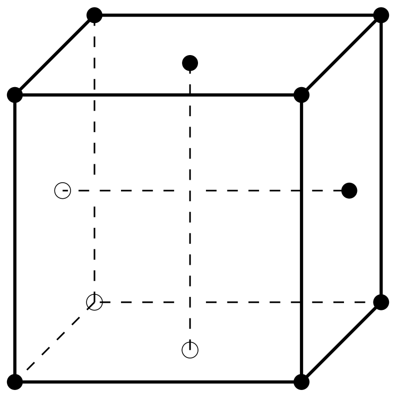


Figure 4.20: A counterexample demonstrating that the three-dimensional Delaunay refinement algorithm is not size-optimal.

the endpoints of the two interior segments. However, the best Delaunay refinement can promise is to fill the region with tetrahedra whose edge lengths are proportional to the distance between the two segments. Because this distance may be arbitrarily small, the algorithm is not size-optimal. If a Delaunay refinement algorithm were developed that offered guaranteed bounds for the dihedral angles, and not merely the circumradius-to-shortest edge ratios, then size-optimality might be proven using ideas like those with which Mitchell and Vavasis [51, 52] demonstrate the size-optimality of their octree-based algorithms.

4.3 Improving the Quality Bound in the Interior of the Mesh

The improvement to two-dimensional Delaunay refinement described in Section 3.6 applies in three dimensions as well. Any of the following three strategies may be used to improve the quality of most of the tetrahedra without jeopardizing the termination guarantee.

- Use a quality bound of $B = 1$ for tetrahedra that are not in contact with facet or segment interiors, a quality bound of $B = \sqrt{2}$ for any tetrahedron that is not in contact with a segment interior but has a vertex that lies in the interior of a facet, and a quality bound of $B = 2$ for any tetrahedron having a vertex that lies in the interior of a segment. The flow diagram for this strategy appears as Figure 4.21.

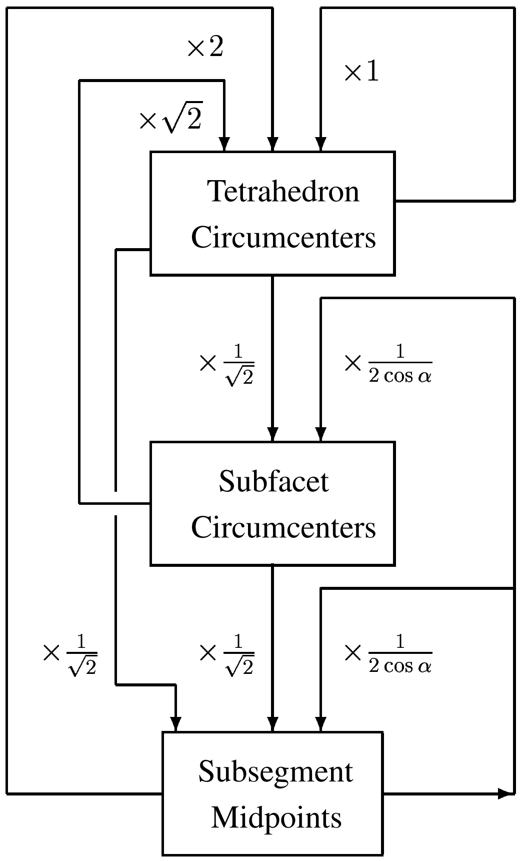
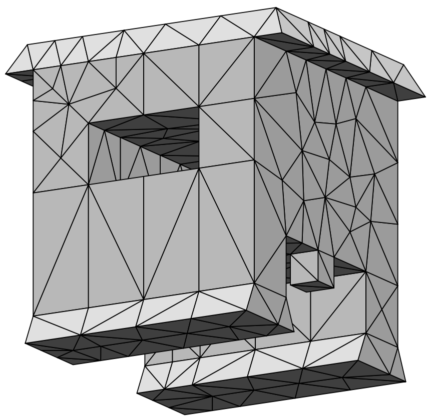

$\times 2$
$\times 1$
$\times\sqrt{2}$
Tetrahedron Circumcenters
$\times\frac{1}{\sqrt{2}}$
$\times\frac{1}{2\cos\alpha}$
Subfacet Circumcenters
$\times\frac{1}{\sqrt{2}}$
$\times\frac{1}{\sqrt{2}}$
$\times\frac{1}{2\cos\alpha}$
Subsegment Midpoints

Figure 4.21: Dataflow diagram for three-dimensional Delaunay refinement with improved quality away from the boundaries.

- Attempt to insert the circumcenter of any tetrahedron whose circumradius-to-shortest edge ratio is larger than one. If any subsegments would be encroached, the circumcenter is rejected as usual, but the encroached subsegments are split only if the tetrahedron's circumradius-to-shortest edge ratio is greater than $\sqrt{2}$. If any subfacets would be encroached, they are split only if the tetrahedron's circumradius-to-shortest edge ratio is greater than 2.

- Attempt to insert the circumcenter of any tetrahedron whose circumradius-to-shortest edge ratio is larger than one. If any subsegments or subfacets would be encroached, the circumcenter is rejected as usual. Each encroached subsegment is checked to determine the insertion radius of the new vertex that might be inserted at its midpoint. Each encroached subfacet is checked to determine whether its circumcenter would encroach upon any subsegments, and if so, what the insertion radius of the new vertices at their midpoints would be. If a subfacet's circumcenter does not encroach upon any subsegments, the insertion radius of the subfacet's circumcenter is determined. The only midpoints and circumcenters inserted are those whose insertion radii are at least as large as the length of the shortest edge of the skinny tetrahedron.

The first strategy differs tends to space segment vertices more closely than facet vertices, which are spaced more closely than free vertices. The other two strategies tend to space vertices of all types somewhat more equally. As in the two-dimensional case, good grading is maintained if the quality bound B_I in the


$B = 1.88$, $\theta_{\min} = 23.5°$, $\theta_{\max} = 144.8°$,
$h_{\min} = 0.765$, 307 vertices, 891 tetrahedra.

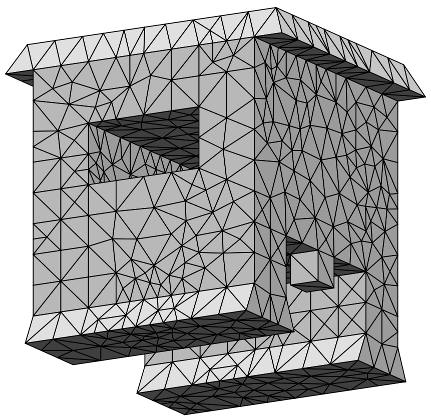

$B = 2.02$, $\theta_{\min} = 21.3°$, $\theta_{\max} = 148.8°$,
$h_{\min} = 0.185$, 1761 vertices, 7383 tetrahedra.

Figure 4.22: Meshes created by Delaunay refinement with bounds on the smallest dihedral angle $\theta_{\min}$. Also listed for each mesh is its largest dihedral angle $\theta_{\max}$ and its shortest edge length $h_{\min}$. Compare with Figure 4.18 on Page 105.

interior of the mesh is greater than one. Then Inequality 4.1 is accompanied by the inequality

$$\frac{B_I}{B_I - 1} \leq D_T,$$

which is familiar from Section 3.6.

4.4 Sliver Removal by Delaunay Refinement

Although I have proven no theoretical guarantees about Delaunay refinement's ability to remove sliver tetrahedra, it is nonetheless natural to wonder whether Delaunay refinement might be effective in practice. If one inserts a vertex at the circumcenter of any tetrahedron with a small dihedral angle, will the algorithm fail to terminate?

As Figure 4.22 demonstrates, Delaunay refinement can succeed for useful dihedral angle bounds. Each of the meshes illustrated was generated by applying a lower bound $\theta_{\min}$ on dihedral angles, rather than a circumradius-to-shortest edge ratio bound. However, the implementation prioritizes poor tetrahedra according to their ratios, and thus slivers are split last. I suspect that the program generates meshes with fewer tetrahedra this way, and that the likelihood of termination is greater. Intuitively, one expects that a vertex inserted at the circumcenter of the tetrahedron with the largest ratio is more likely to eliminate more bad tetrahedra. See Section 3.5.2 for further discussion.

Both meshes illustrated have dihedral angles bounded between 21° and 149°. The mesh on the right was generated with bounds on both tetrahedron volume and dihedral angle, so that enough tetrahedra were generated to ensure that the mesh on the left wasn't merely a fluke. (The best attainable lower bound drops by 2.2° as a result.) Experiments with very large meshes suggest that a minimum angle of 19° can be obtained reliably.

Chew [17] offers hints as to why slivers might be eliminated so readily. A sliver can always be eliminated by splitting it, but how can one avoid creating new slivers in the process? Chew observes that a newly

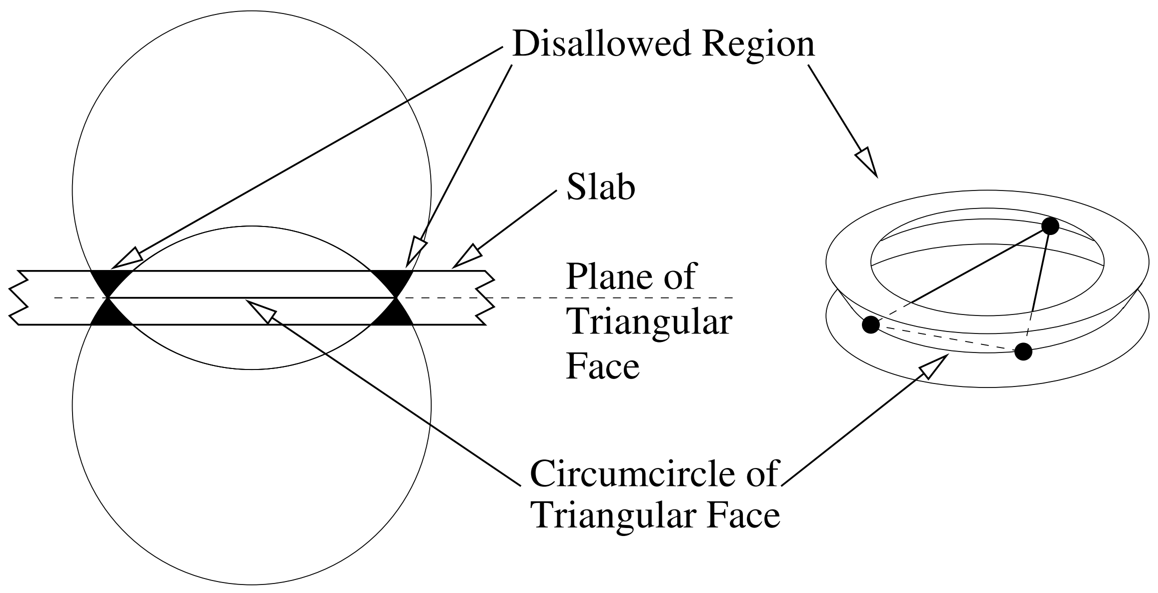


Figure 4.23: Left: A side view of the plane containing a triangular face. In conjunction with this face, a newly inserted vertex can form a sliver with both a bad dihedral angle and a good circumradius-to-shortest edge ratio only if it is inserted in the disallowed region (black). Right: An oblique view of the disallowed region of a triangular face.

inserted vertex can only take part in a sliver if it is positioned badly relative to a triangular face already in the mesh. Figure 4.23 illustrates the set of bad positions. At left, a side view of the plane containing a face of the tetrahedralization is drawn. A tetrahedron formed by the face and a new vertex can have a small dihedral angle only if the new vertex lies within the slab depicted; this slab is the set of all points within a certain distance from the plane. Late in the Delaunay refinement process, such a tetrahedron can only arise if its circumradius-to-shortest edge ratio is small, which implies that it must lie in the region colored black in Figure 4.23 (left). This *disallowed region*, depicted at right, is shaped like a ring with an hourglass cross-section.

Chew shows that if the slab associated with each face is sufficiently thin, a randomized Delaunay refinement algorithm can avoid ever placing a vertex in the disallowed region of any face. The key idea is that each new vertex is not inserted precisely at a circumcenter; rather, a candidate vertex is generated at a randomly chosen location in the inner half of the circumsphere's radius. If the candidate vertex lies in some face's disallowed region, the candidate is rejected and a new one generated in its stead.

The algorithm will eventually generate a successful candidate, because the number of nearby triangular faces is bounded, and the volume of each disallowed region is small. If the sum of the volumes of the disallowed regions is less than the volume of the region in which candidate vertices are generated, a good candidate will eventually be found. To ensure that this condition is met, the slabs are made very thin.

Chew derives an explicit bound on the worst-case tetrahedron aspect ratio, which is too small to serve as a practical guarantee. However, there is undoubtedly a great deal of slack in the derivation. Even if the slabs are made thick enough to offer a useful bound on the minimum dihedral angle, the small volume of the disallowed region suggests that the practical prospects are good. My non-randomized Delaunay refinement implementation seems to verify this intuition. I have not yet tested whether randomization is helpful in practice. Although randomization may reduce the frequency with which slivers are generated, the act of inserting vertices off-center in circumspheres weakens the bound on circumradius-to-shortest edge ratio.

Unfortunately, my success in removing slivers is probably due in part to the severe restrictions on input angles I have imposed. Practitioners report that they have the most difficulty removing slivers at the boundary of a mesh, especially near small angles. Figure 2.48 on Page 47 offers a demonstration of this observation. Mesh improvement techniques such as optimization-based smoothing and topological transformations, discussed in Section 2.2.4, can likely remove some of the imperfections that cannot be removed directly by Delaunay refinement.

Bibliography

[1] Franz Aurenhammer. *Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure*. ACM Computing Surveys **23**(3):345–405, September 1991.

[2] Ivo Babuška and A. K. Aziz. *On the Angle Condition in the Finite Element Method*. SIAM Journal on Numerical Analysis **13**(2):214–226, April 1976.

[3] Brenda S. Baker, Eric Grosse, and C. S. Rafferty. *Nonobtuse Triangulation of Polygons*. Discrete and Computational Geometry **3**(2):147–168, 1988.

[4] T. J. Baker. *Automatic Mesh Generation for Complex Three-Dimensional Regions using a Constrained Delaunay Triangulation*. Engineering with Computers **5**:161–175, 1989.

[5] T. J. Barth and D. C. Jesperson. *The Design and Application of Upwind Schemes on Unstructured Meshes*. Proceedings of the AIAA 27th Aerospace Sciences Meeting (Reno, Nevada), 1989.

[6] Marshall Bern and David Eppstein. *Mesh Generation and Optimal Triangulation*. Computing in Euclidean Geometry (Ding-Zhu Du and Frank Hwang, editors), Lecture Notes Series on Computing, volume 1, pages 23–90. World Scientific, Singapore, 1992.

[7] Marshall Bern, David Eppstein, and John R. Gilbert. *Provably Good Mesh Generation*. 31st Annual Symposium on Foundations of Computer Science, pages 231–241. IEEE Computer Society Press, 1990.

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

[9] Scott A. Canann, S. N. Muthukrishnan, and R. K. Phillips. *Topological Refinement Procedures for Triangular Finite Element Meshes*. Engineering with Computers **12**(3 & 4):243–255, 1996.

[10] Graham F. Carey and John Tinsley Oden. *Finite Elements: Computational Aspects*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[11] James C. Cavendish, David A. Field, and William H. Frey. *An Approach to Automatic Three-Dimensional Finite Element Mesh Generation*. International Journal for Numerical Methods in Engineering **21**(2):329–347, February 1985.

[12] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. *Sliver Exudation*. Proceedings of the Fifteenth Annual Symposium on Computational Geometry (Miami Beach, Florida), pages 1–13. Association for Computing Machinery, June 1999.

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

[14] ______. *Guaranteed-Quality Triangular Meshes.* Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[15] ______. *Building Voronoi Diagrams for Convex Polygons in Linear Expected Time.* Technical Report PCS-TR90-147, Department of Mathematics and Computer Science, Dartmouth College, 1990.

[16] ______. *Guaranteed-Quality Mesh Generation for Curved Surfaces.* Proceedings of the Ninth Annual Symposium on Computational Geometry (San Diego, California), pages 274–280. Association for Computing Machinery, May 1993.

[17] ______. *Guaranteed-Quality Delaunay Meshing in 3D.* Proceedings of the Thirteenth Annual Symposium on Computational Geometry, pages 391–393. Association for Computing Machinery, June 1997.

[18] Kenneth L. Clarkson and Peter W. Shor. *Applications of Random Sampling in Computational Geometry, II.* Discrete & Computational Geometry **4**(1):387–421, 1989.

[19] E. F. D'Azevedo and R. B. Simpson. *On Optimal Interpolation Triangle Incidences.* SIAM Journal on Scientific and Statistical Computing **10**:1063–1075, 1989.

[20] Boris N. Delaunay. *Sur la Sphère Vide.* Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk **7**:793–800, 1934.

[21] Tamal Krishna Dey, Chanderjit L. Bajaj, and Kokichi Sugihara. *On Good Triangulations in Three Dimensions.* International Journal of Computational Geometry & Applications **2**(1):75–95, 1992.

[22] Rex A. Dwyer. *A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations.* Algorithmica **2**(2):137–151, 1987.

[23] ______. *Higher-Dimensional Voronoi Diagrams in Linear Expected Time.* Discrete & Computational Geometry **6**(4):343–367, 1991.

[24] Steven Fortune. *A Sweepline Algorithm for Voronoi Diagrams.* Algorithmica **2**(2):153–174, 1987.

[25] ______. *Voronoi Diagrams and Delaunay Triangulations.* Computing in Euclidean Geometry (Ding-Zhu Du and Frank Hwang, editors), Lecture Notes Series on Computing, volume 1, pages 193–233. World Scientific, Singapore, 1992.

[26] Lori A. Freitag, Mark Jones, and Paul Plassman. *An Efficient Parallel Algorithm for Mesh Smoothing.* Fourth International Meshing Roundtable (Albuquerque, New Mexico), pages 47–58. Sandia National Laboratories, October 1995.

[27] Lori A. Freitag and Carl Ollivier-Gooch. *A Comparison of Tetrahedral Mesh Improvement Techniques.* Fifth International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 87–100. Sandia National Laboratories, October 1996.

[28] ______. *Tetrahedral Mesh Improvement Using Swapping and Smoothing.* To appear in International Journal for Numerical Methods in Engineering, 1997.

[29] William H. Frey. *Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes.* International Journal for Numerical Methods in Engineering **24**(11):2183–2200, November 1987.

[30] William H. Frey and David A. Field. *Mesh Relaxation: A New Technique for Improving Triangulations*. International Journal for Numerical Methods in Engineering **31**:1121–1133, 1991.

[31] P. L. George, F. Hecht, and E. Saltel. *Automatic Mesh Generator with Specified Boundary*. Computer Methods in Applied Mechanics and Engineering **92**(3):269–288, November 1991.

[32] N. A. Golias and T. D. Tsiboukis. *An Approach to Refining Three-Dimensional Tetrahedral Meshes Based on Delaunay Transformations*. International Journal for Numerical Methods in Engineering **37**:793–812, 1994.

[33] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. *Randomized Incremental Construction of Delaunay and Voronoi Diagrams*. Algorithmica **7**(4):381–413, 1992. Also available as Stanford University Computer Science Department Technical Report STAN-CS-90-1300 and in Springer-Verlag Lecture Notes in Computer Science, volume 443.

[34] Leonidas J. Guibas and Jorge Stolfi. *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*. ACM Transactions on Graphics **4**(2):74–123, April 1985.

[35] Carol Hazlewood. *Approximating Constrained Tetrahedrizations*. Computer Aided Geometric Design **10**:67–87, 1993.

[36] L. R. Hermann. *Laplacian-Isoparametric Grid Generation Scheme*. Journal of the Engineering Mechanics Division of the American Society of Civil Engineers **102**:749–756, October 1976.

[37] A. Jameson, T. J. Baker, and N. P. Weatherill. *Calculation of Inviscid Transonic Flow over a Complete Aircraft*. Proceedings of the 24th AIAA Aerospace Sciences Meeting (Reno, Nevada), 1986.

[38] Barry Joe. *Three-Dimensional Triangulations from Local Transformations*. SIAM Journal on Scientific and Statistical Computing **10**:718–741, 1989.

[39] ———. *Construction of Three-Dimensional Triangulations using Local Transformations*. Computer Aided Geometric Design **8**:123–142, 1991.

[40] ———. *Construction of k-Dimensional Delaunay Triangulations using Local Transformations*. SIAM Journal on Scientific Computing **14**(6):1415–1436, November 1993.

[41] ———. *Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations*. SIAM Journal on Scientific Computing **16**(6):1292–1307, November 1995.

[42] Anton Szandor LaVey. *The Satanic Bible*. Avon Books, New York, 1969.

[43] Charles L. Lawson. *Software for C^1 Surface Interpolation*. Mathematical Software III (John R. Rice, editor), pages 161–194. Academic Press, New York, 1977.

[44] Der-Tsai Lee and Bruce J. Schachter. *Two Algorithms for Constructing a Delaunay Triangulation*. International Journal of Computer and Information Sciences **9**(3):219–242, 1980.

[45] Rainald Löhner. *Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method*. Proceedings of the 26th AIAA Aerospace Sciences Meeting (Reno, Nevada), 1988.

[46] David L. Marcum and Nigel P. Weatherill. *Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection*. Twelfth AIAA Applied Aerodynamics Conference (Colorado Springs, Colorado), number AIAA 94-1926, June 1994.

[47] Dimitri J. Mavriplis. *An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness*. Technical Report 92-49, ICASE, October 1992.

[48] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. *A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition*. Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing (Las Vegas, Nevada), pages 683–692, May 1995.

[49] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. *Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening*. Fifth International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 47–61, October 1996.

[50] Scott A. Mitchell. *Cardinality Bounds for Triangulations with Bounded Minimum Angle*. Proceedings of the Sixth Canadian Conference on Computational Geometry (Saskatoon, Saskatchewan, Canada), pages 326–331, August 1994.

[51] Scott A. Mitchell and Stephen A. Vavasis. *Quality Mesh Generation in Three Dimensions*. Proceedings of the Eighth Annual Symposium on Computational Geometry, pages 212–221, 1992.

[52] ———. *Quality Mesh Generation in Higher Dimensions*. Submitted to SIAM Journal on Computing. Manuscript available from http://www.cs.cornell.edu/Info/People/vavasis/vavasis.html, February 1997.

[53] Friedhelm Neugebauer and Ralf Diekmann. *Improved Mesh Generation: Not Simple but Good*. Fifth International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 257–270. Sandia National Laboratories, October 1996.

[54] V. N. Parthasarathy and Srinivas Kodiyalam. *A Constrained Optimization Approach to Finite Element Mesh Smoothing*. Finite Elements in Analysis and Design **9**:309–320, 1991.

[55] V. T. Rajan. *Optimality of the Delaunay Triangulation in* $\mathbb{R}^d$. Proceedings of the Seventh Annual Symposium on Computational Geometry, pages 357–363, 1991.

[56] James Martin Ruppert. *Results on Triangulation and High Quality Mesh Generation*. Ph.D. thesis, University of California at Berkeley, Berkeley, California, 1992.

[57] Jim Ruppert. *A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation*. Technical Report UCB/CSD 92/694, University of California at Berkeley, Berkeley, California, 1992.

[58] ———. *A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation*. Proceedings of the Fourth Annual Symposium on Discrete Algorithms, pages 83–92. Association for Computing Machinery, January 1993.

[59] ———. *A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation*. Journal of Algorithms **18**(3):548–585, May 1995.

[60] Jim Ruppert and Raimund Seidel. *On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra*. Discrete & Computational Geometry **7**(3):227–254, 1992.

[61] H. Samet. *The Quadtree and Related Hierarchical Data Structures*. Computing Surveys **16**:188–260, 1984.

[62] E. Schönhardt. *Über die Zerlegung von Dreieckspolyedern in Tetraeder*. Mathematische Annalen **98**:309–312, 1928.

[63] Raimund Seidel. *Backwards Analysis of Randomized Geometric Algorithms*. Technical Report TR-92-014, International Computer Science Institute, University of California at Berkeley, Berkeley, California, February 1992.

[64] Michael I. Shamos and Dan Hoey. *Closest-Point Problems*. 16th Annual Symposium on Foundations of Computer Science (Berkeley, California), pages 151–162. IEEE Computer Society Press, October 1975.

[65] Mark S. Shephard and Marcel K. Georges. *Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique*. International Journal for Numerical Methods in Engineering **32**:709–749, 1991.

[66] Jonathan Richard Shewchuk. *A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 76–85. Association for Computing Machinery, June 1998.

[67] Peter Su. *Efficient Parallel Algorithms for Closest Point Problems*. Ph.D. thesis, Dartmouth College, Hanover, New Hampshire, May 1994.

[68] Peter Su and Robert L. Scot Drysdale. *A Comparison of Sequential Delaunay Triangulation Algorithms*. Proceedings of the Eleventh Annual Symposium on Computational Geometry (Vancouver, British Columbia, Canada), pages 61–70. Association for Computing Machinery, June 1995.

[69] Joe F. Thompson and Nigel P. Weatherill. *Aspects of Numerical Grid Generation: Current Science and Art*. 1993.

[70] David F. Watson. *Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes*. Computer Journal **24**(2):167–172, 1981.

[71] Nigel P. Weatherill. *Delaunay Triangulation in Computational Fluid Dynamics*. Computers and Mathematics with Applications **24**(5/6):129–150, September 1992.

[72] Nigel P. Weatherill and O. Hassan. *Efficient Three-Dimensional Grid Generation using the Delaunay Triangulation*. Proceedings of the First European Computational Fluid Dynamics Conference (Brussels, Belgium) (Ch. Hirsch, J. Périaux, and W. Kordulla, editors), pages 961–968, September 1992.

[73] Nigel P. Weatherill, O. Hassan, D. L. Marcum, and M. J. Marchant. *Grid Generation by the Delaunay Triangulation*. Von Karman Institute for Fluid Dynamics 1993–1994 Lecture Series, 1994.

[74] M. A. Yerry and Mark S. Shephard. *A Modified Quadtree Approach to Finite Element Mesh Generation*. IEEE Computer Graphics and Applications **3**:39–46, January/February 1983.

[75] ———. *Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique*. International Journal for Numerical Methods in Engineering **20**:1965–1990, 1984.