Lecture Notes on Delaunay Mesh Generation

Jonathan Richard Shewchuk

February 5, 2012

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

Supported in part by the National Science Foundation under Awards CMS-9318163, ACI-9875170, CMS-9980063, CCR-0204377, CCF-0430065, CCF-0635381, and IIS-0915462, in part by the University of California Lab Fees Research Program, 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 Natural Sciences and Engineering Research Council of Canada under a 1967 Science and Engineering Scholarship, in part by gifts from the Okawa Foundation and the Intel Corporation, and in part by an Alfred P. Sloan Research Fellowship.

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

Contents

Chapter 1

Introduction

One of the central tools of scientific computing is the fifty-year old *finite element method*—a numerical method for approximating solutions to partial differential equations. The finite element method and its cousins, the finite volume method and the boundary element method, simulate physical phenomena including fluid flow, heat transfer, mechanical deformation, and electromagnetic wave propagation. They are applied heavily in industry and science for marvelously diverse purposes—evaluating pumping strategies for petroleum extraction, modeling the fabrication and operation of transistors and integrated circuits, optimizing the aerodynamics of aircraft and car bodies, and studying phenomena from quantum mechanics to earthquakes to black holes.

The aerospace engineer Joe F. Thompson, who commanded a multi-institutional mesh generation effort called the National Grid Project [124], wrote in 1992 that

> An essential element of the numerical solution of partial differential equations (PDEs) on general regions is the construction of a grid (mesh) on which to represent the equations in finite form. . . . [A]t present it can take orders of magnitude more man-hours to construct the grid than it does to perform and analyze the PDE solution on the grid. This is especially true now that PDE codes of wide applicability are becoming available, and grid generation has been cited repeatedly as being a major pacing item. The PDE codes now available typically require much less esoteric expertise of the knowledgeable user than do the grid generation codes.

Two decades later, meshes are still a recurring bottleneck. The *automatic mesh generation problem* is to divide a physical domain with a complicated geometry—say, an automobile engine, a human's blood vessels, or the air around an airplane—into small, simple pieces called *elements*, such as triangles or rectangles (for two-dimensional geometries) or tetrahedra or rectangular prisms (for three-dimensional geometries), as illustrated in Figure 1.1. Millions or billions of elements may be needed.

A mesh must satisfy nearly contradictory requirements: it must conform to the shape of the object or simulation domain; its elements may be neither too large nor too numerous; it may have to grade from small to large elements over a relatively short distance; and it must be composed of elements that are of the right shapes and sizes. "The right shapes" typically include elements that are nearly equilateral and equiangular, and typically exclude elements that are long and thin, e.g. shaped like a needle or a kite. However, some applications require *anisotropic* elements that are long and thin, albeit with specified orientations and eccentricities, to interpolate fields with anisotropic second derivatives or to model anisotropic physical phenomena such as laminar air flow over an airplane wing.

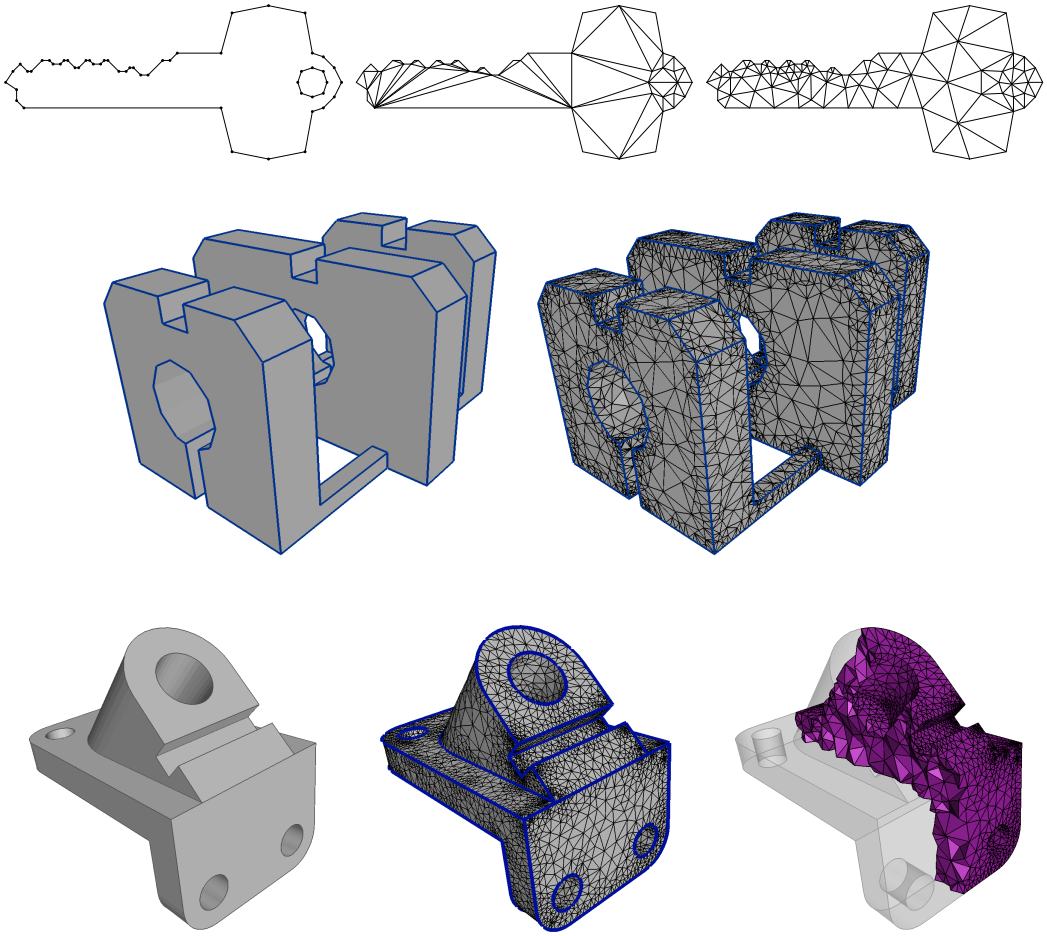


Figure 1.1: Finite element meshes of a polygonal, a polyhedral, and a curved domain. One mesh of the key has poorly shaped triangles and no Steiner points; the other has Steiner points and all angles between 30° and 120°. The cutaway view at lower right reveals some of the tetrahedral elements inside a mesh.

By my reckoning, the history of mesh generation falls into three periods, conveniently divided by decade. The pioneering work was done by researchers from several branches of engineering, especially mechanics and fluid dynamics, during the 1980s—though as we shall see, the earliest work dates back to at least 1970. This period brought forth most of the techniques used today: the Delaunay, octree, and advancing front methods for mesh generation, and mesh "clean-up" methods for improving an existing mesh. Unfortunately, nearly all the algorithms developed during this period are fragile, and produce unsatisfying meshes when confronted by complex domain geometries and stringent demands on element shape.

Around 1988, these problems attracted the interest of researchers in computational geometry, a branch of theoretical computer science. Whereas most engineers were satisfied with mesh generators that usually work for their chosen domains, computational geometers set a loftier goal: *provably good mesh generation*, the design of algorithms that are mathematically guaranteed to produce a satisfying mesh, even for domain geometries unimagined by the algorithm designer. This work flourished during the 1990s and continues to this day.

During the first decade of the 2000s, mesh generation became bigger than the finite element methods that gave birth to it. Computer animation uses triangulated surface models extensively, and the most novel new

ideas for using, processing, and generating meshes often debut at computer graphics conferences. In economic terms, the videogame and motion picture industries probably now exceed the finite element industries as users of meshes.

Meshes find heavy use in hundreds of other applications, such as aerial land surveying, image processing, geographic information systems, radio propagation analysis, shape matching, and population sampling. Mesh generation has become a truly interdisciplinary topic.

An excellent source for many aspects of mesh generation not covered by these notes is the *Handbook of Grid Generation* [125], which includes many chapters on the generation of structured meshes, chapters that describe advancing front methods in unusual detail by Peraire, Peiró, and Morgan [94] and Marcum [78], and a fine survey of quadrilateral and hexahedral meshing by Schneiders [105]. Further surveys of the mesh generation literature are supplied by Bern and Eppstein [8] and Thompson and Weatherill [126]. Boissonnat, Cohen-Steiner, Mourrain, Rote, and Vegter [18] survey algorithms for surface meshing. There is a large literature on how to numerically evaluate the quality of an element; see Field [50] for a survey.

1.1 Meshes and the Goals of Mesh Generation

Meshes are categorized according to their dimensionality and choice of elements. *Triangular meshes*, *tetrahedral meshes*, *quadrilateral meshes*, and *hexahedral meshes* are named according to the shapes of their elements. The two-dimensional elements—triangles and quadrilaterals—serve both in modeling two-dimensional domains and in *surface meshes* embedded in three dimensions, which are prevalent in computer graphics, boundary element methods, and simulations of thin plates and shells.

Tetrahedral elements are the simplest of all polyhedra, having four vertices and four triangular faces. Quadrilateral elements are four-sided polygons; their sides need not be parallel. Hexahedral elements are brick-like polyhedra, each having six quadrilateral faces, but their faces need not be parallel or even planar. These notes discuss only *simplicial meshes*—triangular and tetrahedral meshes—which are easier to generate than quadrilateral and hexahedral ones. For some applications, quadrilateral and hexahedral meshes offer more accurate interpolation and approximation. Non-simplicial elements sometimes make life easier for the numerical analyst; simplicial elements nearly always make life easier for the mesh generator. For topological reasons, hexahedral meshes can be extraordinarily difficult to generate for geometrically complicated domains.

Meshes are also categorized as structured or unstructured. A *structured mesh*, such as a regular cubical grid, or the triangular mesh at left in Figure 1.2, has the property that its vertices can be numbered so that simple arithmetic suffices to determine which vertices share an element with a selected vertex. These notes discuss only *unstructured meshes*, which entail explicitly storing each vertex's neighboring vertices or elements. All the meshes in Figure 1.1 are unstructured, as is the mesh at right in Figure 1.2. Structured meshes have been studied extensively [125]; they are suitable primarily for domains that have tractable geometries and do not require a strongly graded mesh. Unstructured meshes are much more versatile because of their ability to combine good element shapes with odd domain shapes and element sizes that grade from very small to very large.

For most applications, the elements comprising a mesh must intersect "nicely," meaning that if two elements intersect, their intersection is a vertex or edge or entire face of both. Formally, a mesh must be a *complex*, defined in Section 1.3. *Nonconforming elements* like those illustrated in Figure 1.3 rarely alleviate the underlying numerical problems, so they are rarely used in unstructured meshes.

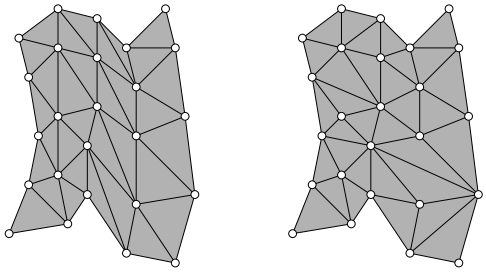


Figure 1.2: Structured vs. unstructured mesh.

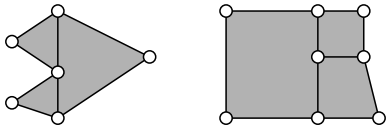


Figure 1.3: Nonconforming elements.

The goal of mesh generation is to create elements that *conform* to the shape of the geometric domain and meet constraints on their sizes and shapes. The next two sections discuss domain conformity and element quality.

1.1.1 Domain Conformity

Mesh generation algorithms vary in what domains they can mesh and how those domains are specified. The input to a mesh generator—particularly one in the theory literature—might be a simple polygon or polyhedron. Meshing becomes more difficult if the domain can have *internal boundaries* that no element is permitted to cross, such as a boundary between two materials in a heat transfer simulation. Meshing is substantially more difficult for domains that have curved edges and surfaces, called *ridges* and *patches*, which are typically represented as splines or subdivision surfaces. Each of these kinds of geometry requires a different definition of what it means to *triangulate* a domain. Let us consider these geometries in turn.

A polygon whose boundary is a closed loop of straight edges can be subdivided into triangles whose vertices all coincide with vertices of the polygon; see Section 2.8.1 for a proof of that fact. The set containing those triangles, their edges, and their vertices is called a *triangulation* of the polygon. But as the illustration at top center in Figure 1.1 illustrates, the triangles may be badly shaped. To mesh a polygon with only high-quality triangles, as illustrated at upper right in the figure, a mesh generator usually introduces additional vertices that are not vertices of the polygon. The added vertices are often called *Steiner points*, and the mesh is called a *Steiner triangulation* of the polygon.

Stepping into three dimensions, we discover that polyhedra can be substantially more difficult to triangulate than polygons. It comes as a surprise to learn that many polyhedra do not have triangulations, if a *triangulation* is defined to be a subdivision of a polyhedron into tetrahedra whose vertices are all vertices of the polyhedron. In other words, Steiner points are sometimes mandatory. See Section 4.5 for an example.

Internal boundaries exist to help apply boundary conditions for partial differential equations and to support discontinuities in physical properties, like differences in heat conductivity in a multi-material simulation. A boundary, whether internal or external, must be represented by a union of edges or faces of the mesh. Elements cannot cross boundaries, and where two materials meet, their meshes must have matching edges and faces. This requirement may seem innocuous, but it makes meshing much harder if the domain has small

angles. We define geometric structures called *piecewise linear complexes* to formally treat polygonal and polyhedral domains, like those at upper left and center left in Figure 1.1, in a manner that supports internal boundaries. Piecewise linear complexes and their triangulations are defined in Sections 2.8.1 and 4.5.1.

Curved domains introduce more difficulties. Some applications require elements that curve to match a domain. Others approximate a curved domain with a piecewise linear mesh at the cost of introducing inaccuracies in shape, finite element solutions, and surface normal vectors (which are important for computer graphics). In finite element methods, curved domains are sometimes approximated with elements whose faces are described by parametrized quadratic, cubic, bilinear, or trilinear patches. In these notes, the elements are always linear triangles and tetrahedra.

Domains like that at lower left in Figure 1.1 can be specified by geometric structures called *piecewise smooth complexes*. These complexes are composed of smoothly curved patches and ridges, but patches can meet nonsmoothly at ridges and vertices, and internal boundaries are permitted. A ridge where patches meet nonsmoothly is sometimes called a *crease*.

1.1.2 Element Quality

Most applications of meshes place constraints on both the shapes and sizes of the elements. These constraints come from several sources. First, large angles (near 180°) can cause large interpolation errors. In the finite element method, these errors induce a large *discretization error*—the difference between the computed approximation and the true solution of the PDE. Second, small angles (near 0°) can cause the stiffness matrices associated with the finite element method to be ill-conditioned. Small angles do not harm interpolation accuracy, and many applications can tolerate them. Third, smaller elements offer more accuracy, but cost more computationally. Fourth, small or skinny elements can induce instability in the explicit time integration methods employed by many time-dependent physical simulations. Consider these four constraints in turn.

The first constraint forbids large angles, including large plane angles in triangles and large dihedral angles in tetrahedra. Most applications of triangulations use them to interpolate a multivariate function whose true value might or might not be known. For example, a surveyor may know the altitude of the land at each point in a large sample, and use interpolation over a triangulation to approximate the altitude at points where readings were not taken. There are two kinds of *interpolation error* that matter for most applications: the difference between the interpolated function and the true function, and the difference between the gradient of the interpolated function and the gradient of the true function. Element shape is largely irrelevant for the first kind—the way to reduce interpolation error is to use smaller elements.

However, the error in the gradient depends on both the shapes and the sizes: it can grow arbitrarily large as an element's largest angle approaches 180° [122, 5, 65, 116], as Figure 1.4 illustrates. Three triangulations, each having 200 triangles, are used to render a paraboloid. The mesh of long thin triangles at right has no angle greater than 90°, and visually performs only slightly worse than the isotropic triangulation at left. The slightly worse performance is because of the longer edge lengths. However, the middle paraboloid looks like a washboard, because the triangles with large angles have very inaccurate gradients.

Figure 1.5 shows why this problem occurs. Let f be a function—perhaps some physical quantity like temperature—linearly interpolated on the illustrated triangle. The values of f at the vertices of the bottom edge are 35 and 65, so the linearly interpolated value of f at the center of the edge is 50. This value is independent of the value associated with the top vertex. As the angle at the upper vertex approaches 180°, the interpolated point with value 50 becomes arbitrarily close to the upper vertex with value 40. Hence, the interpolated gradient ∇f can become arbitrarily large, and is clearly specious as an approximation of the

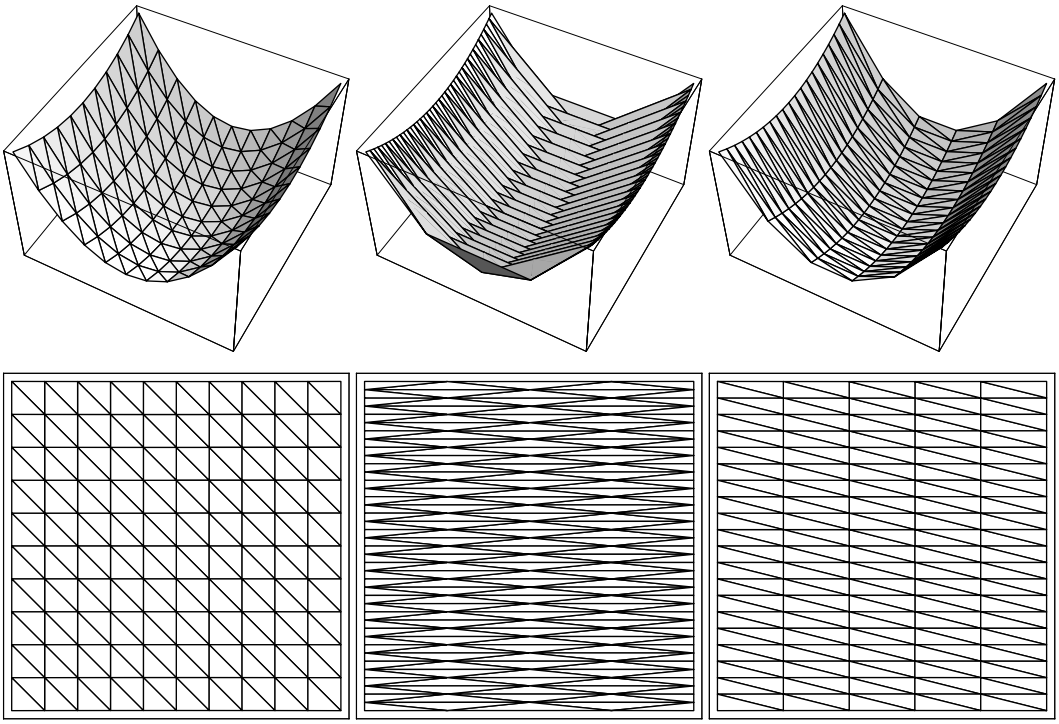


Figure 1.4: An illustration of how large angles, but not small angles, can ruin the interpolated gradients. Each triangulation uses 200 triangles to render a paraboloid.

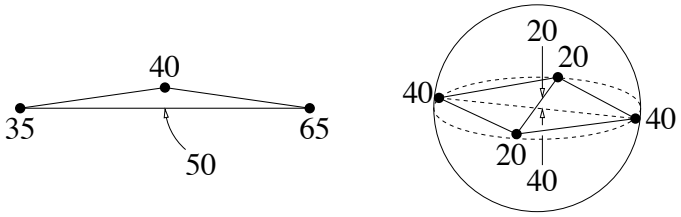


Figure 1.5: As the large angle of the triangle approaches 180°, or the sliver tetrahedron becomes arbitrarily flat, the magnitude of the interpolated gradient becomes arbitrarily large.

true gradient. The same effect is seen between two edges of a sliver tetrahedron that pass near each other, also illustrated in Figure 1.5.

In the finite element method, the discretization error is usually proportional to the error in the gradient, although the relationship between the two depends on the PDE and the order of the basis functions used to discretize it. In surface meshes for computer graphics, large angles cause triangles to have normal vectors that poorly approximate the normal to the true surface, and these can create visual artifacts in rendering.

For tetrahedral elements, usually it is their largest dihedral angles (defined in Section 1.5) that matter most [71, 116]. Nonconvex quadrilateral and hexahedral elements, with angles exceeding 180°, sabotage interpolation and the finite element method.

The second constraint on meshes is that many applications forbid small angles, although fewer than those that forbid large angles. If your application is the finite element method, then the eigenvalues of the stiffness matrix associated with the method ideally should be clustered as close together as possible. Matrices with poor eigenvalue spectra affect linear equation solvers by slowing down iterative methods and

introducing large roundoff errors into direct methods. The relationship between element shape and matrix conditioning depends on the PDE being solved and the basis functions and test functions used to discretize it, but as a rule of thumb, it is the small angles that are deleterious: the largest eigenvalue of the stiffness matrix approaches infinity as an element's smallest angle approaches zero [57, 7, 116]. Fortunately, most linear equation solvers cope well with a few bad eigenvalues.

The third constraint on meshes governs element size. Many mesh generation algorithms take as input not just the domain geometry, but also a space-varying *size field* that specifies the ideal size, and sometimes anisotropy, of an element as a function of its position in the domain. (The size field is often implemented by interpolation over a *background mesh*.) A large number of *fine* (small) elements may be required in one region where they are needed to attain good accuracy—often where the physics is most interesting, as amid turbulence in a fluid flow simulation—whereas other regions might be better served by *coarse* (large) elements, to keep their number small and avoid imposing an overwhelming computational burden on the application. 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 incur excessively large computational demands.

A *graded mesh* is one that has large disparities in element size. Ideally, a mesh generator should be able to *grade* from very small to very large elements over a short distance. However, overly aggressive grading introduces skinny elements in the transition region. The size field alone does not determine element size: mesh generators often create elements smaller than specified to maintain good element quality in a graded mesh, and to conform to small geometric features of a domain.

Given a coarse mesh—one with relatively few elements—it is typically easy to *refine* it, guided by the size field, to produce another mesh having a larger number of smaller elements. The reverse process is much harder. Hence, mesh generation algorithms often set themselves the goal of being able, in principle, to generate as coarse a mesh as possible.

The fourth constraint forbids unnecessarily small or skinny elements for time-dependent PDEs solved with explicit time integration methods. The stability of explicit time integration is typically governed by the *Courant–Friedrichs–Lewy condition* [41], which implies that the computational time step must be small enough that a wave or other time-dependent signal cannot cross more than one element per time step. Therefore, elements with short edges or short altitudes may force a simulation to take unnecessarily small time steps, at great computational cost, or risk introducing a large dose of spurious energy that causes the simulation to "explode."

Some meshing problems are impossible. A polygonal domain that has a corner bearing a 1° angle obviously cannot be meshed with triangles whose angles all exceed 30°; but suppose we merely ask that all angles be greater than 30° *except* the 1° angle? This request can always be granted for a polygon with no internal boundaries, but Figure 1.6 depicts a domain composed of two polygons glued together that, surprisingly, provably has no mesh whose *new* angles are all over 30° [112]. Simple polyhedra in three dimensions inherit this hurdle, even without internal boundaries. One of the biggest challenges in mesh generation is three-dimensional domains with small angles and internal boundaries, wherein an arbitrary number of ridges and patches can meet at a single vertex.

1.2 A Brief History of Mesh Generation

Three classes of mesh generation algorithms predominate nowadays: advancing front methods, wherein elements crystallize one by one, coalescing from the boundary of a domain to its center; grid, quadtree,

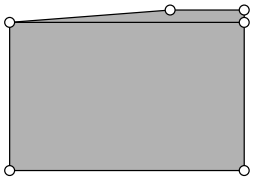


Figure 1.6: A mesh of this domain must have a *new* small angle.

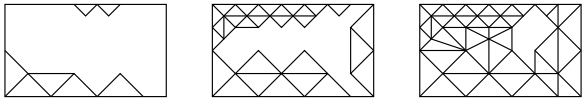


Figure 1.7: Advancing front mesh generation.

and octree algorithms, which overlay a structured background grid and use it as a guide to subdivide a domain; and Delaunay refinement algorithms, the subject of these notes. An important fourth class is mesh improvement algorithms, which take an existing mesh and make it better through local optimization. The few fully unstructured mesh generation algorithms that do not fall into one of these four categories are not yet in widespread use.

Automatic unstructured mesh generation for finite element methods began in 1970 with an article by C. O. Frederick, Y. C. Wong, and F. W. Edge entitled "Two-Dimensional Automatic Mesh Generation for Structural Analysis" in the *International Journal for Numerical Methods in Engineering* [53]. This startling paper describes, to the best of our knowledge, the first Delaunay mesh generation algorithm, the first advancing front method, and the first algorithm for Delaunay triangulations in the plane besides slow exhaustive search—all one and the same. The irony of this distinction is that the authors appear to have been unaware that the triangulations they create are Delaunay. Moreover, a careful reading of their paper reveals that their meshes are *constrained* Delaunay triangulations, a sophisticated variant of Delaunay triangulations discussed in Section 2.8.2. The paper is not well known, perhaps because it was two decades ahead of its time.

Advancing front methods construct elements one by one, starting from the domain boundary and advancing inward, as illustrated in Figure 1.7—or occasionally outward, as when meshing the air around an airplane. The frontier where elements meet unmeshed domain is called the *front*, which ventures forward until the domain is paved with elements and the front vanishes. Advancing front methods are characterized by exceptionally high quality elements at the domain boundary. The worst elements appear where the front collides with itself, and assuring their quality is difficult, especially in three dimensions; there is no literature on provably good advancing front algorithms. Advancing front methods have been particularly successful in fluid mechanics, because it is easy to place extremely anisotropic elements or specialized elements at the boundary, where they are needed to model phenomena such as laminar air flow.

Most early methods created vertices then triangulated them in two separate stages [53, 24, 76]. For instance, Frederick, Wong, and Edge [53] use "a magnetic pen to record node point data and a computer program to generate element data." The simple but crucial next insight—arguably, the "true" advancing front technique—was to interleave vertex creation with element creation, so the front can guide the placement of vertices. Alan George [58] took this step in 1971, but it was forgotten and reinvented in 1980 by Sadek [104] and again in 1987 by Peraire, Vahdati, Morgan, and Zienkiewicz [95], who also introduced support for anisotropic triangles. Soon thereafter, methods of this design appeared for tetrahedral meshing [77, 93], quadrilateral meshing [15], and hexahedral meshing [14, 119].

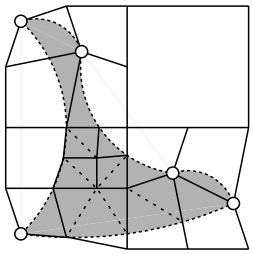


Figure 1.8: A quadtree mesh.

These notes are about provably good mesh generation algorithms that employ the *Delaunay triangulation*, a geometric structure possessed of mathematical properties uniquely well suited to creating good triangular and tetrahedral meshes. The defining property of a Delaunay triangulation in the plane is that no vertex of the triangulation lies in the interior of any triangle's *circumscribing disk*—the unique circular disk whose boundary touches the triangle's three vertices. In three dimensions, no vertex is enclosed by any tetrahedron's circumscribing sphere. Delaunay triangulations optimize several valuable geometric criteria, including some related to interpolation accuracy.

Delaunay refinement algorithms construct a Delaunay triangulation and refine it by inserting new vertices, chosen to eliminate skinny or oversized elements, while always maintaining the Delaunay property of the mesh. The key to ensuring good element quality is to prevent the creation of unnecessarily short edges. The Delaunay triangulation serves as a guide to finding locations to place new vertices that are far from existing ones, so that short edges and skinny elements are not created needlessly.

Most Delaunay mesh generators, unlike advancing front methods, create their worst elements near the domain boundary and their best elements in the interior. The early Delaunay mesh generators, like the early advancing front methods, created vertices and triangulated them in two separate stages [53, 25, 64]. The era of modern meshing began in 1987 with the insight, care of William Frey [56], to use the triangulation as a search structure to decide where to place the vertices. *Delaunay refinement* is the notion of maintaining a Delaunay triangulation while inserting vertices in locations dictated by the triangulation itself. The advantage of Delaunay methods, besides the optimality properties of the Delaunay triangulation, is that they can be designed to have mathematical guarantees: that they will always construct a valid mesh and, at least in two dimensions, that they will never produce skinny elements.

The third class of mesh generators is those that overlay a domain with a background grid whose resolution is small enough that each of its cells overlaps a very simple, easily triangulated portion of the domain, as illustrated in Figure 1.8. A variable-resolution grid, usually a quadtree or octree, yields a graded mesh. Element quality is usually assured by warping the grid so that no short edges appear when the cells are triangulated, or by improving the mesh afterward.

Grid meshers place excellent elements in the domain interior, but the elements near the domain boundary are worse than with other methods. Other disadvantages are the tendency for most mesh edges to be aligned in a few preferred directions, which may influence subsequent finite element solutions, and the difficulty of creating anisotropic elements that are not aligned with the grid. Their advantages are their speed, their ease of parallelism, the fact that some of them have mathematical guarantees, and most notably, their robustness for meshing imprecisely specified geometry and dirty CAD data. Mark Yerry and Mark Shephard published the first quadtree mesher in 1983 and the first octree mesher in 1984 [130, 131].

From nearly the beginning of the field, most mesh generation systems have included a mesh "clean-up" component that improves the quality of a finished mesh. Today, simplicial mesh improvement heuristics

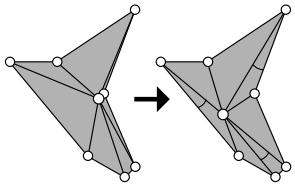


Figure 1.9: Smoothing a vertex to maximize the minimum angle.

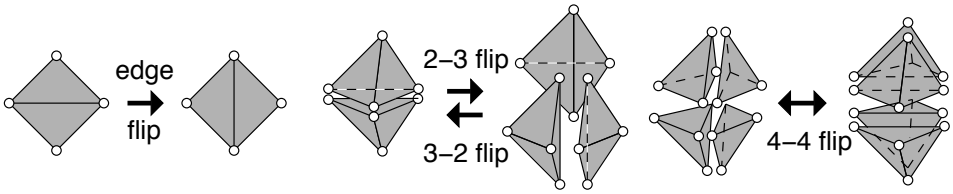


Figure 1.10: Bistellar flips.

offer by far the highest quality of all the methods, and excellent control of anisotropy. Their disadvantages are the requirement for an initial mesh and a lack of mathematical guarantees. (They can guarantee they will not make the mesh worse.)

The ingredients of a mesh improvement method are a set of local transformations, which replace small groups of tetrahedra with other tetrahedra of better quality, and a schedule that searches for opportunities to apply them. *Smoothing* is the act of moving a vertex to improve the quality of the elements adjoining it. Smoothing does not change the topology (connectivity) of the mesh. *Topological transformations* are operations that change the mesh topology by removing elements from a mesh and replacing them with a different configuration of elements occupying the same space.

Smoothing is commonly applied to each interior vertex of the mesh in turn, perhaps for several passes over the mesh. The simplest and most famous way to smooth an interior vertex is to move it to the centroid of the vertices that adjoin it. This method, which dates back at least to Kamel and Eisenstein [68] in 1970, is called *Laplacian smoothing* because of its interpretation as a Laplacian finite difference operator. It usually works well for triangular meshes, but it is unreliable for tetrahedra, quadrilaterals, and hexahedra.

More sophisticated optimization-based smoothers began to appear in the 1990s [91, 23, 90]. Slower but better smoothing is provided by the nonsmooth optimization algorithm of Freitag, Jones, and Plassmann [54], which can optimize the worst element in a group—for instance, maximizing the minimum dihedral angle among the tetrahedra that share a specified vertex. For some quality measures, optimal mesh smoothing can be done with generalized linear programming [1]. Figure 1.9 illustrates a smoothing step that maximizes the minimum angle among triangles.

The simplest topological transformation is the *edge flip* in a triangular mesh, which replaces two triangles with two different triangles. Figure 1.10 also illustrates several analogous transformations for tetrahedra, which mathematicians call *bistellar flips*. There are analogous transformations for tetrahedra, quadrilaterals, and hexahedra. Similar flips exist for quadrilaterals and hexahedra; see Bern, Eppstein, and Erickson [9] for a list.

Mesh improvement is usually driven by a schedule that searches the mesh for elements that can be improved by local transformations, ideally as quickly as possible. Canann, Muthukrishnan, and Phillips [22] provide a fast triangular mesh improvement schedule. Sophisticated schedules for tetrahedral mesh improvement are provided by Joe [67], Freitag and Ollivier-Gooch [55], and Klingner and Shewchuk [70]. For

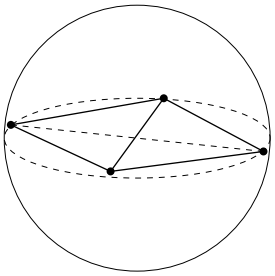


Figure 1.11: The mesh generator's nemesis: a sliver tetrahedron.

a list of flips for quadrilateral and hexahedral meshes, see Bern, Eppstein, and Erickson [9]. Kinney [69] describes mesh improvement methods for quadrilateral meshes. There does not seem to have been much work on applying hexahedral flips.

The story of provably good mesh generation is an interplay of ideas between Delaunay methods and methods based on grids, quadtrees, and octrees. The first provably good mesh generation algorithm, by Baker, Grosse, and Rafferty [6] in 1988, employs a square grid. The first provably good Delaunay refinement algorithm in the plane, by Chew [35], followed the next year. The first provably good three-dimensional Delaunay refinement algorithm is by Dey, Bajaj, and Sugihara [45]. Although their algorithm is guaranteed to eliminate most types of bad tetrahedra, a few bad tetrahedra slip through: a type of tetrahedron called a *sliver* or *kite*.

The canonical sliver is formed by arranging four vertices around the equator of a sphere, equally spaced, then perturbing one of the vertices slightly off the equator, as Figure 1.11 illustrates. A sliver can have dihedral angles arbitrarily close to 0° and 180° yet have no edge that is particularly short. Provably good sliver removal is one of the most difficult theoretical problems in mesh generation, although mesh improvement algorithms beat slivers consistently in practice.

None of the provably good algorithms discussed above produce graded meshes. The first mesh generator offering provably good grading is the 1990 quadtree algorithm of Bern, Eppstein, and Gilbert [10], which meshes a polygon so no new angle is less than 18.4°. It has been influential in part because the meshes it produces are not only graded, but *size-optimal*: the number of triangles in a mesh is at most a constant factor times the number in the smallest possible mesh (measured by triangle count) having no angle less than 18.4°. Ironically, the algorithm produces too many triangles to be practical—but only by a constant factor. Neugebauer and Diekmann [88] improve the algorithm by replacing square quadrants with rhomboids.

A groundbreaking 1992 paper by Jim Ruppert [100, 102] on triangular meshing brought guaranteed good grading and size optimality to Delaunay refinement algorithms. Ruppert's algorithm, described in Chapter 6, accepts nonconvex domains with internal boundaries and produces graded meshes of modest size and high quality in practice.

The first tetrahedral mesh generator offering size optimality is the 1992 octree algorithm of Mitchell and Vavasis [84]. Remarkably, Mitchell and Vavasis [85] extended their mathematical guarantees to meshes of polyhedra of any dimensionality by using d-dimensional 2^d-trees. Shewchuk [113, 114] generalized the tetrahedral Delaunay refinement algorithm of Dey, Bajaj, and Sugihara from convex polyhedra to piecewise linear complexes; the algorithm appears in Chapter 7.

The first provably good meshing algorithm for curved surfaces in three dimensions is by Chew [37]; see the aforementioned survey by Boissonnat et al. [18] for a discussion of subsequent algorithms. Guaranteed-quality triangular mesh generators for two-dimensional domains with curved boundaries include those by

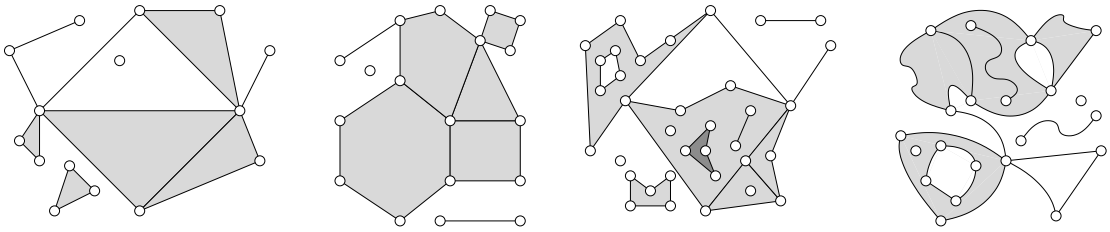


Figure 1.12: From left to right, a simplicial complex, a polyhedral complex, a piecewise linear complex, and a piecewise smooth complex. The shaded areas are triangles, convex polygons, linear 2-cells, and smooth 2-cells, respectively. In the piecewise linear complex, observe that several linear cells have holes, one of which is filled by another linear cell (darkly shaded).

Boivin and Ollivier-Gooch [19] and Pav and Walkington [92]. Labelle and Shewchuk [72] provide a provably good triangular mesh generator that produces anisotropic meshes in the plane, and Cheng, Dey, Ramos, and Wenger [32] generalize it to generate anisotropic meshes of curved surfaces in three-dimensional space.

1.3 Simplices, Complexes, and Polyhedra

Tetrahedra, triangles, edges, and vertices are instances of *simplices*. In these notes, I represent meshes and the domains we wish to mesh as *complexes*. There are several different types of complexes, illustrated in Figure 1.12, which all share two common properties. First, a complex is a set that contains not only volumes such as tetrahedra, but also the faces, edges, and vertices of those volumes. Second, the cells in a complex must intersect each other according to specified rules, which depend on the type of complex.

The simplest type of complex is a *simplicial complex*, which contains only simplices. The mesh generation algorithms in these notes produce simplicial complexes. More general are *polyhedral complexes*, composed of convex polyhedra; these "polyhedra" can be of any dimension from zero on up. The most important polyhedral complexes for mesh generation are the famous *Voronoi diagram* and the *Delaunay subdivision*, defined in Section 2.2.

Theorists use two other kinds of complexes to specify domains to be triangulated. *Piecewise linear complexes*, defined in Sections 2.8.1 and 4.5.1, differ from polyhedral complexes by permitting nonconvex polyhedra and by relaxing the rules of intersection of those polyhedra. *Piecewise smooth complexes*, introduced by Cheng, Dey, and Ramos [31] generalize straight edges and flat facets to curved ridges and patches.

To a mathematician, a "triangle" is a set of points, which includes all the points inside the triangle as well as the points on the three edges. Likewise, a polyhedron is a set of points covering its entire volume. A complex is a set of sets of points. We define these and other geometric structures in terms of affine hulls and convex hulls. Simplices, convex polyhedra, and their faces are convex sets of points. A point set C is *convex* if for every pair of points $p, q \in C$, the line segment pq is included in C.

Definition 1 (affine hull)**.** *Let $X = \{x_1, x_2, \ldots, x_k\}$ be a set of points in $\mathbb{R}^d$. A point p is an* affine combination *of the points in X if it can be written $p = \sum_{i=1}^{k} w_i x_i$ for a set of scalar* weights *w_i such that $\sum_{i=1}^{k} w_i = 1$. A point p is* affinely independent *of X if it is not an affine combination of points in X. The points in X are* affinely independent *if no point in X is an affine combination of the others. In $\mathbb{R}^d$, no more than $d+1$ points can be affinely independent. The* affine hull *of X, denoted* aff *X, is the set of all affine combinations of points in X, as illustrated in Figure 1.13. A k-*flat*, also known as an* affine subspace*, is the affine hull of $k+1$*

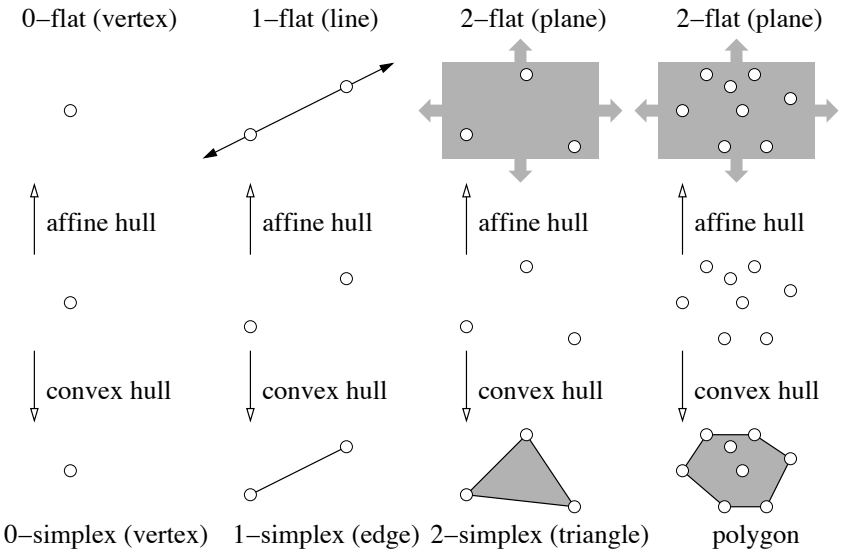


Figure 1.13: Examples of affine hulls and convex hulls in the plane.

affinely independent points; so a 0*-flat is a vertex, a* 1*-flat is a line, a* 2*-flat is a plane, etc. A* $(d-1)$*-flat in* $\mathbb{R}^d$ *is called a* hyperplane. *A k-flat is said to have* dimension k.

Definition 2 (convex hull)**.** *A point p is a* convex combination *of the points in X if it can be written as an affine combination with all the weights nonnegative; i.e.* $w_i \geq 0$ *for all i. The* convex hull *of X, denoted* conv X*, is the set of all convex combinations of points in X, as illustrated in Figure 1.13. An alternative definition is that* conv X *is the most exclusive convex point set such that* $X \subseteq \text{conv } X$.

Simplices and convex polyhedra are convex hulls of finite point sets, with k-simplices being the simplest possible k-dimensional polyhedra. One way that mathematical language deviates from lay usage is that a "face" of a polyhedron can be of any dimension; mathematicians use "facet" to denote what a layman calls a "face."

Definition 3 (simplex)**.** *A* k-simplex τ *is the convex hull of a set X of* $k+1$ *affinely independent points. In particular, a* 0*-simplex is a* vertex*, a* 1*-simplex is an* edge*, a* 2*-simplex is a* triangle*, and a* 3*-simplex is a* tetrahedron. *A k-simplex is said to have* dimension k. *A simplex is a* face *of* τ *if it is the convex hull of a nonempty subset of X. Faces of* τ *come in all dimensions from zero*[1] *(*τ*'s vertices) to k;* τ *is a face of* τ*. A simplex is a* proper face *of* τ *if it is the convex hull of a proper subset of X; i.e. any face except* τ*. In particular, the* $(k-1)$*-faces of* τ *are called* facets *of* τ*;* τ *has* $k+1$ *facets. For instance, the facets of a tetrahedron are its four triangular faces.*

Definition 4 (simplicial complex)**.** *A* simplicial complex $\mathcal{T}$*, also known as a* triangulation*, is a set containing finitely*[2] *many simplices that satisfies the following two restrictions.*

- $\mathcal{T}$ *contains every face of every simplex in* $\mathcal{T}$.

[1] Some writers use the convention that the empty set is a simplex of dimension -1 and a face of every simplex, albeit not a proper face. I make no use of this convention.

[2] Topologists usually define complexes so they have countable cardinality. I restrict complexes to finite cardinality to avoid some interesting quirks, like the possibility that a polygon with a 1° angle can be meshed with a countably infinite set of triangles having no angle less than 20°.

- *For any two simplices $\sigma, \tau \in \mathcal{T}$, their intersection $\sigma \cap \tau$ is either empty or a face of both σ and τ.*

Convex polyhedra are as easy to define as simplices, but their faces are trickier. Whereas the convex hull of a subset of a simplex's vertices is a face of the simplex, the convex hull of an arbitrary subset of a cube's vertices is usually not a face of the cube. The faces of a polyhedron are defined below in terms of *supporting hyperplanes*; observe that this definition is consistent with the definition of a face of a simplex above.

Definition 5 (convex polyhedron)**.** *A* convex polyhedron *is the convex hull of a finite point set. A polyhedron whose affine hull is a k-flat is called a* k-polyhedron *and is said to have* dimension k*. A* 0*-polyhedron is a vertex, a* 1*-polyhedron is an edge, and a* 2*-polyhedron is a* polygon*. The* proper faces *of a convex polyhedron C are the polyhedra that can be generated by taking the intersection of C with a hyperplane that intersects C's boundary but not C's interior; such a hyperplane is called a* supporting hyperplane *of C. For example, the proper faces of a cube are six squares, twelve edges, and eight vertices. The* faces *of C are the proper faces of C and C itself. The* facets *of a k-polyhedron are its $(k-1)$-faces.*

A polyhedral complex imposes exactly the same restrictions as a simplicial complex.

Definition 6 (polyhedral complex)**.** *A* polyhedral complex *$\mathcal{P}$ is a set containing finitely many convex polyhedra that satisfies the following two restrictions.*

- *$\mathcal{P}$ contains every face of every polyhedron in $\mathcal{P}$.*
- *For any two polyhedra $C, D \in \mathcal{P}$, their intersection $C \cap D$ is either empty or a face of both C and D.*

Piecewise linear complexes are sets of polyhedra that are not necessarily convex. I call these polyhedra *linear cells*.

Definition 7 (linear cell)**.** *A* linear k-cell *is the union of a finite number of convex k-polyhedra, all included in some common k-flat. A linear* 0*-cell is a vertex, a linear* 2*-cell is sometimes called a* polygon*, and a linear* 3*-cell is sometimes called a* polyhedron*.*

For $k \geq 1$, a linear k-cell can have multiple connected components. These do no harm; removing a linear cell from a complex and replacing it with its connected components, or vice versa, makes no material difference. To simplify the exposition, I will forbid disconnected linear 1-cells in complexes; i.e. the only linear 1-cells are edges. For $k \geq 2$, a linear cell can be only tenuously connected; e.g. a union of two squares that intersect at a single point is a linear 2-cell, even though it is not a simple polygon.

Another difference between linear cells and convex polyhedra is that we define the faces of a linear cell in a fundamentally different way that supports configurations like those in Figures 1.3 and 1.12. A linear cell's faces are not an intrinsic property of the linear cell alone, but depend on the complex that contains it. I defer the details to Section 2.8.1, where I define piecewise linear complexes.

Piecewise smooth complexes are sets of cells called *smooth cells*, which are similar to linear cells except that they are not linear, but are smooth manifolds.

A complex or a mesh is a representation of a domain. The former is a set of sets of points, and the latter is a set of points. The following operator collapses the former to the latter.

Definition 8 (underlying space)**.** *The* underlying space *of a complex $\mathcal{P}$, denoted $|\mathcal{P}|$, is the union of its cells; that is, $|\mathcal{P}| = \bigcup_{C \in \mathcal{P}} C$.*

Ideally, a complex provided as input to a mesh generation algorithm and the mesh produced as output should cover exactly the same points. This ideal is not always possible—for example, if we are generating a linear tetrahedral mesh of a curved domain. When it is achieved, we call it *exact conformity*.

Definition 9 (exact conformity)**.** *A complex $\mathcal{T}$* exactly conforms *to a complex $\mathcal{P}$ if $|\mathcal{T}| = |\mathcal{P}|$ and every cell in $\mathcal{P}$ is a union of cells in $\mathcal{T}$. We also say that $\mathcal{T}$ is a* subdivision *of $\mathcal{P}$.*

1.4 Metric Space Topology

This section introduces basic notions from point set topology that underlie triangulations and other complexes. They are prerequisites for more sophisticated topological ideas—manifolds and homeomorphisms—introduced in Sections 1.6 and 1.7. A complex of linear elements cannot exactly conform to a curved domain, which raises the question of what it means for a triangulation to be a mesh of such a domain. To a layman, the word topology evokes visions of "rubber-sheet topology": the idea that if you bend and stretch a sheet of rubber, it changes shape but always preserves the underlying structure of how it is connected to itself. Homeomorphisms offer a rigorous way to state that a mesh preserves the topology of a domain.

Topology begins with a set $\mathbb{T}$ of points—perhaps the points comprising the d-dimensional Euclidean space $\mathbb{R}^d$, or perhaps the points on the surface of a volume such as a coffee mug. We suppose that there is a *metric* $d(p, q)$ that specifies the scalar *distance* between every pair of points $p, q \in \mathbb{T}$. In the Euclidean space $\mathbb{R}^d$ we choose the Euclidean distance. On the surface of the coffee mug, we could choose the Euclidean distance too; alternatively, we could choose the *geodesic distance*, namely the length of the shortest path from p to q on the mug's surface.

Let us briefly review the Euclidean metric. We write points in $\mathbb{R}^d$ as $p = (p_1, p_2, \ldots, p_d)$, where each p_i is a real-valued *coordinate*. The *Euclidean norm* of a point $p \in \mathbb{R}^d$ is $\|p\| = \left(\sum_{i=1}^{d} p_i^2\right)^{1/2}$, and the *Euclidean distance* between two points $p, q \in \mathbb{R}^d$ is $d(p, q) = \|p - q\| = \left(\sum_{i=1}^{d} (p_i - q_i)^2\right)^{1/2}$. I also use the notation $d(\cdot, \cdot)$ to express minimum distances between point sets $P, Q \subseteq \mathbb{T}$,

$$\begin{aligned} d(p, Q) &= \inf\{d(p, q) : q \in Q\} \text{ and} \\ d(P, Q) &= \inf\{d(p, q) : p \in P, q \in Q\}. \end{aligned}$$

The heart of topology is the question of what it means for a set of points—say, a squiggle drawn on a piece of paper—to be *connected*. After all, two distinct points cannot be adjacent to each other; they can only be connected to another by an uncountably infinite bunch of intermediate points. Topologists solve that mystery with the idea of *limit points*.

Definition 10 (limit point)**.** *Let $Q \subseteq \mathbb{T}$ be a point set. A point $p \in \mathbb{T}$ is a* limit point *of Q, also known as an* accumulation point *of Q, if for every real number $\epsilon > 0$, however tiny, Q contains a point $q \neq p$ such that that $d(p, q) < \epsilon$.*

In other words, there is an infinite sequence of points in Q that get successively closer and closer to p—without actually being p—and get arbitrarily close. Stated succinctly, $d(p, Q \setminus \{p\}) = 0$. Observe that it doesn't matter whether $p \in Q$ or not.

Definition 11 (connected)**.** *Let $Q \subseteq \mathbb{T}$ be a point set. Imagine coloring every point in Q either red or blue. Q is* disconnected *if there exists a coloring having at least one red point and at least one blue point, wherein no red point is a limit point of the blue points, and no blue point is a limit point of the red points. A disconnected point set appears at left in Figure 1.14. If no such coloring exists, Q is* connected*, like the point set at right in Figure 1.14.*

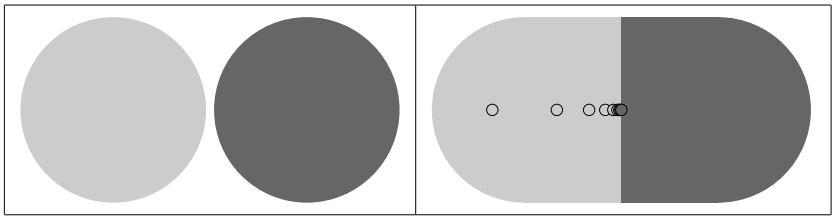


Figure 1.14: The disconnected point set at left can be partitioned into two connected subsets, which are colored differently here. The point set at right is connected. The dark point at its center is a limit point of the lightly colored points.

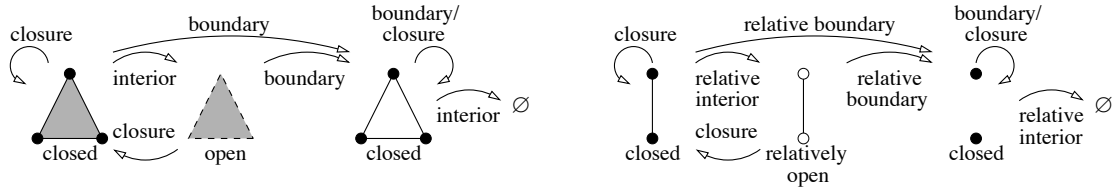


Figure 1.15: Closed, open, and relatively open point sets in the plane. Dashed edges and open circles indicate points missing from the point set.

In these notes, I frequently distinguish between closed and open point sets. Informally, a triangle in the plane is *closed* if it contains all the points on its edges, and *open* if it excludes all the points on its edges, as illustrated in Figure 1.15. The idea can be formally extended to any point set.

Definition 12 (closure)**.** *The* closure *of a point set* $Q \subseteq \mathbb{T}$*, denoted* $\operatorname{Cl} Q$*, is the set containing every point in* Q *and every limit point of* Q*. A point set* Q *is* closed *if* $Q = \operatorname{Cl} Q$*, i.e.* Q *contains all its limit points. The* complement *of a point set* Q *is* $\mathbb{T} \setminus Q$*. A point set* Q *is* open *if its complement is closed, i.e.* $\mathbb{T} \setminus Q = \operatorname{Cl}(\mathbb{T} \setminus Q)$.

For example, let $(0, 1)$ denote an *open interval* on the real number line—the set containing every $r \in \mathbb{R}$ such that $r > 0$ and $r < 1$, and let $[0, 1]$ denote a *closed interval* $(0, 1) \cup \{0\} \cup \{1\}$. The numbers zero and one are both limit points of the open interval, so $\operatorname{Cl}(0, 1) = [0, 1] = \operatorname{Cl}[0, 1]$. Therefore, $[0, 1]$ is closed and $(0, 1)$ is not. The numbers zero and one are also limit points of the *complement* of the closed interval, $\mathbb{R} \setminus [0, 1]$, so $(0, 1)$ is open, but $[0, 1]$ is not.

The terminology is misleading because "closed" and "open" are not opposites. In every nonempty metric space $\mathbb{T}$, there are at least two point sets that are both closed and open: $\emptyset$ and $\mathbb{T}$. The interval $(0, 1]$ on the real number line is neither open nor closed.

The definition of *open set* hides a subtlety that often misleads newcomers to point set topology: a triangle τ that is open in the metric space $\operatorname{aff} \tau$ is not open in the metric space $\mathbb{R}^3$. Every point in τ is a limit point of $\mathbb{R}^3 \setminus \tau$, because you can find sequences of points that approach τ from the side. In recognition of this quirk, a simplex $\sigma \subset \mathbb{R}^d$ is said to be *relatively open* if it is open relative to its affine hull. It is commonplace to abuse terminology by writing "open simplex" for a simplex that is only relatively open, and I follow this convention in these notes. Particularly useful is the concept of an "open edge," an edge that is missing its endpoints, illustrated in Figure 1.15.

Informally, the boundary of a point set Q is the set of points where Q meets its complement $\mathbb{T} \setminus Q$. The interior of Q contains all the other points of Q. Limit points provide formal definitions.

Definition 13 (boundary; interior)**.** *The* boundary *of a point set* Q *in a metric space* $\mathbb{T}$*, denoted* $\operatorname{Bd} Q$*, is*

the intersection of the closures of Q and its complement; i.e. $\operatorname{Bd} Q = \operatorname{Cl} Q \cap \operatorname{Cl}(\mathbb{T} \setminus Q)$. *The* interior *of Q, denoted* $\operatorname{Int} Q$, *is* $Q \setminus \operatorname{Bd} Q = Q \setminus \operatorname{Cl}(\mathbb{T} \setminus Q)$.

For example, $\operatorname{Bd}[0,1] = \{0,1\} = \operatorname{Bd}(0,1)$ and $\operatorname{Int}[0,1] = (0,1) = \operatorname{Int}(0,1)$. The boundary of a triangle (closed or open) in the Euclidean plane is the union of the triangle's three edges, and its interior is an open triangle, illustrated in Figure 1.15. The terms *boundary* and *interior* have the same misleading subtlety as open sets: the boundary of a triangle embedded in $\mathbb{R}^3$ is the whole triangle, and its interior is the empty set. Therefore, the *relative boundary* and *relative interior* of a simplex are its boundary and interior relative to its affine hull rather than the entire Euclidean space. Again, I often abuse terminology by writing "boundary" for relative boundary and "interior" for relative interior.

Definition 14 (bounded; compact)**.** *The* diameter *of a point set Q is* $\sup_{p,q \in Q} d(p,q)$. *The set Q is* bounded *if its diameter is finite, or* unbounded *if its diameter is infinite. A point set Q in a metric space is* compact *if it is closed and bounded.*

Besides simplices and polyhedra, the point sets we use most in these notes are balls and spheres.

Definition 15 (Euclidean ball)**.** *In $\mathbb{R}^d$, the* Euclidean d-ball *with center c and radius r, denoted $B(c,r)$, is the point set $B(c,r) = \{p \in \mathbb{R}^d : d(p,c) \leq r\}$. A 1-ball is an edge, and a 2-ball is sometimes called a* disk*. A* unit ball *is a ball with radius* 1*. The boundary of the d-ball is called the* Euclidean $(d-1)$-sphere *and denoted $S(c,r) = \{p \in \mathbb{R}^d : d(p,c) = r\}$. For example, a circle is a 1-sphere, and a layman's "sphere" in $\mathbb{R}^3$ is a 2-sphere. If we remove the boundary from a ball, we have the* open Euclidean d-ball $B_o(c,r) = \{p \in \mathbb{R}^d : d(p,c) < r\}$.

The foregoing text introduces point set topology in terms of metric spaces. Surprisingly, it is possible to define all the same concepts without the use of a metric, point coordinates, or any scalar values at all. *Topological spaces* are a mathematical abstraction for representing the topology of a point set while excluding all information that is not topologically essential. In these notes, all the topological spaces have metrics.

1.5 How to Measure an Element

Here, I describe ways to measure the size, angles, and quality of a simplicial element, and I introduce some geometric structures associated with simplices—most importantly, their circumspheres and circumcenters.

Definition 16 (circumsphere)**.** *Let τ be a simplex embedded in $\mathbb{R}^d$. A* circumsphere*, or* circumscribing sphere*, of τ is a $(d-1)$-sphere whose boundary passes through every vertex of τ, illustrated in Figure 1.16. A* circumball*, or* circumscribing ball*, of τ is a d-ball whose boundary is a circumsphere of τ. A* closed circumball *includes its boundary—the circumsphere—and an* open circumball *excludes it. If τ is a k-simplex, the k-*circumball *of τ is the unique k-ball whose boundary passes through every vertex of τ, and its relative boundary is the* $(k-1)$-circumsphere *of τ. I sometimes call a 2-circumball a* circumdisk *and a 1-circumsphere a* circumcircle.

If τ is a d-simplex in $\mathbb{R}^d$, it has one unique circumsphere and circumball; but if τ has dimension less than d, it has an infinite set of circumspheres and circumballs. Consider a triangle τ in $\mathbb{R}^3$, for example. There is only one circumcircle of τ, which passes through τ's three vertices, but τ has infinitely many circumspheres, and the intersection of any of those circumspheres with τ's affine hull is τ's circumcircle. The smallest of these circumspheres is special, because its center lies on τ's affine hull, it has the same radius as τ's circumcircle, and τ's circumcircle is its equatorial cross-section. Call τ's smallest circumcircle, illustrated in Figure 1.17, its *diametric circle*; and call τ's smallest circumdisk its *diametric disk*.

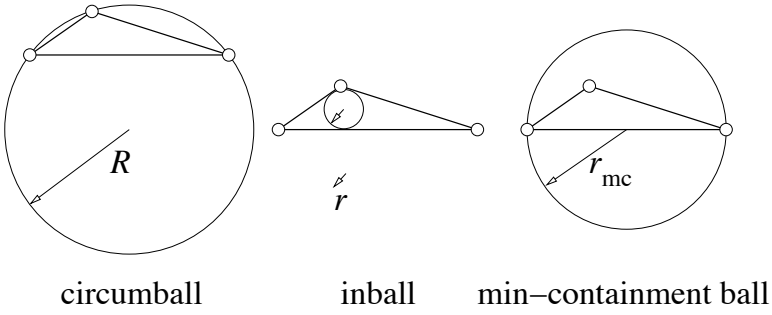


Figure 1.16: Three spheres associated with a triangle.

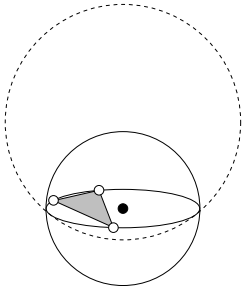


Figure 1.17: A triangle, two circumspheres of the triangle of which the smaller (solid) is the triangle's diametric sphere, the triangle's circumcircle (the equatorial cross-section of the diametric sphere), and the triangle's circumcenter.

Definition 17 (diametric sphere)**.** *The* diametric sphere *of a simplex τ is the circumsphere of τ with the smallest radius, and the* diametric ball *of τ is the circumball of τ with the smallest radius, whose boundary is the diametric sphere. The* circumcenter *of τ is the point at the center of τ's diametric sphere, which always lies on* aff τ*. The* circumradius *of τ is the radius of τ's diametric sphere.*

The significance of circumcenters in Delaunay refinement algorithms is that the best place to insert a new vertex into a mesh is often at the circumcenter of a poorly shaped element, domain boundary triangle, or domain boundary edge. In a Delaunay mesh, these circumcenters are locally far from other mesh vertices, so inserting them does not create overly short edges.

Other spheres associated with simplicial elements are the insphere and the min-containment sphere, both illustrated in Figure 1.16.

Definition 18 (insphere)**.** *The* inball*, or* inscribed ball*, of a k-simplex τ is the largest k-ball $B \subset \tau$. Observe that B is tangent to every facet of τ. The* insphere *of τ is the boundary of B, the* incenter *of τ is the point at the center of B, and the* inradius *of τ is the radius of B.*

Definition 19 (min-containment sphere)**.** *The* min-containment ball*, or* minimum enclosing ball*, of a k-simplex τ is the smallest k-ball $B \supset \tau$. The min-containment ball is always a diametric ball of a face of τ. The* min-containment sphere *of τ is the boundary of B.*

Finite element practitioners often represent the size of an element by the length of its longest edge, but one could argue that the radius of its min-containment sphere is a slightly better measure, because there are sharp error bounds for piecewise linear interpolation over simplicial elements that are directly proportional to the squares of the radii of their min-containment spheres. Details appear in Section 4.4.

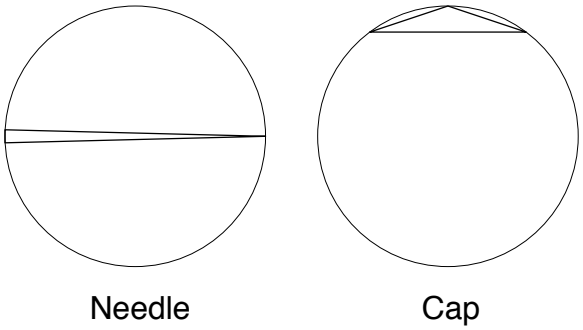


Figure 1.18: Skinny triangles have circumdisks larger than their shortest edges.

A *quality measure* is a mapping from elements to scalar values that estimates the suitability of an element's shape independently of its size. The most obvious quality measures of a triangle are its smallest and largest angles, and a tetrahedron can be judged by its dihedral angles. I denote the angle between two vectors $\mathbf{u}$ and $\mathbf{v}$ as

$$\angle(\mathbf{u}, \mathbf{v}) = \arccos \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}.$$

I compute an angle $\angle xyz$ of a triangle as $\angle(x - y, z - y)$.

A *dihedral angle* is a measure of the angle separating two planes or polygons in $\mathbb{R}^3$—for example, the facets of a tetrahedron or 3-polyhedron. Suppose that two flat facets meet at an edge yz, where y and z are points in $\mathbb{R}^3$. Let w be a point lying on one of the facets, and let x be a point lying on the other. It is helpful to imagine the tetrahedron $wxyz$. The dihedral angle separating the two facets is the same angle separating $\triangle wyz$ and $\triangle xyz$, namely $\angle(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} = (y - w) \times (z - w)$ and $\mathbf{v} = (y - x) \times (z - x)$ are vectors normal to $\triangle wyz$ and $\triangle xyz$.

Elements can go bad in different ways, and it is useful to distinguish types of skinny elements. There are two kinds of skinny triangles, illustrated in Figure 1.18: needles, which have one edge much shorter than the others, and caps, which have an angle near 180° and a large circumdisk. Figure 1.19 offers a taxonomy of types of skinny tetrahedra. The tetrahedra in the top row are skinny in one dimension and fat in two. Those in the bottom row are skinny in two dimensions and fat in one. Spears, spindles, spades, caps, and slivers have a dihedral angle near 180°; the others may or may not. Spikes, splinters, and all the tetrahedra in the top row have a dihedral angle near 0°; the others may or may not. The cap, which has a vertex quite close to the center of the opposite triangle, is notable for a large solid angle, near 360°. Spikes also can have a solid angle arbitrarily close to 360°, and all the skinny tetrahedra can have a solid angle arbitrarily close to zero.

There are several surprises. The first is that spires, despite being skinny, can have all their dihedral angles between 60° and 90°, even if two edges are separated by a plane angle near 0°. Spires with good dihedral angles are harmless in many applications, and are indispensable at the tip of a needle-shaped domain, but some applications eschew them anyway. The second surprise is that a spear or spindle tetrahedron can have a dihedral angle near 180° without having a small dihedral angle. By contrast, a triangle with an angle near 180° must have an angle near 0°.

For many purposes—mesh improvement, for instance—it is desirable to have a single quality measure that punishes both angles near 0° and angles near 180°, and perhaps spires as well. Most quality measures are designed to reach one extreme value for an equilateral triangle or tetrahedron, and an opposite extreme value for a degenerate element—a triangle whose vertices are collinear, or a tetrahedron whose vertices are coplanar. In these notes, the most important quality measure is the *radius-edge ratio*, because it is

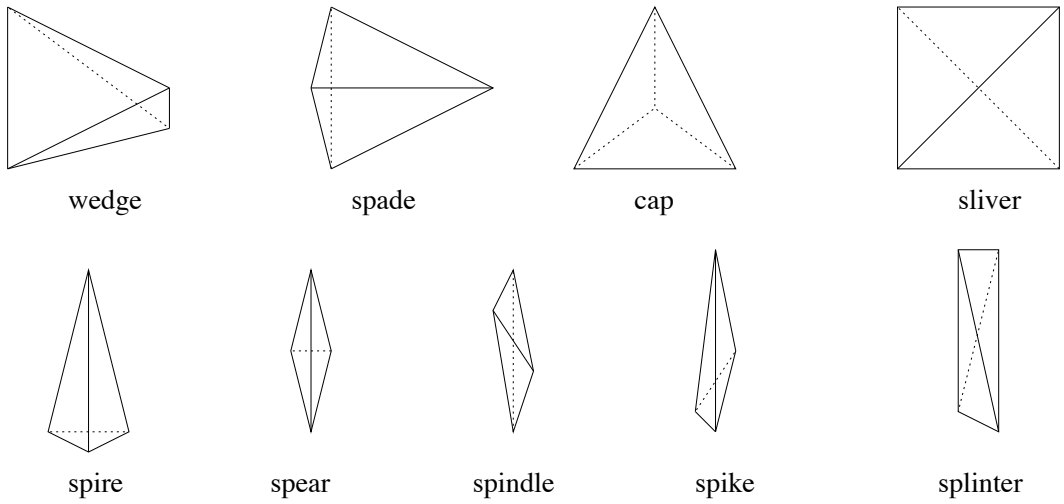


Figure 1.19: A taxonomy of skinny tetrahedra, adapted from Cheng, Dey, Edelsbrunner, Facello, and Teng [29].

naturally bounded by Delaunay refinement algorithms (a fact first pointed out by Miller, Talmor, Teng, and Walkington [81]).

Definition 20 (radius-edge ratio)**.** *The* radius-edge ratio *of a simplex τ is $R/\ell_{\min}$, where R is τ's circumradius and $\ell_{\min}$ is the length of its shortest edge.*

We would like the radius-edge ratio to be as small as possible; it ranges from ∞ for most degenerate simplices down to $1/\sqrt{3} \doteq 0.577$ for an equilateral triangle or $\sqrt{6}/4 \doteq 0.612$ for an equilateral tetrahedron. But is it a good estimate of element quality?

In two dimensions, the answer is yes. A triangle's radius-edge ratio is related to its smallest angle $\theta_{\min}$ by the formula

$$\frac{R}{\ell_{\min}} = \frac{1}{2\sin\theta_{\min}}.$$

Figure 1.20 illustrates how this identity is derived for a triangle xyz with circumcenter c. Observe that the triangles ycz and xcz are isosceles, so their apex angles are $\angle ycz = 180° - 2\phi$ and $\angle xcz = 180° - 2\phi - 2\theta$. Therefore, $\varphi = 2\theta_{\min}$ and $\ell_{\min} = 2R\sin\theta_{\min}$. This reasoning holds even if ϕ is negative.

The smaller a triangle's radius-edge ratio, the larger its smallest angle. The angles of a triangle sum to 180°, so the triangle's largest angle is at most $180° - 2\theta_{\min}$; hence an upper bound on the radius-edge ratio places bounds on both the smallest and largest angles.

In three dimensions, however, the radius-edge ratio is a flawed measure. It screens out all the tetrahedra in Figure 1.19 except slivers. A degenerate sliver can have a radius-edge ratio as small as $1/\sqrt{2} \doteq 0.707$, which is not far from the 0.612 of an equilateral tetrahedron. Delaunay refinement algorithms are guaranteed to remove all tetrahedra with large radius-edge ratios, but they do not promise to remove all slivers.

There are other quality measures that screen out all the skinny tetrahedra in Figure 1.19, including slivers and spires, but Delaunay refinement does not promise to bound these measures. A popular measure is the *radius ratio* r/R, suggested by Cavendish, Field, and Frey [25], where r is τ's inradius and R is its circumradius. It obtains a maximum value of $1/2$ for an equilateral triangle or $1/3$ for an equilateral tetrahedron, and a minimum value of zero for a degenerate element, which implies that it approaches zero

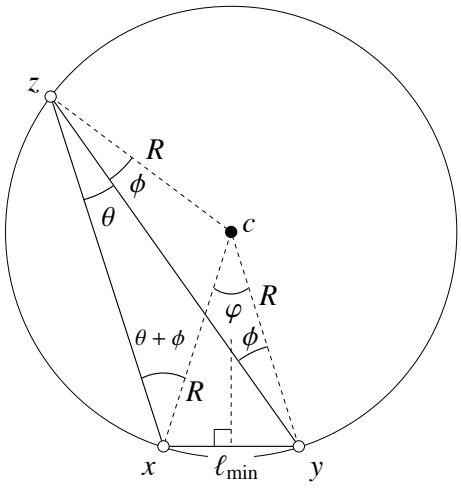


Figure 1.20: Relationships between the circumradius R, shortest edge $\ell_{\min}$, and smallest angle θ.

as any dihedral angle separating τ's faces approaches $0°$ or $180°$, any plane angle separating τ's edges approaches $0°$ or $180°$, or any solid angle at τ's vertices approaches $0°$ or $360°$.

For a triangle τ, the radius ratio is related to the smallest angle $\theta_{\min}$ by the inequalities

$$2\sin^2\frac{\theta_{\min}}{2} \;\le\; \frac{r}{R} \;\le\; 2\tan\frac{\theta_{\min}}{2},$$

which implies that it approaches zero as $\theta_{\min}$ approaches zero, and vice versa.

Two unfortunate properties of the circumradius are that it is relatively expensive to compute for a tetrahedron, and it can be numerically unstable. A tiny perturbation of the position of one vertex of a skinny tetrahedron can induce an arbitrarily large change in its circumradius. Both the radius-edge ratio and the radius ratio inherit these problems. In these respects, a better quality measure for tetrahedra is the *volume-length measure* V/ℓ_{rms}^3, suggested by Parthasarathy, Graichen, and Hathaway [90], where V is the volume of a tetrahedron and ℓ_{rms} is the root-mean-squared length of its six edges. It obtains a maximum value of $1/(6\sqrt{2})$ for an equilateral tetrahedron and a minimum value of zero for a degenerate tetrahedron. The volume-length measure is numerically stable and faster to compute than a tetrahedron's circumradius. It has proven itself as a filter against all poorly shaped tetrahedra and as an objective function for mesh improvement algorithms, especially optimization-based smoothing [70].

1.6 Maps and Homeomorphisms

Two metric spaces are considered to be the same if the points that comprise them are connected the same way. For example, the boundary of a cube can be deformed into a sphere without cutting or gluing it. They have the same topology. We formalize this notion of topological equality by defining a function that maps the points of one space to points of the other and preserves how they are connected. Specifically, the function preserves limit points.

A function from one space to another that preserves limit points is called a *continuous function* or a *map*.[3] Continuity is just a step on the way to topological equivalence, because a continuous function can

[3]There is a small caveat with this characterization: a function g that maps a neighborhood of x to a single point $g(x)$ may be

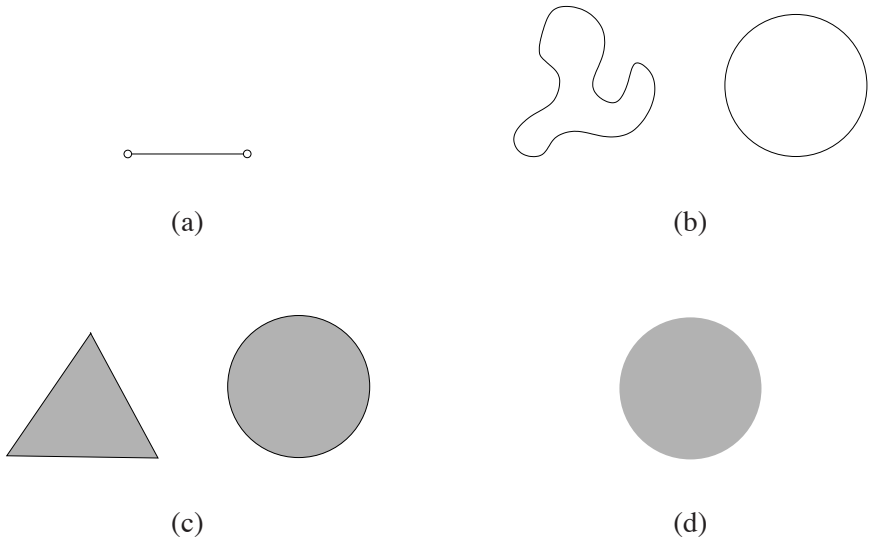


Figure 1.21: (a) A 1-ball. (b) Spaces homeomorphic to the 1-sphere. (c) Spaces homeomorphic to the 2-ball. (d) An open 2-ball. It is homeomorphic to $\mathbb{R}^2$, but not to a closed 2-ball.

map many points to a single point in the target space, or map no points to a given point in the target space. True equivalence is marked by a *homeomorphism*, a one-to-one function from one space to another that possesses both continuity and a continuous inverse, so that limit points are preserved in both directions.

Definition 21 (continuous function; map)**.** *Let $\mathbb{T}$ and $\mathbb{U}$ be metric spaces. A function $g : \mathbb{T} \to \mathbb{U}$ is continuous if for every set $Q \subseteq \mathbb{T}$ and every limit point $p \in \mathbb{T}$ of Q, $g(p)$ is either a limit point of the set $g(Q)$ or in $g(Q)$. Continuous functions are also called maps.*

Definition 22 (homeomorphism)**.** *Let $\mathbb{T}$ and $\mathbb{U}$ be metric spaces. A homeomorphism is a bijective (one-to-one) map $h : \mathbb{T} \to \mathbb{U}$ whose inverse is continuous too. Two metric spaces are homeomorphic if there exists a homeomorphism between them.*

Homeomorphism induces an equivalence relation among metric spaces, which is why two homeomorphic metric spaces are called *topologically equivalent*. Figure 1.21(b, c) shows pairs of metric spaces that are homeomorphic. A less obvious example is that the open unit d-ball $\mathbb{B}_o^d = \{x \in \mathbb{R}^d : |x| < 1\}$ is homeomorphic to the Euclidean space $\mathbb{R}^d$, a fact demonstrated by the map $h(p) = (1/(1 - |p|))p$. The same map shows that the open unit halfball $\mathbb{H}^d = \{x \in \mathbb{R}^d : |x| < 1 \text{ and } x_d \geq 0\}$ is homeomorphic to the Euclidean halfspace $\{x \in \mathbb{R}^d : x_d \geq 0\}$.

Homeomorphism gives us a purely topological definition of what it means to triangulate a domain.

Definition 23 (triangulation of a metric space)**.** *A simplicial complex $\mathcal{K}$ is a triangulation of a metric space $\mathbb{T}$ if its underlying space $|\mathcal{K}|$ is homeomorphic to $\mathbb{T}$.*

1.7 Manifolds

A manifold is a set of points that is locally connected in a particular way. A 1-manifold has the structure of a piece of string, possibly with its ends tied in a loop, and a 2-manifold has the structure of a piece of paper

continuous, but technically $g(x)$ is not a limit point of itself, so in this sense a continuous function might not preserve all limit points. This technicality does not apply to homeomorphisms because they are bijective.

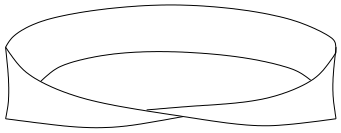


Figure 1.22: Mobiüs band.

or rubber sheet that has been cut and perhaps glued along its edges—a category that includes disks, spheres, tori, and Mobiüs bands.

Definition 24 (manifold)**.** *A metric space* Σ *is a* k*-manifold, or simply manifold, if every point* $x \in \Sigma$ *has a neighborhood homeomorphic to* $\mathbb{R}^k$ *or* $\mathbb{H}^k$*. The dimension of* Σ *is* k*.*

A manifold can be viewed as a purely abstract metric space, or it can be embedded into a metric space or a Euclidean space. Even without an embedding, every manifold can be partitioned into boundary and interior points. Observe that these words mean very different things for a manifold than they do for a metric space.

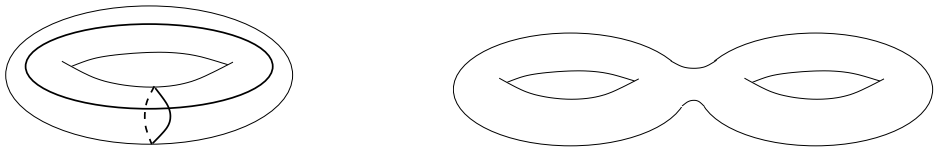
Definition 25 (boundary, interior)**.** *The interior* $\operatorname{Int}\Sigma$ *of a manifold* Σ *is the set of points in* Σ *that have a neighborhood homeomorphic to* $\mathbb{R}^k$*. The boundary* $\operatorname{Bd}\Sigma$ *of* Σ *is the set of points* $\Sigma \setminus \operatorname{Int}\Sigma$*. Except for the case of* 0*-manifolds (points) whose boundary is empty,* $\operatorname{Bd}\Sigma$ *consists of points that have a neighborhood homeomorphic to* $\mathbb{H}^k$*. If* $\operatorname{Bd}\Sigma$ *is the empty set, we say that* Σ *is without boundary.*

For example, the closed disk $\mathbb{B}^2$ is a 2-manifold whose interior is the open disk $\mathbb{B}^2_o$ and whose boundary is the circle $\mathbb{S}^1$. The open disk $\mathbb{B}^2_o$ is a 2-manifold whose boundary is the empty set. So is the Euclidean space $\mathbb{R}^2$, and so is the sphere $\mathbb{S}^2$. The open disk is homeomorphic to $\mathbb{R}^2$, but the sphere is topologically different from the other two. Moreover, the sphere is compact (bounded and closed with respect to $\mathbb{R}^3$) whereas the other two are not.

A 2-manifold Σ is *non-orientable* if starting from a point p one can walk on Σ and end up on the opposite side of Σ when returning to p. Otherwise, Σ is orientable. Spheres and balls are orientable, whereas the *Mobiüs band* in Figure 1.22 is a non-orientable 2-manifold.

A *surface* is a 2-manifold that is a subspace of $\mathbb{R}^d$. Any compact surface without boundary in $\mathbb{R}^3$ is an orientable 2-manifold. To be non-orientable, a compact surface must have a nonempty boundary or be embedded in a 4- or higher-dimensional Euclidean space.

A surface can sometimes be disconnected by removing one or more *loops* (1-manifolds without boundary) from it. The *genus* of a surface is g if $2g$ is the maximum number of loops that can be removed from the surface without disconnecting it; here the loops are permitted to intersect each other. For example, the sphere has genus zero as any loop cuts it into two surfaces. The torus has genus one: a circular cut around its neck and a second circular cut around its circumference, illustrated in Figure 1.23(a), allow it to unfold into a rectangle, which topologically is a disk. A third loop would cut it into two pieces. Figure 1.23(b) shows a 2-manifold without boundary of genus 2. Although a high-genus surface can have a very complex shape, all compact 2-manifolds of genus g without boundary are homeomorphic to each other.



(a) (b)

Figure 1.23: (a) Removal of the bold loops opens up the torus into a topological disk. (b) Every surface without boundary in $\mathbb{R}^3$ resembles a sphere or a conjunction of one or more tori.

Chapter 2

Two-Dimensional Delaunay Triangulations

The Delaunay triangulation is a geometric structure that engineers have used for meshes since mesh generation was in its infancy. In two dimensions, it has a striking advantage: among all possible triangulations of a fixed set of points, the Delaunay triangulation maximizes the minimum angle. It also optimizes several other geometric criteria related to interpolation accuracy. If it is your goal to create a triangulation without small angles, it seems almost silly to consider a triangulation that is not Delaunay. Delaunay triangulations have been studied thoroughly, and excellent algorithms are available for constructing and updating them.

A constrained triangulation is a triangulation that enforces the presence of specified edges—for example, the boundary of a nonconvex object. A constrained Delaunay triangulation relaxes the Delaunay property just enough to recover those edges, while enjoying optimality properties similar to those of a Delaunay triangulation. Constrained Delaunay triangulations are nearly as popular as their unconstrained ancestors.

This chapter surveys two-dimensional Delaunay triangulations, constrained Delaunay triangulations, and their geometric properties. See Fortune [52] for an alternative survey of Delaunay triangulations, and Aurenhammer [4] for a survey of many more types of Voronoi diagrams.

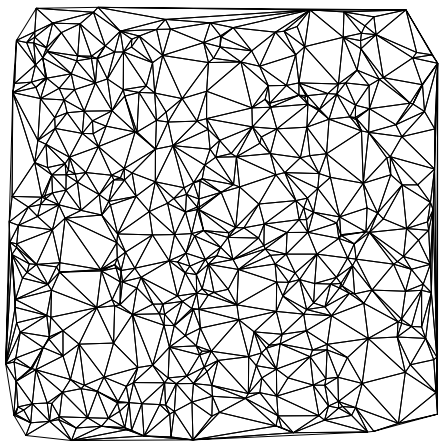


Figure 2.1: A Delaunay triangulation.

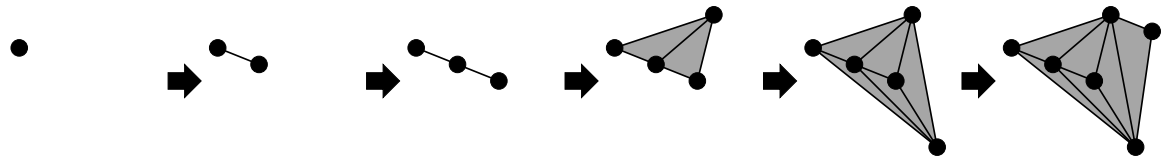


Figure 2.2: Incremental construction of a lexicographic triangulation.

2.1 Triangulations of a Planar Point Set

The word *triangulation* usually refers to a simplicial complex, but it has multiple meanings when we discuss a *triangulation of* some geometric entity that is being triangulated. There are triangulations of point sets, polygons, polyhedra, and many other structures. Consider points in the plane (or in any Euclidean space).

Definition 26 (triangulation of a point set)**.** *Let V (for "vertices") be a finite set of points in the plane. A* triangulation of V *is a simplicial complex $\mathcal{T}$ such that*

- *V is the set of vertices in $\mathcal{T}$, and*
- *the union of all the simplices in $\mathcal{T}$ is the convex hull of V.*

Does every point set have a triangulation? Yes. Consider the *lexicographic triangulation* illustrated in Figure 2.2. To construct one, sort the points *lexicograpically* (that is, by x-coordinate, ordering points with the same x-coordinate according to their y-coordinates), yielding a sorted sequence $v_1, v_2, \ldots, v_n$ of points. Define the lexicographic triangulation $\mathcal{T}_i$ of the first i points by induction as follows. The first triangulation is $\mathcal{T}_1 = \{v_1\}$. Each subsequent triangulation is $\mathcal{T}_i = \mathcal{T}_{i-1} \cup \{v_i\} \cup \{\text{conv}(\{v_i\} \cup \sigma) : \sigma \in \mathcal{T}_{i-1}$ and the relative interior of $\text{conv}(\{v_i\} \cup \sigma)$ intersects no simplex in $\mathcal{T}_{i-1}\}$.

V has a triangulation even if all its points are collinear: $\mathcal{T}_n$ contains n vertices, $n-1$ collinear edges connecting them, and no triangles.

A triangulation of n points in the plane has at most $2n-5$ triangles and $3n-6$ edges as a consequence of Euler's formula. With no change, Definition 26 defines triangulations of point sets in higher-dimensional Euclidean spaces as well.

2.2 The Delaunay Triangulation

The *Delaunay triangulation* of a point set V, introduced by Boris Nikolaevich Delone [43] in 1934, is a triangulation of V whose triangles are particularly nicely shaped. Figure 2.1 illustrates a Delaunay triangulation. Its defining characteristic is the *empty circumcircle property*: no triangle has a circumscribing circle that encloses any point in V.

Definition 27 (circumcircle)**.** *The* circumcircle*, or* circumscribing circle*, of a triangle is the unique circle that passes through all three of its vertices. A* circumcircle *of an edge is any circle that passes through both its vertices.*

In the plane, an edge has an infinite set of circumcircles; a triangle has only one.

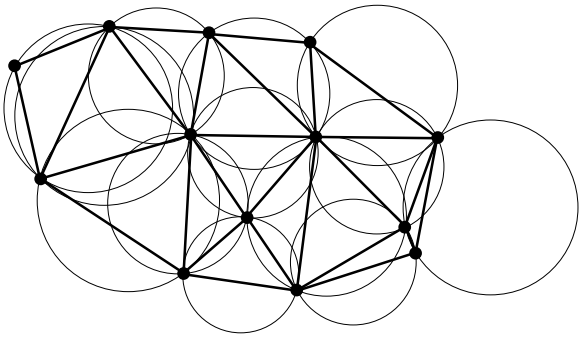


Figure 2.3: Every triangle in a Delaunay triangulation has an empty circumcircle.

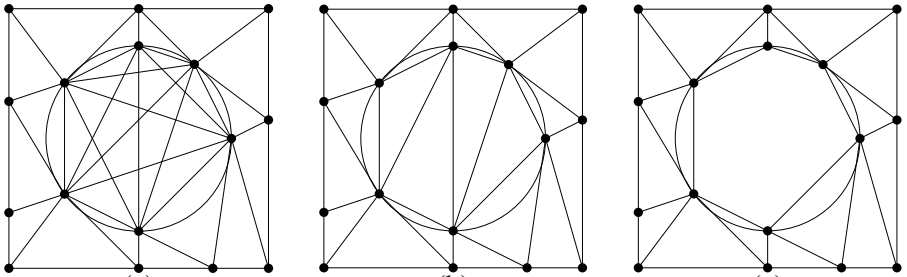


Figure 2.4: Three ways to define a Delaunay structure in the presence of cocircular vertices. (a) Include all the Delaunay simplices. (b) Choose a subset of Delaunay simplices that comprises a triangulation. (c) Exclude all crossing Delaunay edges, and fuse overlapping Delaunay triangles into Delaunay polygons.

Definition 28 (Delaunay)**.** *In the context of a finite point set V, a triangle is* Delaunay *if its vertices are in V and its circumcircle is* empty*—encloses no point in V. Note that any number of points can lie* on *a Delaunay triangle's circumcircle.*

An edge is Delaunay *if its vertices are in V and it has at least one empty circumcircle.*

A Delaunay triangulation *of V, denoted* Del V*, is a triangulation of V in which every triangle is Delaunay, as illustrated in Figure 2.3.*

You might wonder whether every point set has a Delaunay triangulation, and how many Delaunay triangulations a point set can have. The answer to the first question is "yes." Section 2.3 gives some intuition for why this is true, and Section 2.4 gives a proof.

The Delaunay triangulation of V is unique if and only if no four points in V lie on a common empty circle, a fact proven in Section 2.7. Otherwise, there are Delaunay triangles and edges whose interiors intersect, as illustrated in Figure 2.4(a). Most applications omit some of these triangles and edges so that the survivors form a simplicial complex, as in Figure 2.4(b). Depending on which Delaunay simplices you keep and which you discard, you obtain different Delaunay triangulations.

It is sometimes useful to unite the intersecting triangles into a single polygon, depicted in Figure 2.4(c). The *Delaunay subdivision* obtained this way is a polyhedral complex, rather than a simplicial complex. It has the advantage of being the geometric dual of the famous *Voronoi diagram*.

Clearly, a simplex's being Delaunay does not guarantee that it is in *every* Delaunay triangulation of a point set. But a slightly stronger property does provide that guarantee.

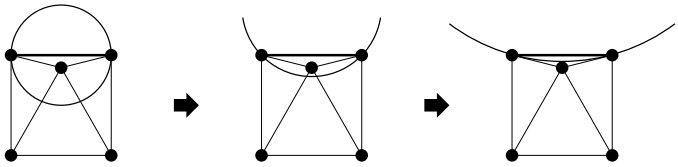


Figure 2.5: Every edge on the boundary of a convex triangulation is strongly Delaunay, because it is always possible to find an empty circle that passes through its endpoints and no other vertex.

Definition 29 (strongly Delaunay)**.** *In the context of a finite point set V, a triangle τ is* strongly Delaunay *if its vertices are in V and no point in V lies inside* or on *its circumcircle, except the vertices of τ.*

An edge e is strongly Delaunay *if its vertices are in V and it has at least one circumcircle that no point in V lies inside or on, except the vertices of e.*

Every point in V is a strongly Delaunay vertex.

Every Delaunay triangulation of V contains every strongly Delaunay simplex, a fact proven in Section 2.7. The Delaunay subdivision contains the strongly Delaunay edges and triangles, and no others.

Consider two examples of strongly Delaunay edges. First, every edge on the boundary of a triangulation of V is strongly Delaunay. Figure 2.5 shows why. Second, the edge connecting a point $v \in V$ to its nearest neighbor $w \in V$ is strongly Delaunay, because the smallest circle passing through v and w does not enclose nor touch any other point in V. Therefore, every Delaunay triangulation connects every vertex to its nearest neighbor.

2.3 The Parabolic Lifting Map

Given a finite point set V, the *parabolic lifting map* of Seidel [107, 48] transforms the Delaunay subdivision of V into faces of a convex polyhedron in three dimensions, as illustrated in Figure 2.6. This relationship between Delaunay triangulations and convex hulls has two consequences. First, it makes many properties of the Delaunay triangulation intuitive. For example, from the fact that every finite point set has a polyhedral convex hull, it follows that every finite point set has a Delaunay triangulation. Second, it brings to mesh generation the power of a huge literature on polytope theory and algorithms: every convex hull algorithm is a Delaunay triangulation algorithm!

The parabolic lifting map sends each point $p = (x, y) \in E^2$ to a point $p^+ = (x, y, x^2 + y^2) \in E^3$. Call p^+ the *lifted companion* of p.

Consider the convex hull $\text{conv}(V^+)$ of the lifted points $V^+ = \{v^+ : v \in V\}$. Figure 2.6 illustrates its downward-facing faces. Formally, a face f of $\text{conv}(V^+)$ is *downward-facing* if no point in $\text{conv}(V^+)$ is directly below any point in f, with respect to the z-axis. Call the collection of downward-facing faces the *underside* of $\text{conv}(V^+)$. Projecting the underside of $\text{conv}(V^+)$ to the x-y plane (by discarding every point's z-coordinate) yields the Delaunay subdivision of V. If V has more than one Delaunay triangulation, this Delaunay subdivision has non-triangular polygons, like the hexagon in Figure 2.4(c). Triangulating these polygonal faces yields a Delaunay triangulation.

For a simplex σ in the plane, its *lifted companion* σ^+ is the simplex embedded in E^3 whose vertices are the lifted companions of the vertices of σ. Note that σ^+ is flat, and does not curve to hug the paraboloid. The following lemma and theorem show that every Delaunay simplex's lifted companion is included in a downward-facing face of $\text{conv}(V^+)$.

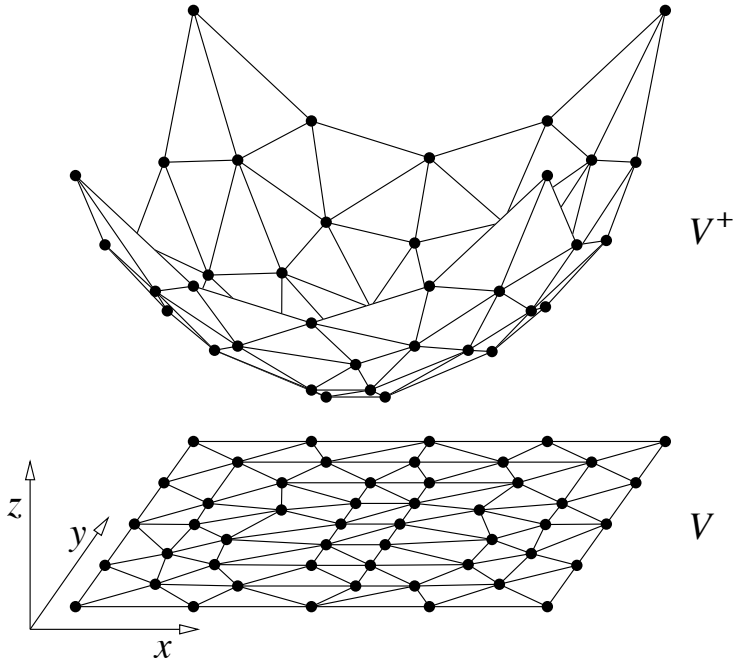


Figure 2.6: The parabolic lifting map.

Lemma 1. *Let C be a circle in the plane. Let $C^+ = \{p^+ : p \in C\}$ be the ellipse obtained by lifting C to the paraboloid. Then the points of C^+ lie on a plane h, which is not parallel to the z-axis. Furthermore, every point p inside C lifts to a point p^+ below h, and every point p outside C lifts to a point p^+ above h. Therefore, testing whether a point p is inside, on, or outside C is equivalent to testing whether the lifted point p^+ is below, on, or above h.*

Proof. Let O and r be the center and radius of C, respectively. Let p be a point in the plane. The z-coordinate of p^+ is $|p|^2$. By expanding $|O - p|^2$, we have the identity $|p|^2 = 2O \cdot p - |O|^2 + |O - p|^2$.

With O and r fixed and $p \in E^2$ varying, the equation $z = 2O \cdot p - |O|^2 + r^2$ defines a plane h in E^3, not parallel to the z-axis. For every point $p \in C$, $|O - p| = r$, so $C^+ \subset h$. For every point $p \notin C$, if $|O - p| < r$, then the lifted point p^+ lies below h, and if $|O - p| > r$, then p^+ lies above h. ■

Theorem 2 (Seidel [107])**.** *Let σ be a simplex whose vertices are in V, and let σ^+ be its lifted companion. Then σ is Delaunay if and only if σ^+ is included in some downward-facing face of* conv(V^+). *The simplex σ is strongly Delaunay if and only if σ^+ is a downward-facing face of* conv(V^+).

Proof. If σ is Delaunay, σ has a circumcircle C that encloses no point in V. Let h be the unique plane in E^3 that includes C^+. By Lemma 1, no point in V^+ lies below h. Because the vertices of σ^+ are in C^+, $h \supset \sigma^+$. Therefore, σ^+ is included in a downward-facing face of the convex hull of V^+. If σ is strongly Delaunay, every point in V^+ lies above h except the vertices of σ^+. Therefore, σ^+ is a downward-facing face of the convex hull of V^+.

The converse implications follow by reversing the argument. ■

The parabolic lifting map works equally well for Delaunay triangulations in three or more dimensions; Lemma 1 and Theorem 2 generalize to higher dimensions without any new ideas. Theorem 2 implies that

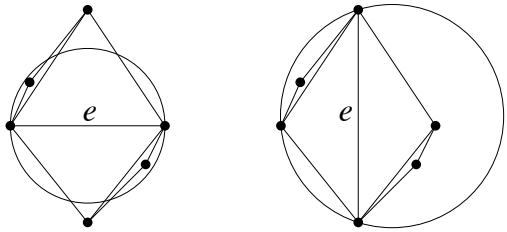


Figure 2.7: At left, e is locally Delaunay. At right, e is not.

any algorithm for constructing convex hulls of point sets in E^{d+1} can construct the Delaunay triangulation of a point set in E^d.

The relationship between Delaunay triangulations and convex hulls was first discovered by Brown [21], who proposed a different lifting map that projects V onto a sphere. The parabolic lifting map is numerically better behaved than the spherical one.

2.4 The Delaunay Lemma

Perhaps the most important result about Delaunay triangulations is the *Delaunay Lemma*, proven by Boris Delaunay himself [43]. It provides an alternative characterization of the Delaunay triangulation: a triangulation whose edges are *locally Delaunay*.

Definition 30 (locally Delaunay)**.** *Let e be an edge in a triangulation $\mathcal{T}$ of a planar point set. If e is an edge of fewer than two triangles in $\mathcal{T}$, then e is said to be* locally Delaunay. *If e is an edge of exactly two triangles τ_1 and τ_2 in $\mathcal{T}$, then e is* locally Delaunay *if it has a circumcircle enclosing no vertex of τ_1 nor τ_2. Equivalently, the circumcircle of τ_1 encloses no vertex of τ_2. Equivalently, the circumcircle of τ_2 encloses no vertex of τ_1.*

Figure 2.7 shows two different triangulations of six vertices. In the triangulation at left, the edge e is locally Delaunay, because the depicted circumcircle of e does not enclose either vertex opposite e. Nevertheless, e is not Delaunay, thanks to other vertices inside e's circumcircle. In the triangulation at right, e is not locally Delaunay; every circumcircle of e encloses at least one of the two vertices opposite e.

The Delaunay Lemma has several uses. First, it provides a linear-time algorithm to determine whether a triangulation is Delaunay: simply test whether every edge is locally Delaunay. Second, it implies a simple algorithm for producing a Delaunay triangulation called the *flip algorithm* (Section 2.5). The flip algorithm helps to prove that Delaunay triangulations have useful optimality properties. Third, the Delaunay Lemma helps to prove the correctness of other algorithms for constructing Delaunay triangulations.

As with many properties of Delaunay triangulations, the lifting map provides intuition for the Delaunay Lemma. On the lifting map, the Delaunay Lemma is essentially the observation that a simple polyhedron is convex if and only if its has no *reflex edge*. A reflex edge is an edge where the polyhedron is locally nonconvex; that is, the dihedral angle at which the two adjoining faces meet along that edge exceeds 180°, measured through the interior of the polyhedron. If a triangulation has an edge that is not locally Delaunay, that edge's lifted companion is a reflex edge of the lifted triangulation (by Lemma 1).

Theorem 3 (Delaunay Lemma [43])**.** *Let $\mathcal{T}$ be a triangulation of a point set V. The following three statements are equivalent.*

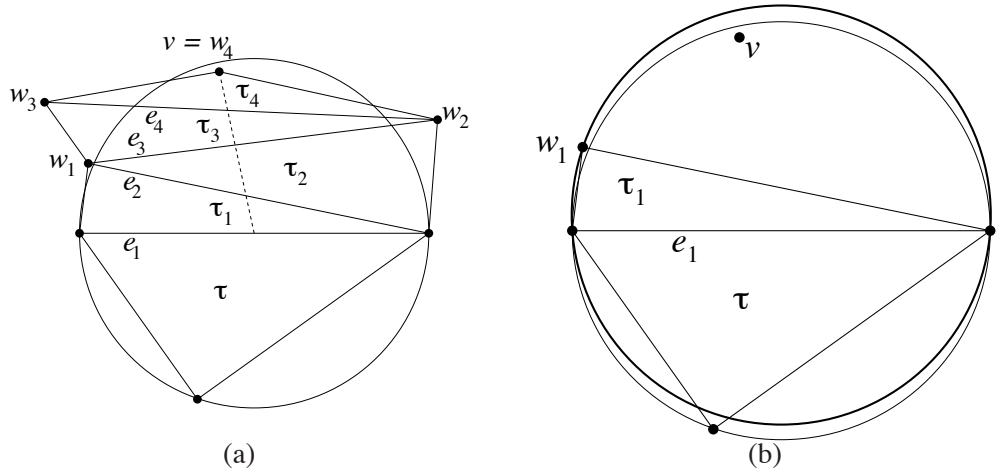


Figure 2.8: (a) Because τ's circumcircle encloses v, some edge between v and τ is not locally Delaunay. (b) Because v lies above e_1 and inside τ's circumcircle, and because w_1 lies outside (or on) τ's circumcircle, v must lie inside τ_1's circumcircle.

A. Every triangle in $\mathcal{T}$ is Delaunay (i.e. $\mathcal{T}$ is Delaunay).

B. Every edge in $\mathcal{T}$ is Delaunay.

C. Every edge in $\mathcal{T}$ is locally Delaunay.

Proof. If the points in V are all collinear, V has only one triangulation, which trivially satisfies all three properties.

Otherwise, let e be an edge in $\mathcal{T}$; e is an edge of at least one triangle $\tau \in \mathcal{T}$. If τ is Delaunay, τ's circumcircle is empty, and because τ's circumcircle is also a circumcircle of e, e is Delaunay. Therefore, Property A implies Property B. If an edge is Delaunay, it is clearly locally Delaunay too, so Property B implies Property C. The proof is complete if Property C implies Property A. Of course, this is the hard part.

Suppose that every edge in $\mathcal{T}$ is locally Delaunay. Suppose for the sake of contradiction that Property A does not hold. Then some triangle $\tau \in \mathcal{T}$ is not Delaunay, and some vertex $v \in V$ is inside τ's circumcircle. Let e_1 be the edge of τ that separates v from the interior of τ, as illustrated in Figure 2.8(a). Without loss of generality, assume that e_1 is oriented horizontally, with τ below e_1.

Draw a line segment from the midpoint of e_1 to v—see the dashed line in Figure 2.8(a). If the line segment intersects some vertex other than v, replace v with the lowest such vertex. Let $e_1, e_2, e_3, \ldots, e_m$ be the sequence of triangulation edges (from bottom to top) whose relative interiors the line segment intersects. Because $\mathcal{T}$ is a triangulation of V, every point on the line segment lies either in a single triangle or on an edge. Let w_i be the vertex above e_i that forms a triangle τ_i in conjunction with e_i. Observe that $w_m = v$.

By assumption, e_1 is locally Delaunay, so w_1 lies on or outside the circumcircle of τ. As Figure 2.8(b) shows, it follows that the circumcircle of τ_1 encloses every point above e_1 inside the circumcircle of τ, and hence encloses v. Repeating this argument inductively, we find that the circumcircles of $\tau_2, \ldots, \tau_m$ enclose v. But $w_m = v$ is a vertex of τ_m, which contradicts the claim that v is inside the circumcircle of τ_m. ■

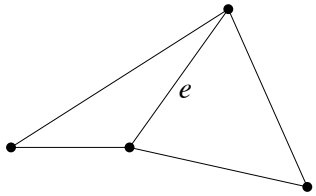


Figure 2.9: In this nonconvex quadrilateral, e cannot be flipped.

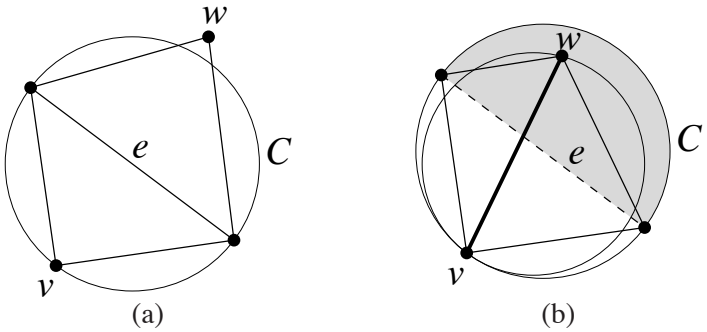


Figure 2.10: (a) The edge e is locally Delaunay. (b) The edge e is not locally Delaunay. The edge created by a flip of e is locally Delaunay.

2.5 The Flip Algorithm

The *flip algorithm* has at least three uses: it is a simple algorithm for computing a Delaunay triangulation, it is the core of a constructive proof that every finite set of points in the plane has a Delaunay triangulation, and it is the core of a proof that the Delaunay triangulation optimizes several geometric criteria when compared with all other triangulations of the same point set.

Let V be a point set you wish to triangulate. The flip algorithm begins with any triangulation $\mathcal{T}$ of V; for instance, the lexicographic triangulation described in Section 2.1. The Delaunay Lemma tells us that $\mathcal{T}$ is Delaunay if and only if every edge in $\mathcal{T}$ is locally Delaunay. The flip algorithm repeatedly chooses any edge that is not locally Delaunay, and *flips* it.

The union of two triangles that share an edge is a quadrilateral, and the shared edge is a diagonal of the quadrilateral. To flip an edge is to replace it with the quadrilateral's other diagonal, as illustrated in Figure 2.7. An edge flip is legal only if the two diagonals cross each other—equivalently, if the quadrilateral is convex. Figure 2.9 shows that not every edge can be flipped, because the quadrilateral might not be convex. Fortunately, unflippable edges are always locally Delaunay.

Lemma 4. *Let e be an edge in 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. Consider the quadrilateral defined by e, v, and w, illustrated in Figure 2.10. Let C be the circle that passes through v and the vertices of e.

If w is on or outside C, as in Figure 2.10(a), then the empty circle C demonstrates that e is locally Delaunay.

Otherwise, w is inside the section of C bounded by e and opposite v. This section is shaded in Figure 2.10(b). The quadrilateral is thus strictly convex, so e is flippable. Furthermore, the circle that is tangent

to C at v and passes through w does not enclose the endpoints of e, because C encloses it, as Figure 2.10(b) demonstrates. Therefore, the edge vw is locally Delaunay. ■

Lemma 4 shows that the flip algorithm can flip any edge that is not locally Delaunay, thereby creating an edge that is. Unfortunately, the outer four edges of the quadrilateral might discover that they are no longer locally Delaunay, even if they were locally Delaunay before the flip. If the flip algorithm repeatedly flips edges that are not locally Delaunay, will it go on forever?

Theorem 5. *Given a triangulation of n points, the flip algorithm terminates after $O(n^2)$ edge flips, yielding a Delaunay triangulation.*

Proof: Let $\mathcal{T}$ be the initial triangulation provided as input to the flip algorithm. Let $\mathcal{T}^+ = \{\sigma^+ : \sigma \in \mathcal{T}\}$ be the initial triangulation lifted to the parabolic lifting map; $\mathcal{T}^+$ is a simplicial complex embedded in E^3. If $\mathcal{T}$ is Delaunay, then $\mathcal{T}^+$ triangulates the underside of $\text{conv}(V^+)$; otherwise, by Lemma 1, the edges of $\mathcal{T}$ that are not locally Delaunay lift to reflex edges of $\mathcal{T}^+$.

By Lemma 4, an edge flip replaces an edge that is not locally Delaunay with one that is. In the lifted triangulation $\mathcal{T}^+$, a flip replaces a reflex edge with a convex edge. Let Q be the set containing the four vertices of the two triangles that share the flipped edge. Then $\text{conv}(Q^+)$ is a tetrahedron whose upper faces are the pre-flip simplices and whose lower faces are the post-flip simplices. Imagine the edge flip as the act of gluing the tetrahedron $\text{conv}(Q^+)$ to the underside of $\mathcal{T}^+$.

Each edge flip monotonically lowers the lifted triangulation, so once flipped, an edge can never reappear. The flip algorithm can perform no more than $n(n-1)/2$ flips—the number of edges that can be defined on n vertices—so it must terminate. But the flip algorithm terminates only when every edge is locally Delaunay. By the Delaunay Lemma, the final triangulation is Delaunay. ■

The fact that the flip algorithm terminates helps to prove that point sets have Delaunay triangulations.

Corollary 6. *Every finite set of points in the plane has a Delaunay triangulation.*

Proof: Section 2.1 demonstrates that every finite point set has at least one triangulation. By Theorem 5, the application of the flip algorithm to that triangulation produces a Delaunay triangulation. ■

If a point set has more than one Delaunay triangulation, the flip algorithm will find one of them. Which one it finds depends upon the starting triangulation and the order in which flips are performed.

An efficient implementation of the flip algorithm requires one extra ingredient. How quickly can you find an edge that is not locally Delaunay? To repeatedly test every edge in the triangulation would be slow. Instead, the flip algorithm maintains a list of edges that might not be locally Delaunay. The list initially contains every edge in the triangulation. Thereafter, the flip algorithm iterates the following procedure until the list is empty, whereupon the algorithm halts.

- Remove an edge from the list.
- Check whether the edge is still in the triangulation, and if so, whether it is locally Delaunay.
- If the edge is present but not locally Delaunay, flip it, and add the four edges of the flipped quadrilateral to the list.

The list may contain multiple copies of the same edge, but they do no harm.

Implemented this way, the flip algorithm runs in $O(n + k)$ time, where n is the number of vertices (or triangles) of the triangulation, and k is the number of flips performed. In the worst case, $k \in \Theta(n^2)$, giving $O(n^2)$ running time. But there are circumstances where the flip algorithm is fast in practice. For instance, if the vertices of a Delaunay mesh are perturbed by small displacements during a physical simulation, it might take only a small number of flips to restore the Delaunay property. In this circumstance, the flip algorithm probably outperforms any algorithm that reconstructs the triangulation from scratch.

2.6 The Optimality of the Delaunay Triangulation

Delaunay triangulations are valuable in part because they optimize several geometric criteria: the smallest angle, the largest circumcircle, and the largest min-containment circle. The *min-containment circle* of a triangle is the smallest circle that encloses it. For a triangle with no obtuse angles, the circumcircle and the min-containment circle are the same, but for an obtuse triangle, the min-containment circle is smaller.

Theorem 7. *Among all the triangulations of a point set, there is a Delaunay triangulation that maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle.*

Proof: Each of these properties is locally improved when an edge that is not locally Delaunay is flipped; Lemma 8 below demonstrates this for the first two properties. (I omit the min-containment property in favor of a general-dimensional proof in Section 4.4.) There is at least one optimal triangulation $\mathcal{T}$. If $\mathcal{T}$ has an edge that is not locally Delaunay, flipping that edge produces another optimal triangulation. When the flip algorithm runs with $\mathcal{T}$ as its input, every triangulation it iterates through is optimal by induction, and by Theorem 5, that includes a Delaunay triangulation. ■

Lemma 8. *Flipping an edge that is not locally Delaunay increases the minimum angle and reduces the largest circumcircle among the triangles changed by the flip.*

Proof: Let uv be the flipped edge, and let $\triangle wvu$ and $\triangle xuv$ be the triangles deleted by the flip, so $\triangle wxu$ and $\triangle xwv$ are the triangles created by the flip.

The angle opposite the edge uw is $\angle wvu$ before the flip, and $\angle wxu$ after the flip. As Figure 2.11 illustrates, because the circumcircle of $\triangle wvu$ encloses x, the latter angle is greater than the former angle by Thales' Theorem, a standard and ancient fact about circle geometry. Likewise, the flip increases the angles opposite wv, vx, and xu.

Each of the other two angles of the new triangles, $\angle xuw$ and $\angle wvx$, is a sum of two pre-flip angles that merge when uv is deleted. It follows that all six angles of the two post-flip triangles exceed the smallest of the four angles that uv participates in before the flip.

Suppose without loss of generality that the circumcircle of $\triangle wxu$ is at least as large as the circumcircle of $\triangle xwv$, and that $\angle wxu \leq \angle uwx$, implying that $\angle wxu$ is acute. Because the circumcircle of $\triangle wvu$ encloses x, it is larger than the circumcircle of $\triangle wxu$, as illustrated in Figure 2.11. It follows that the largest pre-flip circumcircle is larger than the largest post-flip circumcircle. ■

Theorem 7 guarantees that if a point set has only one Delaunay triangulation, the Delaunay triangulation is optimal. But what if a point set has more than one Delaunay triangulation? *Every* Delaunay triangulation

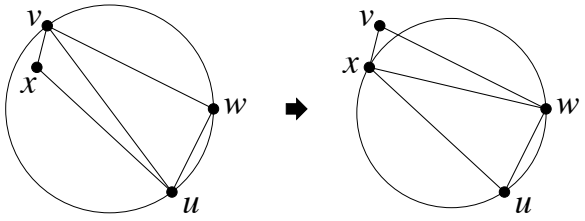


Figure 2.11: A Delaunay flip increases the angle opposite edge uw and, if $\angle wxu$ is acute, reduces the circumcircle of the triangle adjoining that edge.

optimizes these criteria, because any Delaunay triangulation can be transformed to any other Delaunay triangulation of the same points by a sequence of edge flips such that every intermediate triangulation is Delaunay, and each flip preserves optimality.

Unfortunately, the optimality properties of Theorem 7 do not generalize to Delaunay triangulations in dimensions higher than two, with the exception of minimizing the largest min-containment circle. However, the list of optimality properties in Theorem 7 is not complete. Section 4.4 discusses criteria related to interpolation error for which Delaunay triangulations of any dimension are optimal.

The flip algorithm and the Delaunay triangulation's property of maximizing the minimum angle were both introduced by a classic paper by Charles Lawson [73], which also introduced the incremental insertion algorithm for constructing a Delaunay triangulation. D'Azevedo and Simpson [42] were the first to show that two-dimensional Delaunay triangulations minimize the largest min-containment circle. Rajan [96] shows that higher-dimensional Delaunay triangulations minimize the largest min-containment hypersphere.

2.7 The Uniqueness of the Delaunay Triangulation

The strength of a strongly Delaunay simplex is that it appears in *every* Delaunay triangulation of a point set. If a point set has multiple Delaunay triangulations, they differ only in their choices of simplices that are merely Delaunay. Hence, if a point set has no four cocircular points, it has only one Delaunay triangulation.

Let us prove these facts. Loosely speaking, the following theorem says that strongly Delaunay simplices intersect nicely.

Theorem 9. *Let σ be a strongly Delaunay simplex, and let τ be a Delaunay simplex. Then $\sigma \cap \tau$ is either empty or a shared face of both σ and τ.*

Proof. If τ is a face of σ, the theorem follows immediately. Otherwise, τ has a vertex v that σ does not have. Because τ is Delaunay, it has an empty circumcircle C_τ. Because σ is strongly Delaunay, it has an empty circumcircle C_σ that does not pass through v, illustrated in Figure 2.12. But v lies on C_τ, so $C_\sigma \neq C_\tau$.

The intersection of circumcircles $C_\sigma \cap C_\tau$ contains either zero, one, or two points. In the first two cases, the theorem follows easily, so suppose it is two points w and x, and let ℓ be the unique line through w and x. On one side of ℓ, an arc of C_σ encloses an arc of C_τ, and because C_σ is empty, no vertex of τ lies on this side of ℓ. Symmetrically, no vertex of σ lies on the other side of ℓ. Therefore, $\sigma \cap \tau \subset \ell$. It follows that $\sigma \cap \ell$ is either $\emptyset$, $\{w\}$, $\{x\}$, or the edge wx. The same is true of $\tau \cap \ell$, and therefore of $\sigma \cap \tau$. ■

Theorem 9 leads us to see that if a point set has several Delaunay triangulations, they differ only by the simplices that are not strongly Delaunay.

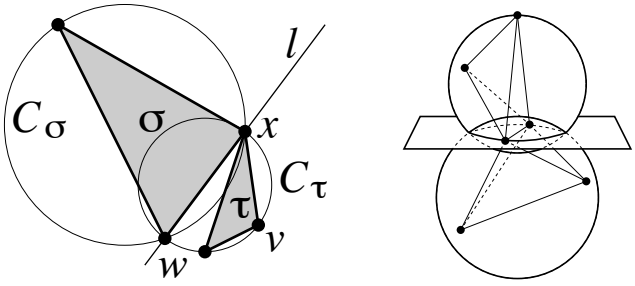


Figure 2.12: A strongly Delaunay simplex σ intersects any Delaunay simplex τ at a shared face of both. The illustration at right foreshadows the fact that this result holds for higher-dimensional Delaunay triangulations too.

Theorem 10. *Every Delaunay triangulation of a point set contains every strongly Delaunay simplex.*

Proof. Let $\mathcal{T}$ be any Delaunay triangulation of a point set V. Let σ be any strongly Delaunay simplex. Let p be a point in the relative interior of σ.

Some Delaunay simplex τ in $\mathcal{T}$ contains the point p. By Theorem 9, $\sigma \cap \tau$ is a shared face of σ and τ. But $\sigma \cap \tau$ contains p, which is in the relative interior of σ, so $\sigma \cap \tau = \sigma$. Therefore, σ is a face of τ, so $\sigma \in \mathcal{T}$. ■

An immediate consequence of this theorem is that "most" point sets—at least, most point sets with randomly perturbed real coordinates—have just one Delaunay triangulation.

Corollary 11. *Let V be a point set. Suppose no four points in V lie on a common empty circle. Then V has at most one Delaunay triangulation.*

Proof. Because no four points lie on a common empty circle, every Delaunay simplex is strongly Delaunay. By Theorem 10, every Delaunay triangulation of V contains every Delaunay simplex. By definition, no Delaunay triangulation contains a triangle that is not Delaunay. Hence, the Delaunay triangulation is uniquely defined as the set of all Delaunay triangles and their faces. ■

2.8 Constrained Delaunay Triangulations in the Plane

As planar Delaunay triangulations maximize the minimum angle, do they solve the problem of triangular mesh generation? No, for two reasons illustrated in Figure 2.13. First, skinny triangles might appear anyway. Second, the Delaunay triangulation of a domain's vertices might not respect the domain's boundary. Both these problems can be solved by introducing additional vertices, as illustrated.

An alternative solution to the second problem is to use a *constrained Delaunay triangulation* (CDT). A CDT is defined with respect to a set of points and *segments* that demarcate the domain boundary. Every segment is required to become an edge of the CDT. The triangles of a CDT are *not* required to be Delaunay; instead, they must be *constrained Delaunay*, a property that partly relaxes the empty circumcircle property.

One virtue of a CDT is that it can respect arbitrary segments without requiring the insertion of any additional vertices (besides the vertices of the segments). Another is that the CDT inherits the Delaunay triangulation's optimality: among all triangulations of a point set *that include all the segments*, the CDT

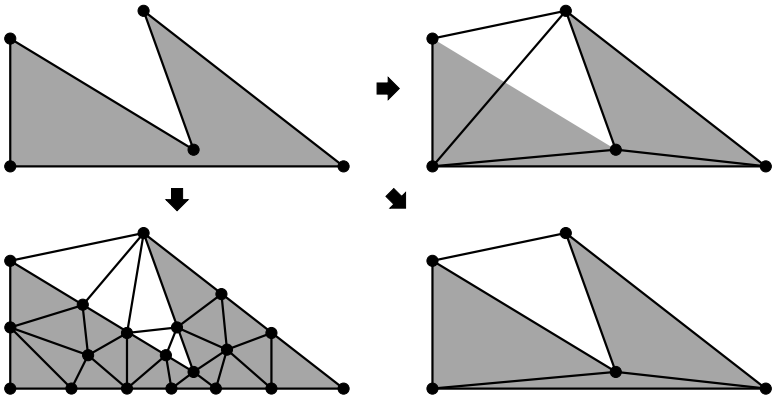


Figure 2.13: The Delaunay triangulation (upper right) may omit domain edges and contain skinny triangles. A Steiner Delaunay triangulation (lower left) can fix these faults by introducing new vertices. A constrained Delaunay triangulation (lower right) fixes the first fault without introducing new vertices.

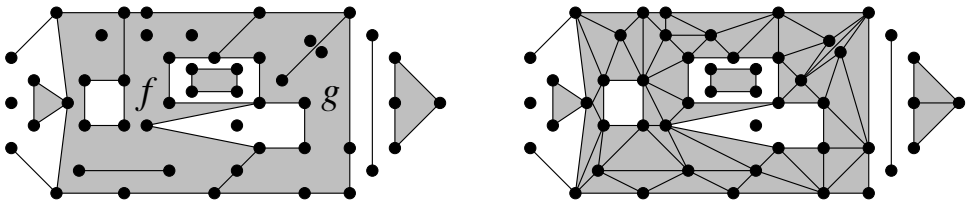


Figure 2.14: A two-dimensional piecewise linear complex and its constrained Delaunay triangulation. Each polygon may have holes, slits, and vertices in its interior.

maximizes the minimum angle [74], minimizes the largest circumcircle, and minimizes the largest min-containment circle. CDTs in the plane were mathematically formalized by Lee and Lin [74] in 1986, though algorithms that unwittingly construct CDTs appeared much sooner [53, 89].

2.8.1 Piecewise Linear Complexes and their Triangulations

The domain over which a CDT is defined (and the input to a CDT construction algorithm) is not just a set of points, but rather a complex composed of points, edges, and polygons, illustrated in Figure 2.14. The purpose of the edges is to dictate that triangulations of the complex must contain those edges. The purpose of the polygons is to specify the region to be triangulated. The polygons are not necessarily convex, and they may have holes.

Definition 31 (piecewise linear complex; segment; wall)**.** *In the plane, a* piecewise linear complex *(PLC)* $\mathcal{X}$ *is a finite set of vertices, edges, and polygons that satisfies the following properties.*

- *The vertices and edges in* $\mathcal{X}$ *form a simplicial complex. That is,* $\mathcal{X}$ *contains both vertices of every edge in* $\mathcal{X}$*, and the relative interior of an edge in* $\mathcal{X}$ *intersects no vertex in* $\mathcal{X}$ *nor any other edge in* $\mathcal{X}$*.*
- *For each polygon* f *in* $\mathcal{X}$*, the boundary of* f *is a union of edges in* $\mathcal{X}$*.*
- *If two polygons in* $\mathcal{X}$ *intersect, their intersection is a union of edges and vertices in* $\mathcal{X}$*. (For example, in Figure 2.14* $f \cap g$ *is a union of three edges and a vertex in* $\mathcal{X}$*.) This rule implies that two polygons' interiors cannot intersect.)*

Following Ruppert [102], the edges in a PLC $\mathcal{X}$ are called segments *to distinguish them from other edges in a triangulation of $\mathcal{X}$. The polygons in a PLC are called* walls.

The underlying space *of a PLC $\mathcal{X}$, denoted $|\mathcal{X}|$, is the union of its contents; that is, $|\mathcal{X}| = \bigcup_{f \in \mathcal{X}} f$. Usually, the underlying space is the domain to be triangulated.*[1]

Every simplicial complex and every polyhedral complex is a PLC. But PLCs are more general, and not just because they permit nonconvex polygons. As Figure 2.14 illustrates, segments and isolated vertices can float in a wall's interior, constraining how the wall can be triangulated. One purpose of these floating constraints is to permit the application of boundary conditions at appropriate locations in a mesh of a PLC.

Whereas the faces of a simplex are defined in a way that depends solely on the simplex, and the faces of a convex polyhedron are too, the faces of a wall are defined in a fundamentally different way that depends on both the wall and the PLC it is a part of. An edge of a wall might be a union of several segments in the PLC; these segments and their vertices are faces of the wall. A PLC may contain segments and edges that lie in the relative interior of a wall; these are also considered to be faces of the wall, because they constrain how the wall can be subdivided into triangles.

Definition 32 (face of a linear cell)**.** *The* faces *of a linear cell (polygon, edge, or vertex) f in a PLC $\mathcal{X}$ are the linear cells in $\mathcal{X}$ that are subsets of f, including f itself.*

A triangulation of $\mathcal{X}$ must cover every wall and include every segment.

Definition 33 (triangulation of a planar PLC)**.** *Let $\mathcal{X}$ be a PLC in the plane. A* triangulation of $\mathcal{X}$ *is a simplicial complex $\mathcal{T}$ such that*

- *$\mathcal{X}$ and $\mathcal{T}$ have the same vertices,*
- *$\mathcal{T}$ contains every edge in $\mathcal{X}$ (and perhaps additional edges), and*
- $|\mathcal{T}| = |\mathcal{X}|$.

It is not difficult to see that a simplex can appear in a triangulation of $\mathcal{X}$ only if it *respects* $\mathcal{X}$.

Definition 34 (respect)**.** *A simplex σ* respects *a PLC $\mathcal{X}$ if $\sigma \subseteq |\mathcal{X}|$ and for every $f \in \mathcal{X}$ that intersects σ, $f \cap \sigma$ is a union of faces of σ. (Usually, but not always, that union is one face of σ or σ itself.) In other words, f fully includes every face of σ whose relative interior intersects f.*

Theorem 12. *Every simple polygon has a triangulation. Every PLC in the plane has a triangulation too.*

Proof: Let P be a simple polygon. If P is a triangle, it clearly has a triangulation. Otherwise, consider the following procedure for triangulating P. Let $\angle uvw$ be a corner of P having an interior angle less than 180°. Two such corners are found by letting v be the lexicographically least or greatest vertex of P.

If the open edge uw lies strictly in P's interior, then cutting $\triangle uvw$ from P yields a polygon having one edge fewer; triangulate it recursively. Otherwise, $\triangle uvw$ contains at least one vertex of P besides u, v, and w, as illustrated in Figure 2.15. Among those vertices, let x be the vertex furthest from the line aff uw. The open edge vx must lie strictly in P's interior, because if it intersected an edge of P, that edge would have a vertex further from aff uw. Cutting P at vx produces two simple polygons, each with fewer edges than P; triangulate them recursively. In either case, the procedure produces a triangulation of P.

[1] If you take the vertices and edges of a planar PLC and discard the polygons, you have a simplicial complex with no triangles. This complex is called a *planar straight line graph* (PSLG). Most publications about CDTs take a PSLG as the input, and assume that the CDT should cover the PSLG's convex hull. PLCs are more expressive, as they can restrict the triangulation to a nonconvex region of the plane.

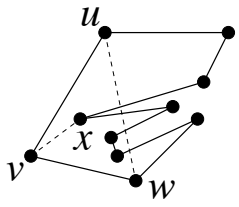


Figure 2.15: The edge vx cuts this simple polygon into two simple polygons.

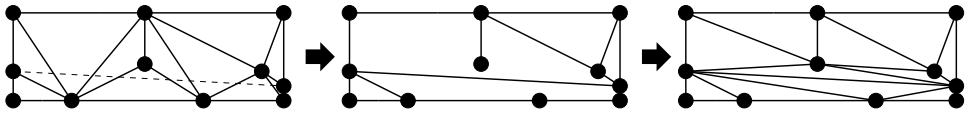


Figure 2.16: Inserting a segment into a triangulation.

Let $\mathcal{X}$ be a planar PLC. Consider the following procedure for triangulating $\mathcal{X}$. Begin with an arbitrary triangulation of the vertices in $\mathcal{X}$, such as the lexicographic triangulation described in Section 2.1. Examine each segment in $\mathcal{X}$ to see if it is already an edge of the triangulation. Insert each missing segment into the triangulation by deleting all the edges and triangles that intersect its relative interior, creating the new segment, and retriangulating the two polygonal cavities thus created (one on each side of the segment), as illustrated in Figure 2.16. The cavities might not be simple polygons, because they might have edges dangling in their interiors, as shown. But it is straightforward to verify that the procedure discussed above for triangulating a simple polygon works equally well for a cavity with dangling edges.

The act of inserting a segment never deletes another segment, because two segments in $\mathcal{X}$ cannot cross. Therefore, after every segment is inserted, the triangulation contains all of them. Finally, delete any simplices not included in $|\mathcal{X}|$. ■

Definition 33 does not permit $\mathcal{T}$ to have vertices absent from $\mathcal{X}$, but mesh generation usually entails adding new vertices to guarantee that the triangles have high quality. This motivates the notion of a Steiner triangulation.

Definition 35 (Steiner triangulation of a planar PLC; Steiner Delaunay triangulation; Steiner point)**.** *Let $\mathcal{X}$ be a PLC in the plane. A* Steiner triangulation of $\mathcal{X}$*, also known as a* conforming triangulation of $\mathcal{X}$ *or a* mesh of $\mathcal{X}$*, is a simplicial complex $\mathcal{T}$ such that*

- *$\mathcal{T}$ contains every vertex in $\mathcal{X}$ (and perhaps additional vertices),*
- *every edge in $\mathcal{X}$ is a union of edges in $\mathcal{T}$, and*
- $|\mathcal{T}| = |\mathcal{X}|$.

The new vertices in $\mathcal{T}$, absent from $\mathcal{X}$, are called Steiner points.

A Steiner Delaunay triangulation *of $\mathcal{X}$, also known as a* conforming Delaunay triangulation *of $\mathcal{X}$, is a Steiner triangulation of $\mathcal{X}$ in which every simplex is Delaunay.*

If the Delaunay triangulation of the vertices in a planar PLC $\mathcal{X}$ does not respect all the segments in $\mathcal{X}$, it is always possible to find a Steiner Delaunay triangulation of $\mathcal{X}$ by adding Steiner points, as illustrated at

lower left in Figure 2.13. Unfortunately, the number of Steiner points might be large. The best algorithm to date, by Bishop [13], triangulates a PLC having m segments and n vertices with the addition of $O(m^{2.5} + mn)$ Steiner points. Edelsbrunner and Tan [49] exhibit a PLC requiring $\Theta(mn)$ Steiner points. Closing the gap between the $O(m^{2.5} + mn)$ and $\Omega(mn)$ bounds remains an open problem. The large number of Steiner points that some PLCs need motivates the constrained Delaunay triangulation, which needs none.

2.8.2 The Constrained Delaunay Triangulation

Constrained Delaunay triangulations (CDTs) offer a way to force a triangulation to respect the edges in a PLC without introducing new vertices, while maintaining some of the advantages of Delaunay triangulations. However, it is necessary to relax the requirement that all triangles be Delaunay. The terminology can be confusing: whereas every Steiner Delaunay triangulation is a Delaunay triangulation (of some point set), constrained Delaunay triangulations generally are not.

Recall the Delaunay Lemma: a triangulation of a point set is Delaunay if and only if every edge is locally Delaunay. Likewise, there is a Constrained Delaunay Lemma (Section 2.8.3) that offers the simplest definition of a CDT: a triangulation of a PLC is constrained Delaunay if and only if every edge is locally Delaunay *or* a segment. Thus, a CDT differs from a Delaunay triangulation in three ways: it is not necessarily convex, it is required to contain the edges in a PLC, and those edges are exempted from being locally Delaunay.

The defining characteristic of a CDT is that every triangle is constrained Delaunay, as defined below.

Definition 36 (visibility)**.** *Two points x and y are* visible *to each other if the line segment xy respects $\mathcal{X}$; recall Definition 34. We also say that x and y can* see *each other. A linear cell in $\mathcal{X}$ that intersects the relative interior of xy but does not include xy is said to* occlude *the visibility between x and y.*

Definition 37 (constrained Delaunay)**.** *In the context of a PLC $\mathcal{X}$, a simplex σ is* constrained Delaunay *if it satisfies the following three conditions.*

- *$\mathcal{X}$ contains σ's vertices.*
- *σ respects $\mathcal{X}$.*
- *There is a circumcircle of σ that encloses no vertex in $\mathcal{X}$ that is visible from a point in the relative interior of σ.*

Figure 2.17 illustrates examples of a constrained Delaunay edge e and a constrained Delaunay triangle τ. Bold lines indicate PLC segments. Although e has no empty circumcircle, the depicted circumcircle of e encloses no vertex that is visible from the relative 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 τ encloses two vertices, but both are hidden from the interior of τ by segments, so τ is constrained Delaunay.

Definition 38 (constrained Delaunay triangulation)**.** *A* constrained Delaunay triangulation *(CDT) of a PLC $\mathcal{X}$ is a triangulation of $\mathcal{X}$ in which every triangle is constrained Delaunay.*

Figure 2.18 illustrates a PLC, a Delaunay triangulation of its vertices, and a constrained Delaunay triangulation of the PLC. In the CDT, every triangle is constrained Delaunay, every edge that is not a PLC segment is constrained Delaunay, and every vertex is trivially constrained Delaunay.

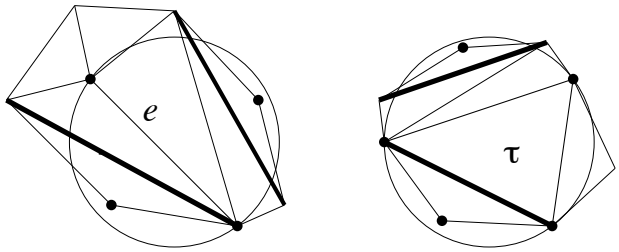


Figure 2.17: The edge e and triangle τ are constrained Delaunay. Bold lines represent segments.

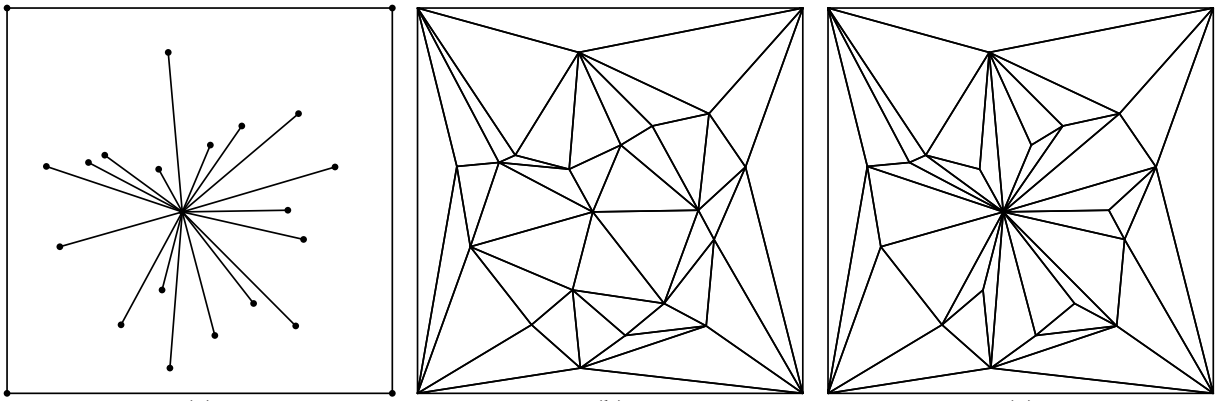


Figure 2.18: (a) A piecewise linear complex. (b) The Delaunay triangulation of its vertices. (c) Its constrained Delaunay triangulation.

CDTs and Steiner Delaunay triangulations are two different ways to force a triangulation to conform to the boundary of a geometric domain. CDTs partly sacrifice the Delaunay property for the benefit of requiring no new vertices. For mesh generation, new vertices are usually needed anyway to obtain good triangles, so many Delaunay meshing algorithms use Steiner Delaunay triangulations. But some algorithms use a hybrid of CDTs and Steiner Delaunay triangulations because it helps to reduce the number of new vertices. A *Steiner CDT* or *conforming CDT* of $\mathcal{X}$ is a Steiner triangulation of $\mathcal{X}$ in which every triangle is constrained Delaunay.

2.8.3 Properties of the Constrained Delaunay Triangulation

For every property of Delaunay triangulations discussed in this chapter, there is an analogous property of constrained Delaunay triangulations. This section summarizes them. Proofs are omitted, but each of them is a straightforward extension of the corresponding proof for Delaunay triangulations.

The Delaunay Lemma generalizes to CDTs, and provides a useful alternative definition: a triangulation of a PLC $\mathcal{X}$ is a CDT if and only if every one of its edges is locally Delaunay *or* a segment in $\mathcal{X}$.

Theorem 13 (Constrained Delaunay Lemma)**.** *Let $\mathcal{T}$ be a triangulation of a PLC $\mathcal{X}$. The following three statements are equivalent.*

- *Every triangle in $\mathcal{T}$ is constrained Delaunay (i.e. $\mathcal{T}$ is constrained Delaunay).*
- *Every edge in $\mathcal{T}$ not in $\mathcal{X}$ is constrained Delaunay.*

- *Every edge in $\mathcal{T}$ not in $\mathcal{X}$ is locally Delaunay.* ■

One way to construct a constrained Delaunay triangulation of a PLC $\mathcal{X}$ is to begin with any triangulation of $\mathcal{X}$. Apply the flip algorithm, modified so that it never flips a segment: repeatedly choose any edge of the triangulation that is not in $\mathcal{X}$ and not locally Delaunay, and flip it. When no such edge survives, the Delaunay Lemma tells us that the triangulation is constrained Delaunay.

Theorem 14. *Given a triangulation of a PLC having n vertices, the modified flip algorithm (which never flips a PLC segment) terminates after $O(n^2)$ edge flips, yielding a constrained Delaunay triangulation.* ■

Corollary 15. *Every PLC has a constrained Delaunay triangulation.* ■

The CDT has the same optimality properties as the Delaunay triangulation, except that the optimality is with respect to a smaller set of triangulations—those that include the PLC's edges.

Theorem 16. *Among all the triangulations of a PLC, every constrained Delaunay triangulation maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle.* ■

A sufficient but not necessary condition for the CDT to be unique is that no four vertices are cocircular.

Theorem 17. *If no four vertices in a PLC lie on a common circle, then the PLC has one unique constrained Delaunay triangulation.* ■

Chapter 3

Algorithms for Constructing Delaunay Triangulations

The first published Delaunay triangulation algorithm I know of appears in a 1967 paper by J. Desmond Bernal and John Finney [12]. Bernal was a father of structural biology who discovered the structure of graphite and was awarded a Stalin Peace Prize. Finney, Bernal's last Ph.D. student, implemented a program that produces a three-dimensional Voronoi diagram and used it to characterize the structures of liquids, amorphous metal alloys, protein molecules, and random packings. Finney's is the brute force algorithm: test every possible tetrahedron (every combination of four vertices) to see if its circumsphere is empty, taking $O(n^5)$ time—or more generally, $O(n^{d+2})$ time for d-dimensional Delaunay triangulations.

Besides this brute force algorithm and the flip algorithm, there are three classic types of algorithm for constructing Delaunay triangulations.

Gift-wrapping—also called *graph traversal*, *pivoting*, and *incremental search*—is an obvious algorithm that is rediscovered frequently [53, 26, 79, 121, 123]. Gift-wrapping algorithms construct Delaunay triangles one at a time, using the previously computed triangles as a seed on which new triangles crystallize. They are closely related to *advancing front methods* for mesh generation. Gift-wrapping generalizes easily to CDTs and to higher dimensions, and it is easy to implement, but it is difficult to make fast. Section 3.8 describes a basic gift-wrapping algorithm that triangulates n points in the plane in $O(n^2)$ worst-case time, or a PLC in the plane with n vertices and m segments in $O(n^2 m)$ time. The bottleneck of gift-wrapping is identifying new triangles, so the fastest gift-wrapping algorithms are differentiated by sweep orderings for constructing the triangles [108, 51] or sophisticated vertex search strategies [47, 120].

In the decision-tree model of computation, sets of n points in the plane sometimes require $\Omega(n \log n)$ time to triangulate. The first Delaunay triangulation algorithm to run in optimal $O(n \log n)$ time was the 1975 *divide-and-conquer algorithm* of Shamos and Hoey [111], subsequently simplified by Lee and Schachter [75] and Guibas and Stolfi [60] and sped up by Dwyer [46]. The divide-and-conquer algorithm partitions a set of points into two halves separated by a line, recursively computes the Delaunay triangulation of each subset, and merges the two triangulations into one. This algorithm remains the fastest planar Delaunay triangulator in practice [112], and the reader interested in implementing it is urged to read the guide by Guibas and Stolfi [60], which includes detailed pseudocode. However, the divide-and-conquer strategy is not fast in three dimensions.

Incremental insertion algorithms insert vertices into a Delaunay triangulation one at a time, always restoring the Delaunay property to the triangulation before inserting another vertex. Some incremental insertion algorithms run in worst-case optimal time. The fastest three-dimensional Delaunay triangulators in

practice are in this class. Moreover, the difference between a Delaunay triangulation algorithm and a modern Delaunay mesh generator is that the former is given all the vertices at the outset, whereas the latter uses the triangulation to decide where to place additional vertices, making incremental insertion obligatory. Sections 3.3–3.5 study incremental insertion, and Section 5.4 introduces a more sophisticated vertex ordering method called a *biased randomized insertion order* that speeds up incremental insertion for large points sets.

All three types of algorithm extend to constrained Delaunay triangulations. There are a divide-and-conquer algorithm [34] and a gift-wrapping algorithm [109] that both run in worst-case optimal $O(n \log n)$ time, but because they are complicated, these algorithms are rarely implemented.

The most commonly used CDT construction method in practice is incremental insertion: first construct a Delaunay triangulation of the PLC's vertices, then insert the PLC's segments one by one. The algorithms commonly used to perform segment insertion in practice are slow, but a specialized incremental algorithm described in Section 3.9 runs in expected $O(n \log n + n \log^2 m)$ time. Realistic PLCs have few segments long enough to cross many edges, and it is typical to observe $O(n \log n)$ running time in practice.

3.1 The Orientation and Incircle Predicates

Most geometric algorithms perform a mix of combinatorial and numerical computations. The numerical computations are usually packaged as *geometric primitives* of two types: *geometric constructors* that create new entities, such as the point where two specified lines intersect, and *geometric predicates* that determine relationships among entities, such as whether or not two lines intersect at all. Many Delaunay triangulation algorithms require just two predicates, called the *orientation* and *incircle* tests.

The most used predicate in computational geometry is the orientation test. Let a, b, and c be three points in the plane. Consider a function Orient2D(a, b, c) that returns a positive value if the points a, b, and c are arranged in counterclockwise order, a negative value if the points are in clockwise order, and zero if the points are collinear. Another interpretation, important for many geometric algorithms, is that Orient2D returns a positive value if a lies to the left of the directed line $\vec{bc}$. The orientation test can be implemented as a matrix determinant that computes the signed area of the parallelogram determined by the vectors $a - c$ and $b - c$,

$$\text{Orient2D}(a, b, c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{3.1}$$

$$= \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}. \tag{3.2}$$

These expressions extend to higher dimensions by adding rows and columns for additional points and coordinate axes. Given four points a, b, c, and d in E^3, define Orient3D(a, b, c, d) to be the signed volume of the parallelepiped determined by the vectors $a - d$, $b - d$, and $c - d$. It is positive if the points occur in the orientation illustrated in Figure 3.1, negative if they occur in the mirror-image orientation, and zero if the four points are coplanar. You can apply a *right-hand rule*: orient your right hand with fingers curled to follow the circular sequence bcd. If your thumb points toward a, Orient3D is positive.

$$\text{Orient3D}(a, b, c, d) = \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix} = \begin{vmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ b_x - d_x & b_y - d_y & b_z - d_z \\ c_x - d_x & c_y - d_y & c_z - d_z \end{vmatrix}.$$

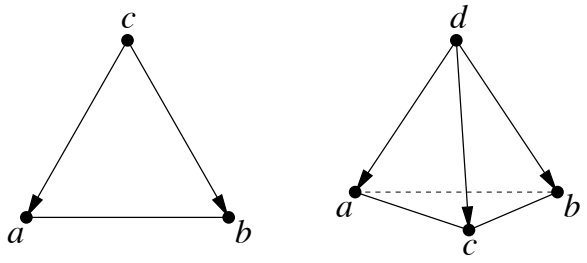


Figure 3.1: A triangle and a tetrahedron, both having positive orientation.

Most planar Delaunay triangulation algorithms use a predicate InCircle(a, b, c, d) that returns a positive value if d lies inside the unique (and possibly degenerate) circle through a, b, and c, assuming that the latter three points occur in counterclockwise order around the circle. InCircle returns zero if and only if all four points lie on a common circle or line. InCircle is derived from Orient3D and Lemma 1, which shows that testing whether a point is inside a circle is equivalent to an orientation test on the points lifted by the parabolic lifting map.

$$\text{InCircle}(a,b,c,d) = \begin{vmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ d_x & d_y & d_x^2+d_y^2 & 1 \end{vmatrix} \tag{3.3}$$

$$= \begin{vmatrix} a_x-d_x & a_y-d_y & (a_x-d_x)^2+(a_y-d_y)^2 \\ b_x-d_x & b_y-d_y & (b_x-d_x)^2+(b_y-d_y)^2 \\ c_x-d_x & c_y-d_y & (c_x-d_x)^2+(c_y-d_y)^2 \end{vmatrix}. \tag{3.4}$$

These expressions also extend easily to higher dimensions. Let a, b, c, d, and e be five points in E^3, with the first four ordered so that Orient3D(a, b, c, d) is nonnegative. The function InSphere(a, b, c, d, e) is positive if e lies inside the sphere passing through a, b, c, and d; negative if e lies outside the sphere; and zero if all five points are cospherical or coplanar.

$$\text{InSphere}(a,b,c,d,e) = \begin{vmatrix} a_x & a_y & a_z & a_x^2+a_y^2+a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2+b_y^2+b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2+c_y^2+c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2+d_y^2+d_z^2 & 1 \\ e_x & e_y & e_z & e_x^2+e_y^2+e_z^2 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} a_x-e_x & a_y-e_y & a_z-e_z & (a_x-e_x)^2+(a_y-e_y)^2+(a_z-e_z)^2 \\ b_x-e_x & b_y-e_y & b_z-e_z & (b_x-e_x)^2+(b_y-e_y)^2+(b_z-e_z)^2 \\ c_x-e_x & c_y-e_y & c_z-e_z & (c_x-e_x)^2+(c_y-e_y)^2+(c_z-e_z)^2 \\ d_x-e_x & d_y-e_y & d_z-e_z & (d_x-e_x)^2+(d_y-e_y)^2+(d_z-e_z)^2 \end{vmatrix}.$$

Orient2D, Orient3D, InCircle, and InSphere have the symmetry property that interchanging any two of their parameters reverses their sign. If the points a, b, c occur in clockwise order, InCircle behaves as if the circle's outside were its inside. Likewise, if Orient3D(a, b, c, d) is negative, the sign returned by InSphere is reversed.

Expressions (3.1) and (3.2) can be shown to be equivalent by simple algebraic transformations, as can Expressions (3.3) and (3.4) with a little more effort. Expressions (3.2) and (3.4) should be strongly preferred

Procedure	Purpose
AddTriangle(u, v, w)	Add a positively oriented triangle $\triangle uvw$
DeleteTriangle(u, v, w)	Delete a positively oriented triangle $\triangle uvw$
Adjacent(u, v)	Return a vertex w such that $\triangle uvw$ is a positively oriented triangle
AdjacentOne(u)	Return vertices v, w such that $\triangle uvw$ is a positively oriented triangle

Figure 3.2: An interface for a triangulation data structure.

over Expressions (3.1) and (3.3) for fixed precision floating-point computation, because they lose far less accuracy to roundoff error. Ideally, some form of exact arithmetic should be used to perform these tests, or the triangulation algorithms cannot be guaranteed to work correctly.

3.2 A Dictionary Data Structure for Triangulations

Two data structures are commonly used to implement triangulation algorithms: edge-based data structures, of which the best known is the doubly connected edge list [86], and triangle-based data structures. What these two data structures have in common is that records represent edges or triangles, and the records store pointers that point at neighboring edges or triangles. Many implementations of triangulation algorithms read and change these pointers directly; experience shows that these implementations are difficult to code and debug.

Here, I advocate an interface that does not expose pointers to the triangulation algorithms that use the data structure. Triangulation algorithms access the triangulation in a natural way, by adding or deleting triangles specified by their vertices. It is wholly the responsibility of the triangulation storage library to determine triangle adjacencies, and to correctly maintain any pointers it uses internally. This policy, which originates with Blelloch et al. [17, 16], improves programmer productivity and simplifies debugging.

The interface appears in Figure 3.2. Two procedures, AddTriangle and DeleteTriangle, create and delete triangles by specifying the vertices of a triangle, ordered so that all the triangles stored in the data structure have positive orientation. The data structure enforces the invariant that only two triangles may adjoin an edge, and only one on each side of the edge. Therefore, if the data structure contains a positively oriented triangle $\triangle uvw$ and an application calls AddTriangle(u, v, x), the triangle $\triangle uvx$ is rejected and the data structure does not change.

At least two query operations are supported. The procedure Adjacent(u, v) returns a vertex w if the triangulation includes a positively oriented triangle $\triangle uvw$, or the empty set otherwise. Adjacent(u, v) and Adjacent(v, u) return different triangles, on opposite sides of the edge uv. The procedure AdjacentOne(u) identifies an arbitrary triangle having vertex u, or returns the empty set if no such triangle exists.

A fast way to implement Adjacent efficiently is to store each triangle $\triangle uvw$ three times in a hash table, keyed on the directed edges uv, vw, and wu. A hash table can store triangles and query edges in expected constant time. For a more compact representation that is still reasonably fast, see Blandford, Blelloch, Clemens, and Kadow [16].

Unfortunately, it takes substantial additional memory to guarantee that the AdjacentOne query will run fast. Many algorithms for Delaunay triangulation and meshing can be implemented without it, so I recommend using AdjacentOne as little as possible, and settling for a slow but memory-efficient implementation, perhaps even searching the entire hash table. A good compromise implementation is to maintain an array

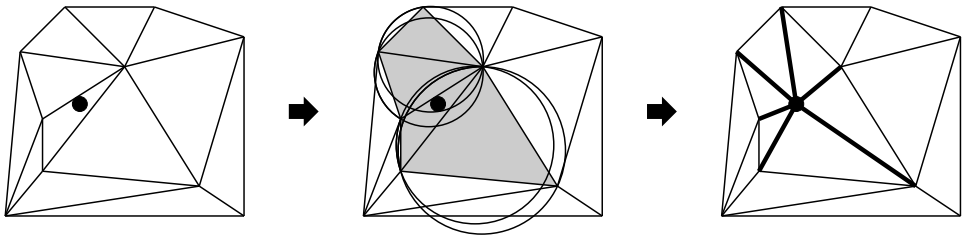


Figure 3.3: The Bowyer–Watson algorithm in the plane. At left, a Delaunay triangulation and a new vertex to insert. At center, every triangle whose circumcircle encloses the new vertex is shaded. These triangles are no longer Delaunay. At right, the shaded triangles disappear, replaced by new triangles that connect the new vertex to the edges of the cavity.

that stores, for each vertex u, a vertex v such that the most recently added triangle adjoining u also had v for a vertex. When AdjacentOne(u) is invoked, it looks up the edges uv and vu in the hash table to find an adjoining triangle in expected constant time. The problem with this implementation is that the triangles having edge uv may have been subsequently deleted, in which case a triangle adjoining u must be found some other way (e.g. searching the entire hash table). However, observe that this catastrophe will not occur if every triangle deletion is followed by triangle creations that cover all the same vertices—which is true of most of the algorithms discussed in this book.

The interface and data structure extend easily to permit the storage of edges that are not part of any triangle. For example, an edge uv that is not an edge of any triangle can be represented by storing $\triangle uvg$ keyed on the directed edge uv, where g is a special entity called the *ghost vertex*.

3.3 Inserting a Vertex into a Delaunay Triangulation

Lawson [73] invented the first algorithm for inserting a vertex into a Delaunay triangulation and restoring the Delaunay property, but it works only in the plane. A slightly faster algorithm that works in any dimensionality was discovered independently by Bowyer [20], Hermeline [62, 63], and Watson [128]. Bowyer and Watson simultaneously submitted it to *Computer Journal* and found their articles published side by side.

Consider inserting a new vertex v into a Delaunay triangulation. If a triangle's circumcircle encloses v, that triangle is no longer Delaunay, so it must be deleted. This suggests the *Bowyer–Watson algorithm*.

- Find one triangle whose circumcircle encloses v.
- Find all the others (in time linear in their number) by a depth-first search in the triangulation.
- Delete them all, evacuating a polyhedral cavity, which is shaded in Figure 3.3.
- For each edge of this cavity, create a new triangle joining it with v, as illustrated.

The first step is called *point location*. Most Delaunay mesh generation algorithms generate new vertices inside the circumcircles of badly shaped or oversized triangles, in which case point location is free. However, point location is not free for the domain vertices provided as input to the mesh generator. Locating these points in the triangulation is sometimes the most costly and complicated part of the incremental insertion algorithm. Incremental insertion is really a class of Delaunay triangulation algorithms, differentiated by their point location methods. Clarkson and Shor [39] describe a point location method that helps a randomized

```
InsertVertex(u, v, w, x)
{ u is the vertex to insert. △vwx is a positively oriented triangle whose circumcircle encloses u. }
1       DeleteTriangle(v, w, x)
2       DigCavity(u, v, w)
3       DigCavity(u, w, x)
4       DigCavity(u, x, v)

DigCavity(u, v, w)
{ u is a new vertex. Is the oriented triangle △uvw Delaunay? }
5       x ← Adjacent(w, v)      { Find △wvx opposite the edge vw from u }
6       if x ≠ ∅      { Do nothing if the triangle has already been deleted }
7           if InCircle(u, v, w, x) > 0
8               DeleteTriangle(w, v, x)      { △uvw and △wvx are not Delaunay }
9               DigCavity(u, v, x)
10              DigCavity(u, x, w)
11          else AddTriangle(u, v, w)      { vw is an edge of the cavity and △uvw is Delaunay }
```

Figure 3.4: Algorithm for inserting a vertex u into a Delaunay triangulation, given a triangle $\triangle vwx$ whose circumcircle encloses u.

incremental insertion algorithm to construct the Delaunay triangulation of n vertices in expected $O(n \log n)$ time. Section 5.5 describes a point location method that seems to be even faster in practice, albeit only if the vertices are carefully ordered as described in Section 5.4.

Figure 3.4 gives pseudocode for vertex insertion, omitting the point location step. It interleaves the second, third, and fourth steps of the Bowyer–Watson algorithm (rather than performing them in sequence), thereby achieving simplicity and speed although obscuring the algorithm's workings.

The following three results demonstrate the correctness of the Bowyer–Watson algorithm if a correct point location algorithm is available. The first result shows that the deleted triangles—those that are no longer Delaunay—comprise a star-shaped polygon. This fact guarantees that a depth-first search (the second Bowyer–Watson step) will find all the triangles that are no longer Delaunay, and that the third and fourth Bowyer–Watson steps yield a simplicial complex.

Lemma 18. *The union of the triangles whose circumcircles enclose v is a connected star-shaped polygon, meaning that for every point p in the polygon, the polygon includes the line segment pv.*

Proof: Prior to the insertion of v, the triangulation is Delaunay, so all of its edges are locally Delaunay. Let τ be a triangle whose circumcircle encloses v. Let p be any point in the interior of τ. By the same inductive reasoning employed in the proof of the Delaunay Lemma (Section 2.4), every triangle that intersects the line segment pv also has v inside its circumcircle. The result follows. ■

The lifting map gives us intuition for why Lemma 18 is unsurprising: it says that the facets of the lifted triangulation that are visible from the lifted vertex v^+ are connected.

The key to proving that the updated triangulation is Delaunay is to show that all its edges are Delaunay and apply the Delaunay Lemma. The following lemma shows that every newly created edge is strongly Delaunay, and therefore appears in every Delaunay triangulation of the vertices.

Lemma 19. *Let v be a newly inserted vertex. Let τ be a triangle that is deleted because its circumcircle encloses v. Let w be a vertex of τ. The edge vw is strongly Delaunay.*

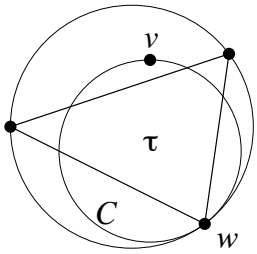


Figure 3.5: Because τ was Delaunay before v was inserted, vw is strongly Delaunay.

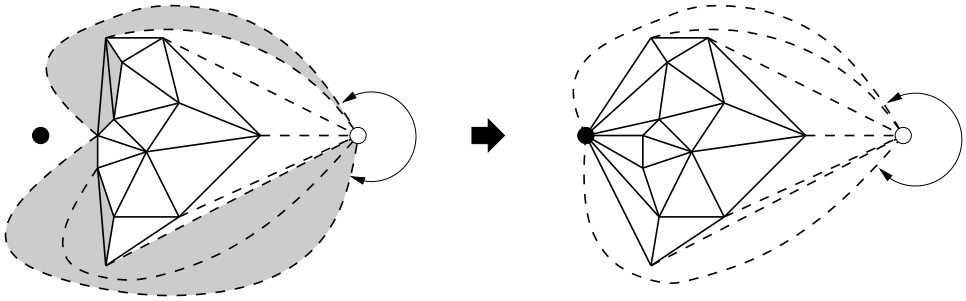


Figure 3.6: Inserting a vertex outside the triangulation. The open circle is the ghost vertex. The circular arrow indicates two ghost edges that are really the same edge. Three ghost triangles and three solid triangles (shaded) are deleted and replaced with two new ghost triangles and six new solid triangles.

Proof: See Figure 3.5. The circumcircle of τ encloses no vertex but v. Let C be the circle that is tangent to τ's circumcircle at w and passes through v. C demonstrates that vw is strongly Delaunay. ■

Theorem 20. *A triangulation produced by applying the Bowyer–Watson algorithm to a Delaunay triangulation is Delaunay.*

Proof: It follows from Lemma 18 that the update produces a triangulation of the point set augmented with the new point. All the surviving old triangles are Delaunay; otherwise they would have been deleted. It follows that their edges are Delaunay too. By Lemma 19, all of the newly created edges are Delaunay as well. By the Delaunay Lemma, the new triangulation is Delaunay. ■

3.4 Inserting a Vertex Outside a Delaunay Triangulation

The Bowyer–Watson algorithm works only if the newly inserted vertex lies in the triangulation. However, there is an elegant way to represent a triangulation so that the algorithm, with almost no changes, can insert a vertex outside the triangulation equally well. Imagine that every edge on the boundary of the triangulation adjoins a *ghost triangle*, as illustrated in Figure 3.6. The third vertex of every ghost triangle is the *ghost vertex*, a vertex "at infinity" shared by every ghost triangle. Every ghost triangle has two *ghost edges* that adjoin the ghost vertex. A triangle that is not a ghost is called a *solid triangle*.

The ghost triangles are explicitly stored in the triangulation data structure. They are not merely cosmetic; they make it possible for the Bowyer–Watson algorithm to efficiently traverse the triangulation boundary, and thus they are essential to obtaining an incremental insertion algorithm with optimal running time.

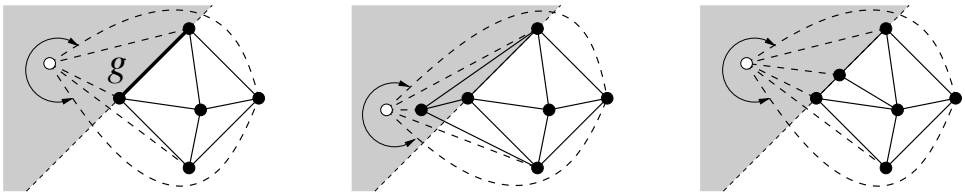


Figure 3.7: The ghost triangle uvg is deleted if a new vertex is inserted in the shaded open halfplane (as at center) or on uv (as at right). The union of the open halfplane and uv is the outer halfplane of uvg.

Consider an edge uv on the boundary of a triangulation, directed clockwise around the boundary. Define a positively oriented ghost triangle $\triangle uvg$, where g is the ghost vertex. Like any other triangle, $\triangle uvg$ has a circumcircle—albeit a degenerate one—and must be deleted if a new vertex is inserted "inside" it. The definition of "circumcircle" is a bit tricky, though. The circumcircle degenerates to the line aff uv, which divides the plane into two open halfplanes.

There are two cases in which the ghost triangle $\triangle uvg$ must be deleted (i.e. uv is no longer a boundary edge of the triangulation), both illustrated in Figure 3.7: if a vertex is inserted in the open halfplane on the other side of aff uv from the triangulation, or if a vertex is inserted on the open edge uv. Call the union of these two regions the *outer halfplane* of uv. It is neither an open nor closed halfplane, but something in between. It is the set of points enclosed by the circumcircle of $\triangle uvg$ in the limit as g moves away from the triangulation.

A new vertex inserted outside the triangulation causes at least one ghost triangle to be deleted, and perhaps some solid (non-ghost) triangles as well. Two new boundary edges, two new ghost triangles, and an arbitrary number of solid triangles are created, as illustrated in Figure 3.6.

Ghost triangles have an intuitive interpretation in terms of the lifting map. Imagine that in E^3, the solid triangles are lifted to the paraboloid, and the ghost triangles and ghost edges are vertical—parallel to the z-axis. By magic, the ghost vertex is interpreted as being directly above every other vertex at an infinite height. The faces of the convex hull of this three-dimensional point set, including the magic ghost vertex, are in one-to-one correspondence with the faces and ghost faces of the Delaunay triangulation.

A popular alternative to ghost triangles is to enclose the input vertices in a giant triangular bounding box, illustrated in Figure 3.8. After all the vertices have been inserted, every triangle having a bounding box vertex is deleted. The difficulty with this approach is that the bounding box vertices may leave concave divots in the triangulation if they are too close, and it is not easy to determine how far away they need to be. One solution to this problem is to compute a weighted Delaunay triangulation, assigning the three bounding box vertices weights of negative infinity. These three infinite weights must be incomparable—say, ∞, 2^∞, and 2^{2^∞}—so that InCircle tests involving two of the bounding box vertices operate consistently. This approach seems to run more slowly (perhaps by 10%) than the ghost triangle implementation. Another solution is to fill the divots with the segment insertion algorithm described in Section 3.9.

3.5 The Running Time of Vertex Insertion

How expensive is vertex insertion, leaving out the cost of point location? This section considers two cases: the worst case, and the expected case when vertices are inserted in random order. The latter case is a part of an incremental insertion algorithm that computes the Delaunay triangulation of n vertices in expected $O(n \log n)$ time, and it also introduces an elegant algorithm analysis technique called *backward analysis*.

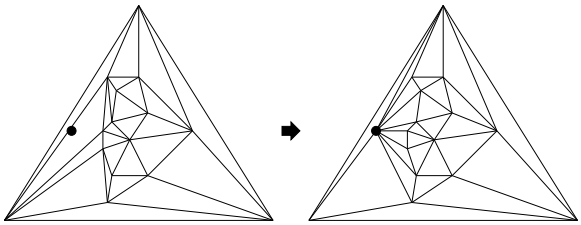


Figure 3.8: Enclosing the vertices in a large triangular bounding box.

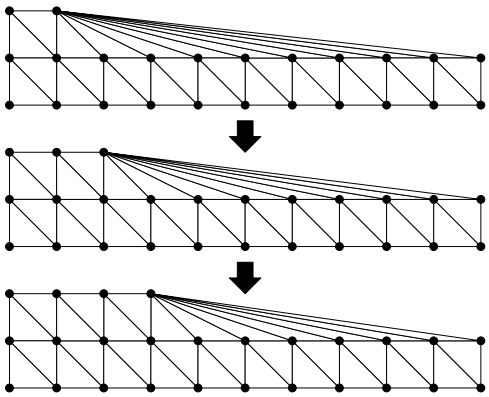


Figure 3.9: Each vertex insertion can delete $\Theta(n)$ triangles and create $\Theta(n)$ others.

Figure 3.9 illustrates the worst case. A single vertex insertion can delete $\Theta(n)$ triangles and create $\Theta(n)$ others, taking $\Theta(n)$ time. Moreover, this dismal performance can be repeated for $\Theta(n)$ successive vertex insertions. Therefore, the incremental insertion algorithm for constructing a Delaunay triangulation takes $\Theta(n^2)$ time if the vertices and their insertion order are chosen badly. The grid arrangement and vertex ordering in the figure are common in practice.

Fortunately, there are better ways to order the vertex insertion operations. The *randomized incremental insertion algorithm* inserts the vertices in random order, with each permutation of the vertices being equally likely. Surprisingly, the *expected* number of triangles created by each successive vertex insertion operation is less than six, as Theorem 21 below shows. The catch is that all the vertices must be known in advance, so that a random permutation can be computed. The randomized algorithm is excellent for creating an initial triangulation of the vertices of a domain, but its analysis does not apply to the vertices that are subsequently generated during mesh generation, because their order cannot be randomized. Nevertheless, the theorem provides intuition for why constant-time vertex insertion is so commonly observed in mesh generation.

Theorem 21. *Let V be a set of n vertices in the plane. Let $\langle v_1, v_2, \ldots, v_n \rangle$ be a permutation of V chosen uniformly at random from the set of all such permutations. For $i \in [0, n]$, let $\mathcal{T}_i$ be the Delaunay triangulation constructed by inserting the first i vertices in order. When v_i is inserted into $\mathcal{T}_{i-1}$ to create $\mathcal{T}_i$, the expected number of new triangles (including ghost triangles) created is less than six. An expected total of $O(n)$ triangles are created and deleted during the n vertex insertions that construct $\mathcal{T}_n$.*

This theorem is most easily proved with *backward analysis*, a remarkable analysis technique that Seidel [110] summarizes thus: "*Analyze an algorithm as if it was running backwards in time, from output to input*." Imagine that instead of inserting a randomly chosen vertex into $\mathcal{T}_{i-1}$, you are deleting a randomly chosen vertex from $\mathcal{T}_i$. Because a random permutation written backward is still a random permutation, each vertex in $\mathcal{T}_i$ is deleted with equal probability.

Proof of Theorem 21: For every vertex v of $\mathcal{T}_i$, the number of triangles adjoining v, including ghost triangles, is equal to the degree of v, counting one ghost edge if v is on the boundary of the triangulation. When v_i is inserted into $\mathcal{T}_{i-1}$ to construct $\mathcal{T}_i$, every new triangle created has v_i for a vertex. Therefore, the expected number of new triangles created is equal to the expected degree of v_i.

There is one technical difficulty: if four vertices of $\mathcal{T}_i$ lie on a common empty circle, then $\mathcal{T}_i$ depends on the order in which the vertices are inserted. Thus, let S_i be the Delaunay subdivision of $\{v_1, v_2, \ldots, v_i\}$, wherein triangles in $\mathcal{T}_i$ sharing a common circumcircle are merged into a polygon. Recall from Section 2.2 that S_i contains the strongly Delaunay edges of $\mathcal{T}_i$ and no others, and is therefore unique. By Lemma 19, every edge adjoining v_i in $\mathcal{T}_i$ is strongly Delaunay, so the degree of v_i in $\mathcal{T}_i$ is equal to the degree of v_i in S_i.

Because the permutation is chosen uniformly at random, each vertex of S_i is equally likely to be v_i. The expected degree of a randomly chosen vertex in S_i (or any planar graph) is less than six, by the following reasoning.

Let $i + 1$, e, and f denote the number of vertices, edges, and triangles of $\mathcal{T}_i$, respectively, with the ghost vertex, ghost edges, and ghost triangles included. By Euler's formula, $i + 1 - e + f = 2$. Each triangle has three edges, and each edge is shared by two triangles, so $2e = 3f$. Eliminating f from Euler's formula gives $e = 3i - 3$. Each edge has two vertices, so the total number of edge-vertex incidences is $6i - 6$, and the average degree of a non-ghost vertex in $\mathcal{T}_i$ is less than $6 - 6/i$. The average degree of a non-ghost vertex in S_i cannot be greater.

Each vertex insertion creates, in expectation, fewer than six new triangles, so the expected total number of triangles created during the n vertex insertions is in $O(n)$. A triangle cannot be deleted unless it is created first, so the expected total number of triangles deleted is also in $O(n)$. ■

Theorem 21 bounds not only the number of structural changes, but also the running time of the depth-first search in the Bowyer–Watson algorithm. This search visits all the triangles that are deleted and all the triangles that share an edge with a deleted triangle. Depth-first search takes time linear in the number of visited triangles, and therefore linear in the number of deleted triangles.

It follows that the expected running time of the randomized incremental insertion algorithm, *excluding* point location, is in $O(n)$. We shall see that point location is the dominant cost of the algorithm.

A general fact about randomized algorithms is that there is a chance that they will run much, much more slowly than their expected running time, but the probability of that is exceedingly small. If the incremental insertion algorithm gets unlucky and endures a slow vertex insertion like those depicted in Figure 3.9, other, faster vertex insertions will probably make up for it. The probability that many such slow vertex insertions will occur in one run is tiny, but it can happen.

The argument in the proof of Theorem 21, which is the first known use of backward analysis in computational geometry, originates in a paper by Chew [36]. Chew's paper describes an algorithm for computing the Delaunay triangulation of a convex polygon, or deleting a vertex from a Delaunay triangulation, in expected linear time. Backward analysis was popularized by a charming paper by Seidel [110].

3.6 Inserting a Vertex into a Constrained Delaunay Triangulation

To "insert a vertex into a CDT" is to take as input the CDT of some PLC $\mathcal{X}$ and a new vertex v to insert, and produce the CDT of $\mathcal{X} \cup \{v\}$. An implementation might also support the insertion of a vertex v on a

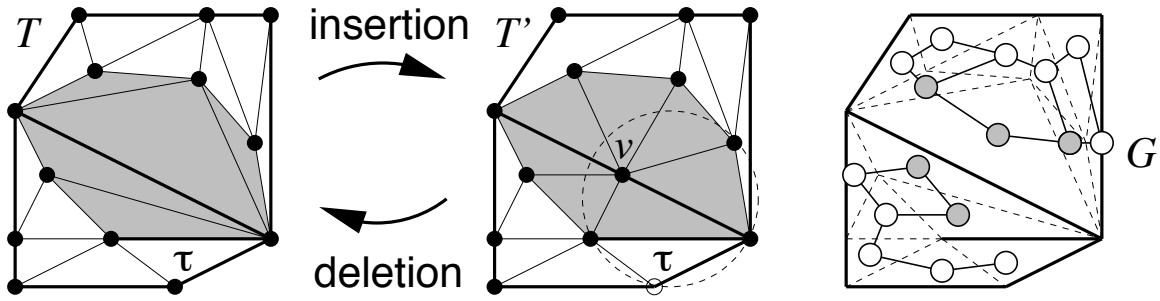


Figure 3.10: Inserting or deleting a vertex v in a CDT. Bold edges are segments. The shaded polygon is the union of the deleted/created triangles. Simplices not intersecting the interior of the shaded polygon are constrained Delaunay before and after. When v is inserted, depth-first search on the graph G identifies the deleted triangles. Observe that although v lies inside τ's circumcircle, v's insertion does not delete τ because G does not connect τ to any deleted triangle.

segment $s \in \mathcal{X}$, in which case the algorithm subdivides s into two subsegments s_1 and s_2 having vertex v, and produces the CDT of $\mathcal{X} \cup \{v, s_1, s_2\} \setminus \{s\}$.

With a small change, the Bowyer–Watson algorithm can insert a vertex v into a CDT, as Figure 3.10 illustrates. The change, of course, is that the algorithm deletes the triangles that are no longer *constrained* Delaunay. Fortunately, it is possible to enumerate those triangles without performing expensive visibility tests. To accomplish that, the first step—point location—finds the triangle that contains v. There may be two such triangles, if v lies on a triangulation edge. The second step—the depth-first search that identifies triangles that are no longer constrained Delaunay—should never walk across a segment. As Figure 3.10 shows, this restriction suffices to ensure that only triangles whose interiors are visible from v will be deleted. If v lies on a segment in $\mathcal{X}$, depth-first searches must be run from both of the two adjoining triangles. The third and fourth steps of the Bowyer–Watson algorithm do not change. In a straightforward extension of the proofs in Section 3.3, one can show that the depth-first search finds all the triangles that are no longer constrained Delaunay, the cavity is always star-shaped, and the algorithm works correctly.

3.7 The Gift-Wrapping Step

Gift-wrapping algorithms rely on a simple procedure that constructs a triangle adjoining a specified edge. Let $e = uw$ be an oriented edge. The *front* of e is the open halfplane to the left of $\vec{uw}$, and a positively oriented triangle $\triangle uwv$ is said to be *in front of* e. The *back* of e is the open halfplane to the right of $\vec{uw}$, and a positively oriented triangle $\triangle wuv$ is said to be *behind* e.

During the execution of a gift-wrapping algorithm, an oriented edge constructed by the algorithm is said to be *unfinished* if the algorithm has not yet identified the triangle in front of the edge. A gift-wrapping step *finishes* the edge by constructing that triangle, or by determining that there is no such triangle because the edge is on the boundary of the domain.

A edge e has an infinite number of circumcircles, any one of which can be continuously deformed into any other such that every intermediate circle is also a circumcircle of e. Imagine beginning with a circumcircle that encloses no vertex in front of e, then deforming it so it expands in front of e and shrinks behind e, always remaining a circumcircle of e, as illustrated in Figure 3.11. As the circumcircle deforms, its center always lies on e's bisector.

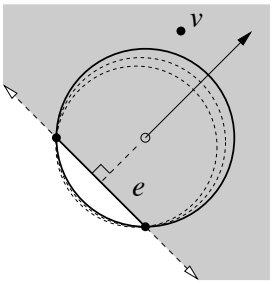


Figure 3.11: An empty circumcircle of e, expanding in search of a vertex v.

```
FINISH(e, V, K)
{ e is an oriented edge. V is the set of vertices in a PLC X. K is the set of segments in X. }
{ FINISH returns a triangle that finishes e, or ∅ if none exists. }
1     τ ← ∅
2     p ← an arbitrary point in the relative interior of e (e.g. its midpoint)
3     for each vertex v ∈ V
4         if v is in front of e and (τ = ∅ or the circumcircle of τ encloses v)
5             if no segment s ∈ K occludes the visibility between v and p
6                 τ ← conv(e ∪ v)
7     return τ
```

Figure 3.12: Algorithm to gift-wrap one constrained Delaunay triangle. For an ordinary Delaunay triangulation, omit Line 5.

Eventually, the expanding portion of the circumcircle might touch a vertex v that is visible from the relative interior of e, in which case the gift-wrapping step constructs $\tau = \text{conv}(e \cup v)$, thereby finishing e. Lemma 22 below shows that τ is constrained Delaunay if e is constrained Delaunay or a segment. Alternatively, the expanding portion of the circumcircle might never touch a vertex, in which case e is on the boundary of the convex hull of the vertices.

Although the expanding circumcircle gives the right intuition for which vertex is chosen, the algorithm that implements a gift-wrapping step works the opposite way, by shrinking the front of the circumcircle: it scans through the vertices in front of e and remembers which vertex, so far, minimizes the portion of the circumcircle in front of e.

Figure 3.12 gives pseudocode for the gift-wrapping step. One gift-wrapping step takes $O(n)$ time for a Delaunay triangulation, or $O(nm)$ time for a CDT, where $n = |V|$ is the number of vertices and $m = |K|$ is the number of segments. Line 5 of the pseudocode accounts for the factor of m.

Lemma 22. *If the edge e is constrained Delaunay or a segment, the algorithm* FINISH *returns a constrained Delaunay triangle τ.*

Proof. There is a total ordering of the set of all e's circumcircles such that, if one circumcircle precedes another, the former circumcircle encloses the portion of the latter in front of e. It is easy to see that, among the vertices in V that are in front of e and visible from e's relative interior, Lines 3–6 choose the vertex v such that $\tau = \text{conv}(e \cup v)$ has the last circumcircle in this ordering. Hence, τ's circumcircle encloses no vertex that is in front of e and visible from e's relative interior.

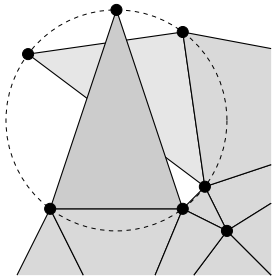


Figure 3.13: A gift-wrapping failure because of cocircular vertices.

If e is a segment, then no vertex that is behind e and inside τ's circumcircle is visible from the interior of τ, so τ is constrained Delaunay.

If e is constrained Delaunay, it has a circumcircle that encloses no vertex visible from the relative interior of e. This circumcircle does not enclose v, so it must enclose every point on or behind e that τ's circumcircle encloses. It follows that τ's circumcircle encloses no vertex visible from the interior of τ, so τ is constrained Delaunay. ■

3.8 The Gift-Wrapping Algorithm

This section describes a basic gift-wrapping algorithm for constructing Delaunay trianglations and CDTs. Be forewarned that if the input PLC or point set has four corcircular vertices, gift-wrapping can make decisions that are mutually inconsistent and fail to construct a valid triangulation. Figure 3.13 depicts a simple, unconstrained example where Delaunay gift-wrapping fails. Gift-wrapping can be modified to handle these inputs by symbolically perturbing the vertex weights, or by identifying groups of cocircular vertices that can see each other and triangulating them all at once.

The gift-wrapping algorithm begins with the segments of the PLC, upon which the constrained Delaunay triangles crystallize one by one. The core of the algorithm is a loop that selects an unfinished edge and finishes it by invoking the procedure Finish in Figure 3.12. Often, the new triangle finishes more than one unfinished edge. To detect this circumstance, the algorithm maintains the unfinished edges in a dictionary (e.g. a hash table) so they can be quickly looked up by their vertex indices. The data structure in Section 3.2 is easily modified to serve this purpose while also storing the triangulation. Pseudocode for the gift-wrapping algorithm appears in Figure 3.14.

The algorithm can construct Delaunay triangulations too, but the pseudocode assumes that $\mathcal{X}$ contains at least one segment that can serve as a seed upon which to build the triangulation. When there are no segments, seed the algorithm by constructing one strongly Delaunay edge—an arbitrary vertex and its nearest neighbor will do—and entering it (twice, with both orientations) in the dictionary.

The algorithm takes $O(n^2)$ time for a Delaunay triangulation, or $O(n^2m)$ time for a CDT, where $n = |V|$ is the number of vertices and $m = |K|$ is the number of segments. These are not impressive speeds, especially when compared to the incremental insertion or divide-and-conquer algorithms. However, gift-wrapping is easy to implement and fast enough for small jobs, such as retriangulating the cavity evacuated when a vertex is deleted or a segment is inserted in a triangulation. Moreover, there are several ways to speed up gift-wrapping that make it more practical.

```
GiftWrapCDT(V, K)
{ V is the set of vertices in a PLC X. K is the set of segments in X. }
1     for each segment s ∈ K that adjoins a wall on one side only
2         Enter s in the dictionary, oriented so its front adjoins a wall
3     while the dictionary is not empty
          { Loop invariant: the dictionary contains all the unfinished edges. }
4         Remove an oriented edge e from the dictionary
5         τ ← Finish(e, V, K)
6         if τ ≠ ∅
7             AddTriangle(t)
8             for each oriented edge f of τ except e
9                 if f is in the dictionary
10                    Remove f from the dictionary
11                else Enter f in the dictionary with reversed orientation
                           (facing away from τ)
```

Figure 3.14: Gift-wrapping algorithm for constructing a CDT.

One specialized class of gift-wrapping algorithms are *sweepline algorithms* that construct the triangles in a disciplined order, making it possible to determine which vertex finishes each edge without an exhaustive search. Fortune [51] developed such an algorithm for Delaunay triangulations. Seidel [109] extended it to CDTs. Both algorithms run in $O(n \log n)$ time.

Another way to avoid exhaustive search is to subdivide the plane into square buckets, record the vertices in their respective buckets, and finish each edge by searching through the buckets in an appropriate order. Dwyer [47] shows that if the vertices are distributed uniformly at random in a disk, this technique finishes each face in $O(1)$ expected time, so an entire Delaunay triangulation can be constructed in $O(n)$ expected time. Moreover, the algorithm extends to higher dimensions, still with expected linear running time! Unfortunately, this method does not extend easily to CDTs, and not all real-world point sets are so nicely distributed. It is easy to construct a point set for which most of the points fall into one bucket.

3.9 Inserting a Segment into a Constrained Delaunay Triangulation

To "insert a segment into a CDT" is to take as input the CDT of a PLC $\mathcal{X}$ and a new segment s to insert, and produce the CDT of $\mathcal{X} \cup \{s\}$. It is only meaningful if $\mathcal{X} \cup \{s\}$ is a valid PLC—that is, $\mathcal{X}$ already contains the vertices of s (otherwise, they must be inserted first, as described in Section 3.6), and the relative interior of s intersects no segment or vertex in $\mathcal{X}$. This section presents a gift-wrapping algorithm for segment insertion. It appears in an article by Anglada [3], but it was part of the folklore of the field before that paper appeared. It is difficult to trace who thought of the algorithm first.

Let $\mathcal{T}$ be the CDT of $\mathcal{X}$. If $s \in T$, then $\mathcal{T}$ is also the CDT of $\mathcal{X} \cup \{s\}$. Otherwise, the algorithm begins by deleting from $\mathcal{T}$ the edges and triangles that intersect the relative interior of s. All of $\mathcal{T}$'s simplices not deleted remain constrained Delaunay after s is inserted. Next, the algorithm adds s to the complex, and it retriangulates the two polygonal cavities on each side of s with constrained Delaunay triangles. Recall from Figure 2.16 that the cavities might have segments dangling in their interiors.

Let P be one of the two polygons; its edges include s. The algorithm GiftWrapCDT could gift-wrap P starting from any edge of P, but there are several advantages to gift-wrapping from s outward. First,

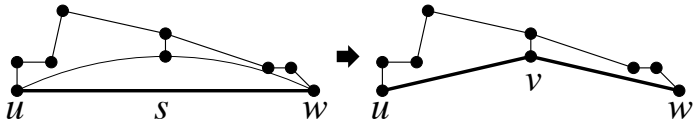


Figure 3.15: Gift-wrapping one triangle from a newly inserted segment s.

gift-wrapping from s makes it possible to eliminate the visibility tests (Line 5 of FINISH) and the dictionary of unfinished edges, speeding up the algorithm by a factor of m. Second, gift-wrapping from s gives the best likelihood of subdividing P into two half-sized polygons, improving the speed of the algorithm. Third, the algorithm is guaranteed to work correctly even if X has four or more cocircular vertices. Be forewarned that gift-wrapping without visibility tests does not work correctly for all polygons, but it works for segment insertion.

The cavity retriangulation algorithm is as follows. Begin by gift-wrapping one constrained Delaunay triangle in front of s, as illustrated in Figure 3.15. Let u and w be the vertices of s, and let $\triangle uwv$ be the positively oriented triangle produced by the gift-wrapping step. The cavity retriangulation algorithm calls itself recursively on the oriented edges uv and vw (if they are unfinished).

The running time of the first gift-wrapping step is proportional to the number m of vertices of P. If we are lucky, it will split P into two polygons of roughly half the size, and the recursive calls will also enjoy balanced splits, so the time required to triangulate P will be in $O(m \log m)$. In the worst case, each gift-wrapping step might simply cut one triangle off of P without subdividing P into smaller polygons, and it will take $\Theta(m^2)$ time to triangulate P. In practice, if m is large, P is probably long and thin and will enjoy well-balanced recursive calls.

Chapter 4

Three-Dimensional Delaunay Triangulations

Three-dimensional triangulations are sometimes called tetrahedralizations. Delaunay tetrahedralizations are not quite as effective as planar Delaunay triangulations at producing elements of good quality, but they are nearly as popular in the mesh generation literature as their two-dimensional cousins. Many properties of Delaunay triangulations in the plane generalize to higher dimensions, but many of the optimality properties do not. Notably, Delaunay tetrahedralizations do not maximize the minimum angle (whether plane angle or dihedral angle). Figure 4.1 depicts a three-dimensional counterexample. The hexahedron at the top is the convex hull of its five vertices. The Delaunay triangulation of those vertices, to the left, includes a thin tetrahedron known as a *sliver* or *kite*, whose vertices are nearly coplanar and whose dihedral angles can be arbitrarily close to 0° and 180°. A triangulation of the same vertices that is not Delaunay, at lower right, has better quality.

This chapter surveys Delaunay triangulations and constrained Delaunay triangulations in three (and occasionally more) dimensions. Constrained Delaunay triangulations generalize uneasily to three dimensions, because there are polyhedra that do not have any tetrahedralization at all.

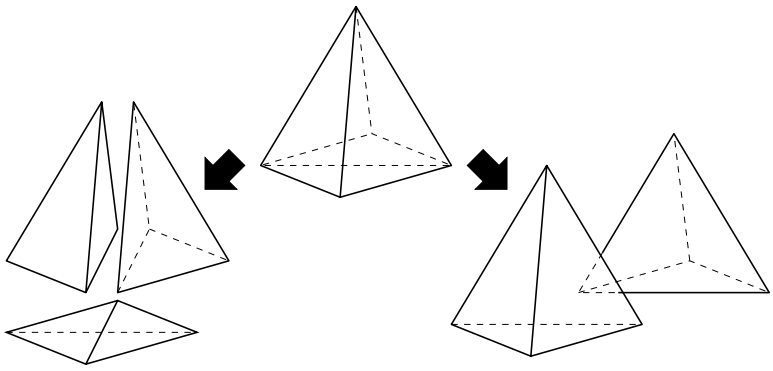


Figure 4.1: This hexahedron has two tetrahedralizations. The Delaunay tetrahedralization at left includes an arbitrarily thin sliver tetrahedron. The non-Delaunay tetrahedralization at right consists of two nicely shaped tetrahedra.

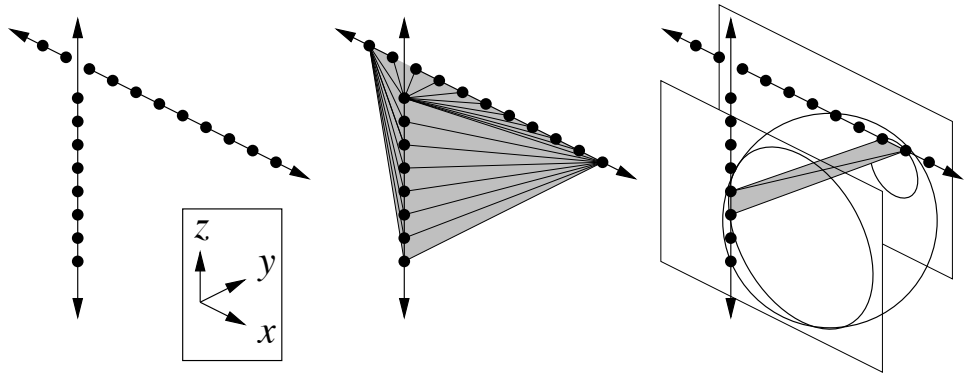


Figure 4.2: At center, the Delaunay tetrahedralization of the points at left. At right, the circumsphere of one Delaunay tetrahedron with two cross-sections showing it is empty.

4.1 Triangulations of a Point Set in E^d

Definition 26 in Section 2.1 defines a triangulation of a set of points to be a simplicial complex whose vertices are the points and whose union is the convex hull of the points. With no change, the definition holds in any finite dimension d. Figures 4.1–4.4 illustrate triangulations of point sets in three dimensions. Every finite point set in E^d has a triangulation; for example, the lexicographic triangulation of Section 2.1 also generalizes to higher dimensions with no change.

Let V be a set of n points in E^d. Recall from Section 2.1 that if all the points in V are collinear, they have one triangulation having n vertices and $n - 1$ collinear edges connecting them. This is true regardless of d; the triangulation is one-dimensional, although it is embedded in E^d. More generally, if the affine hull of V is k-dimensional, then every triangulation of V is a k-dimensional triangulation embedded in E^d: the simplicial complex has at least one k-simplex but no $(k + 1)$-simplex.

The *complexity* of a triangulation is its total number of simplices of all dimensions. Whereas a planar triangulation of n points has $O(n)$ triangles and edges, a surprising property of higher-dimensional triangulations is that they can have superlinear complexity. Figure 4.2 shows a triangulation of n points that has $\Theta(n^2)$ edges and tetrahedra, which is asymptotically the largest number possible in three dimensions. Every vertex lies on one of two non-intersecting lines, and there is one tetrahedron for each pairing of an edge on one line and an edge on the other. This is the *only* triangulation of these points, and it is Delaunay.

An n-vertex triangulation in E^d can have a maximum of $\Theta(n^{\lceil d/2 \rceil})$ d-simplices. Of course, most applications do best with linear-complexity meshes. The existence of triangulations with much higher complexity is a potential pitfall for mesh generation algorithms, especially if the input vertices resemble those in Figure 4.2.

4.2 The Delaunay Triangulation in E^d

Delaunay triangulations generalize easily to higher dimensions. Let V be a finite set of points in E^d, for $d \geq 1$. Let σ be a k-simplex (for any $k \leq d$) whose vertices are in V. Let S be a hypersphere in E^d; S is a *circumsphere*, or *circumscribing sphere*, of σ if S passes through every vertex of σ. If $k = d$, then σ has a unique circumsphere; otherwise, σ has infinitely many circumspheres.

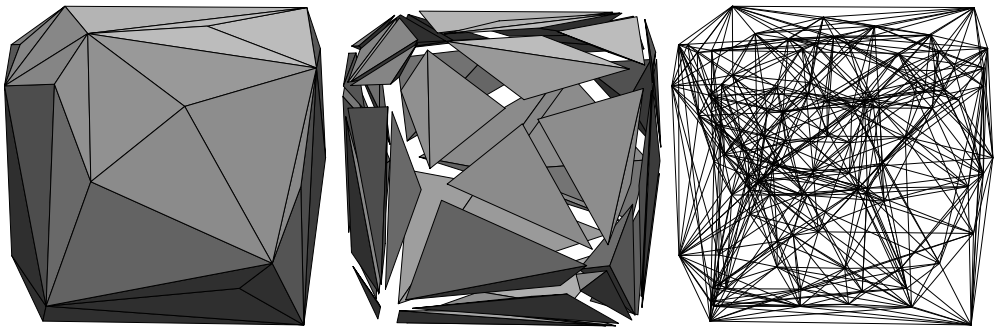


Figure 4.3: Three renderings of a Delaunay tetrahedralization.

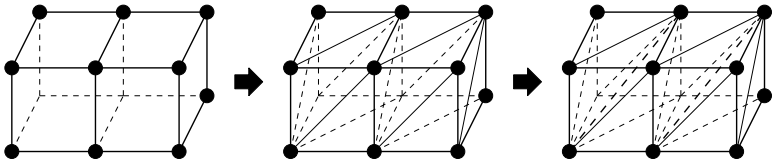


Figure 4.4: A Delaunay subdivision comprising two cubic cells and their faces. The least-vertex Delaunay triangulation subdivides each 2-face into triangles adjoining the face's lexicographically minimum vertex, and likewise subdivides each 3-face into tetrahedra.

A simplex σ is *Delaunay* if there exists a circumsphere of σ that encloses no point in V. Clearly, every face of a Delaunay simplex is Delaunay too. A simplex σ is *strongly Delaunay* if there exists a circumsphere S of σ such that no point in V lies inside *or on* S, except the vertices of σ. Every point in V is trivially a strongly Delaunay vertex. A *Delaunay triangulation* $\mathrm{Del}\, V$ of V is a triangulation of V in which every d-simplex is Delaunay, as Figure 4.2 shows. Figure 4.3 depicts a more typical Delaunay tetrahedralization, with complexity linear in the number of vertices.

The parabolic lifting map generalizes to higher dimensions too. It maps each point $p = (p_1, p_2, \ldots, p_d) \in E^d$ to its *lifted companion*, the point $p^+ = (p_1, p_2, \ldots, p_d, p_1^2 + p_2^2 + \cdots + p_d^2)$ in E^{d+1}. Consider the $(d+1)$-dimensional convex hull of the lifted points, $V^+ = \{v^+ : v \in V\}$. Projecting the downward-facing faces of $\mathrm{conv}(V^+)$ to E^d yields a polyhedral complex called the *Delaunay subdivision* of V, which is easily transformed to its complement, the Voronoi diagram of V.

If V is *generic*, its Delaunay subdivision is simplicial and V has exactly one Delaunay triangulation,

Definition 39 (generic)**.** *Let V be a point set in E^d. Let k be the dimension of the affine hull of V. V is* generic *if no $k+2$ points in V lie on a common hypersphere.*

If V if not generic, the Delaunay subdivision may have non-simplicial faces; recall Figure 2.4. In that case, V has multiple Delaunay triangulations, which differ according to how the non-simplicial faces are triangulated.

Whereas each non-simplicial face in a two-dimensional Delaunay subdivision can be triangulated independently, in higher dimensions the triangulations are not always independent. Figure 4.4 illustrates a set of twelve points in E^3 whose Delaunay subdivision includes two cubic cells that share a square 2-face. The square face can be divided into two triangles in two different ways, and each cube can be divided into five or six tetrahedra in several ways, but they are not independent: the triangulation of the square face constrains how both cubes are triangulated.

A *least-vertex triangulation* provides one way to safely subdivide a polyhedral complex into a simplicial complex. To construct it, triangulate the 2-faces through the d-faces in order of increasing dimension. To triangulate a non-simplicial k-face f, subdivide it into k-simplices of the form $\text{conv}(v \cup g)$, where v is the lexicographically minimum vertex of f, and g varies over the $(k-1)$-simplices on f's subdivided boundary that do not contain v. The choice of the lexicographically minimum vertex of each face ensures that the face triangulations are compatible with each other.

4.3 Properties of Delaunay Triangulations in E^d

Many properties of planar Delaunay triangulations discussed in Chapter 2 generalize to higher dimensions. A few of them are summarized below. Proofs are omitted, but each of them is a straightforward extension of the corresponding proof for two dimensions.

Recall that a *facet* of a polyhedral complex is a $(d-1)$-face, and a facet of a triangulation is a $(d-1)$-simplex. The Delaunay Lemma provides an alternative definition of a Delaunay triangulation: a triangulation of a point set in which every facet is locally Delaunay. A facet f in a triangulation $\mathcal{T}$ is said to be *locally Delaunay* if it is a face of fewer than two d-simplices in $\mathcal{T}$, or it is a face of exactly two d-simplices τ_1 and τ_2 and it has a circumsphere enclosing no vertex of τ_1 nor τ_2. (Equivalently, the circumsphere of τ_1 encloses no vertex of τ_2. Equivalently, the circumsphere of τ_2 encloses no vertex of τ_1.)

Theorem 23 (Delaunay Lemma)**.** *Let $\mathcal{T}$ be a triangulation of a finite set V of points in E^d. The following three statements are equivalent.*

- *Every d-simplex in $\mathcal{T}$ is Delaunay (i.e. $\mathcal{T}$ is Delaunay).*
- *Every facet in $\mathcal{T}$ is Delaunay.*
- *Every facet in $\mathcal{T}$ is locally Delaunay.* ■

As in the plane, a generic point set has exactly one Delaunay triangulation, composed of every strongly Delaunay simplex. The following three theorems have essentially the same proofs as in Section 2.7.

Theorem 24. *Let σ be a strongly Delaunay simplex, and let τ be a Delaunay simplex. Then $\sigma \cap \tau$ is either empty or a shared face of both σ and τ.*

Theorem 25. *Every Delaunay triangulation of a point set contains every strongly Delaunay simplex.*

Corollary 26. *A generic point set has exactly one Delaunay triangulation.*

4.4 The Optimality of the Delaunay Triangulation in E^d

Some optimality properties of Delaunay triangulations hold in any dimension. Rippa [98] investigates the use of two-dimensional triangulations for piecewise linear interpolation of a quadratic bivariate function. If the function is isotropic—of the form $\alpha(x^2+y^2)+\beta x+\gamma y+\delta$—then the Delaunay triangulation minimizes the interpolation error measured in the L_q-norm for every $q \geq 1$, compared with all other triangulations of the same points. Melissaratos [80] generalizes Rippa's result to higher dimensions. (If the function is not isotropic, but it is parabolic rather than hyperbolic, then the optimal triangulation is a weighted Delaunay triangulation in which the function determines the vertex heights.)

D'Azevedo and Simpson [42] show that a two-dimensional Delaunay triangulation minimizes the radius of the largest min-containment circle of its simplices, and Rajan [96] generalizes this result to Delaunay triangulations and min-containment spheres of any dimensionality. The *min-containment sphere* of a simplex is the smallest hypersphere that encloses the simplex.

Rajan's result and a theorem of Waldron [127] together imply a third optimality result, also related to multivariate piecewise linear interpolation. Suppose you must choose a triangulation to interpolate an unknown function, and you wish to minimize the largest pointwise error in the domain. After you choose the triangulation, an adversary will choose the worst possible smooth function for your triangulation to interpolate, subject to a fixed upper bound on the absolute curvature (i.e. second directional derivative) of the function anywhere in the domain. The Delaunay triangulation is your optimal choice.

To better understand these three optimality properties, consider multivariate piecewise linear interpolation on a triangulation $\mathcal{T}$ of a point set V. Let $\mathcal{T}^+ = \{\sigma^+ : \sigma \in \mathcal{T}\}$ be the triangulation lifted by the parabolic lifting map; $\mathcal{T}^+$ is a simplicial complex embedded in E^{d+1}. Think of $\mathcal{T}^+$ as inducing a continuous piecewise linear function $\mathcal{T}^+(p)$ that maps each point $p \in \text{conv}(V)$ to a real value.

How well does $\mathcal{T}^+$ approximate the paraboloid? Let $e(p) = \mathcal{T}^+(p) - |p|^2$ be the error in the interpolated function $\mathcal{T}^+$ as an approximation of the paraboloid $|p|^2$. At each vertex $v \in V$, $e(v) = 0$. Because $|p|^2$ is convex, the error satisfies $e(p) \geq 0$ for all $p \in \text{conv}(V)$.

Theorem 27. *At every point $p \in \text{conv}(V)$, every Delaunay triangulation $\mathcal{T}$ of V minimizes $\mathcal{T}^+(p)$, and therefore minimizes the interpolation error $e(p)$, among all triangulations of V.*

Proof. If $\mathcal{T}$ is Delaunay, then $\mathcal{T}^+$ is the set of faces of the underside of the convex hull $\text{conv}(V^+)$ of the lifted vertices (or a subdivision of those faces if some of them are not simplicial). No triangulation whose vertices are V^+ can pass through any point below $\text{conv}(V^+)$. ■

Corollary 28 (Melissaratos [80])**.** *Every Delaunay triangulation of V minimizes $\|e\|_{L_q}$ for every Lebesgue norm L_q, and every other norm monotonic in e.* ■

Theorem 29 (Rajan [96])**.** *Every Delaunay triangulation of V minimizes the largest min-containment sphere, compared with all other triangulations of V.* ■

I omit the proof because of its length.

The optimality of the Delaunay triangulation for controlling the largest min-containment radius dovetails nicely with an error bound for piecewise linear interpolation derived by Waldron [127]. Let C_c be the space of scalar functions defined over $\text{conv}(V)$ that have C^1 continuity and whose absolute curvature nowhere exceeds c. In other words, for every $f \in C_c$, every point $p \in \text{conv}(V)$, and every unit direction vector $\mathbf{d}$, the magnitude of the second directional derivative $f''_{\mathbf{d}}(p)$ is at most c. This is a common starting point for analyses of piecewise linear interpolation error.

Let f be a function in C_c. Let $\sigma \subseteq \text{conv}(V)$ be a simplex (of any dimensionality) with min-containment radius r_{mc}. Let h_σ be a linear function that interpolates f at the vertices of σ. Waldron shows that for all $p \in \sigma$, the absolute error $|e(p)| = |h_\sigma(p) - f(p)|$ is at most $cr_{\text{mc}}^2/2$. Furthermore, this bound is sharp: for every simplex σ with min-containment radius r_{mc}, there is a function $f \in C_c$ and a point $p \in \sigma$ such that $|e(p)| = cr_{\text{mc}}^2/2$. That function is $f(p) = c|p|^2/2$, and that point is $p = O_{\text{mc}}$.

Theorem 30. *Every Delaunay triangulation $\mathcal{T}$ of V minimizes*

$$\max_{f \in C_c} \max_{p \in \text{conv}(V)} |\mathcal{T}^+(p) - f(p)|,$$

the worst-case pointwise interpolation error, among all triangulations of V.

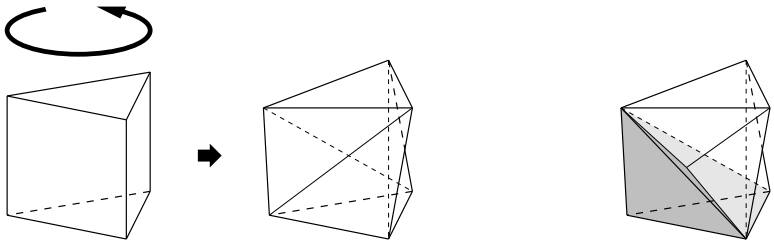


Figure 4.5: Schönhardt's untetrahedralizable polyhedron (center) is formed by rotating one end of a triangular prism (left), thereby creating three diagonal reflex edges. The convex hull of any four polyhedron vertices (right) sticks out.

Proof. For any triangulation $\mathcal{T}$, $\max_{f \in C_c} \max_{p \in \text{conv}(V)} |\mathcal{T}^+(p) - f(p)| = cr_{\max}^2/2$, where $r_{\max}$ is the largest min-containment radius among all the simplices in $\mathcal{T}$. The result follows immediately from Theorem 29. ■

One of the reasons why Delaunay triangulations are important is because, in the senses of Theorems 27 and 30, the Delaunay triangulation is an optimal piecewise linear interpolating surface. Of course, $e(p)$ is not the only criterion for the merit of a triangulation used for interpolation. Many applications need the interpolant to approximate the gradient—that is, not only must $\mathcal{T}^+(p)$ approximate $f(p)$, but $\nabla\mathcal{T}^+(p)$ must approximate $\nabla f(p)$ well too. For the goal of approximating $\nabla f(p)$ in three or more dimensions, the Delaunay trianguation is sometimes far from optimal even for simple functions like the paraboloid $f(p) = |p|^2$. This is why eliminating slivers is a crucial problem in Delaunay mesh generation.

4.5 Three-Dimensional Constrained Delaunay Triangulations

Constrained Delaunay triangulations generalize to three or more dimensions, but whereas every piecewise linear complex in the plane has a CDT, not every three-dimensional PLC has one. Worse yet, there exist simple polyhedra that do not have triangulations at all—that is, they cannot be subdivided into tetrahedra without creating new vertices (i.e. tetrahedron vertices that are not vertices of the polyhedron).

E. Schönhardt [106] furnishes an example depicted in Figure 4.5. The easiest way to envision this polyhedron is to begin with a triangular prism. Imagine grasping the prism so that its bottom triangular face cannot move, while twisting the top triangular face so it rotates slightly about its center while remaining horizontal. This rotation breaks each of the three square faces into two triangular faces along a diagonal *reflex edge*—an edge at which the polyhedron is locally nonconvex. 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 does not lie entirely within the polyhedron. Therefore, Schönhardt's polyhedron cannot be triangulated without additional vertices. It can be subdivided into tetrahedra with the addition of one vertex at its center.

Adding to the difficulty, Ruppert and Seidel [103] prove that it is NP-hard to determine whether a polyhedron has a triangulation, or whether it can be subdivided into tetrahedra with only k additional vertices for an arbitrary constant k.

The following sections discuss triangulations and CDTs of polyhedra and PLCs in three dimensions. It is possible to refine any polyhedron or PLC by adding new vertices on its edges so that it has a constrained Delaunay triangulation. This fact makes CDTs useful in three dimensions.

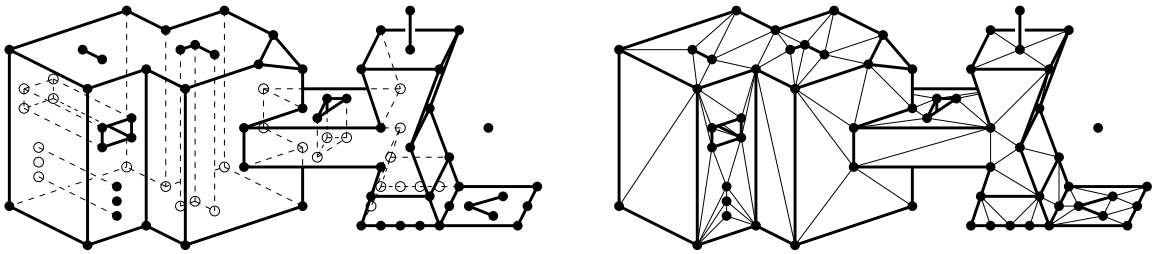


Figure 4.6: A three-dimensional piecewise linear complex and its constrained Delaunay triangulation. Each polygon and polyhedron may have holes, slits, and vertices in its relative interior. Each polyhedron may also have polygons in its interior.

4.5.1 Piecewise Linear Complexes and their Triangulations in E^d

The domain over which a general-dimensional CDT is defined is a general-dimensional piecewise linear complex, which is a set of linear cells—points, edges, polygons, and polyhedra—as illustrated in Figure 4.6. The linear cells constrain how the complex can be triangulated: each linear cell in the complex must be a union of simplices in the triangulation. The union of the linear cells specifies the region to be triangulated.

Definition 40 (piecewise linear complex; segment; wall; face)**.** *A* piecewise linear complex *(PLC)* $\mathcal{X}$ *is a finite set of linear cells that satisfies the following properties.*

- *The vertices and edges in* $\mathcal{X}$ *form a simplicial complex.*
- *For each linear cell* $f \in \mathcal{X}$*, the boundary of* f *is a union of linear cells in* $\mathcal{X}$*.*
- *If two distinct linear cells* $f, g \in \mathcal{X}$ *intersect, their intersection is a union of linear cells in* $\mathcal{X}$*, all having lower dimension than at least one of* f *or* g*. (See Figures 2.14 and 4.6.)*

As in the plane, $\mathcal{X}$*'s edges are called* segments *and its polygons are called* walls*. Its* underlying space *is* $|\mathcal{X}| = \bigcup_{f \in \mathcal{X}} f$*, which is usually the domain to be triangulated. The* faces *of a linear cell* $f \in \mathcal{X}$ *are the linear cells in* $\mathcal{X}$ *that are subsets of* f*, including* f *itself.*

The notion of a PLC was proposed by Miller, Talmor, Teng, Walkington, and Wang [82].[1] A triangulation of a PLC must cover every polyhedron, respect every polygon, and include every segment.

Definition 41 (triangulation of a PLC)**.** *Let* $\mathcal{X}$ *be a PLC. A* triangulation of $\mathcal{X}$ *is a simplicial complex* $\mathcal{T}$ *such that*

- $\mathcal{X}$ *and* $\mathcal{T}$ *have the same vertices,*
- *every linear cell in* $\mathcal{X}$ *is a union of simplices in* $\mathcal{T}$ *(which implies that every edge in* $\mathcal{X}$ *is in* $\mathcal{T}$*), and*
- $|\mathcal{T}| = |\mathcal{X}|$.

[1] Miller et al. call it a *piecewise linear system*, but their construction is so obviously a complex that a change in name seems obligatory. The present definition is different from Miller's, but nearly equivalent, with one true difference: Miller does not impose the first condition given here, but permits vertices to lie in the relative interior of an edge. Disallowing such vertices simplifies the presentation while entailing no essential loss of generality, because edges with vertices in their relative interiors can be subdivided into edges that obey the condition.

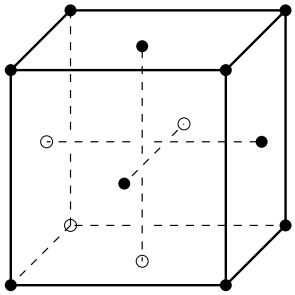


Figure 4.7: A convex PLC with no triangulation.

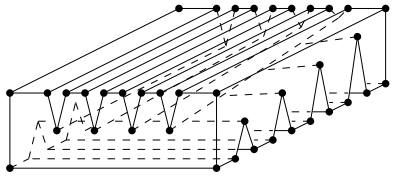


Figure 4.8: Chazelle's polyhedron.

Schönhardt's polyhedron shows that not every PLC has a triangulation. Every convex polyhedron has a triangulation; what about convex polyhedra with internal segments? Figure 4.7 illustrates a PLC with no triangulation, consisting of a cube inside which three orthogonal segments pass by each other but do not intersect. If any one of the segments is omitted, the PLC has a triangulation. This example shows that, unlike with planar triangulations, it is not always possible to insert a new edge into a tetrahedralization.

Because some polyhedra and PLCs do not have triangulations, Steiner triangulations (Definition 35) are even more important in three dimensions than in the plane. Chazelle [27] shows that every n-vertex polyhedron has a Steiner triangulation with at most $O(n^2)$ vertices, found by constructing a *vertical decomposition* of the polyhedron. The same is true for PLCs of complexity n. Unfortunately, there are polyhedra for which it is not possible to do better; Figure 4.8 depicts Chazelle's polyhedron [27], which has n vertices and $O(n)$ edges, but cannot be divided into fewer than $\Theta(n^2)$ convex bodies. Chazelle and Palios [28] show that the worst-case complexity of subdividing a polyhedron is related to its number of reflex edges: they give an algorithm that divides any polyhedron with r reflex edges into $O(n + r^2)$ tetrahedra, and they show that some polyhedra with r reflex edges cannot be divided into fewer than $\Omega(n + r^2)$ convex bodies.

It appears likely, though it is proven only in two dimensions, that there exist PLCs whose smallest Steiner Delaunay triangulations are asymptotically larger than their smallest Steiner triangulations. Algorithms by Murphy, Mount, and Gable [87], Cohen-Steiner, Colin de Verdière, and Yvinec [40], Cheng and Poon [33], and Rand and Walkington [97] can find a Steiner Delaunay tetrahedralization of any three-dimensional polyhedron, but they might introduce a superpolynomial number of new vertices. No known algorithm for finding Steiner Delaunay tetrahedralizations is guaranteed to introduce only a polynomial number of new vertices, and no algorithm of any complexity has been offered for four- or higher-dimensional Steiner Delaunay triangulations. Moreover, the existing algorithms all seem to introduce an unnecessarily large number of vertices near small domain angles. These problems can be partly remediated by Steiner CDTs.

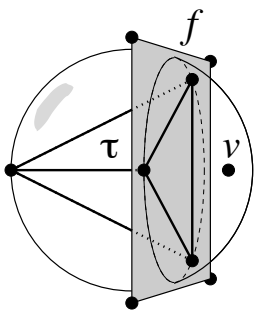


Figure 4.9: A constrained Delaunay tetrahedron τ.

4.5.2 The Constrained Delaunay Triangulation in E^3

Three-dimensional constrained Delaunay triangulations aspire to retain most of the advantages of Delaunay triangulations while respecting constraints. But Figures 4.5, 4.7, and 4.8 demonstrate that some PLCs, even some polyhedra, have no triangulation at all. Moreover, some polyhedra that do have triangulations do not have CDTs. Nevertheless, CDTs are useful because, if we are willing to add new vertices, a Steiner CDT might require many fewer vertices than a Steiner Delaunay triangulation.

As in the plane, a Constrained Delaunay Lemma states that there are several equivalent definitions of "constrained Delaunay triangulation." The simplest is that a CDT is a triangulation of a PLC in which every facet not included in a wall is locally Delaunay. A CDT differs from a Delaunay triangulation in three ways: it is not necessarily convex, it is required to respect a PLC, and the facets of the CDT that are included in walls are exempt from being locally Delaunay.

The primary definition of CDT specifies that every tetrahedron is constrained Delaunay, defined as follows.

Definition 42 (constrained Delaunay)**.** *In the context of a PLC $\mathcal{X}$, a simplex σ is* constrained Delaunay *if it satisfies the following three conditions.*

- *$\mathcal{X}$ contains σ's vertices.*
- *σ respects $\mathcal{X}$. (Recall Definition 34.)*
- *There is a circumsphere of σ that encloses no vertex in $\mathcal{X}$ that is visible from any point in the relative interior of σ.*

Two points are visible *to each other (equivalently, can* see *each other) if $|\mathcal{X}|$ includes the open line segment connecting the two points, but no linear cell in $\mathcal{X}$ intersects only part of that open line segment. A linear cell that intersects the open line segment but does not entirely include it is said to* occlude *the visibility between the two points.*

Figure 4.9 depicts a constrained Delaunay tetrahedron τ. The faces of τ whose relative interiors intersect the wall f are included in f, so τ respects $\mathcal{X}$. The circumsphere of τ encloses one vertex v, but v is not visible from any point in the interior of τ, because f occludes its visibility.

Definition 43 (constrained Delaunay triangulation)**.** *A* constrained Delaunay triangulation *(CDT) of a PLC $\mathcal{X}$ is a triangulation of $\mathcal{X}$ in which every tetrahedron is constrained Delaunay, and every triangle that is not a face of a tetrahedron is constrained Delaunay.*

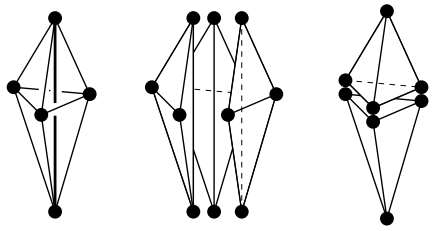


Figure 4.10: Left: a PLC with no CDT. Center: the sole tetrahedralization of this PLC. Its three tetrahedra are not constrained Delaunay. Right: the two Delaunay tetrahedra do not respect the central segment.

Figure 4.6 illustrates a PLC and its CDT. Observe that the PLC has a polygon that is not a face of any polyhedron; this face is triangulated with constrained Delaunay triangles.

Figure 4.10 illustrates a PLC that has no CDT because of a segment that runs vertically through the domain interior. There is only one tetrahedralization of this PLC—composed of three tetrahedra encircling the central segment—and its tetrahedra are not constrained Delaunay, because each of them has a visible vertex inside its circumsphere. Whereas walls usually block enough visibility to ensure their presence in a CDT, segments usually do not. But segments can dictate that a CDT does not exist at all. If the central segment in Figure 4.10 is removed, the PLC has a CDT made up of two tetrahedra.

A *Steiner CDT* or *conforming CDT* of $\mathcal{X}$ is a Steiner triangulation of $\mathcal{X}$ in which every tetrahedron is constrained Delaunay, and every triangle that is not a face of a tetrahedron is constrained Delaunay. Algorithms for constructing Steiner CDTs (e.g. mesh generation algorithms) must sometimes place new vertices on segments to force the triangulation to respect them.

4.5.3 The CDT Theorem

Although not all piecewise linear complexes have constrained Delaunay triangulations, there is an easily tested, sufficient (but not necessary) condition that guarantees that a CDT exists. A three-dimensional PLC $\mathcal{X}$ is *edge-protected* if every edge in $\mathcal{X}$ is strongly Delaunay.

Theorem 31 (CDT Theorem [118])**.** *Every edge-protected PLC has a CDT.* □

It is not sufficient for every edge in $\mathcal{X}$ to be Delaunay. If all six vertices of Schönhardt's polyhedron lie on a common sphere, then all of its edges (and all its faces) are Delaunay, but it still has no 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.

What if a PLC that you wish to triangulate is not edge-protected? You can make it edge-protected by adding vertices on its segments—a task that any Delaunay mesh generation algorithm must do anyway. The augmented PLC has a CDT, which is a Steiner CDT of the original PLC.

Figure 4.11 illustrates the difference between using a Delaunay triangulation and using a CDT for mesh generation. With a Delaunay triangulation, the mesh generator must insert new vertices that guarantee that every segment is a union of Delaunay (preferably strongly Delaunay) edges, and every wall is a union of Delaunay (preferably strongly Delaunay) triangles. With a CDT, new vertices must be inserted that guarantee that every segment is a union of strongly Delaunay edges; but then the augmented PLC is edge-protected, and the CDT Theorem guarantees that the walls can be recovered without inserting any additional vertices. The advantage of a CDT is that many fewer vertices might be required.

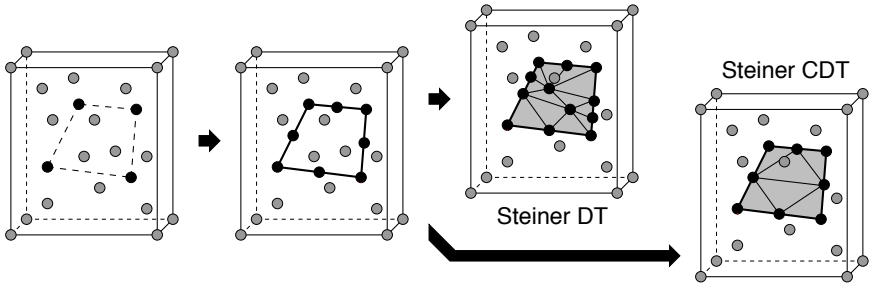


Figure 4.11: Comparison of Steiner Delaunay triangulations and Steiner CDTs. For clarity, vertices inside each box are shown, but tetrahedra are not. For both types of triangulation, missing segments are recovered by inserting new vertices until each segment is a union of strongly Delaunay edges. In a Steiner Delaunay triangulation, additional vertices are inserted until each wall is a union of strongly Delaunay triangles. In a Steiner CDT, no additional vertices need be inserted; the walls are recovered by computing a CDT.

Testing whether a PLC $\mathcal{X}$ is edge-protected is straightforward. Form the Delaunay triangulation of the vertices in $\mathcal{X}$. If a segment $s \in \mathcal{X}$ is missing from the triangulation, then s is not strongly Delaunay, and $\mathcal{X}$ is not edge-protected. If s is present, it is Delaunay. Testing whether a Delaunay segment s is strongly Delaunay is equivalent to determining whether the Voronoi polygon dual to s is nondegenerate.

4.5.4 Properties of the Constrained Delaunay Triangulation in E^3

This section summarizes the properties of three-dimensional CDTs. Proofs of the claims in this section may be found elsewhere [118].

The Delaunay Lemma for three-dimensional CDTs provides an alternative definition of CDT: a triangulation of a PLC $\mathcal{X}$ is a CDT if and only if every one of its facets is locally Delaunay *or* is included in a wall in $\mathcal{X}$.

Theorem 32 (Constrained Delaunay Lemma)**.** *Let $\mathcal{X}$ be a PLC in which every linear cell is a face of some polyhedron in $\mathcal{X}$, so there are no dangling polygons. Let $\mathcal{T}$ be a triangulation of $\mathcal{X}$. The following three statements are equivalent.*

A. Every tetrahedron in $\mathcal{T}$ is constrained Delaunay (i.e. $\mathcal{T}$ is constrained Delaunay).

B. Every facet in $\mathcal{T}$ not included in a wall in $\mathcal{X}$ is constrained Delaunay.

C. Every facet in $\mathcal{T}$ not included in a wall in $\mathcal{X}$ is locally Delaunay. ■

A constrained Delaunay triangulation $\mathcal{T}$ of $\mathcal{X}$ induces a two-dimensional triangulation of each wall $f \in \mathcal{X}$, namely $\mathcal{T}|_f = \{\sigma \in \mathcal{T} : \sigma \subseteq f\}$. Statement B above implies that the triangles in $\mathcal{T}|_f$ need not be constrained Delaunay with respect to $\mathcal{X}$—but they *are* constrained Delaunay with respect to the wall f, in the following sense.

Theorem 33. *Let $\mathcal{T}$ be a CDT of a three-dimensional PLC $\mathcal{X}$. Let $f \in \mathcal{X}$ be a wall. Let $\mathcal{T}|_f$ be the set of simplices in $\mathcal{T}$ that are included in f. Let $\mathcal{X}|_f$ be the set of faces of f (including f itself); $\mathcal{X}|_f$ is a two-dimensional PLC embedded in three-dimensional space. Then $\mathcal{T}|_f$ is a CDT of $\mathcal{X}|_f$.* ■

A PLC is *generic* if its vertices are generic. A generic PLC has a unique CDT, if it has one at all.

Theorem 34. *A generic piecewise linear complex has at most one constrained Delaunay triangulation.* ■

A consequence of Theorems 33 and 34 is that, if a PLC is generic, a CDT construction algorithm can begin by computing the two-dimensional CDTs of the walls, then use them to help compute the three-dimensional CDT of the PLC, secure in the knowledge that the wall triangulations will match the volume triangulation.

CDTs inherit the optimality properties of Delaunay triangulations described in Section 4.4, albeit with respect to a smaller set of triangulations, namely the triangulations of a PLC. However, if a PLC has no CDT, finding the optimal triangulation is an open problem.

Theorem 35. *If a PLC X has a CDT, then every CDT of X minimizes the largest min-containment sphere, compared with all other triangulations of X. Every CDT of X also optimizes the criteria discussed in Theorems 27, 28, and 30.* ■

Chapter 5

Algorithms for Constructing Delaunay Triangulations in E^3

The most popular algorithms for constructing Delaunay tetrahedralizations are incremental insertion and gift-wrapping algorithms, both of which generalize to three or more dimensions with little difficulty. This chapter reprises those algorithms, with attention to the aspects that are different in three dimensions. In particular, the analysis of the running time of point location with conflict lists is more complicated in three dimensions than in the plane. I use this gap as an opportunity to introduce a more sophisticated vertex ordering and its analysis. Instead of fully randomizing the order in which vertices are inserted, I recommend using a *biased randomized insertion order* that employs just enough randomness to ensure that the expected running time is the worst-case optimal $O(n^2)$—or better yet, $O(n \log n)$ time for the classes of point sets most commonly triangulated in practice—while maintaining enough spatial locality that implementations of the algorithm use the memory hierarchy more efficiently. This vertex ordering, combined with a simpler point location method, yields the fastest three-dimensional Delaunay triangulators in practice.

CDTs have received much less study in three dimensions than in two. There are two classes of algorithm available: gift-wrapping and incremental wall insertion. Gift-wrapping is easier to implement; it is not much different in three dimensions than in two. It runs in $O(nh)$ time for Delaunay triangulations and $O(nmh)$ time for CDTs, where n is the number of vertices, m is the total complexity of the PLC's polygons, and h is the number of tetrahedra produced. There is a variant of the gift-wrapping algorithm that, by constructing the tetrahedra in a disciplined order and using other tricks to avoid visibility computations [115], runs in $O(nh)$ worst-case time, but I omit it here.

Perhaps the fastest three-dimensional CDT construction algorithm in practice is similar to the one I advocate in two dimensions. First, construct a Delaunay triangulation of the PLC's vertices, then insert its walls one by one with a flip algorithm [117]. This algorithm constructs a CDT in $O(n^2 \log n)$ time, though there are reasons to believe it will run in $O(n \log n)$ time on most PLCs in practice. Be forewarned, however, that this algorithm only works on edge-protected PLCs! This is rarely a fatal restriction, because a mesh generation algorithm that uses CDTs should probably insert vertices on the PLC's edges to make it edge-protected and ensure that it has a CDT. Some PLCs have CDTs despite not being edge-protected; if they are generic, their CDTs can be constructed by gift-wrapping.

Procedure	Purpose
AddTetrahedron(u, v, w, x)	Add a positively oriented tetrahedron $uvwx$
DeleteTetrahedron(u, v, w, x)	Delete a positively oriented tetrahedron $uvwx$
Adjacent(u, v, w)	Return a vertex x such that $uvwx$ is a positively oriented tetrahedron
AdjacentOne(u)	Return vertices v, w, x such that $uvwx$ is a positively oriented tetrahedron

Figure 5.1: An interface for a tetrahedralization data structure.

5.1 A Dictionary Data Structure for Tetrahedralizations

Figure 5.1 summarizes an interface for storing a tetrahedral complex, analogous to the interface for planar triangulations in Section 3.2. Two procedures, AddTetrahedron and DeleteTetrahedron, specify a tetrahedron to be added or deleted by listing its vertices with a positive orientation, as described in Section 3.1. The procedure Adjacent recovers the tetrahedron adjoining a specified oriented triangular face, or returns $\emptyset$ if there is no such tetrahedron. The vertices of a tetrahedron may include the ghost vertex. The data structure enforces the invariant that only two tetrahedra may adjoin a triangular face, and only one on each side of the face.

The simplest fast implementation echoes the implementation described in Section 3.2. Store each tetrahedron $\triangle uvwx$ four times in a hash table, keyed on the oriented faces $\triangle uvw$, $\triangle uxv$, $\triangle uwx$, and $\triangle vxw$. To support AdjacentOne queries, an array stores, for each vertex u, a triangle $\triangle uvw$ such that the most recently added tetrahedron adjoining u has $\triangle uvw$ for a face.

The interface and data structure extend easily to permit the storage of triangles or edges that are not part of any tetrahedron, but it does not support fast adjacency queries on edges. For a substantially more space-efficient representation, see Blandford, Blelloch, Clemens, and Kadow [16].

5.2 Delaunay Vertex Insertion in E^3

The Bowyer–Watson algorithm extends in a straightforward way to three (or more) dimensions. Recall that the algorithm inserts a vertex v into a Delaunay triangulation in four steps. First, find one tetrahedron whose circumsphere encloses v (point location). Second, a depth-first search in the triangulation finds all the other tetrahedra whose circumspheres enclose v, in time proportional to their number. Third, delete these tetrahedra, as illustrated in Figure 5.2. The union of the deleted tetrahedra is a star-shaped polyhedral cavity. Fourth, for each triangular face of the cavity, create a new tetrahedron joining it with v, as illustrated. Figure 5.3 gives pseudocode that interleaves the second, third, and fourth steps.

To support inserting vertices that lie outside the triangulation, each triangular face on the boundary of the triangulation adjoins a *ghost tetrahedron* analogous to the ghost triangles of Section 3.4, having three real vertices and a ghost vertex g. Let $\triangle uvw$ be a boundary triangle, oriented so the triangulation is behind it. The incremental insertion algorithm stores a positively oriented tetrahedron $uvwg$ in the triangulation data structure.

There are two cases in which $uvwg$ must be deleted, i.e. $\triangle uvw$ is no longer a boundary triangle: if a vertex is inserted in the open halfspace in front of $\triangle uvw$, or if a newly inserted vertex lies in the open circumdisk of $\triangle uvw$ (i.e. it is coplanar with $\triangle uvw$ and enclosed by its circumcircle). Call the union of these two regions the *outer halfspace* of $\triangle uvw$. It is the set of points enclosed by the circumsphere of $uvwg$ in the limit as g moves away from the triangulation.

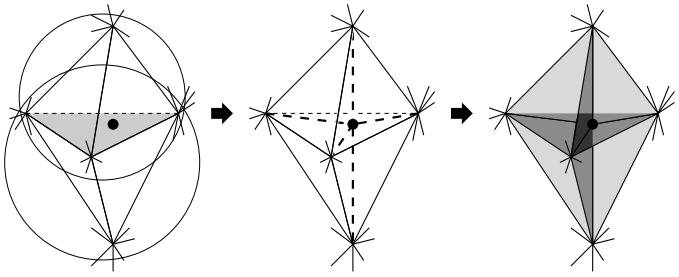


Figure 5.2: The Bowyer–Watson algorithm in three dimensions. A new vertex falls inside the circumspheres of the two tetrahedra illustrated at left. These tetrahedra may be surrounded by other tetrahedra, which for clarity are not shown. The two tetrahedra and the face they share (shaded) are deleted. At center, the five new Delaunay edges. At right, the nine new Delaunay triangles—one for each edge of the cavity. Six new tetrahedra are created—one for each face of the cavity.

```
InsertVertex3D(u, v, w, x, y)
{ u is the vertex to insert. vwxy is a positively oriented tetrahedron whose circumsphere encloses u. }
1     DeleteTetrahedron(v, w, x, y)
2     ConsiderTetrahedron(u, x, w, v)
3     ConsiderTetrahedron(u, y, v, w)
4     ConsiderTetrahedron(u, v, y, x)
5     ConsiderTetrahedron(u, w, x, y)

ConsiderTetrahedron(u, v, w, x)
{ u is a new vertex. Is the oriented tetrahedron uvwx Delaunay? }
6     y ← Adjacent(v, w, x)      { Find tetrahedron vwxy opposite the facet vwx from u }
7     if y ≠ ∅      { Do nothing if the tetrahedron has already been deleted }
8         if InSphere(u, v, w, x, y) > 0
9             DeleteTetrahedron(v, w, x, y)      { Tetrahedra uvwx and vwxy are not Delaunay }
10            ConsiderTetrahedron(u, v, w, y)
11            ConsiderTetrahedron(u, w, x, y)
12            ConsiderTetrahedron(u, x, v, y)
13        else AddTetrahedron(u, v, w, x)      { vwx is a facet of the cavity and uvwx is Delaunay }
```

Figure 5.3: Algorithm for inserting a vertex u into a Delaunay triangulation, given a tetrahedron $vwxy$ whose circumsphere encloses u. To adapt the code for a weighted Delaunay triangulation, replace the InSphere test in Line 8 with Orient4D(u^+, v^+, w^+, x^+, y^+), and choose an input tetrahedron $vwxy$ whose witness hyperplane is above u^+.

5.3 The Running Time of Vertex Insertion in E^3

How expensive is vertex insertion, leaving out the cost of point location? The insertion of a single vertex into an n-vertex Delaunay triangulation can delete $\Theta(n^2)$ tetrahedra if the triangulation is the one depicted in Figure 4.2. However, a single vertex insertion can only create $\Theta(n)$ tetrahedra: observe that the boundary of the cavity is a planar graph, so the cavity has fewer than $2n$ boundary triangles.

It follows that during a sequence of n vertex insertion operations, at most $\Theta(n^2)$ tetrahedra are created. A tetrahedron can only be deleted if it is first created, so at most $\Theta(n^2)$ tetrahedra are deleted, albeit possibly most of them in a single vertex insertion.

Randomizing the vertex insertion order does not improve these numbers. For d-dimensional Delaunay triangulations, the worst-case total number of simplices created and deleted by the incremental insertion algorithm is in $\Theta(n^{\lfloor d/2 \rfloor + 1})$, and the expected total number of simplices created and deleted by *random* incremental insertion is in $\Theta(n^{\lceil d/2 \rceil})$. In the worst case, randomization makes an asymptotic difference only when d is even.

However, a special case that occurs frequently in practice—by all accounts it seems to be the norm—is the circumstance where the Delaunay triangulation has complexity linear, rather than quadratic, in the number of vertices, and moreover the intermediate triangulations produced during incremental insertion have expected linear complexity. For point sets with this property, a random insertion order guarantees that each vertex insertion will create and delete an expected constant number of tetrahedra, just as it does in the plane, and we shall see that the random incremental insertion algorithm runs in expected $O(n \log n)$ time. This running time is often observed in practice, even in higher dimensions. Be forewarned, however, that there are point sets for which the final triangulation has linear complexity but the intermediate triangulations have expected quadratic complexity, thereby slowing down the algorithm dramatically.

Moreover, even for worst-case point sets, randomization helps to support fast point location. Recall that the last three steps of the Bowyer–Watson algorithm run in time proportional to the number of tetrahedra they delete and create, so the running time of the three-dimensional incremental insertion algorithm, *excluding* point location, is in $O(n^2)$. With conflict lists and a random insertion order, point location is no more expensive than this, so the random incremental insertion algorithm achieves an optimal expected running time of $O(n^2)$.

5.4 Biased Randomized Insertion Orders

The advantage of inserting vertices in random order is that it guarantees that the expected running time of point location is optimal, and that pathologically slow circumstances like those illustrated in Figure 3.9 are unlikely to happen. But there is a serious disadvantage: random vertex insertions tend to interact poorly with the memory hierarchy in modern computers, especially virtual memory. Ideally, data structures representing tetrahedra and vertices that are close together geometrically should be close together in memory—a property called *spatial locality*—for better cache and virtual memory performance.

Amenta, Choi, and Rote [2] show that the permutation of vertices does not need to be uniformly random for the running time to be optimal. A *biased randomized insertion order* (BRIO) is a permutation of the vertices that has strong spatial locality but retains enough randomness to obtain an expected running time in $O(n^2)$. Their experiments show that a BRIO greatly improves the efficiency of the memory hierarchy—especially virtual memory.

Their experiments also show that incremental insertion achieves superior running times in practice when it uses a BRIO but replaces the conflict list with a point location method that simply walks from the previously inserted vertex toward the next inserted vertex; see Section 5.5. Although walking point location does not offer as strong a theoretical guarantee on running time as a conflict list, this incremental insertion algorithm is perhaps the most attractive in practice, as it combines excellent speed with a simple implementation.

Let n be the number of vertices to triangulate. A BRIO orders the vertices in a sequence of *rounds* numbered zero through $\lceil \log_2 n \rceil$. Each vertex is assigned to the final round, round $\lceil \log_2 n \rceil$, with probability $1/2$. The remaining vertices are assigned to the second-last round with probability $1/2$, and so on. Each

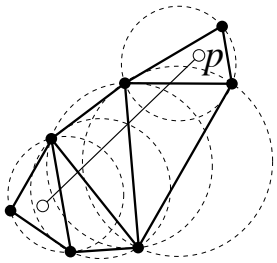


Figure 5.4: Walking to the triangle that contains p.

vertex is assign to round zero with probability $(1/2)^{\lceil \log_2 n \rceil} \leq 1/n$. The incremental insertion algorithm begins by inserting the vertices in round zero, then round one, and so on to round $\lceil \log_2 n \rceil$.

Within any single round, the vertices can be arranged in any order without threatening the worst-case expected running time of the algorithm. Hence, we order the vertices within each round to create as much spatial locality as possible. One way to do this is to insert the vertices in the order they are encountered on a space-filling curve such as a Hilbert curve or a z-order curve. Another way, which Amenta et al. tested, is to store the vertices in an octree or k-d tree, refined so each leaf node contains only a few vertices; then order the vertices by a traversal of the tree. (Octree traversal is one way to sort vertices along a Hilbert or z-order curve.)

The tendency of vertices that are geometrically close together to be close together in the ordering does not necessarily guarantee that the data structures associated with them will be close together in memory. Amenta et al. addressed this question experimentally by implementing BRIOs in three different Delaunay triangulation programs written by three different people, and showing that all three run faster with a BRIO than with a vertex permutation chosen uniformly at random, especially when the programs run out of main memory and have to resort to virtual memory.

Whether you use the traditional random incremental insertion algorithm or a BRIO, you face the problem of bootstrapping the algorithm. The most practical approach is to choose four affinely independent vertices, construct their Delaunay triangulation (a single tetrahedron), create four adjoining ghost tetrahedra, construct a conflict list, and insert the remaining vertices in a random order (a uniformly chosen permutation or a BRIO). Even if the four bootstrap vertices are not chosen randomly, it is possible to prove that the expected asymptotic running time of the algorithm is not compromised.

5.5 Point Location by Walking

By experimentation, Amenta, Choi, and Rote [2] demonstrate that, in conjuction with a BRIO, a simple point location method called *walking* appears to outperform conflict lists in practice, although there is no guarantee of a fast running time. A walking point location algorithm simply traces a straight line through the triangulation, visiting tetrahedra that intersect the line as illustrated in Figure 5.4, until it arrives at a tetrahedron that contains the new vertex [60]. In conjunction with a vertex permutation chosen uniformly at random (rather than a BRIO), walking point location visits many tetrahedra and is very slow. But walking is fast in practice if it follows two guidelines: the vertices should be inserted in an order that has much spatial locality, such as a BRIO, and each walk should begin at the most recently created tetrahedron. Then the typical walk visits a small constant number of tetrahdra.

To avoid a long walk between rounds of a BRIO, the vertex order (e.g. the tree traversal or the direction

of the space-filling curve) should be reversed on even-numbered rounds, so each round begins near where the previous round ends.

Amenta et al. observe that the three-dimensional incremental insertion algorithm with a BRIO and walking point location appears to run in linear time, not counting the initial $O(n \log n)$-time computation of a BRIO. For the point sets in their experiments, this observation holds whether they use a BRIO or a spatial ordering (generated by traversing an octree) that has no randomness at all. Randomness is often unnecessary in practice—frequently, simply sorting the vertices along a space-filling curve will yield excellent speed—but because points sets like that illustrated in Figure 3.9 are common in practice, I recommend choosing a BRIO to prevent the possibility of a pathologically slow running time.

5.6 The Gift-Wrapping Algorithm in E^3

The gift-wrapping algorithm described in Section 3.8 requires few new ideas to work in three (or more) dimensions. The algorithm constructs tetrahedra one at a time, and maintains a dictionary of unfinished facets. The pseudocode for Finish and GiftWrapCDT can be reused, with triangles replaced by tetrahedra, oriented edges replaced by oriented facets, and circumcircles replaced by circumspheres.

The biggest change is that triangles, not segments, seed the algorithm. But the walls in a PLC are polygons, and are not always triangles. Recall from Theorem 33 that a CDT of a PLC $\mathcal{X}$ induces a two-dimensional CDT of each wall in $\mathcal{X}$. To seed the three-dimensional gift-wrapping algorithm, one can first compute the two-dimensional CDT of each wall, then use the triangles in these CDTs to seed the three-dimensional algorithm.

To gift-wrap a Delaunay triangulation, seed the algorithm with one strongly Delaunay triangle. One way to find one is to choose an arbitrary input point and its nearest neighbor. For the third vertex of the triangle, choose the input point that minimizes the radius of the circle through the three vertices. If the set of input points is generic, the triangle having these three vertices is strongly Delaunay.

If the input (PLC or point set) is not generic, gift-wrapping is in even greater danger in three dimensions than in the plane. Whereas the planar gift-wrapping algorithm can handle subsets of four or more cocircular points by identifying them and giving them special treatment, no such approach works reliably in three dimensions. Imagine a point set that includes six points lying on a common empty sphere. Suppose that gift-wrapping inadvertently tetrahedralizes the space around these points so they are the vertices of a hollow cavity shaped like Schönhardt's polyhedron (from Section 4.5). The algorithm will be unable to fill the cavity. By far the most practical solution is to symbolically perturb the points so that they are generic. The same perturbation should also be used to compute the two-dimensional CDTs of the PLC's walls.

Another difficulty is that the input PLC might not have a CDT, in which case gift-wrapping will fail in one of two ways. One possibility is that the algorithm will fail to finish an unfinished facet, even though there is a vertex in front of that facet, because no vertex in front of that facet is visible from the facet's interior. This failure is easy to detect. The second possibility is that the algorithm will finish a facet by constructing a tetrahedron that is not constrained Delaunay, either because the tetrahedron's circumsphere encloses a visible vertex, or because the tetrahedron intersects the preexisting simplices wrongly (not in a complex). An attempt to gift-wrap Schönhardt's polyhedron brings about the last fate. The algorithm becomes substantially slower if it tries to detect these failures. Perhaps a better solution is to run the algorithm only on PLCs that are edge-protected or otherwise known to have CDTs.

A strange property of the CDT is that it is NP-hard to determine whether a PLC has a CDT, if the PLC is

not generic [59]. However, a polynomial-time algorithm is available for generic PLCs: run the gift-wrapping algorithm, and check whether it succeeded.

Gift-wrapping takes $O(nh)$ time for a Delaunay triangulation, or $O(nmh)$ time for a CDT, where n is the number of input points, m is the total complexity of the input walls, and h is the number of tetrahedra in the CDT; h is usually linear in n, but could be quadratic in the worst case.

5.7 Inserting a Vertex into a Constrained Delaunay Triangulation in E^3

Section 3.6 describes how to adapt the Bowyer–Watson vertex insertion algorithm to CDTs in the plane. The same adaptions work for three-dimensional CDTs, but there is a catch: even if a PLC $\mathcal{X}$ has a CDT, an augmented PLC $\mathcal{X} \cup \{v\}$ might not have one. This circumstance can be diagnosed after the depth-first search step of the Bowyer–Watson algorithm in one of two ways: by the fact that the cavity is not star-shaped, thus one of the newly created tetrahedra has nonpositive orientation, or by the fact that a segment or polygon runs through the interior of the cavity. An implementation can check explicitly for these circumstances, and signal that the vertex v cannot be inserted.

Chapter 6

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 for their construction 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 Section 6.1 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 [101] 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.

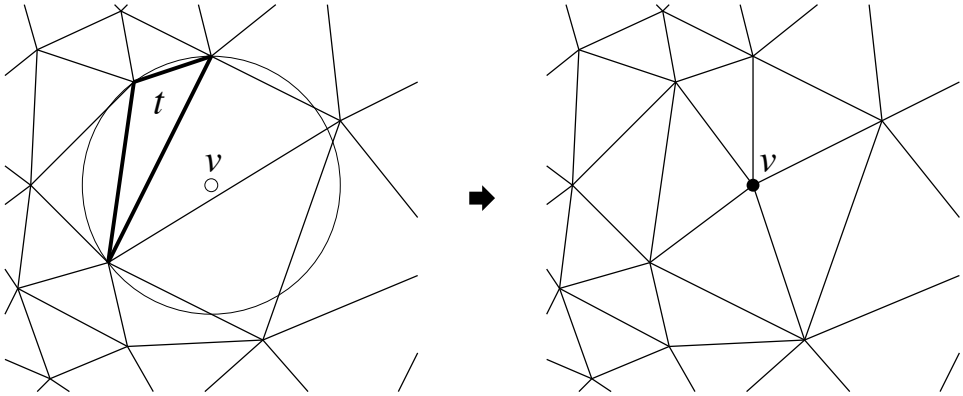


Figure 6.1: Any triangle whose circumradius-to-shortest edge ratio is larger than some bound B is split by inserting a vertex at its circumcenter. The Delaunay property is maintained, and the triangle is thus eliminated. Every new edge has length at least B times that of shortest edge of the poor triangle.

6.1 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 an element of poor quality. The Delaunay property is maintained, using 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 [56]. 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 6.1). 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 [102] employs a bound of $B = \sqrt{2}$, and Chew's second Delaunay refinement algorithm [37] employs a bound of $B = 1$. Chew's first Delaunay refinement algorithm [35] 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. When it does, all angles are bounded between 20.7° and 138.6° for Ruppert's algorithm, and between 30° and 120° for Chew's.

Henceforth, a triangle whose circumradius-to-shortest edge ratio is greater than B is said to be *skinny*. Figure 6.2 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

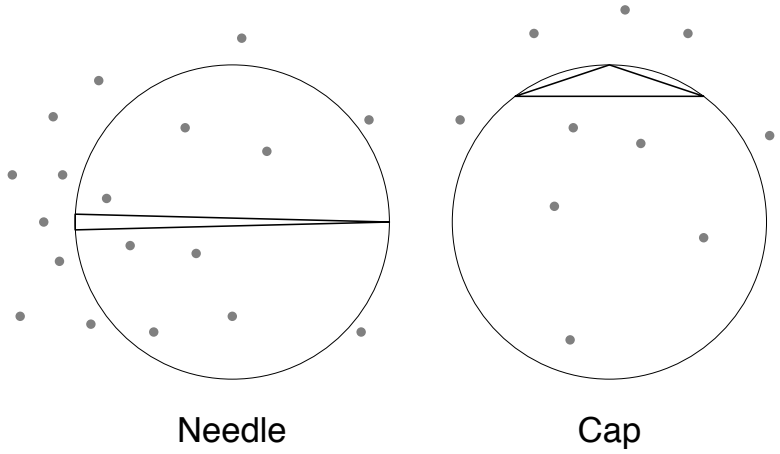


Figure 6.2: Skinny triangles have circumcircles larger than their shortest edges. Each skinny triangle may be classified as a needle, whose longest edge is much longer than its shortest edge, or a cap, which has an angle close to 180°. (The classifications are not mutually exclusive.)

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 6.3.4.

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 periodic space that "wraps around" at the boundaries. Regardless of the dimensionality, Delaunay refinement can eliminate all simplices having a circumradius-to-shortest edge ratio greater than one, without creating any edge shorter than the shortest 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 described 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 algorithms 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.

6.2 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 [35]. The second, which can produce graded meshes [37], is discussed in Section 6.4.

Given a constrained Delaunay 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 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.

Of course, the boundaries of the mesh may prevent some skinny triangles from being eliminated. Figure 6.3 illustrates an example in which there is a poor-quality triangle, but no vertex can be placed inside its

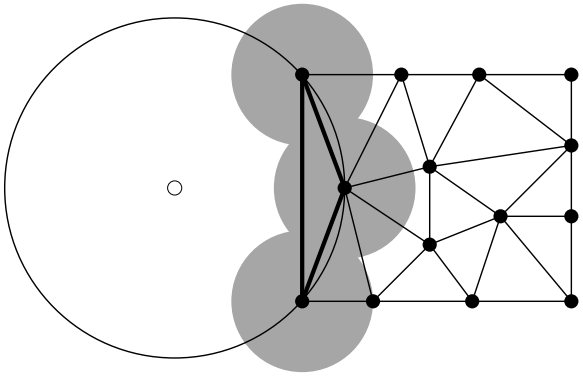


Figure 6.3: The bold triangle could be eliminated by inserting a vertex in its circumcircle. However, a vertex cannot be placed outside the triangulation domain, and it is forbidden to place a vertex within a distance of $h_{\min}$ from any other vertex. The forbidden region includes the shaded disks, which entirely cover the bold triangle.

circumcircle without creating an edge smaller than $h_{\min}$, which would compromise the termination guarantee. Chew circumvents 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. The input PSLG is not allowed to have two adjoining segments separated by an angle less than 30°.

For some parameter h chosen by the user, all segments are divided into subsegments whose lengths are in the range $[h, \sqrt{3}h]$. The parameter h must be chosen small enough that such a partition exists. New vertices are placed at the division points. 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 while maintaining the invariant that the triangulation is constrained Delaunay. 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 6.4.

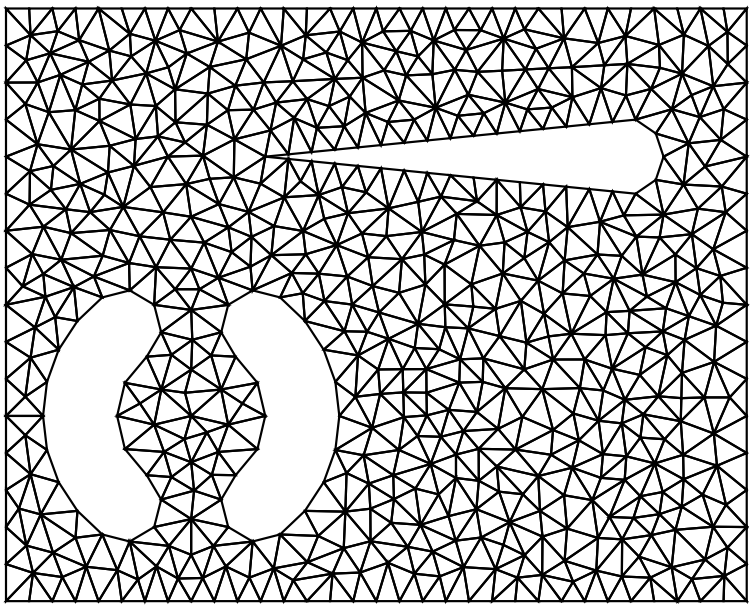


Figure 6.4: A mesh generated by Chew's first Delaunay refinement algorithm. (Courtesy Paul Chew).

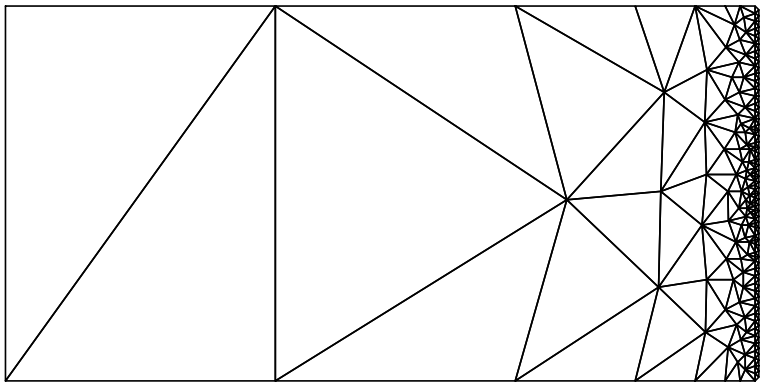


Figure 6.5: A demonstration of the ability of Delaunay refinement to achieve large gradations in triangle size while constraining angles. No angle is smaller than 24°.

6.3 Ruppert's Delaunay Refinement Algorithm

Jim Ruppert's algorithm for two-dimensional quality mesh generation [102] is perhaps the first theoretically guaranteed meshing algorithm to be truly satisfactory in practice. It extends Chew's first Delaunay refinement algorithm by allowing the density of triangles to vary quickly over short distances, as illustrated in Figure 6.5. The number of triangles produced is typically smaller than the number produced either by Chew's algorithm or the Bern–Eppstein–Gilbert quadtree algorithm [11], as Figure 6.6 shows.

I have mentioned that Chew independently developed a second Delaunay refinement algorithm quite similar to Ruppert's [37]. I present Ruppert's algorithm first in part because Ruppert's earliest publications of his results [100, 101] slightly predate Chew's, and mainly because the algorithm is accompanied by a proof that it produces 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)

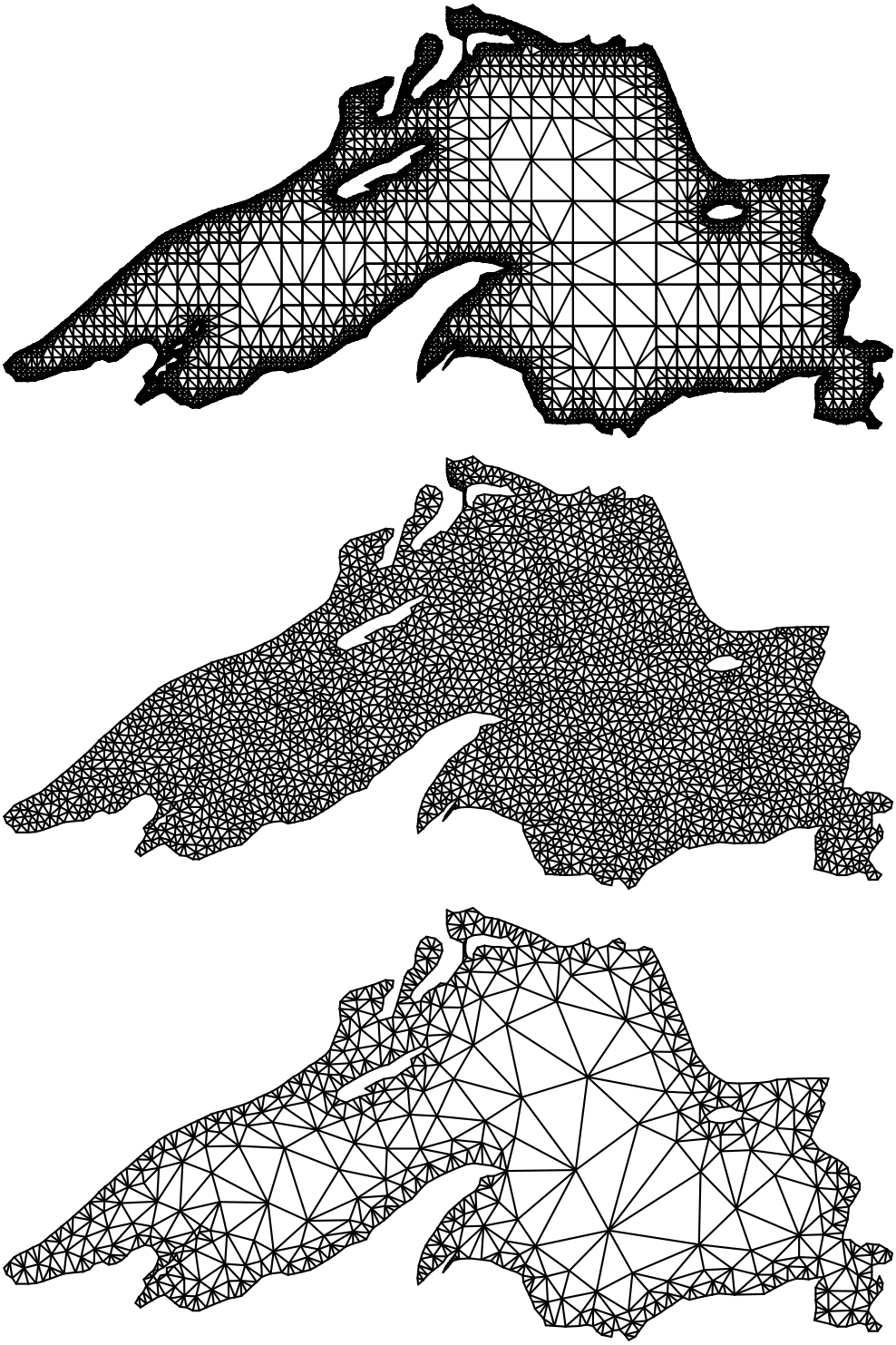


Figure 6.6: Meshes generated by the Bern–Eppstein–Gilbert quadtree-based algorithm (top), Chew's first Delaunay refinement algorithm (center), and Ruppert's Delaunay refinement algorithm (bottom). For this polygon, Chew's second Delaunay refinement algorithm produces nearly the same mesh as Ruppert's. (The first mesh was produced by the program `tripoint`, courtesy Scott Mitchell.)

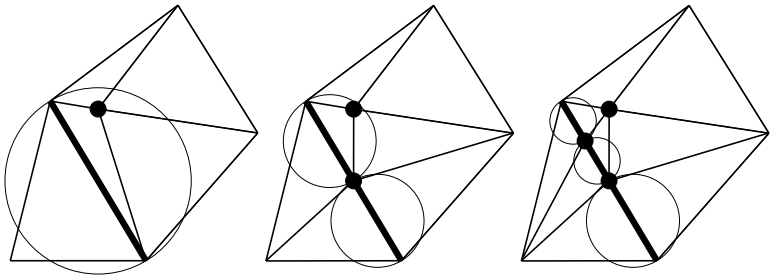


Figure 6.7: Segments are split recursively (while maintaining the constrained Delaunay property) until no subsegment is encroached.

is at most a constant factor 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.) Sections 6.4.2 and 6.4.3 apply Ruppert's analysis method to Chew's algorithm, which yields better bounds on element quality than Ruppert's.

6.3.1 Description of the Algorithm

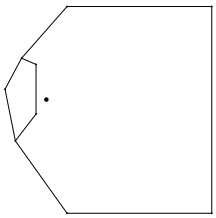
Ruppert's algorithm is presented here with a few modifications from Ruppert's original presentation. The most significant change is that the algorithm here begins with the constrained Delaunay triangulation of the segment-bounded PSLG provided as input. In contrast, Ruppert's presentation begins with a Delaunay triangulation, and the missing segments are recovered through stitching, described in Section 6.

Ruppert's algorithm inserts additional vertices (while using Lawson's algorithm or the Bowyer/Watson algorithm to maintain the constrained Delaunay property) until all triangles satisfy the constraints on quality and size set by the user. Like Chew's algorithm, Ruppert's may divide each segment into subsegments—but not as the first step of the algorithm. Instead, the algorithm interleaves segment splitting with triangle splitting. Initially, each segment comprises one subsegment. Vertex insertion is governed by two rules.

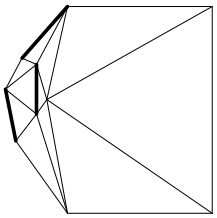
- The *diametral circle* of a subsegment is the (unique) smallest circle that encloses the subsegment. A subsegment is said to be *encroached* if a vertex other than its endpoints lies on or inside its diametral circle, and the encroaching vertex is visible from the interior of the subsegment. (Visibility is obstructed only by other segments.) Any encroached subsegment that arises is immediately split into two subsegments by inserting a vertex at its midpoint, as illustrated in Figure 6.7. 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. The order in which subsegments are split, or skinny triangles are split, is arbitrary.

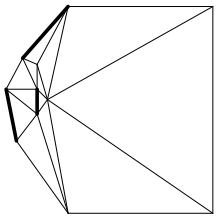
When no encroached subsegments remain, all triangles and edges of the triangulation are Delaunay. A mesh produced by Ruppert's algorithm is Delaunay, and not just constrained Delaunay.



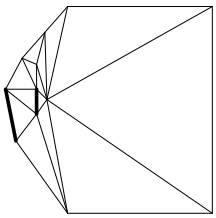
A sample input PSLG.



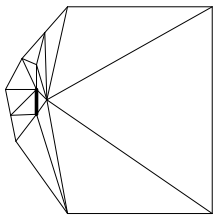
Constrained Delaunay triangulation of the input vertices. Encroached segments are bold.



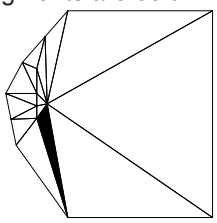
One encroached subsegment is bisected.



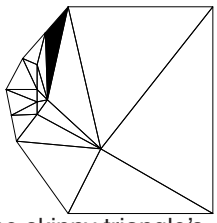
And another.



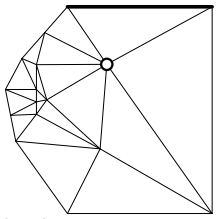
A third encroached subsegment is split.



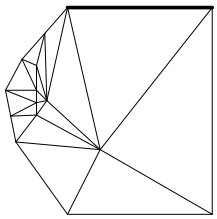
The last encroached subsegment is split. Find a skinny triangle.



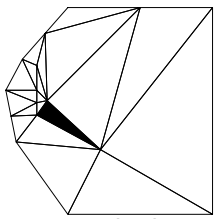
The skinny triangle's circumcenter is inserted. Find another skinny triangle.



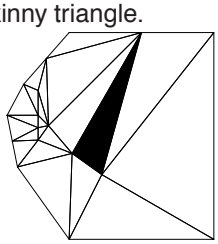
This circumcenter encroaches upon a segment, and is rejected.



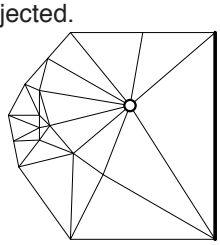
Although the vertex was rejected, the segment it encroached upon is still marked for bisection.



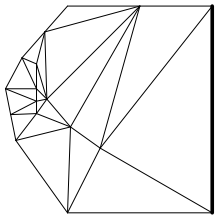
The encroached segment is split, and the skinny triangle that led to its bisection is eliminated.



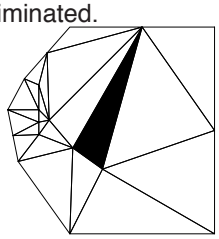
A circumcenter is successfully inserted, creating another skinny triangle.



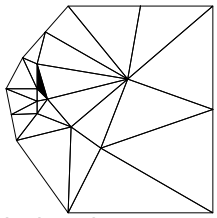
The triangle's circumcenter is rejected.



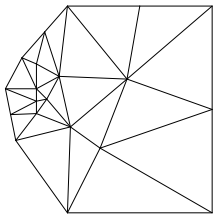
The encroached segment will be split.



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. No angle is smaller than $20.7°$.

Figure 6.8: A complete run of Ruppert's algorithm with an upper bound of $B = \sqrt{2}$ on circumradius-to-shortest edge ratios. The first two images are the input PSLG and the constrained Delaunay triangulation of its vertices. In each image, highlighted subsegments or triangles are about to be split, and open vertices are rejected because they encroach upon a subsegment.

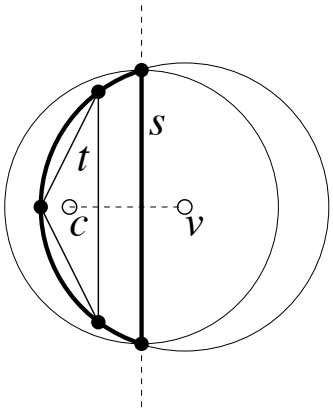


Figure 6.9: If the circumcenter v of a triangle t lies outside the triangulation, then some subsegment s is encroached.

Figure 6.8 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 to the right, but the size of the triangles on the right remains larger than the size of the triangles on the left. The smallest angle in the final mesh is 21.8°.

There is a loose end to tie up. What should happen if the circumcenter of a skinny triangle falls outside the triangulation? Fortunately, the following lemma shows the question is moot.

Lemma 36. *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 6.9 illustrates. (If there are several such subsegments, let s be the subsegment nearest c.) Because cv is entirely enclosed by the circumcircle of t, the circumcircle must enclose a portion of s; but the constrained Delaunay property requires that the circumcircle enclose no vertex visible from c, so the circumcircle cannot enclose the endpoints of s.

Because c and the center of t's circumcircle lie on opposite sides of s, the portion of the circumcircle that lies strictly on the same side of s as c (the bold arc in the illustration) is entirely enclosed by the diametral circle of s. Each vertex of t lies on t's circumcircle and either is an endpoint of s, or lies on the same side of s as c. Up to two of the vertices of t may be 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 36 offers the best 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 is also reassuring to applications (like some finite volume methods) that require all triangle circumcenters to lie within the triangulation.

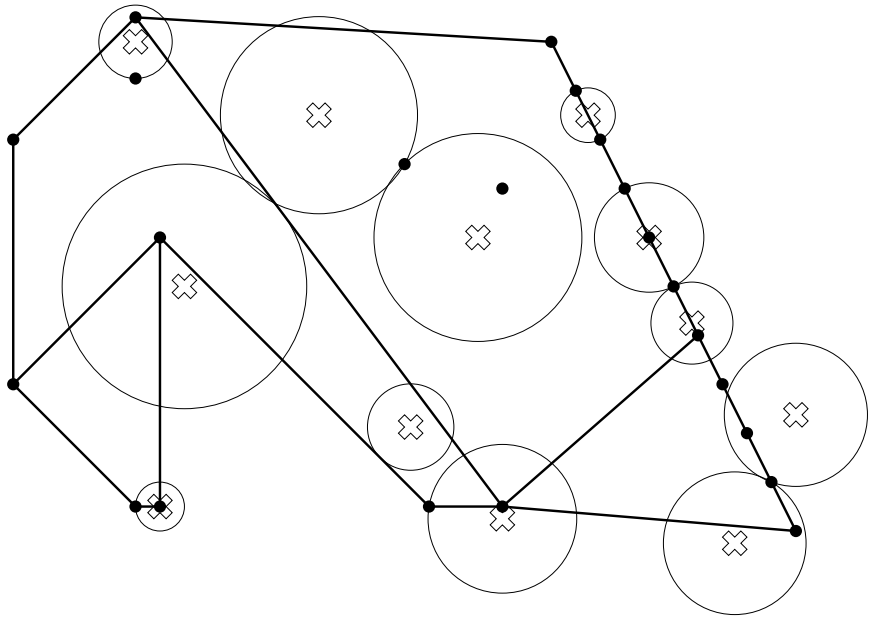


Figure 6.10: The radius of each disk illustrated here is the local feature size of the point at its center.

In addition to being required to satisfy a quality criterion, triangles can also be required to satisfy a maximum size criterion. In a finite element problem, the triangles must be small enough to ensure that the finite element solution accurately approximates the true solution of some partial differential equation. 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. 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 everywhere greater than some positive constant, there is no threat to the algorithm's termination guarantee.

6.3.2 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 the plane relative to a specific PSLG.

Given a PSLG X, the local feature size $\mathrm{lfs}(p)$ 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 6.10 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 triangulation that respects the segments and has no skinny triangles. The function $\mathrm{lfs}(\cdot)$ is continuous and has the property that its directional derivatives (where they exist) are bounded in the range $[-1, 1]$. This property leads to a lower bound (within a constant factor to be derived in Section 6.3.4) on the rate at which edge lengths grade from small to large as one moves away from a small feature. Formally, this is what it means for a mesh to be "nicely graded."

Lemma 37 (Ruppert [102])**.** *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 lfs(u) centered at u intersects two nonincident features of X. The disk having radius 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 lfs(u) + $|uv|$. ■

This lemma generalizes without change to higher dimensions. If the triangulation domain is nonconvex or nonplanar, the lemma can also be generalized to use geodesic distances—lengths of shortest paths that are constrained to lie within the triangulation domain—instead of straight-line distances. The proof relies only on the triangle inequality: if u is within a distance of lfs(u) of each of two nonincident features, then v is within a distance of lfs(u) + $|uv|$ of each of those same two features.

6.3.3 Proof of Termination

Ruppert's algorithm can eliminate any skinny triangle by inserting a vertex, but new skinny triangles might take its place. How can we be sure the process will ever stop? 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 for its intuitive value, and because it offers the best bound on the lengths of the shortest edges. The second proof, adapted from Ruppert, offers better bounds on the lengths of the longer edges of a graded mesh, and thus shows that the algorithm produces meshes that are nicely graded and size-optimal. The presentation here uses a flow graph to expose the intuition behind Ruppert's proof and its natural tendency to bound the circumradius-to-shortest edge ratio.

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 the weaker bound suffices.) 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 nearest input vertex visible from v. See Figure 6.11(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 6.11(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 6.11(c).
- If v is a vertex inserted at the circumcenter of a skinny triangle, then r_v is the circumradius of the triangle. See Figure 6.11(d).

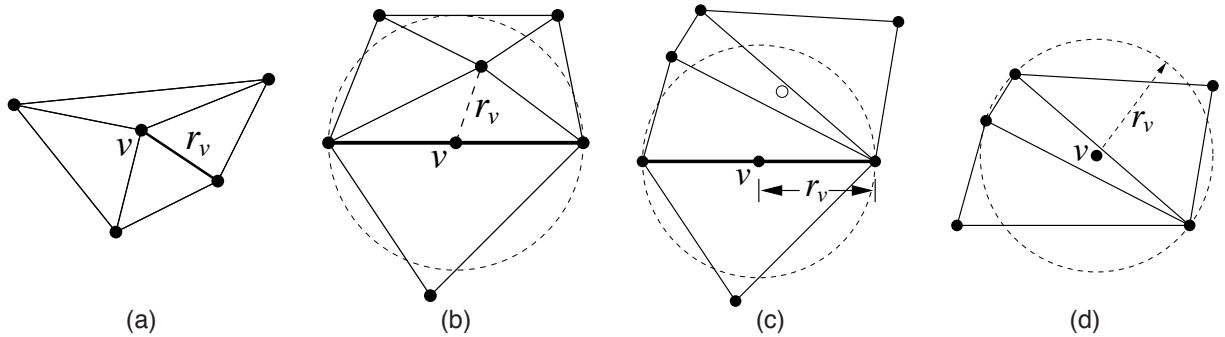


Figure 6.11: The insertion radius r_v of a vertex v is the distance to the nearest vertex when v first appears in the mesh. (a) If v is an input vertex, r_v is the distance to the nearest other input vertex. (b) If v is the midpoint of a subsegment encroached upon by a mesh vertex, r_v is the distance 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 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 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 6.12 illustrates the parents of all vertices inserted or considered for insertion during the sample execution of Ruppert's algorithm in Figure 6.8.

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—unless the descendant's local feature size is even smaller. Therefore, no edge will ever appear that is shorter than the smallest feature in the input PSLG. To prove these facts, consider the relationship between the insertion radii of a vertex and its parent.

Lemma 38. *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 triangle;*
- *$C = 1/\sqrt{2}$ if v is the midpoint of an encroached subsegment and p is the (rejected) circumcenter of a skinny triangle;*

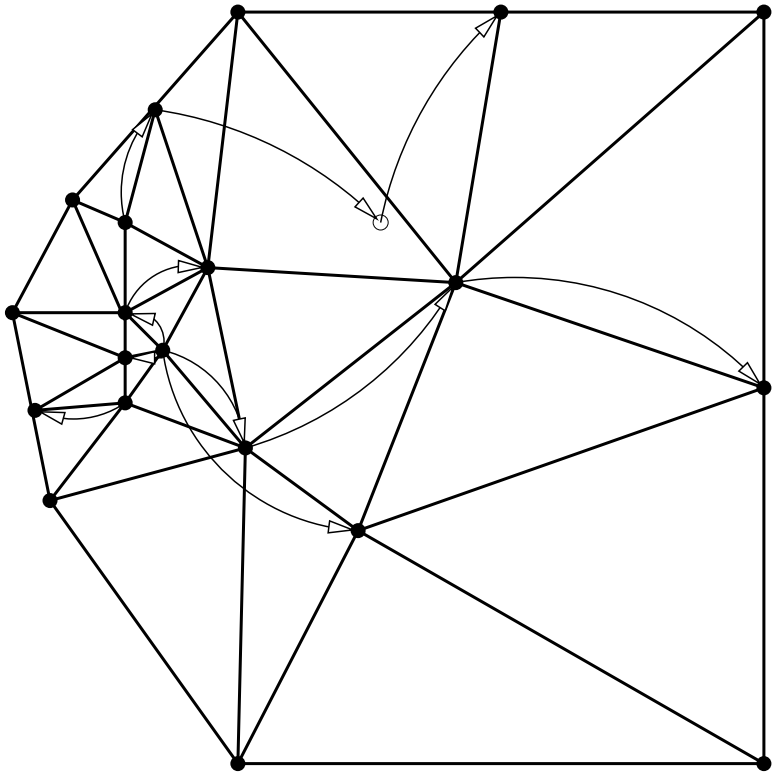


Figure 6.12: Trees of vertices for the example of Figure 6.8. Arrows are directed from parents to their children. Children include all inserted vertices and one rejected vertex.

- $C = 1/(2\cos\alpha)$ *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 $\text{lfs}(v) \leq r_v$, and the lemma holds.

If v is inserted at the circumcenter of a skinny triangle, then its parent p is the most recently inserted endpoint of the shortest edge of the triangle; see Figure 6.13(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 the termination of Ruppert's algorithm if no angle smaller than 90° is 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, $\text{lfs}(v) \leq r_v$.
- If p is a circumcenter that was considered for insertion but rejected because it encroaches upon s, then p lies on or inside the diametral circle of s. Because the mesh is constrained Delaunay, one can show that the circumcircle centered at p contains neither endpoint of s. Hence, $r_v \geq r_p/\sqrt{2}$. See Figure 6.13(b) for an example where the relation is equality.

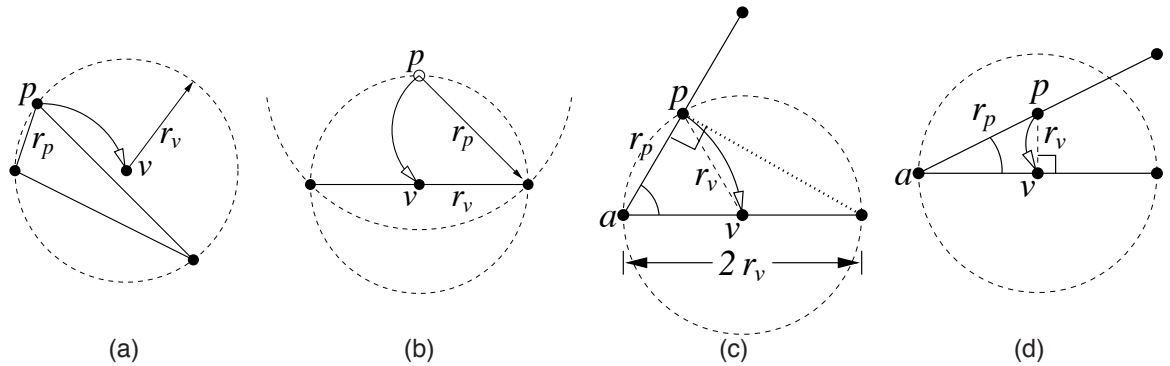


Figure 6.13: The relationship between the insertion radii of a child and its parent. (a) When a skinny triangle is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by the circumcenter of a skinny triangle, the child's insertion radius may be 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 a vertex in an incident segment, the relationship depends upon the angle α separating the two segments.

- 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 6.13(c). Because s is encroached upon by p, p lies on or inside its diametral circle. To find the worst-case (smallest) value of 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 $|s| = 2r_v$. Basic trigonometry shows that $|s| \geq r_p/\cos\alpha$, and therefore $r_v \geq r_p/(2\cos\alpha)$.
- If v and p lie on incident segments separated by an angle α where $\alpha \leq 45°$, then 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 6.13(d). Hence, $r_v \geq r_p \sin\alpha$. ■

Lemma 38 limits how quickly the insertion radii can decrease through a sequence of descendants of a vertex. If vertices with ever-smaller insertion radii cannot be generated, then edges shorter than existing features cannot be introduced, and Delaunay refinement is guaranteed to terminate.

Figure 6.14 expresses this notion as a flow 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* inserted 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.

Theorem 39. *Let* $\mathrm{lfs}_{\min}$ *be the shortest distance between two nonincident entities (vertices or segments) of the input PSLG*[1].

[1]Equivalently, $\mathrm{lfs}_{\min} = \min_u \mathrm{lfs}(u)$, 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 $\mathrm{lfs}_{\min}$ is not a lower bound for $\mathrm{lfs}(\cdot)$ over the entire domain; for instance, a segment may have length $\mathrm{lfs}_{\min}$, in which case the local feature size at its midpoint is $\mathrm{lfs}_{\min}/2$.

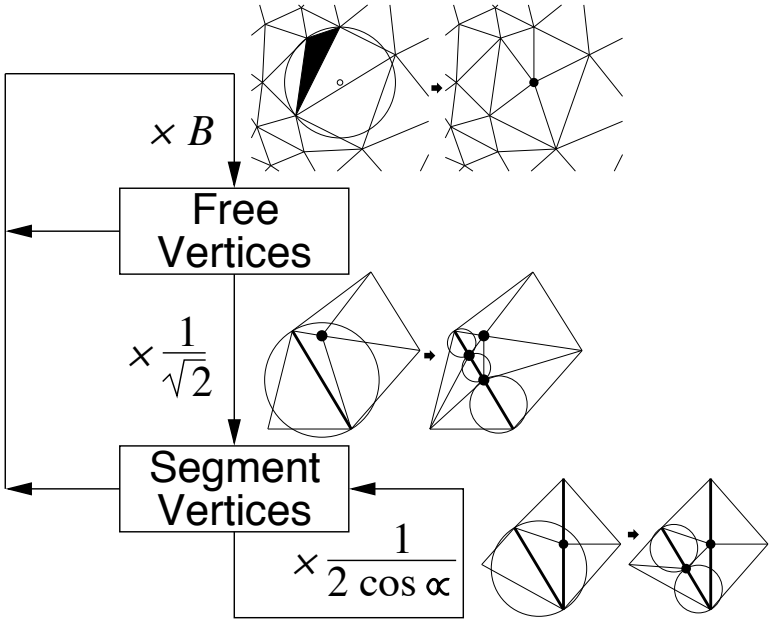


Figure 6.14: Flow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycles have a product smaller than one, Ruppert's Delaunay refinement algorithm will terminate. Input vertices are omitted from the diagram because they cannot contribute to cycles.

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 38, $r_v \geq Br_p \geq \sqrt{2}r_p$.
- If v is the midpoint of an encroached subsegment and p is the circumcenter of a skinny triangle, then by Lemma 38, $r_v \geq r_p/\sqrt{2} \geq Br_g/\sqrt{2} \geq r_g$. (Recall that p is rejected.)
- If v and p lie on incident segments, then by Lemma 38, $r_v \geq r_p/(2\cos\alpha)$. Because $\alpha \geq 60°$, $r_v \geq r_p$.

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

By design, 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 is no angle 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°. Section 6.4 describes a way to improve this bound.

What about running time? A constrained Delaunay triangulation can be constructed in $O(n \log n)$ time [34], where n is the size of the input PSLG. Once the initial triangulation is complete, well-implemented Delaunay refinement algorithms invariably take time linear in the number of additional vertices that are inserted. Ruppert (personal communication) exhibits a PSLG on which his algorithm takes $\Theta(h^2)$ time, where h is the size of the final mesh, but the example is contrived and such pathological examples do not arise in practice.

6.3.4 Proof of Good Grading and Size-Optimality

Theorem 39 guarantees that no edge of a mesh produced by Ruppert's algorithm is shorter than $\text{lfs}_{\min}$. This guarantee may be satisfying for a user who desires a uniform mesh, but it 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 lengths 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. Readers may skip this section without affecting their understanding of the rest of the chapter.

Lemma 38 was concerned with the relationship between the insertion radii of a child and its parent. The next lemma is concerned with the relationship between $\text{lfs}(v)/r_v$ and $\text{lfs}(p)/r_p$. For any vertex v, define $D_v = \text{lfs}(v)/r_v$. 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 40. *Let v be a vertex with parent $p = p(v)$. Suppose that $r_v \geq Cr_p$ (following Lemma 38). Then $D_v \leq 1 + D_p/C$.*

Proof: By Lemma 37, $\text{lfs}(v) \leq \text{lfs}(p) + |vp|$. By definition, the insertion radius r_v is $|vp|$ if p is a mesh vertex, whereas if p is a rejected circumcenter, then $r_v \geq |vp|$. Hence, we have

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

The result follows by dividing both sides by r_v. ■

Lemma 40 generalizes to any dimension, because it relies only upon Lemma 37. Ruppert's first main result follows.

Lemma 41 (Ruppert [102])**.** *Consider a mesh produced by Ruppert's algorithm. 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 $D_p = \text{lfs}(p)/r_p \leq 1$. 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 38, $r_v \geq Br_p$. Thus, by Lemma 40, $D_v \leq 1 + \max\{D_T, D_S\}/B$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen so that

$$1 + \frac{\max\{D_T, D_S\}}{B} \leq D_T. \tag{6.1}$$

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), $r_v \geq r_p/\sqrt{2}$ by Lemma 38, so by Lemma 40, $D_v \leq 1 + \sqrt{2}D_T$.

Alternatively, if p, like v, is a segment vertex, and p and v lie on incident segments, then $r_v \geq r_p/(2\cos\alpha)$ by Lemma 38, and thus by Lemma 40, $D_v \leq 1 + 2D_S\cos\alpha$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen so that

$$1 + \sqrt{2}D_T \leq D_S, \qquad \text{and} \tag{6.2}$$

$$1 + 2D_S\cos\alpha \leq D_S. \tag{6.3}$$

If the quality bound B is strictly larger than $\sqrt{2}$, Inequalities (6.1) and (6.2) 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 (6.3) and (6.1) 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 $\alpha_{\min}$. In either case, if $B > \sqrt{2}$ and $\alpha_{\min} > 60°$, there are values of D_T and D_S that satisfy all the inequalities. ■

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 density approaches infinity only after B drops below one.

Theorem 42 (Ruppert [102])**.** *For any vertex v of the output mesh, the distance to its nearest neighbor w is at least* $\text{lfs}(v)/(D_S + 1)$.

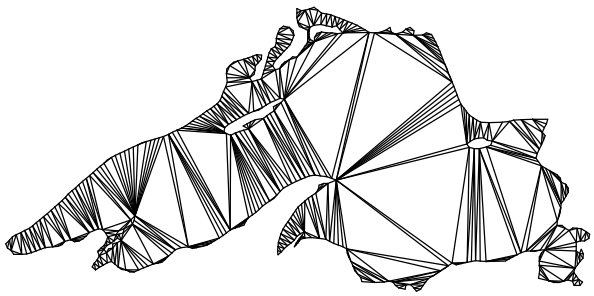
Proof: Inequality (6.2) indicates that $D_S > D_T$, so Lemma 41 shows that $\text{lfs}(v)/r_v \leq D_S$ for any vertex v. If v was added after w, then the distance between the two vertices is $r_v \geq \text{lfs}(v)/D_S$, and the theorem holds. If w was added after v, apply the lemma to w, yielding

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

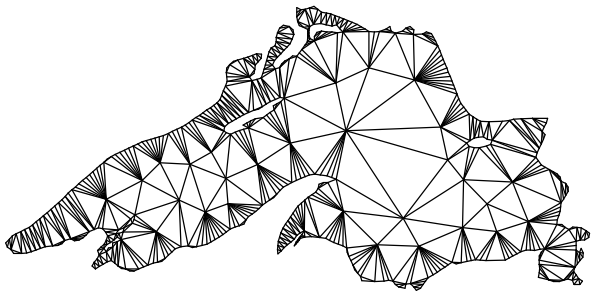
By Lemma 37, $\text{lfs}(w) + |vw| \geq \text{lfs}(v)$, so

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

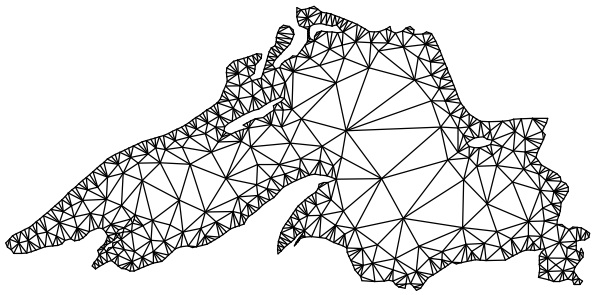
Lake Superior PSLG.



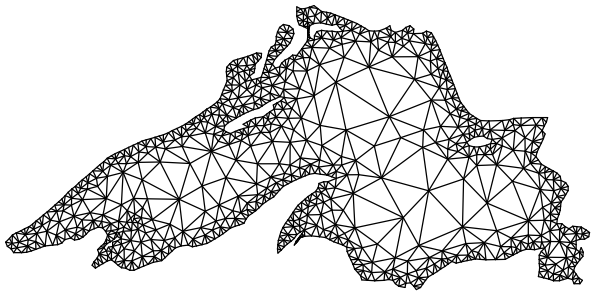
Triangulated with no minimum angle.



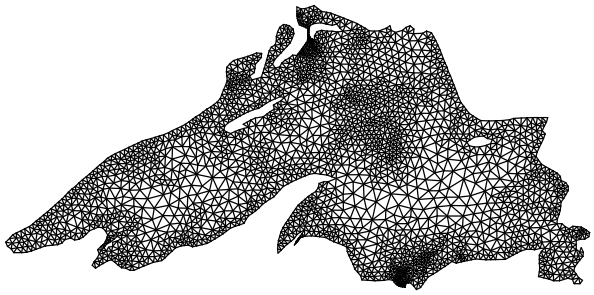
Triangulated with 5° minimum angle.



Triangulated with 15° minimum angle.



Triangulated with 25° minimum angle.



Triangulated with 34.2° minimum angle.

Figure 6.15: Meshes generated with Ruppert's algorithm for several different angle bounds. The algorithm does not terminate for angle bounds of 34.3° or higher on this PSLG.

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

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 seven times smaller than the local feature size.

Figure 6.15 illustrates the algorithm's grading 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 [102] uses Theorem 42 to prove the size optimality of the meshes his algorithm generates, and his result has been improved by Mitchell [83]. Mitchell's theorem is stated below, but the lengthy proof is omitted.

Theorem 43 (Mitchell [83]). *Let* $\mathrm{lfs}_T(p)$ *be the local feature size at* p *with respect to a triangulation* T *(treating* T *as a PSLG), whereas* $\mathrm{lfs}(p)$ *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, $k_1 \mathrm{lfs}_T(p) \geq \mathrm{lfs}(p)$. *Then the cardinality (number of triangles) of* T *is less than* k_2 *times the cardinality of any other triangulation of the input PSLG with smallest angle* θ, *where* $k_2 \in O(k_1^2/\theta)$. ■

Theorem 42 can be used to show that the precondition of Theorem 43 is satisfied by meshes generated by Ruppert's algorithm (with $k_1 \propto D_S$). Hence, the cardinality of a mesh generated by Ruppert's algorithm is within a constant factor of the cardinality of the best possible mesh satisfying the angle bound. However, the constant factor hidden in Mitchell's theorem is much too large to be a meaningful guarantee in practice.

6.4 Chew's Second Delaunay Refinement Algorithm

Compared to Ruppert's algorithm, Chew's second Delaunay refinement algorithm [37] offers an improved guarantee of good grading in theory, and splits fewer subsegments in practice. This section shows that the algorithm exhibits good grading and size optimality for angle bounds of up to 26.5° (compared with 20.7° for Ruppert's algorithm).

Chew's paper also discusses triangular meshing of curved surfaces in three dimensions, but I consider the algorithm only in its planar context.

6.4.1 Description of the Algorithm

Chew's second Delaunay refinement algorithm begins with the constrained Delaunay triangulation of a segment-bounded PSLG, and eliminates skinny triangles through Delaunay refinement, but Chew does not use diametral circles to determine if subsegments are encroached. Instead, it may arise that a skinny triangle t cannot be split because t and its circumcenter c lie on opposite sides of a subsegment s. (Lemma 36 does not apply to Chew's algorithm, so c may even lie outside the triangulation.) Although Chew does not use the word, let us say that s is *encroached* when this circumstance occurs.

Because s is a subsegment, inserting a vertex at c will not remove t from the mesh. Instead, c is rejected, and all free vertices that lie inside the diametral circle of s and are visible from the midpoint of s are deleted from the triangulation. (Input vertices and segment vertices are not deleted.) Then, a new vertex is inserted at the midpoint of s. The constrained Delaunay property is maintained throughout all vertex deletions and insertions. Figure 6.16 illustrates a subsegment split in Chew's algorithm.

If several subsegments lie between t and c, only the subsegment nearest t is split. If no subsegment lies between t and c, but c lies precisely on a subsegment, then that subsegment is considered encroached and split at its midpoint.

Chew's second algorithm produces a mesh that is not guaranteed to be Delaunay (only constrained Delaunay). For the few applications that require truly Delaunay triangles, Ruppert's algorithm is preferable. For the majority of applications, however, Chew has two advantages. First, some subsegment splits are avoided that would otherwise have occurred, so the final mesh may have fewer triangles. Consider two contrasting examples. In Figure 6.6 (bottom), the segments are so short that few are ever encroached, so Ruppert and Chew generate virtually the same mesh. In Figure 6.17, the segments are long compared to their local feature sizes, and Chew produces many fewer triangles.

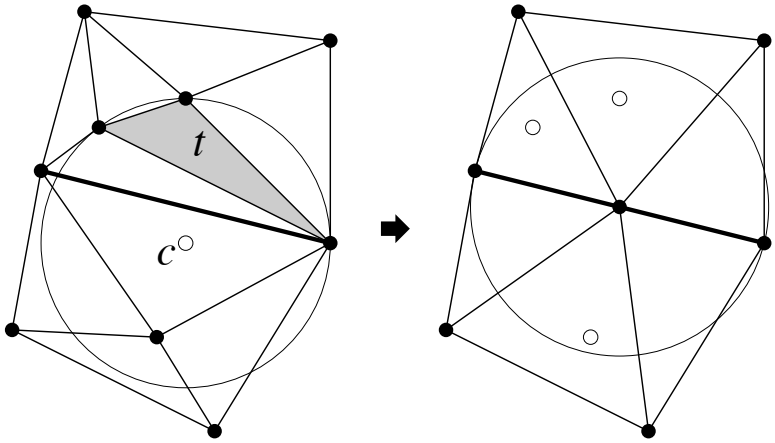


Figure 6.16: At left, a skinny triangle and its circumcenter lie on opposite sides of a subsegment. At right, all vertices in the subsegment's diametral circle have been deleted, and a new vertex has been inserted at the subsegment's midpoint.

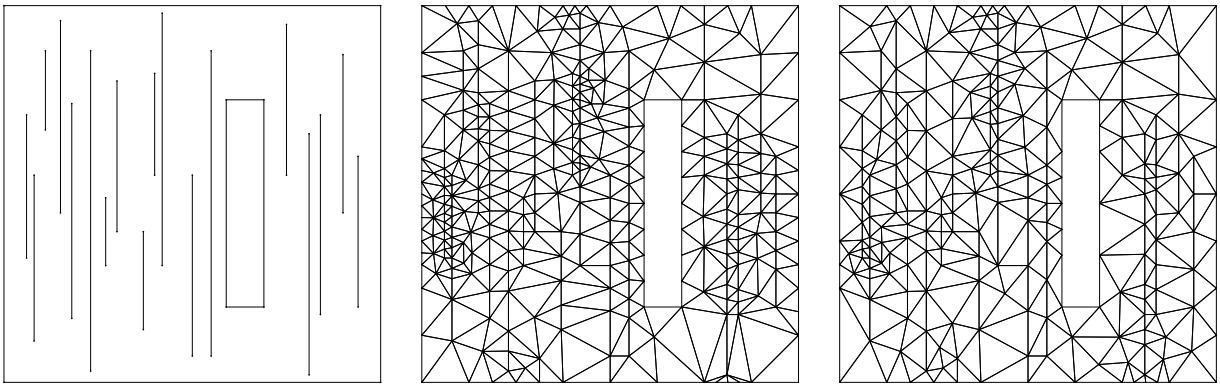


Figure 6.17: A PSLG, a 559-triangle mesh produced by Ruppert's algorithm, and a 423-triangle mesh produced by Chew's second algorithm. No angle in either mesh is smaller than 25°.

The second advantage is that when a subsegment is split by a vertex v with parent p, a better bound can be found for the ratio between r_v and r_p than Lemma 38's bound. This improvement leads to better bounds on the minimum angle, the edge lengths, and the mesh cardinality.

6.4.2 Proof of Termination

If no input angle is less than 60°, Chew's algorithm terminates for any bound on circumradius-to-shortest edge ratio B such that $B \geq \sqrt{5}/2 \doteq 1.12$. Therefore, the smallest angle can be bounded by up to $\arcsin(1/\sqrt{5}) \doteq 26.56°$.

By the reasoning of Lemma 36, if a triangle and its circumcenter lie on opposite sides of a subsegment, or if the circumcenter lies on the subsegment, then some vertex of the triangle (other than the subsegment's endpoints) lies on or inside the subsegment's diametral circle. Hence, Chew's algorithm never splits a subsegment that Ruppert's algorithm would not split. It follows that the inequalities in Lemma 38 are as true for Chew's algorithm as they are for Ruppert's algorithm. However, Chew will often decline to split a subsegment that Ruppert would split, and thus splits fewer subsegments overall. A consequence is that the

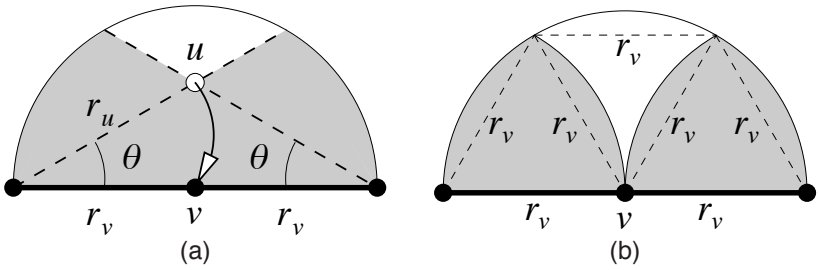


Figure 6.18: (a) The case where exactly one vertex is in the semicircle. (b) The case where more than one vertex is in the semicircle.

relationship between the insertion radii of a subsegment midpoint and its parent can be tightened.

Lemma 44. *Let $\theta = \arcsin \frac{1}{2B}$ be the angle bound below which a triangle is considered skinny. Let s be a subsegment that is encroached because some skinny triangle t and its circumcenter c lie on opposite sides of s (or c lies on s). Let v be the vertex inserted at the midpoint of s. Then one of the following four statements is true. (Only the fourth differs from Lemma 38.)*

- $r_v \geq \text{lfs}(v)$*;*
- $r_v \geq r_p/(2\cos\alpha)$*, where p is a vertex that encroaches upon s and lies in a segment separated by an angle of $\alpha \geq 45°$ from the segment containing s;*
- $r_v \geq r_p \sin\alpha$*, where p is as above, with $\alpha \leq 45°$; or*
- *there is some vertex p (which is deleted from inside the diametral circle of s or lies precisely on the diametral circle) such that $r_v \geq r_p \cos\theta$.*

Proof: Chew's algorithm deletes all free vertices inside the diametral circle of s that are visible from v. If any vertex remains visible from v inside the diametral circle, it is an input vertex or a segment vertex. Define the parent p of v to be the closest such vertex. If p is an input vertex or lies on a segment not incident to the segment that contains s, then $\text{lfs}(v) \leq r_v$ and the lemma holds. If p lies on an incident segment, then $r_v \geq r_p/(2\cos\alpha)$ for $\alpha \geq 45°$ or $r_v \geq r_p \cos\theta$ for $\alpha \leq 45°$ as in Lemma 38.

Otherwise, no vertex inside the diametral circle of s is visible after the deletions, so r_v is equal to the radius of the diametral circle. This is the reason why Chew's algorithm deletes the vertices: when v is inserted, the nearest visible vertices are the subsegment endpoints, and no short edge appears.

Mentally jump back in time to just before the vertex deletions. Assume without loss of generality that t lies above s, with c below. Following Lemma 36, at least one vertex of t lies on or inside the upper half of the diametral circle of s. There are two cases, depending on the total number of vertices on or inside this semicircle.

If the upper semicircle encloses only one vertex u visible from v, then t is the triangle whose vertices are u and the endpoints of s. Because t is skinny, u must lie in the shaded region of Figure 6.18(a). The insertion radius r_u cannot be greater than the distance from u to the nearest endpoint of s, so $r_v \geq r_u \cos\theta$. (For a fixed r_v, r_u is maximized when u lies at the apex of the isosceles triangle whose base is s and whose base angles are θ.) Define the parent of v to be u.

If the upper semicircle encloses more than one vertex visible from v, consider Figure 6.18(b), in which the shaded region represents points within a distance of r_v from an endpoint of s. If some vertex u lies in

the shaded region, then $r_u \leq r_v$; define the parent of v to be u. If no vertex lies in the shaded region, then there are at least two vertices visible from v in the white region of the upper semicircle. Let u be the most recently inserted of these vertices. The vertex u is at a distance of at most r_v from any other vertex in the white region, so $r_u \leq r_v$; define the parent of v to be u. ■

Lemma 44 extends the definition of parent to accommodate the new type of encroachment defined in Chew's algorithm. When a subsegment s is encroached, the parent p of its newly inserted midpoint v is defined to be a vertex on or inside the diametral circle of s, just as in Ruppert's algorithm.

Chew's algorithm can be shown to terminate in the same manner as Ruppert's. Do the differences between Chew's and Ruppert's algorithms invalidate any of the assumptions used in Theorem 39 to prove termination? The most important difference is that vertices may be deleted from the mesh. When a vertex is deleted from a constrained Delaunay triangulation, no surviving vertex finds itself adjoining a shorter edge than the shortest edge it adjoined before the deletion. (This fact follows because a constrained Delaunay triangulation connects every vertex to its nearest visible neighbor.) Hence, each vertex's insertion radius still serves as a lower bound on the lengths of all edges that connect the vertex to vertices older than itself.

If vertices can be deleted, are we certain that the algorithm will run out of places to put new vertices? Observe that vertex deletions only occur when a subsegment is split, and vertices are never deleted from segments. Theorem 39 sets a lower bound on the length of each subsegment, so only a finite number of subsegment splits can occur. After the last subsegment split, no more vertex deletions occur, and eventually there will be no space left for new vertices. Therefore, Theorem 39 and Lemma 41 hold for Chew's algorithm as well as Ruppert's.

The consequence of the bound proven by Lemma 44 is illustrated in the flow graph of Figure 6.19. Recall that termination is guaranteed if no cycle has a product less than one. Hence, a condition of termination is that $B \cos\theta \geq 1$. As $\theta = \arcsin \frac{1}{2B}$, the best bound that satisfies this criterion is $B = \sqrt{5}/2 \doteq 1.12$, which corresponds to an angle bound of $\arcsin(1/\sqrt{5}) \doteq 26.56°$.

6.4.3 Proof of Good Grading and Size Optimality

The main point of this section is to demonstrate that Chew's algorithm offers better theoretical guarantees about triangle quality, edge lengths, and good grading than Ruppert's. (We should not forget, though, that it is Ruppert's analysis technique that allows us to draw this conclusion.) Whereas Ruppert only guarantees good grading and size optimality for angle bounds less than about 20.7°, Chew can make these promises for angle bounds less than about 26.5°, and offer better bounds on edge lengths for the angle bounds where Ruppert's guarantees do hold. However, the differences are not as pronounced in practice as in theory. Readers whose interests are purely practical may skip this section without affecting their understanding of the rest of the chapter.

Let's compare Chew's algorithm.

Lemma 45. *Consider a mesh produced by Chew's algorithm. Suppose the quality bound B is strictly larger than* $\sqrt{5}/2$*, 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 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,* $D_v \leq D_S$*.*

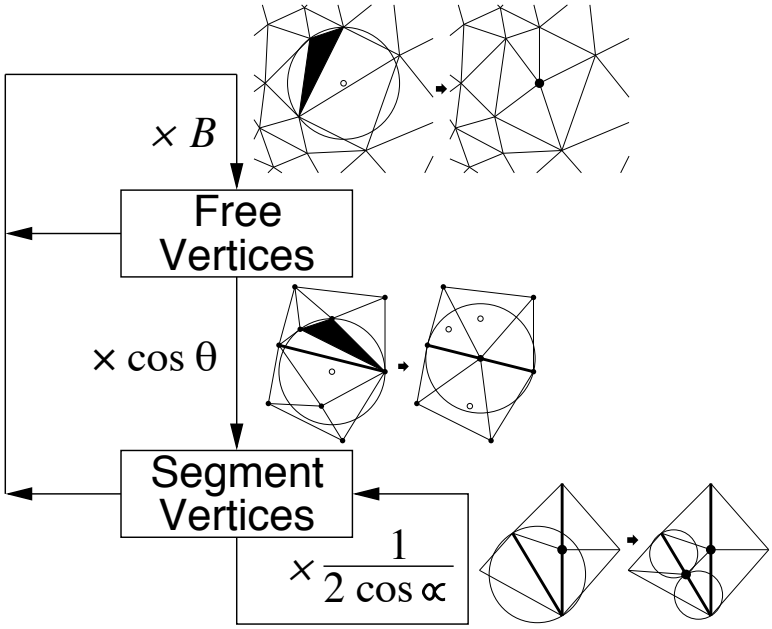


Figure 6.19: Flow diagram for Chew's algorithm.

Proof: Essentially the same as the proof of Lemma 41, except that Lemma 44 makes it possible to replace Inequality (6.2) with

$$\begin{aligned} D_S &\geq 1 + \frac{D_T}{\cos\theta} \\ &\geq 1 + \frac{2BD_T}{\sqrt{4B^2 - 1}} \end{aligned} \tag{6.4}$$

If the quality bound B is strictly larger than $\sqrt{5}/2$, Inequalities (6.1) and (6.4) are simultaneously satisfied by choosing

$$D_T = \frac{\left(1 + \frac{1}{B}\right)\sqrt{4B^2 - 1}}{\sqrt{4B^2 - 1} - 2}, \qquad D_S = \frac{\sqrt{4B^2 - 1} + 2B}{\sqrt{4B^2 - 1} - 2}.$$

D_T and D_S must also satisfy Inequality (6.3), so larger values of D_T and D_S may be needed, as in Lemma 41. However, If $B > \sqrt{5}/2$ and $\alpha_{\min} > 60°$, there are values of D_T and D_S that satisfy all the inequalities. ■

Theorem 42, which bounds the edge lengths of the mesh, applies to Chew's algorithm as well as Ruppert's, but the values of D_T and D_S are different. As in Section 6.3.4, consider triangulating a PSLG (free of acute angles) so that no angle of the output mesh is smaller than 15°. Then $D_T \doteq 3.27$, and $D_S \doteq 4.39$, compared to the corresponding values of 5.66 and 9.01 for Ruppert's algorithm. Hence, the spacing of vertices is at worst a little more than five times the local feature size, and a little more than four times the local feature size away from segments.

Because the worst-case number of triangles is proportional to the square of D_S, Chew's algorithm is size-optimal with a constant of optimality almost four times better than Ruppert's algorithm. However, worst-case behavior is never seen in practice, and the observed difference between the two algorithms is less dramatic.

Chapter 7

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 [45] illustrate, most tetrahedra with poor angles have circumcircles much larger than their shortest edges, including the needle, wedge, and cap illustrated in Figure 7.1. 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 1.11 illustrates. A sliver can have

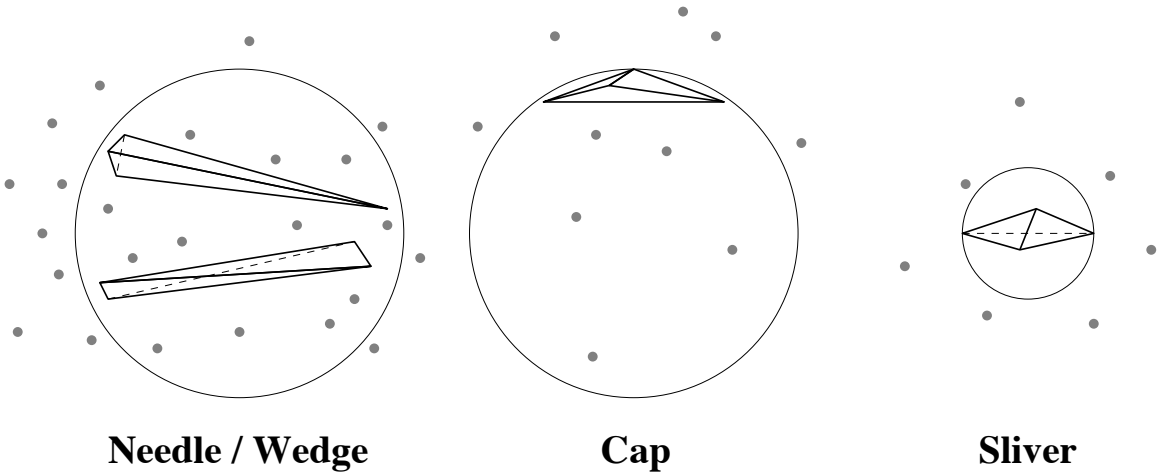


Figure 7.1: Tetrahedra with poor angles. Needles and wedges have edges of greatly disparate length; caps have a large solid angle; slivers have neither, and can have good circumradius-to-shortest edge ratios. Needles, wedges, and caps have circumspheres significantly larger than their shortest edges, 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.

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 7.1 illustrates). A perfectly flat sliver whose edge lengths are $\text{lfs}_{\min}$ about the equator and $\sqrt{2}\text{lfs}_{\min}$ across the diagonals is guaranteed to survive any Delaunay refinement method that does not introduce edges smaller than $\text{lfs}_{\min}$, because every point in the interior of its circumsphere is a distance less than $\text{lfs}_{\min}$ 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 7.3 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. The most notable of these is the sliver exudation algorithm of Cheng, Dey, Edelsbrunner, Facello, and Teng [30], 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 [81].

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.

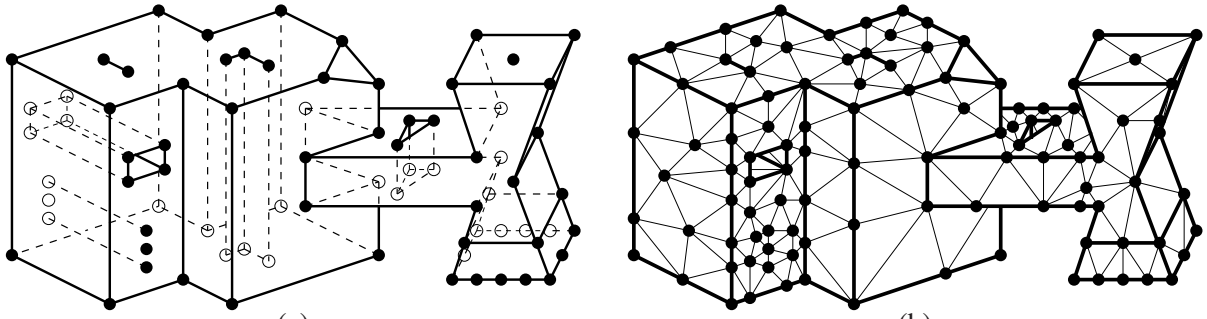


Figure 7.2: (a) Any facet of a PLC may contain holes, slits, and vertices. (b) When a PLC is tetrahedralized, each facet of the PLC is partitioned into triangular subfacets, which respect the holes, slits, and vertices.

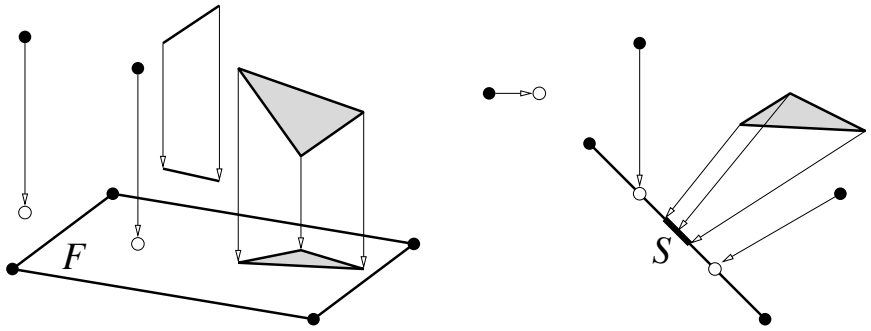


Figure 7.3: The orthogonal projections of points and sets of points onto facets and segments.

7.1 Definitions

Tetrahedral mesh generation necessarily divides each facet of a PLC, like that depicted in Figure 7.2(a), into triangular faces, as illustrated in Figure 7.2(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 7.2(b) are subsegments; other edges are not. All of the triangular faces visible in Figure 7.2(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 7.3. 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 $\{\text{proj}_F(p) : p \in G\}$.

7.2 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, Tal-

mor, Teng, Walkington, and Wang [82] have developed a related algorithm that does not use Delaunay refinement.

7.2.1 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 [20, 128] or three-dimensional flipping [66, 96], 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 7.4(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 7.4(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.

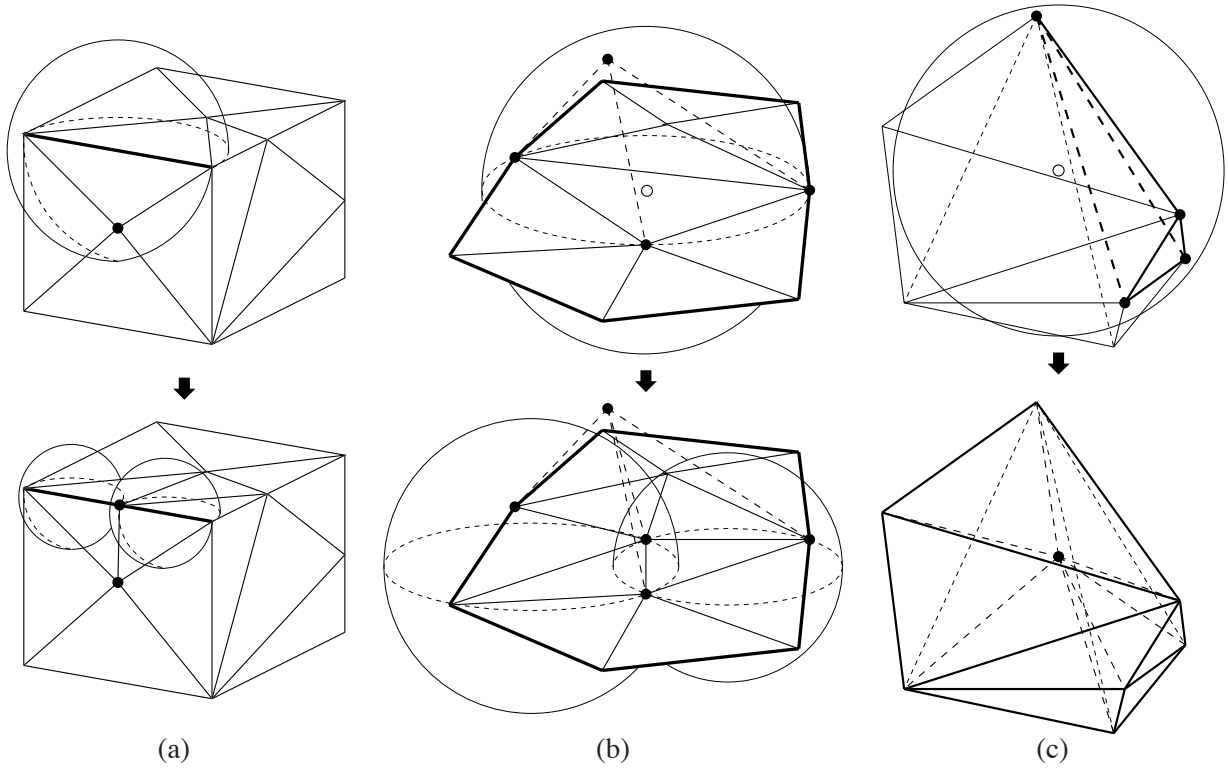


Figure 7.4: Three operations for three-dimensional Delaunay refinement. (a) Splitting an encroached subsegment. Dotted arcs indicate where diametral spheres intersect faces. The original subsegment is encroached because there is a vertex in its diametral sphere. In this example, the two subsegments created 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.

- 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 by inserting a vertex at its circumcenter, thus eliminating the tetrahedron; see Figure 7.4(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 36 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, described in Section 6. If a subsegment is missing from a Delaunay triangulation, then the subsegment is not strongly Delaunay, so there must be a vertex (other than

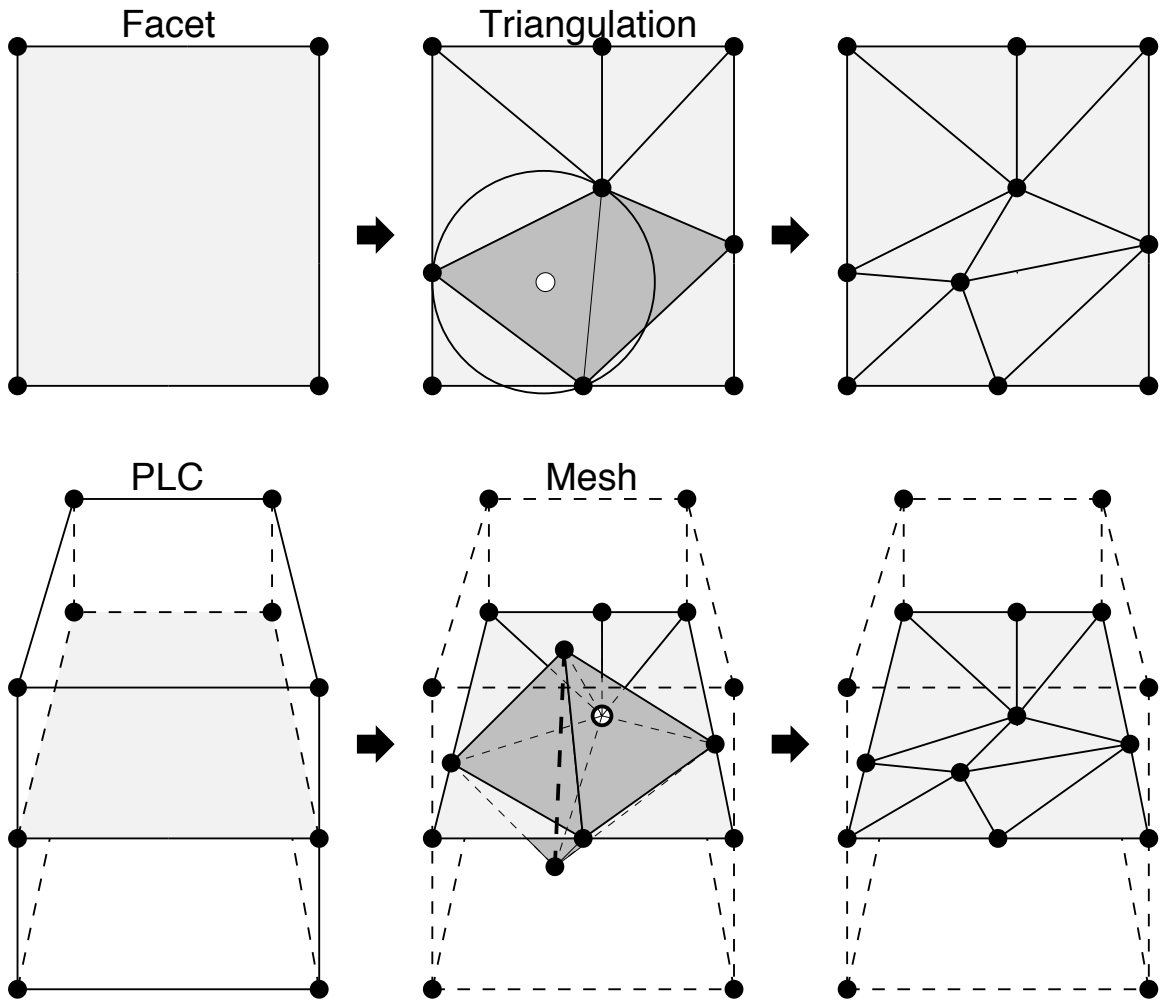


Figure 7.5: The top illustrations depict a rectangular facet and its triangulation. The bottom illustrations depict the facet's position as an interior boundary of a PLC, and its progress as it is recovered. Most of the vertices and tetrahedra of the mesh are omitted for clarity. The facet triangulation 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.

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.

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 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 tetra-

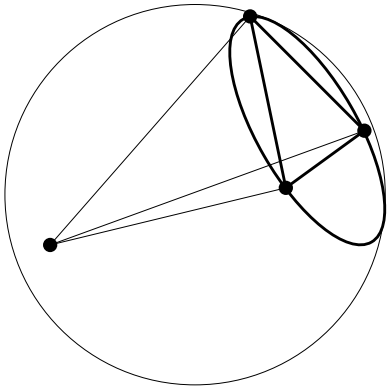


Figure 7.6: If a tetrahedron is Delaunay, the circumcircle of each of its faces is empty, because each face's circumcircle is a cross-section of the tetrahedron's circumsphere.

hedralization, 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 7.5. The new vertex is independently inserted into both the facet triangulation and the tetrahedralization. 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 7.5, 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 7.6. 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 4.5.1 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 7.7 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

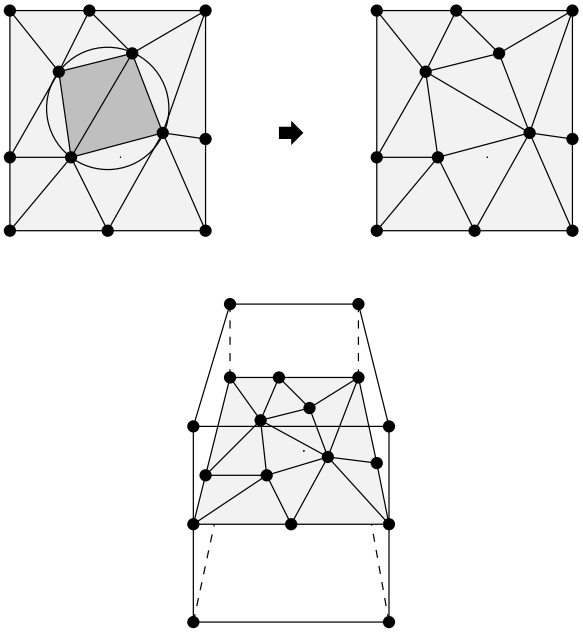


Figure 7.7: A facet triangulation and a tetrahedralization may disagree due to cocircular vertices. This occurrence should be diagnosed and fixed as shown here.

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 [61, 129, 99]. In many tetrahedral mesh generators, facets are inserted by identifying points where the edges of the tetrahedralization intersect a missing facet, and inserting vertices at these points. The perils of so doing are illustrated in Figure 7.8. 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 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. 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

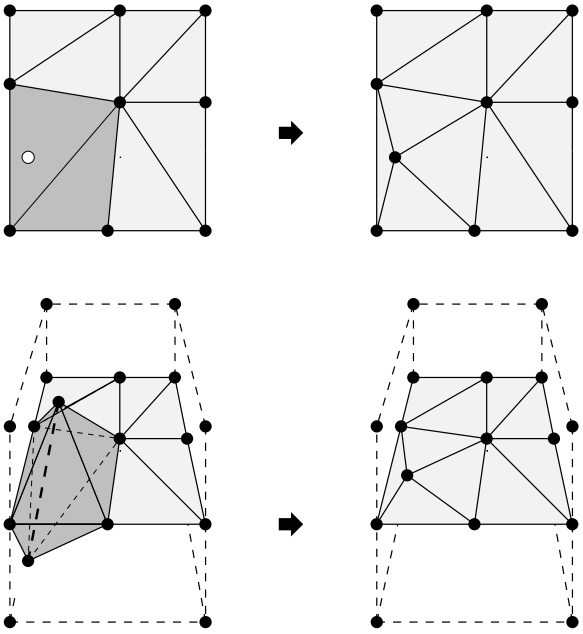


Figure 7.8: One may recover a missing facet by inserting vertices at the intersections of the facet with edges of the tetrahedralization, but this method might create arbitrarily small features by placing vertices close to segments.

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 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 38, 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.

Consider the two examples of Figure 7.9. 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 7.9(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 7.9(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

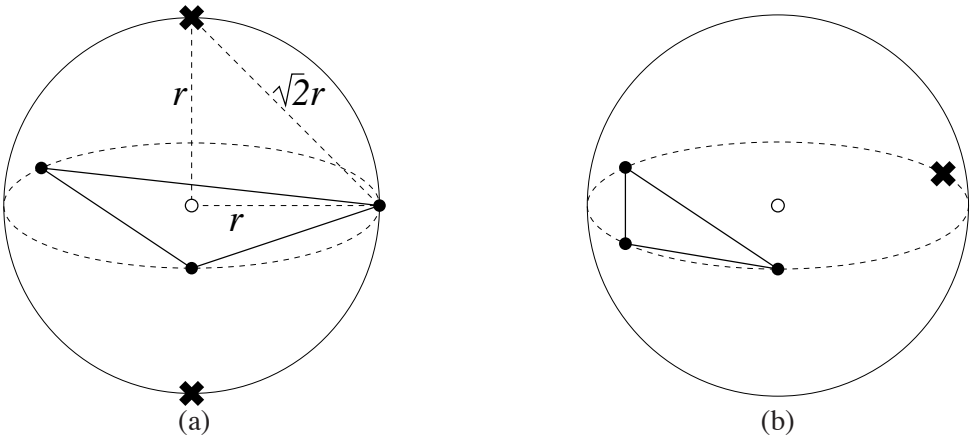


Figure 7.9: The relationship between the insertion radii of the circumcenter of an encroached subfacet and the encroaching vertex. Crosses identify the location of an encroaching vertex having maximum distance from the nearest subfacet vertex. (a) If the encroached subfacet contains its own circumcenter, 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.

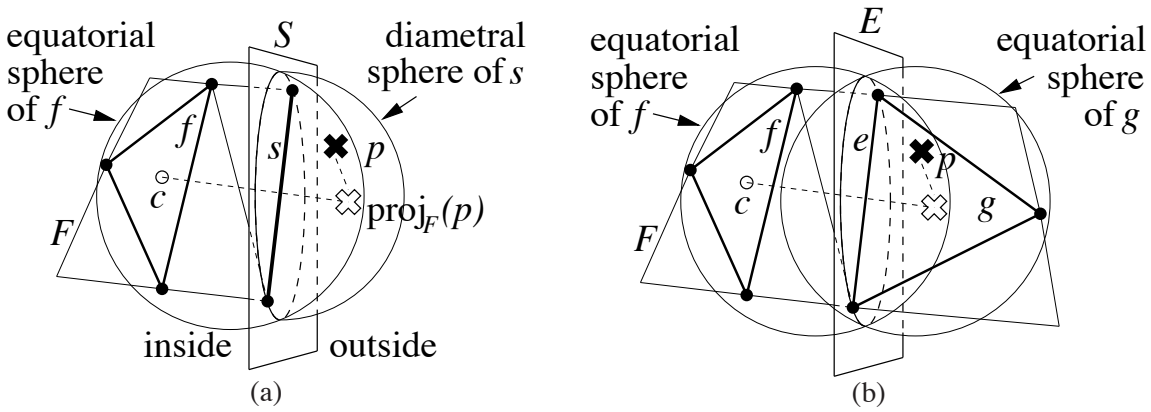


Figure 7.10: Two properties of encroached Delaunay subfacets. (a) If a vertex p encroaches upon a Delaunay subfacet f of a facet F, but its projection into the plane containing F lies outside F, then p encroaches upon some subsegment s of F as well. (b) If a vertex p encroaches upon a subfacet f of a 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 $\text{proj}_F(p)$.

splits g first and delays the splitting of f indefinitely.

Lemma 46 (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 36. 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 7.10(a).

Because f is a Delaunay subfacet of F, its circumcircle (in the plane containing F) encloses no vertex

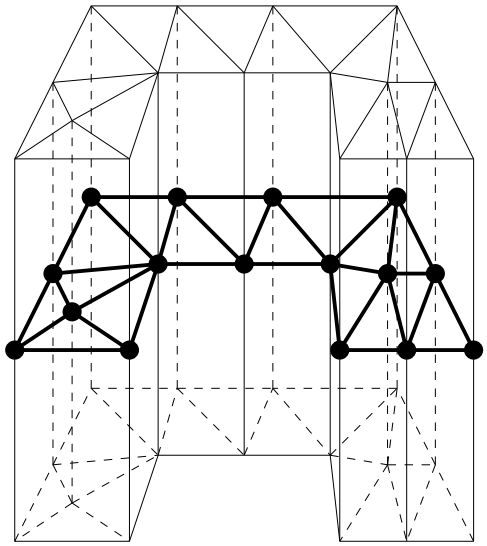


Figure 7.11: Each subfacet's equatorial sphere dominates the triangular prism defined by extending the subfacet orthogonally.

of F. However, its equatorial sphere may enclose vertices—including p—and f might not appear in the tetrahedralization.

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 36. 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 7.10(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 7.11). 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.

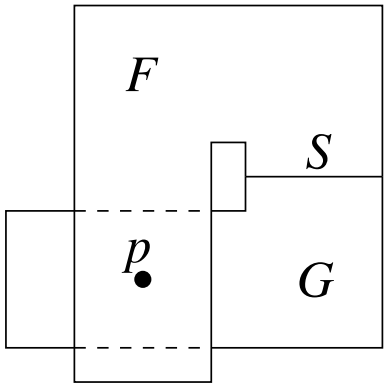


Figure 7.12: Two incident facets separated by a dihedral angle of nearly 180°. What is the local feature size at p?

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.

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.

7.2.2 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 7.12 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 7.13 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

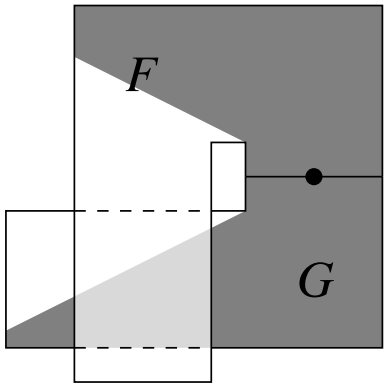


Figure 7.13: Shaded areas are two incident facet regions. Both regions are visible from the indicated point.

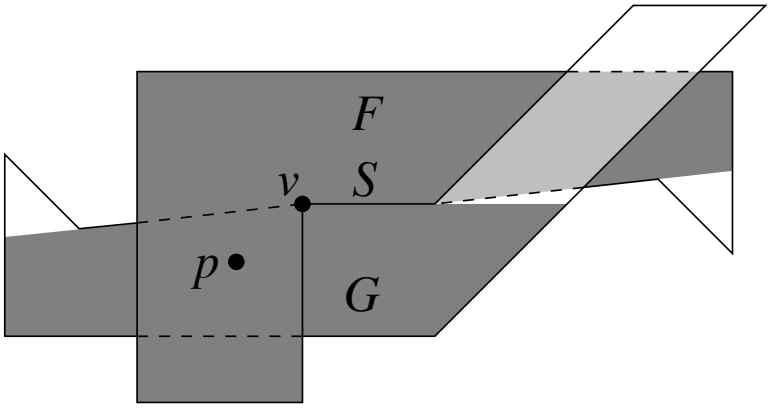


Figure 7.14: Two incident facets separated by a dihedral angle of nearly 180°. The definition of local feature size should not approach zero near v, but it is nonetheless difficult to mesh the region between F and G near v.

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 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 $\mathrm{lfs}(p)$ 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 7.14 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 7.12: 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, $\mathrm{lfs}(p)$ 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, $\mathrm{lfs}(p)$ 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 7.14—a difficulty that is not present in Figure 7.12. In Figure 7.14, it is not possible to mesh the region between F and G at v without resorting to poorly shaped tetrahedra. The facets of Figure 7.12 can be meshed entirely with well-shaped tetrahedra. The three-dimensional Delaunay refinement algorithm discussed here outlaws inputs like Figure 7.14, at least for the purposes of analysis.

Lemma 37, 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 37, besides the triangle inequality, is that there be a consistent definition of which pairs of points are nonincident feature points.

7.2.3 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 $\arccos \frac{1}{2\sqrt{2}} \doteq 69.3°$. 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 46, 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 7.14, 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 38 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 47. *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 $45° \leq \alpha < 90°$.*

Proof: If v is an input vertex, the circumcenter of a tetrahedron (Figure 7.15(a)), or the midpoint of an encroached subsegment, then it may be treated exactly as in Lemma 38. 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 7.15(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 7.15(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 $\alpha \geq 45°$ (Figure 7.15(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

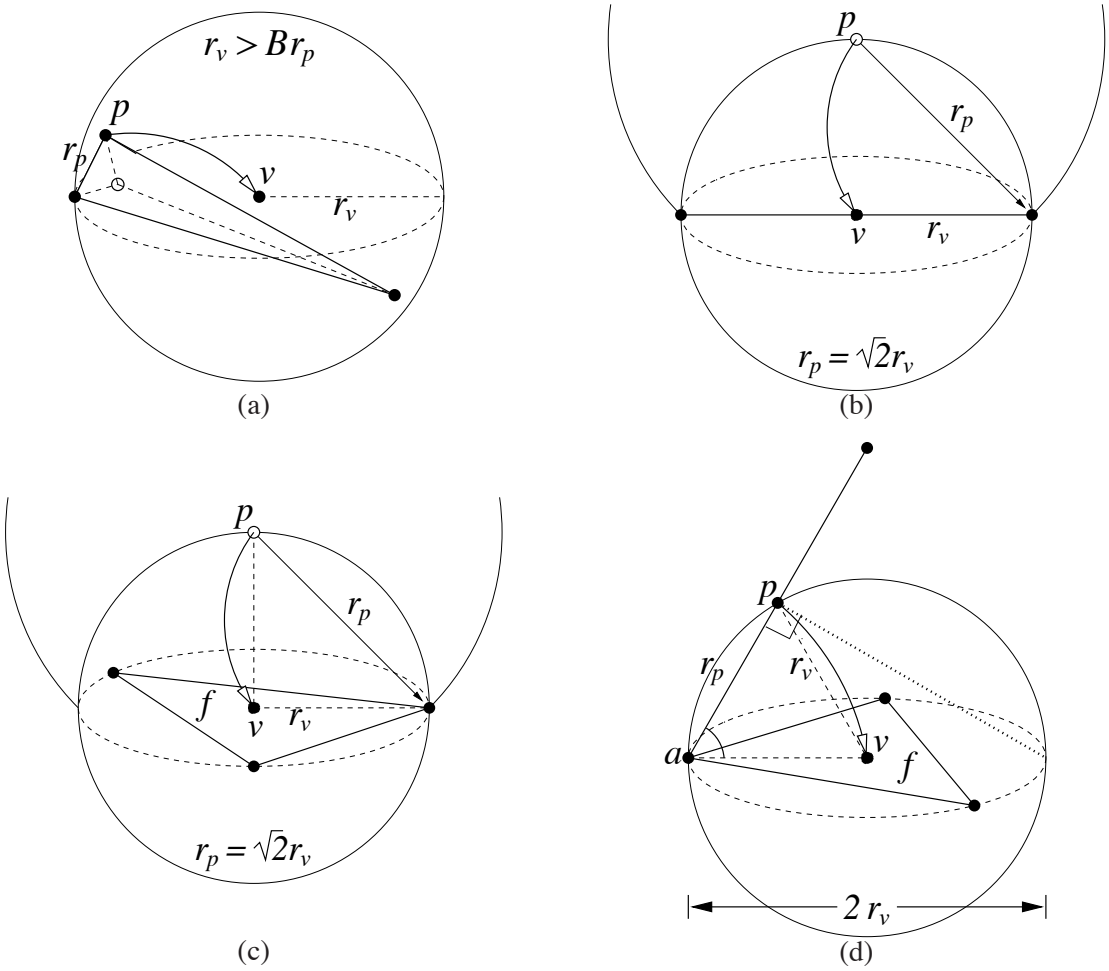


Figure 7.15: The relationship between the insertion radii of a child and its parent. (a) When a skinny tetrahedron is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by a circumcenter, the child's insertion radius may be 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.

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, p might lie on its equatorial sphere in a degenerate case.) Analogously to the case of two incident segments (see Lemma 38), 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 47 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.

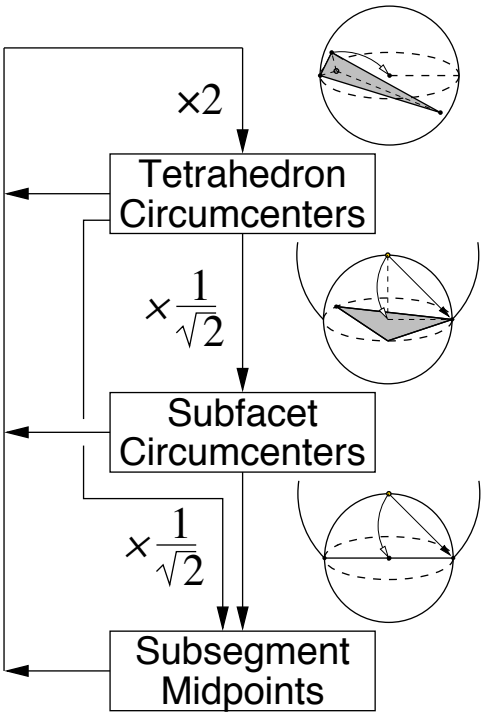


Figure 7.16: Flow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycle has a product smaller than one, the three dimensional Delaunay refinement algorithm will terminate.

Figure 7.16 depicts a flow graph corresponding to Lemma 47. Mesh vertices are divided into four classes: input vertices (which cannot contribute to cycles), segment vertices (inserted into segments), facet 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 $\arccos \frac{1}{2\sqrt{2}} \doteq 69.3°$ 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 47 that $r_v \geq \frac{r_p}{2\cos\alpha}$.)

Theorem 48. *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* $\text{lfs}_{\min}$.

Proof: Suppose for the sake of contradiction that the algorithm introduces one or more edges 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 feature points. 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, 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 47, $r_v \geq Br_p \geq 2r_p$.
- 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 47, $r_v \geq \frac{1}{\sqrt{2}} r_p \geq \frac{B}{\sqrt{2}} r_g \geq \sqrt{2} r_g$.
- 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 47, $r_v \geq \frac{1}{\sqrt{2}} r_p \geq \frac{1}{2} r_g \geq \frac{B}{2} r_h \geq r_h$.
- If v and p lie on incident segments, then by Lemma 47, $r_v \geq \frac{r_p}{2\cos\alpha}$. Because $\alpha \geq 60°$, $r_v \geq r_p$.
- 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 47, $r_v \geq \frac{r_p}{2\cos\alpha}$. 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 47, $r_v \geq \frac{1}{\sqrt{2}} r_p \geq \frac{1}{2\sqrt{2}\cos\alpha} r_g$. 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 $\text{lfs}_{\min}$, contradicting the assumption that e has length less than $\text{lfs}_{\min}$.

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

7.2.4 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 40, 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 $D_v = \frac{\text{lfs}(v)}{r_v}$ and $D_p = \frac{\text{lfs}(p)}{r_p}$.

Lemma 49. *Suppose the quality bound B is strictly larger than 2, and all angles between segments and facets satisfy the conditions listed in Theorem 48, 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 47. 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 47, $r_v \geq Br_p$. Therefore, by Lemma 40, $D_v \leq 1 + \frac{\max\{D_T, D_F, D_S\}}{B}$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen sufficiently large that

$$1 + \frac{\max\{D_T, D_F, D_S\}}{B} \leq D_T. \tag{7.1}$$

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 47, so by Lemma 40, $D_v \leq 1 + \sqrt{2}D_T$.

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 47, and thus by Lemma 40, $D_v \leq 1 + 2D_S \cos\alpha$. It follows that one can prove that $D_v \leq D_F$ if D_F is chosen sufficiently large that

$$1 + \sqrt{2}D_T \leq D_F, \qquad \text{and} \tag{7.2}$$

$$1 + 2D_S \cos\alpha \leq D_F. \tag{7.3}$$

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 47, so by Lemma 40, $D_v \leq 1 + \sqrt{2}\max\{D_T, D_F\}$.

Alternatively, if p and v lie on incident segments, then $r_v \geq \frac{r_p}{2\cos\alpha}$ by Lemma 47, and thus by Lemma 40, $D_v \leq 1 + 2D_S \cos\alpha$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen sufficiently large that

$$1 + \sqrt{2}\max\{D_T, D_F\} \leq D_S \qquad \text{and} \tag{7.4}$$

$$1 + 2D_S \cos\alpha \leq D_S. \tag{7.5}$$

If the quality bound B is strictly larger than 2, Inequalities (7.1), (7.2), and (7.4) 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 $\arccos\frac{1}{2\sqrt{2}} \doteq 69.3°$, Inequalities (7.3) and (7.4) 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 (7.1).

If the smallest angle α_{SS} between two segments is strictly greater than 60°, Inequality (7.5) 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 (7.1) and (7.2). ■

Theorem 50. *For any vertex v of the output mesh, the distance to its nearest neighbor is at least* $\frac{\text{lfs}(v)}{D_S+1}$.

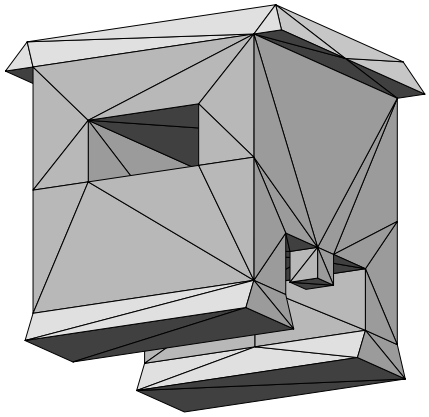
Proof: Inequality (7.4) indicates that D_S is larger than D_T and D_F. The remainder of the proof is identical to that of Theorem 42. ■

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.

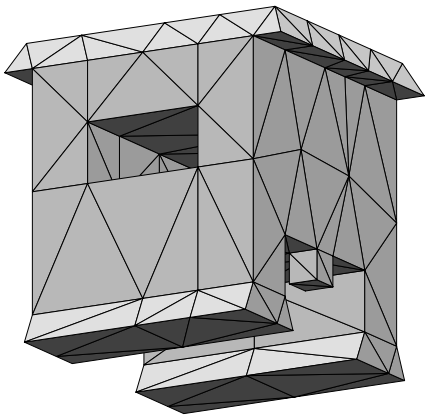
As Figure 7.17 shows, the algorithm performs much better in practice. The upper mesh is the initial tetrahedralization after all segments and facets are inserted and unwanted tetrahedra have been removed from the holes. (Some subsegments remain encroached because during the segment and facet recovery 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 $\text{lfs}_{\min} = \sqrt{5}$, 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 7.18 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.

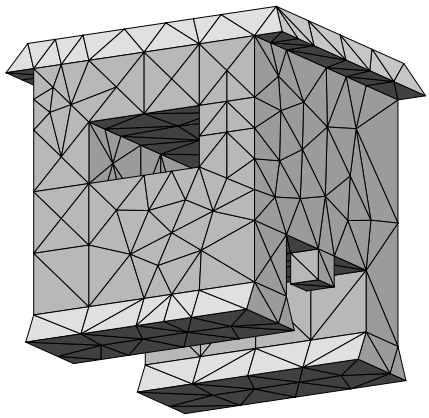
Unfortunately, the proof of good grading does not yield a size-optimality proof as it does in the two-dimensional case. Gary Miller and Dafna Talmor (private communication) have pointed out the counterexample depicted in Figure 7.19. Inside this PLC, two segments pass very close to each other without 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 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 [84, 85] demonstrate the size-optimality of their octree-based algorithms.



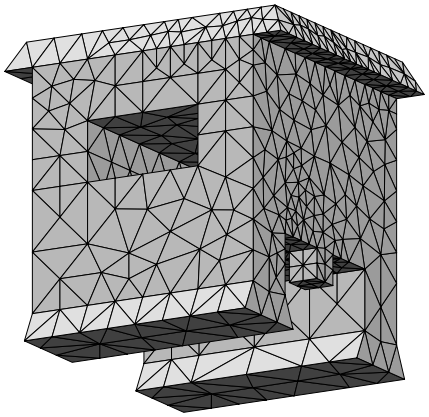
Initial tetrahedralization after segment and facet recovery. 71 vertices, 146 tetrahedra.



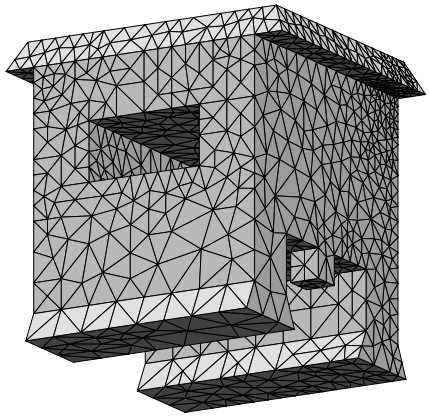
$B = 2.095, \theta_{\min} = 1.96°, \theta_{\max} = 176.02°,$
$h_{\min} = 1$, 143 vertices, 346 tetrahedra.



$B = 1.2, \theta_{\min} = 1.20°, \theta_{\max} = 178.01°,$
$h_{\min} = 0.743$, 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 7.17: 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 $B = 1.04$.

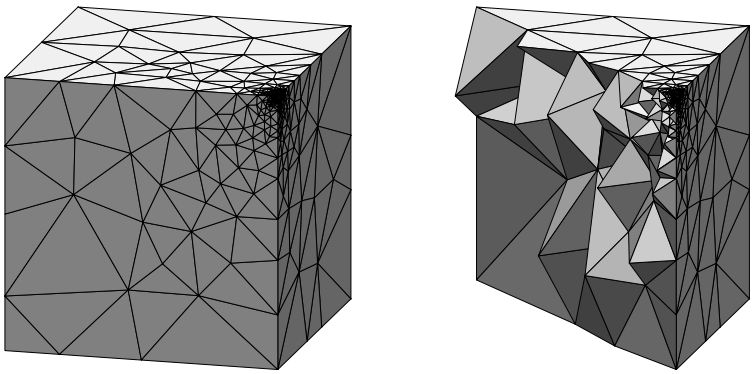


Figure 7.18: At left, a mesh of a truncated cube. At right, a cross-section through a diagonal of the top face.

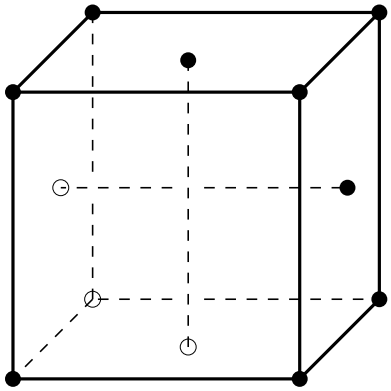


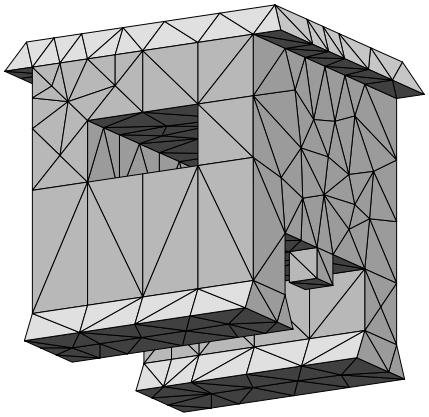
Figure 7.19: A counterexample demonstrating that the three-dimensional Delaunay refinement algorithm is not size-optimal.

7.3 Sliver Removal by Delaunay Refinement

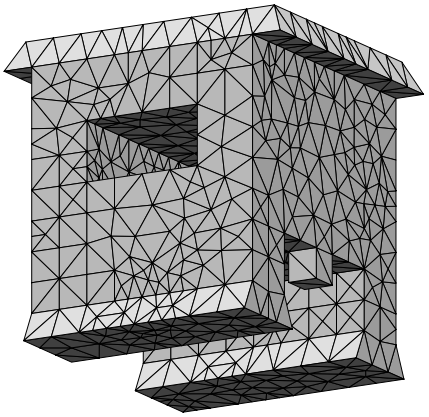
Although I have proven no theoretical guarantees about Delaunay refinement's ability to remove sliver tetrahedra, it is nonetheless natural to wonder whether Delaunay refinement might be effective in practice. If one inserts a vertex at the circumcenter of any tetrahedron with a small dihedral angle,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

As Figure 7.20 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.

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.



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



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

Figure 7.20: 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 7.17 on Page 123.

Chew [38] 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 inserted vertex can only take part in a sliver if it is positioned badly relative to a triangular face already in the mesh. Figure 7.21 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 7.21 (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 bound-

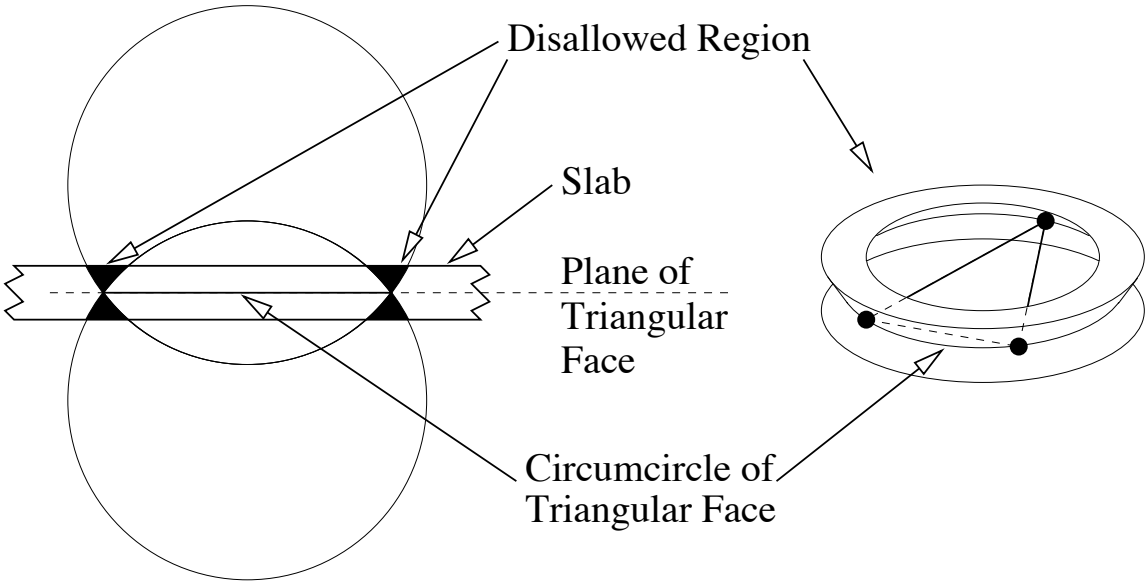


Figure 7.21: Left: A side view of the plane containing a triangular face. In conjunction with this face, a newly inserted vertex can form a sliver with both a bad dihedral angle and a good circumradius-to-shortest edge ratio only if it is inserted in the disallowed region (black). Right: An oblique view of the disallowed region of a triangular face.

ary of a mesh, especially near small angles. Mesh improvement techniques such as optimization-based smoothing and topological transformations can remove some of the imperfections that cannot be removed directly by Delaunay refinement.

Bibliography

[1] Nina Amenta, Marshall Bern, and David Eppstein. *Optimal Point Placement for Mesh Smoothing*. Proceedings of the Eighth Annual Symposium on Discrete Algorithms (New Orleans, Louisiana), pages 528–537. Association for Computing Machinery, January 1997.

[2] Nina Amenta, Sunghee Choi, and Günter Rote. *Incremental Constructions con BRIO*. Proceedings of the Nineteenth Annual Symposium on Computational Geometry (San Diego, California), pages 211–219. Association for Computing Machinery, June 2003.

[3] Marc Vigo Anglada. *An Improved Incremental Algorithm for Constructing Restricted Delaunay Triangulations*. Computers and Graphics **21**(2):215–223, March 1997.

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

[5] 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.

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

[7] Randolph E. Bank and L. Ridgway Scott. *On the Conditioning of Finite Element Equations with Highly Refined Meshes*. SIAM Journal on Numerical Analysis **26**(6):1383–1394, December 1989.

[8] 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.

[9] Marshall Bern, David Eppstein, and Jeff Erickson. *Flipping Cubical Meshes*. Engineering with Computers **18**(3):173–187, October 2002.

[10] 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.

[11] _____. *Provably Good Mesh Generation*. Journal of Computer and System Sciences **48**(3):384–409, June 1994.

[12] John Desmond Bernal and John Leslie Finney. *Random Close-Packed Hard-Sphere Model. II. Geometry of Random Packing of Hard Spheres*. Discussions of the Faraday Society **43**:62–69, 1967.

[13] Christopher J. Bishop. *Nonobtuse Triangulation of PSLGs*. Unpublished manuscript, 2011.

[14] Ted D. Blacker and Ray J. Meyers. *Seams and Wedges in Plastering: A 3-D Hexahedral Mesh Generation Algorithm*. Engineering with Computers **9**:83–93, 1993.

[15] Ted D. Blacker and Michael B. Stephenson. *Paving: A New Approach to Automated Quadrilateral Mesh Generation*. International Journal for Numerical Methods in Engineering **32**(4):811–847, September 1991.

[16] Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. *Compact Representations of Simplicial Meshes in Two and Three Dimensions*. International Journal of Computational Geometry and Applications **15**(1):3–24, February 2005.

[17] Guy E. Blelloch, Hal Burch, Karl Crary, Robert Harper, Gary L. Miller, and Noel J. Walkington. *Persistent Triangulations*. Journal of Functional Programming **11**(5):441–466, September 2001.

[18] Jean-Daniel Boissonnat, David Cohen-Steiner, Bernard Mourrain, Günter Rote, and Gert Vegter. *Meshing of Surfaces*. Effective Computational Geometry for Curves and Surfaces (Jean-Daniel Boissonnat and Monique Teillaud, editors), chapter 5, pages 181–229. Springer, 2006.

[19] Charles Boivin and Carl Ollivier-Gooch. *Guaranteed-Quality Triangular Mesh Generation for Domains with Curved Boundaries*. International Journal for Numerical Methods in Engineering **55**(10):1185–1213, 20 August 2002.

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

[21] Kevin Q. Brown. *Voronoi Diagrams from Convex Hulls*. Information Processing Letters **9**(5):223–228, December 1979.

[22] 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.

[23] Scott A. Canann, Michael Stephenson, and Ted Blacker. *Optismoothing: An Optimization-Driven Approach to Mesh Smoothing*. Finite Elements in Analysis and Design **13**:185–190, 1993.

[24] James C. Cavendish. *Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method*. International Journal for Numerical Methods in Engineering **8**(4):679–696, 1974.

[25] 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.

[26] Donald R. Chand and Sham S. Kapur. *An Algorithm for Convex Polytopes*. Journal of the Association for Computing Machinery **17**(1):78–86, January 1970.

[27] Bernard Chazelle. *Convex Partitions of Polyhedra: A Lower Bound and Worst-Case Optimal Algorithm*. SIAM Journal on Computing **13**(3):488–507, August 1984.

[28] Bernard Chazelle and Leonidas Palios. *Triangulating a Nonconvex Polytope*. Discrete & Computational Geometry **5**(1):505–526, December 1990.

[29] Siu-Wing Cheng, Tamal Krishna Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. *Sliver Exudation*. Journal of the Association for Computing Machinery **47**(5):883–904, September 2000.

[30] ———. *Sliver Exudation*. Proceedings of the Fifteenth Annual Symposium on Computational Geometry (Miami Beach, Florida), pages 1–13. Association for Computing Machinery, June 1999.

[31] Siu-Wing Cheng, Tamal Krishna Dey, and Edgar A. Ramos. *Delaunay Refinement for Piecewise Smooth Complexes*. Discrete & Computational Geometry **43**(1):121–166, 2010.

[32] Siu-Wing Cheng, Tamal Krishna Dey, Edgar A. Ramos, and Rephael Wenger. *Anisotropic Surface Meshing*. Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms (Miami, Florida), pages 202–211, January 2006.

[33] Siu-Wing Cheng and Sheung-Hung Poon. *Graded Conforming Delaunay Tetrahedralization with Bounded Radius-Edge Ratio*. Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms (Baltimore, Maryland), pages 295–304. Society for Industrial and Applied Mathematics, January 2003.

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

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

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

[37] ———. *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.

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

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

[40] David Cohen-Steiner, Éric Colin de Verdière, and Mariette Yvinec. *Conforming Delaunay Triangulations in 3D*. Proceedings of the Eighteenth Annual Symposium on Computational Geometry (Barcelona, Spain), pages 199–208. Association for Computing Machinery, June 2002.

[41] Richard Courant, Kurt Friedrichs, and Hans Lewy. *Über die Partiellen Differenzengleichungen der Mathematischen Physik*. Mathematische Annalen **100**:32–74, August 1928.

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

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

[44] Olivier Devillers. *On Deletion in Delaunay Triangulations*. Proceedings of the Fifteenth Annual Symposium on Computational Geometry (Miami Beach, Florida), pages 181–188. Association for Computing Machinery, June 1999.

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

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

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

[48] Herbert Edelsbrunner and Raimund Seidel. *Voronoi Diagrams and Arrangements*. Discrete & Computational Geometry **1**:25–44, 1986.

[49] Herbert Edelsbrunner and Tiow Seng Tan. *An Upper Bound for Conforming Delaunay Triangulations*. Discrete & Computational Geometry **10**(2):197–213, 1993.

[50] David A. Field. *Qualitative Measures for Initial Meshes*. International Journal for Numerical Methods in Engineering **47**:887–906, 2000.

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

[52] ———. *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.

[53] C. O. Frederick, Y. C. Wong, and F. W. Edge. *Two-Dimensional Automatic Mesh Generation for Structural Analysis*. International Journal for Numerical Methods in Engineering **2**:133–144, 1970.

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

[55] Lori A. Freitag and Carl Ollivier-Gooch. *Tetrahedral Mesh Improvement Using Swapping and Smoothing*. International Journal for Numerical Methods in Engineering **40**(21):3979–4002, November 1997.

[56] 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.

[57] Isaac Fried. *Condition of Finite Element Matrices Generated from Nonuniform Meshes*. AIAA Journal **10**(2):219–221, February 1972.

[58] John Alan George. *Computer Implementation of the Finite Element Method*. Ph.D. thesis, Stanford University, Stanford, California, March 1971. Technical report STAN-CS-71-208.

[59] Nicolas Grislain and Jonathan Richard Shewchuk. *The Strange Complexity of Constrained Delaunay Triangulation*. Proceedings of the Fifteenth Canadian Conference on Computational Geometry (Halifax, Nova Scotia, Canada), pages 89–93, August 2003.

[60] 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.

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

[62] François Hermeline. *Une Methode Automatique de Maillage en Dimension n*. Ph.D. thesis, Université Pierre et Marie Curie, 1980.

[63] _____. *Triangulation Automatique d'un Polyèdre en Dimension N*. RAIRO Analyse Numérique **16**(3):211–242, 1982.

[64] Antony Jameson, Timothy J. Baker, and Nigel P. Weatherill. *Calculation of Inviscid Transonic Flow over a Complete Aircraft*. Proceedings of the 24th AIAA Aerospace Sciences Meeting (Reno, Nevada), January 1986. AIAA paper 86-0103.

[65] Pierre Jamet. *Estimations d'Erreur pour des Éléments Finis Droits Presque Dégénérés*. RAIRO Analyse Numérique **10**(1):43–60, 1976.

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

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

[68] H. A. Kamel and K. Eisenstein. *Automatic Mesh Generation in Two- and Three-Dimensional Interconnected Domains*. Symposium on High Speed Computing of Elastic Structures (Liege, Belgium), 1970.

[69] Paul Kinney. *CleanUp: Improving Quadrilateral Finite Element Meshes*. Sixth International Meshing Roundtable (Park City, Utah), pages 449–461. Sandia National Laboratories, October 1997.

[70] Bryan Matthew Klingner and Jonathan Richard Shewchuk. *Aggressive Tetrahedral Mesh Improvement*. Proceedings of the 16th International Meshing Roundtable (Seattle, Washington), pages 3–23. Springer, October 2007.

[71] Michal Křížek. *On the Maximum Angle Condition for Linear Tetrahedral Elements*. SIAM Journal on Numerical Analysis **29**(2):513–520, April 1992.

[72] François Labelle and Jonathan Richard Shewchuk. *Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation*. Proceedings of the Nineteenth Annual Symposium on Computational Geometry (San Diego, California), pages 191–200. Association for Computing Machinery, June 2003.

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

[74] Der-Tsai Lee and Arthur K. Lin. *Generalized Delaunay Triangulations for Planar Graphs*. Discrete & Computational Geometry **1**:201–217, 1986.

[75] 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.

[76] S. H. Lo. *A New Mesh Generation Scheme for Arbitrary Planar Domains*. International Journal for Numerical Methods in Engineering **21**(8):1403–1426, August 1985.

[77] Rainald Löhner and Paresh Parikh. *Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method*. International Journal for Numerical Methods in Fluids **8**:1135–1149, 1988.

[78] David L. Marcum. *Unstructured Grid Generation Using Automatic Point Insertion and Local Reconnection*. Handbook of Grid Generation (Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors), chapter 18, pages 18.1–18.31. CRC Press, 1999.

[79] D. H. McLain. *Two Dimensional Interpolation from Random Data*. The Computer Journal **19**(2):178–181, May 1976.

[80] Elefterios A. Melissaratos. *L_p Optimal d Dimensional Triangulations for Piecewise Linear Interpolation: A New Result on Data Dependent Triangulations*. Technical Report RUU-CS-93-13, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, April 1993.

[81] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel J. 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.

[82] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel J. 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.

[83] 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.

[84] 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.

[85] _____. *Quality Mesh Generation in Higher Dimensions*. SIAM Journal on Computing **29**(4):1334–1370, 2000.

[86] D. E. Muller and Franco P. Preparata. *Finding the Intersection of Two Convex Polyhedra*. Theoretical Computer Science **7**:217–236, 1978.

[87] Michael Murphy, David M. Mount, and Carl W. Gable. *A Point-Placement Strategy for Conforming Delaunay Tetrahedralization*. Proceedings of the Eleventh Annual Symposium on Discrete Algorithms, pages 67–74. Association for Computing Machinery, January 2000.

[88] 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.

[89] Van Phai Nguyen. *Automatic Mesh Generation with Tetrahedral Elements*. International Journal for Numerical Methods in Engineering **18**:273–289, 1982.

[90] V. N. Parthasarathy, C. M. Graichen, and A. F. Hathaway. *A Comparison of Tetrahedron Quality Measures*. Finite Elements in Analysis and Design **15**(3):255–261, January 1994.

[91] V. N. Parthasarathy and Srinivas Kodiyalam. *A Constrained Optimization Approach to Finite Element Mesh Smoothing*. Finite Elements in Analysis and Design **9**(4):309–320, September 1991.

[92] Steven E. Pav and Noel J. Walkington. *Delaunay Refinement by Corner Lopping*. Proceedings of the 14th International Meshing Roundtable (San Diego, California), pages 165–181. Springer, September 2005.

[93] Jaime Peraire, Joaquim Peiró, L. Formaggia, Ken Morgan, and Olgierd C. Zienkiewicz. *Finite Element Euler Computations in Three Dimensions*. International Journal for Numerical Methods in Engineering **26**(10):2135–2159, October 1988.

[94] Jaime Peraire, Joaquim Peiró, and Ken Morgan. *Advancing Front Grid Generation*. Handbook of Grid Generation (Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors), chapter 17, pages 17.1–17.22. CRC Press, 1999.

[95] Jaime Peraire, M. Vahdati, Ken Morgan, and Olgierd C. Zienkiewicz. *Adaptive Remeshing for Compressible Flow Computations*. Journal of Computational Physics **72**(2):449–466, October 1987.

[96] V. T. Rajan. *Optimality of the Delaunay Triangulation in* $\mathbb{R}^d$. Proceedings of the Seventh Annual Symposium on Computational Geometry (North Conway, New Hampshire), pages 357–363, June 1991.

[97] Alexander Rand and Noel J. Walkington. *Collars and Intestines: Practical Conformal Delaunay Refinement*. Proceedings of the 18th International Meshing Roundtable (Salt Lake City, Utah), pages 481–497. Springer, October 2009.

[98] Shmuel Rippa. *Long and Thin Triangles Can Be Good for Linear Interpolation*. SIAM Journal on Numerical Analysis **29**(1):257–270, February 1992.

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

[100] 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.

[101] _____. *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.

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

[103] Jim Ruppert and Raimund Seidel. *On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra*. Discrete & Computational Geometry **7**(3):227–253, 1992.

[104] Edward A. Sadek. *A Scheme for the Automatic Generation of Triangular Finite Elements*. International Journal for Numerical Methods in Engineering **15**(12):1813–1822, December 1980.

[105] Robert Schneiders. *Quadrilateral and Hexahedral Element Meshes*. Handbook of Grid Generation (Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors), chapter 21, pages 21.1–21.27. CRC Press, 1999.

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

[107] Raimund Seidel. *Voronoi Diagrams in Higher Dimensions*. Diplomarbeit, Institut für Informationsverarbeitung, Technische Universität Graz, 1982.

[108] ———. *Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face*. Proceedings of the Eighteenth Annual ACM Symposium on the Theory of Computing, pages 404–413. Association for Computing Machinery, 1986.

[109] ———. *Constrained Delaunay Triangulations and Voronoi Diagrams with Obstacles*. 1978–1988 Ten Years IIG (H. S. Poingratz and W. Schinnerl, editors), pages 178–191. Institute for Information Processing, Graz University of Technology, 1988.

[110] ———. *Backwards Analysis of Randomized Geometric Algorithms*. New Trends in Discrete and Computational Geometry (János Pach, editor), Algorithms and Combinatorics, volume 10, pages 37–67. Springer-Verlag, Berlin, 1993.

[111] Michael Ian 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.

[112] Jonathan Richard Shewchuk. *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*. Applied Computational Geometry: Towards Geometric Engineering (Ming C. Lin and Dinesh Manocha, editors), Lecture Notes in Computer Science, volume 1148, pages 203–222. Springer-Verlag, Berlin, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[113] ———. *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997. Available as Technical Report CMU-CS-97-137.

[114] ———. *Tetrahedral Mesh Generation by Delaunay Refinement*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 86–95. Association for Computing Machinery, June 1998.

[115] ———. *Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations*. Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong), pages 350–359. Association for Computing Machinery, June 2000.

[116] ———. *What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures*. Eleventh International Meshing Roundtable (Ithaca, New York), pages 115–126. Sandia National Laboratories, September 2002.

[117] ———. *Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips*. Proceedings of the Nineteenth Annual Symposium on Computational Geometry (San Diego, California), pages 181–190. Association for Computing Machinery, June 2003.

[118] ———. *General-Dimensional Constrained Delaunay Triangulations and Constrained Regular Triangulations I: Combinatorial Properties*. Discrete & Computational Geometry **39**(1–3):580–637, March 2008.

[119] Matthew L. Staten, Steven J. Owen, and Ted D. Blacker. *Paving & Plastering: Progress Update*. Proceedings of the 15th International Meshing Roundtable (Birmingham, Alabama), pages 469–486. Springer, September 2006.

[120] 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.

[121] Garret Swart. *Finding the Convex Hull Facet by Facet*. Journal of Algorithms **6**(1):17–48, March 1985.

[122] J. L. Synge. *The Hypercircle in Mathematical Physics*. Cambridge University Press, New York, 1957.

[123] Masaharu Tanemura, Tohru Ogawa, and Naofumi Ogita. *A New Algorithm for Three-Dimensional Voronoi Tessellation*. Journal of Computational Physics **51**(2):191–207, August 1983.

[124] Joe F. Thompson. *The National Grid Project*. Computer Systems in Engineering **3**(1–4):393–399, 1992.

[125] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors. *Handbook of Grid Generation*. CRC Press, 1999.

[126] Joe F. Thompson and Nigel P. Weatherill. *Aspects of Numerical Grid Generation: Current Science and Art*. Eleventh AIAA Applied Aerodynamics Conference, pages 1029–1070, 1993. AIAA paper 93-3593-CP.

[127] Shayne Waldron. *The Error in Linear Interpolation at the Vertices of a Simplex*. SIAM Journal on Numerical Analysis **35**(3):1191–1200, 1998.

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

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

[130] Mark 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.

[131] _____. *Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique*. International Journal for Numerical Methods in Engineering **20**(11):1965–1990, November 1984.