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 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 [215], who commanded a multi-institutional mesh generation effort called the National Grid Project, 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

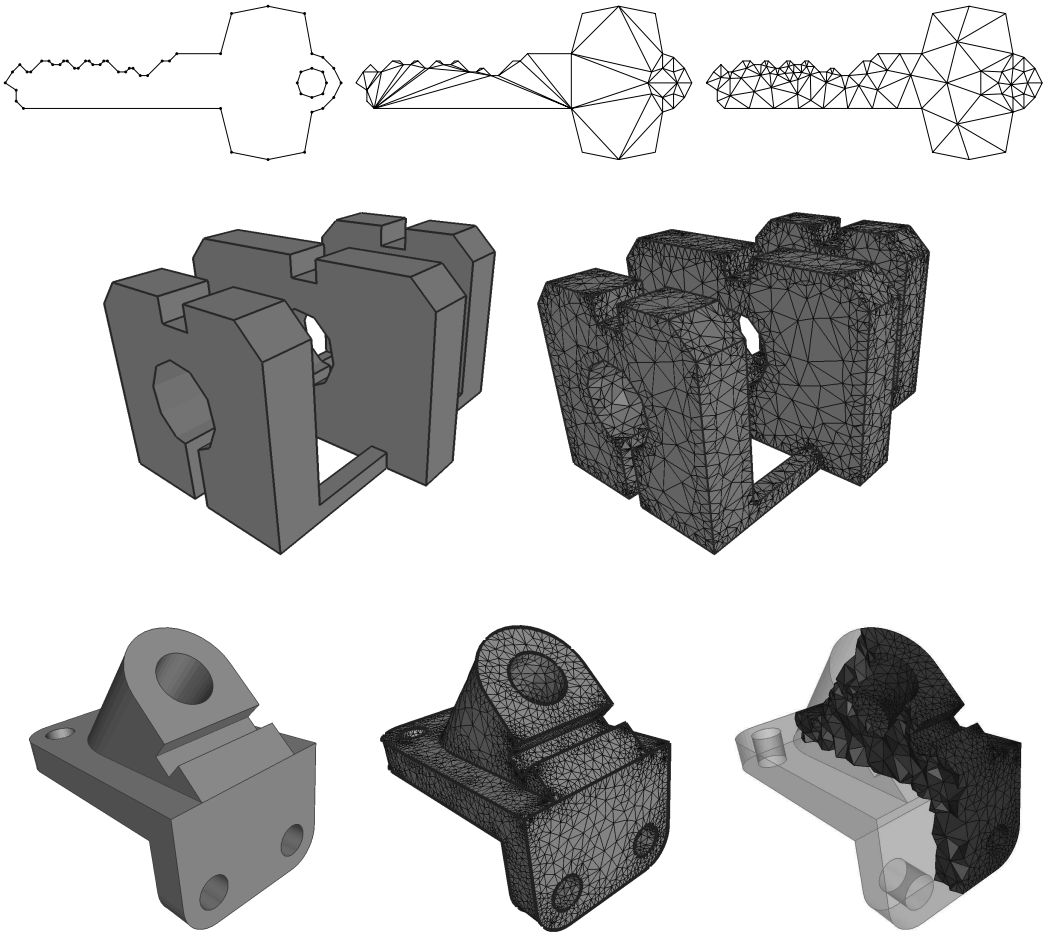

Figure 1.1: Finite element meshes of a polygonal, a polyhedral, and a curved domain. One mesh of the key has poorly shaped triangles and no Steiner points; the other has Steiner points and all angles between 30° and 120°. The cutaway view at lower right reveals some of the tetrahedral elements inside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

shapes" typically include elements that are nearly equilateral and equiangular, and typically exclude elements that are long and thin, for example, 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.

By our 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. It is the subject of this book.

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. By economic measures, the videogame and motion picture industries probably now exceed the finite element industries as users of meshes.

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

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. This book discusses 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, 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. This book discusses only *unstructured meshes*, which entail explicitly storing each vertex's neighboring vertices or elements. All the meshes in Figure 1.1 are unstructured. Structured

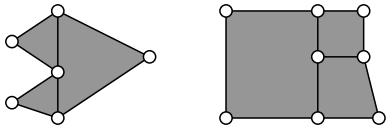

Figure 1.2: Nonconforming elements.

meshes 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 constituting 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 1.5. The mesh generation problem becomes superficially easier if we permit what finite element practitioners call *nonconforming elements* like those illustrated in Figure 1.2, where an element shares an edge with two other elements each abutting half of that edge. But nonconforming elements rarely alleviate the underlying numerical problems and can be computationally expensive when they do, so they find limited use in unstructured meshes.

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 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 by splines, implicit equations, 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 2.10.1 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, such as 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 2.10.1 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 the shape, the finite element solutions, and the 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 this book, the elements are always linear triangles and tetrahedra.

We study algorithms for several types of curved domain: in Chapters 12–14, we study how to mesh smooth surfaces with triangles and how to mesh volumes bounded by smooth surfaces with tetrahedra. Then we mesh more general domains like that at lower left in Figure 1.1, specified by geometric structures called *piecewise smooth complexes*. These complexes are composed of smoothly curved patches and ridges, but patches can meet non-smoothly at ridges and vertices, and internal boundaries are permitted. Piecewise smooth complexes and their triangulations are defined in Chapter 15.

In this book, we assume that we have mathematically exact representations of domains and ignore the difficulties of numerical robustness and real-world CAD models, but we acknowledge that they are important issues.

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. Let us consider these four constraints in turn.

The first constraint forbids large angles, including large plane angles in triangles and large dihedral angles (defined in Section 1.7) in tetrahedra. Most applications of triangulations use them to interpolate a multivariate function whose true value might or might not be

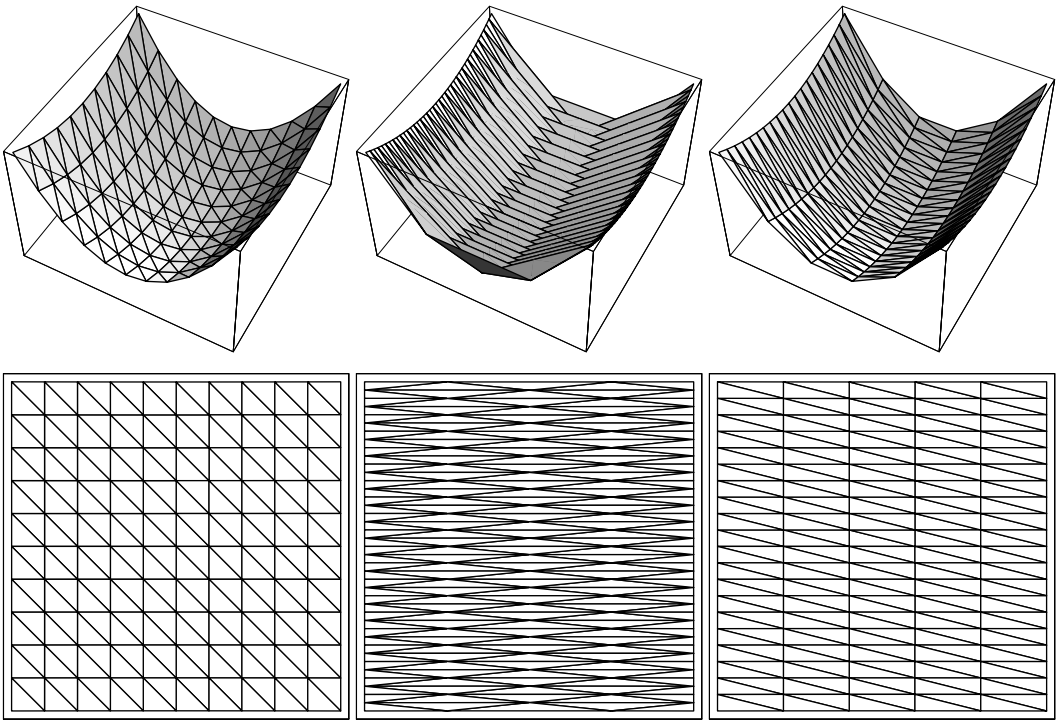

Figure 1.3: An illustration of how large angles, but not small angles, can ruin the interpolated gradients. Each triangulation uses 200 triangles to render a paraboloid.

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 toğ 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 interpolating function and the true function, and the difference between the gradient of the interpolating 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 of the elements: it can grow arbitrarily large as an element's largest angle approaches 180°. In Figure 1.3, 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 high-quality 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 1.4 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

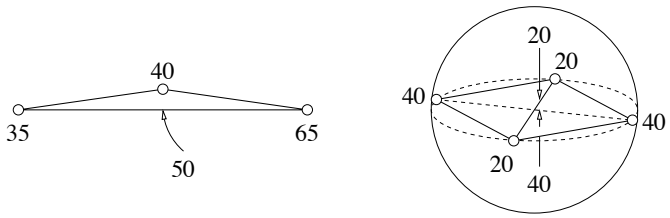

Figure 1.4: As the large angle of the triangle approaches 180°, or the sliver tetrahedron becomes arbitrarily flat, the magnitude of the interpolated gradient becomes arbitrarily large.

true gradient. The same effect is seen between two edges of a sliver tetrahedron that pass near each other, also illustrated in Figure 1.4.

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. We derive bounds on this approximation in Section 12.7.2.

For tetrahedral elements, usually it is their largest dihedral angles that matter most. Nonconvex quadrilateral and hexahedral elements, with interior angles exceeding 180°, sabotage interpolation and the finite element method.

The second constraint on mesh generators is that many applications forbid small angles, although fewer than those that forbid large angles. If the 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. Fortunately, most linear equation solvers cope well with a few bad eigenvalues.

The third constraint on mesh generators 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.

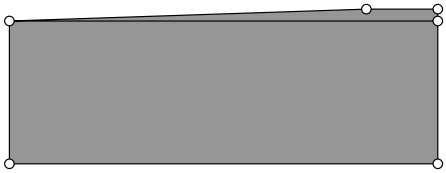

Figure 1.5: A mesh of this domain must have a *new* small angle.

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*, which implies that the computational time step must be small enough that a half-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."

These four constraints can be difficult to reconcile. 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 1.5 depicts a domain composed of two polygons glued together that, surprisingly, provably has no mesh whose *new* angles are all over 30°. 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. Chapters 9 and 15 present algorithms for meshing linear and curved domains with these difficulties.

1.2 Delaunay triangulations and Delaunay refinement algorithms

This book is 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

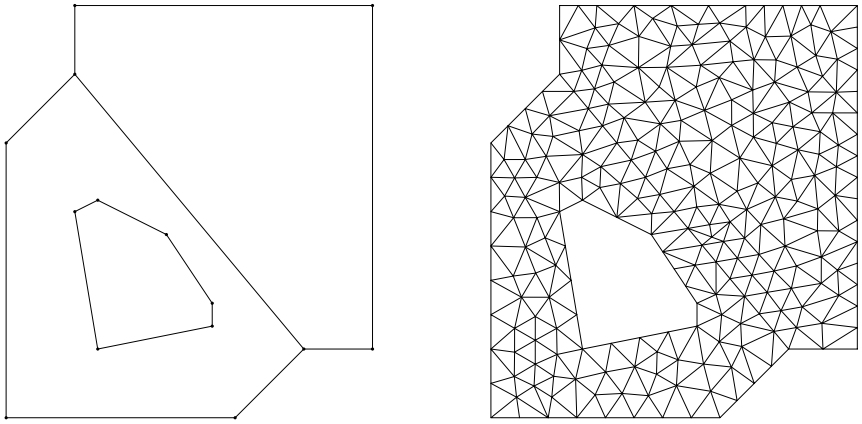

Figure 1.6: A mesh generated by Chew's first Delaunay refinement algorithm.

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 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.

As a preview, consider the first provably good Delaunay refinement algorithm, invented by Paul Chew, which takes as input a polygonal domain and generates a triangular mesh whose angles are all between 30° and 120°. (The input polygon may not have an angle less than 30°.) Chew begins by subdividing the polygon's edges so that all the edge lengths are in a range $[h, \sqrt{3}h]$, where h is chosen small enough that such a subdivision exists with no two edge endpoints closer to each other than h. Next, he constructs the *constrained Delaunay triangulation* of the subdivision, defined in Section 2.10.2. Finally, he refines the triangulation by repeatedly choosing a triangle whose circumscribing disk has radius greater than h and inserting a new vertex at the center of the circumscribing disk, until no such triangle survives. The vertex is inserted by an algorithm that maintains the constrained Delaunay property of the mesh and thereby eliminates the skinny triangle. Chew's algorithm is quite useful in practice, but it generates only meshes with uniformly sized triangles, as Figure 1.6 illustrates, and not graded meshes.

The first third of this book lays out the mathematical underpinnings of Delaunay triangulations and the most practical algorithms for constructing them. The second third of this book describes Delaunay refinement algorithms for domains expressed as *piecewise linear complexes*, which generalize polygons and polyhedra to support internal boundaries. The final third of this book describes Delaunay refinement algorithms for curved domains—specifically, smooth surfaces, volumes bounded by smooth surfaces, and piecewise smooth

domains that have curved ridges and patches and are represented by *piecewise smooth complexes*.

1.3 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, 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 this book. 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*. 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 which we discuss in Section 2.10.2. 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 and then triangulated them in two separate stages. For instance, Frederick, Wong, and Edge 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 took this step in his 1971 doctoral dissertation, but it was forgotten and reinvented several times, and finally became widespread around 1988.

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

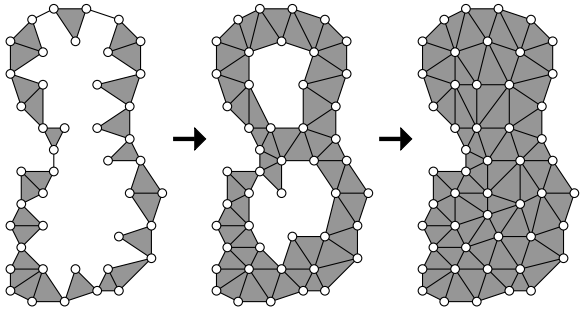

Figure 1.7: Advancing front mesh generation.

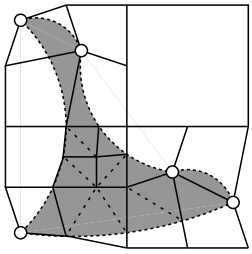

Figure 1.8: A quadtree mesh.

mesh generators, like the early advancing front methods, created vertices and triangulated them in two separate stages. The era of modern meshing began in 1987 with the insight, courtesy of William Frey, 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.

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 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 connectivity (topology) of the mesh. *Topological transformations* are operations that change the mesh connectivity by removing elements from a mesh and replacing them with a different configuration of elements occupying the same space.

The simplest topological transformation is the *edge flip* in a triangular mesh, which replaces two adjacent triangles with two different triangles. There are analogous transformations for tetrahedra, quadrilaterals, and hexahedra. Simple transformations called *bistellar flips* that act on triangles and tetrahedra are discussed in Section 4.4. In Chapter 10, we describe a provably good algorithm called *sliver exudation* that uses bistellar flips to improve Delaunay meshes.

The story of provably good mesh generation is an interplay of ideas between Delaunay methods and methods based on grids, quadtrees, and octrees. It began in 1988, when Brenda Baker, Eric Grosse, and Conor Rafferty gave an algorithm to triangulate a polygon so that all the new angles in the mesh are between 14° and 90°. They overlay the polygon with a fine square grid, create new vertices at some grid points and at some intersections between grid lines and the polygon boundary, and triangulate them with a complicated case analysis.

The following year, Paul Chew gave a more practical algorithm, which we have described in Section 1.2, that uses Delaunay refinement to guarantee angles between 30° and 120°. In 1992, Dey, Bajaj, and Sugihara generalized Chew's algorithm to generate tetrahedral meshes of convex polyhedral domains. 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 1.9 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 1990 quadtree algorithm of Marshall Bern, David Eppstein, and John Gilbert 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.

In 1992, Scott Mitchell and Stephen Vavasis developed an octree algorithm that offers guarantees on dihedral angles, grading, and size optimality. The bounds are not strong

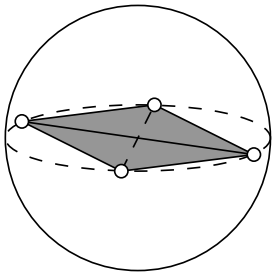

Figure 1.9: The mesh generator's nemesis: a sliver tetrahedron.

enough to be meaningful in practice and are not explicitly stated. Nevertheless, the papers by Bern et al. and Mitchell and Vavasis decidedly broadened the ambitions of provably good meshing.

A groundbreaking 1992 paper by Jim Ruppert 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.

Domains with curved geometries, represented by splines, isosurfaces, or other surface representations, increase the challenge appreciably. Most early algorithms for meshing surfaces work in the parametric space of a spline, but most grid and octree methods work directly in three-dimensional space, as do a few advancing front methods. A 1993 paper by Paul Chew partly generalizes Delaunay triangulations to curved surfaces. He proposes an algorithm that takes a triangulation of a spline patch, flips its edges to make it Delaunay, and refines it. If the initial mesh is fine enough, the triangles in the final mesh are guaranteed to have high quality.

These early works in guaranteed-quality mesh generation launched a rapid escalation of research on the subject, in which we were fortunate to participate.

1.4 A personal history of working in mesh generation

When we came to study mesh generation in the 1990s, we were drawn by the unusually strong way it combines theory and practice, complexity and elegance, and combinatorial and numerical computing. There is a strong tradition of practical meshing algorithms in scientific computing and computer graphics, yet their difficulty and fragility bring up fundamental theoretical questions in approximation theory, surface sampling, topology, algorithm design, numerical computing, and the structure of Delaunay triangulations and their weighted and constrained relatives. Mesh generation demands an understanding of both combinatorial and numerical algorithms, because meshing is geometric and most meshes are used by numerical applications. Lastly, meshes and their applications are as attractive to the eye as their mathematics are to the soul.

Galvanized by the publication of Ruppert's algorithm, Jonathan generalized it to three dimensions in 1997. The tetrahedral Delaunay refinement algorithm described in Chapter 8 accepts nonconvex domains with internal boundaries and offers guaranteed good grading.

However, it is not guaranteed to eliminate slivers, which implies (for technical reasons) that it cannot guarantee size optimality.

It soon became apparent that there are two serious difficulties in developing a truly satisfying meshing algorithm for polyhedral domains. First, domain faces that meet at small angles are particularly difficult to mesh, especially if many edges and faces, including internal boundaries, converge at a point. Second, although Delaunay refinement algorithms naturally eliminate most types of bad tetrahedra, they cannot guarantee that there will be no slivers, and even successful attempts to eliminate slivers in practice tend to overrefine the mesh. Researchers have made progress on both problems, but they are still areas of active research.

It is sometimes impossible to place high-quality tetrahedra at the apex of a small domain angle, so a mesh generation algorithm must know when and where to relax its guarantees on tetrahedron quality. In Chapter 9, we present a new algorithm that uses a variant of the Delaunay triangulation called a *weighted* Delaunay triangulation to help enforce domain conformity near small angles. The algorithm includes contributions from all three of us and several other collaborators, as we have collectively worked on this problem for over a decade.

In 1999, Siu-Wing and Tamal participated in the development of a provably good technique called *sliver exudation* for removing the worst slivers from a Delaunay mesh. Like our method for treating small domain angles, sliver exudation uses a weighted Delaunay triangulation; it removes slivers by shifting the weights of the vertices. We describe this technique in Chapter 10, and how to combine it with Delaunay refinement in Chapter 11. Since the original paper, Jonathan has joined the collaboration and together we have tightened the analysis considerably.

Surface meshing has been a particularly absorbing and rewarding research topic for us. It has compelled researchers to bring topology and approximation theory into mesh generation to help prove that certain meshes are topologically and geometrically accurate representations of curved domains. In 1997, Herbert Edelsbrunner and Nimish Shah took a large step forward by introducing the *restricted Delaunay triangulation*, a subcomplex of the three-dimensional Delaunay triangulation that serves as a surface mesh under the right conditions. Specifically, they prove a result known as the Topological Ball Theorem, which states that if the intersection of each face of a Voronoi diagram with a surface is a topological ball of the right dimension, then the restricted Delaunay triangulation is topologically equivalent to the surface. We define restricted Delaunay triangulations in Section 13.1 and state the Topological Ball Theorem in Section 13.2.

Provably good surface meshing draws on ideas in sampling theory originally developed for the problem of reconstructing the shape of a three-dimensional object from a finite set of points sampled from its surface by a laser scanner or stereo photography. In a seminal work from 1999, Nina Amenta and Marshall Bern use sampling theory and the Topological Ball Theorem to show that if a smooth surface is sampled sufficiently densely, the Delaunay tetrahedralization of the sample points includes a subset of triangles that accurately reconstruct the surface, by both topological and geometric criteria.

The recognition of these remarkable connections prompted us and other researchers to develop surface meshing algorithms with topological and geometric guarantees. Tamal collaborated with Ho-Lun Cheng, Herbert Edelsbrunner, and John Sullivan in 2001 to develop

a Delaunay refinement algorithm that chooses sample points and computes topologically correct triangular meshes for a class of smooth surfaces called *skin surfaces*. This work includes further developments in sampling theory that suggest how to choose vertices to ensure that the preconditions of the Topological Ball Theorem hold for more general classes of surfaces. Independently in 2003, Jean-Daniel Boissonnat and Steve Oudot developed a similar sampling theory and a simple Delaunay refinement algorithm for a more general class of smooth surfaces. We devote Chapters 12 and 13 to developing an updated sampling theory for smooth surfaces and restricted Delaunay triangulations. In Chapter 14, we study mesh generation algorithms that depend on this theory, including several algorithms for generating a triangular mesh of a smooth surface, and an algorithm for generating a tetrahedral mesh of a volume bounded by a smooth surface.

Meshing is yet more difficult for curved domains that are only piecewise smooth, with smooth surface patches meeting along smoothly curved ridges. With Edgar Ramos, Siu-Wing and Tamal introduced the first provably good algorithm for such domains in 2007. It uses a weighted Delaunay triangulation to enforce domain conformity at the corners and creases (ridges) of the domain, and motivates the development of additional sampling theory to ensure the topological correctness of the mesh. In the years following, this algorithm was made more practical with additional contributions from Josh Levine. Our final chapter describes a considerably updated and improved version of this algorithm.

As part of our research, we have developed several mesh generation packages that are publicly available on the web. Most of the images of meshes in this book were generated by these programs. Jonathan Shewchuk's program TRIANGLE[1] robustly constructs constrained Delaunay triangulations and high-quality triangular meshes in the plane, using Ruppert's algorithm to generate the latter. In 2003, TRIANGLE received the James Hardy Wilkinson Prize in Numerical Software. Bryan Klingner and Jonathan Shewchuk also offer a tetrahedral mesh improvement program STELLAR[2] that employs algorithms not discussed in this book (as they do not use Delaunay triangulations).

In collaboration with Edgar Ramos and Tathagata Ray, Siu-Wing and Tamal developed an algorithm for generating tetrahedral meshes of polyhedral domains with small angles and another algorithm for remeshing polygonal surfaces. Tathagata Ray implemented these two algorithms and released the programs QUALMESH[3] and SURFREMESH[4]. SURFREMESH is a precursor of the more practical algorithm DELSURF we describe in Chapter 14. Together with Josh Levine, Tamal designed an algorithm for generating triangular and tetrahedral meshes of piecewise smooth complexes. Josh Levine implemented the algorithm and released the program DELPSC[5], which is a precursor of the algorithm we describe in Chapter 15. We have taken the liberty of including illustrations generated by these programs throughout the book as prototypes, even though we have subsequently improved many of the algorithms.

[1] http://www.cs.cmu.edu/~quake/triangle.html
[2] http://www.cs.berkeley.edu/~jrs/stellar
[3] http://www.cse.ohio-state.edu/~tamaldey/qualmesh.html
[4] http://www.cse.ohio-state.edu/~tamaldey/surfremesh.html
[5] http://www.cse.ohio-state.edu/~tamaldey/delpsc.html

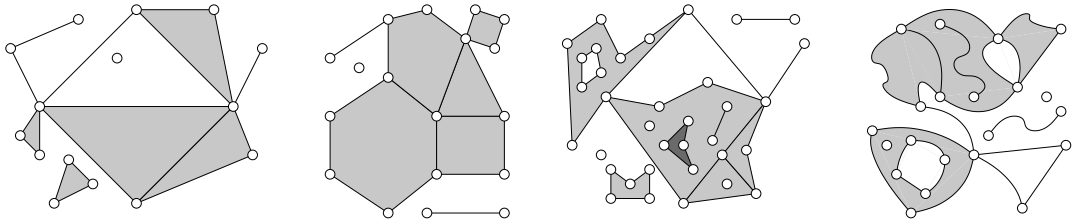

Figure 1.10: From left to right, a simplicial complex, a polyhedral complex, a piecewise linear complex, and a piecewise smooth complex. The shaded areas are triangles, convex polygons, linear 2-cells, and smooth 2-cells, respectively. In the piecewise linear complex, observe that several linear cells have holes, one of which is filled by another linear cell (darkly shaded).

1.5 Simplices, complexes, and polyhedra

Tetrahedra, triangles, edges, and vertices are instances of *simplices*. In this book, we represent meshes and the domains we wish to mesh as *complexes*. There are several different types of complexes, illustrated in Figure 1.10, which all share two common properties. First, a complex is a set that contains not only volumes such as tetrahedra, but also the facets, 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. All the mesh generation algorithms in this book 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 we study in this book are the famous *Voronoi diagram*, defined in Section 7.1, and the *Delaunay subdivision*, defined in Section 2.2.

We use two other kinds of complexes to specify domains to be triangulated. *Piecewise linear complexes*, defined in Sections 2.10.1 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*, defined in Section 15.1, 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 1.1 (affine hull; flat)**.** Let $X = \{x_1, x_2, \ldots, x_k\}$ be a set of points in $\mathbb{R}^d$. An *affine combination* of the points in X is a point p that 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 $\operatorname{aff} X$, is the set of all affine combinations of

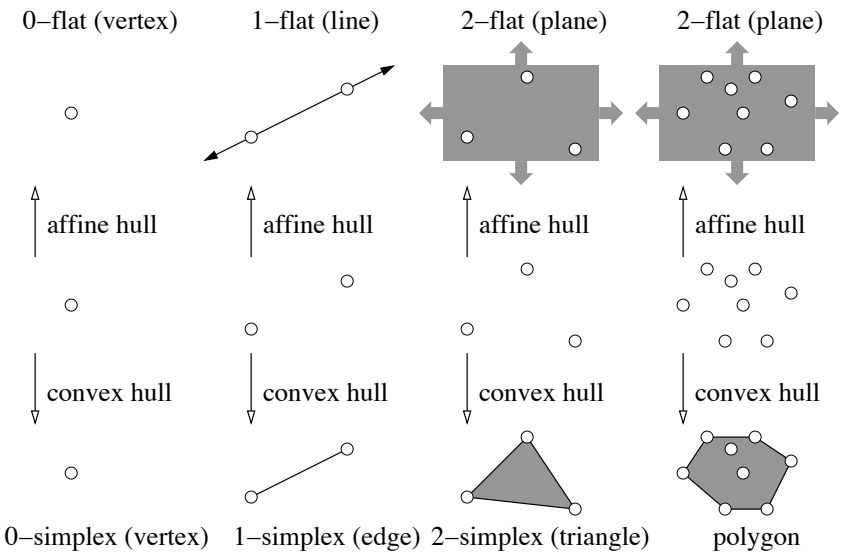

Figure 1.11: Examples of affine hulls and convex hulls in the plane.

points in X, as illustrated in Figure 1.11. A *k-flat*, also known as an *affine subspace*, is the affine hull of $k + 1$ 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 1.2 (convex hull)**.** A *convex combination* of the points in X is a point that 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 $\operatorname{conv} X$, is the set of all convex combinations of points in X, as illustrated in Figure 1.11. An alternative definition is that $\operatorname{conv} X$ is the most exclusive convex point set such that $X \subseteq \operatorname{conv} X$.

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 1.3 (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 *face* of τ is a simplex that is the convex hull of a nonempty subset of X. Faces of τ come in all dimensions from zero[6] (τ's vertices) to k; τ is a face of τ. A *proper face* of τ is a simplex that 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.

[6]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. We make no use of this convention.

Definition 1.4 (simplicial complex)**.** A *simplicial complex* $\mathcal{T}$, also known as a *triangulation*, is a set containing finitely[7] many simplices that satisfies the following two restrictions.

- $\mathcal{T}$ contains every face of every simplex in $\mathcal{T}$.
- 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 the definition of a face of a polyhedron below is consistent with the definition of a face of a simplex above.

Definition 1.5 (convex polyhedron)**.** A *convex polyhedron* is the convex hull of a finite point set. A convex 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 *convex 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 1.6 (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.

To support Voronoi diagrams, we will later extend Definition 1.5 to permit polyhedra that are *unbounded*—that is, infinitely large. Specifically, Section 7.1 redefines convex polyhedra as intersections of halfspaces instead of convex hulls of points.

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

Definition 1.7 (linear cell)**.** A *linear k-cell* is the pointwise 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*.

[7] Topologists usually define complexes so they have countable cardinality. We 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°.

Thus, we can build nonconvex polyhedra by uniting convex ones. 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, we will forbid disconnected linear 1-cells in our complexes; i.e. the only linear 1-cells we use 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 1.2 and 1.10. A linear cell's faces are not an intrinsic property of the linear cell alone, but depend on the complex that contains it. We defer the details to Section 2.10.1, where we define piecewise linear complexes.

Piecewise smooth complexes are sets of cells we call *smooth cells*, which are similar to linear cells except that they are not linear, but are smooth manifolds. See Chapter 15 for details.

Two cells in a complex are said to *adjoin* each other if they intersect each other, which implies that they have a face in common or one is a face of the other. Two cells that do not adjoin each other are *disjoint*.

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 1.8 (underlying space)**.** The *underlying space* of a complex $\mathcal{P}$, denoted $|\mathcal{P}|$, is the pointwise 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 1.9 (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 $\mathcal{P}$.

1.6 Metric space topology

This section introduces basic notions from point set topology that underlie triangulations and other complexes. These notions are prerequisites for more sophisticated topological ideas—manifolds, homeomorphism, and isotopy—introduced in Chapter 12 to study algorithms for meshing domains with curved boundaries. 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, and isotopy offers a rigorous way to state that the domain can be deformed into the shape of the linear mesh without ever colliding with itself.

Topology begins with a set $\mathbb{T}$ of points—perhaps the points constituting 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 inner product* of two points $p, q \in \mathbb{R}^d$ is $\langle p, q\rangle = \sum_{i=1}^{d} p_i q_i$. The *Euclidean norm* of a point $p \in \mathbb{R}^d$ is $\|p\| = \langle p, p\rangle^{1/2} = (\sum_{i=1}^{d} p_i^2)^{1/2}$, and the *Euclidean distance* between two points $p, q \in \mathbb{R}^d$ is $d(p, q) = \|p - q\| = (\sum_{i=1}^{d} (p_i - q_i)^2)^{1/2}$. We 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 1.10 (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 $d(p, q) < \epsilon$.

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 1.11 (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 1.12. If no such coloring exists, Q is *connected*, like the point set at right in Figure 1.12.

In this book, we 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 1.13. The idea can be formally extended to any point set.

Definition 1.12 (closure; closed; open)**.** The *closure* of a point set $Q \subseteq \mathbb{T}$, denoted $\mathrm{Cl}\, Q$, is the set containing every point in Q and every limit point of Q. A point set Q is *closed* if $Q = \mathrm{Cl}\, Q$, 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. $\mathbb{T} \setminus Q = \mathrm{Cl}\,(\mathbb{T} \setminus Q)$.

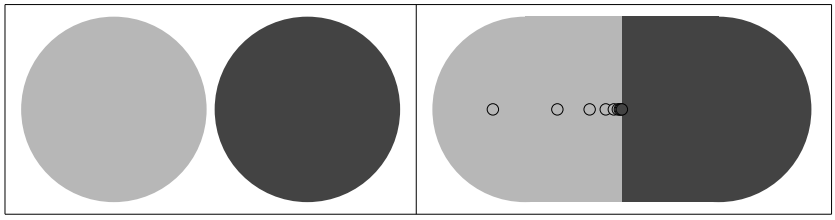

Figure 1.12: The disconnected point set at left can be partitioned into two connected subsets, which are shaded differently here. The point set at right is connected. The dark point at its center is a limit point of the lightly shaded points.

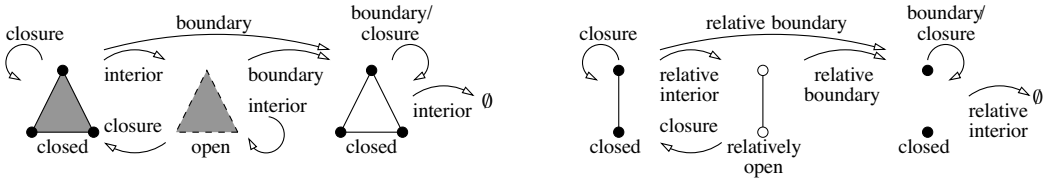

Figure 1.13: Closed, open, and relatively open point sets in the plane. Dashed edges and open circles indicate points missing from the point set.

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 $\mathrm{Cl}\,(0, 1) = [0, 1] = \mathrm{Cl}\,[0, 1]$. 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 missing the points on its edges, and therefore is open in the two-dimensional metric space $\mathrm{aff}\,\tau$, 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 we 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 we sometimes follow this convention in this book. Particularly useful is the concept of an "open edge," an edge that is missing its endpoints, illustrated in Figure 1.13.

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 1.13 (boundary; interior)**.** The *boundary* of a point set Q in a metric space $\mathbb{T}$, denoted $\mathrm{Bd}\,Q$, is the intersection of the closures of Q and its complement; i.e. $\mathrm{Bd}\,Q = \mathrm{Cl}\,Q \cap \mathrm{Cl}\,(\mathbb{T} \setminus Q)$. The *interior* of Q, denoted $\mathrm{Int}\,Q$, is $Q \setminus \mathrm{Bd}\,Q = Q \setminus \mathrm{Cl}\,(\mathbb{T} \setminus Q)$.

For example, $\mathrm{Bd}\,[0,1] = \{0,1\} = \mathrm{Bd}\,(0,1)$ and $\mathrm{Int}\,[0,1] = (0,1) = \mathrm{Int}\,(0,1)$. 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 1.13. 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. Hence the following terms.

Definition 1.14 (relative boundary; relative interior)**.** The *relative boundary* of a convex polyhedron $C \subset \mathbb{R}^d$ is its boundary with respect to the metric space of its affine hull—that is, $\mathrm{Cl}\,C \cap \mathrm{Cl}\,((\mathrm{aff}\,C) \setminus C)$. The *relative interior* of C is C minus its relative boundary.

Again, we often abuse terminology by writing *boundary* for relative boundary and *interior* for relative interior. The same subtlety arises with curved ridges and surface patches, but these have fundamentally different definitions of *boundary* and *interior* which we give in Section 12.3.

Definition 1.15 (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.

As we have defined them, simplices and polyhedra are bounded, but in Section 7.1 we will see how to define unbounded polyhedra, which arise in Voronoi diagrams. Besides simplices and polyhedra, the point sets we use most in this book are balls.

Definition 1.16 (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 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. Section 12.1 discusses *topological spaces*, a mathematical abstraction for representing the topology of a point set while excluding all information that is not topologically essential. In this book, all our topological spaces have metrics.

1.7 How to measure an element

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

Definition 1.17 (circumball)**.** Let τ be a simplex embedded in $\mathbb{R}^d$. A *circumball*, or *circumscribing ball*, of τ is a d-ball whose boundary passes through every vertex of τ, illustrated in Figure 1.14. Its boundary, a $(d-1)$-sphere, is called a *circumsphere*, or *circumscribing sphere*, of τ. A *closed circumball* includes its boundary—the circumsphere—and an

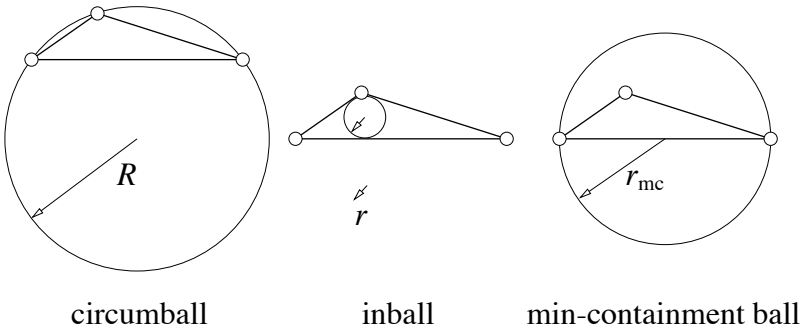

Figure 1.14: Three balls associated with a triangle.

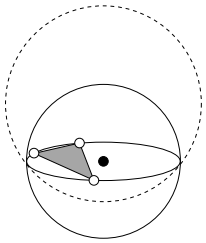

Figure 1.15: A triangle, two circumballs of the triangle of which the smaller (solid) is the triangle's diametric ball, the triangle's circumdisk (the equatorial cross-section of the diametric ball), and the triangle's circumcenter.

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. We 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 circumdisk of τ, whose boundary passes through τ's three vertices, but τ has infinitely many circumballs, and the intersection of any of those circumballs with τ's affine hull is τ's circumdisk. The smallest of these circumballs is special, because its center lies on τ's affine hull, it has the same radius as τ's circumdisk, and τ's circumdisk is its equatorial cross-section. We call τ's smallest circumball, illustrated in Figure 1.15, its *diametric ball*.

Definition 1.18 (diametric ball; circumcenter)**.** The *diametric ball* of a simplex τ is the circumball of τ with the smallest radius. The *circumcenter* of τ is the point at the center of τ's diametric ball, which always lies on aff τ. The *circumradius* of τ is the radius of τ's diametric ball.

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 balls associated with simplicial elements are the inball and the min-containment ball, both illustrated in Figure 1.14.

Definition 1.19 (inball)**.** 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 *incenter* of τ is the point at the center of B, and the *inradius* of τ is the radius of B.

Definition 1.20 (min-containment ball)**.** 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 of τ is always a diametric ball of a face of τ; that face could be of any dimension from an edge up to τ itself.

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 ball 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 balls. Details appear in Section 4.3.

A *quality measure* is a map 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. We denote the *plane angle* between two vectors $\mathbf{u}$ and $\mathbf{v}$ as

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

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

For a pair of intersecting lines or line segments ℓ_1 and ℓ_2, we generally measure the acute angle between them, denoted $\angle_a(\ell_1, \ell_2)$. When we replace ℓ_1 or ℓ_2 with a vector $\mathbf{v}$, the affine hull of $\mathbf{v}$ is implied; $\angle_a(\mathbf{u}, \mathbf{v})$ denotes the acute angle between the affine hulls of $\mathbf{u}$ and $\mathbf{v}$. Thus, $\angle_a$ disregards the vector orientation whereas $\angle$ does not. These angles satisfy a *triangle inequality*

$$\angle_a(\ell_1, \ell_2) \leq \angle_a(\ell_1, \ell_3) + \angle_a(\ell_3, \ell_2).$$

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 wyz and 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 wyz and 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 1.16: needles, which have one edge much shorter than the others, and caps, which have an angle near 180° and a large circumdisk. Figure 1.17 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

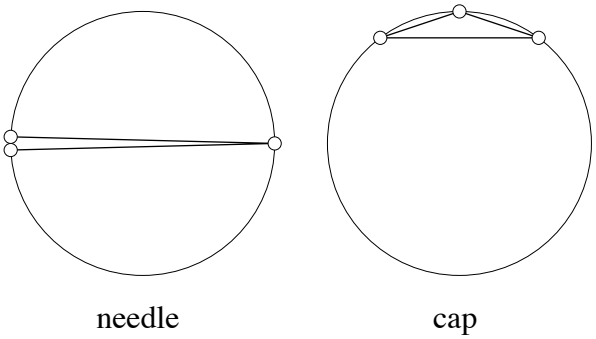

Figure 1.16: Skinny triangles have circumdisks larger than their shortest edges.

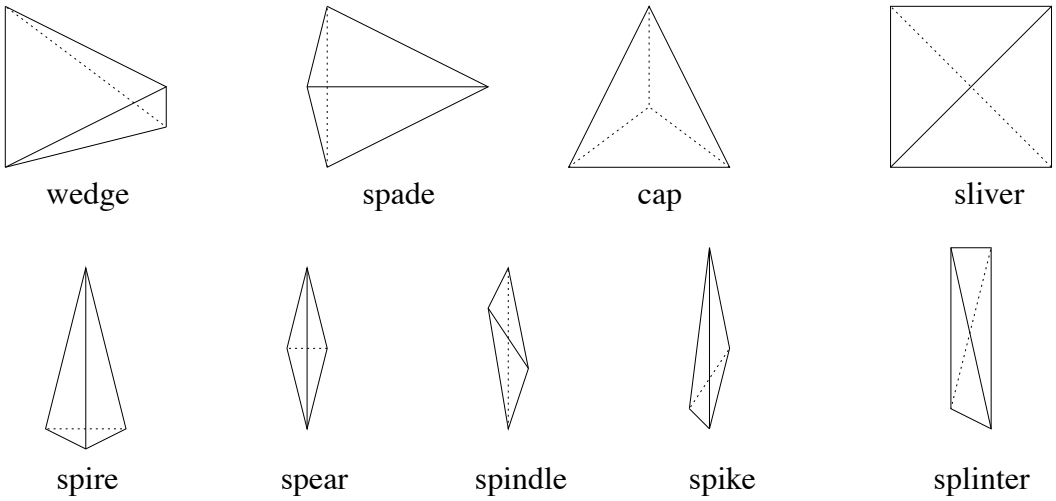

Figure 1.17: A taxonomy of skinny tetrahedra.

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

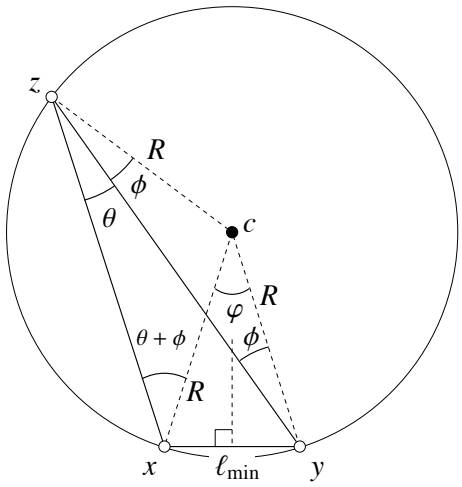

Figure 1.18: Relationships between the circumradius R, shortest edge $\ell_{\min}$, and smallest angle θ.

triangle or tetrahedron, and a value at the opposite extreme for a *degenerate* element—a triangle whose vertices are collinear, or a tetrahedron whose vertices are coplanar. In this book, the most important quality measure is the *radius-edge ratio*, because Delaunay refinement algorithms naturally tend to improve it.

Definition 1.21 (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 1.18 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, $\varphi = 2\theta$ and $\ell_{\min} = 2R \sin \theta$. 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 1.17 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 1.17, including slivers and spires, but Delaunay refinement does not promise to bound these measures. A popular measure is r/R, where r is τ's inradius and R is its circumradius. This measure is sometimes called the *aspect ratio* or the *radius ratio*. 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. This implies that it approaches zero 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 aspect 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 the aspect ratio 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 aspect ratio inherit these problems. In these respects, a better quality measure for tetrahedra is the *volume-length measure* V/ℓ_{rms}^3, 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.

1.8 Notes and exercises

This chapter's opening quote comes from Thompson [215]. An excellent source for many aspects of mesh generation not covered by this book is the *Handbook of Grid Generation* [216], which includes many chapters on the generation of structured meshes, chapters that describe advancing front methods in unusual detail by Peraire, Peiró, and Morgan [169] and Marcum [142], and a fine survey of quadrilateral and hexahedral meshing by Schneiders [186]. Further surveys of the mesh generation literature are supplied by Bern and Eppstein [16] and Thompson and Weatherill [217]. Boissonnat, Cohen-Steiner, Mourrain, Rote, and Vegter [27] survey algorithms for surface meshing.

For evidence that the discretization error and the error in the gradient under piecewise linear interpolation grow with a triangle's largest angle, see Synge [212], Babuška and Aziz [12], and Jamet [117]. For similar evidence for a tetrahedron's largest dihedral angle, see Křížek [125]. The association between the largest eigenvalue of a stiffness matrix and the smallest angle of an element is noted by Fried [101] and Bank and Scott [14]. All these connections are summarized and elaborated by Shewchuk [202]. The Courant–Friedrichs–Lewy condition for stable explicit time integration is, not surprisingly, by Courant, Friedrichs, and Lewy [66]. Adaptive mesh refinement is surveyed by Oden and Demkowicz [161]. There is a large literature on how to numerically evaluate the quality of an element; see Field [95] for a survey.

Advancing front methods that create vertices and triangulate them in two separate stages include those by Frederick, Wong, and Edge [97]; Cavendish [38]; and Lo [139]. Early advancing front methods that interleave vertex creation and element creation include triangular mesh generators by George [102], Sadek [182], and Peraire, Vahdati, Morgan, and Zienkiewicz [170]; tetrahedral meshers by Löhner and Parikh [140] and Peraire, Peiró, Formaggia, Morgan, and Zienkiewicz [168]; a quadrilateral mesher by Blacker and Stephenson [21]; and a hexahedral mesher by Blacker and Meyers [20].

Delaunay mesh generators that create vertices and triangulate them in two separate; stages include those by Frederick et al. [97]; Cavendish, Field, and Frey [39]; and Jameson, Baker, and Weatherill [116]. The first Delaunay refinement algorithm we know of that interleaves the two operations is by Frey [100].

Yerry and Shephard [226, 227] published the first quadtree and octree meshers. Readers not familiar with quadtrees and octrees may consult Samet's book [183].

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 [121] 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 [164, 37, 163]. Slower but better smoothing is provided by the nonsmooth optimization algorithm of Freitag, Jones, and Plassmann [98], 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 [4].

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 [36] provide a fast triangular mesh improvement schedule. Sophisticated schedules for tetrahedral mesh improvement are provided by Joe [120], Freitag and Ollivier-Gooch [99], and Klingner and Shewchuk [124]. For a list of flips for quadrilateral and hexahedral meshes, see Bern, Eppstein, and Erickson [17]. Kinney [122] describes mesh improvement methods for quadrilateral meshes. There does not seem to have been much work on applying hexahedral flips.

The first provably good mesh generation algorithm, by Baker, Grosse, and Rafferty [13], employs a square grid. The first provably good Delaunay refinement algorithm in the plane is by Chew [59], and the most successful is by Ruppert [178, 180]. The first provably good three-dimensional Delaunay refinement algorithm is by Dey, Bajaj, and Sugihara [74]. For a proof that the domain in Figure 1.5 has no mesh whose new angles all exceed 30°, see Shewchuk [196].

The first mesh generator offering provably good grading and size optimality is the quadtree algorithm of Bern, Eppstein, and Gilbert [18]. Neugebauer and Diekmann [158] improve the algorithm by replacing square quadrants with rhomboids. They produce triangles with angles between 30° and 90°, many of them equilateral. The first tetrahedral mesh generator offering size optimality is the octree algorithm of Mitchell and Vavasis [151]. Remarkably, Mitchell and Vavasis [152] extended their mathematical guarantees to meshes of polyhedra of any dimensionality by using d-dimensional 2^d-trees.

The first paper to suggest a generalization of the Delaunay property to meshes of curved surfaces in three dimensions and the first algorithm offering a guarantee on the aspect ratios of the triangles in a surface mesh are by Chew [61]. See the bibliographical notes in Section 14.6 and the aforementioned survey by Boissonnat et al. [27] for a discussion of subsequent surface meshing algorithms. Guaranteed-quality triangular mesh generators for two-dimensional domains with curved boundaries include those by Boivin and Ollivier-Gooch [32] and Pav and Walkington [166]. Labelle and Shewchuk [127] provide a provably good triangular mesh generator that produces anisotropic meshes in the plane, and Cheng, Dey, Ramos, and Wenger [54] generalize it to generate anisotropic meshes of curved surfaces in three-dimensional space.

Bibliographic information for the developments discussed in Section 1.4 is provided in the notes of the chapters listed there, so we omit details here. Publications noted in that section include papers by Shewchuk [197, 198]; Cheng, Dey, Edelsbrunner, Facello, and Teng [49]; Cheng and Dey [48]; Edelsbrunner and Shah [92]; Amenta and Bern [3]; Cheng, Dey, Edelsbrunner, and Sullivan [47]; Boissonnat and Oudot [29]; Cheng, Dey, and Ramos [51]; Cheng, Dey, and Levine [50]; and Dey and Levine [77]. Companion papers are available for each of the programs Triangle [196], Stellar [124], QualMesh [52], SurfRemesh [53], and DelPSC [77].

Books by Hocking and Young [113], Munkres [155], and Weeks [223] are standard texts on point set topology, giving detailed definitions of topological spaces and maps. Books by Hatcher [110] and Stillwell [209] are good sources for algebraic and combinatorial topology; they describe simplicial complexes and their use in triangulations of topological spaces. Some useful definitions in computational topology are collected in the survey paper by Dey, Edelsbrunner, and Guha [75]. Books by Hadwiger [108] and Ziegler [228] are good sources for the mathematics of polyhedra and polytopes. A recent book by De Loera, Rambau, and Santos [68] surveys the mathematical properties of triangulations. Hadwiger popularized Definition 1.7 for nonconvex polyhedra, which we call linear cells. The notion of a piecewise linear complex was introduced by Miller, Talmor, Teng, Walkington, and Wang [149]. Piecewise smooth complexes were introduced by Cheng, Dey, and Ramos [51]. The classification of tetrahedra with tiny angles in Figure 1.17 is adapted from Cheng, Dey, Edelsbrunner, Facello, and Teng [49].

Miller, Talmor, Teng, and Walkington [148] pointed out that the radius-edge ratio is the most natural and elegant measure for analyzing Delaunay refinement algorithms. The use of the incenter-circumcenter ratio as a quality measure was suggested by Cavendish, Field, and Frey [39]. The volume-length measure was suggested by Parthasarathy, Graichen, and Hathaway [163]. See Klingner and Shewchuk [124] for evidence of its utility and the instability of the incenter-circumcenter ratio in mesh improvement algorithms.

Exercises

1. Let X be a point set, not necessarily finite, in $\mathbb{R}^d$. Prove that the following two definitions of the convex hull of X are equivalent.

   - The set of all points that are convex combinations of the points in X.
   - The intersection of all convex sets that include X.

2. Suppose we change the second condition in Definition 1.4, which defines *simplicial complex*, to

   - For any two simplices $\sigma, \tau \in \mathcal{T}$, their intersection $\sigma \cap \tau$ is either empty or a simplex in $\mathcal{T}$.

   Give an illustration of a set that is a simplicial complex under this modified definition but not under the true definition.

3. In every metric space $\mathbb{T}$, the point sets $\emptyset$ and $\mathbb{T}$ are both closed and open.

   (a) Give an example of a metric space that has more than two sets that are both closed and open, and list all of those sets.

   (b) Explain the relationship between the idea of connectedness and the number of sets that are both closed and open.

4. Prove that for every subset X of a metric space, $\operatorname{Cl}\operatorname{Cl} X = \operatorname{Cl} X$. In other words, augmenting a set with its limit points does not give it more limit points.

5. Show that among all triangles whose longest edge has length one, the circumradius approaches infinity if and only if the largest angle approaches 180°, whereas the inradius approaches zero if and only if the smallest angle approaches 0°.

6. One quality measure for a simplex is its minimum altitude divided by the length of its longest edge, which unfortunately is also called the *aspect ratio*. For triangles, prove that this ratio approaches zero if and only if the smallest angle approaches zero. For tetrahedra, prove that this ratio approaches zero if a dihedral angle approaches 0° or 180°, but that the converse is not true.

7. Another measure, which has been seriously proposed in the literature as a quality measure for a tetrahedron by researchers who will remain anonymous, is the length of its longest edge divided by its circumradius. Explain why this is a bad quality measure. In a few words, what triangle shapes maximize it? What is the worst tetrahedron shape that maximizes it?

8. The classification of tetrahedra shown in Figure 1.17 is qualitative. To make it precise, set two thresholds ρ and ℓ, and call a radius-edge ratio large if it exceeds ρ and an edge length small if it is less than ℓ. Describe how to use ρ and ℓ to decide into which class in Figure 1.17 a tetrahedron falls.