Chapter 10

Unstructured Mesh Generation

Jonathan Richard Shewchuk

University of California, Berkeley

10.1	Introduction	259
10.2	Meshes	261
10.2.1	Domain Conformity	262
10.2.2	What Is a Good Element?	263
10.3	Methods of Mesh Generation	266
10.3.1	Advancing Front Mesh Generation	267
10.3.1.1	A Generic Advancing Front Method	267
10.3.1.2	Some Specific Algorithms	268
10.3.2	Delaunay Mesh Generation	272
10.3.2.1	Delaunay and Constrained Delaunay Triangulations	272
10.3.2.2	Algorithms for Constructing Delaunay Triangulations ..	274
10.3.2.3	A Generic Delaunay Refinement Algorithm	276
10.3.2.4	Some Specific Algorithms	277
10.3.3	Grid, Quadtree, and Octree Mesh Generation	280
10.3.3.1	A Generic Octree Mesher	280
10.3.3.2	Some Specific Algorithms	281
10.3.4	Mesh Improvement	283
10.4	Guaranteed-Quality Mesh Generation	285

10.1 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 [112], 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). 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. Some problems in mesh generation—for example, dividing an arbitrary three-dimensional domain into a modest number of nicely shaped hexahedra—are suspected to have no general solution.

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 octree, Delaunay, and advancing front methods for mesh generation, and mesh “clean-up” methods such as smoothing and topological transformations. Unfortunately, nearly all the algorithms developed during this period are fragile, and produce unsatisfying meshes when confronted by complex domain geometries (e.g. domain faces meeting at small angles or in topologically complicated ways), stringent demands on element shape, or numerically imprecise CAD models.

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.

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 are used for multivariate interpolation in hundreds of other applications. For example, an airborne laser scanning technology called LIDAR (Light Detection and Ranging) enables an airplane to measure the altitude of the ground at millions of locations in a few hours. A cartographer can generate a mesh of triangles whose corners lie at those locations, estimate the altitude at any other point by interpolation, and generate a contour map of the terrain. Similarly, meshes find heavy use in image processing, geographic information systems, radio propagation analysis, shape matching, and population sampling. Mesh generation has become a truly interdisciplinary topic.

10.2 Meshes

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.

Quadrilateral elements are four-sided polygons; their sides need not be parallel. Hexahedral elements are brick-like polyhedra, but their faces need not be parallel or even planar. Simplicial meshes (triangular and tetrahedral) are much easier to generate, but 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.

As a compromise, some applications can use *hex-dominant* meshes (and occasionally their two-dimensional analog, *quad-dominant* meshes) in which

most of the elements are hexahedra, but other element shapes—tetrahedra, pyramids, wedges—fill the regions where the meshing algorithm cannot produce good hexahedra.

Meshes are also categorized as structured or unstructured. A *structured mesh* (near right), 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 chapter discusses only *unstructured meshes* (far right), which entail explicitly storing each vertex's neighboring vertices or elements. Structured meshes have been studied extensively [113]; they are suitable primarily for domains that have tractable geometries and do not require a strongly graded mesh.

The elements comprising a mesh must entirely cover the domain without overlapping each other. For most applications their faces must intersect “nicely,” so that if two elements intersect, their intersection is a vertex or edge or entire face of both. (Formally, a mesh must be a *cell complex*.) The mesh generation problem becomes superficially easier if we permit *nonconforming elements* like the triangle at right that shares one edge with two other triangles each having half of that edge. But nonconforming elements rarely alleviate the underlying numerical problems, so they are rarely used 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.

10.2.1 Domain Conformity

Mesh generation algorithms vary immensely 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. Consider these geometries in turn.

Simply cutting a polyhedron into tetrahedra without producing unreasonably many is thorny, even if the element sizes and shapes are unconstrained. Illustrated at right is a polyhedron of Chazelle [25] that has n vertices and $\mathcal{O}(n)$ edges, but cannot be divided into fewer than $\Theta(n^2)$ convex bodies. Chazelle and

Palios [26] show that the worst-case complexity of subdividing a polyhedron is related to its number of *reflex edges*: edges at which the interior dihedral angle exceeds 180° . (A *dihedral angle* is the angle at which two faces meet in three dimensions. A simple polyhedron is convex if and only if none of its interior dihedral angles exceeds 180° .) Specifically, some polyhedra with r reflex edges cannot be divided into fewer than $\Omega(n + r^2)$ convex bodies. Conversely, Chazelle and Palios give an algorithm that divides any polyhedron into $\mathcal{O}(n + r^2)$ tetrahedra.

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.

Curved domains introduce more difficulties. Some applications require elements that curve to match a domain. Others approximate a 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. Regardless of the choice, it is tricky to mesh a *cusp* where two curved patches meet at a dihedral angle of zero. If the two patches are broken into linear triangles with insufficient care, they can easily intersect each other where they should not.

Curved domains are often represented ambiguously. In many CAD models, two patches that are supposed to meet at a shared ridge do not actually meet, or they extend too far and pass through each other. These defects are usually tiny and do no harm when the model is visualized or used for manufacturing, but they can be fatal to a mesh generator that tries to conform to the defects. The topology of the ridges and patches (i.e. the ways that they meet) is sometimes ambiguous, and so is the question of what is inside the domain and what is outside. The problem of *CAD clean-up*—the preparation of solid models so they are suitable as input to mesh generators—has received substantial attention in industry, albeit little from academia. A notable virtue of the Finite Octree mesh generator of Shephard and Georges [103] is that it queries CAD modeling programs directly and copes with their inconsistencies.

10.2.2 What Is a Good Element?

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 be-

tween 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. Most applications of triangulations use them to interpolate a multivariate function whose true value might or might not be known. The most popular interpolation methods are the simplest ones: piecewise linear interpolation over triangles and tetrahedra (though quadratic and higher-order elements are not rare), piecewise bilinear interpolation over quadrilaterals, and piecewise trilinear interpolation over hexahedra. There are two types 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 of these—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° [111, 5, 60, 106]. (In three dimensions, it is the element's largest dihedral angle that usually matters most [66, 106].) This error is the reason why large angles are forbidden. In the finite element method, this error contributes to the discretization error [5]. In surface meshes for computer graphics, it causes triangles to have normal vectors that poorly approximate the normal to the true surface, and these can create visual artifacts in rendering. Nonconvex quadrilateral and hexahedral elements, with angles exceeding 180° , sabotage interpolation and the finite element method.

The second constraint forbids small angles, albeit not for all applications. 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 approaches infinity as an element's smallest angle approaches zero [7, 106]. Fortunately, most linear equation solvers cope well with a few bad eigenvalues.

Although large and sometimes small angles are bad as they approach the limits of 180° and 0° , there are two circumstances in which the ideal element is not equilateral but anisotropic. First, if you wish to interpolate a function as accurately as possible with a fixed number of elements, you should tailor their anisotropy according to the function. This is a vague statement,

because it depends on the order of the interpolation method; to be more concrete, if you use linear interpolation over triangular or tetrahedral elements, their anisotropy should be determined by the anisotropy of the second derivatives of the function being interpolated, with the greatest density of elements along the directions maximizing the second directional derivatives. That is why aerodynamics simulations use extremely thin wedge-shaped elements to model laminar air flow at the surface of an aerofoil: the velocities and pressures vary little along the surface of the aerofoil, but vary rapidly as you move away. These elements make simulations possible that would be computationally intractable with isotropic elements, because they would require too many of them. Second, if you are solving an anisotropic PDE—for instance, modeling a material with anisotropic material properties, such as wood with its grain—you can achieve better matrix conditioning with elements having matching anisotropy. It is possible, though not common, for these two drivers of anisotropy (interpolation error and matrix conditioning) to conflict sharply in their preferences.

The third constraint 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. 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. Hexahedral meshes are especially difficult to grade without including bad elements. Hence, 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.

There is a chicken-and-egg problem: element size and anisotropy should be tailored to the function being interpolated, but in the finite element method, you need a mesh to discover what that function is. This motivates *adaptive refinement*: use a coarse mesh to generate a rough initial solution of a PDE, then generate a better, finer mesh by using the coarse mesh and the approximate solution to determine the size field and perhaps the anisotropy. Solve the PDE again, more accurately. Repeat as desired.

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 [37], 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.”

The constraints on domain conformity and element quality are more difficult to satisfy together than separately. The simplest algorithms for triangulating a polygon [8] or polyhedron [26] usually produce very skinny triangles or tetrahedra. Conversely, we know how to tile space or slabs with acute tetrahedra [44] or perfectly cubical hexahedra, but the same is not true for interestingly shaped domains.

Some meshing problems are impossible. An obvious example is a polygonal domain that has a corner with a 1° angle, and a demand to mesh it with triangles having all their angles greater than 30° . The problem becomes possible again if we relax it, and demand angles greater than 30° everywhere *except* in the 1° corner. Polygonal domains with small angles can be meshed by cutting off the sharp corners before applying a standard meshing algorithm to the remainder [11]. But for domains with internal boundaries, the problem is impossible again—the illustration at right depicts a domain composed of two polygons glued together that, surprisingly, provably has no mesh whose new angles are all over 30° [104]. Simple polyhedra in three dimensions inherit this hurdle—even without internal boundaries. Three-dimensional domains with small angles and internal boundaries, wherein an arbitrary number of ridges and patches can meet at a single vertex, are one of the biggest challenges in mesh generation.

10.3 Methods of Mesh Generation

This section compares three classes of mesh generation algorithms: advancing front methods, wherein elements crystallize one by one, coalescing from the boundary of a domain to its center; Delaunay refinement algorithms, which construct a Delaunay triangulation and insert additional vertices to ensure that its elements are good; and grid, quadtree, and octree algorithms, which overlay a structured background grid and use it as a guide to subdivide a domain. A fourth subsection describes 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.

I give short shrift to many worthy topics. Schneiders [98] surveys quadrilateral and hexahedral meshing far more deeply. Bern and Eppstein [8] survey triangulation algorithms that optimize many criteria not discussed here. I ignore rich literatures on surface meshing [15] and parallelism.

Before diving in, I want to highlight a startling 1970 article by Frederick,

Wong, and Edge [47] that describes, to the best of my 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. Perhaps this paper is still so little known because it was two decades ahead of its time.

10.3.1 Advancing Front Mesh Generation

Advancing front methods construct elements one by one, starting from the domain boundary and advancing inward—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 [47, 22, 73]. For instance, Frederick, Wong, and Edge [47] 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 [51] took this step in 1971, but it was forgotten and reinvented in 1980 by Sadek [96] and again in 1987 by Peraire, Vahdati, Morgan, and Zienkiewicz [90], who also introduced support for anisotropic triangles. Immediately thereafter, methods of this design appeared for tetrahedral meshing [75, 88].

10.3.1.1 A Generic Advancing Front Method

Advancing front algorithms mesh the domain features in order of increasing dimension: first, the domain edges and ridges are subdivided into mesh edges whose lengths are specified by the size field; then, the domain faces and patches are subdivided into triangles or quadrilaterals; then, the domain volumes are subdivided into tetrahedra or hexahedra. Each such subdivision is

usually done with an advancing front subroutine. (There are some odd exceptions. Marcum and Weatherill [77] use a Delaunay mesh generator to create a complete tetrahedralization, then throw it all away except for the surface mesh, which they use to seed their advancing front algorithm.)

For concreteness, consider meshing a volume with tetrahedra. The algorithm maintains a *front* composed of *active* triangles. A triangle is active if we desire to construct a tetrahedron upon it: either it is a domain boundary triangle with no tetrahedron yet adjoining it, or it is a face of one and only one tetrahedron and should be shared by a second adjoining tetrahedron.

The core of the algorithm is a loop that selects an active triangle and constructs a new tetrahedron upon it. Often, the new tetrahedron adjoins more than one active triangle. To detect this circumstance, the algorithm maintains the active triangles in a dictionary (e.g. a hash table) that maps a triple of vertex indices to the triangle having those vertices, if that triangle is active. Each triangle is assigned an *orientation* that specifies which of its two sides is active. Consider the following generic advancing front algorithm.

```

ADVANCINGFRONT( $B$ )
{  $B$  is a set of boundary faces, with orientations. }
1   $D \leftarrow$  a dictionary containing the oriented faces in  $B$ 
2  while  $D$  is not empty
    { Loop invariant: at this point,  $D$  contains all the active faces. }
3    Remove a face  $f$  from  $D$ 
4    Choose a vertex  $v$  (new or preexisting) so that  $t = \text{conv}(f \cup v)$  is
        nicely shaped and nicely intersects the front
5    Output the element  $t$ 
6    for each face  $g$  of  $t$  except  $f$ 
7        if  $g$  is in the dictionary
8            Remove  $g$  from the dictionary
9        else Enter  $g$  in the dictionary, oriented facing away from  $t$ 

```

Implementations of this generic algorithm can vary in many ways.

- How do we choose which face to advance (Line 3)?
- How do we choose a new or preexisting vertex (Line 4)?
- How do we *quickly* find the preexisting vertices that are good candidates?
- How do we determine if a new element intersects the front nicely (i.e. forms a cell complex)?

The first two questions crucially affect not only the quality of the mesh, but also whether the algorithm will complete its task at all. The next two are questions solely of speed: they can be done by exhaustive checks, at great computational cost, or they can be accelerated by spatial search algorithms that identify candidates close to the active face f .

10.3.1.2 Some Specific Algorithms

Let us see how different researchers address these four questions. Consider the choice of face to advance. Peraire, Peiró, and Morgan [89] find that for tri-

angular meshes, advancing the shortest edge gives the best results, because it makes it easier to generate good triangles where fronts collide. For tetrahedral meshes, they choose the active face with the shortest altitude. Marcum [76], who specializes in meshes for aerodynamics, prefers to lay down a whole row of elements before starting another, so that the elements near the boundary are as structured as possible. Sadek [96] pri-

A tetrahedral advancing front mesh of the air around an airplane nacelle. Courtesy of David Marcum.

oritizes elements within a row by noting corners in the front where the angle is not close to 180° and filling them in early. He constructs all the elements adjoining a corner simultaneously to find the best compromise among them.

Second, consider the choice of a vertex (Line 4). For a triangular mesh and an active edge f , Peraire et al. compute a preexisting candidate vertex v' and a new candidate vertex v'' . The preexisting candidate is chosen by the Delaunay empty circle criterion (see Section 10.3.2.1), subject to the constraint that the candidate triangle $\text{conv}(f \cup v')$ must nicely intersect the front. (It appears that their two-dimensional algorithm nearly always produces a constrained Delaunay triangulation.) The new candidate v'' is chosen on f 's bisector, so the candidate triangle $\text{conv}(f \cup v'')$ is isosceles. The lengths of this candidate triangle's two new edges are dictated by the size field, subject to the constraint that the candidate triangle not be too skinny.

Peraire et al. choose the new candidate v'' over the preexisting candidate v' if and only if the circle centered at v'' and passing through the endpoints of f encloses no valid preexisting candidate, as illustrated at right. In other words, the angle of the triangle $\text{conv}(f \cup v'')$ at v'' is at least twice the angle of the triangle $\text{conv}(f \cup v')$ at v' . Thus, they never create a new vertex too close to a preexisting one.

Unfortunately, sometimes every candidate vertex— v'' and every preexisting vertex—yields an invalid new element. This circumstance occurs where two fronts collide, especially when a front with small elements meets a front with much larger elements, and the latter front lacks good candidate vertices for the former. A new candidate vertex must be generated close enough to f to create a valid element. Ideally, the size field prevents such disparities between colliding fronts; but sometimes small geometric features of the domain boundary or the inherent difficulty of filling a polyhedral cavity with tetrahedra interfere.

For tetrahedral meshes, the process of generating a new candidate vertex and comparing it with preexisting vertices is more complicated. I omit details,

but Peraire et al. do not use the Delaunay criterion, which is less effective in three dimensions (see Section 10.4).

Colliding fronts in tetrahedral meshes can be more difficult to resolve than you might guess. The difficulty arises because there are polyhedra that cannot be subdivided into tetrahedra without creating new vertices. At right is Schönhardt's untetrahedralizable polyhedron [99], which has three reflex edges. If an advancing front algorithm produces a cavity of this shape, it must introduce a new vertex inside the cavity to complete the mesh. One vertex at the center of Schönhardt's polyhedron will do, but if the mesher is not careful about where it places the vertex, it will need to introduce more. In general, it is NP-hard to determine how many new vertices are required to tetrahedralize a cavity [95]. I am not aware of any tetrahedral advancing front algorithm that guarantees it can fill any polyhedral cavity.

Advancing front methods generate anisotropic meshes easily. The user specifies the locally desired anisotropy as a space-varying tensor field akin to the size field. The mesher simply applies an affine transformation to the mesh when it performs a front-advancing step; the candidate vertices v' and v'' are computed and compared in the transformed space, so the algorithm constructs an anisotropic element whose orientation and eccentricity are approximately as specified.

Marcum [76] notes that for meshes in aerodynamics, especially anisotropic ones, it is sometimes better to create new candidate vertices with an *advancing-point* or *advancing-normal* placement, which means that vertices near important surfaces are aligned along lines orthogonal to those surfaces.

Our third question is, how do we screen out preexisting vertices as candidates without checking every one of them? Peraire et al. store vertices in a spatial search data structure called an *alternating digital tree*, which is similar enough to a quadtree or octree that the differences are unimportant. With this data structure, one can query all the front vertices in a ball around f , and the query is usually efficient in practice (though there is no guarantee).

For triangular meshes, Mavriplis [79] maintains a constrained Delaunay triangulation of the unmeshed part of the domain, and uses it as a search structure. It has the nice property that the best preexisting candidate vertex for an edge f is already connected to f by a triangle. Other preexisting candidate vertices (e.g. for anisotropic meshing) can be found efficiently by a depth-first search through adjoining triangles. For tetrahedral meshes, Marcum uses the same strategy of maintaining a triangulation of the unmeshed region, but the triangulation is generally not Delaunay. Each newly constructed tetrahedron is introduced into this triangulation (if it is not already there) by a sequence of tetrahedral transformations: tetrahedron subdivisions to introduce new vertices, and topological flips (see Section 10.3.4)

to force the edges and faces of the new tetrahedron into the triangulation. There is no guarantee that these flips will always work!

Our fourth question is, how do we determine if a new element t intersects the front illegally? For triangular meshes, it suffices to check whether t 's new edges intersect any active edges (and if so, *how* they intersect them). For tetrahedral meshes, we must check t 's new faces for edge-face intersections with the active faces. Löhner and Parikh [75] describe how to perform these tests for any pair of triangular faces. But how do we avoid the computational cost of checking *every* active face?

One simple solution is to use a quadtree or octree, and store each active face in every leaf octant it intersects. (These octants are identified by a simple depth-first search.) For each prospective new element, determine which leaf octants its new faces intersect, look up which active faces intersect those octants, and check only those faces—in practice, a small fraction of the total—against the new faces. Unfortunately, this data structure can store a face in many leaves. Peraire et al. avoid this cost by casting the problem as a six-dimensional *orthogonal range search* query: enclose each face in an axis-aligned bounding box, represent each box as a point in six-dimensional space (listing the three-dimensional coordinates of the minimum and maximum corners of the box), and store these points in a six-dimensional octree. A query asks which stored bounding boxes intersect another axis-aligned bounding box; this query is answered with a simple depth-first search of the octree.

Mavriplis and Marcum use an entirely different method: they exploit the fact that they have a triangulation of the unmeshed region by tracing the new edges/faces through the triangles/tetrahedra of that triangulation.

In practice, these search structures improve the speed substantially, and advancing front meshers typically generate an n -vertex mesh in $\mathcal{O}(n \log n)$ time. There are pathological circumstances in which the running time is asymptotically greater, but they are rare in practice.

Perhaps the most famous quadrilateral mesh generator is an advancing front method, Blacker and Stephenson's *Paving* algorithm [14]. It is substantially more intricate than even the tetrahedral methods. Consider the following difficulty. Because every quadrilateral has four edges, and every edge in the interior of the domain is shared by exactly two quadrilaterals, a quadrilateral mesh must have an even number of edges on its boundary. As the fronts advance into a domain, their collisions sometimes subdivide

A quadrilateral mesh generated by Paving [14]. Courtesy of Ted Blacker.

a cavity into two smaller cavities. If each smaller cavity has an odd number of edges, the mesher is doomed. To prevent this peril, Paving sometimes deforms a row of elements so that two colliding fronts meet with the correct parity.

Hexahedral meshing is fundamentally harder, and no reliable unstructured hexahedral meshing algorithm exists yet. Early efforts at advancing front methods were thwarted by cavities that could not be finished. A fascinating recent idea called *Unconstrained Plastering* [110] advances the front one sheet at a time without immediately deciding how to subdivide the sheet into hexahedra. An element is formed where three such sheets intersect. It seems to be crucial *not* to pre-mesh the boundary with quadrilaterals.

10.3.2 Delaunay Mesh Generation

The Delaunay triangulation is a geometric structure introduced by Boris Nikolaevich Delaunay [38] in 1934. In two dimensions, it has a striking advantage: among all possible triangulations of a fixed set of vertices, the Delaunay triangulation maximizes the minimum angle [70]. Sadly, this property does not generalize to tetrahedra, but Delaunay triangulations of any dimensionality optimize several geometric criteria related to interpolation accuracy [91, 80, 115].

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 triangulation guides the algorithm to place new vertices far from existing ones, so that short edges and skinny elements are not created unnecessarily.

Most Delaunay methods, unlike advancing front methods, create their worst elements near the domain boundary and their best elements in the interior, which suggests the wisdom of combining the two methods and using Delaunay refinement in place of nasty advancing front collisions; see Section 10.3.2.4. The advantage of Delaunay methods, besides their optimality properties, 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.

10.3.2.1 Delaunay and Constrained Delaunay Triangulations

Let V be a set of vertices in the plane, not all collinear. The *circumscribing circle*, or *circumcircle*, of a triangle is the unique circle that passes through all three of its vertices. A triangle whose vertices are in V is *Delaunay* if its circumcircle encloses no vertex in V . Note that any number of vertices might lie *on* the triangle's circumcircle; so long as the circle is *empty*—encloses no vertex—the triangle is Delaunay. A *Delaunay triangulation* of V is a subdivision of V 's convex hull into Delaunay triangles with disjoint interiors.

It is not obvious that a triangulation of V exists whose triangles are all Delaunay. The easiest way to see this is through a *lifting map* that transforms V 's Delaunay triangles into faces of a convex polyhedron in three dimensions. The first lifting map, by Brown [20], projects V onto a sphere, but the *parabolic lifting map* of Seidel [101] is numerically better behaved. It maps each vertex $(x, y) \in V$ to a lifted point $(x, y, x^2 + y^2)$ in E^3 . Consider the convex hull of the lifted points $\{(x, y, x^2 + y^2) : (x, y) \in V\}$, a polyhedron illustrated above. Taking the underside of this convex hull and projecting it down to the x - y plane yields the Delaunay triangulation of V .

This relationship between Delaunay triangulations and convex hulls has two consequences. First, from the fact that every finite point set has a convex hull, it follows that every finite point set has a Delaunay triangulation. Second, it brings to meshing the power of a huge literature on polytope theory and algorithms: every convex hull algorithm is a Delaunay triangulation algorithm!

The Delaunay triangulation of V is unique if and only if no four vertices in V lie on a common empty circle. (In that case, the Delaunay triangulation is the planar dual of the well-known *Voronoi diagram*.) If it is not unique, the lifting map produces a convex hull that has faces with more than three sides; any arbitrary triangulation of these faces yields a Delaunay triangulation.

Delaunay triangulations extend to any dimensionality. The *circumsphere* of a tetrahedron is the unique sphere that passes through all four of its vertices. A tetrahedron is *Delaunay* if its circumsphere encloses no vertex. A set of vertices in three dimensions, not all coplanar, has a Delaunay tetrahedralization, which is unique if and only if no five vertices lie on a common empty sphere. The lifting map relates the triangulation to a convex, four-dimensional polyhedron.

As Delaunay triangulations in the plane maximize the minimum angle, do they solve the problem of triangular mesh generation? No, for two reasons illustrated at right. 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 inserting additional vertices, as illustrated. Section 10.3.2.3 explains where to place them.

An alternative solution to the second problem is to maintain a *constrained Delaunay triangulation* (CDT), illustrated below. A CDT is defined with respect to a set of vertices and *segments* that demarcate the domain boundary (including internal boundaries). Every segment is required to appear as an edge of the CDT. The triangles of a CDT are not required to be Delaunay; instead, they must be *constrained Delaunay*. The definition, which I omit, partly relaxes the empty circumcircle criterion to compensate for the demand to include every segment.

One merit of a CDT is that it can respect arbitrary segments without requiring the insertion of any additional vertices besides the endpoints of

the segments. CDTs were mathematically formalized by Lee and Lin [71] in 1986, though algorithms that unwittingly construct CDTs appeared much sooner [47, 85]. Lee and Lin show that the CDT inherits the Delaunay triangulation's optimality: among all possible triangulations of the vertex set *that include all the segments*, the CDT maximizes the minimum angle.

CDTs generalize to three dimensions [108]. However, there exist simple polyhedra that do not have CDTs—recall Schönhardt's and Chazelle's polyhedra [99, 25], which cannot be subdivided into tetrahedra without new vertices that are not vertices of the polyhedra. To create a CDT that respects the boundary of a polyhedral domain, you sometimes must add new vertices on the domain's segments. Once the (unconstrained) Delaunay triangulation of the vertices respects the segments, the augmented domain has a CDT; no additional vertices are needed on the domain faces.

10.3.2.2 Algorithms for Constructing Delaunay Triangulations

The first published Delaunay triangulation algorithm I know of tests every possible d -simplex—every combination of $d + 1$ vertices, where d is the dimension—to see if its circumsphere is empty, taking $\mathcal{O}(n^{d+2})$ time. This brute force algorithm (in its dual, Voronoi form) and a three-dimensional implementation appear in a 1967 paper by John Bernal, a father of structural biology who discovered the structure of graphite and was awarded a Stalin Peace Prize, and John Finney, his last Ph.D. student [12].

The next-simplest Delaunay triangulation algorithms are *gift-wrapping algorithms*, which are advancing front methods minus the creation of new vertices. This class includes the pioneering 1970 algorithm of Frederick, Wong, and Edge [47]. The first gift-wrapping algorithm for three or more dimensions appeared in 1979 when Brown [20] published his lifting map and observed that the 1970 gift-wrapping algorithm of Chand and Kapur [24] for computing general-dimensional convex hulls can construct Voronoi diagrams. The two- and three-dimensional algorithms have been rediscovered many times since.

In two dimensions, the empty circumcircle property implies a simple gift-wrapping rule: to advance an active edge (Line 4 of ADVANCINGFRONT), choose the candidate vertex v that maximizes the angle at v in the new triangle. (For numerical reasons, an implementation should not actually compute this angle; it should choose among the vertices with a polynomial test described by Guibas and Stolfi [54]—which, unlike the angle test, also works in three dimensions.)

A useful observation from computational geometry is that we can exploit

the uniqueness of the Delaunay triangulation (enforced by the careful use of symbolic weight perturbations [43, Section 5.4]) to remove the tests for invalid element intersections from Line 4 of ADVANCINGFRONT, thereby saving a lot of time. However, keeping those tests intact enables gift-wrapping to compute a *constrained* Delaunay triangulation, and that is what Frederick et al. did, and what Nguyen [85] did for tetrahedra, long before the mathematics were developed to justify the rightness of what they were doing.

Gift-wrapping has the disadvantage that the running time of a naïve implementation is quadratic in the number of vertices—sometimes cubic in three dimensions. For uniform vertex distributions, clever bucketing strategies can improve Delaunay gift-wrapping to expected linear time, even in three or more dimensions [41]. Nevertheless, this leaves us in want of a fast algorithm for graded vertex distributions and for CDTs.

In the decision-tree model of computation, sets of n vertices in the plane sometimes require $\Omega(n \log n)$ time to triangulate. The first algorithm to run in optimal $\mathcal{O}(n \log n)$ time was the *divide-and-conquer algorithm* of Shamos and Hoey [102], subsequently simplified by Lee and Schachter [72] and Guibas and Stolfi [54] and sped up by Dwyer [40]. The algorithm partitions a set of vertices 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 [104], but 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. Incremental insertion algorithms are the fastest three-dimensional Delaunay triangulators in practice. 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 insert additional vertices, making incremental insertion obligatory.

Lawson [70] invented an algorithm for incremental insertion, but it works only in two dimensions. A faster algorithm that works in any dimensionality was discovered independently by Bowyer [18], Hermeline [56, 57], and Watson [116]. Bowyer and Watson simultaneously submitted it to *Computer Journal* and found their articles published side by side.

The idea of vertex insertion is simple.

Suppose you are inserting a new vertex v into a triangulation. If a triangle's circum-circle (or a tetrahedron's circumsphere) encloses v , that element is no longer De-

launay, so it must be deleted. This suggests the *Bowyer–Watson algorithm*.

- Find one element whose circumsphere encloses v .
- Find all the others by a depth-first search in the triangulation.
- Delete them all, creating a polyhedral cavity; see the figure above.
- For each edge/face of this cavity, create a new element joining it with v .

It is not difficult to prove that the new elements are Delaunay.

The first step is called *point location*. Most Delaunay mesh generation algorithms generate new vertices inside the circumspheres of badly shaped or oversized elements, in which case point location is free. However, this is not true for the input vertices—vertices of the domain and other vertices a user wishes to incorporate. Locating these points in the triangulation is sometimes the most costly and complicated part of the algorithm. Incremental insertion algorithms are differentiated mainly by their point location methods.

In one of the great papers of computational geometry, Ken Clarkson and Peter Shor [36] show (among many other wonders) that a point location method based on a *conflict graph* makes incremental insertion run in expected $\mathcal{O}(n \log n)$ time in the plane and expected $\mathcal{O}(n^2)$ time in three dimensions. Both times are worst-case optimal. Crucial to this performance is the idea that the vertices are inserted in random order, which is only possible for vertices that are known from the start. The three-dimensional algorithm often runs in $\mathcal{O}(n \log n)$ time in practice [2], but certainly not always—it is possible for a Delaunay triangulation to have $\Theta(n^2)$ tetrahedra! If you wish to implement a three-dimensional Delaunay triangulator, I recommend an improved variant of the Clarkson–Shor algorithm by Amenta, Choi, and Rote [2].

All three types of algorithm extend to CDTs. Gift-wrapping algorithms are the simplest, but the collision tests make them slow. A divide-and-conquer algorithm of Chew [32] constructs planar CDTs in $\mathcal{O}(n \log n)$ time, but it is complicated. The algorithm most commonly used in practice begins with a Delaunay triangulation of the domain vertices, then incrementally inserts the domain edges into the CDT [3]. The running time could be as slow as $\Theta(n^3)$ but is usually in $\mathcal{O}(n \log n)$ in practice, and the algorithm’s simplicity suggests that the theoretically optimal algorithms would rarely be competitive. Three-dimensional CDTs can be constructed by incremental face insertion in $\mathcal{O}(n^2 \log n)$ time [107], but this algorithm also is likely to be fast in practice.

10.3.2.3 A Generic Delaunay Refinement Algorithm

The early Delaunay mesh generators, like the early advancing front methods, created vertices and triangulated them in two separate stages [47, 23, 59]; and as with advancing front methods, the era of modern meshing began in 1987 with the insight, care of William Frey [50], 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.

Unlike advancing front algorithms, most Delaunay algorithms do not triangulate the domain boundary before creating tetrahedra, but there are exceptions. The following generic Delaunay refinement algorithm inserts vertices for two reasons: to enforce boundary conformity, and to eliminate bad elements.

```

DELAUNAYREFINEMENT( $V, R, P$ )
{  $V$ ,  $R$ , and  $P$  are sets of boundary vertices, ridges, and patches. }
{ Each patch is bounded by ridges in  $R$ ; each ridge's endpoints are in  $V$ . }
1   $T \leftarrow$  the Delaunay triangulation of  $V$ 
2  while  $T$  contains a skinny or oversized element or
       $T$  does not respect all the ridges in  $R$  and patches in  $P$ 
3      if  $T$  fails to respect some ridge  $r \in R$ 
4           $v \leftarrow$  a point on the disrespected portion of  $r$ 
5      else if  $T$  fails to respect some patch  $p \in P$ 
6           $v \leftarrow$  a point on  $p$ , on or near the disrespected portion
7      else if  $T$  contains a skinny or oversized element  $t$ 
8           $v \leftarrow$  a point inside  $t$ 's circumsphere
9      Insert  $v$  into  $T$ , updating  $T$  so it is still Delaunay
10 Delete from  $T$  the elements that lie outside the domain

```

Implementations of this generic algorithm vary primarily in two ways.

- How do we choose a new vertex to eliminate a bad element (Line 8)?
- How do we choose vertices to enforce domain conformity (Lines 3–6)?

10.3.2.4 Some Specific Algorithms

Let us see how different researchers address these two questions. Where should we position a new vertex to remove an element t (Line 8)? Any point inside t 's circumsphere will suffice. Frey [50] chooses t 's *circumcenter*—the center of its circumsphere, illustrated at right.

Weatherill [117] chooses t 's centroid. But circumcenters are usually a better choice. Why?

If a new vertex is too close to a prior vertex, the two vertices generate a short edge, which gives rise to skinny elements. The centroid of an element with a large angle can be very

close to the vertex at the large angle. A Delaunay circumcenter, being the center of an empty circumsphere, cannot. It is therefore an excellent place to put a new vertex. This is a powerful advantage of the Delaunay property, essential to proving that Delaunay algorithms are good (Section 10.4). The guiding principle is “create no short edges.”

If a skinny triangle's circumcircle is substantially larger than its shortest edge, it is often better to place the new vertex closer to the short edge, so they form an acceptable new triangle. The effect is to make Delaunay refinement behave like an advancing front method—at least until the fronts collide, whereupon Delaunay refinement performs better. Frey [50] and Üngör [114] report that these “off-centers” give an excellent compromise between the quality and the number of triangles. A more aggressive algorithm of Erten and Üngör [45] optimizes the placement of a new vertex in a circumcircle; it often generates triangular meshes that have no angle smaller than 41° .

Line 8 carries a potential hazard: the new vertex v might lie too close to

the domain boundary, begetting an undesirably small or skinny element in the gap. Sometimes v lies outside the domain, and cannot be inserted.

Most algorithms prevent vertices from lying too close to the boundary by one of three alternatives. First, if an algorithm begins by spacing vertices on the boundary closely enough (relative to the size field and the sizes of the domain's geometric features), no element's circumcenter or centroid will be too close to the boundary [50, 33, 39]. Second, an algorithm can explicitly check if a new vertex is too close to, or beyond, the boundary (a condition called *encroachment*); if so, move the vertex onto the boundary before inserting it, applying the vertex placement rules discussed below for disrespected boundary ridges and patches [94, 105, 17]. Third, an algorithm can fix a new vertex after insertion by smoothing it (see Section 10.3.4).

A mesh generated by Ruppert's Delaunay refinement algorithm [94].

The biggest differences among Delaunay refinement algorithms are in how they enforce domain conformity (Lines 3–6). The earliest algorithms assure boundary integrity simply by refining the boundary enough before generating a mesh [50, 33, 39]. But it is difficult to know how much is enough, especially if the mesh should be graded.

We want each ridge to be approximated by a sequence of edges of the Delaunay mesh. If we keep track of which vertices of the mesh lie on a ridge, as well as their order along the ridge, we know what that sequence of edges should be, and we can check if those edges are actually present. Where an edge is missing, Line 4 generates a new vertex on the ridge, and Line 9 inserts it into the mesh, as illustrated at right. Ideally, two new edges will appear in the Delaunay triangulation to bridge the gap. If not, the missing edges are refined in turn. Schroeder and Shephard [100] call this process *stitching*.

Schroeder and Shephard place the new vertex where the ridge intersects an edge or face of the mesh. By contrast, provably good algorithms usually place the new vertex as far as possible from other vertices on the ridge, to avoid creating short edges. For polyhedral geometries, the new vertex goes at the midpoint of the missing edge [117, 94, 105].

Stitching works for patches much as it does for ridges: we insert vertices on a patch until the mesh conforms to it. But what is a patch's analog to a missing edge? Each patch should be approximated by triangular faces of the Delaunay mesh. Assume that the ridges that bound each patch have been

recovered—Line 3 gives ridges priority over patches. We look at the union of the mesh faces that have all three vertices on the patch, and check for holes.

A hole implies that some edge of the mesh must pass through the patch. Schroeder and Shephard stitch vertices where mesh edges intersect the patch. This strategy carries some risk of inadvertently creating short edges. For polyhedral domains (i.e. all the patches are flat), a provably good algorithm of my own design [105] maintains a two-dimensional Delaunay triangulation of each polygonal patch. If a patch triangulation includes a triangular face that is missing from the tetrahedral mesh, the algorithm generates a new vertex at the circumcenter of the missing triangle, and inserts it into the tetrahedral mesh *and* the patch triangulation. Weatherill [117] stitches new vertices at centroids of missing triangles.

If a domain has ridges or patches separated by small angles, stitching is difficult, because the vertices stitched into one ridge

or patch can knock edges or triangles out of another, as illustrated above. A *conforming Delaunay triangulation* is a Delaunay mesh such that every domain edge is a union of mesh edges, and every domain face is a union of mesh faces. Computational geometers have studied this problem and concluded that a conforming Delaunay triangulation often entails the addition of many, many new vertices. In two dimensions, Bishop [13] shows that every set of m segments and n vertices has a conforming Delaunay triangulation with $\mathcal{O}(m^{2.5} + mn)$ additional vertices. It might be possible to improve this bound, but domains are known for which $\Theta(mn)$ additional vertices are needed. Closing the gap between the $\mathcal{O}(m^{2.5} + mn)$ and $\Omega(mn)$ bounds remains an open problem.

In three dimensions, no polynomial bound is known! Algorithms do exist; Murphy, Mount, and Gable [83] gave the first conforming Delaunay tetrahedralization algorithm, and better subsequent algorithms have been coupled with Delaunay refinement [92]. But if you insist on a truly Delaunay mesh, you will probably always need to tolerate an unnecessarily large number of vertices near small domain angles.

CDTs obviate these problems in two dimensions, without the need for any new vertices at all. In three dimensions, CDTs starkly reduce the number of new vertices that must be added, but a polynomial bound is still not known.

The advantage of CDTs over Delaunay triangulations is less pronounced when coupled with stringent demands for element quality. When a bad element's circumcenter strays too close to the domain boundary, Delaunay refinement algorithms must refine the boundary, CDT or not. Nevertheless, algorithms that employ CDTs refine less and deliver meshes with fewer elements. Examples include the triangular meshers of Chew [33, 34] and the tetrahedral mesher of Si [109]. Boivin and Ollivier-Gooch [17] show that CDTs can also aid the recovery of curved ridges in triangular meshes.

Another advantage of CDTs is that the simplices outside the domain can be deleted (Line 10) before the refinement loop (Lines 2–9), saving time and

unnecessary refinement. CDTs are invaluable when a user has subdivided the boundary of a domain, and demands that it not be subdivided further.

Many Delaunay tetrahedral meshers, including algorithms by George, Hecht, and Saltel [53], Weatherill [118], and Hazlewood [55], choose to sacrifice the Delaunay (and constrained Delaunay) property so that ridges and patches can be recovered with fewer new vertices, and so that stitching cannot knock them back out. They use bistellar flips (see Section 10.3.4) to recover ridges and patches when possible, resorting to vertex insertion when necessary. The final meshes are “almost Delaunay” and are usually good in practice, but their quality cannot, so far as we know, be mathematically guaranteed, because circumcenters can be arbitrarily close to prior vertices.

Delaunay mesh generators, like advancing front methods, adapt easily to anisotropy. George and Borouchaki [52] modify the Bowyer–Watson algorithm [18, 116] to use circumellipses instead of circumspheres. However, if the anisotropy field varies over space, the newly created elements might not have empty circumellipses, and the quality of the mesh cannot be guaranteed.

10.3.3 Grid, Quadtree, and Octree Mesh Generation

Another 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. A variable-resolution grid, usually a quadtree or octree, yields a graded mesh. (Readers not familiar with quadtrees may consult Samet [97].) 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.

10.3.3.1 A Generic Octree Mesher

Most quadtree and octree meshers perform roughly the following steps.

OCTREEMESH(B)

- { B is the set of boundary vertices, ridges, and patches. }
- 1 Create an octree, refined as dictated by B and the size field
- 2 Balance the octree
- 3 Warp the octree by moving its vertices as dictated by B
- 4 **for** each leaf octant O that intersects the domain
- 5 Triangulate the intersection of O with the domain

The rules for refining the octree (Line 1) differ radically between algorithms, and are usually complicated. Most algorithms refine any octant containing more than one vertex of B , so every vertex lies in its own leaf octant. Most algorithms also refine any octant that intersects two patches, ridges, or vertices that do not intersect each other. A mesher that supports curved surfaces will subdivide an octant if its intersection with a ridge or patch is not well approximated by a single edge or polygon.

Most algorithms require the octree to satisfy some *balance constraint* (Line 2). The most common balance constraint is to require that any two leaf octants that share an edge differ in width by no more than a factor of two. (Octants that do not intersect the domain are exempted.) Wherever this constraint is violated, the larger octant is refined.

The purpose of warping the octree in Line 3 is to ensure that the elements created by Line 5 have good quality—in particular, that they have no short edges. Warping means to move octree vertices, usually onto the domain boundary. Typically, for each domain vertex in B , the nearest octree vertex is warped (moved) onto the domain vertex, as illustrated at right. Next, for each domain ridge that passes through an octant face, either an octree vertex is warped onto the ridge, or a new vertex is created at the intersection point. The same happens where a domain patch intersects an octant edge.

Finally, the mesher produces a triangulation for each leaf octant, covering the intersection of the (possibly warped) octant with the domain (Lines 4 and 5). The vertices of this triangulation include octant vertices, domain vertices, and possibly new vertices introduced wherever a ridge intersects an octant face or a patch intersects an octant edge. In some algorithms, there are only a small number of cases governing how the domain can intersect a warped octant, and for each case there is a fixed *template* or *stencil* that determines how to triangulate the octant [120, 121, 69]. Meshers using templates can be very fast. Other algorithms use a Delaunay or advancing front triangulation to fill each octant with elements [11, 103]. Yet others triangulate simply by adding a new vertex in the interior of each octant [82].

10.3.3.2 Some Specific Algorithms

The first quadtree mesher, published in 1983 by Yerry and Shephard [120], set the pattern for the generic OCTREEMESH algorithm. Yerry's warping step follows the triangulation step, but this difference is cosmetic. The intersection between a leaf quadrant and the domain is approximated by a square clipped by a single line, with the clipping line passing through two of sixteen points on the quadrant's boundary. Therefore, there is a finite set of possible leaf quadrants, and these are triangulated by templates. There are templates for both triangular and quad-dominant meshes. The approximation implies that the mesh usually does not have the same shape as the domain, so a postprocess-

ing step warps the mesh boundary vertices onto the domain boundary. Finally, the element quality is improved by smoothing (see Section 10.3.4). Best results are obtained if the user aligns the domain vertices with quadtree vertices. The following year, Yerry and Shephard [121] generalized their method to three dimensions.

Bern, Eppstein, and Gilbert [11] designed a provably good triangular mesher for polygons. To guarantee high quality, it refines the quadtree severely—for instance, each input vertex must be in the center square of a 5×5 grid of equal-sized quadrants, empty except for that one vertex. Quadtree balancing and warping follow the generic OCTREEMESH algorithm closely, with an interesting twist: sometimes, the intersection of an octant with the domain comprises two or more connected components. These are meshed independently—the algorithm creates one copy of the quadrant for each connected component, and each copy is warped and perhaps even refined independently. After warping, each leaf quadrant's component has at most eight vertices, and is subdivided into optimal (e.g. Delaunay) triangles.

Probably the most famous octree mesher is the Finite Octree method of Shephard and Georges [103]. Rather than warping the octree before triangulating the octants, Shephard and Georges create a vertex wherever a patch intersects an octant edge or a ridge intersects an octant face, and triangulate each octant with an advancing front or Delaunay method; then they use edge contractions to remove short edges. Finally, they rely on mesh improvement methods (Section 10.3.4) to ensure the quality of the tetrahedra.

A noteworthy feature of the Finite Octree method is its ability to interface with a variety of CAD packages, and to work robustly with dirty CAD data. Recognizing that it is expensive to compute intersections of patches with faces of octants, the algorithm tries to infer as much domain geometry as possible from patch-edge and ridge-face intersections, which are single points. If the intersection of a patch with an octant is a topological disk, these points often (but not always) suffice to infer how the octant should be triangulated.

The provably good *isosurface stuffing* algorithm of Labelle and Shewchuk [69] generates tetrahedral meshes of domains that have entirely smooth surfaces (i.e. no vertices or ridges), like biological tissues. The sole interface required between the algorithm and the domain geometry is the ability to query whether a specified point is inside or outside the domain; this lends the

A tetrahedral mesh generated by a Finite Octree successor in the software *Simmetrix*. A cut shows the interior octree structure. Courtesy of Mark Beall.

algorithm some robustness to imprecise geometry. Isosurface stuffing uses an octree to generate a tetrahedral background mesh, and it warps that mesh

instead of the octree itself. Wherever the surface intersects an edge of the background mesh, either a new vertex is created (if neither edge endpoint is close enough) or an endpoint of the edge is warped onto the intersection. Background tetrahedra are subdivided into mesh tetrahedra according to a set of specified stencils. The parameters that determine whether a background mesh vertex warps or not were chosen by the authors to optimize the worst possible mesh tetrahedron.

A tetrahedral mesh by isosurface stuffing [69]. A cut shows the graded interior tetrahedra.

Some quadrilateral and hexahedral meshers fill the deep interior of a domain with a regular grid, then bridge the gap by projecting or pulling the vertices on the grid boundary to the domain boundary. See Schneiders [98] for details.

10.3.4 Mesh Improvement

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 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. Domain boundary vertices can be smoothed too, subject to constraints that keep them on the boundary. 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 is called *Laplacian smoothing* because of its interpretation as a Laplacian finite difference operator. It dates back at least to a 1970 paper by Kamel and Eisenstein [62]. It

usually works well for triangular meshes, but it is unreliable for tetrahedra, quadrilaterals, and hexahedra.

A better smoothing method begins by recalling that the Delaunay triangulation of a fixed vertex set is the triangulation that minimizes the volume bounded between a triangulation lifted to the parabolic lifting map and the paraboloid itself. What if the domain's interior vertices are not fixed? Chen and Xu [27] suggest globally smoothing the interior vertices by minimizing the same volume. Alliez, Cohen-Steiner, Yvinec, and Desbrun [1] implemented an iterative tetrahedral mesh improvement method that alternates between this global smoothing step and recomputing the Delaunay triangulation—in other words, it alternates between numerical and combinatorial optimization of the triangulation with respect to the same objective function.

A slower but even better smoother is the nonsmooth optimization algorithm of Freitag, Jones, and Plassmann [48], which can optimize the worst element in a group—for instance, maximizing the minimum dihedral angle among the tetrahedra that share a specified vertex. The figure at right shows smoothing to maximize the minimum plane angle.

The simplest topological transformation is the *edge flip* in a triangular mesh, which replaces two triangles with two different triangles. Also illustrated at right are several analogous transformations for tetrahedra. These are examples of what mathematicians call *bistellar flips*. Similar flips exist for quadrilaterals and hexahedra; see Bern, Eppstein, and Erickson [10] for a list.

Some topological transformations are more complicated than the basic flips. *Edge contraction* removes an edge from the mesh by merging its two endpoints into a single vertex. The triangles or tetrahedra that share the contracted edge are deleted. It is wise to use smoothing to position the new vertex. Edge contraction is useful both to remove elements that have poor quality because an edge is too short, and to coarsen a mesh where its tetrahedra are unnecessarily small.

Edge removal, proposed by Brière de l'Isle and George [19], is a transformation that deletes all the tetrahedra that share a specified edge, then retriangulates the cavity with other tetrahedra chosen to maximize the quality of the worst new tetrahedron. This choice can be efficiently made by a dynamic programming algorithm of Klincsek [64]. The 3-2 and 4-4 flips are the simplest examples of edge removal, which replaces m tetrahedra with $2m - 4$.

The job of a schedule is to search the mesh for tetrahedra that can be improved by local transformations, ideally as quickly as possible. Most schedules use simple hill-climbing optimization: they apply a transformation only if it

improves the mesh. This usually means that the quality of the worst element created by the transformation is better than the quality of the worst element deleted; but in some implementations it means that the average element quality improves. There is a large literature on how to evaluate the quality of an element; see Field [46] for a survey.

Influential mesh improvement schedules include those by Joe [61] and Freitag and Ollivier-Gooch [49]. The latter authors present a schedule that uses flips, edge removal, and optimization-based smoothing to eliminate most poorly shaped tetrahedra, and they offer empirical recommendations about what makes some schedules better than others. This work was extended by Klingner and Shewchuk [65], whose most notable addition is a local transformation that inserts new vertices and thereby eliminates skinny tetrahedra that other transformations cannot touch. Their additions make mesh improvement more reliable: instead of removing most skinny tetrahedra, you can generally remove them all. They report that in their improved test meshes, no dihedral angle is smaller than 31° or larger than 149° .

Mesh improvement is a surprisingly powerful way to introduce anisotropy into a mesh. Löhner and Morgan [74] used flips and vertex insertion to make triangular meshes anisotropic as early as 1986, and the same has subsequently been done for tetrahedral meshes [86, 119]. An exciting new application of mesh improvement is *dynamic meshing*, the modeling of materials undergoing radical changes in shape, such as flow or fracture [21, 78, 119], that a fixed mesh cannot model.

Effective mesh improvement methods also exist for quadrilateral meshes [63]. There does not seem to have been much work on applying hexahedral flips.

Mesh improvement algorithms can be surprisingly slow—often slower than mesh generators—because they make multiple passes over the mesh, and computing element quality is expensive. When simulation accuracy is crucial or a single mesh will be used for many simulations, the cost is worth it.

10.4 Guaranteed-Quality Mesh Generation

The story of guaranteed-quality meshing is an interplay of ideas between Delaunay methods and methods based on grids, quadtrees, and octrees.

Provably good mesh generation began in 1988, when Baker, Grosse, and Rafferty [6] 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 [33] gave a more practical algorithm, based on Delaunay refinement, that guarantees angles between 30° and 120° . (The

input polygon may not have an angle less than 30° .) He 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 . He computes the CDT of the subdivision, then repeatedly inserts a vertex at the circumcenter of a skinny triangle.

At the core of Chew's paper, we find a powerful idea that has appeared in some form in every guaranteed-quality Delaunay refinement algorithm. A skinny element can always be refined away, so there are only two possible outcomes: eventually the algorithm will delete the last skinny element and succeed, or it will run forever, creating skinny elements as fast as it deletes them. If we prove it does not run forever, we prove it succeeds. Can we do that?

We call an element "skinny" only if its circumradius—the radius of its circumcircle—is longer than its shortest edge. When we insert a new vertex at the center of that empty circle, all the newly created edges must be at least as long as the circumradius, and therefore at least as long as the shortest prior edge. It follows that *the Delaunay refinement algorithm never creates an edge shorter than the shortest edge already existing*. This fact places a perpetual lower bound on the distance between two vertices. The algorithm cannot run forever, because first it will run out of places to put new vertices! To complete the proof for Chew's algorithm, we just need to verify that no circumcenter ever falls outside the domain. (Hint: no edge is ever shorter than h ; no circumcenter ever comes closer to the boundary than $h/2$.)

But are those really the skinny elements? In two dimensions, yes. A triangle's circumradius is longer than its shortest edge if and only if its smallest angle is less than 30° . What about tetrahedra?

Dey, Bajaj, and Sugihara [39] generalize Chew's algorithm to generate tetrahedral meshes of convex polyhedral domains. They refine every tetrahedron whose circumradius is more than twice as long as its shortest edge. (The factor of two is lost to ensure that no new vertex is placed too close to the boundary.) Their algorithm is guaranteed to terminate by the same reasoning as for Chew's. But a few bad tetrahedra slip through. Although most thin tetrahedra have large circumradii, there are exceptions: 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. A sliver can have a circumradius shorter than its shortest edge, yet have dihedral angles arbitrarily close to 0° and 180° . Slivers have no two-dimensional analog, but they occur frequently in three-dimensional Delaunay triangulations. In practice, slivers are not difficult to remove [105]—they die when you insert vertices at their circumcenters, just like any other element. But you cannot *prove* that the algorithm will not run forever. Also, slivers removed by refinement sometimes leave undesirably small elements in their wake. 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 these algorithms produce graded meshes. The quadtree algorithm of Bern, Eppstein, and Gilbert [11], discussed in Section 10.3.3.2, 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 [84] improve the algorithm by replacing square quadrants with rhomboids. They produce triangles with angles between 30° and 90° , many of them equilateral.

Mitchell and Vavasis [82] developed an octree algorithm that offers guarantees on dihedral angles (i.e. the very worst slivers are ruled out), grading, and size optimality. The bounds are not strong enough to be satisfying in practice, and are not explicitly stated. Nevertheless, these papers broadened the ambitions of provably good meshing tremendously.

A seminal triangular meshing paper by Jim Ruppert [94] brought guaranteed good grading and size optimality to Delaunay refinement algorithms. Ruppert's was the first guaranteed-quality mesh generation algorithm to be truly satisfying in practice [104]. It accepts nonconvex domains with internal boundaries and produces graded meshes that are not overrefined in practice. It began to grapple with the difficulty of domains with small angles, and it established the theoretical strength of the DELAUNAYREFINEMENT template.

I generalized Ruppert's algorithm to three dimensions [105]. This tetrahedral algorithm has much in common with Dey's, but like Ruppert's, it 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.

Subsequently, provably good Delaunay refinement algorithms have been extended to curved surface meshing [34, 16, 31], anisotropy [68, 31], and domains with curved boundaries [17, 87, 29, 93]. Researchers have also studied how to guarantee a sub-quadratic running time even if the Delaunay triangulation of the input has a quadratic number of tetrahedra [58]. Most of these algorithms are practical as well as provably good.

Only a few results are known on guaranteed quality quadrilateral meshing, and none for hexahedra. Bern and Eppstein [9] use circle packings to quadrilateralize polygons, producing no angle greater than 120° . A tetrahedral Delaunay mesh generated by the software *DelPSC* [29]. Courtesy of Josh Levine.

Atalay, Ramaswami, and Xu [4] use quadtrees to quadrilateralize polygons, producing no angle less than 18.4° .

Today, the two research issues presenting the most difficulty in tetrahedral meshing are sliver removal and domains with small angles.

A method of Chew [35] that randomly perturbs a circumcenter and the *sliver exudation* algorithm of Cheng, Dey, Edelsbrunner, Facello, and Teng [28, 42] both remove the very worst slivers from a Delaunay mesh, but the quality bounds are very weak and are not explicitly stated. Significant provable bounds on dihedral angles (1° or over) are virtually unheard of outside of space-filling or slab-filling tetrahedralizations. There are two exceptions. An algorithm by Labelle [67] is a true hybrid of the Delaunay and octree methods: it refines a Delaunay triangulation by inserting vertices at lattice points near the circumcenters of skinny tetrahedra. The lattice helps to guarantee that deep in the interior of the mesh, all the dihedral angles are between 30° and 135° . These bounds do not hold near the boundary—another argument for combining the advancing front and Delaunay methods.

The *isosurface stuffing* algorithm [69], discussed in Section 10.3.3.2, uses these insights about lattices to mesh volumes bounded by smooth surfaces. Smooth boundaries are just easy enough to permit strong quality guarantees: the dihedral angles of the tetrahedra are all between 10.7° and 164.8° . These bounds are derived by a computer-assisted proof that checks many cases. Unfortunately, it appears to be excruciatingly difficult to extend guarantees like this to domains with sharp corners or sharp ridges.

The problem of domains with small angles is solved for triangular meshes by a remarkably elegant tweak by Miller, Pav, and Walkington [81] that perfects Ruppert's algorithm. In three dimensions, Cheng, Dey, Ramos, and Ray [30] have a satisfying algorithm for meshing polyhedra, but the problem remains difficult for domains with internal boundaries. Existing algorithms, such as one by Rand and Walkington [92], sometimes refine much more than necessary because they must maintain the Delaunay property.

Another research direction deserving investigation is guaranteed-quality mesh improvement. Although sliver exudation can be seen as a provably good method that uses bistellar flips to locally improve a Delaunay mesh, there are no provably good local mesh improvement methods for meshes that are not Delaunay. There seem to be deep questions about how “tangled” a tetrahedralization can be, and whether it can always be fixed by local transformations. The same questions loom for quadrilateral and hexahedral meshes.

Acknowledgments. I thank Mark Beall, Ted Blacker, Josh Levine, David Marcum, Heather Myers, and Mark Shephard for providing meshes. I thank Anand Kulkarni and Martin Wicke for suggesting improvements to drafts of this chapter.

Bibliography

- [1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational Tetrahedral Meshing. *ACM Transactions on Graphics*, 24:617–625, 2005. Special issue on Proceedings of SIGGRAPH 2005.
- [2] Nina Amenta, Sunghee Choi, and Günter Rote. Incremental Constructions con BRIO. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 211–219, San Diego, California, 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] F. Betul Atalay, Suneeta Ramaswami, and Dianna Xu. Quadrilateral Meshes with Bounded Minimum Angle. In *Proceedings of the 17th International Meshing Roundtable*, pages 73–91, Pittsburgh, Pennsylvania, October 2008.
- [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. In Ding-Zhu Du and Frank Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.
- [9] Marshall Bern and David Eppstein. Quadrilateral Meshing by Circle Packing. In *Sixth International Meshing Roundtable*, pages 7–19, Park City, Utah, October 1997.

- [10] Marshall Bern, David Eppstein, and Jeff Erickson. Flipping Cubical Meshes. *Engineering with Computers*, 18(3):173–187, October 2002.
- [11] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. In *31st Annual Symposium on Foundations of Computer Science*, pages 231–241, 1990.
- [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 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.
- [15] Jean-Daniel Boissonnat, David Cohen-Steiner, Bernard Mourrain, Günter Rote, and Gert Vegter. Meshing of Surfaces. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, Mathematics and Visualization, chapter 5, pages 181–229. Springer, 2006.
- [16] Jean-Daniel Boissonnat and Steve Oudot. Provably Good Sampling and Meshing of Surfaces. *Graphical Models*, 67(5):405–451, September 2005.
- [17] 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.
- [18] Adrian Bowyer. Computing Dirichlet Tessellations. *Computer Journal*, 24(2):162–166, 1981.
- [19] Eric Brière de l’Isle and Paul-Louis George. Optimization of Tetrahedral Meshes. In Ivo Babuska, Joseph E. Flaherty, William D. Henshaw, John E. Hopcroft, Joseph E. Oliger, and Tayfun Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, volume 75 of *IMA Volumes in Mathematics and its Applications*, pages 97–127. Springer-Verlag, 1995.
- [20] Kevin Q. Brown. Voronoi Diagrams from Convex Hulls. *Information Processing Letters*, 9(5):223–228, December 1979.
- [21] David Cardoze, Alexandre Cunha, Gary L. Miller, Todd Phillips, and Noel J. Walkington. A Bézier-Based Approach to Unstructured Moving Meshes. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 310–319, Brooklyn, New York, June 2004. Association for Computing Machinery.

- [22] 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.
- [23] 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.
- [24] 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.
- [25] Bernard Chazelle. Convex Partitions of Polyhedra: A Lower Bound and Worst-Case Optimal Algorithm. *SIAM Journal on Computing*, 13(3):488–507, August 1984.
- [26] Bernard Chazelle and Leonidas Palios. Triangulating a Nonconvex Polytope. *Discrete & Computational Geometry*, 5(1):505–526, December 1990.
- [27] Long Chen and Jin-chao Xu. Optimal Delaunay Triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.
- [28] 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.
- [29] Siu-Wing Cheng, Tamal Krishna Dey, and Joshua A. Levine. A Practical Delaunay Meshing Algorithm for a Large Class of Domains. In *Proceedings of the 16th International Meshing Roundtable*, pages 477–494, Seattle, Washington, October 2007.
- [30] Siu-Wing Cheng, Tamal Krishna Dey, Edgar A. Ramos, and Tathagata Ray. Quality Meshing for Polyhedra with Small Angles. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 290–299, Brooklyn, New York, June 2004.
- [31] Siu-Wing Cheng, Tamal Krishna Dey, Edgar A. Ramos, and Rephael Wenger. Anisotropic Surface Meshing. In *Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms*, pages 202–211, Miami, Florida, January 2006.
- [32] L. Paul Chew. Constrained Delaunay Triangulations. *Algorithmica*, 4(1):97–108, 1989.
- [33] L. Paul Chew. Guaranteed-Quality Triangular Meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

- [34] L. Paul Chew. Guaranteed-Quality Mesh Generation for Curved Surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280, San Diego, California, May 1993.
- [35] L. Paul Chew. Guaranteed-Quality Delaunay Meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393, Nice, France, June 1997.
- [36] Kenneth L. Clarkson and Peter W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry*, 4(1):387–421, December 1989.
- [37] Richard Courant, Kurt Friedrichs, and Hans Lewy. Über die Partiellen Differenzengleichungen der Mathematischen Physik. *Mathematische Annalen*, 100:32–74, August 1928.
- [38] Boris Nikolaevich Delaunay. Sur la Sphère Vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.
- [39] 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.
- [40] Rex A. Dwyer. A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations. *Algorithmica*, 2(2):137–151, 1987.
- [41] Rex A. Dwyer. Higher-Dimensional Voronoi Diagrams in Linear Expected Time. *Discrete & Computational Geometry*, 6(4):343–367, 1991.
- [42] Herbert Edelsbrunner and Damrong Guoy. An Experimental Study of Sliver Exudation. In *Tenth International Meshing Roundtable*, pages 307–316, Newport Beach, California, October 2001. Sandia National Laboratories.
- [43] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.
- [44] David Eppstein, John M. Sullivan, and Alper Üngör. Tiling Space and Slabs with Acute Tetrahedra. *Computational Geometry: Theory and Applications*, 27(3):237–255, March 2004.
- [45] Hale Erten and Alper Üngör. Triangulations with Locally Optimal Steiner Points. In *Symposium on Geometry Processing 2007*, pages 143–152, July 2007.
- [46] David A. Field. Qualitative Measures for Initial Meshes. *International Journal for Numerical Methods in Engineering*, 47:887–906, 2000.

- [47] 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.
- [48] Lori A. Freitag, Mark Jones, and Paul E. Plassmann. An Efficient Parallel Algorithm for Mesh Smoothing. In *Fourth International Meshing Roundtable*, pages 47–58, Albuquerque, New Mexico, October 1995.
- [49] 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.
- [50] 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.
- [51] John Alan George. *Computer Implementation of the Finite Element Method*. PhD thesis, Stanford University, Stanford, California, March 1971. Technical report STAN-CS-71-208.
- [52] Paul-Louis George and Houman Borouchaki. *Delaunay Triangulation and Meshing: Application to Finite Elements*. Hermès, Paris, 1998.
- [53] Paul-Louis George, F. Hecht, and E. Saltel. Automatic Mesh Generator with Specified Boundary. *Computer Methods in Applied Mechanics and Engineering*, 92(3):269–288, November 1991.
- [54] 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.
- [55] Carol Hazlewood. Approximating Constrained Tetrahedrizations. *Computer Aided Geometric Design*, 10:67–87, 1993.
- [56] François Hermeline. *Une Methode Automatique de Maillage en Dimension n*. PhD thesis, Université Pierre et Marie Curie, 1980.
- [57] François Hermeline. Triangulation Automatique d’un Polyèdre en Dimension N . *RAIRO Analyse Numérique*, 16(3):211–242, 1982.
- [58] Benoît Hudson, Gary L. Miller, and Todd Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–358, Birmingham, Alabama, September 2006. Springer.
- [59] Antony Jameson, Timothy J. Baker, and Nigel P. Weatherill. Calculation of Inviscid Transonic Flow over a Complete Aircraft. In *Proceedings of the 24th AIAA Aerospace Sciences Meeting*, Reno, Nevada, January 1986. AIAA paper 86-0103.

- [60] Pierre Jamet. Estimations d'Erreur pour des Éléments Finis Droits Presque Dégénérés. *RAIRO Analyse Numérique*, 10(1):43–60, 1976.
- [61] Barry Joe. Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations. *SIAM Journal on Scientific Computing*, 16(6):1292–1307, November 1995.
- [62] H. A. Kamel and K. Eisenstein. Automatic Mesh Generation in Two- and Three-Dimensional Inter-Connected Domains. In *Symposium on High Speed Computing of Elastic Structures*, Les Congrès et Colloques de l'Université de Liège, Liège, Belgium, 1970.
- [63] Paul Kinney. CleanUp: Improving Quadrilateral Finite Element Meshes. In *Sixth International Meshing Roundtable*, pages 449–461, Park City, Utah, October 1997.
- [64] G. T. Klintsek. Minimal Triangulations of Polygonal Domains. *Annals of Discrete Mathematics*, 9:121–123, 1980.
- [65] Bryan Matthew Klingner and Jonathan Richard Shewchuk. Aggressive Tetrahedral Mesh Improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, Seattle, Washington, October 2007.
- [66] Michal Křížek. On the Maximum Angle Condition for Linear Tetrahedral Elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, April 1992.
- [67] François Labelle. Sliver Removal by Lattice Refinement. *International Journal of Computational Geometry and Applications*, 18(6):509–532, December 2008.
- [68] François Labelle and Jonathan Richard Shewchuk. Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 191–200, San Diego, California, June 2003.
- [69] François Labelle and Jonathan Richard Shewchuk. Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles. *ACM Transactions on Graphics*, 26(3):57.1–57.10, July 2007. Special issue on Proceedings of SIGGRAPH 2007.
- [70] Charles L. Lawson. Software for C^1 Surface Interpolation. In John R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, New York, 1977.
- [71] Der-Tsai Lee and Arthur K. Lin. Generalized Delaunay Triangulations for Planar Graphs. *Discrete & Computational Geometry*, 1:201–217, 1986.

- [72] 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.
- [73] 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.
- [74] Rainald Löhner and Ken Morgan. Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations. In *AIAA 24th Aerospace Sciences Meeting*, Reno, Nevada, 1986. AIAA paper 86-0499.
- [75] 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.
- [76] David L. Marcum. Unstructured Grid Generation Using Automatic Point Insertion and Local Reconnection. In Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors, *Handbook of Grid Generation*, chapter 18. CRC Press, 1999.
- [77] David L. Marcum and Nigel P. Weatherill. Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection. In *Twelfth AIAA Applied Aerodynamics Conference*, Colorado Springs, Colorado, June 1994. AIAA paper 94-1926.
- [78] Sean Mauch, Ludovic Noels, Zisu Zhao, and Raúl A. Radovitzky. Lagrangian Simulation of Penetration Environments via Mesh Healing and Adaptive Optimization. In *Proceedings of the 25th Army Science Conference*, November 2006.
- [79] Dimitri J. Mavriplis. An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness. *Journal of Computational Physics*, 117:90–101, 1995.
- [80] Eleftherios 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, Steven E. Pav, and Noel J. Walkington. When and Why Ruppert’s Algorithm Works. *International Journal of Computational Geometry and Applications*, 15(1):25–54, February 2005.
- [82] Scott A. Mitchell and Stephen A. Vavasis. Quality Mesh Generation in Three Dimensions. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 212–221, 1992.

- [83] Michael Murphy, David M. Mount, and Carl W. Gable. A Point-Placement Strategy for Conforming Delaunay Tetrahedralization. In *Proceedings of the Eleventh Annual Symposium on Discrete Algorithms*, pages 67–74, January 2000.
- [84] Friedhelm Neugebauer and Ralf Diekmann. Improved Mesh Generation: Not Simple but Good. In *Fifth International Meshing Roundtable*, pages 257–270, Pittsburgh, Pennsylvania, October 1996.
- [85] Van Phai Nguyen. Automatic Mesh Generation with Tetrahedral Elements. *International Journal for Numerical Methods in Engineering*, 18:273–289, 1982.
- [86] C. C. Pain, A. P. Umpheby, C. R. E. de Oliveira, and A. J. H. Goddard. Tetrahedral Mesh Optimisation and Adaptivity for Steady-State and Transient Finite Element Calculations. *Computer Methods in Applied Mechanics and Engineering*, 190:3771–3796, 2001.
- [87] Steven E. Pav and Noel J. Walkington. Delaunay Refinement by Corner Lopping. In *Proceedings of the 14th International Meshing Roundtable*, pages 165–181, San Diego, California, September 2005.
- [88] 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.
- [89] Jaime Peraire, Joaquim Peiró, and Ken Morgan. Advancing Front Grid Generation. In Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors, *Handbook of Grid Generation*, chapter 17. CRC Press, 1999.
- [90] 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.
- [91] V. T. Rajan. Optimality of the Delaunay Triangulation in $\mathbb{R}^d$. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 357–363, North Conway, New Hampshire, June 1991.
- [92] Alexander Rand and Noel Walkington. Collars and Intestines: Practical Conformal Delaunay Refinement. In *Proceedings of the 18th International Meshing Roundtable*, pages 481–497, Salt Lake City, Utah, October 2009.
- [93] Laurent Rineau and Mariette Yvinec. Meshing 3D Domains Bounded by Piecewise Smooth Surfaces. In *Proceedings of the 16th International Meshing Roundtable*, pages 443–460, Seattle, Washington, October 2007.

- [94] Jim Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, May 1995.
- [95] Jim Ruppert and Raimund Seidel. On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra. *Discrete & Computational Geometry*, 7(3):227–253, 1992.
- [96] 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.
- [97] H. Samet. The Quadtree and Related Hierarchical Data Structures. *Computing Surveys*, 16:188–260, 1984.
- [98] Robert Schneiders. Quadrilateral and Hexahedral Element Meshes. In Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors, *Handbook of Grid Generation*, chapter 21. CRC Press, 1999.
- [99] E. Schönhardt. Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Mathematische Annalen*, 98:309–312, 1928.
- [100] William J. Schroeder and Mark S. Shephard. Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation. *International Journal for Numerical Methods in Engineering*, 26(11):2503–2515, November 1988.
- [101] Raimund Seidel. Voronoi Diagrams in Higher Dimensions. Diplomarbeit, Institut für Informationsverarbeitung, Technische Universität Graz, 1982.
- [102] Michael Ian Shamos and Dan Hoey. Closest-Point Problems. In *16th Annual Symposium on Foundations of Computer Science*, pages 151–162, Berkeley, California, October 1975.
- [103] Mark S. Shephard and Marcel K. Georges. Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique. *International Journal for Numerical Methods in Engineering*, 32(4):709–749, September 1991.
- [104] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, Berlin, May 1996. From the First ACM Workshop on Applied Computational Geometry.
- [105] Jonathan Richard Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *Proceedings of the Fourteenth Annual Symposium on*

- Computational Geometry*, pages 86–95, Minneapolis, Minnesota, June 1998.
- [106] Jonathan Richard Shewchuk. What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Eleventh International Meshing Roundtable*, pages 115–126, Ithaca, New York, September 2002.
- [107] Jonathan Richard Shewchuk. Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 181–190, San Diego, California, June 2003.
- [108] Jonathan Richard Shewchuk. General-Dimensional Constrained Delaunay Triangulations and Constrained Regular Triangulations I: Combinatorial Properties. *Discrete & Computational Geometry*, 39(1–3):580–637, March 2008.
- [109] Hang Si. Constrained Delaunay Tetrahedral Mesh Generation and Refinement. *Finite Elements in Analysis and Design*, 46:33–46, January–February 2010.
- [110] Matthew L. Staten, Steven J. Owen, and Ted D. Blacker. Paving & Plastering: Progress Update. In *Proceedings of the 15th International Meshing Roundtable*, pages 469–486, Birmingham, Alabama, September 2006.
- [111] J. L. Synge. *The Hypercircle in Mathematical Physics*. Cambridge University Press, New York, 1957.
- [112] Joe F. Thompson. The National Grid Project. *Computer Systems in Engineering*, 3(1–4):393–399, 1992.
- [113] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors. *Handbook of Grid Generation*. CRC Press, 1999.
- [114] Alper Üngör. Off-Centers: A New Type of Steiner Points for Computing Size-Optimal Quality-Guaranteed Delaunay Triangulations. *Computational Geometry: Theory and Applications*, 42(2):109–118, February 2009.
- [115] Shayne Waldron. The Error in Linear Interpolation at the Vertices of a Simplex. *SIAM Journal on Numerical Analysis*, 35(3):1191–1200, 1998.
- [116] David F. Watson. Computing the n -dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *Computer Journal*, 24(2):167–172, 1981.
- [117] Nigel P. Weatherill. Delaunay Triangulation in Computational Fluid Dynamics. *Computers and Mathematics with Applications*, 24(5/6):129–150, September 1992.

- [118] Nigel P. Weatherill and O. Hassan. Efficient Three-Dimensional Grid Generation Using the Delaunay Triangulation. In Ch. Hirsch, J. Périaux, and W. Kordulla, editors, *Proceedings of the First European Computational Fluid Dynamics Conference*, pages 961–968, Brussels, Belgium, September 1992.
- [119] Martin Wicke, Daniel Ritchie, Bryan M. Klingner, Sebastian Burke, Jonathan R. Shewchuk, and James F. O’Brien. Dynamic Local Remeshing for Elastoplastic Simulation. *ACM Transactions on Graphics*, 29(3), July 2010. Special issue on Proceedings of SIGGRAPH 2010.
- [120] 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.
- [121] Mark A. Yerry and Mark S. Shephard. Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique. *International Journal for Numerical Methods in Engineering*, 20(11):1965–1990, November 1984.