Mesh Generation

Marshall Bern * Paul Plassmann †

1 Introduction

A *mesh* is a discretization of a geometric domain into small simple shapes, such as triangles or quadrilaterals in two dimensions and tetrahedra or hexahedra in three. Meshes find use in many application areas. In geography and cartography, meshes give compact representations of terrain data. In computer graphics, most objects are ultimately reduced to meshes before rendering. Finally, meshes are almost essential in the numerical solution of differential equations arising in physical simulation. In this chapter, we concentrate on algorithms for this last application, assuming an audience including both practitioners such as engineers and theoreticians such as computational geometers and numerical analysts.

1.1 Types of Geometric Domains

We divide the possible inputs first according to dimension—two or three. We distinguish four types of planar domains, as shown in Figure 1. For us, a *simple polygon* includes both boundary and interior. A *polygon with holes* is a simple polygon minus the interiors of some other simple polygons; its boundary has more than one connected component. A *multiple domain* is more general still, allowing internal boundaries; in fact, such a domain may be any planar straight-line graph in which the infinite face is bounded by a simple cycle. Multiple domains model objects made from more than one material. *Curved domains* allow sides that are algebraic curves such as splines. As in the first three cases, collectively known as *polygonal domains*, curved domains may or may not include holes and internal boundaries.

Three-dimensional inputs have analogous types. A *simple polyhedron* is topologically equivalent to a ball. A *general polyhedron* may be multiply connected, meaning that it is topologically equivalent to a solid torus or some other higher genus solid; it may also have cavities, meaning that its boundary may have more than one connected component. We do assume, however, that at every point on the boundary of a general polyhedron a sufficiently small sphere encloses one connected piece of the polyhedron's interior and one connected piece of its exterior. Finally, there are *multiple polyhedral domains*—general polyhedra with internal boundaries—and *three-dimensional curved domains*, which typically have boundaries defined by spline patches.

The construction and modeling of domain geometry lies outside the scope of this chapter, and we shall simply assume that domains are given in some sort of boundary representation, without specifying the exact form of this representation. For concreteness, the reader may imagine a linked list representation for simple polygons and polygons with holes, and

*Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304. bern@parc.xerox.com
†Mathematics and Computer Science Division, Argonne National Laboratory.

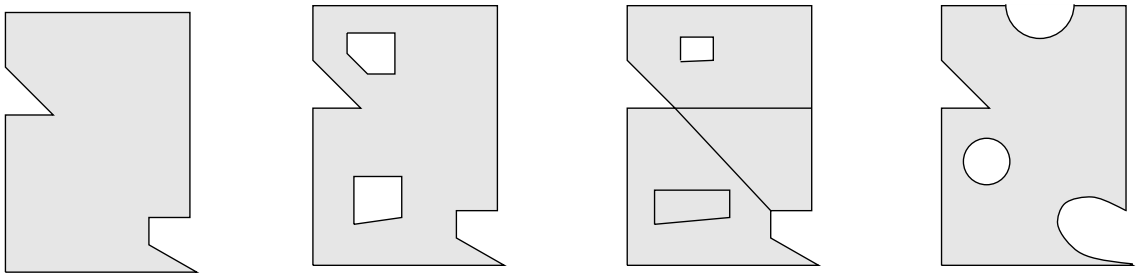

Figure 1. Types of two-dimensional inputs: simple polygon, polygon with holes, multiple domain, and curved domain.

a doubly connected edge list [88] or quad-edge data structure [59] for multiple domains. Curved domains can be defined by adding spline curves [14] to these data structures. For three-dimensional domains, there are analogous but more complicated boundary representations such as the winged edge data structure [42, 58]; spline patches can be defined either over quadrilaterals [14] or triangles [99].

There may be some advantages to domain representations other than boundary representations. For example, a constructive solid geometry formula for the domain may help in mesh generation. Solid modeling and mesh generation are currently separate steps, performed by software from different sources, but we expect greater integration in the future.

1.2 Types of Meshes

A *structured* mesh is one in which all interior vertices are topologically alike. An *unstructured* mesh is one in which vertices may have arbitrarily varying local neighborhoods. A *hybrid* mesh is formed by a number of small structured meshes combined in an overall unstructured pattern.

In general, structured meshes offer simplicity and easy data access, while unstructured meshes offer more convenient mesh adaptivity and a better fit to complicated domains. (As might be expected, hybrid meshes fall somewhere in between.) Moreover, certain numerical methods are more compatible with one type of mesh than another, as we shall explain in Section 2. We shall discuss unstructured mesh generation at much greater length than structured mesh generation, both because the unstructured approach seems to gaining ground and because it is more closely connected to computational geometry.

The division between structured and unstructured meshes usually extends to the shape of the elements: two-dimensional structured meshes typically use quadrilaterals, while unstructured meshes use triangles. In three dimensions the analogous element shapes are *hexahedra*, meaning topological cubes, and tetrahedra. There is, however, no essential reason for structured and unstructured meshes to use different element shapes. In fact it is possible to subdivide elements in order to convert between triangles and quadrilaterals and between tetrahedra and hexahedra. Figure 2 shows the transformations in the two-dimensional case. The transformation from triangles to quadrilaterals or tetrahedra to hexahedra is related to *barycentric subdivision*, which divides each face with its center of mass.

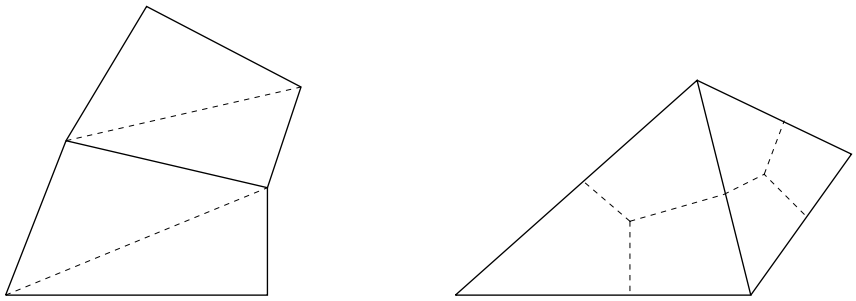

Figure 2. (a) Triangulating quadrilaterals. (b) Subdividing triangles to form quadrilaterals.

1.3 Organization

Section 2 gives a brief survey of numerical methods and their impact on mesh generation. Sections 3 and 4 cover structured and unstructured two-dimensional meshes. Section 5 discusses three-dimensional hexahedral mesh generation, including structured, hybrid, and unstructured approaches. Finally, Section 6 describes three-dimensional unstructured tetrahedral mesh generation.

We shall explain the fundamental computational geometry results as they arise within a larger context; however, Section 4 concludes with a separate theoretical discussion, because unstructured planar mesh generation is especially rich in interesting geometric questions. Throughout this article, we emphasize practical issues; an earlier survey by Bern and Eppstein [21] emphasized theoretical results. Although there is inevitably some overlap between these two surveys, we intend them to be complementary.

Mesh generation has a huge literature and we cannot hope to cover all of it. There are excellent references on numerical methods [108, 31], structured mesh generation [32, 57, 113], and unstructured mesh generation [21, 56]. There are also several nice Web sites [85, 97, 102, 124] on mesh generation.

2 Background on Numerical Methods

Scientific computing seeks accurate discrete models for continuous physical phenomena. We can divide the process into three interdependent steps: problem formulation, mesh generation, and equation solution. In this section, we discuss discretization and solution methods and their impact on mesh generation.

2.1 Discrete Formulation

There are a number of approaches to the discrete approximation of partial differential equations modeling a physical system. Here we briefly review the standard discretization methods: finite-difference, finite-element, and finite-volume. Although these methods result in linear systems of similar structure, the desired characteristics of meshes for these methods may be quite dissimilar.

Finite-difference methods [112] are the simplest to implement. This approach replaces continuous differential operators with difference approximations. Consider the partial differential equation

$$\mathcal{L}u = f \tag{1}$$

where u is a function of position and possibly also of time. We seek an approximate solution of (1) on some geometric domain Ω. A standard finite-difference approach replaces the differential operator $\mathcal{L}$ with a discrete *stencil*. Writing $u_k = u(x_k)$ for the value of u at mesh vertex position x_k, the action of the stencil at x_i can be represented by

$$\mathcal{L}u(x_i) = \sum_{k \in adj(x_i)} A_{ik} u_k \ ,$$

where $adj(x_i)$ is the set of points adjacent to x_i in the mesh and A_{ik} is a set of weights. The right-hand side of (1) can also be discretized yielding a system of linear equations

$$\sum_{k=1}^{n} A_{ik} u_k = f_i \tag{2}$$

to be solved for the unknowns u_k. Because the finite-difference stencil gives nonzero weight only to neighboring vertices, this system will be quite sparse.

It is convenient to use the same stencil throughout the mesh. This restriction simplifies not only the software but also the mathematics; the convergence properties of a particular stencil can be analyzed quite easily by Taylor series expansion. A finite-difference stencil gives a more accurate approximation of a continuous operator when the edges meeting at vertices are nearly orthogonal. For these two reasons, finite-difference approaches usually rely on structured meshes topologically equivalent to regular grids.

Finite-element methods [108] have become enormously popular in the engineering and scientific communities in part because they overcome many of the limitations of the finite-difference approach. The essential idea is to replace the continuous function $u(x)$ with the finite-dimensional approximation $\bar{u}(x) = \sum_{k=1}^{n} a_k \phi_k(x)$, where the ϕ_k are basis functions with local support. These basis functions are typically low-order polynomials, so that the action of the differential operator, $\mathcal{L}\phi_k$, can easily be computed. Because the approximation $\bar{u}(x)$ is defined everywhere on the domain, an analysis of convergence can be made in a continuous norm instead of pointwise as in the finite-difference method.

The finite-element method obtains a discrete approximation by demanding that the differential equations be satisfied for some set of test functions $\psi_i(x)$ by the relations

$$\begin{aligned} \int_\Omega (\mathcal{L}\bar{u})\psi_i &= \int_\Omega \bar{f}\psi_i \\ &= f_i \ . \end{aligned}$$

The efficiency of the method depends on the size of the set I_{ik} of elements where both the basis function ϕ_k and the test function ψ_i are nonzero. This set is usually quite small, consisting only of the elements adjacent to the vertex which corresponds to the unknown coefficient a_k. An implementation of the finite-element method reduces to the computation of the factors

$$\begin{aligned} A_{ik} &= \int_\Omega (\mathcal{L}\phi_k)\psi_i \\ &= \sum_{e \in I_{ik}} \int_e (\mathcal{L}\phi_k)\psi_i \ . \end{aligned}$$

The contribution from each element in I_{ik} can be computed and summed (or assembled) to obtain A_{ik}. The result of this process is again a sparse linear system of equations, of the same form as (2), that can be solved for the unknowns a_k.

Finite-element methods are typically no more complicated on unstructured meshes than on structured meshes. Furthermore, there is no real advantage to mesh edges meeting orthogonally. Elements of poor aspect ratio however, can seriously degrade accuracy. Early theoretical results [37] showed convergence of finite-element methods as elements shrink, assuming aspect ratios remain bounded. Babuška and Aziz [4] showed convergence in two dimensions assuming that angles are bounded away from 180°, a strictly weaker condition. The generalization of this result to three dimensions assumes dihedrals bounded away from 0°, thereby showing that the needle in Figure 19 is permissible.

In two dimensions, the Delaunay triangulation of a point set has the desirable property that it maximizes the minimum angle. Moreover, the Delaunay triangulation gives an M-matrix—diagonally dominant with negative off-diagonal entries—for Laplacian operators [15, 38]. M-matrices are exactly those matrices that satisfy a discrete maximum principle; this desirable property rules out oscillation of the numerical method. These results do not generalize to three dimensions; in this case nonobtuse face angles are a sufficient but not a necessary condition for an M-matrix.

Finite-volume methods are motivated by the need to conserve certain physical quantities in a discrete model. The infinitesimal version of a conservation law is of the form

$$\frac{d\rho}{dt} + \nabla \cdot \mathbf{\Phi} = 0 \ ,$$

where ρ is the density and $\mathbf{\Phi}$ is the flux of the conserved quantity. In order to maintain the same physical conservation law on a discrete level, the finite-volume method defines small volumes called *control volumes*, and requires that on each control volume Ω_v

$$\frac{d}{dt} \int_{\Omega_v} \rho + \int_{\partial \Omega_v} \mathbf{\Phi} \cdot \mathbf{n} = 0 \, ,$$

where $\mathbf{n}$ is the normal to the surface of the volume. *Cell-centered* control volumes are usually identical to mesh elements, while *vertex-centered* control volumes form a dual mesh with one cell for each vertex of the original mesh.

The finite volume method with vertex-centered control volumes matches fluid dynamics problems well, because pressure and velocity—in some sense dual variables—can be represented at centers and vertices of volumes, respectively. There are several ways to define vertex-centered control volumes. Two regular grids may be overlaid, staggered by half an element. Or, in the case of unstructured meshes, the Delaunay triangulation may be used as the mesh, and its dual—the Voronoi diagram—used to define control volumes. A two-dimensional mesh without obtuse angles [7, 24] (or a three-dimensional mesh with fully self-centered tetrahedra [19]) gives particularly nice control volumes, one in which control volume edges cross element edges only at right angles.

2.2 Solution Methods

The solution of the sparse linear system is usually the most computationally demanding phase of the entire modeling process. Solution methods include direct factorization and preconditioned iterative methods. These methods can vary dramatically in required storage and computational cost for different problems. Moreover, the discrete formulation and mesh generation steps can greatly influence the efficacy of a solution method. Higher-order basis functions in the finite element method allow the use of a coarser mesh, but give a denser

linear system. Poorly shaped mesh elements can give an ill-conditioned linear system, which will be much harder to solve.

Direct factorization methods, such as sparse Cholesky or LU factorization, can be very expensive, especially for three-dimensional problems. Direct methods, however, are more robust than iterative methods, and the computational cost can be amortized when the factorization is reused to solve for more than one right-hand side. The theoretical efficiency of certain direct methods depends upon the existence of small graph separators for the mesh. Any planar graph admits separators of size $O(n^{1/2})$; reasonable three-dimensional meshes admit separators of size $O(n^{2/3})$ [77].

Iterative methods have proved effective in solving the linear systems arising in physical modeling. There are numerous iterative methods varying in ease of implementation, storage requirements, existence of software, and theoretical convergence bounds. Most large problems cannot be effectively solved without the use of preconditioning; a popular approach involves an incomplete factorization. Rather than computing the exact factors for the matrix $A = LU$, approximate factors are computed such that $A \approx \tilde{L}\tilde{U}$ and the preconditioned system

$$\tilde{L}^{-1}A\tilde{U}^{-1}(\tilde{U}u) = \tilde{L}^{-1}f$$

is solved iteratively. Ideally, the incomplete factors should be easy to compute and require a modest amount of storage, and the condition number of the preconditioned system should be much better than the original system.

Multigrid methods can achieve the ultimate goal of iterative methods, convergence in $O(1)$ iterations, for certain classes of problems. These methods use a sequence of meshes, graded from fine (small elements) to coarse (large elements). Mesh generation techniques such as regular bisection (Section 4.4 below) naturally give a sequence of nested meshes suitable for multigrid methods.

Domain decomposition methods [105] represent something of a hybrid between iterative and direct approaches. This approach divides the domain into possibly overlapping small domains; it solves the subproblems on the small domains directly, but iterates to the global solution in which neighboring subproblem solutions agree. This approach enjoys some of the superior convergence properties of multigrid methods, while imposing less stringent requirements on the mesh generator. In fact, the domain is often partitioned so that subproblems admit structured meshes.

3 Structured Two-Dimensional Meshes

Structured meshes offer simplicity and efficiency. A structured mesh requires significantly less memory—say a factor of three less—than an unstructured mesh with the same number of elements, because array storage can define neighbor connectivity implicitly. A structured mesh can also save time: to access neighboring cells when computing a finite-difference stencil, software simply increments or decrements array indices. Compilers produce quite efficient code for these operations; in particular, they can optimize the code for vector machines.

On the other hand, it can be difficult or impossible to compute a structured mesh for a complicated geometric domain. Furthermore, a structured mesh may require many more elements than an unstructured mesh for the same problem, because elements cannot grade in size as rapidly. These two difficulties can often be circumvented by a hybrid

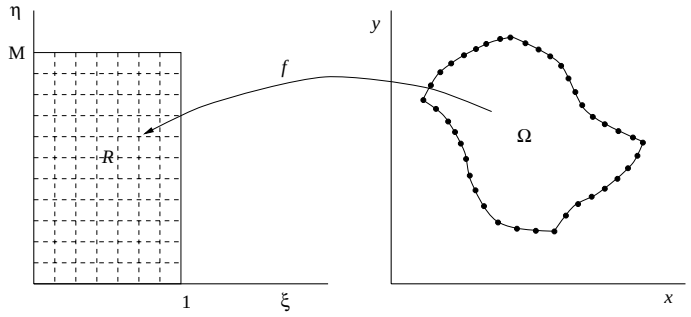

Figure 3. The conformal mapping f from the domain Ω, defined by a boundary discretization, to a rectangle R. The inverse of the mapping maps a grid on R onto a structured mesh for Ω.

structured/unstructured approach, which decomposes a complicated domain into patches supporting structured grids. Hybrid approaches, however, are typically not fully automatic, requiring some user guidance in the decomposition step; hence we shall not discuss them at much length. (See Section 5.2 for an example of a three-dimensional hybrid mesh.)

Structured mesh generation can be roughly classified into hand-generated and other elementary approaches, algebraic or interpolation methods, and PDE or variational methods [114]. The PDE approach [32, 67] solves partial differential equations in order to map the domain Ω onto another domain with a convenient coordinate system. In this section, we discuss an elliptic PDE approach with a connection to the classical topic of conformal mapping.

A mapping of a region Ω of the complex plane is *conformal* if it preserves angles; in other words, the angle between any two curves intersecting at a point $z \in \Omega$ is preserved by the mapping. The Riemann mapping theorem states that for any topological disk Ω, there exists a conformal mapping f that takes the interior of Ω one-to-one onto the interior of any other topological disk (such as the unit disk or square). There is an obvious connection to mesh generation: a conformal mapping of Ω onto a square grid induces a structured mesh on Ω with the property that element angles tend towards 90° in the limit of an increasingly fine discretization.

Unfortunately, the Riemann mapping theorem only proves the existence of a conformal mapping; it does not give an algorithm. Let us write $z = x + \imath y$ and consider the the complex function $f(z) = \xi(x, y) + \imath\eta(x, y)$. If f is analytic—as a conformal f will be, assuming $f'(z) \neq 0$—then it satisfies the Cauchy-Riemann equations: $\xi_x = \eta_y$ and $\xi_y = -\eta_x$. Thus the functions ξ and η must each be harmonic and satisfy Laplace's equation, so that $\nabla^2\xi = 0$ and $\nabla^2\eta = 0$. If f is conformal, its inverse is as well; therefore, x and y as functions of ξ and η are also harmonic and satisfy $\nabla^2 x = 0$ and $\nabla^2 y = 0$.

Consider the regions Ω and R in Figure 3, and assume we already have a discretization of the boundary of Ω. (Finding a suitable boundary discretization may itself be a difficult task.) The obvious algorithm is to solve $\nabla^2 x = 0$ and $\nabla^2 y = 0$, assuming x and y are given on the boundary of R. However, this approach may not work. One may obtain poorly shaped or even inverted elements as shown in Figure 4(a). The problem is that the solutions x and y may be harmonic, but not harmonic conjugate (i.e., satisfy the Cauchy-Riemann equations).

The algorithm can be partially mended by obtaining a better estimate for M, the rectangle height implied by the discretization of the boundary of Ω. If we scale the original

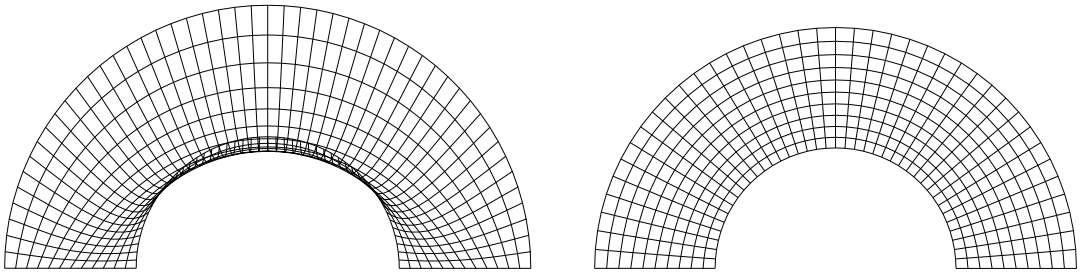

Figure 4. The grid on left was obtained by solving (3) with unit aspect ratio, resulting in a folded-over mesh. On the right, a more appropriate aspect ratio has been chosen.

coordinates of the rectangle (ξ, η) onto a square with coordinates (μ, ν) with the mapping $\mu = \xi$ and $\nu = \eta/M$ we obtain the system

$$M^2 x_{\mu\mu} + x_{\nu\nu} = 0 \qquad\qquad M^2 y_{\mu\mu} + y_{\nu\nu} = 0. \tag{3}$$

From the first-order Cauchy-Reimann equations we have

$$M^2 = (x_\nu^2 + y_\nu^2)/(x_\mu^2 + y_\mu^2).$$

Barfield [13] obtained reasonable nonoverlapping meshes by estimating the average value of the right hand side of the above equation and using this value for M. One can think of M as an average aspect ratio for the original domain; if ideal aspect ratio varies significantly over the domain one can also make this number a function of position. This approach can be successful for many physical problems, and can be improved significantly if the generated grid is smoothed as a postprocessing step. This approach can also be extended to three dimensions—where the Riemann mapping theorem no longer holds—by the addition of another "average aspect ratio" term.

Although the approach just sketched works quite well for some domains, it is does not guarantee the generation of a valid mesh. It is interesting that the inverse problem, solving the harmonic equations

$$\begin{aligned} \mu_{xx} + \mu_{yy} &= 0 \\ \nu_{xx} + \nu_{yy} &= 0 \end{aligned} \tag{4}$$

does guarantee a solution with no inverted elements and a nonvanishing Jacobian [47, 106]. Solving the problem in this form is more difficult, because it requires a discretization of the domain for which we want to find a grid. However, the system can be inverted to form the nonlinear elliptic system [33]

$$\begin{aligned} \alpha x_{\mu\mu} - 2\beta x_{\mu\nu} + \gamma x_{\nu\nu} &= 0 \\ \alpha y_{\mu\mu} - 2\beta y_{\mu\nu} + \gamma y_{\nu\nu} &= 0 \ , \end{aligned}$$

where

$$\begin{aligned} \alpha &= x_\nu^2 + y_\nu^2 \\ \beta &= x_\mu x_\nu + y_\mu y_\nu \\ \gamma &= x_\mu^2 + y_\mu^2 \ . \end{aligned}$$

Software designed to solve these systems often includes an additional source term on the right-hand sides of the harmonic systems in (4) to control the local point spacing in the domain [114].

In the case that Ω is a simple polygon, the Schwarz-Christoffel formula provides an explicit form for the conformal mappings from the unit disk D to Ω. Such a mapping can in turn be used to find conformal mappings from Ω to a square or rectangle. The Schwarz-Christoffel formula, however, does not seem to be widely used in mesh generation, perhaps because true conformal mapping does not allow local control of point spacing.

Let the points in the complex plane defining the polygon (in counterclockwise order) be $z_1, \ldots, z_n$, the interior angles at these points be $\alpha_1, \ldots, \alpha_n$, and define the normalized angles as $\beta_k = \alpha_k/\pi - 1$. Using $\omega_1, \ldots, \omega_n$ as the preimages of $z_1, \ldots, z_n$ on the edge of the disk, the Schwarz-Christoffel formula gives the form of the conformal mapping as

$$f(\omega) = A + B \int_0^{\omega} \prod_{k=1}^{n} (1 - \xi/\omega_k)^{\beta_k} \, d\xi. \tag{5}$$

There are several programs available to solve for the unknown ω_k values: SCPACK by Trefethen [115], the SC Toolbox by Driscoll [46], and CRDT by Driscoll and Vavasis [45]. One difficulty in the numerical solution is "crowding", enormous variation in spacing between the ω_k points. The latest, and apparently best, Schwarz-Christoffel algorithm, CRDT, overcomes this difficulty by repeatedly remapping so that no crowding occurs near the points being evaluated.

4 Unstructured Two-Dimensional Meshes

We have already mentioned the advantages of unstructured meshes: flexibility in fitting complicated domains, rapid grading from small to large elements, and relatively easy refinement and derefinement.

Unlike structured mesh generation, unstructured mesh generation has been part of mainstream computational geometry for some years. Well-studied geometric constructions such as Delaunay triangulation are central to unstructured mesh generation. We consider three principled approaches to unstructured mesh generation in some detail; these approaches use the Delaunay triangulation, constrained Delaunay triangulation, and quadtrees. In the fourth and fifth sections we discuss mesh refinement and improvement. In the final section, we describe some geometric problems abstracted from unstructured mesh generation.

4.1 Delaunay Triangulation

Our first approach to unstructured mesh generation partitions the task into two phases: placement of Steiner points, followed by triangulation. If the placement phase is smart enough, the triangulation phase can be especially simple, considering only the input vertices and Steiner points and ignoring the input edges.

The placement phase typically places points along the domain boundary before placing points in the interior. The boundary should be lined with enough Steiner points that the Delaunay triangulation of all vertices will *conform* to the domain. For a polygonal domain Ω, this means that each edge of Ω must be the union of edges in the Delaunay triangulation.

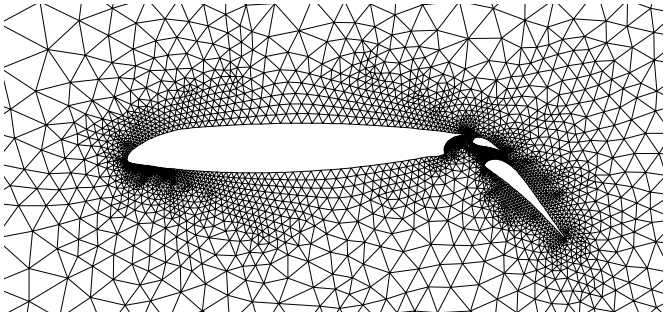

Figure 5. Delaunay triangulation of points placed by an advancing front. (T. Barth)

For applications such as aerodynamics, this subproblem—called *conforming Delaunay triangulation*—usually takes care of itself, because other spacing constraints, such as approximating a spline boundary or resolving small details in the air flow, predominate. Applications that involve oddly shaped domains—for example, stress analysis of machine parts—require an explicit solution. A solution due to Saalfeld [96] lines the edges of Ω with a large number of Steiner points, uniformly spaced except near the endpoints. A more efficient solution [83] covers the edges of Ω by disks that do not overlap other edges. Edelsbrunner and Tan [50] gave the best theoretical result, an algorithm that uses $O(n^3)$ Steiner points for an n-vertex multiple domain.

There are several approaches to placing interior Steiner points. One approach combines the vertices from a number of structured meshes [74]. This method offers local control over element shapes and orientations, enabling, for example, highly stretched, "anisotropic" elements in laminar regions of a viscous flow simulation. On the other hand, this method has some difficulties with complicated geometries, and points may end up poorly spaced where two blocks meet. Spacing can be improved by filtering unwanted points and by mesh smoothing.

Another approach to point placement, called *advancing front*, adds Steiner points in successive layers, working in from the domain boundary [8, 82]. This approach can also achieve anisotropy, either by setting the spacing between close-in layers much smaller than the spacing between points within layers, or by computing the Delaunay triangulation in a stretched space [41, 34]. The advancing front approach may space points improperly where fronts collide, but as in the first method, poorly shaped elements can usually be corrected by smoothing. Before moving on to our last placement approach, we remark that many advancing front mesh generators [60] place the triangles themselves, rather than just the Steiner points. Such an approach gives more direct control over the triangles near the boundary, at the cost of reduced flexibility where fronts collide.

The last placement approach is the most robust, but gives the least control. In this approach, interior points are chosen at random according to some distribution [76, 119], which can be interpolated from a (possibly very coarse and nonconforming) quadtree or "background" triangulation. The distribution may also include user-defined point sources at regions of special interest An independent random sample is likely to force some badly shaped triangles [23], so the generator should oversample and then filter out points too close to previously chosen points. There are also deterministic methods that achieve essentially the same effect as random sampling with filtering; these methods [26, 104] define birth and death rules that depend on the density of neighboring points.

The triangulation phase uses the well-known *Delaunay triangulation*. The Delaunay triangulation of a point set $S = \{s_1, s_2, \ldots, s_n\}$ is defined by the *empty circle condition*: a triangle $s_i s_j s_k$ appears in the Delaunay triangulation $DT(S)$ if and only if its circumcircle encloses no other points of S. There is an exception for points in special position: if an empty circle passes through four or more points of S, we may triangulate these points—*complete* the triangulation—arbitrarily. So defined, $DT(S)$ is a triangulation of the convex hull of S. For our purposes, however, we can discard all triangles that fall outside the original domain Ω.

There are a number of practical Delaunay triangulation algorithms [54]. We describe only one, called the *edge flipping* algorithm, because it is most relevant to our subsequent discussion. Its worst-case running time of $O(n^2)$ is suboptimal, but it performs quite well in practice. The edge flipping algorithm starts from any triangulation of S and then locally optimizes each edge. Let e be an internal (non-convex-hull) edge and Q_e be the triangulated quadrilateral formed by the triangles sharing e. Quadrilateral Q_e is *reversed* if the two angles without the diagonal sum to more than 180°, or equivalently, if each triangle circumcircle contains the opposite vertex. If Q_e is reversed, we "flip" it by exchanging e for the other diagonal.

```
compute an initial triangulation of S
place all internal edges onto a queue
while the queue is not empty do
    remove the first edge e
    if quadrilateral Q_e is reversed then
        flip it and add the outside edges of Q_e to the queue endif
endwhile
```

An initial triangulation can be computed by a sweep-line algorithm. This algorithm adds the points of S by x-coordinate order. Upon each addition, the algorithm walks around the convex hull of the already-added points, starting from the rightmost previous point and adding edges until the slope reverses. The following theorem [43] guarantees the success of edge flipping: a triangulation in which no quadrilateral is reversed must be a completion of the Delaunay triangulation.

4.2 Constrained Delaunay triangulation

There is another way, besides conforming Delaunay triangulation, to extend Delaunay triangulation to polygonal domains. The *constrained Delaunay triangulation* of a (possibly multiple) domain Ω does not use Steiner points, but instead redefines Delaunay triangulation in order to force the edges of Ω into the triangulation.

A point p is *visible* to a point q in Ω if the relatively open line segment pq lies within Ω and does not intersect any edges or vertices of Ω. The constrained Delaunay triangulation $CDT(\Omega)$ contains each triangle with an *empty* circumcircle, where empty now means that the circle does not contain any vertices of Ω visible to points interior to the triangle. The visibility requirement means that external proximities, where Ω wraps around to nearly touch itself, have no effect. Figure 6 provides an example; here vertex v is not visible to any point in the interior of triangle abc.

The edge flipping algorithm can be generalized to compute the constrained Delaunay triangulation, only this time we do not allow edges of Ω onto the queue. Obtaining an

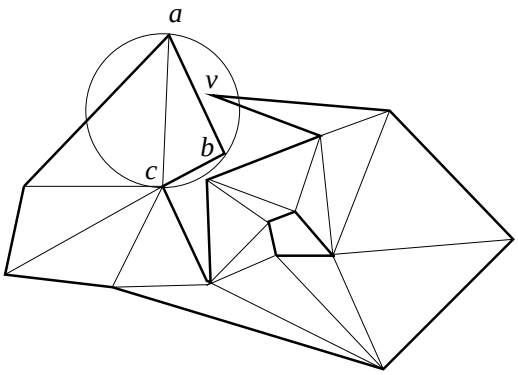

Figure 6. The constrained Delaunay triangulation of a polygon with holes.

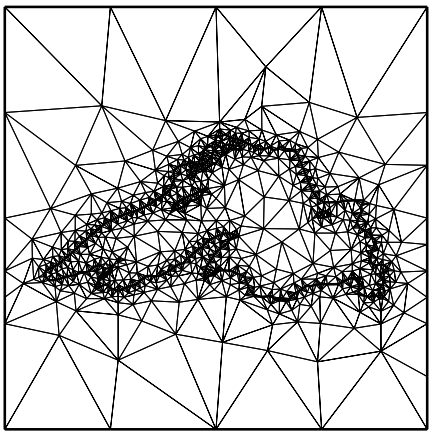

Figure 7. A mesh computed by Ruppert's algorithm. (J. Ruppert)

initial triangulation is somewhat more difficult for polygonal domains than for point sets. The textbook by Preparata and Shamos [88] describes an $O(n \log n)$-time algorithm for computing an initial triangulation. This algorithm first adds edges to Ω to subdivide it into easy-to-triangulate "monotone" faces.

Ruppert [95], building on work of Chew [36], gave a mesh-generation algorithm based on constrained Delaunay triangulation. (Shewchuk [102, 103] has subsequently made further improvements to this algorithm and made an implementation available on the Web.) This algorithm computes the constrained Delaunay triangulation at the outset and then adds Steiner points to improve the mesh, thus uniting the two phases of the approach described in the last section. In choosing Ruppert's approach, the user gives up some control over point placement, but obtains a more efficient mesh with fewer and rounder triangles.

The first step of Ruppert's mesh generator cuts off all vertices of the domain Ω at which the interior angle measures less than 45°. The cutting line at such a vertex v should not introduce a new small feature to Ω; it is best to cut off an isosceles triangle whose base is about halfway from v to its closest visible neighbor. If v has degree greater than two, as might be the case in a multiple domain, then the bases of the isosceles triangles around v should match up so that no isosceles triangle receives a Steiner point on one of its legs.

Next the algorithm computes the constrained Delaunay triangulation of the modified

domain. The algorithm then goes through the loop given below. The last line of the loop repairs a constrained Delaunay triangulation after the addition of a new Steiner point c. To accomplish this step, there is no need to recompute the entire triangulation. The removed old triangles are exactly those with circumcircles containing c, which can be found by searching outwards from the triangle that contains c, and the new triangles that replace the removed triangles must all be incident to the new vertex c.

while there exists a triangle t with an angle smaller than 20° **do**
    let c be the center of t's circumcircle
    if c lies within the diameter semicircle of a boundary edge e **then**
        add the midpoint m of e
    else add c **endif**
    recompute the constrained Delaunay triangulation
endwhile

The loop is guaranteed to halt with all angles larger than 20°. At this point, the cut-off isosceles triangles are returned to the domain, and the mesh is complete. Ruppert's algorithm comes with a strong theoretical guaranty: all new angles (not present in the input) are greater than 20°, and the total number of triangles in the mesh is at most a constant times the minimum number of triangles in any such *no-small-angle* mesh. To prove this efficiency result, Ruppert shows that each triangle in the final mesh is within a constant factor of the *local feature size* at its vertices. The local feature size at point $p \in \Omega$ is defined to be the radius of the smallest circle centered at p that contains two connected components of the boundary; this is a spacing function intrinsic to the domain.

4.3 Quadtrees

A quadtree mesh generator [6, 22, 122] starts by enclosing the entire domain Ω inside an axis-aligned square. It splits this *root* square into four congruent squares, and continues splitting squares recursively until each minimal—or *leaf*—square intersects Ω in a simple way. Further splits may be dictated by a user-defined spacing function or balance condition. Quadtree squares are then warped and cut to conform to the boundary. A final triangulation step gives an unstructured triangular mesh.

We now describe a particular quadtree mesh generator due to Bern, Eppstein, and Gilbert [22]. As first presented, the algorithm assumes that Ω is a polygon with holes; however, the algorithm can be extended to multiple and even to curved domains. In fact, the quadtree approach handles curved domains more gracefully than the Delaunay and constrained Delaunay approaches, because the splitting phase can automatically adapt to the curvature of enclosed boundary pieces.

The algorithm of Bern et al. splits each quadtree square b until each connected component of $b \cap D$ has only one connected piece of Ω's boundary, with at most one vertex. It "clones" squares that intersect Ω in more than one connected component—an idea due to Mitchell and Vavasis [80])—and assigns one connected component to each of the clones, which are superimposed in the sense of Riemann sheets. The algorithm then splits squares near vertices of Ω a couple more times, so that each vertex lies within a buffer zone of equal size squares.

Next the mesh generator imposes a *balance* condition: no square should be adjacent to one less than one-half its size. This causes more splits to propagate across the quadtree,

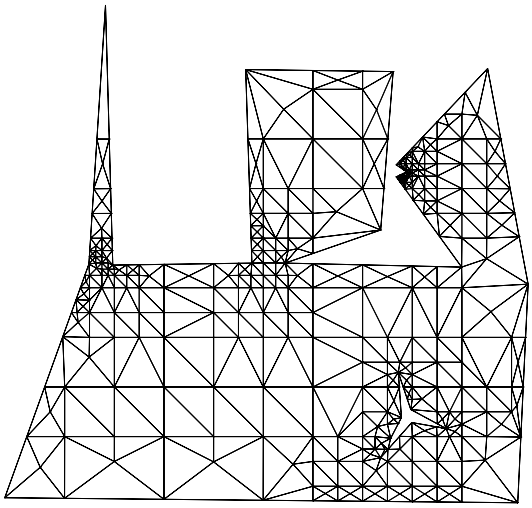

Figure 8. A mesh computed by a quadtree-based algorithm. (S. Mitchell)

increasing the total number of leaf squares by a constant factor (at most 8). Squares are then *warped* to conform to the domain Ω. Various warping rules work; we give just one possibility. In the following pseudocode, $|b|$ denotes the side length of square b.

for each vertex v of Ω **do**
    let y be the closest quadtree vertex
    move y to v
endfor
for each leaf square b still crossed by an edge e **do**
    move the vertices of b that are closer than $|b|/4$ to e to their closest points on e
endfor
discard faces of the warped quadtree that lie outside Ω

Finally, the cells of the warped quadtree are triangulated so that all angles are bounded away from 0°. Figure 8 gives a mesh computed by a variant of the quadtree algorithm. This figure shows that cloning ensures appropriate element sizes around holes and "almost holes". Notice that a quadtree-based mesh exhibits preferred directions—horizontal and vertical. If this artifact poses a problem, mesh improvement steps can be used to redistribute element orientations.

The quadtree algorithm enjoys the same efficiency guaranty as Ruppert's algorithm. In fact, the quadtree algorithm was the first to be analyzed in this way [22].

4.4 Mesh Refinement and Derefinement

Adaptive mesh refinement places more grid points in areas where the error in the solution is known or suspected to be large. Local error estimates based on a solution computed on an initial mesh are known as *a posteriori* error estimates [5] and can be used to determine which elements should be refined. For elliptic problems these estimators asymptotically bound the true error and can be computed locally using only the information on an element [120].

One approach to mesh refinement [34] iteratively inserts extra vertices into the triangulation, typically at edge bisectors or triangle circumcenters as in Section 4.2. New vertices

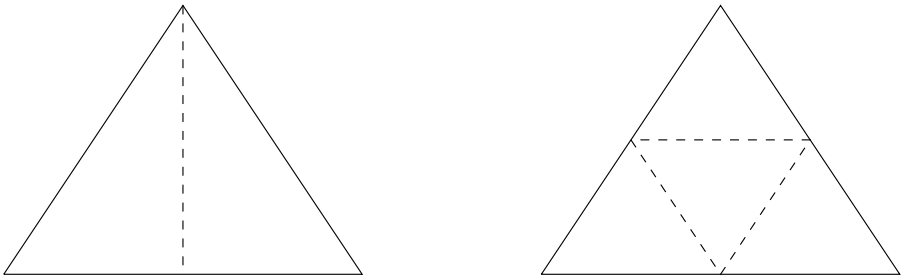

Figure 9. A triangle divided by (a) bisection, and (b) regular refinement.

along the boundaries of curved domains should be computed using the curved boundary rather than the current straight edge, thereby giving a truer approximation of the domain as the mesh refines. Insertion may be followed by edge flipping (Section 4.1) and mesh smoothing (Section 4.5).

This approach gives a finer mesh, but not an *edge conforming* refinement of the original mesh, meaning a mesh that includes the boundaries of the original triangles. Edge conforming refinements are especially convenient for the multigrid method (Section 2.2). To compute such a refinement, we turn to another approach. This approach *splits* triangles in need of refinement, by adding the midpoints of sides. The pseudocode below gives the overall approach.

$k = 0$
solve the differential equation on the initial mesh T_0
estimate the error on each triangle
while the maximum error on a triangle is larger than the given tolerance **do**
    based on error estimates, mark a set of triangles S_k to refine
    $\star$ divide the triangles in S_k, and any other triangles necessary to form T_{k+1}
    solve the differential equation on T_{k+1}
    estimate the error on each triangle
    $k = k + 1$
endwhile

There are a number of popular alternatives for step $\star$, in which the current mesh T_k is adaptively refined. In *regular refinement* [9, 10], the midpoints of the sides of a marked triangle are connected, as in Figure 9(b), to form four similar triangles. Unmarked triangles that received two or three midpoints are split in the same way. Unmarked triangles that received only one midpoint are *bisected* by connecting the midpoint to the opposite vertex as in Figure"9(a). Before the next iteration of $\star$, bisected triangles are glued back together and marked for refinement; this precaution guarantees that each triangle in T_{k+1} will either be similar to a triangle in T_0 or be the bisection of a triangle similar to a triangle in T_0. Hence regular refinement, regardless of the number of times through the refinement loop, produces a mesh with minimum angle at least half the minimum angle in T_0, and the angles in T_{k+1} are bounded away from 0 and π.

Rivara [91, 92, 93] proposed several alternatives for step $\star$ based on triangle bisection. One method refines each marked triangle by cutting from the opposite vertex to the midpoint of the longest edge. Neighboring triangles are now *invalid*, meaning that one side

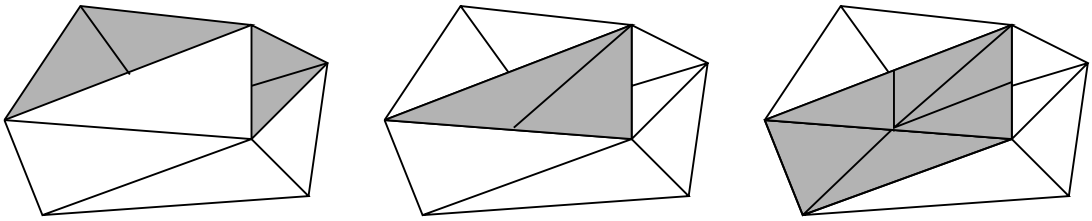

Figure 10. The bisection algorithm bisects marked triangles before invalid triangles. Invalid children of refined triangles are split to their subdivision points

contains an extra vertex; these triangles are then bisected in the same way. Bisections continue until there are no remaining invalid triangles. Refinement can propagate quite far from marked triangles; however, propagation cannot fall into an infinite loop, because along a propagation path each bisected edge is longer than its predecessor. This approach, like the previous one, produces triangles that fall into only a finite number of *similarity classes*, and the minimum angle is again at least half the smallest angle in T_0. Quite often longest-edge refinement actually improves angles.

A second Rivara refinement method is given in the pseudocode below and illustrated in Figure 10. This method does not always bisect the longest edge, so bisections tend to propagate less, yet the method retains the same final angle bound as the first Rivara method.

$i = 0$
$Q_i = S_k$ { Q *always denotes triangles not yet refined*}
$R_i = \emptyset$ { R *always denotes children of refined triangles*}
while $(Q_i \cup R_i) \neq \emptyset$ **do**
    bisect each triangle in Q_i across its longest edge
    bisect each triangle in R_i across its subdivided edge
    add all invalid triangles in $\cup_{j=0}^{i} Q_j$ to R_{i+1}
    add all other invalid triangles to Q_{i+1}
    $i = i + 1$
endwhile

We now discuss the reverse process: coarsening or derefinement of a mesh. This process helps reduce the total number of elements when tracking solutions to time-varying differential equations. Coarsening can also be used to turn a single highly refined mesh into a sequence of meshes for use in the multigrid method [84].

Figure 11 shows a sequence of meshes computed by a coarsening algorithm. The algorithm marks a set of vertices to deleted from the fine mesh, eliminates all marked vertices, and then retriangulates the mesh. The resulting mesh is *node conforming*, meaning that every vertex of the coarse mesh appears in the fine mesh, but not edge conforming. One difficulty is that the shapes of the triangles degrade as the mesh is coarsened due to increasing disparity between the interior and boundary point densities. Meshes produced by refinement methods, such as regular refinement, are typically much easier to coarsen than are less hierarchical meshes such as Delaunay triangulations. Teng, Talmor, and Miller [75] have recently devised a coarsening algorithm that produces a sequence of bounded-aspect-ratio, node-conforming meshes of approximately minimum depth.

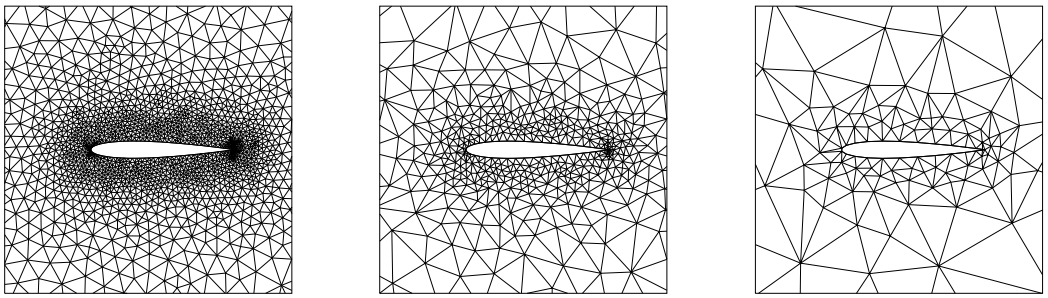

Figure 11. A sequence of meshes used by the multigrid method for solving the linear systems arising in modeling airflow over an airfoil. (C. Ollivier-Gooch)

4.5 Mesh Smoothing

Mesh smoothing adjusts the locations of mesh vertices in order to improve element shapes and overall mesh quality [2, 30, 53, 86]. Mesh smoothing has a certain advantage over other mesh improvement methods such as edge flipping: the topology of the mesh remains invariant, thus preserving important data structures such as the nonzero pattern of the associated sparse linear system.

Laplacian smoothing [53, 71] is the most commonly used smoothing technique. This method sweeps over the entire mesh several times, repeatedly moving each adjustable vertex to the arithmetic average of the vertices adjacent to it. A variation weights each adjacent vertex by the total area of the elements around it. Laplacian smoothing is computationally inexpensive, but it does not in general guarantee improvement in element quality. In fact, Laplacian smoothing can even invert an element, unless the algorithm performs an explicit check before moving a vertex.

Another class of smoothing algorithms uses optimization techniques to determine new vertex locations. Both global and local optimization-based smoothing offer guaranteed mesh improvement and validity. Global techniques simultaneously adjust all unconstrained vertices; such an approach involves an optimization problem as large as the number of unconstrained vertices, and consequently, is computationally expensive [30, 86]. Local techniques adjusts vertices one by one—or an independent set of vertices in parallel [55]—resulting in a cost more comparable to Laplacian smoothing.

Figure 12 shows the results of a local optimization-based smoothing algorithm [55] applied to a mesh generated adaptively during the finite element solution of the linear elasticity equations on a two-dimensional rectangular domain with a hole. The mesh on the left was generated using the bisection algorithm for refinement; the edges from the coarse mesh are still evident after many levels of refinement. The mesh on the right was generated by a similar algorithm, only with vertex locations optimized after each refinement step. Overall, the global minimum angle has improved from 11.3° to 21.7° and the average minimum element angle from 35.7° to 41.1°. Only two to three optimization steps were necessary to find approximately optimal vertex positions.

4.6 Theoretical Questions

We have mentioned some theoretical results—conforming Delaunay triangulation, no-small-angle triangulation—in context. In this section, we describe some other theoretical work

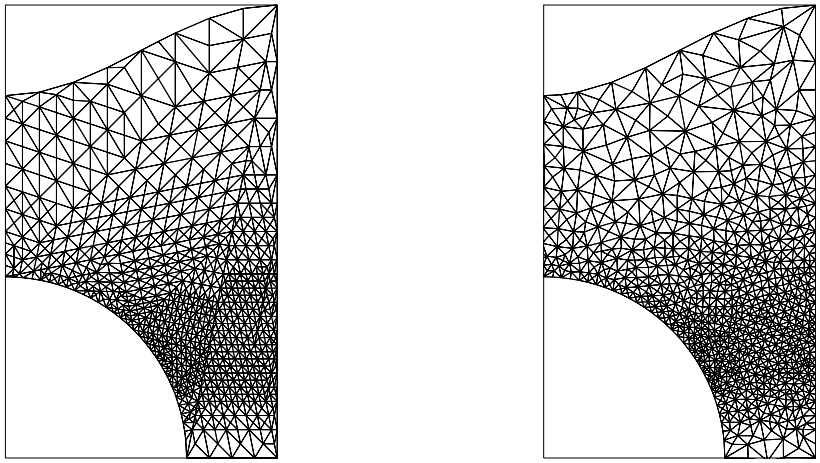

Figure 12. (a) A mesh resulting from bisection refinement without smoothing. (b) The same mesh after local optimization-based smoothing.

related to mesh generation.

4.6.1 Optimal Triangulation

Computational geometers have studied a number of problems of the following form: given a planar point set or polygonal domain, find a best triangulation, where "best" is judged according to some specific quality measure. Quality measures relevant to mesh generation include maxmin angle, minmax angle, minmax edge length, and minimum total edge length. If the input is a simple polygon, most optimal triangulation problems are solvable by dynamic programming, but if the input is a point set, polygon with holes, or multiple domain, these problems become much harder.

The Delaunay triangulation—constrained Delaunay triangulation in the case of polygonal domains—optimizes any quality measure that is improved by flipping a reversed quadrilateral; this statement follows from the theorem that a triangulation without reversed quadrilaterals must be Delaunay. Quality criteria in this category include the first one mentioned above—maxmin angle—along with a number of more esoteric optimizations, such as minmax circumcircle radius, minmax enclosing circle radius, and minimum "roughness" of a piecewise-linear interpolating surface [90].

Edge flipping can also be used as a general optimization heuristic. For example, we can flip quadrilaterals to minimize the maximum angle instead of maximizing the minimum angle. For this quality measure, edge flipping works reasonably well [51], but it does not always find an exact optimum. A more general local improvement method called *edge insertion* [20, 51] exactly solves the minmax angle problem, as well as several other minmax optimization problems.

Edge insertion starts from an arbitrary triangulation and repeatedly inserts *candidate* edges. If minmax angle is the goal, the candidate edge e subdivides the maximum angle; in general the candidate edge is always incident to a "worst vertex" of a worst triangle. The algorithm then removes the edges that are crossed by e, forming two polygonal holes alongside e. Holes are retriangulated by repeatedly removing *ears* (triangles with two sides

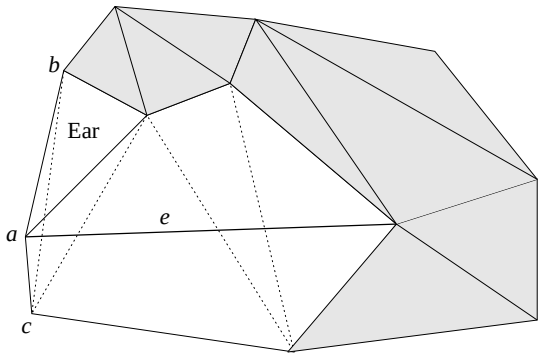

Figure 13. Edge insertion retriangulates holes by removing sufficiently good ears.

on the boundary, as shown in Figure 13) with maximum angle smaller than the old worst angle $\angle cab$. If retriangulation runs to completion, then the overall triangulation improves and edge bc is eliminated as a future candidate. If retriangulation gets stuck, then the overall triangulation is returned to its state before the insertion of e, and e is eliminated as a future candidate. Each candidate insertion takes time $O(n)$, giving a total running time of $O(n^3)$.

compute an initial triangulation with all $\binom{n}{2}$ edge slots unmarked
while $\exists$ an unmarked edge e cutting the worst vertex a of worst triangle abc **do**
    add e and remove all edges crossed by e
    try to retriangulate by removing ears better than abc
    if retriangulation succeeds **then** mark bc
    else mark e and undo e's insertion **endif**
endwhile

Edge insertion can compute the minmax "eccentricity" triangulation or the minmax slope interpolating surface [20] in time $O(n^3)$. By inserting candidate edges in a certain order and saving old partial triangulations, the running time can be improved to $O(n^2 \log n)$ for minmax angle [51] and maxmin triangle height.

We close with some results for the other two optimization criteria mentioned in the introductory paragraph: minmax edge length and minimum total length. Edelsbrunner and Tan [49] showed that a triangulation of a point set that minimizes the maximum edge must contain the edges of a minimum spanning tree. The tree divides the input into simple polygons, which can be filled in by dynamic programming, giving an $O(n^3)$-time algorithm (improvable to $O(n^2)$ with more work). Whether a triangulation minimizing total edge length—"minimum weight triangulation"—can be solved in polynomial time is still open. The most promising approach [44] incrementally computes a set of edges that must appear in the triangulation. If the required edges form a connected spanning graph, then the triangulation can be completed with dynamic programming as in the minmax problem.

4.6.2 Steiner Triangulation

The optimal triangulation problems just discussed have limited applicability to mesh generation, since they address only triangulation and not Steiner point placement. Because exact Steiner triangulation problems seem to be intractable, typical theoretical results on

Steiner triangulation prove either an approximation bound such as the guaranties provided by the mesh generators in Sections 4.2 and 4.3, or an order of complexity bound such as Edelsbrunner and Tan's $O(n^3)$ algorithm for conforming Delaunay triangulation.

The mesh generators in Sections 4.2 and 4.3 give constant-factor approximation algorithms for what we may call the *no-small-angle* problem: triangulate a domain Ω using a minimum number of triangles, such that all new angles are bounded away from 0°. The provable constants tend to quite large—in the hundreds—although the actual performance seems to be much better. The number of triangles in a no-small-angle triangulation depends on the geometry of the domain, not just on the number of vertices n; an upper bound is given by the sum of the aspect ratios of triangles in the constrained Delaunay triangulation.

We can also consider the *no-large-angle* problem: triangulate Ω using a minimum number of triangles, such that all new angles are bounded away from 180°. The strictest bound on large angles that does not imply a bound on small angles is *nonobtuse triangulation*: triangulate a domain Ω such that the maximum angle measures at most 90°. Moreover, a nonobtuse mesh has some desirable numerical and geometric properties [7, 117]. Bern, Mitchell, and Ruppert [24] recently developed a circle-based algorithm for nonobtuse triangulation of polygons with holes; this algorithm gives a triangulation with $O(n)$ triangles, regardless of the domain geometry. Figure 14 shows the steps of this algorithm: the domain is packed with nonoverlapping disks until each uncovered region has either 3 or 4 sides; radii to tangencies are added in order to split the domain into small polygons; and finally small polygons are triangulated with right triangles, without adding any new subdivision points.

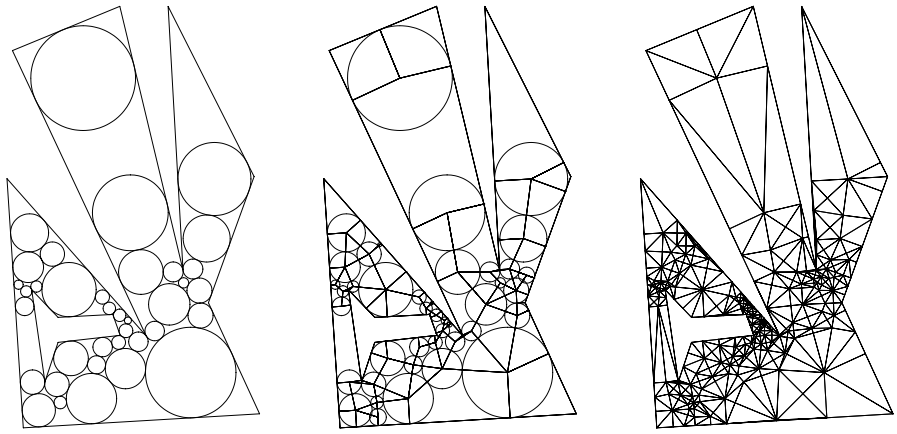

Figure 14. Steps in circle-based nonobtuse triangulation.

It is currently unknown whether multiple domains admit polynomial-size nonobtuse triangulations. Mitchell [78], however, gave an algorithm for triangulating multiple domains using $O(n^2 \log n)$ triangles with maximum angle 157.5°. Tan [109] improved the maximum angle bound to 132° and the complexity to the optimal $O(n^2)$.

5 Hexahedral Meshes

Mesh generation in three dimensions is not as well developed as in two. There are a number of reasons for this lag: lack of standard data representations for three-dimensional domains, greater software complexity, and—most relevant to this article—some theoretical difficulties.

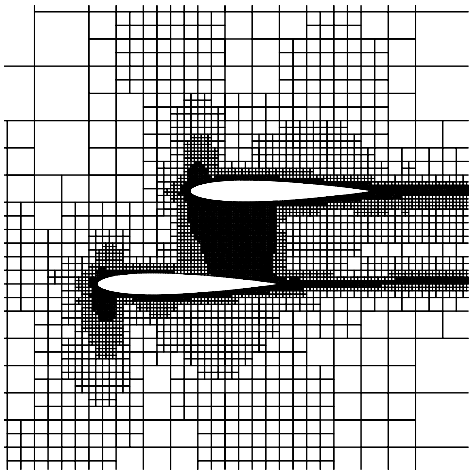

Figure 15. A two-dimensional Cartesian mesh for a biplane wing. (W. Coirier)

This section and the next one survey approaches to three-dimensional mesh generation. We have divided this material according to element shape, hexahedral or tetrahedral. This classification is not completely strict, as many hexahedral mesh generators use triangular prisms and tetrahedra in a pinch. (Careful implementations of numerical methods can in fact handle degenerate hexahedra such as prisms [62, 63].) In this section, we describe three approaches to hexahedral mesh generation that vary in their degree of strictness and in the amount of structure in their outputs.

5.1 Cartesian Meshes

We start with a recently developed "quick and dirty" approach to hexahedral mesh generation. The Cartesian approach offers simple data structures, explicit orthogonality of mesh edges, and robust and straightforward mesh generation. The disadvantage of this approach is that it uses non-hexahedral elements around the domain boundary, which require special handling.

A *Cartesian mesh* is formed by cutting a rectangular box into eight congruent boxes, each of which is split recursively until each minimal box intersects the domain Ω in a simple way or has reached some small target size. (This construction is essentially the same as an octree, described in Section 6.2.) Requiring neighboring boxes to differ in size by at most a factor of two ensures appropriate mesh grading.

Boxes cut by the boundary are classified into a number of patterns by determining which of their vertices lie interior and exterior to Ω. Each pattern corresponds to a different type of non-hexahedral element. Boxes adjacent to ones half their own size can similarly be classified as non-hexahedral elements, or alternatively the solution value at their subdivision vertices can be treated as implicit variables using Lagrange multipliers [1].

Recent fluid dynamics simulations have used Cartesian meshes quite successfully in both finite element and finite volume formulations [40, 39, 123]. The approach can be adapted even to very difficult meshing problems. For example, Berger and Oliger [18] and Berger and Colella [17] have developed adaptive Cartesian-based methods for rotational flows and flows with strong shocks.

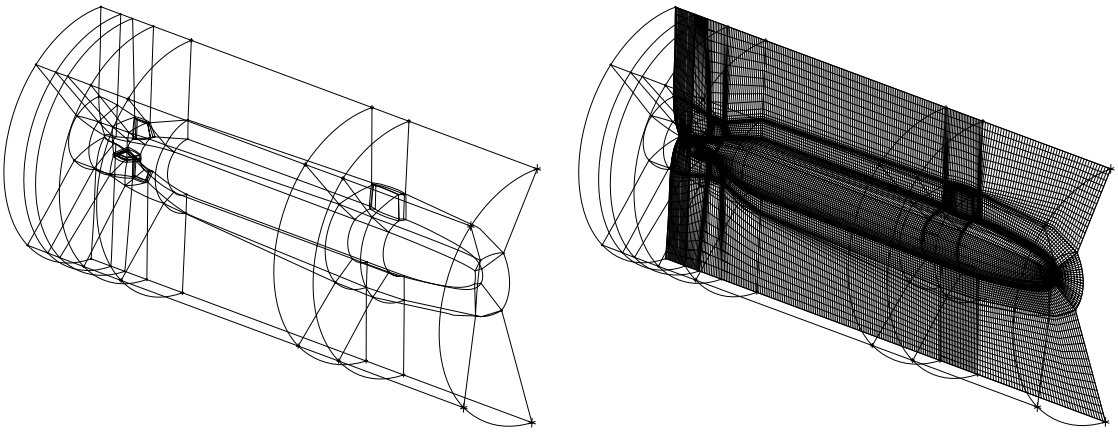

Figure 16. A multiblock hexahedral mesh of a submarine, showing (a) block structure, and (b) a vertical slice through the mesh. (ICEM CFD)

5.2 Multiblock Meshes

A *multiblock mesh* contains a number of small structured meshes that together form a large unstructured mesh. Typically, a user must supply the topology of the unstructured mesh, but the rest of the process is automated. Figure 16 shows a multiblock mesh created by ICEM Hexa, a system developed by ICEM CFD Engineering. In this system the user controls the placement of the block corners, and then the mesh generator projects the implied block edges onto domain curves and surfaces automatically.

5.3 Unstructured Hexahedral Meshes

Hexahedral elements retain some advantages over tetrahedral elements even in unstructured meshes. Hexahedra fit man-made objects well, especially objects produced by CAD systems. The edge directions in a box-shaped hexahedron often have physical significance; for example, hexahedra show a clear advantage over tetrahedra for a stress analysis of a beam [16]. The face normals of a box meet at the center of the element; this property can be used to define control volumes for finite volume methods. Observe, however, that these advantages are not inherent to hexahedra, but rather are properties of box-shaped elements, which degrade as the element grows less rectangular. Thus it will not suffice to generate an unstructured hexahedral mesh by transforming a tetrahedral mesh.

Armstrong et al. [3] are currently developing an unstructured hexahedral mesh generator based on the medial axis transform. The *medial axis* of a domain is the locus of centers of spheres that touch the boundary in two or more faces. This construction is closely related to the Voronoi diagram of the faces of the domain; Srinivasan et al. [107] have previously applied this construction to two-dimensional unstructured mesh generation. The medial axis is a natural tool for mesh generation, as advancing fronts started from faces meet at the medial axis in the limit of small, equal-sized elements, By precomputing this locus, a mesh generator can more gracefully handle the junctures between sections of the mesh.

Tautges and Mitchell [110, 111] are currently developing an all-hexahedral mesh generator based on an algorithm called *whisker weaving*. The basic strategy is an advancing

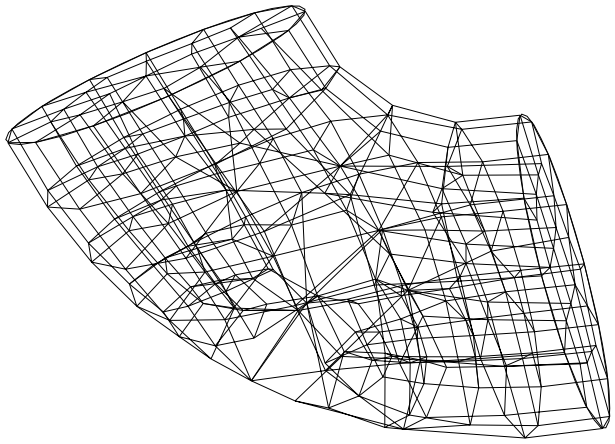

Figure 17. A hexahedral mesh computed by the whisker weaving algorithm.

front approach that fixes the topology of the mesh before the geometry. The dual of the mesh, which has one vertex per hexahedron and one edge per quadrilateral face, provides a convenient way to represent the mesh topology.

Whisker weaving starts from a quadrilateral surface mesh, which can itself be generated by an advancing-front generator within each face [25]. The algorithm forms the planar dual of the surface mesh, and then finds closed loops in the planar dual around the surface of the polyhedron. Each loop will represent the boundary of a *sheet* of hexahedra in the eventual mesh. As the algorithm runs, it fills each sheet from the boundary inwards with a quadrilateral mesh.

A *whisker* is an "open" edge in the planar dual of a sheet's mesh; in other words, a whisker is dual to a segment along the sheet's advancing front. Whisker weaving repeatedly joins the open ends of a triple of whiskers that have the property that each pair is dual to adjacent edges on some sheet's front. This step advances the front on three sheets simultaneously; it corresponds to filling a corner formed by three quadrilaterals with a new hexahedron. Weaving three whiskers may produce invalid connectivity, such as neighboring hexahedra sharing two faces, but the algorithm detects and repairs these problems as they arise.

The advancing front approach to hexahedral meshing raises an interesting theoretical question: which quadrilateral surface meshes can be extended to hexahedral volume meshes? Mitchell [79] and Thurston answered this question in a topological sense by showing that any simple-polyhdron surface mesh with an even number of quadrilaterals can be extended to a volume mesh formed by (possibly curved) topological cubes. In follow-on work, Eppstein [52] showed that $O(n)$ curved cubes suffice for a simple polyhedron with n vertices, and made substantial progress towards the corresponding geometric result in which hexahedra must have straight sides.

6 Tetrahedral Meshes

Tetrahedra claim several important advantages over hexahedra: unique interpolation from vertices to interior, greater flexibility in fitting complicated domains, and greater convenience for refinement and derefinement. In order to realize the last two of these advantages,

tetrahedral meshes are almost always unstructured.

Most of the approaches to unstructured triangular mesh generation that we surveyed in Section 4 can be generalized to tetrahedral mesh generation, but not without some new difficulties. Before describing mesh generators based on octrees and Delaunay triangulation, we discuss three theoretical obstacles to unstructured tetrahedral meshing—ways in which $\mathbb{R}^3$ differs from $\mathbb{R}^2$.

First, not all polyhedral domains can be triangulated without Steiner points. Figure 18(a) gives an example of a non-tetrahedralizable polyhedron, a twisted triangular prism in which rectangular face has been triangulated so that it pokes in towards the interior. None of the top three vertices is visible through the interior to all three of the bottom vertices; hence no tetrahedron formed by the vertices of this polyhedron can include the bottom face. Chazelle [35] gave a quantitative bad example, shown in Figure 18(b). This polyhedron includes $\Omega(n)$ cuts that nearly meet at a doubly-ruled curved surface; any triangulation of this polyhedron must include $\Omega(n^2)$ Steiner points and $\Omega(n^2)$ tetrahedra.

Bad examples such as these appear to rule out the possibility of generalizing constrained Delaunay triangulation to three dimensions.

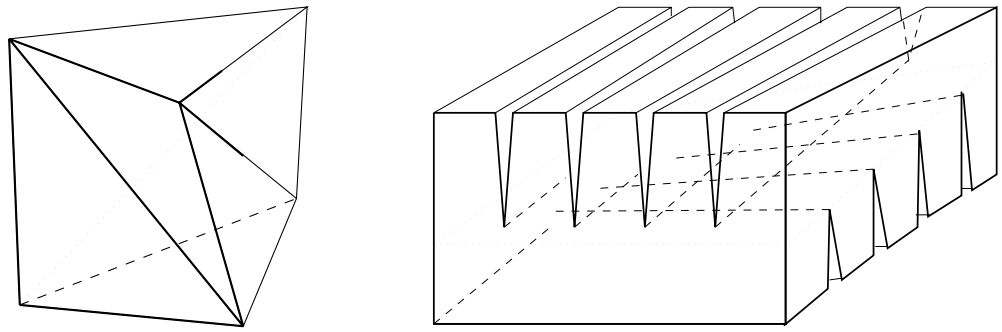

Figure 18. (a) Schoenhardt's twisted prism cannot be tetrahedralized without Steiner points. (b) Chazelle's polyhedron requires $\Omega(n^2)$ Steiner points.

Second, the very same domain may be tetrahedralized with different numbers of tetrahedra. For example, the cube can be triangulated with either 5 or 6 tetrahedra. As we shall see below, the generalization of the edge flip to three dimensions exchanges two tetrahedra for three or vice versa. This variability does not usually pose a problem, except in the extreme cases. For example, n points in $\mathbb{R}^3$ can have a Delaunay triangulation with $\Omega(n^2)$ tetrahedra, even though some other triangulation will have only $O(n)$.

Finally, tetrahedra can be poorly shaped in more ways than triangles. In two dimensions, there are only two types of failure, angles close to $0°$ and angles close to $180°$, and no failures of the first kind implies no failures of the second. In three dimensions, we can classify poorly shaped tetrahedra according to both dihedral and solid angles [19]. There are then five types of bad tetrahedra, as shown in Figure 19. A *needle* permits arbitrarily small solid angles, but not large solid angles and neither large nor small dihedral angles. A *wedge* permits both small solid and dihedral angles, but neither large solid nor large dihedral angles, and so forth. Notice that a *sliver* or a *cap* can have all face angles bounded away from both $0°$ and $180°$, although the tetrahedron itself may have arbitrarily small solid angles and interior volume. An example is the sliver with vertex coordinates $(0,0,0), (0,1,0), (1,0,\epsilon)$, and $(1,1,\epsilon)$, where $\epsilon \to 0$.

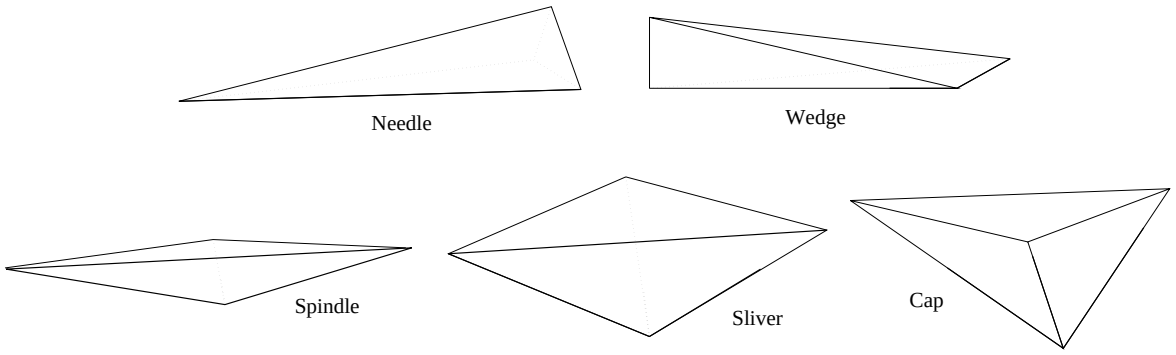

Figure 19. The five types of bad tetrahedra.

Many measures of tetrahedron quality have been proposed [69], most of which have a maximum value for an equilateral tetrahedron and a minimum value for a degenerate tetrahedron. One suitable measure, which forbids all five types of bad tetrahedra, is the minimum solid angle; this measure is essentially equivalent to the *aspect ratio*, usually defined to be the ratio of the radii of the inscribed and circumscribed spheres. A weaker measure, which forbids all types except slivers, is the ratio of the minimum edge length to the radius of the circumsphere [76].

6.1 Delaunay triangulation

As in two dimensions, point placement followed by Delaunay triangulation is a popular approach to mesh generation, especially in aerodynamics. The same point placement methods work fairly well: combining structured meshes [64], advancing front [8, 72, 73], and random scattering with filtering [119]. (There are also successful advancing front generators that place tetrahedra directly [61].) As in two dimensions, the placement phase must put sufficiently many points on the domain boundary to ensure that the Delaunay triangulation will be conforming. Although this problem is not intractable for most domains of practical interest, we do not know of any provably correct published solutions.

The first two point placement methods suffer from the same liability in three dimensions as in two: points may be improperly spaced at junctures between fronts or patches. All three methods suffer from a new sort of problem: even a well spaced point set may include sliver tetrahedra in its Delaunay triangulation, because a sliver does not have an unusually large circumsphere compared to the lengths of its edges. For this reason, some Delaunay mesh generators [8] include a special postprocessing step that finds and removes slivers. (Miller et al. [76] have recently shown that Voronoi cells of bounded aspect ratio suffice for the convergence of a finite volume formulation of Poisson's equation—slivers in the Delaunay triangulation are in fact acceptable. This result contrasts with empirical studies showing poor convergence for a finite element method on a sliver-filled mesh. The disparity between the two formulations is somewhat surprising, because they give the very same matrix in two dimensions.)

The triangulation phase of mesh generation also becomes somewhat more difficult in three dimensions. The generalization of the edge flip exchanges the two possible triangulations of five points in convex position, as shown in Figure 20. We call a flip a *Delaunay flip* if, after the flip, the triangulation of the five points satisfies the empty sphere condition—no

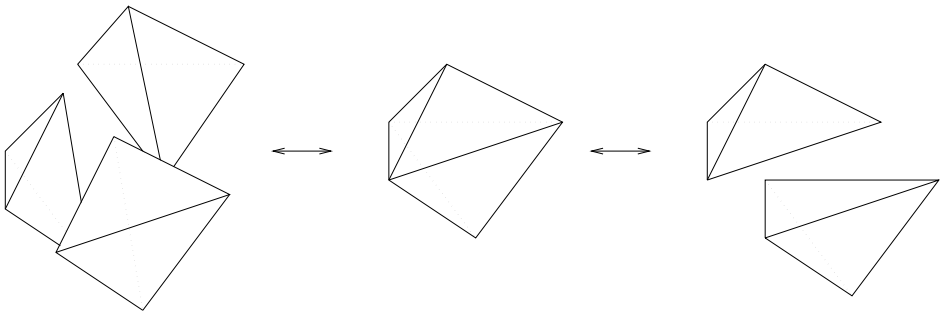

Figure 20. In three dimensions, an edge flip exchanges two tetrahedra sharing a triangle for three tetrahedra sharing an edge, or vice versa.

circumsphere encloses a point. In three dimensions, it is no longer true that any tetrahedralization can be transformed into the Delaunay triangulation by a sequence of Delaunay flips [65], and it is currently unknown whether any tetrahedralization can be tranformed into the Delaunay triangulation by arbitrary flips. Nevertheless, there are incremental Delaunay triangulation algorithms based on edge flipping [48, 66, 89]. These algorithms add one point at a time, splitting the tetrahedron receiving the new point into four by starring from the new point, and then flip to the new Delaunay triangulation. It is convenient to use four dummy points defining an initial bounding tetrahedron; these points and their incident tetrahedra are removed at the end.

There are other practical three-dimensional Delaunay triangulation algorithms as well. Bowyer [27] and Watson [118] gave incremental algorithms that add vertices according to x-coordinate order. Barber [12] implemented a randomized algorithm for computing convex hulls in arbitrary dimension. This algorithm can be used to compute Delaunay triangulations through a well-known reduction [28] which "lifts" the Delaunay triangulation of points in $\mathbb{R}^d$ to the lower convex hull of points in $\mathbb{R}^{d+1}$.

6.2 Octrees

An *octree* is the natural generalization of a quadtree. An initial bounding cube is split into eight congruent cubes, each of which is split recursively until each minimal cube intersects the domain Ω in a simple way. As in two dimensions, a balance condition ensures that no cube is next to one very much smaller than itself; balancing an unbalanced quadtree or octree expands the number of boxes by a constant multiplicative factor. The balance condition need not be explicit, but rather it may be a consequence of an intrinsic local spacing function [116].

Shephard and his collaborators [98, 100, 101, 121] have developed several octree-based mesh generators for polyhedral domains. Their original generator [121] tetrahedralizes leaf cubes using a collection of predefined patterns. To keep the number of patterns manageable, the generator makes the simplifying assumption that each cube is cut by at most three facets of the input polyhedron. Perucchio et al. [87] give a more sophisticated way to conform to boundaries. Buratynski [29] uses rectangular octrees and a hierarchical set of warping rules. Boxes are first warped to domain vertices, then edges, and finally faces. The warping rules are somewhat simplified by the fact that the octree is initially refined so that domain edges intersect boxes of only one size.

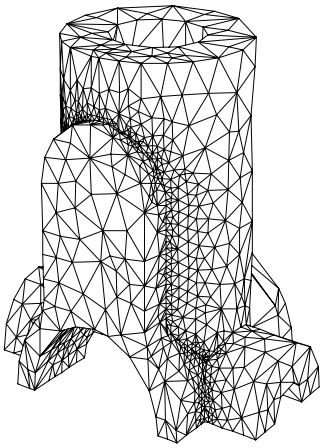

Figure 21. The surface of a tetrahedral mesh derived from an octree. (M. Yerry and M. Shephard)

Mitchell and Vavasis [80] generalized the quadtree mesh generator of Bern et al. [22] to three dimensions. The generalization is not straightforward, primarily because vertices of polyhedra may have very complicated local neighborhoods. This algorithm is guaranteed to avoid all five types of bad tetrahedra, while producing a mesh with only a constant times the minimum number of tetrahedra in any such bounded-aspect-ratio tetrahedralization. So far this is the only three-dimensional mesh generation algorithm with such a strong theoretical guaranty. Vavasis [116] has recently released a modified version of this algorithm as a Matlab software package, including a simple geometric modeler and equation solver to boot. The modified algorithm includes a more systematic set of warping rules; in particular, the new warping method for an octree cube cut by a single facet generalizes to any fixed dimension [81].

6.3 Refinement of Tetrahedral Meshes

In this section we discuss two different refinement algorithms based upon the natural generalization of bisection to three dimensions. To bisect tetrahedron $v_0v_1v_2v_3$ across edge v_0v_1, we add the triangle $v_{01}v_2v_3$, where v_{01} is the midpoint of v_0v_1, as shown in Figure 22. This operation creates two child tetrahedra, $v_0v_{01}v_2v_3$ and $v_{01}v_1v_2v_3$, and bisects the faces $v_0v_1v_2$ and $v_0v_1v_3$, which, unless they lie on the domain boundary, are each shared with an adjacent tetrahedron. Two tetrahedra that share a face must agree on how it is to be bisected; otherwise an invalid mesh will be constructed.

A single bisection of a tetrahedron can approximately square the minimum solid angle, unlike in two dimensions where the minimum angle of a triangle is decreased by no more than a factor of two. Consider the wedge tetrahedron with vertex coordinates $(0, \epsilon, 0)$, $(0, -\epsilon, 0)$, $(1, 0, \epsilon)$, and $(1, 0, -\epsilon)$. Bisection of the longest edge of this tetrahedron creates a new tetrahedron with minimum solid angle about ϵ^2.

Rivara and Levin [94] suggested an extension of longest-edge Rivara refinement (Section 4.4) to three dimensions. Notice that splitting the longest edge in a tetrahedron also splits the longest edge on the two subdivided faces, and thus the bisection of shared faces is uniquely defined. (Ties can be broken by vertex labels.) Neighboring invalid tetrahedra, all those sharing the subdivided longest edge, are refined recursively. Rivara and Levin provide

experimental evidence suggesting that repeated rounds of longest-edge refinement cannot reduce the minimum solid angle below a fixed threshold, but this reassurance has not been proved, nor has a bound on any weaker quality measure such as minimum dihedral angle.

The major obstacle to proving that longest-edge refinement cannot degrade mesh quality arbitrarily is that it is not known whether the algorithm generates only a finite number of tetrahedron similarity classes. However, a bisection algorithm first introduced by Bänsch [11] does have this property. Before describing the algorithm, we sketch the argument of Liu and Joe [68] which motivates the algorithm.

The key observation is that there exists an affine transformation that maps any tetrahedron to a canonical tetrahedron for which longest-edge bisection generates only a finite number of similarity classes. Consider the canonical tetrahedron t_c with coordinates $(-1,0,0)$, $(1,0,0)$, $(0,1/\sqrt{2},0)$, and $(0,0,1)$. In Figure 23 we illustrate the first three levels of longest-edge bisection of $t_c = v_0v_1v_2v_3$. It can be shown that all the tetrahedra generated at each level of refinement are similar and that the eight tetrahedra generated after three levels of refinement are similar to t_c.

The inverse of the affine mapping does not map similar tetrahedra in the canonical space into similar tetrahedra in the original space, unless the similar tetrahedra have the same orientation. Hence, Liu and Joe redefine similarity classes in the canonical space to include only tetrahedra that can be mapped onto each other by a combination of scaling and translation. They then show by direct calculation that longest-edge refinement of t_c produces only a finite number of these similarity classes. Therefore, the refinement in the canonical space induces a refinement in the original space with only a finite number of different tetrahedron shapes.

Bänsch [11] and Liu and Joe [70] give essentially equivalent algorithms for generating the bisection order; we follow Bänsch's presentation. Each face in each tetrahedron elects one of its edges to be its *refinement edge*, so that two conditions hold: the choice for a face is consistent between the two tetrahedra that share it, and exactly one edge in each tetrahedon—the *global refinement edge*—is chosen by two faces of the tetrahedron. These conditions hold initially if each face picks its longest edge and ties are broken in any consistent manner, for example, by vertex or edge label order. In the pseudocode below, a *child face* is a triangle like $v_{01}v_2v_1$ in Figure 22, and a *new face* is one like $v_{01}v_2v_3$.

```
mark the refinement edge of every face in the current mesh
let T_0 be the set of marked tetrahedra;  i = 0
while (T_i ≠ ∅) do
    bisect each tetrahedron in T_i across its global refinement edge
    pick the old edge in each child face as its refinement edge
    pick the longest edge in each new face as its refinement edge
    let T_i be the set of invalid tetrahedra;  i = i + 1
enddo
```

7 Conclusions

We have described the current state of the art in mesh generation for finite element methods. Practical issues in mesh generation are—roughly in order of importance—algorithm robustness, fit with underlying physics, element quality, and mesh efficiency.

Unstructured triangular and tetrahedral mesh generation already makes frequent use of data structures and algorithms familiar in computational geometry. We expect this trend to continue. Open problems abound in unstructured tetrahedral mesh generation. Is the flip graph for a point set connected? Is there a smoothing algorithm guaranteed to remove slivers? Is there an algorithm guaranteed to compute a Delaunay triangulation that conforms to a polyhedron?

We also expect—and recommend—computational geometers to focus some attention on structured meshes and hexahedral meshes. There are a number of interesting open questions in these areas as well. For example, is the new CRDT conformal mapping algorithm provably correct? Can any quadrilateral surface mesh with an even number of quadrilaterals be extended to a hexahedral volume mesh?

References

[1] M. Aftosmis, J. Melton, and M. Berger. Adaptation and surface modeling for Cartesian mesh methods. AIAA Paper 95-1725, 1995.

[2] E. Amezua, M. V. Hormaza, A. Hernandez, and M. B. G. Ajuria. A method of the improvement of 3d solid finite-element meshes. *Advances in Engineering Software*, 22:45–53, 1995.

[3] C. G. Armstrong, D. J. Robinson, R. M. McKeag, T. S. Li, and S. J. Bridgett. Medials for meshing and more. In *Proc. 4th International Meshing Roundtable, Sandia National Laboratories*, 1995.

[4] I. Babuška and A. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Analysis*, 13:214–227, 1976.

[5] I. Babuška and W.C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM Journal of Numerical Analysis*, 15:736–754, 1978.

[6] P. L. Baehmann, S. L. Wittchen, M. S. Shepard, K. R. Grice, and M.A. Yerry. Robust geometrically-based automatic two-dimensional generation. *Int. J. Numer. Meth. Eng.*, 24:1043–1078, 1987.

[7] B.S. Baker, E. Grosse, and C.S. Rafferty. Nonobtuse triangulation of polygons. *Disc. and Comp. Geometry*, 3:147–168, 1988.

[8] T.J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Eng. with Computers*, 5:161–175, 1989.

[9] Randolph E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0.* SIAM Publications, Philadelphia, Penn., 1990.

[10] Randolph E. Bank, Andrew H. Sherman, and Alan Weiser. Refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman et al., editor, *Scientific Computing*, pages 3–17. IMACS/North-Holland Publishing Company, Amsterdam, 1983.

[11] Eberhard Bänsch. Local mesh refinement in 2 and 3 dimensions. *Impact of Computing in Science and Engineering*, 3:181–191, 1991.

[12] C. B. Barber, D. P. Dobkin, and H. T. Hudanpaa. The Quickhull algorithm for convex hulls. Submitted to *ACM Trans. Math. Software*. See http://www.geom.umn.edu/software/qhull/, 1995.

[13] W.D. Barfield. An optimal mesh generator for Lagrangian hydrodynamic calculations in two space dimensions. *Journal of Computational Physics*, 6:417–429, 1970.

[14] R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Modeling*. Morgan Kaufmann, 1987.

[15] T. J. Barth. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. Technical report, von Karman Institute for Fluid Dynamics, Lecture Series 1994-05, 1994.

[16] S. E. Benzley, E. Perry, K. Merkley, B. Clark, and G. Sjaardema. A comparison of all-hexahedral and all-tetrahedral finite element meshes for elastic and elasto-platic analysis. In *Proc. 4th International Meshing Roundtable, Sandia National Laboratories*, pages 179–191, 1995.

[17] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[18] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[19] M. Bern, L. P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proc. 6th ACM-SIAM Symp. Disc. Algorithms*, pages 189–196, 1995.

[20] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T.-S. Tan. Edge-insertion for optimal triangulations. *Disc. and Comp. Geometry*, 10:47–65, 1993.

[21] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry, 2nd Edition*, pages 47–123. World Scientific, Singapore, 1995.

[22] M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. *J. Comp. System Sciences*, 48:384–409, 1994.

[23] M. Bern, D. Eppstein, and F. Yao. The expected extremes in a Delaunay triangulation. *Int. J. Comp. Geometry and Applications*, 1:79–92, 1991.

[24] M. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. In *Proc. 10th ACM Symp. Comp. Geometry*, pages 221–230, 1994.

[25] T. D. Blacker. Paving: a new approach to automated quadrilateral mesh generation. *Int. J. Numer. Meth. Eng.*, 32:811–847, 1991.

[26] F. Bossen. Anisotropic mesh generation with particles. Technical report, Master's thesis, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 1996.

[27] A. Bowyer. Computing Dirichlet tessellations. *Computer J.*, 24:162–166, 1981.

[28] K. Q. Brown. Voronoi diagrams from convex hulls. *Inform. Process. Lett.*, 9:223–228, 1979.

[29] E.K. Buratynski. A fully automatic three-dimensional mesh generator for complex geometries. *Int. J. Numer. Meth. Eng.*, 30:931–952, 1990.

[30] Scott Canann, Michael Stephenson, and Ted Blacker. Optismoothing: an optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design*, 13:185–190, 1993.

[31] G. F. Carey and J. T. Oden. *Finite Elements: Computational Aspects*. Prentice-Hall, 1984.

[32] J.E. Castillo, editor. *Mathematical Aspects of Numerical Grid Generation*. SIAM, 1991.

[33] José E. Castillo. *Mathematical Aspects of Grid Generation*. Society for Industrial and Applied Mathematics, Philadelphia, 1991.

[34] M. J. Castro-Diaz, F. Hecht, and B. Mohammadi. New progress in anisotropic grid adaptation for inviscid and viscid flows simulations. In *Proc. 4th International Meshing Roundtable, Sandia National Laboratories*, 1995.

[35] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984.

[36] L.P. Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Comp. Science Dept., Cornell University, 1989.

[37] P.G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, 1978.

[38] P.G. Ciarlet and P.A. Raviart. Maximum principle and uniform convergence for the finite element method. *Computer Methods in Appl. Mechanics and Engineering*, 2:17–31, 1973.

[39] W. J. Coirier and K. G. Powell. An accuracy assessment of Cartesian-mesh approaches for the Euler equations. *Journal of Computational Physics*, 117:121–131, 1995.

[40] William John Coirier. An adaptively-refined, Cartesian cell-based scheme for the Euler and Navier-Stokes equations. NASA Technical Memorandum 106754, NASA, October 1994.

[41] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM J. Sci. Stat. Comput.*, 10:1063–1075, 1989.

[42] L. De Floriani and B. Falcidieno. A hierarchical boundary model for solid object representation. *ACM Transactions on Graphics*, 7:42–60, 1988.

[43] B. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[44] M. T. Dickerson and M. H. Montague. The exact minimum weight triangulation. In *Proc. 12th ACM Symp. Comp. Geometry*, 1996.

[45] T. A. Driscoll and S. A. Vavasis. Numerical conformal mapping using cross-ratios and delaunay triangulation. Available under http://www.cs.cornell.edu/Info/People/vavasis/vavasis.html, 1996.

[46] Tobin A. Driscoll. A Matlab toolbox for Schwarz-Christoffel mapping. *ACM Transactions on Mathematical Software*, (to appear).

[47] A.S. Dvinsky. Adaptive grid generation from harmonic maps. In S. Sengupta, J. Häuser, P.R. Eiseman, and J.F. Thompson, editors, *Numerical Grid Generation in Computational Fluid Dynamics '88*. Pineridge Press Limited, Swansea, U.K., 1988.

[48] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *Proc. 8th ACM Symp. Comp. Geometry*, pages 43–52, 1992.

[49] H. Edelsbrunner and T.-S. Tan. A quadratic time algorithm for the minmax length triangulation. In *Proc. 32nd IEEE Symp. Foundations of Comp. Science*, pages 414–423, 1991.

[50] H. Edelsbrunner and T.-S. Tan. An upper bound for conforming Delaunay triangulations. *Disc. and Comp. Geometry*, 10:197–213, 1993.

[51] H. Edelsbrunner, T.S. Tan, and R. Waupotitsch. A polynomial time algorithm for the minmax angle triangulation. *SIAM J. Sci. Stat. Comp.*, 13:994–1008, 1992.

[52] D. Eppstein. Linear complexity hexahedral mesh generation. In *Proc. 12th ACM Symp. Comp. Geom.*, 1996.

[53] David A Field. Laplacian smoothing and Delaunay triangulations. *Communications and Applied Numerical Methods*, 4:709–712, 1988.

[54] S. Fortune. Voronoi diagrams and Delaunay triangulations. In F. K. Hwang and D.-Z. Du, editors, *Computing in Euclidean Geometry, 2nd Edition*, pages 225–265. World Scientific, Singapore, 1995.

[55] Lori A. Freitag, Mark T. Jones, and Paul E. Plassmann. An efficient parallel algorithm for mesh smoothing. In Timothy T. Tautges, editor, *Proceedings 4th International Meshing Roundtable*, pages 47–58. Sandia National Laboratories, 1995.

[56] P. L. George. *Automatic Mesh Generation*. Wiley, New York, 1991.

[57] P. L. George, F. Hecht, and E. Saltel. Fully automatic mesh generator for 3D domains of any shape. *Impact of Com. in Sci. and Eng.*, 2:187–218, 1990.

[58] A. S. Glassner. Maintaining winged-edge models. In E J. Arvo, editor, *Graphics Gems II*, pages 191–201. Academic Press Professional, Boston, MA, 1991.

[59] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graphics*, 4:74–123, 1985.

[60] O. Hassan, K. Morgan, E. J. Probert, and J. Peraire. Mesh generation and adaptivity for the solution of compressible viscous high-speed flows. *Int. J. Numer. Meth. Eng.*, 38:1123–1148, 1995.

[61] O. Hassan, K. Morgan, E. J. Probert, and J. Peraire. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *Int. J. Numer. Meth. Eng.*, 39:549–567, 1996.

[62] Thomas J.R̃. Hughes. *The finite element method: linear static and dynamic finite element analysis*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[63] Thomas J.R̃. Hughes and J.Ẽ. Akin. Techniques for developing 'special' finite element shape functions with particular reference to singularities. *International Journal for Numerical Methods in Engineering*, 15:733–751, 1980.

[64] A. Jameson, T. J. Baker, and N. P. Weatherill. Calculation of inviscid transonic flow over a complete aircraft. In *Proc. AIAA 24th Aerospace Sciences Meeting, Reno*, 1986.

[65] B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comput.*, 10:718–741, 1989.

[66] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[67] P. Knupp and Stanley Steinberg. *Fundamentals of Grid Generation*. CRC Press, 1994.

[68] Anwei Liu and Barry Joe. On the shape of tetrahedra from bisection. *Mathematics of Computation*, 63(207):141–154, July 1994.

[69] Anwei Liu and Barry Joe. Relationship between tetrahedron shape measures. *BIT*, 34:268–287, 1994.

[70] Anwei Liu and Barry Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing*, 16(6):1269–1291, November 1995.

[71] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering*, 21:1403–1426, 1985.

[72] S. H. Lo. Volume discretization into tetrahedra. *Computers and Structures*, 39:493–511, 1991.

[73] R. Löhner. Generation of three-dimensional unstructured grids by the advancing-front method. In *Proc. AIAA 26th Aerospace Sciences Meeting, Reno*, 1988.

[74] D. J. Mavriplis. Unstructured and adaptive mesh generation for high Reynolds number viscous flows. Technical Report 91-25, ICASE, NASA Langley Research Center, 1991.

[75] G.L. Miller, D. Talmor, and S.-H. Teng. Geometric mesh coarsening. Manuscript in preparation, 1996.

[76] G.L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*, pages 683–692, 1995.

[77] G.L. Miller, S.-H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Science*, pages 538–547, 1991.

[78] S.A. Mitchell. Refining a triangulation of a planar straight-line graph to eliminate large angles. In *Proc. 34th IEEE Symp. on Foundations of Comp. Science*, pages 583–591, 1993.

[79] S.A. Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS '96)*. Springer-Verlag, LNCS, 1996.

[80] S.A. Mitchell and S. Vavasis. Quality mesh generation in three dimensions. In *Proc. 8th ACM Symp. Comp. Geom.*, pages 212–221, 1992.

[81] S.A. Mitchell and S. Vavasis. An aspect ratio bound for triangulating a d-grid cut by a hyperplane. In *Proc. 12th ACM Symp. Comp. Geom.*, 1996.

[82] J.-D. Müller. Proven angular bounds and stretched triangulations with the frontal Delaunay method. In *Proc. 11th AIAA Comp. Fluid Dyanmics, Orlando*, 1993.

[83] L. R. Nackman and V. Srinivasan. Point placement for Delaunay triangulation of polygonal domains. In *Proc. 3rd Canadian Conf. Comp. Geometry*, pages 37–40, 1991.

[84] Carl F. Ollivier-Gooch. Multigrid acceleration of an upwind Euler solver on unstructured meshes. *AIAA Journal*, 33(10):1822–1827, 1995.

[85] S. Owen. Meshing research corner. http://www.ce.cmu.edu:8000/user/sowen/www/mesh.html, 1995.

[86] V. N. Parthasarathy and Srinivas Kodiyalam. A constrained optimization approach to finite element mesh smoothing. *Finite Elements in Analysis and Design*, 9:309–320, 1991.

[87] R. Perucchio, M. Saxena, and A. Kela. Automatic mesh generation from solid models based on recursive spatial decomposition. *Int. J. Numer. Meth. Eng.*, 28:2469–2502, 1989.

[88] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[89] V.T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proc. 7th ACM Symp. Comp. Geometry*, pages 357–363, 1991.

[90] S. Rippa. Minimal roughness property of the Delaunay triangulation. *Computer Aided Geometric Design*, 7:489–497, 1990.

[91] Maria-Cecilia Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.

[92] Maria-Cecilia Rivara. Design and data structure of fully adaptive, multigrid, finite-element software. *ACM Transactions on Mathematical Software*, 10(3):242–264, September 1984.

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

[94] Maria-Cecilia Rivara and Cristian Levin. A 3-d refinement algorithm suitable for adaptive and multi-grid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[95] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proc. 4th ACM-SIAM Symp. on Disc. Algorithms*, pages 83–92, 1993.

[96] A. Saalfeld. Delaunay edge refinements. In *Proc. 3rd Canadian Conf. Comp. Geometry*, pages 33–36, 1991.

[97] R. Schneiders. Finite element mesh generation. http://www-users.informatik.rwth-aachen.de/ ~roberts/meshgeneration.html, 1995.

[98] W. J. Schroeder and M. S. Shephard. A combined octree/Delaunay method for fully automatic 3-D mesh generation. *Int. J. Numer. Meth. Eng.*, 29:37–55, 1990.

[99] H.-P. Seidel. Polar forms and triangular B-spline surfaces. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, pages 299–350. World Scientific, Singapore, 1995.

[100] M. S. Shephard, F. Guerinoni, J. E. Flaherty, R. A. Ludwig, and P.L. Baehmann. Finite octree mesh generation for automated adaptive three-dimensional flow analysis. In *Proc. 2nd Int. Conf. Numer. Grid Generation in Computational Fluid Mechanics*, pages 709–718, 1988.

[101] Mark Shephard and Marcel Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical methods in Engineering*, 32:709–749, 1991.

[102] J. R. Shewchuk. Triangle: A two-dimensional quality mesh generator and Delaunay triangulator. See http://www.cs.cmu.edu/%7Equake/triangle.html, 1995.

[103] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates in C. In *Proc. 12th ACM Symp. Comp. Geometry*, 1996.

[104] K. Shimada and D. C. Gossard. Computational methods for physically-based FE mesh generation. In *Proc. IFIP TC5/WG5.3 8th Int. Conference on PROLAMAT, Tokyo*, 1992.

[105] Barry Smith, Petter Bjørstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Algorithms for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.

[106] P.W. Smith and S.S. Sritharan. Theory of harmonic grid generation. *Complex Variables*, 10:359–369, 1988.

[107] V. Srinivasan, L. R. Nackman, J.-M. Tang, and S.N. Meshkat. Automatic mesh generation using the symmetric axis tranformation of polygonal domains. Technical Report RC 16132, Comp. Science, IBM Research Division, Yorktown Heights, NY, 1990.

[108] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, 1973.

[109] T.-S. Tan. An optimal bound for conforming quality triangulations. In *Proc. 10th ACM Symp. Comp. Geometry*, pages 240–249, 1994.

[110] T. J. Tautges and S. A. Mitchell. The whisker weaving algorithm for constructing all-hexahedral finite element meshes. In *The 3rd International Meshing Roundtable, Sandia National Laboratories*, 1994.

[111] T. J. Tautges and S. A. Mitchell. Whisker weaving: invalid connectivity resolution and primal construction algorithm. In *Proc. 4th International Meshing Roundtable, Sandia National Laboratories*, pages 115–127, 1995.

[112] J. W. Thomas. *Numerical partial differential equations: finite difference methods.* Springer, New York, 1995.

[113] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation: Foundations and Applications.* North-Holland, 1985.

[114] J.F. Thompson. *Numerical Grid Generation.* Elsevier, Amsterdam, 1982.

[115] L. N. Trefethen. Numerical computation of the Schwarz-Christoffel transformation. *SIAM Journal on Scientific and Statistical Computing*, 1:82–102, 1980.

[116] S. Vavasis. QMG: mesh generation and related software. http://www.cs.cornell.edu/Info/People/vavasis/qmg-home.html, 1995.

[117] S.A. Vavasis. Stable finite elements for problems with wild coefficients. Technical Report TR93-1364, Dept. of Comp. Science, Cornell University, 1993.

[118] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer J.*, 24:167–171, 1981.

[119] N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *Int. J. Numer. Meth. Eng.*, 37:2005–2039, 1994.

[120] A. Weiser. Local-mesh, local-order, adaptive finite element methods with a posteriori error estimates for elliptic partial differential equations. Technical Report 213, Yale University, New Haven, Connecticut, 1981.

[121] M. A. Yerry and M. S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *Int. J. Numer. Meth. Eng.*, 20:1965–1990, 1984.

[122] M. A. Yerry and M. S. Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3:39–46, January/February 1983.

[123] D.P. Young, R.G. Melvin, M.B. Bieterman, and J.E. Bussoletti. A locally refined rectangular grid finite element method: Application to computational fluid dynamics and computational physics. *Journal of Computational Physics*, 92:1–66, 1991.

[124] Roger Young and Ian MacPhedran. Internet finite element resources. http://www.engr.usask.ca/ ~macphed/finite/fe_resources/fe_resources.html, 1995.

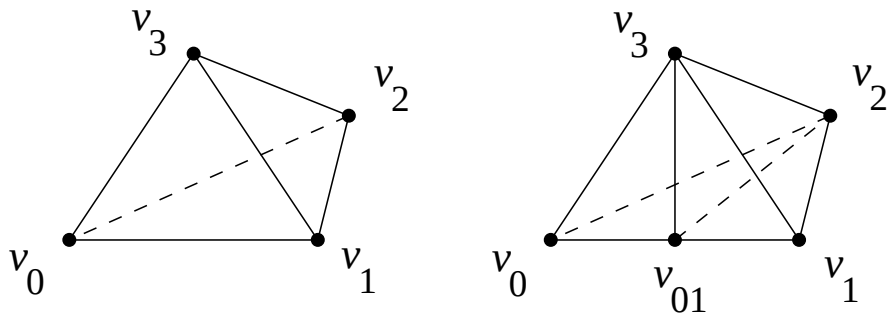

Figure 22. The tetrahedron on the left is bisected to form two new tetrahedra

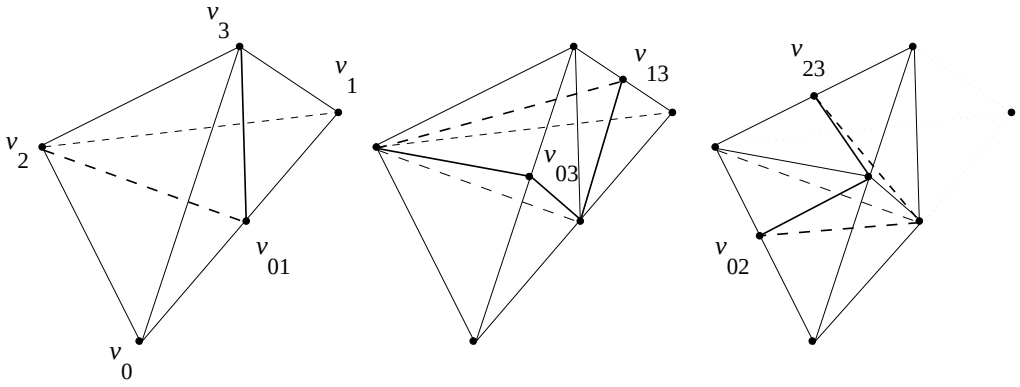

Figure 23. The first three levels of longest-edge bisection of the canonical tetrahedron. Note that the tetrahedra generated at each level are similar. For the final level of refinement we show only the four tetrahedra obtained from $v_0 v_1 v_{12} v_3$. Four similar tetrahedra are obtained from $v_0 v_{12} v_2 v_3$.