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 partial 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.

Construction and modeling of domain geometry lie outside the scope of this chapter, so we shall simply assume that domains are given in some sort of boundary representation,

*Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304; bern@parc.xerox.com

†Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802; plassman@cse.psu.edu Supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

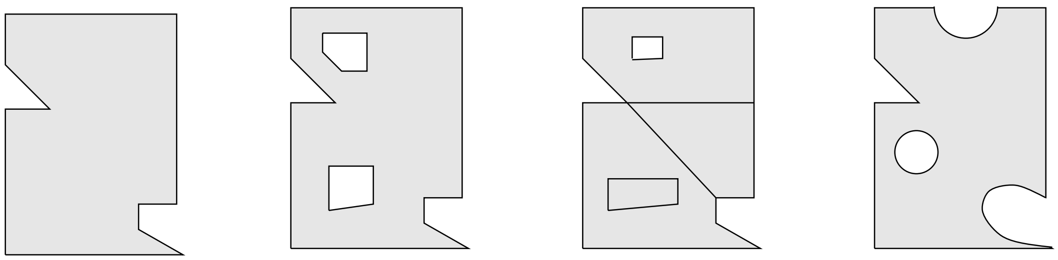

Figure 1. Types of two-dimensional inputs: (a) simple polygon, (b) polygon with holes, (c) multiple domain, and (d) curved domain.

without specifying the exact form of this representation. Computational geometers typically assume exact, combinatorial data structures, such as linked lists for simple polygons and polygons with holes, doubly connected edge lists [103] or quad-edge structures [63] for planar multiple domains, and winged-edge structures [44, 62] for polyhedral domains.

In practice, complicated domains are designed on computer aided design (CAD) systems. These systems use surface representations designed for visual rendering, and then translate the final design to another format for input to the mesh generator. The Stereolithography Tessellation Language (STL) file format, originally developed for the rapid prototyping of solid models, specifies the boundary as a list of surface polygons (usually triangles) and surface normals. The advantages of the STL input format are that a "watertight" model can be ensured and model tolerance (deviation from the CAD model) can be specified by the user. The Initial Graphics Exchange Specification (IGES) format enables a variety of surface representations, including higher-order representations such as B-splines and NURBs. Perhaps due to its greater complexity or to sloppy CAD systems or users, IGES files often contain incorrect geometry (either gaps or extra material) at surface intersections.

An alternative approach to format translation is to directly query the CAD system with, say, point-enclosure queries, and then construct a new representation based on the answers to those queries. This approach is most advantageous when the translation problem is difficult, as it may be in the case of implicit surfaces (level sets of complicated functions) or constructive solid geometry formulas.

With either approach, translation or reconstruction by queries, the CAD model must be topologically correct and sufficiently accurate to enable meshing. We expect to see greater integration between solid modeling and meshing in the future.

1.2 Types of Meshes

A *structured* mesh is one in which all interior vertices are topologically alike. In graph-theoretic terms, a structured mesh is an induced subgraph of an infinite periodic graph such as a grid. An *unstructured* mesh is one in which vertices may have arbitrarily varying local neighborhoods. A *block-structured* or *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 (refinement/derefinement based on an initial solution) and a better fit to complicated domains. High-quality hybrid meshes enjoy the advantages of both approaches, but hybrid meshing is not yet fully automatic. We shall discuss unstructured mesh generation at greater length than structured or hybrid mesh

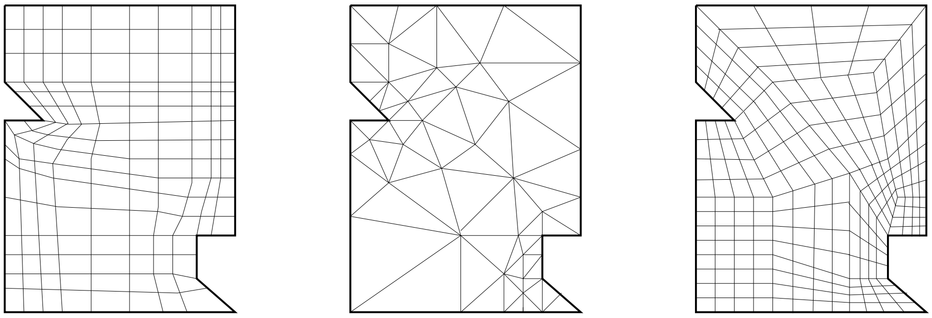

Figure 2. Types of meshes: (a) structured, (b) unstructured, and (c) block-structured.

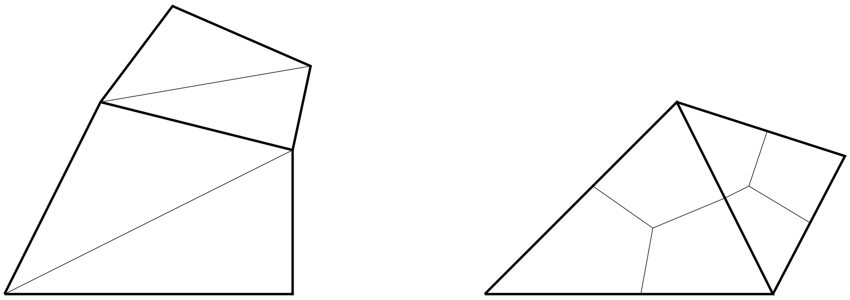

Figure 3. (a) Triangulating quadrilaterals. (b) Subdividing triangles to form quadrilaterals.

generation, both because the unstructured approach seems to be 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 (Figure 3) and between tetrahedra and hexahedra.

1.3 Organization

Section 2 gives a brief survey of numerical methods and their implications for mesh generation. Section 3 discusses the influence of element shape on accuracy and convergence time. Sections 4 and 5 cover structured and unstructured two-dimensional meshes. Section 6 discusses three-dimensional hexahedral mesh generation, including structured, hybrid, and unstructured approaches. Finally, Section 7 describes three-dimensional unstructured tetrahedral mesh generation.

We shall explain the basic computational geometry results as they arise within a larger context; however, Section 5 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 [24] 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 [34, 125], structured mesh generation [35, 61, 86, 130, 131], and unstructured mesh generation [24, 60, 85, 86, 131]. There are also several nice Web sites [99, 113, 117, 145] with up-to-date information on mesh generation.

2 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 (PDEs) 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 differ.

The *finite difference method* [128] replaces a continuous differential operator with a difference approximation. Consider the partial differential equation

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

where $\mathcal{L}$ is some differential operator and 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 $\mathcal{L}$ 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 approximated by

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

where $adj(x_i)$ is the set of points adjacent to x_i in the mesh and A_{ik} is a set of weights depending only on $\mathcal{L}$ and the geometry of the mesh. 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, because the convergence properties of a particular stencil can be analyzed 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 reasons, the finite difference method usually relies on structured meshes.

The *finite element method* [125] approximates the solution rather than the equation. 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, $\mathcal{L}\phi_k$, is easy to compute. 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 $\psi_i(x)$:

$$\int_\Omega (\mathcal{L}\bar{u})\psi_i = \int_\Omega \bar{f}\psi_i .$$

An implementation of the finite element method involves the computation of coefficients

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

where I_{ik} denotes the set of elements for which both the basis function ϕ_k and the test function ψ_i are nonzero. The unknowns a_k can now be expressed in terms of the A_{ik}'s by a sparse linear system of equations, of the same overall form as (2).

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. For these reasons, unstructured meshes find broad and increasing use in finite element methods.

The *finite volume method* is motivated by physical conservation laws. 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 *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. The finite volume method models fluid dynamics problems especially well, because pressure and velocity can be represented at centers and vertices of volumes, respectively.

Cell-centered control volumes are usually identical to mesh elements, while *vertex-centered* control volumes form a dual mesh with one volume for each vertex of the original mesh. 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 for the mesh, and its dual—the Voronoï diagram—for the control volumes.

2.2 Solution Methods

The solution of the sparse linear system is usually the most computationally demanding of the three steps. Solution methods include direct factorization and preconditioned iterative methods, methods which can vary dramatically in required storage and computational cost for different problems. Moreover, the discrete formulation and mesh generation steps greatly influence the efficacy of a solution method. For example, if one chooses to use higher-order basis functions in the finite element method, one can use a coarser mesh, and a smaller but denser linear system.

Direct factorization methods, such as sparse Cholesky or LU factorization, tend be slower but more robust than iterative methods. Luckily the extra computational cost can be amortized when the factorization is reused to solve for more than one right-hand side. An important issue in direct methods is ordering the vertices to minimize the "fill", the number of intermediate nonzeros. Nested dissection [72] uses graph separators to save an asymptotic factor of $O(n^{3/d})$ in the fill. Any planar graph admits separators of size $O(n^{1/2})$; reasonable three-dimensional meshes admit separators of size $O(n^{2/3})$ [89].

Iterative methods [17] have proved effective in solving the linear systems arising in physical modeling. Most large problems cannot be effectively solved without the use of preconditioning. A popular preconditioner 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 [140] 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 to coarse. The iterative solution of the fine linear system is accelerated by solving the coarser systems. The cycle repeatedly projects the current residual from a finer mesh onto the next coarser mesh and interpolates the solution from the coarser mesh onto the finer. One drawback of multigrid is that computing a sequence of meshes may be difficult for complicated geometries.

Domain decomposition [122] represents something of a hybrid between iterative and direct approaches. This approach divides the domain into possibly overlapping small domains; it typically solves 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 of a burden on the mesh generator. In fact, the domain is often partitioned so that subproblems admit structured meshes.

3 Element Shape

The shapes of elements in a mesh have a pronounced effect on numerical methods. For purposes of discussion, let us define the *aspect ratio* of an element to be the ratio of its maximum to its minimum width, where width refers to the distance between parallel supporting hyperplanes. There are many other roughly equivalent definitions of aspect ratio.

In general, elements of large aspect ratio are bad. Large aspect ratios lead to poorly conditioned matrices, worsening the speed and accuracy of the linear solver. Speed degrades before accuracy; a triangular mesh with a rather mild sharpest angle (say $10°$) can be noticeably slower than a triangular mesh with a minimum angle of $45°$.

Moreover, even assuming that the solver gives an exact answer, large aspect ratios may give unacceptable interpolation error. Here it is useful to distinguish between two different types of poor aspect ratio. Early results [39] showed convergence as triangular elements shrink, assuming that all angles are bounded away from $0°$. Later analysis [6], however, showed convergence assuming only that angles are bounded away from $180°$, a weaker condition. The generalization of this result to three dimensions assumes dihedrals bounded away from $0°$. Bank and Smith [13] recently proposed a triangle quality measure based on an analysis of interpolation error: the area of a triangle divided by the sum of squared edge lengths. This measure slightly favors sharp triangles over flat triangles.

Sometimes, however, elements of large aspect ratio are good! If the solution to the differential equation is *anisotropic*, meaning that its second derivative varies greatly with direction, then properly aligned high-aspect-ratio elements give a very efficient mesh. The ideal aspect ratio of a triangle is the square root of the ratio of the largest to smallest eigenvalues of the Hessian [106]. For triangular meshes, it does not make much difference whether long skinny elements have large angles as well as small angles, but if the aspect ratio exceeds the ideal then large angles are worse than small [106].

Fluid flow problems, especially full Navier-Stokes simulation (that is, viscosity included), are strongly anisotropic. For example, in aerodynamic simulations ideal aspect ratios may reach 10,000 along the surface of the aircraft. Quadrilateral and hexahedral meshes have an advantage in accuracy over triangular and tetrahedral meshes for control-volume formulations of these problems, as they allow faces of elements in the boundary layer to be either almost parallel or almost orthogonal to the surface.

Simulations with shock fronts—for example, supersonic air flow over a wing—are also strongly anisotropic. In this case, however, the locations and directions for high-aspect-ratio elements cannot be predicted in advance. The need for *adaptivity* (remeshing based on an initial solution) now tilts the balance in favor of triangles and tetrahedra [36]. Simpson [121] discusses and surveys the literature on anisotropy.

Element shape affects another property of the linear system besides condition number. A triangular mesh with well-shaped elements gives a symmetric M-matrix—positive definite with negative off-diagonal entries—for a finite element formulation of an equation with a Laplacian operator. M-matrices are exactly those matrices that satisfy a discrete maximum principle; this desirable property rules out oscillation of the numerical method. In this case, "well-shaped" has a precise meaning: the two angles opposite each interior edge of the mesh should sum to at most $180°$ [18, 40]. This requirement implies that no quadrilaterals are "reversed" (Section 5.1), so the triangulation must be the Delaunay or constrained Delaunay triangulation. Depending on the boundary conditions associated with the differential equation, an M-matrix may also require that the single angle opposite a boundary edge should measure at most $90°$. This requirement goes beyond Delaunay, but it is not hard to satisfy this requirement for domains without internal boundaries: simply split outwards-facing obtuse angles by dropping perpendiculars to the boundary, flip back to a new Delaunay triangulation, and repeat until there are no reversed quadrilaterals and no outwards-facing obtuse angles.

In three dimensions, an unstructured tetrahedral mesh gives an M-matrix if and only if, for each edge e' in the mesh, the sum $\sum_e |e| \cot \theta_e$ is nonnegative, where the sum is over all edges e that are opposite to e' in tetrahedra of the mesh, and where $|e|$ denotes the length of e and θ_e the dihedral angle at e [141]. All such sums will be nonnegative if all dihedrals in the mesh are nonobtuse, but this condition is more restrictive than necessary.

Finally, in a finite volume formulation of Poisson's equation, a Delaunay mesh with Voronoï control volumes gives an M-matrix even in three dimensions [88]. The finite volume formulation—but not the finite element formulation—can tolerate Delaunay tetrahedra of large aspect ratio, so long as all control volumes have good aspect ratios [88]. The disparity between the two formulations is surprising, because they give the very same matrix in two dimensions.

4 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 in a structured mesh cannot grade in size as rapidly. These two difficulties can be solved by the hybrid structured/unstructured approach, which decomposes a complicated domain into blocks supporting structured grids. Hybrid approaches, however, are not yet fully automatic, requiring user guidance in the decomposition step. A complicated three-dimensional hybrid mesh (see Section 6.1) can take weeks or even months of work; hence hybrid approaches are typically used only late in the design cycle.

Structured mesh generation can be roughly classified into hand-generated and other elementary approaches, algebraic or interpolation methods, and PDE or variational methods [129]. The PDE approach [35, 73] 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 [82] 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 $z = x + iy$ and consider the the complex

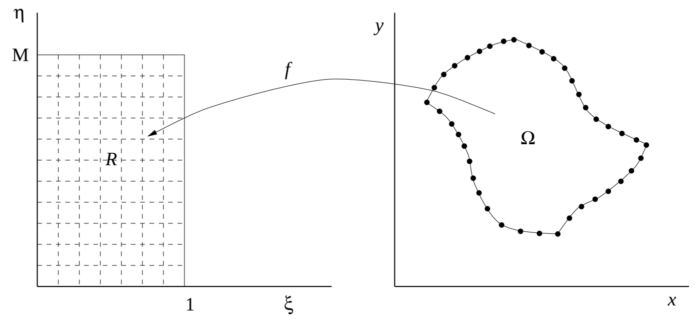

Figure 4. The conformal mapping f from the domain Ω, defined by a boundary discretization, to a rectangle R. The inverse of the mapping maps a grid on R onto a structured mesh for Ω.

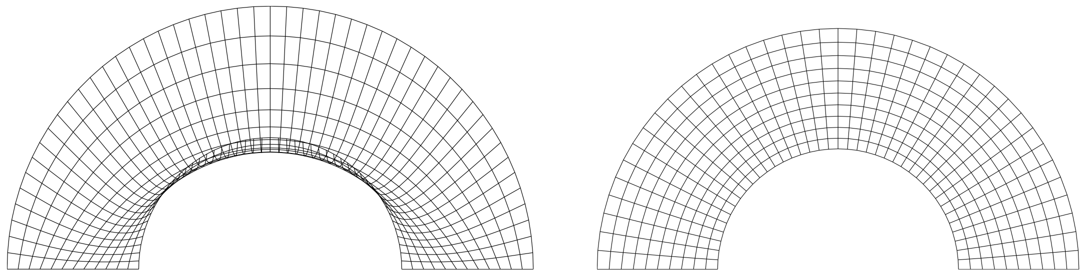

Figure 5. The grid on left was obtained by solving (3) with unit aspect ratio, resulting in a folded-over mesh. On the right, a more appropriate aspect ratio has been chosen.

function $f(z) = \xi(x,y) + i\eta(x,y)$. 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 4, and assume we already have a discretization of the boundary of Ω. (Finding a suitable boundary discretization may itself be a nontrivial 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 5(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 [35] by obtaining a better estimate for M, the rectangle height implied by the discretization of the boundary of Ω. If we scale the original 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-Riemann equations we have

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

Barfield [16] 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 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 [49, 123]. 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 [35]

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

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 [129].

The elliptic method just discussed, though motivated by conformal mapping, does not compute true conformal mappings. A true conformal mapping induces a structured mesh with certain advantages; for example, the Laplacian is the limit of the second-order difference on such a grid. True conformal mapping, however, does not seem to be widely used in mesh generation, perhaps because algorithms to compute such mappings are relatively slow, or because they do not allow local control of point spacing.

In the case that Ω is a simple polygon, the Schwarz-Christoffel formula offers 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. 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 [133], the SC Toolbox by Driscoll [47], and CRDT by Driscoll and Vavasis [48].

One difficulty in the numerical solution is "crowding", enormous variation in spacing between the ω_k points. CRDT, the latest and apparently best Schwarz-Christoffel algorithm, overcomes this difficulty by repeatedly remapping so that crowding does not occur near the points being evaluated.

5 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, and there is a large literature on the subject. We consider three principled approaches to unstructured mesh generation in some detail; these approaches use Delaunay triangulation, constrained Delaunay triangulation, and quadtrees. Then we discuss mesh refinement and improvement. In the final section, we describe some geometric problems abstracted from unstructured mesh generation.

5.1 Delaunay Triangulation

Our first approach to unstructured mesh generation partitions the task into two phases: placement of mesh vertices, followed by triangulation. (Added points are called *Steiner points* to distinguish them from the domain's original vertices.) 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 vertices along the domain boundary before adding them to the interior. The boundary should be lined with enough Steiner points that the Delaunay triangulation of all vertices will *conform* to the domain. This requirement inspires a crisp geometric problem, called *conforming Delaunay triangulation*: given a polygonal domain Ω, add Steiner points so that each edge of Ω is a union of edges in the Delaunay triangulation. An algorithm due to Saalfeld [112] lines the edges of Ω with a large number of Steiner points, uniformly spaced except near the endpoints. A more efficient solution [96] covers the edges of Ω by disks that do not overlap other edges. Edelsbrunner and Tan [52] gave the best theoretical result, an algorithm that uses $O(n^3)$ Steiner points for an n-vertex multiple domain. They also gave an $\Omega(n^2)$ lower bound example.

There are several approaches to placing interior Steiner points. One approach [84] combines the vertices from a number of structured meshes. A second approach [10, 95] adds Steiner points in successive layers, working in from the domain boundary as in advancing front mesh generation (Section 7.2). Figure 6 shows an example. A third approach [88, 137] chooses interior points at random according to some distribution, which may be interpolated from a coarse quadtree or "background" triangulation. An independent random sample is likely to produce some badly shaped triangles [26], so the generator should oversample and then filter out points too close to previously chosen points [88]. Finally, there are deterministic methods that achieve essentially the same effect as random sampling with filtering; these methods [29, 120] define birth and death rules that depend upon the density of neighboring points.

All of these methods can give anisotropy. The first and second approaches, structured submeshes and advancing front, offer local control of element shapes and orientations. These

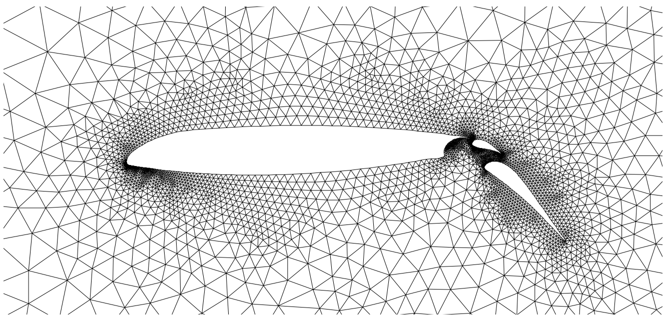

Figure 6. Delaunay triangulation of points placed by an advancing front. (T. Barth)

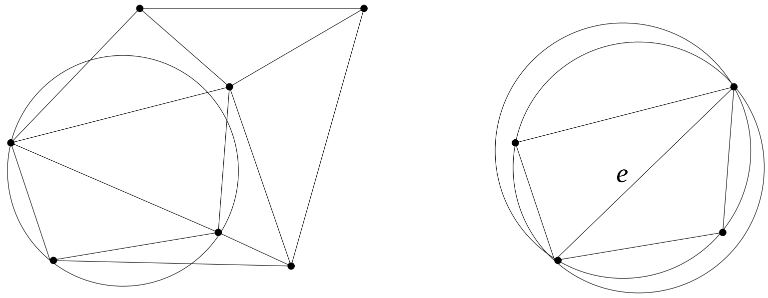

Figure 7. (a) Delaunay triangulation. (b) A reversed quadrilateral.

approaches may space points improperly where structured meshes or advancing fronts collide, but this flaw can usually be corrected by filtering points and later smoothing the mesh. The third and fourth approaches trade direct control over element shapes for ease of fitting complicated geometries. Nevertheless, one can achieve anisotropy with these approaches by computing the Delaunay triangulation within a stretched space [29, 36, 43]. For example, Bossen [29] uses a "background" triangulation to define local affine transformations; Delaunay flips (described below) are then made with respect to transformed circles. Stretched Delaunay triangulations have many more large angles than ordinary Delaunay triangulations, but this should not pose a problem unless the stretching exceeds the desired amount (Section 3).

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. See Figure 7(a). 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 [56]. 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

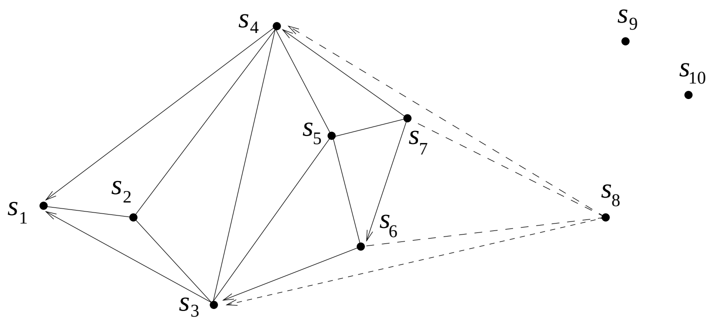

Figure 8. A sweep-line algorithm for computing an initial triangulation

without the diagonal sum to more than 180°, or equivalently, if each triangle circumcircle contains the opposite vertex as in Figure 7(b). 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, as shown in Figure 8. The following theorem [45] guarantees the success of edge flipping: a triangulation in which no quadrilateral is reversed must be a completion of the Delaunay triangulation.

5.2 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 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 not cut by an edge of Ω, that has an an *empty* circumcircle, where empty now means that the circle does not contain any vertices of Ω visible to points inside the triangle. The visibility requirement means that external proximities, where Ω wraps around to nearly touch itself, have no effect. Figure 9 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 initial triangulation is somewhat more difficult for polygonal domains than for point sets. The textbook by Preparata and Shamos [103] describes an $O(n \log n)$-time algorithm for

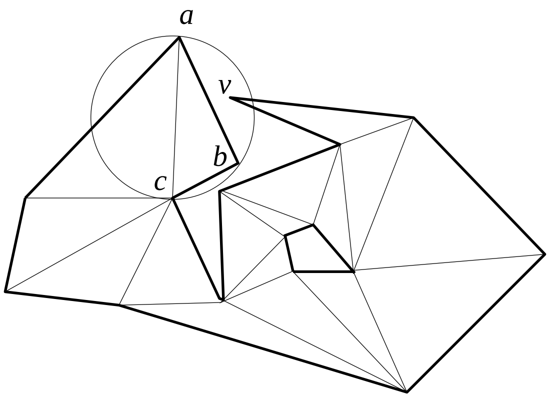

Figure 9. The constrained Delaunay triangulation of a polygon with holes.

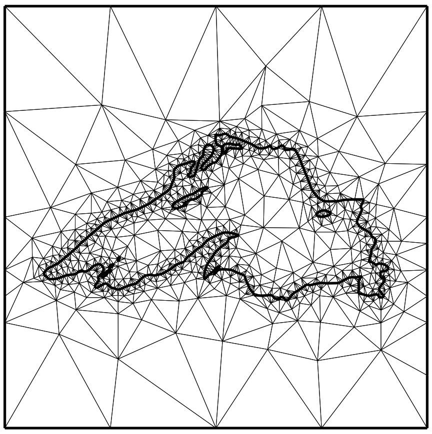

Figure 10. A mesh computed by Ruppert's algorithm. (J. Ruppert)

computing an initial triangulation. This algorithm first adds edges to Ω to subdivide it into easy-to-triangulate "monotone" faces.

Ruppert [111], building on work of Chew [38], gave a mesh-generation algorithm based on constrained Delaunay triangulation. (Subsequently, Mitchell [90] sharpened Ruppert's analysis, and Shewchuk [117, 118] further refined the algorithm and made an implementation available on the Web.) Ruppert's 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 this 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 guarantee: all new angles, that is, 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 touches two nonadjacent edges of the boundary; this is a spacing function intrinsic to the domain.

5.3 Quadtrees

A quadtree mesh generator [8, 25, 143] 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 [25]. 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 squares until each leaf square contains at most one connected component of Ω's boundary, with at most one vertex. Mitchell and Vavasis [91] improved the splitting phase by "cloning" squares that intersect Ω in more than one connected component, so that each copy contains only a single connected component of Ω. The algorithm then splits squares near vertices of Ω two 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, increasing the total number of leaf squares by a constant factor (at most 8). Squares are

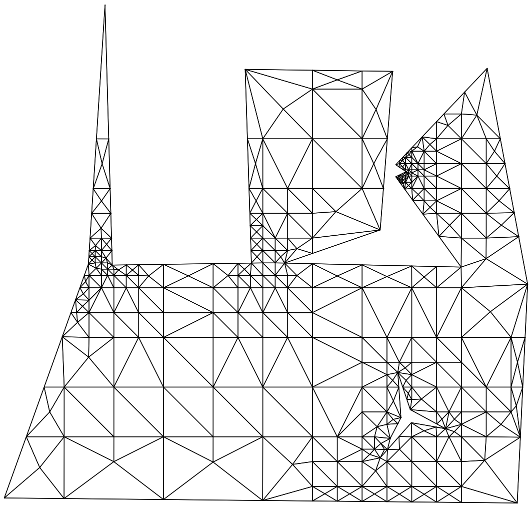

Figure 11. A mesh computed by a quadtree-based algorithm. (S. Mitchell)

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 to v
    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 11 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 guarantee as Ruppert's algorithm. In fact, the quadtree algorithm was the first to be analyzed in this way [25].

5.4 Mesh Refinement and Derefinement

Adaptive mesh refinement places more grid points in areas where the PDE solution error is large. Local error estimates based on an initial solution are known as *a posteriori* error estimates [7] 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 [138].

One approach to mesh refinement [71] iteratively inserts extra vertices into the triangulation, typically at edge bisectors or triangle circumcenters as in Section 5.2. New vertices along the boundaries of curved domains should be computed using the curved boundary

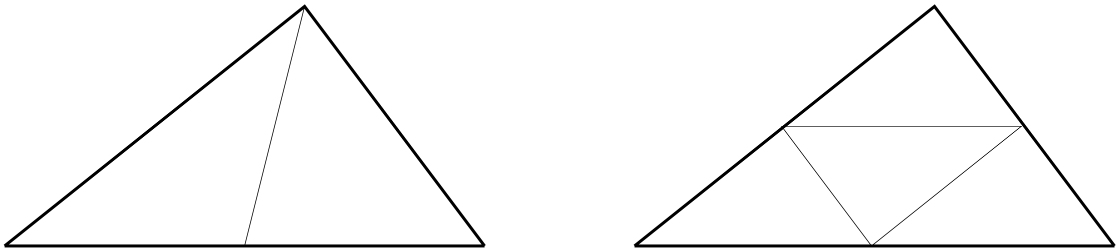

Figure 12. A triangle divided by (a) bisection, and (b) regular refinement.

rather than the current straight edge, thereby giving a truer approximation of the domain as the mesh refines [36]. Iterative vertex insertion may be viewed as a mesh improvement step (Section 5.5), and indeed several generators [29, 119, 139] have combined insertion/deletion, flipping, and smoothing into a single loop.

Iterative vertex insertion gives a finer mesh, but not a *nesting* or *edge conforming*, refinement of the original mesh, meaning a mesh that includes the boundaries of the original triangles. Nesting refinements simplify the interpolation step in 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, along with adjacent invalid triangles to get 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* [11, 12], the midpoints of the sides of a marked triangle are connected, as in Figure 12(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 12(a). Before the next iteration of $\star$, bisected triangles are glued back together and then 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. Thus, 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. Hence the angles in T_{k+1} are bounded away from 0 and π.

Rivara [107, 108, 109] 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 contains an extra vertex; these triangles are then bisected in the same way. Bisections

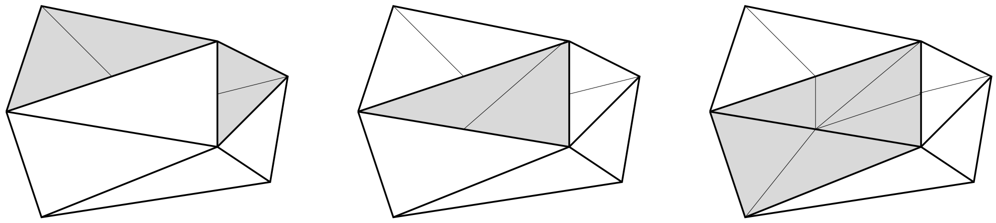

Figure 13. The bisection algorithm given in the pseudocode splits invalid children of refined triangles to their subdivision points, rather than to their longest edges

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 only a finite number of different triangle shapes—*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 13. 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 denotes "marked" triangles to be refined* }
$R_i = \emptyset$ { *R 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 children of Q_i triangles 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 [98].

Figure 14 shows a sequence of meshes computed by a coarsening algorithm due to Ollivier-Gooch. The algorithm marks a set of vertices to delete 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 are typically easier to coarsen than are less hierarchical meshes such as Delaunay triangulations. Teng, Talmor, and Miller [87] have recently devised an algorithm using Delaunay triangulations of well-spaced point sets, which produces a sequence of bounded-aspect-ratio, node-conforming meshes of approximately minimum depth.

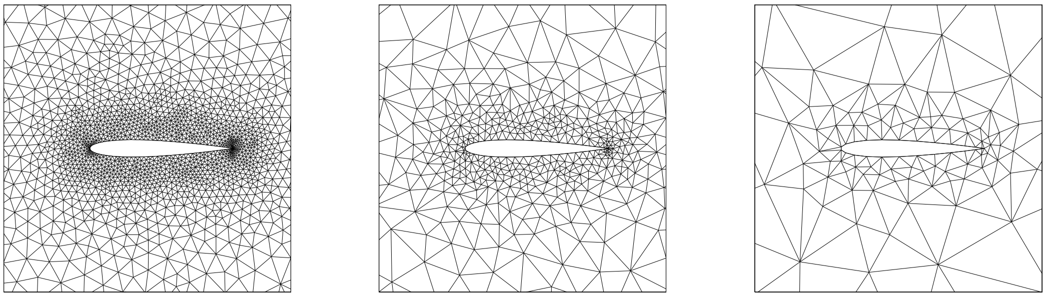

Figure 14. A sequence of meshes used by the multigrid method for solving the linear systems arising in modeling airflow over an airfoil. (C. Ollivier-Gooch)

5.5 Mesh Improvement

The most common mesh improvement techniques are flipping and smoothing. These techniques have proved to be very powerful in two dimensions, and together they can transform very poor meshes into very good ones, so long as the mesh starts with enough vertices.

Flipping exchanges the diagonals of a triangulated quadrilateral as in the edge flipping algorithm for computing Delaunay triangulation (Section 5.1), only the criterion for making the exchange need not be the Delaunay empty circle test. Flipping can be used to regularize vertex degrees, minimize the maximum angle, or improve almost any other quality measure of triangles. For quality measures optimized by the Delaunay triangulation (Section 5.6.1), flipping computes a true global optimum, but for other criteria it computes only a local optimum.

Mesh smoothing adjusts the locations of mesh vertices in order to improve element shapes and overall mesh quality [2, 3, 33, 55, 100]. In mesh smoothing, the topology of the mesh remains invariant, thus preserving important features such as the nonzero pattern of the linear system.

Laplacian smoothing [55, 77] 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. Variations weight each adjacent vertex by the total area of the elements around it, or use the centroid of the incident elements rather than the centroid of the neighboring vertices [139]. Laplacian smoothing is computationally inexpensive and fairly effective, but it does not 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 very expensive [33, 100]. Local techniques adjust vertices one by one—or an independent set of vertices in parallel [58]—resulting in a cost more comparable to Laplacian smoothing. Many quality measures, including maximum angle and area divided by sum of squared edge lengths, can be optimized by techniques related to linear programming [2].

Figure 15 shows the results of a local optimization-based smoothing algorithm developed

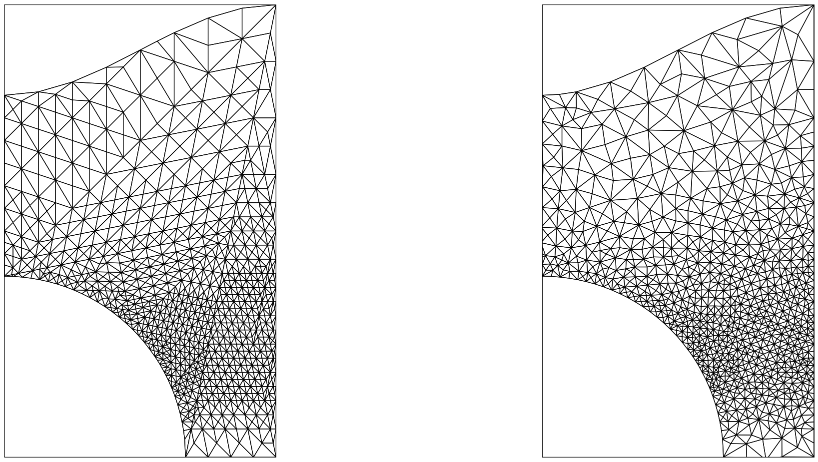

Figure 15. (a) A mesh resulting from bisection refinement without smoothing. (b) The same mesh after local optimization-based smoothing.

by Freitag et al. [58]. The algorithm was 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°.

5.6 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 related to mesh generation.

5.6.1 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 such as maximum angle, minimum angle, maximum edge length, or total edge length. If the input is a simple polygon, most optimal triangulation problems are solvable by dynamic programming in time $O(n^3)$, 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. Thus Delaunay triangulation maximizes the minimum angle, along with optimizing a number of more esoteric quality measures, such as maximum circumcircle radius, maximum enclosing circle radius, and "roughness" of a piecewise-linear interpolating surface [105].

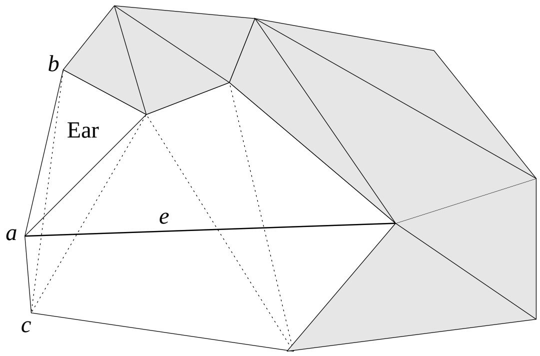

Figure 16. Edge insertion retriangulates holes by removing sufficiently good ears. Dotted lines indicate the old triangulation.

As mentioned in Section 5.5, edge flipping can also be used as a general optimization heuristic. For example, edge flipping works reasonably well for minimizing the maximum angle [53], but it does not in general find a global optimum. A more powerful local improvement method called *edge insertion* [23, 53] 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 minimizing the maximum 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 on the boundary, as shown in Figure 16) 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 [23] 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 [53] and maxmin triangle height.

We close with some results for two other optimization criteria: maximum edge length and total length. Edelsbrunner and Tan [51] showed that a triangulation of a point set that minimizes the maximum edge length 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 $O(n^2)$). Whether a triangulation minimizing total edge length—"minimum weight triangulation"—can be solved in polynomial time is still open. The most promising approach [46] 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.

5.6.2 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 appear to be intractable, typical theoretical results on Steiner triangulation prove either an approximation bound such as the guarantees provided by the mesh generators in Sections 5.2 and 5.3, 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 5.2 and 5.3 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 [9, 135]. Bern, Mitchell, and Ruppert [27] 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 17 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.

It is currently unknown whether multiple domains admit polynomial-size nonobtuse triangulations. Mitchell [93], however, gave an algorithm for triangulating multiple domains using $O(n^2 \log n)$ triangles with maximum angle $157.5°$. Tan [126] improved the maximum angle bound to $132°$ and the complexity to the optimal $O(n^2)$.

6 Hexahedral Meshes

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

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 handle

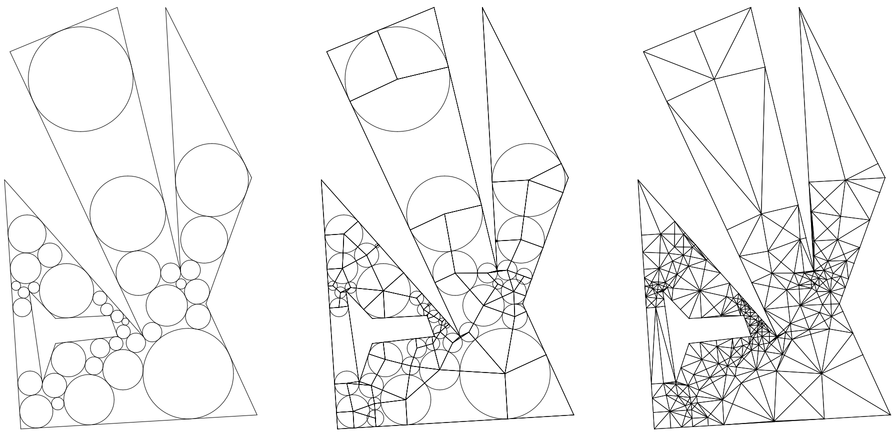

Figure 17. Steps in circle-based nonobtuse triangulation.

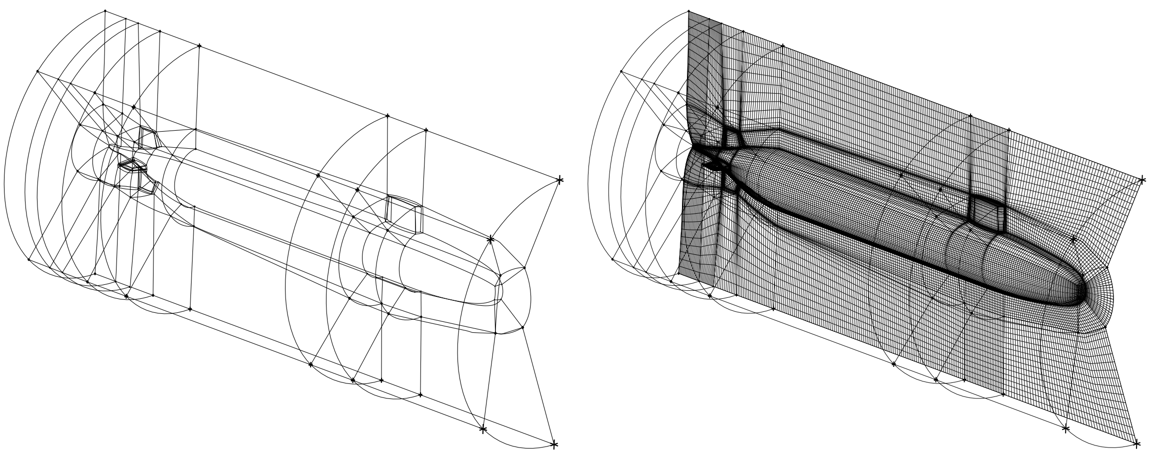

Figure 18. A multiblock hexahedral mesh of a submarine, showing (a) block structure, and (b) a vertical slice through the mesh. (ICEM CFD)

degenerate hexahedra such as prisms [66, 67]. In this section, we describe three approaches to hexahedral mesh generation that vary in their degree of structure and strictness.

6.1 Multiblock Meshes

We start with the approach that produces meshes with the most structure (and quite often the highest quality elements). 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 18 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. Due to the need for human interaction, multiblock meshes are not well suited to adaptive meshing, nor to rapidly iterated design and simulation.

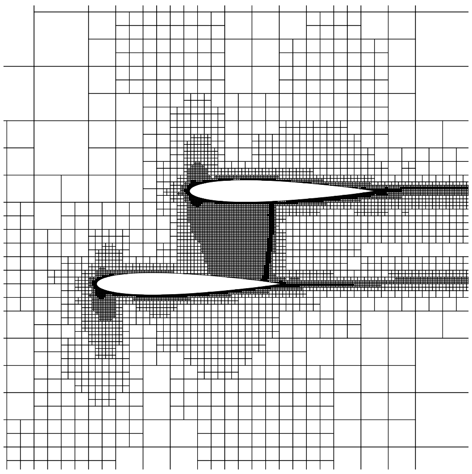

Figure 19. A two-dimensional Cartesian mesh for a biplane wing. (W. Coirier)

6.2 Cartesian Meshes

We move on to 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 then 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 7.3.) 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 [41, 42, 144]. The approach can be adapted even to very difficult meshing problems. For example, Berger and Oliger [21] and Berger and Colella [20] have developed adaptive Cartesian-based methods for rotational flows and flows with strong shocks.

6.3 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 [19]. 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. 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. [4] 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 Voronoï diagram of the faces of the domain; Srinivasan et al. [124] have previously applied this construction to two-dimensional unstructured mesh generation. The medial axis is a natural tool for mesh generation, as advancing fronts 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 [127] are developing an all-hexahedral mesh generation algorithm called *whisker weaving*. Whisker weaving is an advancing front approach that fixes the topology of the mesh before the geometry. It starts from a quadrilateral surface mesh, which can itself be generated by an advancing-front generator within each face [28]. 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 layer of hexahedra in the eventual mesh. A layer of hexahedra can be represented by its dual, called a *sheet*, which has one vertex per hexahedron and edges between adjacent hexahedra. As the algorithm runs, it fills in sheets from the boundary inwards.

This approach to hexahedral meshing raises an interesting theoretical question: which quadrilateral surface meshes can be extended to hexahedral volume meshes? Mitchell [94] and Thurston [132] (see also Eppstein [54]) answered this question in a topological sense by showing that any surface mesh on a simple polyhedron with an even number of quadrilaterals can be extended to a volume mesh formed by (possibly curved) topological cubes. The geometric question remains open.

7 Tetrahedral Meshes

Tetrahedra have several important advantages over hexahedra: unique linear interpolation from vertices to interior, greater flexibility in fitting complicated domains, and ease of 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 5 can be generalized to tetrahedral mesh generation, but not without some new difficulties. Before describing Delaunay, advancing front, and octree mesh generators 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 20(a) gives an example of a non-tetrahedralizable polyhedron, a twisted triangular prism in which each rectangular face has been triangulated so that it bends 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 [37] gave a quantitative bad example, shown in Figure 20(b). This polyhedron includes $\Omega(n)$ grooves 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.

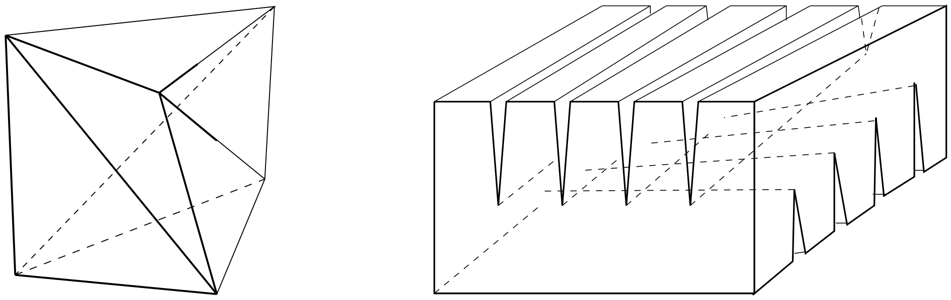

Figure 20. (a) Schönhardt's twisted prism cannot be tetrahedralized without Steiner points. (b) Chazelle's polyhedron requires $\Omega(n^2)$ Steiner points.

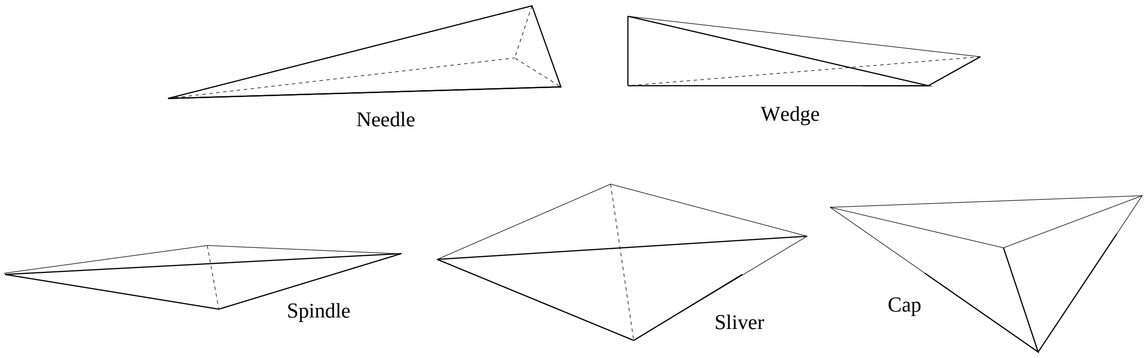

Figure 21. The five types of bad tetrahedra.

Second, the very same domain may be tetrahedralized with different numbers of tetrahedra. For example, a cube can be triangulated with either five or six 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 [22]. There are then five types of bad tetrahedra, as shown in Figure 21. 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 $(0,0,0), (0,1,\epsilon), (1,0,\epsilon)$, and $(1,1,0)$, where $\epsilon \to 0$.

Many measures of tetrahedron quality have been proposed [75], 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. A weaker measure, which forbids all types except slivers, is the ratio of the minimum edge length to the radius of the circumsphere [88].

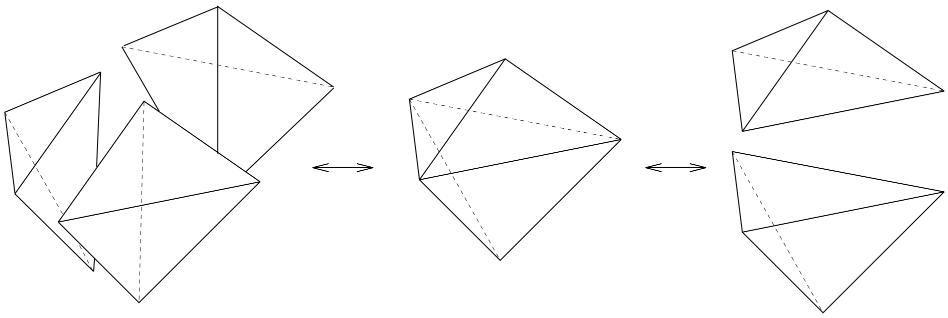

Figure 22. In three dimensions, an edge flip exchanges three tetrahedra sharing an edge for two tetrahedra sharing a triangle, or vice versa.

7.1 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 [68], advancing front [10, 78, 79], and random scattering with filtering [137]. 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 the three-dimensional conforming Delaunay triangulation problem is not too hard for most domains of practical interest, we do not know of 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 [10] include a special postprocessing step that finds and removes slivers. Chew (personal communication) has recently devised an algorithm that removes slivers by adding Steiner points at a random location near their circumcenters.

The triangulation phase of mesh generation also becomes somewhat more difficult in three dimensions. The generalization of edge flipping exchanges the two possible triangulations of five points in convex position, as shown in Figure 22. We call a flip a *Delaunay flip* if, after the flip, the triangulation of the five points satisfies the empty sphere condition—no 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 [69], and it is currently unknown whether any tetrahedralization can be tranformed into the Delaunay triangulation by arbitrary flips. Nevertheless, there are provably correct, incremental Delaunay triangulation algorithms based on edge flipping [50, 70, 104].

There are other practical three-dimensional Delaunay triangulation algorithms as well. Bowyer [30] and Watson [136] gave incremental algorithms with reasonable expected-case performance. Barber [15] implemented a randomized algorithm in arbitrary dimension. This algorithm can be used to compute Delaunay triangulations through a well-known reduction [31] which "lifts" the Delaunay triangulation of points in $\mathbb{R}^d$ to the convex hull of points in $\mathbb{R}^{d+1}$.

7.2 Advancing Front

We have already mentioned an advancing front approach to placing Steiner points for Delaunay triangulation. A pure advancing front mesh generator [77, 79, 97, 101] places the elements themselves, rather than just the Steiner points. This approach gives more direct control of element shapes, especially near the boundary, which is often a region of special interest. Advancing front generators seem to be especially popular in aerodynamics simulations [64, 65, 79, 85, 101].

We describe an advancing front algorithm of Löhner and Parikh [79, 80] as it contains the essential ideas. Desired element size (and perhaps stretching directions) are defined at the vertices of a coarse 'background" tetrahedralization and interpolated to the rest of the domain. The background mesh can also be defined by an octree, the three-dimensional generalization of a quadtree. To get started, the boundaries of the domain are triangulated; the initial front consists of the boundary faces. The algorithm then iteratively chooses a face of the front and builds a tetrahedron over that face. The algorithm attempts to fill in clefts left by the last layer of tetrahedra before starting the next layer; within a layer, the algorithm chooses small faces first in order to minimize collisions. The fourth vertex of the tetrahedron will be either an already existing vertex or a vertex specially created for the tetrahedron. In the latter case, the algorithm tries to choose a smart location for the new vertex; for example, the new vertex may be placed along a normal to the base face at a distance determined by aspect ratios and length functions ultimately derived from the background triangulation [59]. In either case, cleft or new vertex, the tetrahedron must be tested for collisions before final acceptance.

Figure 23 shows the surface of a fairly isotropic tetrahedral mesh computed by an advancing front mesh generator developed by ANSYS, Inc. This generator, like the one just described, places elements directly.

Marcum and Weatherill [81] have devised an algorithm somewhere between pure advancing front and advancing-front point placement followed by Delaunay triangulation. Their algorithm starts with a coarse mesh, and then uses advancing front to place additional Steiner points, simply subdividing the coarse tetrahedra to maintain a triangulation. This mesh is then improved first by Delaunay and then by minmax-solid-angle flips. Other researchers agree that applying flips in this order is more effective than using either type of flip alone.

7.3 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 [134].

Shephard and his collaborators [114, 116, 115, 142] have developed several octree-based mesh generators for polyhedral domains. Their original generator [142] 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

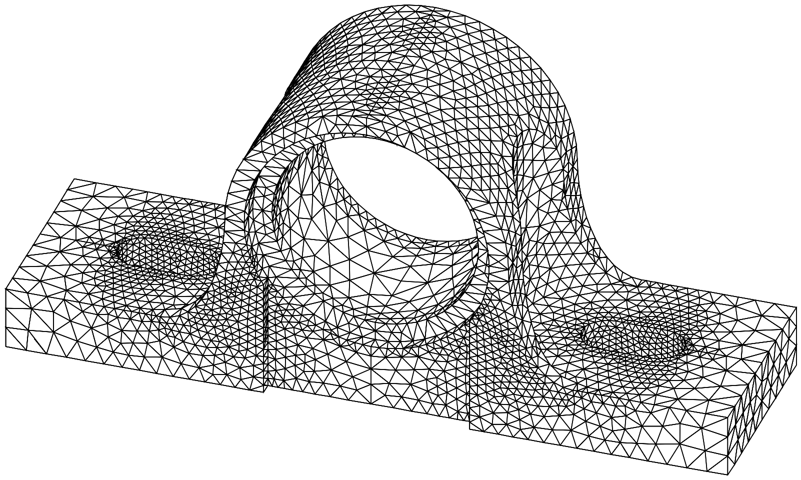

Figure 23. The surface of a tetrahedral mesh computed by an advancing front generator. (ANSYS, Inc.)

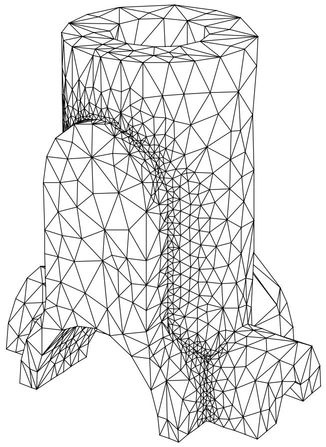

Figure 24. The surface of a tetrahedral mesh derived from an octree. (M. Yerry and M. Shephard)

of the input polyhedron. Perucchio et al. [102] give a more sophisticated way to conform to boundaries. Buratynski [32] uses rectangular octrees and a hierarchical set of warping rules. The octree is refined so that each domain edge intersects boxes of only one size. Boxes are warped to domain vertices, then edges, and finally faces.

Mitchell and Vavasis [91] generalized the quadtree mesh generator of Bern et al. [25] 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 [134] has recently released a modified version of the algorithm (called QMG for "Quality Mesh Generator"), 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 [92].

7.4 Refinement and Improvement

We discuss improvement before refinement, because less is known on the subject. As we mentioned above, edge flipping generalizes to three dimensions, and flipping first by the Delaunay empty sphere criterion and then by the minmax solid angle criterion seems to be fairly effective. Laplacian smoothing also generalizes, although experimental results [57] indicate that it is no longer as effective as in two dimensions. Optimization-based smoothing [2, 57] appears to be more powerful than simple Laplacian smoothing. Freitag and Ollivier-Gooch [57] recommend combining Delaunay flipping with smoothing for maxmin dihedral angle or maxmin dihedral-angle sine.

We now move on to refinement and 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 25. 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.

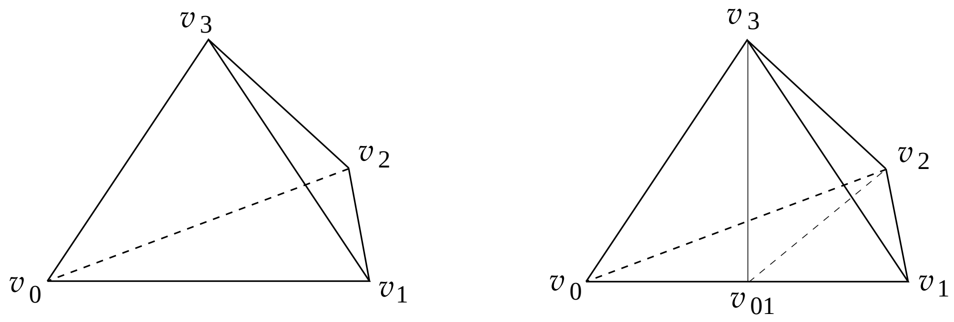

Figure 25. The tetrahedron on the left is bisected to form two new tetrahedra

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 [110] suggested an extension of longest-edge Rivara refinement (Section 5.4) 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, meaning 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 guarantee has not been proved. The guarantee would follow if it could be shown that the algorithm generates only a finite number of tetrahedron similarity classes.

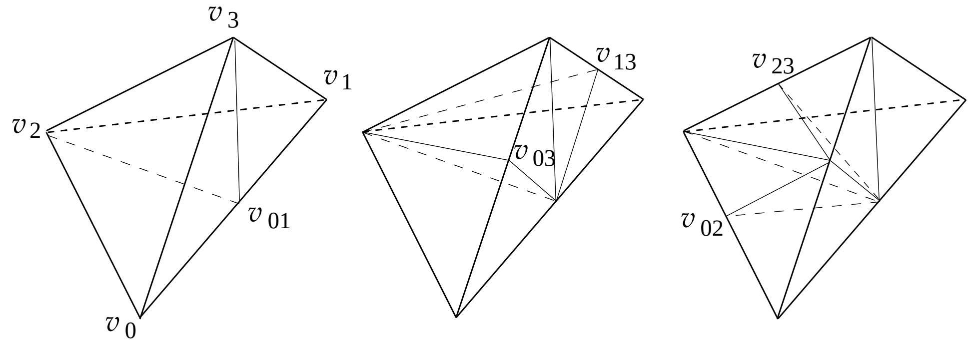

Figure 26. The first three levels of longest-edge bisection of the canonical tetrahedron. Note that the tetrahedra generated at each level are similar. For the final level of refinement we show only the four tetrahedra obtained from $v_0v_{01}v_2v_3$. Four similar tetrahedra are obtained from $v_{01}v_1v_2v_3$.

A bisection algorithm first introduced by Bänsch [14] does indeed generate only a finite number of similarity classes. Before describing the algorithm, we sketch the argument of Liu and Joe [74] 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 26 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. Refinement in the canonical space induces a refinement in the original space with only a finite number of different similarity classes. A subtlety: the similarity classes in the original space correspond to homothets (identical up to scaling and translation), not similarity classes, in the canonical space. Hence the number of similarity classes turns out to be 36 rather than 8 [5, 83].

Bänsch [14], Liu and Joe [76], and Arnold et al. [5] give essentially equivalent [71] 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 pairs of 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 25, 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 $(T_i \neq \emptyset)$ **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
  $i = i + 1$
  let T_i be the set of invalid tetrahedra
enddo

8 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. We also expect—and recommend—computational geometers to focus some attention on structured meshes and hexahedral meshes.

We close with a short list of open problems of both practical and theoretical interest. It is no coincidence that these problems focus on three-dimensional mesh generation.

1. Is the flip graph for a point set in $\mathbb{R}^3$ connected? In other words, is it possible to convert any triangulation of a point set (even a point set in convex position) into any other using only flips (Figure 22)?

2. Is there a smoothing algorithm guaranteed to remove slivers? A sliver (Figure 21) is the only type of bad tetrahedron with well spaced vertices and small circumspheres.

3. Is there an algorithm for conforming Delaunay triangulation in $\mathbb{R}^3$? In other words, place vertices on the boundary of a polyhedron, so that the Delaunay triangulation of all vertices, original and new, contains the polyhedron.

4. Is there an algorithm for unstructured tetrahedral mesh generation that guarantees an M-matrix for the finite element formulation of Poisson's equation?

5. Give an algorithm for computing the blocks in a multiblock mesh. Such an algorithm should give a small number of nicely shaped blocks, quadrilaterals in $\mathbb{R}^2$ and hexahedra in $\mathbb{R}^3$.

6. Can any quadrilateral surface mesh with an even number of quadrilaterals be extended to a hexahedral volume mesh?

Ackonwledgements

We would like to thank Lori Freitag, Paul Heckbert, Scott Mitchell, Carl Ollivier-Gooch, Jonathan Shewchuk, and Steve Vavasis for help in preparing this survey.

References

[1] M. Aftosmis, J. Melton, and M. Berger. Adaptation and surface modeling for Cartesian mesh methods. AIAA Paper 95-1725, 12th AIAA CFD. Conf., San Diego, CA. June, 1995, 1995.

[2] N. Amenta, M. W. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *Proc. 8th ACM-SIAM Symp. Disc. Algorithms*, pages 528–537, 1997.

[3] 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.

[4] 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.

[5] D. N. Arnold, A. Mukherjee, and L. Pouly. Locally adapted tetrahedral meshes using bisection. Manuscript, 1997.

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

[7] I. Babuška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM J. Numer. Analysis*, 15:736–754, 1978.

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

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

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

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

[12] R. E. Bank, A. H. Sherman, and A. 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.

[13] R. E. Bank and R. K. Smith. Mesh smoothing using a posteriori error estimates. *SIAM J. Num. Anal.*, to appear. ftp://math.ucsd.edu/pub/scicomp/reb/ftpfiles/a67.ps.Z.

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

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

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

[17] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, 1994.

[18] 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.

[19] 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*, pages 179–191. Sandia National Laboratories, 1995.

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

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

[22] 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.

[23] 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.

[24] 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.

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

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

[27] M. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. *Disc. and Comp. Geometry*, 14:411–428, 1995.

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

[29] F. Bossen. Anisotropic mesh generation with particles. Technical Report CMU-CS-96-134, Carnegie-Mellon University, School of Computer Science, 1996. http://ltswww.epfl.ch/~bossen/.

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

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

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

[33] S. Canann, M. Stephenson, and T. Blacker. Optismoothing: an optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design*, 13:185–190, 1993.

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

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

[36] 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.

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

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

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

[40] 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.

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

[42] 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.

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

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

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

[46] M. T. Dickerson and M. H. Montague. A (usually?) connected subraph of the minimum weight triangulation. In *Proc. 12th ACM Symp. Comp. Geometry*, pages 204–213, 1996.

[47] T. A. Driscoll. A Matlab toolbox for Schwarz-Christoffel mapping. *ACM Transactions on Mathematical Software*, to appear.

[48] 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.

[49] 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.

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

[51] 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.

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

[53] 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.

[54] D. Eppstein. Linear complexity hexahedral mesh generation. In *Proc. 12th ACM Symp. Comp. Geom.*, pages 58–67, 1996.

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

[56] 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.

[57] L. Freitag and C. Ollivier-Gooch. A comparison of tetrahedral mesh improvement techniques. In *Proc. 5th International Meshing Roundtable*, pages 87–100. Sandia National Laboratories, 1996. http://sass577.endo.sandia.gov:80/9225/Personnel/samitch/roundtable96/.

[58] L. A. Freitag, M. T. Jones, and P. E. Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proc. 4th International Meshing Roundtable*, pages 47–58. Sandia National Laboratories, 1995.

[59] P. J. Frey, H. Borouchaki, and P.-L. George. Delaunay tetrahedralization using an advancing-]front approach. In *Proc. 5th International Meshing Roundtable*, pages 31–46. Sandia National Laboratories, 1996. http://www.cs.cmu.edu/~ph.

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

[61] 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.

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

[63] 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.

[64] 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.

[65] 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.

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

[67] T. J. R. Hughes and J. E. Akin. Techniques for developing 'special' finite element shape functions with particular reference to singularities. *Int. J. Numer. Meth. Eng.*, 15:733–751, 1980.

[68] 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.

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

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

[71] M. T. Jones and P. E. Plassmann. Adaptive refinement of unstructured finite-element meshes. *Finite Elements in Analysis and Design*, 25(1-2):41–60, March 1997.

[72] M. S. Khaira, G. L. Miller, and T. J. Sheffler. Nested dissection: a survey and comparison of various nested dissection algorithms. Technical Report CMU-CS-92-106R, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.

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

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

[75] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT*, 34:268–287, 1994.

[76] A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM J. on Scientific Computing*, 16(6):1269–1291, November 1995.

[77] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *Int. J. Numer. Meth. Eng.*, 21:1403–1426, 1985.

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

[79] R. Löhner and P. Parikh. Three-dimensional grid generation via the advancing-front method. *Int. J. Numer. Meth. Fluids*, 8:1135–1149, 1988.

[80] Rainald Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[81] D. L. Marcum and N. P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal*, 33(9):1619–1625, 1995.

[82] C. W. Mastin. Elliptic grid generation and conformal mapping. In José E. Castillo, editor, *Mathematical Aspects of Grid Generation*, pages 9–18. Society for Industrial and Applied Mathematics, Philadelphia, 1991.

[83] J. M. Maubach. The number of similarity classes created by local n-simplicial bisection refinement. Manuscript, 1996.

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

[85] D. J. Mavriplis. Unstructured mesh generation and adaptivity. Technical Report ICASE 95-26, NASA Langley, Hampton VA, 1995. Abstract at http: //techreports.larc.nasa.gov/cgi-bin/NTRS.

[86] D. J. Mavriplis. Mesh generation and adaptivity for complex geometries and flows. In Roger Peyret, editor, *Handbook of Computational Fluid Mechanics.* Academic, London, 1996.

[87] G.L. Miller, D. Talmor, and S.-H. Teng. Optimal coarsening of unstructured meshes. In *Proc. 8th ACM-SIAM Symp. Disc. Algorithms*, 1997.

[88] 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.

[89] 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.

[90] S. A. Mitchell. Cardinality bounds for triangulations with bounded minimum angle. In *Sixth Canadian Conference on Computational Geometry*, 1994.

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

[92] 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.*, pages 48–57, 1996.

[93] 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.

[94] 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.

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

[96] 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.

[97] Nguyen-Van-Phai. Automatic mesh generation with tetrahedral element. *Int. J. Numer. Meth. Eng.*, 18:273–289, 1982.

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

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

[100] 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.

[101] J. Peraire, J. Peiro, L. Formaggia, K. Morgan, and O.C. Zieniewicz. Finite element Euler computations in three dimensions. *Int. J. Numer. Meth. Eng.*, 26:2135–2159, 1988.

[102] 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.

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

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

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

[106] S. Rippa. Long and thin triangles can be good for linear interpolation. *SIAM J. Numer. Analysis*, 29:257–270, 1992.

[107] M.-C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Int. J. Numer. Meth. Eng.*, 20:745–756, 1984.

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

[109] M.-C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM J. Numer. Analysis*, 21(3):604–613, June 1984.

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

[111] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, May 1995.

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

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

[114] 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.

[115] M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Int. J. Numer. Meth. Eng.*, 32:709–749, 1991.

[116] 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.

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

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

[119] K. Shimada. *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing.* PhD thesis, ME Dept., MIT, 1993.

[120] 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.

[121] R. B. Simpson. Anisotropic mesh transformations and optimal error control. *Applied Numer. Math.*, 14(1-3):183–198, 1994.

[122] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Algorithms for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.

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

[124] 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.

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

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

[127] T. J. Tautges and S. A. Mitchell. The whisker weaving algorithm for constructing all-hexahedral finite element meshes. In *Proc. 4th International Meshing Roundtable*. Sandia National Laboratories, 1995.

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

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

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

[131] J. F. Thompson and N. P. Weatherill. Aspects of numerical grid generation: current science and art. In *Proc. 11th AIAA Applied Aerodynamics Conference*, pages 1029–1070, 1993.

[132] W. Thurston. Re: Hexahedral decompostion of polyhedra, a posting to sci.math newsgroup. http: //www.ics.uci.edu/~eppstein /junkyard/Thurston-hexahedra, 1993.

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

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

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

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

[137] 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.

[138] 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.

[139] W. Welch. *Serious Putty: Topological Design for Variational Curves and Surfaces.* PhD thesis, CS Dept, Carnegie Mellon University, Dec. 1995. CMU-CS-95-217, ftp: //reports.adm.cs.cmu.edu/usr/anon/1995/CMU-CS-95-217A.ps, 217B.ps, 217C.ps.

[140] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.

[141] J. Xu and L. Zikatanov. A monotone finite element scheme for convection diffiusion equations. *Math. of Computation*, to appear.

[142] 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.

[143] 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.

[144] 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.

[145] R. Young and I. MacPhedran. Internet finite element resources. http: //www.engr.usask.ca/ ~macphed/finite/fe_resources/fe_resources.html, 1995.