Adaptive Mesh Generation

Marshall Bern

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

Abstract. These notes cover topics in mesh generation from a computational geometry perspective. This perspective means emphasis on difficiult domain geometry, unstructured triangular and tetrahedral meshes, and provable bounds on quality and complexity. We concentrate on new results, especially those results applicable to solution-adaptive methods in computational fluid dynamics.

1 Introduction

We start with an old story of geometric discretization gone awry. Imagine we discretize a cylinder as shown in Figure 1(a). The picture shows a stack of $n = 4$ congruent prisms, each with $m = 6$ triangulated rectangular sides. As m and n go to infinity, the sum of the areas of the rectangles approaches the surface area of the cylinder just as it should. If $n \gg m$, then the rectangles are long and skinny and the two triangles in each rectangle are *sharp*, meaning having an angle near $0°$.

Now let us discretize the cylinder as shown in Figure 1(b). This picture shows a stack of n "antiprisms", a solid in which the top and bottom m-gons are rotated with respect to each other by a turn of π/m radians. Now as m and n go to infinity, we find something surprising, called the Schwarz paradox. The sum of the areas of the triangles depends upon exactly how fast m and n go to infinity. If $n/m^2 \rightarrow 0$ then the sum converges as it should to the area of the cylinder. But if $n/m^2 \rightarrow \infty$, then the sum diverges, and if n/m^2 is somewhere in between, then the sum could converge to an incorrect finite number. The calculation can be found in [91] or [54]. Now $n \gg m$ implies not only that the triangles are sharp, but even worse, that they are *flat*, meaning having an angle near $180°$.

Notice that any fixed bound on the maximum angle of a triangle less than $180°$ prevents the "paradox". For example, as a triangle with maximum angle $170°$ shrinks to be much smaller than the radius of the cylinder, it must flatten onto the cylinder's surface. Thus a triangulation in which the overall maximum angle is at most $170°$ and the longest edge length is small necessarily approximates the surface area of the cylinder.

Also notice that it is not the flat angles themselves that are the problem; it is the combination of the flat angles and the directions of the triangles. For if we triangulate the cylinder as shown in Figure 1(c), we do not encounter the paradox, no matter how flat the triangles.

With this story to whet our appetite, we now give an overview of the material in this survey article. We start with a discussion of element shape (Section 2). We then give a

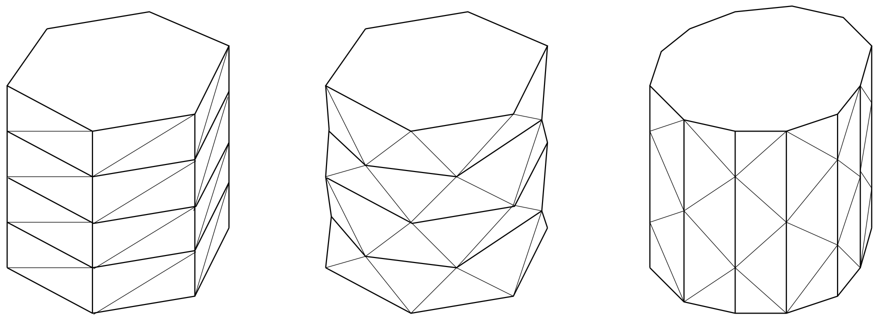

Fig. 1. A cylinder discretized in three different ways: (a) sharp horizontal triangles, (b) flat horizontal triangles, (c) flat vertical triangles.

short course on Delaunay triangulation in Sections 3 and 4. (Some people think that computational geometry *is* the study of Delaunay triangulation!) We consider point placement methods next, first point placement dictated by domain geometry (Section 5) and then solution-adaptive refinement (Section 6). We continue with mesh simplification in Section 7. Finally we look at hierarchical meshing such as Cartesian meshes and subdivision surfaces (Section 8) and data compression of meshes (Section 9). The author of this article has co-authored two previous surveys of mesh generation [21, 27]. This article concentrates on new results, and its overlap with the previous two should be less than 50%. For the most up-to-date information on mesh generation, we recommend the Web [61, 77, 87, 102].

2 Element Shape

The cylinder example of Figure 1 should motivate us to be at least a little bit careful about the shapes of elements used in discretization of geometry. 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 to converge 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 sharp and flat elements as in the cylinder example. Early results [38] showed convergence of finite element methods as triangular elements shrink, assuming that all angles are bounded away from $0°$. In a frequently cited paper, Babuška and Aziz [9] showed convergence assuming only that angles are bounded away from $180°$, a weaker condition. There have been a number of more recent analyses of interpolation error as well. Some analyses [28, 93] include information about the finite element solution into triangle quality estimators.

Bank and Smith [15] gave a not-just-asymptotic analysis of interpolation error and suggested a quality measure that only slightly favors sharp triangles over flat triangles: the area of a triangle divided by the sum of squared edge lengths.

In three dimensions, the equivalent of a sharp triangle is a tetrahedron with a small solid angle and the equivalent of a flat triangle is a tetrahedron with a small dihedral angle. Thus, the generalization of Babuška-Aziz shows convergence assuming that dihedrals are bounded away from $0°$, even if some solid angle is small (as in the 1-3 tetrahedron in Figure 16).

2.1 Anisotropy

Sometimes 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 cylinder serves as a simple example: relatively few sharp or flat vertical triangles (that is, $m \gg n$, even $n = 1$) are needed for a good approximation of the surface area. The ideal aspect ratio of a triangle is the square root of the ratio of the larger to smaller eigenvalue of the Hessian [93]. 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 much worse than small [93].

Fluid flow problems, especially Navier-Stokes simulation, 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 finite volume and finite difference formulations of these problems, because such meshes allow faces of elements in the boundary layers to be either almost parallel or almost orthogonal to the surface. (There is still some debate over exactly how advantageous are quad and hex meshes [12].) However, it may be difficult or impossible to mesh complicated domains with hex boundary layers.

Hybrid meshes with a mix of element shapes are a practical solution. In two dimensions, one can use quad boundary layers along with triangles away from the boundary. In three dimensions, hex boundary layers usually require some other sorts of elements, such as prisms or pyramids, in order to transition to tetrahedra. A more esthetic sort of three-dimensional hybrid mesh uses triangular prisms in its boundary layers and tetrahedra away from the boundary. Hybrid meshing is gaining popularity in fluid dynamics research [114] and in industry [33, 51], but has so far been neglected by the computational geometry community.

Simulations with shock fronts—for example, supersonic air flow over a wing—are also strongly anisotropic. In this case, the locations and directions for high-aspect-ratio elements cannot be predicted in advance. The need for solution-adaptivity—local remeshing based on an initial solution—now tilts the balance back in favor of triangles and tetrahedra. As we shall see in Section 6, triangles and tetrahedra can be refined and derefined (coarsened) much more easily than quads, hexes, or prisms. Hybrid meshes should be a workable solution even with shock fronts, however, because hex or prism surface layers are not as hard to refine as unstructured hex meshes.

2.2 M-Matrix

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 $180°$ [17, 39]. This requirement implies that no quadrilaterals are "reversed" (Section 3), 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 for a finite element formulation 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 [119]. All such sums will be nonnegative if all dihedrals in the mesh are nonobtuse, but this simplified condition is more restrictive than necessary. The condition for obtaining an M-matrix is fairly demanding for a mesh generator (or maybe just difficult to work with); there is no mesh generation algorithm for polyhedral domains that is guaranteed to produce an M-matrix.

For a finite volume formulation using Voronoi control volumes, a sufficient condition for obtaining an M-matrix is that each triangular face and each tetrahedral element be *self-centered*, meaning that it contains its circumcenter [90]. Again nonobtuse dihedrals suffices but is overly restrictive. And as in the finite element case, there is no mesh generation algorithm guaranteed to produce an M-matrix.

3 Voronoi Diagrams and Delaunay Triangulations

Our first topic is mesh generation based on Delaunay triangulation. We start by reviewing the basic geometric constructions. Bear with us if the initial material seems familiar, as we shall soon come to some constructions not so well-known outside computational geometry. In later sections, we shall see how to apply these constructions in mesh generation.

Let $S = \{ s_1, s_2, \ldots, s_n \}$ be a set of distinct points, called *sites*, in the Euclidean plane. The *Voronoi cell* $V(s_i)$ for site s_i is the set of all points at least as close to s_i as to any other site:

$$V(s_i) = \bigcap_{j \neq i} \{ p \in \mathbb{R}^2 \mid d(p, s_i) \leq d(p, s_j) \}.$$

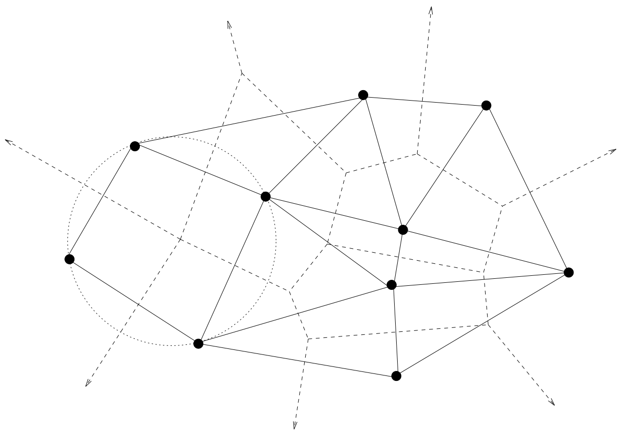

Fig. 2. The Voronoi diagram and Delaunay triangulation of points in the plane. This case includes four cocircular points, and hence the Delaunay triangulation must be completed to form a proper triangulation.

The region of the plane at least as close to s_i as to s_j is a half-plane, bounded by the perpendicular bisector of line segment $s_i s_j$. Since $V(s_i)$ is the intersection of half-planes, it is a convex (possibly unbounded) polygon. The *Voronoi diagram* of S is the subdivision of the plane into Voronoi cells and shared edges and vertices.

A *planar straight-line graph* consists of a set of vertices in the plane, along with line segments representing graph edges between pairs of vertices; edges meet only at shared endpoints. The *Delaunay triangulation* of S, denoted $DT(S)$, is the planar straight-line graph with vertex set S and an edge between s_i and s_j if and only if $V(s_i)$ and $V(s_j)$ meet at an edge in the Voronoi diagram of S. The Delaunay triangulation is thus the *planar dual* of the Voronoi diagram, with a Delaunay vertex corresponding to a Voronoi face and vice versa. If every vertex in the Voronoi diagram is the meeting point of exactly three Voronoi cells, then the Delaunay triangulation is indeed a triangulation of the convex hull of S. But if four or more Voronoi cells meet at a common point, then the Delaunay triangulation includes a face with four or more sides as shown in Figure 2. This is a degenerate case, occurring only when four or more points of S lie on a common circle, and can be easily *completed* by triangulating the untriangulated face.

We can define the Delaunay triangulation directly, without going through the Voronoi diagram, by the *empty circumcircle condition*: a triangle $s_i s_j s_k$ is a face of $DT(S)$ if and only if the circle through s_i, s_j, and s_k neither passes through nor encloses any other sites. We can also state an equivalent empty circle condition for edges of $DT(S)$: $s_i s_j$ is an edge of $DT(S)$ if and only if there is some circle through s_i and s_j that neither passes through nor encloses any other sites.

Figure 3 shows the two possible triangulations of four points forming a convex quadrilateral. Notice that either both circumcircles are empty or both are not. An equivalent condition to both circumcircles being empty is that the angles at the vertices of the quadrilateral, not incident to the diagonal, sum to less than $180°$. Another equivalent condition is that the Delaunay triangulation is the triangulation that maximizes the minimum an-

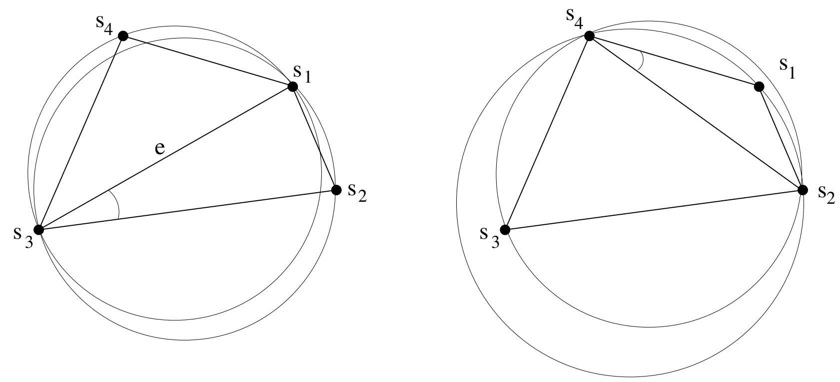

Fig. 3. (a) The Delaunay triangulation of four points forming a convex quadrilateral. (b) The reversed triangulation has smaller minimum angle (marked).

gle in a triangle. In fact, this property holds for arbitrary planar point sets: $DT(S)$ maximizes the minimum angle over all triangulations of S. The Delaunay triangulation actually has a number of global optimality properties, including some rather exotic ones. For example, when used for piecewise-linear interpolation, it maximizes the total "roughness' of the surface [92], meaning the integral of the gradient squared. For another example, if we scale all z-coordinates downwards relative to x and y, at some point before the surface completely flattens out, the Delaunay triangulation gives the least-area piecewise-linear interpolating surface.

Voronoi diagrams and Delaunay triangulations generalize to higher dimensions in a straightforward way. In the absence of degeneracies, a Delaunay triangulation in $\mathbb{R}^3$ is a *tetrahedralization* (sometimes "tetrahedrization") of the convex hull of the sites. Each tetrahedron has an empty circumsphere, meaning that the sphere neither passes through nor encloses any other sites.

3.1 Lifting

It is often useful to view a Delaunay triangulation in $\mathbb{R}^d$ as a projection of a convex polytope in $\mathbb{R}^{d+1}$. Consider *lifting* points in $\mathbb{R}^2$ by projecting them vertically onto the paraboloid Λ in $\mathbb{R}^3$ given by

$$\Lambda = \{\, (x, y, z) \mid z = x^2 + y^2 \,\}.$$

Let $\hat{S}$ be the image of S under this mapping; site $s_i = (x_i, y_i)$ maps to point $\hat{s}_i = (x_i, y_i, z_i = x_i^2 + y_i^2)$. Consider the *lower convex hull* of $\hat{S}$, that is, the part of the convex hull of $\hat{S}$ consisting of faces lying in planes that separate $\hat{S}$ from a point arbitrarily far down the z-axis, denoted $(0, 0, -\infty)$. The Delaunay triangulation $DT(S)$ is the projection of the lower convex hull of $\hat{S}$ onto the xy-plane. See Figure 4.

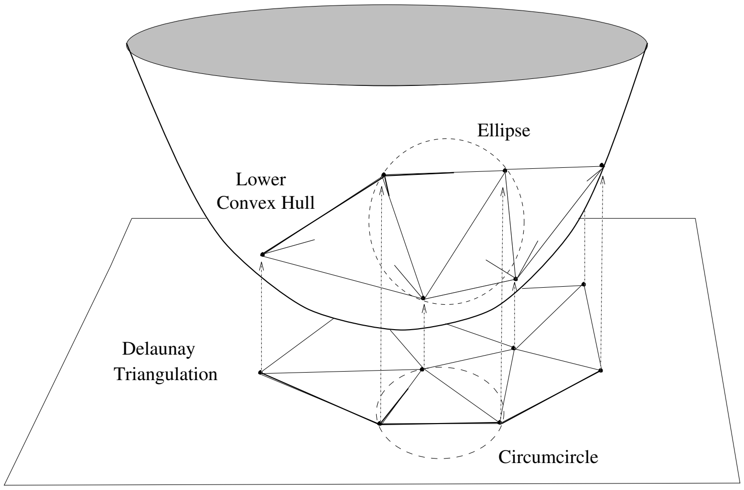

Fig. 4. The lower convex hull of lifted sites projects to the Delaunay triangulation. A plane supporting a face of the lower hull cuts the paraboloid in an ellipse that projects to a circumcircle.

3.2 Power Diagrams and Weighted Delaunay Triangulations

Voronoi diagrams and Delaunay triangulations have been generalized in numerous ways: line segment or polygonal sites, multiplicatively or additively weighted sites, and so forth [8]. Most of these generalizations, however, lose the elegance of the original constructions. For example, imagine that each site s_i is endowed with an additive weight w_i, such that the distance to s_i is the usual Euclidean distance plus w_i. Under this definition, the region closer to s_i than to s_j is bounded by the locus of points (x, y) such that $((x - x_i)^2 + (y - y_i)^2)^{1/2} + w_i = ((x - x_j)^2 + (y - y_j)^2)^{1/2} + w_j$, a hyperbola. Hence, in an additively weighted Voronoi diagram, cells have curved sides.

There is one generalization of Voronoi diagrams and Delaunay triangulations, however, that preserves almost all of the mathematics. This generalization is not very well known to the finite element community, although it is starting to find uses in mesh generation. Define the *power distance* of a point to site s_i to be the square of the Euclidean distance minus the weight w_i. Now the territory closer to $s_i = (x_i, y_i)$ than to $s_j = (x_j, y_j)$ is bounded by the locus of points (x, y) such that $(x - x_i)^2 + (y - y_i)^2 - w_i = (x - x_j)^2 + (y - y_j)^2 - w_j$. This time the x^2 and y^2 terms cancel, and the equation defines a line,

$$2(x_i - x_j)x + 2(y_i - y_j)y + ((x_j - x_i)^2 + (y_j - y_i)^2 + w_i - w_j) = 0. \qquad (1)$$

Hence cells in the *power diagram* are—like Voronoi cells—convex, possibly unbounded, polygons. A power cell, however, may be empty.

We can regard the power diagram as a sort of Voronoi diagram for circular sites C_i, with the radius of site C_i being $\sqrt{w_i}$. For a point p outside the circle, the power distance is the square of the length of a tangent from the circle to p. For a point p inside the circle, the power distance is the square of half the length of the shortest chord through p, only negated. (An interesting geometric fact: the product of the lengths of the two pieces of a chord through p is always the same.) In the case of overlapping circles, the line dividing the power cells given by equation (1) contains the mutual chord. A circle completely enclosed by another circle has an empty cell. See Figure 5.

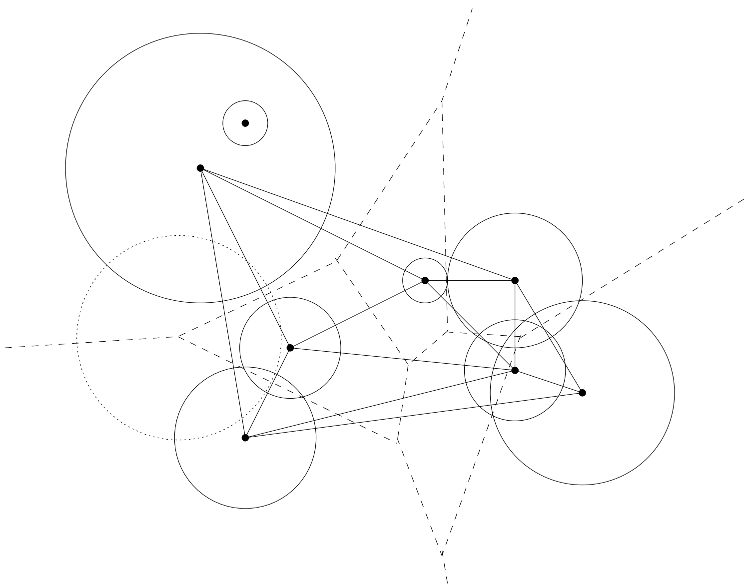

Fig. 5. A power diagram (dashed), along with its dual weighted Delaunay triangulation. The dotted circle is an orthogonal circle.

Following Edelsbrunner [44], we shall call the dual of the power diagram the *weighted Delaunay triangulation* of the set of weighted sites. In general, due to circles with empty cells, the weighted Delaunay triangulation is a graph on a subset of the sites S, but in the special case that all weights are the same, the weighted Delaunay triangulation is exactly the Delaunay triangulation. Unless weights and site locations conspire so that some point has equal power distance to four sites, the weighted Delaunay triangulation is indeed a triangulation. The analogue of the empty circumcircle condition for the weighted Delaunay triangulation is the *empty orthogonal circle condition*: the triangle for circular sites C_i, C_j, and C_k appears in the triangulation if and only if the unique circle C that crosses each of C_i, C_j, and C_k orthogonally crosses no other circles "more deeply" than orthogonal.

The lifting map described above generalizes to weighted Delaunay triangulations. The weighted Delaunay triangulation is the projection of the lower convex hull of points lifted by the map that takes site $s_i = (x_i, y_i)$ to $(x_i, y_i, x_i^2 + y_i^2 - w_i)$. The lifting map is perhaps even more natural for weighted than for unweighted Delaunay triangulations, because it now defines a bijection between weighted points in two dimensions and point sets in three dimensions, whereas it previously mapped only to special sets of points.

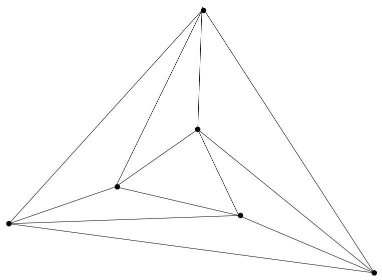

Fig. 6. This triangulation is not the projection of the lower half of a convex polytope, because any set of vertex elevations must make some edge a mountain (or flat) rather than a valley.

Hence weighted Delaunay triangulations are exactly the projections of (lower halves of) convex polytopes. This does not give all triangulations, only the so-called regular triangulations, as shown by the counterexample in Figure 6.

3.3 Algorithms

We now turn to the subject of algorithms for computing Delaunay triangulations. We shall look at two different algorithms, called *edge flipping* and *randomized incremental insertion*.

The edge flipping algorithm, due to Lawson [67], 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 it is triangulated the wrong way, meaning that each triangle's circumcircle contains the opposite vertex as in Figure 3(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, to the left of an imaginary vertical line sweeping across the data. 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 7. The following theorem [43] guarantees the success of edge flipping: a triangulation in which no quadrilateral is reversed must be (a completion of) the Delaunay triangulation. Although the worst-case running time of the edge flipping algorithm is $O(n^2)$ rather than the optimal $O(n \log n)$, it performs reasonably well in practice.

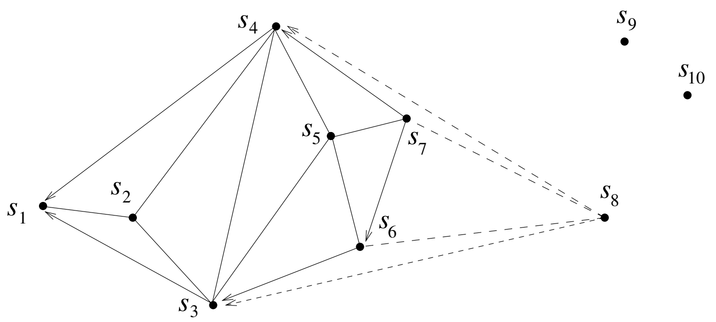

Fig. 7. A sweep algorithm for computing an initial triangulation

Randomized incremental insertion, due to Guibas et al. [55], adds points one at a time in random order, and after each addition flips edges to restore a Delaunay triangulation. The algorithm starts with a bounding box or bounding triangle that contains all the points of S. At the end all the triangles incident to corners of the bounding polygon can be thrown away to give $DT(S)$.

form an initial bounding box B_∞ containing all points
split the bounding box into two triangles
for $i = 1 \dots n$ **do**
  find the triangle Δ that contains s_i
  split Δ into three by adding s_i
  place the outside edges of Δ 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
endfor

The novelty in this algorithm is the point location step in the fourth line of the pseudocode. The suggested solution for this step is quite elegant: the *history* of all triangles ever produced by the algorithm defines a data structure for answering point location queries. We keep the history in a DAG search structure (like a tree, but a node may have more than one parent), with one node for each triangle. Each time we split or flip a triangle, we add new nodes corresponding to the new triangles and make them child nodes of the old triangles, as shown in Figure 8. Notice that all new triangles are incident to the newly inserted vertex, so the flipping does not propagate very far, on the average less than six triangles (the maximum average degree of a planar graph). The fact that S is inserted in random order (rather than say left-to-right) is important to keep the expected height of the DAG for any input to $O(\log n)$ and hence the expected running time of the algorithm to the optimal $O(n \log n)$.

Randomized incremental insertion generalizes to three and higher dimensions, but there are some subtleties. The generalization of edge flipping exchanges the two possible

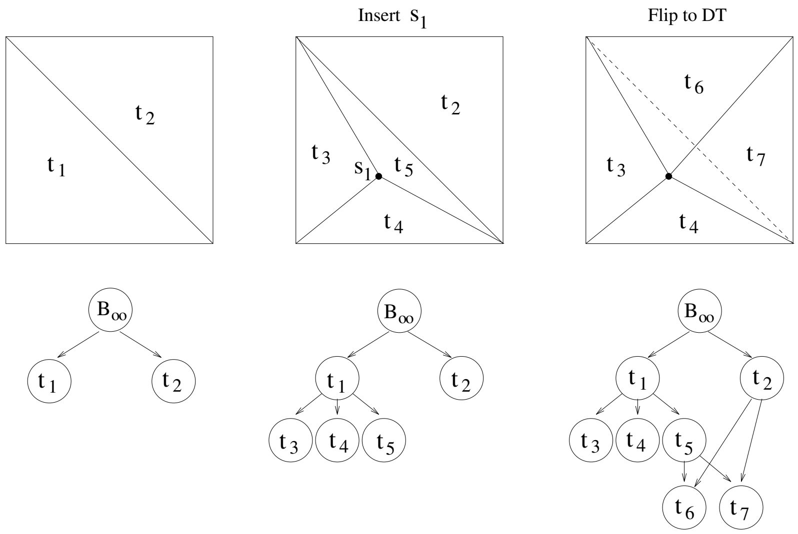

Fig. 8. Randomized incremental insertion uses a history DAG to locate inserted points.

triangulations of five points in convex position, as shown in Figure 9. The two sides of the flip are no longer so symmetric: one side consists of three tetrahedra sharing an edge and the other consists of two tetrahedra sharing a triangle; the flipping part of the algorithm must consider both cases. In Section 5.5 we shall take a more general view of flipping, and flips will include splitting a triangle into three or a tetrahedron into four by adding an interior vertex.

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 triangulation can be transformed into the Delaunay triangulation by a sequence of Delaunay flips; Joe [63] gave a simple example of a tetrahedralization that is not the Delauany triangulation and yet has no Delaunay flip. The triangulations produced by randomized incremental insertion, however, are all nicely behaved (in particular they are all regular triangulations) and the algorithm cannot get stuck.

Finally we say something about computing large 3D Delaunay triangulations. Randomized incremental insertion works quite well for computing the 3D Delaunay triangulation of up to 100,000 points. We obtained the best results [4] with a non-random, strategic order: the first 1000 points are random from all over the mesh in order to sketch out the history DAG, but the rest of the points are kept in some locality-preserving order so as to cut down on the amount of memory paging. For larger point sets, we must be smarter still. Inspired by a parallel algorithm [82], Amenta (personal communication) has proposed a divide-and-conquer approach that repeatedly cuts the points into slightly-overlapping subsets, until the subsets are small enough to fit into the computer memory. The lifting map can be used to make the divisions. We divide the lifted points in $\mathbb{R}^4$ with a vertical hyperplane P, and think of the lifted points as lying on a paraboloid centered on P. Only the lifted points that appear on the convex hull when projected horizontally

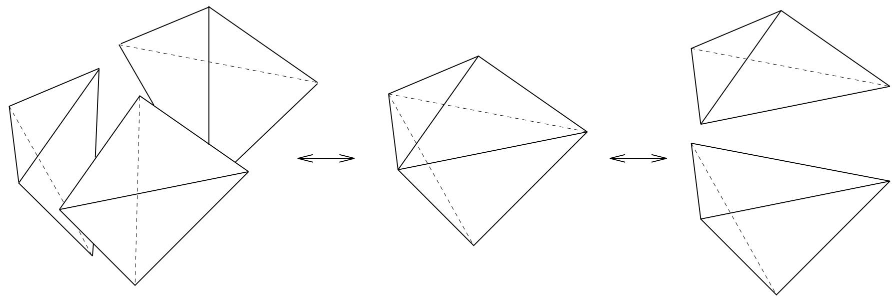

Fig. 9. In three dimensions, an edge flip exchanges three tetrahedra sharing an edge for two tetrahedra sharing a triangle, or vice versa.

onto P need be included in both halves. We add right-half points to the left half (and vice versa) by increasing distance from P; when the projection of the remaining part of the paraboloid lies interior to the convex hull we can stop the sweep.

4 Constrained and Conforming Delaunay Triangulation

In mesh generation, we typically must conform to a domain boundary and not just to a set of vertices. Figure 10 shows that simply taking the Delaunay triangulation of the vertices of a polygonal domain does not suffice. We are now getting into the central problems of mesh generation: conforming to domain geometry, defining initial elements of appropriate sizes and shapes, and adapting a mesh based on a solution. These problems are respectively the topics Sections 4, 5, and 6; of course there is a certain amount of overlap among the three topics.

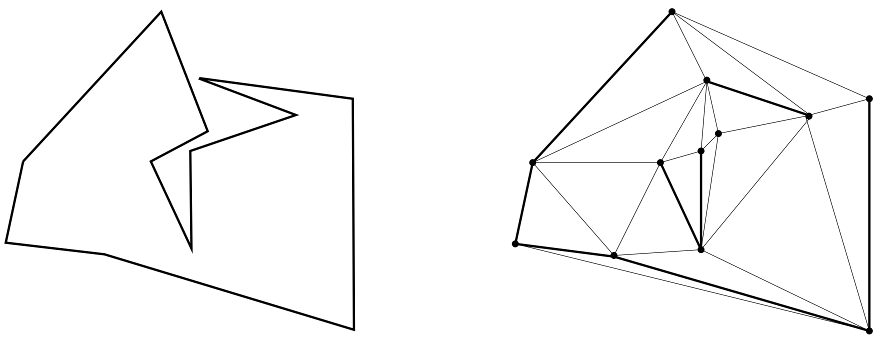

Fig. 10. The Delaunay triangulation of a polygon's vertices may not include all edges.

4.1 Two Dimensions

Let us start with two-dimensional domains. How do we define the Delaunay triangulation of a planar straight-line graph G?

One possible answer is to line the edges of G with closely-spaced vertices, so that the Delaunay triangulation of the augmented vertex set—the extra vertices are often called *Steiner points*—includes all the edges of G. This approach is called *conforming Delaunay triangulation*.

We shall sketch an algorithm due to Edelsbrunner and Tan [48] that uses only $O(n^3)$ Steiner points, regardless of the input geometry. A polynomial-size, geometry independent bound is a theoretical triumph, but such a bound is probably of no great practical importance for this problem because we shall eventually need a geometry-dependent number of interior Steiner points. Indeed in Section 5.1, we shall see another algorithm (Shewchuk's version of Delaunay refinement) that solves conforming Delaunay triangulation as a preliminary stage to interior point placement.

Edelsbrunner and Tan's algorithm places Steiner points in two stages, the "blocking" and "propagation" phases. The blocking phase computes a set of $O(n)$ non-overlapping disks whose union is connected and touches all the vertices of G. A Steiner point is added wherever a disk crosses an edge or meets another disk, $O(n^2)$ Steiner points total. The propagation phase treats the segments of input edges that are not yet covered by empty circles. This phase takes any not-yet-covered segment ab and attempts to add its diameter circle. If the diameter circle encloses another vertex c visible to ab, the orthogonal projection of c onto ab is added as a new vertex, forming two shorter segments. (We say that c is *visible* to ab if c can be connected to some point on ab by a line segment that does not cross any other vertices or edges of G.) Because the blocking phase has cut to all the vertices of G, new points cannot propagate in a spiral (which would necessarily enclose a vertex) to hit the same edge twice.

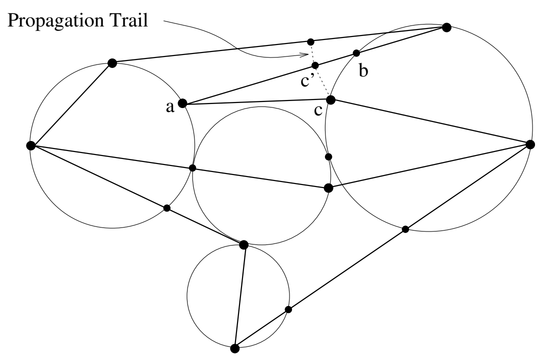

Fig. 11. The Edelsbrunner-Tan algorithm for conforming Delaunay triangulation first computes a set of disks spanning the vertices, and then propagates vertices encroaching on input edges out to the boundary of the planar straight-line graph.

Constrained Delaunay triangulation offers a more intrinsic solution to meshing a polygonal domain than does conforming Delaunay triangulation. The edges of the input domain are forced into a constrained Delaunay triangulation; all other edges arise naturally.

The *constrained Delaunay triangulation* $CDT(G)$ contains all vertices and edges of G. In addition, $CDT(G)$ contains the edge $\{a, b\}$ between two vertices of G, if and only if a is visible to b, and some circle passes through a and b, but no other vertices of G, and encloses no vertex c visible to segment ab. If G consists only of isolated vertices, this definition reduces to the Delaunay triangulation. Figure 12 depicts the constrained Delaunay triangulations of a planar straight-line graph and a simple polygon; the circles of two edges are shown. The polygon is a special case of a planar straight-line graph, but here we triangulate only the interior of the polygon, rather than all the way out to the convex hull.

The flipping algorithm for ordinary Delaunay triangulation can be adapted to compute the constrained Delaunay triangulation. As in Section 3.3, an initial triangulation can be computed by a sweep-line algorithm, and then edge flips can be performed in arbitrary order until no "flippable" quadrilateral is reversed. In the constrained case, a quadrilateral whose diagonal is one of the input edges is not flippable. The sweep-line algorithm is slightly more complicated in the constrained case. Whereas previously the algorithm kept a sorted list of all points on the current convex hull to the left of the sweep-line, the algorithm must now maintain such a list for each interval along the sweep-line between two input line segments. A sorted list or dictionary data structure can be used to determine which interval contains a newly encountered vertex.

4.2 Three Dimensions

A natural generalization of a planar-straight line graph to three dimensions is called a *piecewise-linear complex* [104]. A piecewise-linear complex X consists of a set of vertices, line segments, and facets. The facets can be arbitrary planar polygons, possibly with holes. Line segments can only intersect at shared endpoints (which must be vertices of X), and facets can only intersect at shared boundary segments (which must be

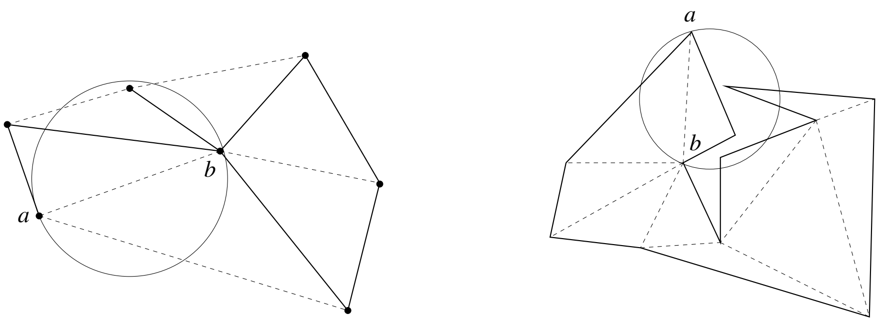

Fig. 12. Constrained Delaunay triangulations of a planar straight-line graph and a polygon.

segments of X). A line segment can intersect a facet only if it is a boundary segment of the facet, or if one of its endpoints is a vertex of the facet. If a line segment intersects a facet at a point in the segment's relative interior, then the entire segment must lie in the facet.

Given a piecewise-linear complex X, the conforming Delaunay triangulation problem in three dimensions asks for a set of Steiner points such that the Delaunay triangulation of the vertices, input plus Steiner points, contains all the segments and facets of X (possibly subdivided). Some mesh generation programs [50, 115, 116] take an approach that computes the Delaunay triangulation of the vertices, finds missing input edges and facets, and then adds Steiner points where input segments cross Delaunay balls or faces and where input facets cross Delaunay edges. This algorithm has not been analyzed. Murphy et al. [85] generalize a planar algorithm due to Saalfeld [101]; this algorithm covers the vertices, segments, and facets of X with small tangent balls, and then places Steiner points where the Voronoi diagram of the ball centers crosses X. This algorithm is at least guaranteed to terminate, but it has no simple-to-state bound on the number of Steiner points, not even a bound that depends on the geometry as well as on the combinatorial complexity of X.

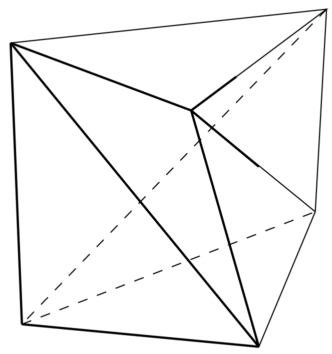

Fig. 13. Schönhardt's polyhedron is a triangular prism, twisted so that its three rectangular sides bend inwards and form a spiral of reflex edges. This polyhedron cannot be tetrahedralized without Steiner points.

There is no exact analogue of constrained Delaunay triangulation in three dimensions. One fundamental obstacle is that for some inputs triangulation is impossible without Steiner points. Figure 13 shows a classic example due to Schönhardt [103] of a polyhedron that cannot be triangulated without Steiner points. To form this polyhedron, start with a triangular prism and then twist the top relative to the bottom so that all three rectangular faces bend inwards. For this to work, the diagonals of the rectangular faces must be pairwise disjoint, forming a sort of a spiral. Now none of the three bottom vertices can see all of the top triangle, so we have no fourth vertex available for a tetrahedron containing this triangle.

Shewchuk [104] devised the closest thing to a 3D constrained Delaunay triangulation. He calls a segment of X *strongly Delaunay* if it has a circumsphere that neither encloses nor passes through any other vertex of X. He calls a simplex (line, triangle, tetrahedron) *constrained Delaunay* if it has a circumsphere that encloses no vertex of X visible to any point in the relative interior of the simplex. He shows that if each of X's segments is strongly Delaunay, then X has a triangulation in which each simplex is constrained Delaunay. In the non-degenerate case of no five cospherical vertices, the constrained Delaunay triangulation of X is unique.

Schönhardt's polyhedron provides a good example. This polyhedron can obviously be inscribed in a sphere. The edges of the top and bottom triangles are strongly Delaunay right from the start. By pushing two of the top vertices into the sphere we can also make any two of the three reflex edges strongly Delaunay, but it is not possible to make all three strongly Delaunay. We can subdivide the third reflex edge into two strongly Delaunay segments, but now the polyhedron becomes tetrahedralizable, as indeed it must for Shewchuk's result to hold. At this point we can apply Shewchuk's result, without adding any further Steiner points. This is the practical advantage of constrained over conforming Delaunay triangulation: fewer Steiner points are needed and in particular no Steiner points interior to facets. In Section 5 we shall see some 3D constrained Delaunay meshes. An interesting theoretical question [48, 104] is whether there is a polynomial bound (or even a simple-to-state bound depending upon geometry) on the number of Steiner points needed to make all segments of X strongly Delaunay.

Shewchuk proved his result by showing that each point in the convex hull of X lies in some constrained Delaunay tetrahedron, and that together the constrained Delaunay tetrahedra fill the convex hull without overlapping. For the first part, Shewchuk showed that it is possible to grow a circumsphere of any constrained Delaunay simplex σ of X (edge of X or constrained Delaunay triangle on a facet of X) into a circumsphere of a constrained Delaunay tetrahedron. The key to this proof is an acyclicity result: facets of X bounded by strongly Delaunay segments cannot overlap in a cycle to block completely σ's visibility of vertices (Figure 14).

In order to compute 3D constrained Delaunay triangulations, Shewchuk [104, 105] recommends the following steps: subdivide segments until they are all strongly Delaunay, compute the unconstrained Delaunay triangulation of the vertices, and then recover each facet in turn by deleting all the tetrahedra it crosses and retriangulating the holes on either side. More recently, Shewchuk [107] gave sweep algorithms with asymptotically faster running times for computing constrained Delaunay triangulations. Shewchuk's 3D constrained Delaunay triangulation also generalizes to $\mathbb{R}^d$ for $d > 3$ [104]. Here the requirement is that each *ridge*, that is, $(d-2)$-dimensional face, of the piecewise-linear complex be strongly Delaunay.

5 Point Placement

In this section, we discuss methods for placing points in order to achieve a high-quality initial mesh. There is a close connection between the topic of this section (point placement) and that of the next section (mesh refinement). In this section, we cover techniques driven by domain geometry and *a priori* knowledge rather than by *a posteriori* error es-

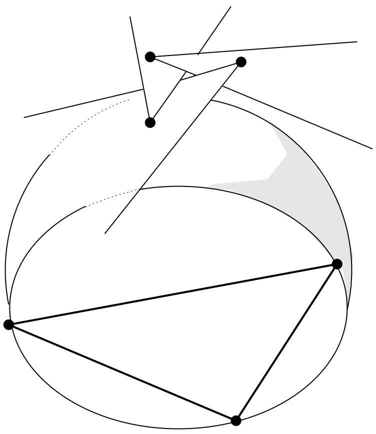

Fig. 14. Shewchuk's argument rules out the case that a cycle of facets blocks a growing circumsphere from finding a visible vertex.

timates, but much of the discussion, especially the material on mesh improvement (Sections 5.4, 5.5, and 5.6), applies to remeshing as well.

5.1 Delaunay Refinement

Delaunay refinement starts from a conforming or constrained Delaunay triangulation and then iteratively adds new Steiner points in order to improve the sizes and shapes of elements.

We first explain an algorithm for planar domains due to Ruppert [100], who extended work of Chew [36]. 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* (a domain with internal boundaries such as general planar straight-line graph), 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. Shewchuk [106] recently gave a better algorithm for handling domains with sharp angles.

Next the algorithm computes a Delaunay triangulation of the modified domain; this Delaunay triangulation may be either constrained or conforming. (Figure 15, however, was actually computed started from the unconstrained Delaunay triangulation of input vertices, as in Shewchuk's 3D algorithm given below.) Ruppert's 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, but only to repair the triangulation around c as in randomized incremental insertion. The removed old triangles are exactly those with circumcircles containing c, which can be found by searching outwards from the

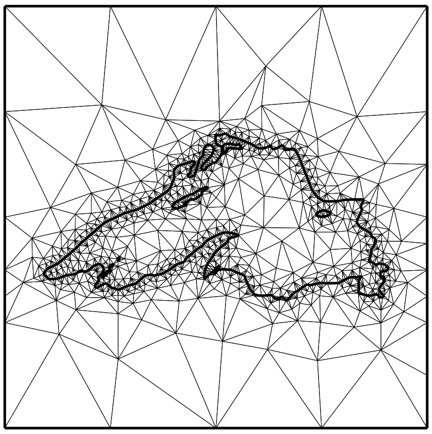

Fig. 15. A mesh computed by Ruppert's algorithm. (J. Ruppert)

triangle that contains c, and the new triangles that replace the removed triangles are all 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 or on the diameter semicircle of a boundary edge e **then**
    add the midpoint m of e
  else add c **endif**
  recompute the Delaunay (or constrained Delaunay) triangulation
endwhile

The loop is guaranteed to halt with all angles larger than $20°$. After the loop finishes, the cut-off isosceles triangles are returned to the domain, and the mesh is complete.

Ruppert's algorithm comes with a strong theoretical guarantee: for non-multiple domains it produces a triangulation in which no new angle (angle not already present in the domain boundary) measures less than $20°$ and in which the number of triangles is no more than a constant times the minimum number in any such *no-small-angle triangulation* of the same domain. Shewchuk showed that a similar theorem holds for multiple domains [106], only now the sharpest new angle may be as small as about one-third of the minimum input angle. The number of triangles needed is a function of geometry and not just of the number of vertices, as can be seen by considering the simple example of a long skinny rectangular domain. Such a guarantee was first proved by Bern et al. [24] for a quadtree mesh generator. In fact, it was the opportunity to prove such guarantees that attracted the attention of computational geometers to mesh generation.

To prove the bound on the number of triangles, 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. In later sections, we shall see externally defined spacing functions.

Shewchuk [105] extended Delaunay refinement to three dimensions using his definition of 3D constrained Delaunay triangulation. The input domain is assumed to be a "facet-bounded" piecewise-linear complex, meaning that it has a watertight outer boundary; the input is also assumed to have no overly sharp diheral angles. The first step computes the unconstrained Delaunay triangulation of the vertices. Then the algorithm goes into a loop, as in Ruppert's 2D algorithm, this time adding new vertices according to three different rules. (1) If the diametral sphere of an input segment encloses or passes through a vertex not on the segment, the segment is subdivided by adding its midpoint, and midpoint subdivision continues until subsegment diametral spheres are empty, (2) If the equatorial sphere of a triangular boundary face encloses or passes through a vertex not on the face, the face is split by adding its circumcenter. Rule (1), however, takes precedence over rule (2), so that if the circumcenter would encroach upon an input segment (that is, land in its diametral sphere) then we split that segment first. (3) If a tetrahedron is *badly shaped* we add the center of its circumsphere. Rules (1) and (2) take precedence over rule (3), so we may have to split input segments and faces before adding the tetrahedron's circumcenter.

The meaning of the term "badly shaped" deserves more attention. Following [44] we can classify all the badly shaped tetrahedra into five types of skinny tetrahedra (Figure 16) and four types of flat tetrahedra (Figure 17). A previous, somewhat simpler, classification [20], divided tetrahedra by no-small- and no-large-angle criteria for each of solid and dihedral angles. In either classification, a *sliver* is a flat tetrahedron whose projection onto a plane that passes near all its vertices is fairly square. This bad tetrahedron is special: it is the only type of bad tetrahedron that has a small ratio of circumsphere radius (circumradius) to shortest edge.

Shewchuk proved that his mesh generator can eliminate all types of bad tetrahedra except slivers. When the circumcenter of a sufficiently bad tetrahedron of any other type is added, the worst circumradius to shortest-edge ratio in the entire mesh necessarily improves because the new vertex reduces the circumradius without creating any new short edges. (This observation was also used in earlier 3D Delaunay refinement algorithms by Miller et al. [83] and Talmor [112].) One can attempt to remove slivers using Delau-

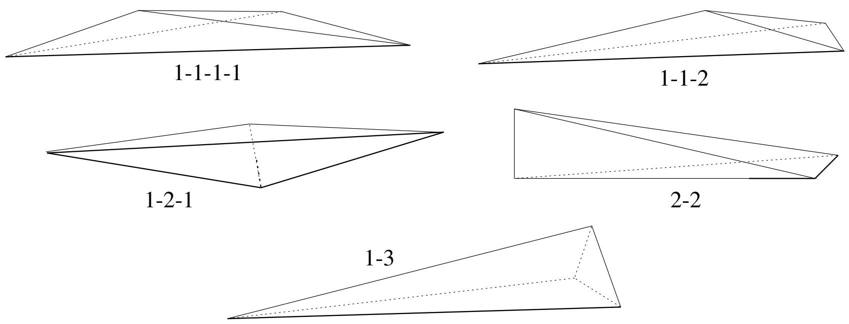

Fig. 16. Five fuzzy types of skinny tetrahedra. Vertices are close to colinear; in a 1-1-1-1 tetrahedron the vertices are all well-separated along the line, in a 2-2 tetrahedron the vertices fall into two pairs, and so forth.

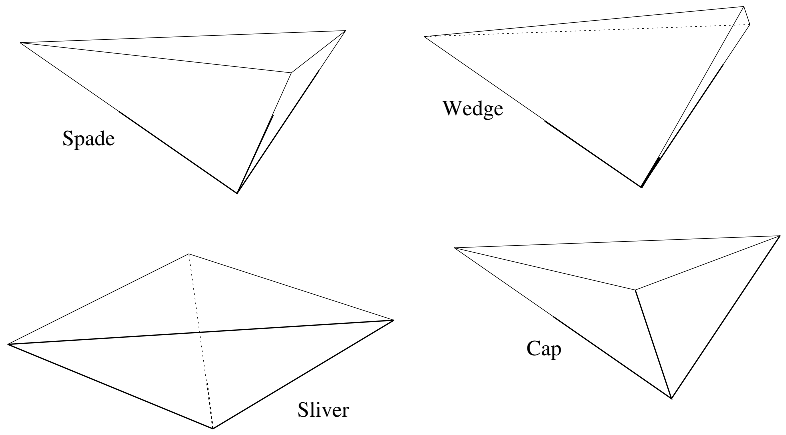

Fig. 17. Four fuzzy types of flat tetrahedra, classified according to the projection of the vertices onto the plane. The sliver is the unique type of badly shaped tetrahedron with a good circumradius to shortest edge ratio.

nay refinement—Shewchuk gives some encouraging empirical results—but there is no guarantee that the algorithm will halt.

Recently, Edelsbrunner and Guoy [45] proposed a more selective form of Delaunay refinement called sink insertion. This method applies to both 2D and 3D Delaunay triangulation; for continuity we explain it in 3D. Call a Delaunay tetrahedron that contains its own circumcenter *self-centered* (see for example [20]) and the circumcenter of such a tetrahedron a *sink*. The name refers to the fact that self-centered tetrahedra are sinks (vertices without out-edges) in the directed graph in which each tetrahedron points to the neighboring tetrahedron in the direction of its circumcenter. Moreover, sinks locally maximize the distance to the closest vertex, and also locally maximize the error in interpolating a spherical quadratic function by linear basis functions over the tetrahedral elements. Sink insertion follows a path in the directed graph from a badly shaped tetrahedron to a self-centered tetrahedron and then adds the associated sink. Sink insertion gives the same theoretical guarantee as Delaunay refinement (no bad tetrahedra except slivers); experimentally it appears to be much faster and more parallelizable, yet equally successful at removing bad tetrahedra [45, 57]. See Figure 25 below.

5.2 Disks, Balls, and Bubbles

You may have already noticed that computational geometers are crazy about circles and spheres. In this section, we really indulge ourselves and look at point placement methods based on these lovely objects.

We start with hard-edged disks in the plane. *Disk packings* in which each gap between disks has exactly three sides are, in a very real and rigorous way, discrete analogues of conformal mappings (see for example [111]). A conformal mapping is the ultimate in "provably good" meshing: a conformal mapping of a fine square grid onto a planar domain gives all interior angles tending to 90° as elements shrink. Conformal

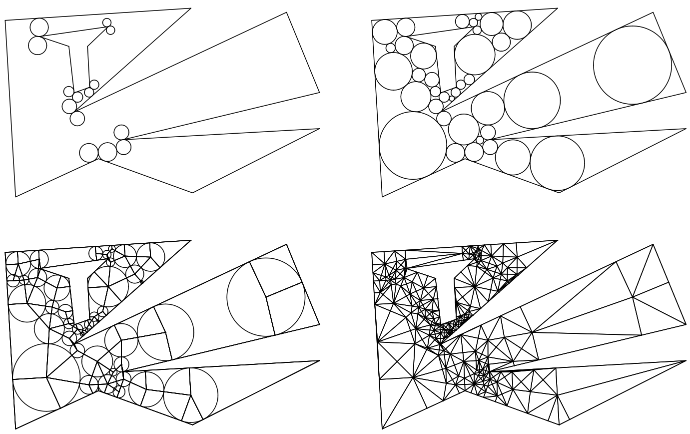

Fig. 18. An algorithm for nonobtuse triangulation of a polygonal domain: (1) place disks at reflex vertices and holes, (2) pack disks so that each gap has three or four sides, (3) connect centers to break the domain into small polygons, and (4) triangulate small polygons using circumcenters of gaps.

mappings and three-sided-gap disk packings, however, are too global to be of much use in adaptive mesh generation; there are only three degree of freedom—choice of conformal center and orientation—in a Riemann mapping from a planar domain without holes to a disk. Nevertheless, it makes an interesting theoretical problem to optimize some geometric criterion over these three degrees of freedom [23].

If we relax the requirements so that gaps can have either three or four sides, we gain a lot of flexibility while retaining some control over the angles in the mesh. Figure 18 illustrates an algorithm [26] for nonobtuse triangulation of a polygon. This algorithm is not intended to give usable finite element meshes, but rather it proves an interesting theorem: any n-vertex polygon with holes can be triangulated with only $O(n)$ nonobtuse triangles. This theorem shows that *no-large-angle triangulation* is quite different from no-small-angle triangulation, which as mentioned above requires a number of triangles dependent upon the geometry of the domain rather than just n.

Disk-packing with three- and four-sided gaps can also be used for quadrilateral meshing. Bern and Eppstein [25] gave $O(n)$-size quad meshes for four different theoretical meshing problems: all elements are Voronoi cells (Figure 19), all elements are kites, all elements have two opposite right angles (implying that they have cocircular vertices), and finally all elements have maximum angle at most $120°$. These results—or at least the algorithms used to prove them—could be relevant to automatic multiblock meshing, but to our knowledge no-one has ever pursued this possibility.

A still more relaxed sort of disk packing—and in three dimensions ball packing—has been used by Miller, Talmor, Teng, and their collaborators [69, 83, 112]. Say we have a *spacing function* $f(p)$ defined on points p of the domain Ω. Let $B(p, r)$ denote the ball with center p and radius r. We call a set of balls $\mathcal{P}$ a *β-packing* for some $\beta > 0$ if (1) no two balls of $\mathcal{P}$ overlap, (2) the radius of a ball of $\mathcal{P}$ with center p is $f(p)$, and (3) for each

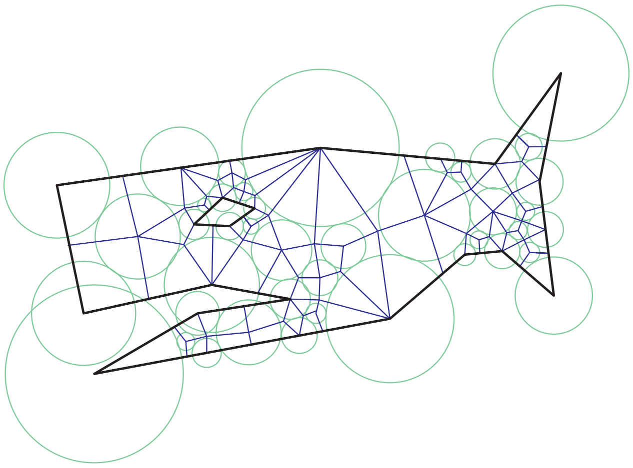

Fig. 19. A quadrilateral mesh, produced by disk-packing, in which each element is a Voronoi cell.

point $q \in \Omega$, $B(q, \beta f(p))$ intersects a ball of $\mathcal{P}$. Thus the sizes of balls in a β-packing grade according to the spacing function.

The algorithm for creating a β-packing on Ω places balls centered on the vertices of Ω first, then on the segments, then on the facets, and finally in the interior. A simple way of ensuring non-overlap is to generate balls on facets and interior randomly, and then discard them if they overlap previously placed balls. A grid or quadtree can be used to keep track of uncovered regions in order to accelerate this process. Talmor [112] shows that the Delaunay triangulation of the ball centers of a β-packing generated in this way conforms to the domain. In addition, in $\mathbb{R}^2$ the Delaunay triangulation has no new small angles and optimal size up to constant factors just like Ruppert's meshes. In $\mathbb{R}^3$ the size remains optimal, but the quality guarantee is somewhat weaker: the mesh contains no bad tetrahedra except slivers. Figure 26 below shows some 2D meshes in which the vertices were placed by β-packing.

Finally *bubble systems*, introduced by Shimada [108, 109], use methods from mesh smoothing (Section 5.4) inside the point placement process. Balls are now allowed to overlap each other a bit; neighboring balls exert forces on each other—typically repulsive at short range and attractive at long range—in order to space themselves appropriately. Sometimes "birth" and "death" processes are used to create vertices in overly sparse regions and destroy vertices in overly crowded regions, and damping forces are used to ensure convergence.

We give a few more details on one particular bubble algorithm due to Bossen and Heckbert [29]. Bossen and Heckbert use a force whose dependence upon distance d is $(1-d^4)\cdot\exp(-d^4)$, and for efficiency they define neighboring balls to be those that are connected by the current Delaunay triangulation of ball centers. They delete balls that are too crowded and subdivide edges that are too long, with both measurements taken relative to an externally defined spacing function. The Delaunay triangulation is repaired at each iteration.

5.3 Anisotropy Again

As mentioned in Section 2, anisotropy is often a desirable feature in meshes for fluid dynamics. Boundary layers in the flow demand anisotropic elements surrounding domain boundaries, for example, long skinny elements over an airfoil. A popular approach to point placement for resolving boundary layers is the *advancing front* method [78, 89]. With a few recent exceptions [70], computational geometers have not worked on advancing front, so we will not say much about this approach.

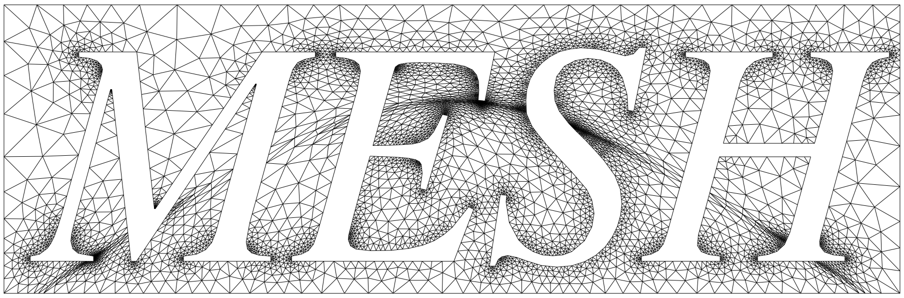

Fig. 20. A mesh computed by Bossen and Heckbert's bubble-based anisotropic mesh generator. (F. Bossen and P. Heckbert)

A more generic approach to anisotropy, which we shall call *stretching*, was first introduced by Mavriplis [80], Castro-Díaz [31], and others. Stretching replaces the usual scalar spacing function with a matrix-valued function that controls the desired local element size and orientation. We can picture this function as an "ellipse field", giving the linearly transformed unit ball at each position. Edge lengths are now measured relative to the local unit ball. Delaunay triangulation is replaced by a modified Delaunay triangulation in which the linear transformation associated with one of its points is applied to a quadrilateral before testing whether it is reversed or not. We must be a little bit careful about exactly which linear transformation to apply, so that we do not fall into an infinite flipping loop (personal communication with S. Teng). Bossen and Heckbert included stretching in their two-dimensional bubble mesher, obtaining esthetically pleasing results (Figure 20). Yamakawa and Shimada [120] incorporated stretching into a three-dimensional bubble mesher.

One possible drawback of stretching is that it can produce flat angles as well as the desired sharp angles. Recall that some numerical methods, such as finite difference and finite volume formulations, prefer anisotropic meshes in which skinny elements have their shortest edges in the direction of high second derivative. It should be possible to modify bubble methods to produce such meshes by building anisotropy into the simulated physical forces, rather than just into the length measurements. Shimada (personal communication) recommends rectangular or stretched square bubbles, which would tend to form stacks normal to the surface. It may also be possible to use stacks of stiffly connected spherical bubbles in order to transition more gracefully to isotropic regions. See Figure 21 for illustrations of these two ideas.

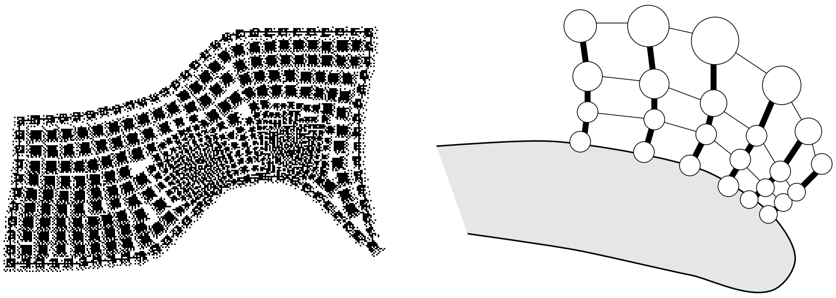

Fig. 21. Either (a) square packings (IBM - Japan) or (b) bubble packings with a mix of stiff and loose connections may be able to produce anisotropy without flat angles.

5.4 Mesh Smoothing

This section and the next two discuss mesh improvement methods, ways to improve element shape and size that can be applied as postprocessing steps or—as we saw in bubble meshing—at the same time as point placement. Mesh smoothing adjusts the locations of mesh vertices in order to improve element shapes and overall mesh quality [3, 5, 30, 49, 88]. In mesh smoothing, the topology of the mesh remains invariant, thus preserving important features such as the nonzero pattern of the linear system.

In bubble meshing, the smoothing procedure uses a spacing function or the radii of the bubbles to determine ideal distances between vertices, but more often smoothing proceeds without such outside information. Laplacian smoothing [49, 75] is the most commonly used technique. This method sweeps over the entire mesh several times, repeatedly moving each vertex, not constrained to lie on a boundary feature, to the centroid (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 [118]. Laplacian smoothing is computationally inexpensive and fairly effective, but it does not guarantee improvement in element quality, and it can even invert an element, unless the algorithm performs an explicit check before moving a vertex.

Another class of smoothing algorithms uses global or local optimization techniques to determine new vertex locations. Global techniques simultaneously adjust all unconstrained vertices; such an approach involves an optimization problem as large as the number of unconstrained vertices, and consequently is computationally expensive [30, 88]. Local techniques adjust vertices one by one—or an independent set of vertices in parallel [52]—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 [3]. Figure 22 shows the results of a local optimization-based smoothing algorithm developed by Freitag et al. [52]; smoothing was applied 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 $41.1°$.

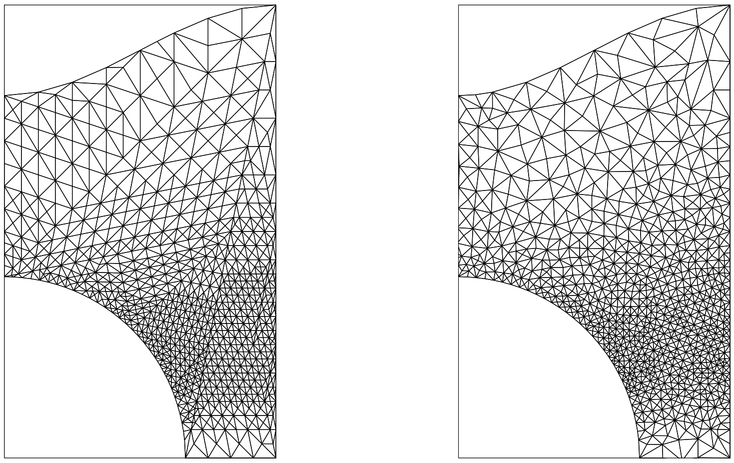

Fig. 22. (a) A mesh resulting from bisection refinement without smoothing. (b) The same mesh after local optimization-based smoothing. (L. Freitag)

5.5 Flipping Revisited

The flip moves used in the algorithms for computing Delaunay triangulations also make useful mesh improvement steps. As before, 2D flipping exchanges the diagonals of a triangulated quadrilateral as in Figure 3, and 3D flipping exchanges two tetrahedra for three as in Figure 9, only now the criterion for making the exchange need not be the Delaunay empty sphere test. Flipping can instead be used to regularize vertex degrees, minimize the maximum angle, or improve almost any other quality measure of triangles or tetrahedra. For quality measures optimized by the Delaunay triangulation, flipping computes a true global optimum, but for other criteria it computes only a local optimum.

There is a nice way to view flipping that unifies it with the splitting move used in the randomized incremental insertion algorithm. Using the lifting map, we can view the two different ways to triangulate a convex quadrilateral as projections of the lower and upper convex hulls of the lifted points, whose entire convex hull is simply a tetrahedron. Similarly, the split move exchanges the lower and upper convex hulls of a tetrahedron, only this time the lower hull consists of a single triangle and the upper hull consists of three triangles (or vice versa), because back on the plane one point lies inside the triangle formed by the other three. Thus the split is sometimes called a 1-3 flip.

The 3D flip moves (Figure 9) are now seen to be the exchange of lower and upper convex hulls of a four-dimensional simplex. The split move, which subdivides a single tetrahedron into four by adding an interior point, is the only other such exchange; it is a 1-4 flip.

Although it does not seem to be well-known to the meshing community, there are canonical sets of flip moves for quad and hex meshes as well. The flip moves for quad meshes exchange the lower and upper hulls of a three-dimensional cubical polytope, as shown in Figure 23. There are 1-5 and 2-4 flips and two different kinds of 3-3 flips. We

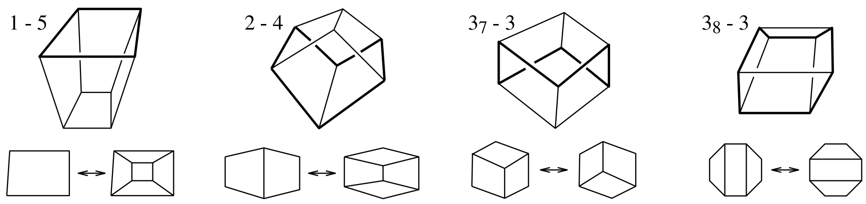

Fig. 23. Flips for quad meshes.

have denoted the two kinds of 3-3 flips as 3_7-3 and 3_8-3, depending upon the number of vertices appearing in the "left hand side" of the flip. Notice that the flip moves do not change the parity (evenness or oddness) of the number of quads in a mesh; we have recently proved [22] that for a given polygonal domain, any quad mesh with the same parity is obtainable from any other by a finite sequence of flips. There is a simple parity-switching move as well: exchange a hexagon split into two quads for a hexagon split into three quads all incident to an interior vertex.

The analogous flips for hex meshes exchange the upper and lower hulls of a 4D cubical polytope; they are shown in Figure 24. Some of these moves have already been discovered and applied in an *ad hoc* way [84]. We believe that the full set may be useful in refining and improving hex meshes. (We also wonder if there is any sort of construction that could be deemed a cubical analogue to Delaunay triangulation.) We have shown that a quad surface mesh can support both even- and odd-parity hex volume meshes, but we do not know if there is a simple parity reversing operation for hex meshes.

5.6 Sliver Removal

Our final topic on mesh improvement is the one that has received the most recent attention from the computational geometry community. Recall that the sliver from Figure 17 is the unique type of bad tetrahedron that appears in a Delaunay triangulation of well-spaced points (say centers of a β-packing for a uniform spacing function). Fortunately slivers turn out to be rather fragile, and there are now several different "provably good" sliver removers.

Chew [37] showed that a randomized form of Delaunay refinement will, with high probability, remove all slivers flatter than a certain constant. The provable constant is very weak, but experiments are encouraging. Chew's sliver removal algorithm inserts a randomly chosen point within a ball around a bad tetrahedron's circumcenter, rather than the exact circumcenter. As in other forms of Delaunay refinement, if this randomly chosen point falls within the diameter sphere of a boundary edge or equatorial sphere of a boundary facet, then the boundary face is subdivided instead.

Edelsbrunner et al. [35, 44] gave a provably good sliver removal algorithm based on weighted Delaunay triangulation. This intriguing algorithm, called *sliver exudation*, removes interior slivers without even adding new Steiner points. We can think of sliver exudation as pushing slivers ("exuding" them) to the boundary of the domain, where they can be fixed by subdividing the boundary. Sliver exudation treats the Delaunay triangulation as a weighted Delaunay triangulation with uniform weights and iteratively adjusts

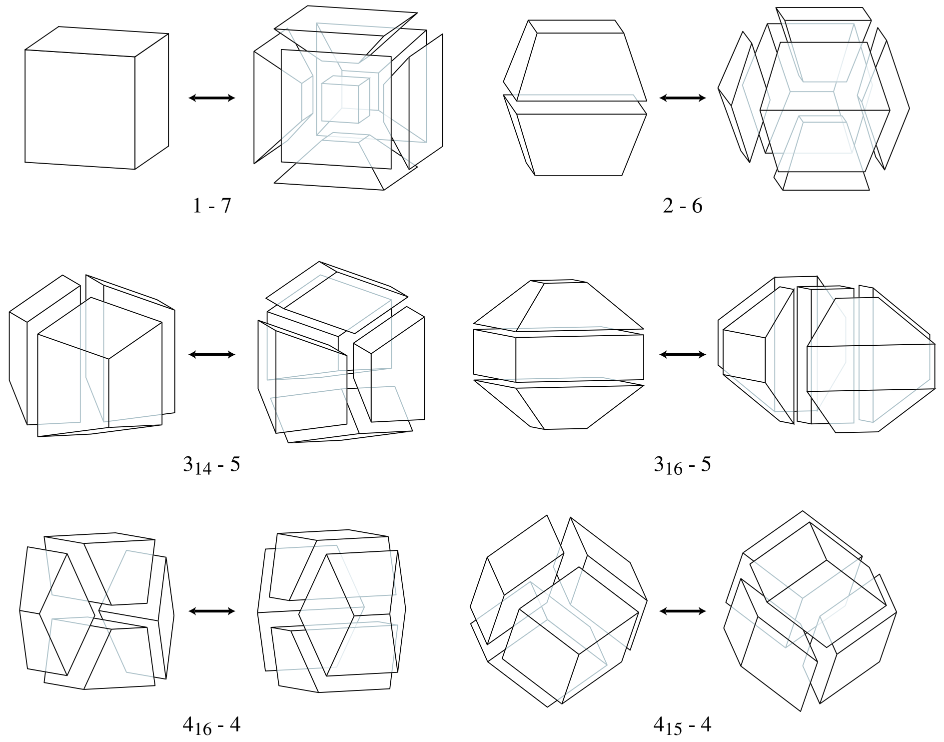

Fig. 24. Flips for hex meshes.

vertex weights in order to remove slivers. The key to the success of this algorithm is that a mild change in the vertex weights of a sliver will radically inflate the orthogonal sphere of the sliver. Thus as weights change, the orthogonal spheres of slivers will penetrate too deeply into vertex spheres and slivers will drop out of the tetrahedralization before the vertices themselves drop out. We can also think of sliver exudation as displacing lifted points in the vertical direction; a mild displacement will remove a sliver facet from the lower convex hull. As in Chew's algorithm, the provable constants are very weak, but experiments [46, 57] look fairly persuasive. See Figure 25.

Edelsbrunner et al. [47] also proved that perturbing a well-spaced point set in 3D can remove slivers. As in Chew's algorithm [37] the key observation is that the locus of sliver-forming positions for the fourth vertex of a tetrahedron with three fixed vertices has relatively small volume. Most recently, Li and Teng [71] and Cheng and Dey [34] combined Delaunay refinement (as in Chew's algorithm) with perturbation (as in the two Edelsbrunner et al. algorithms) to incorporate sliver removal into the main meshing loop.

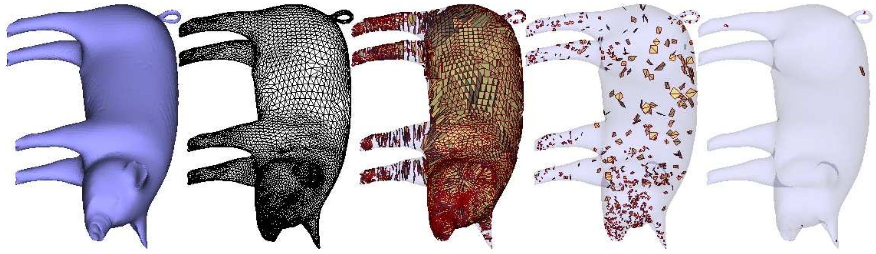

Fig. 25. Sliver exudation modifies the vertex weights in a weighted Delaunay triangulation in order to remove slivers. These figures show (a) the input boundary mesh, (b) a wire-frame of the boundary, (c) initial badly-shaped tetrahedra in a surface-reconstructing tetrahedralization (produced by *Wrap* from Rainbow Geomagic, Inc.), (d) remaining slivers (tetrahedra with a dihedral less than 5°) after sink insertion, and (e) remaining surface slivers after exudation. (D. Guoy)

6 Solution-Adaptive Mesh Refinement

Solution-adaptive mesh refinement reduces the sizes of elements in areas where the PDE solution error is large. Local error estimates based on an initial solution are known as *a posteriori* error estimates [10] 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 [117].

One way to achieve solution-adaptive refinement is rather anticlimactic: use a spacing function (such as $f(p)$ in Section 5.2 above) to control mesh density, change the spacing function based on the solution, and then completely remesh the domain based on the new spacing function. The drawback of this approach is inefficiency: it is faster to remesh only locally and retain as much of the old mesh as possible. In multigrid methods, local remeshing may enable complete reuse of the coarser levels of the initial mesh.

6.1 Iterative Vertex Insertion

One approach to solution-adaptive mesh refinement inserts extra vertices into the triangulation and then locally repairs the mesh. Reasonable choices of locations to insert new vertices include at edge bisectors as in Bossen and Heckbert's mesher (Section 5.2) and at triangle circumcenters as in Delaunay refinement (Section 5.1). New vertices along the boundaries of curved domains should be computed using the curved boundary rather than the current straight edge, thereby giving a truer approximation of the domain as the mesh refines [31].

Li, Teng, and Üngör [69] show how to adaptively refine and coarsen β-packings, described in Section 5.2 above. (We shall say more about coarsening in Section 7 below.) They change the spacing function $f(p)$ based on the initial solution and then shrink or grow the radii of balls in regions where the spacing function has changed. They then fill gaps that have opened up with new non-overlapping balls in order to restore the β-packing property, and then recompute the Delaunay triangulation. Figure 26 shows two snapshots from a sequence of meshes for a growing circular "shock front".

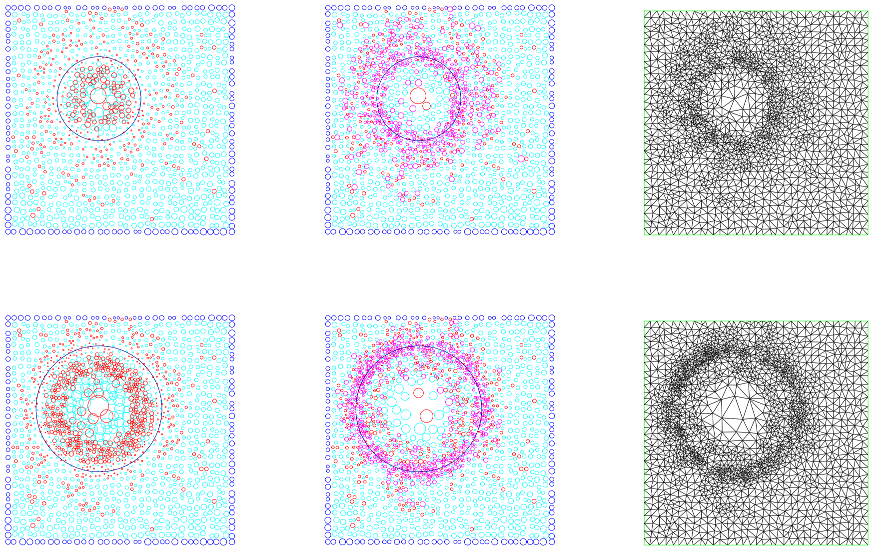

Fig. 26. Ball packings can be used in solution-adaptive remeshing. The top row shows (a) an initial overlapping set of balls, (b) a maximal set of non-overlapping balls (β-packing), and (c) the Delaunay triangulation of ball centers. The bottom row shows a remeshing derived from the initial packing by adjusting radii and filling gaps. (A. Üngör)

If balls only shrink, meaning refinement but no coarsening, this method will produce a *node conforming* refinement: each vertex of the initial mesh will appear in the refined mesh. It does not produce an *edge conforming* refinement, one in which each edge of the initial mesh appears (possibly subdivided) in the refined mesh.

6.2 Nested Refinement

In this section, we describe techniques used to produce edge conforming or *nested* refinements. These techniques subdivide elements in the current mesh rather than remeshing, even locally. Nested refinements simplify the interpolation step in the multigrid method, because each element from the refined mesh lies inside exactly one element from the initial mesh. Nested refinements are also close relatives of subdivision meshes which will be described in Section 8. To compute nested refinements, we *split* 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 1:4 refinement* [13, 14], the midpoints of the sides of a marked triangle are connected, as in Figure 27(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 27(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.

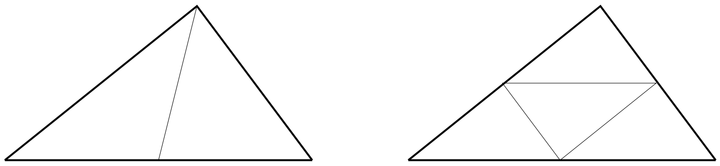

Fig. 27. A triangle divided by (a) bisection, and (b) regular 1:4 refinement.

Rivara [94–96] 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 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 [98] 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 28. This method does not always bisect the longest edge, so bisections tend to propagate less, yet 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**

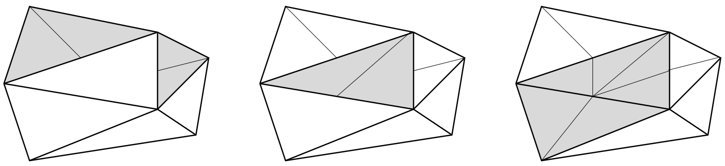

Fig. 28. The bisection algorithm given in the pseudocode splits invalid children of refined triangles to their subdivision points, rather than to their longest edges.

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 move up to three dimensions and discuss two different generalizations of bisection. 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 29. 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.

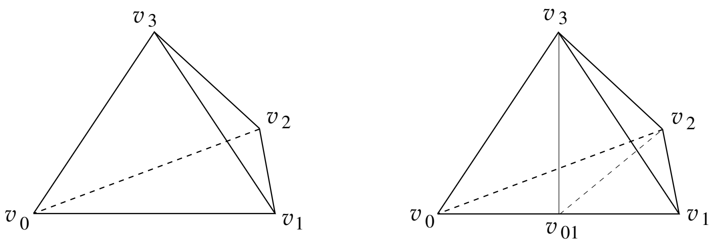

Fig. 29. The tetrahedron on the left is bisected to form two new tetrahedra

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 [97] suggested an extension of longest-edge Rivara refinement 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.

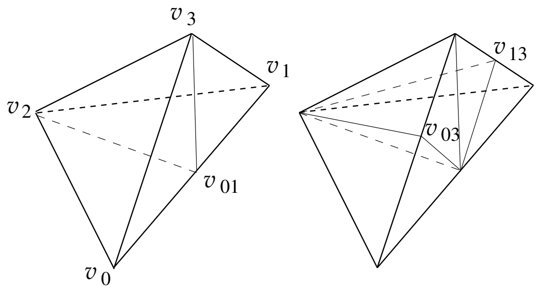

Fig. 30. The first three levels of longest-edge bisection of the canonical tetrahedron. Note that the tetrahedra generated at each level are similar. For the final level of refinement we show only the four tetrahedra obtained from $v_0v_{01}v_2v_3$. Four similar tetrahedra are obtained from $v_{01}v_1v_2v_3$.

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.

A bisection algorithm first introduced by Bänsch [16] does indeed generate only a finite number of similarity classes. Before describing the algorithm, we sketch the argument of Liu and Joe [73] 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 vertices $v_0 = (-1, 0, 0)$, $v_1 = (1, 0, 0)$, $v_2 = (0, 1/\sqrt{2}, 0)$, and $v_3 = (0, 0, 1)$. In Figure 30 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: due to the affine transformation, similarity in the original space requires homothety (identical up to scaling and translation) not just similarity in the canonical space. Hence the number of similarity classes turns out to be 36, corresponding to 3 similarity classes in the canonical space times 12 non-inverting affine transformations [6, 79].

Bänsch [16], Liu and Joe [74], and Arnold et al. [6] give essentially equivalent [64] 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 an adjacent pair 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. and the other refinement edges to either v_0v_3. In the pseudocode below, a *child face* is a triangle like $v_{01}v_1v_3$ in Figure 29, and a *new face* is one like $v_{01}v_2v_3$. The key point is that the refinement edge of a child face is its *old edge*, that is, the edge like v_1v_3 that was in the mesh before bisection; thus the next bisection that splits child face $v_{01}v_1v_3$ will agree with the corresponding split on the canonical tetrahedron.

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

7 Mesh Simplification

We now discuss the reverse of mesh refinement: simplifying or coarsening a mesh. Local *a posteriori* simplification helps reduce the total number of elements when tracking solutions to time-varying differential equations. Global *a priori* simplification can be used to turn a single highly refined mesh into a sequence of meshes with varying levels of detail for multigrid methods or other uses.

In Figure 26 we already saw an example of a straightforward algorithm for *a posteriori* coarsening. This algorithm marks a set of vertices in areas that are more refined than necessary, eliminates all the marked vertices, and then retriangulates to give a node conforming coarse mesh. One potential difficulty is that the shapes of the triangles may degrade as the mesh is coarsened due to increasing disparity between the interior and boundary point densities. Hence the boundary may need to be coarsened at the same time as the interior.

Another method of *a posteriori* simplification, very popular for the surface meshes used in computer graphics [62], is *edge contraction*. An edge contraction coalesces a pair of adjacent vertices, producing a new vertex adjacent to all the neighbors of the pair. The new vertex is usually placed at the midpoint of the contracted edge, so that the resulting mesh is neither node nor edge conforming. (Notice that if the new vertex is coincident with one of the old vertices, then an edge contraction is the same as a vertex removal followed by a particular retriangulation.) It is possible to determine locally whether the edge contraction will change the topological type of the mesh [44]; if it would then perhaps some other edge should be constracted instead.

Garland and Heckbert [53] showed that there is a smarter way to place the vertex than to simply use the edge midpoint. They place the vertex at the position that minimizes *quadric error measure*, the sum of the squares of the distances to the planes supporting the triangles incident to the original pair. These planes are "carried along" with the coalesced vertex, so that if that vertex is then coalesced with another vertex, the next position is the least-squares position taking into account all the relevant planes from the initial triangulation. A priority queue is used to determine which edge to contract next, the edge whose contraction would introduce the least quadric error. Figure 31 shows an example.

Quadric-error simplification may sound expensive, but it is actually quite efficient: each coalesced vertex carries along only a single matrix, the sum of all the matrices of the vertices that went into it. To be more specific, a plane in $\mathbb{R}^3$ can be given by an equation

$n_i \cdot u = d_i$, where n_i is a normal vector and d_i is the plane's distance from the origin. The square of the distance from a vertex $v \in \mathbb{R}^3$, given as a column vector, to this plane is then

$$(v, 1)^T \cdot (n_i, d_i) \cdot (n_i, d_i)^T \cdot (v, 1).$$

The outer product $(n_i, d_i) \cdot (n_i, d_i)^T$ is a 4×4 matrix M_i called the *quadric* matrix. The sum of the squares of the distances from v to a number of different planes, (n_1, d_2), $(n_2, d_2), \ldots$, can be computed as $(v, 1)^T \cdot (\sum_i M_i) \cdot (v, 1)$. Quadric-error simplification generalizes easily to higher dimensions and even to higher-order basis functions; we think it should be useful in solution-adaptive meshing. Quadric error may also be useful in refinement and smoothing. Refinement could, for example, split a given triangle by adding the point that minimizes the sum of distances to the planes at the three neighboring triangles, and split triangles throughout the mesh according to priority order. In addition to repositioning vertices, smoothing could balance quadric matrices of adjacent vertices, subtracting an outer product off one and adding it to the other, in order to reduce total quadric error.

Teng, Talmor, and Miller [81] gave an algorithm for *a priori* simplification. Their algorithm uses Delaunay triangulations of centers of ball packings, and produces a sequence of bounded-aspect-ratio, node-conforming meshes of approximately minimum depth (number of levels of refinement). They pointed out that the minimum depth of the sequence of meshes may be dictated by the initial mesh and be much larger than $O(\log n)$. Their algorithm recovers a spacing function and then changes the spacing function in order to decide which balls centers to remove. An alternative algorithm [86], which gives a sequence of depth $O(\log n)$ but cannot guarantee bounded aspect ratios, simply removes a maximal independent set of vertices at each stage. (A *maximal independent set* is a set of vertices with no pair adjacent such that no other vertex can be added to the set without forming an adjacency.)

8 Cartesian Meshes and Subdivision Surfaces

In this section we mention two popular types of meshes based on recursive subdivision. Neither of these constructions comes out of the computational geometry community, and they are somewhat out of the scope of this survey (and beyond the range of expertise of the author!), so our remarks will be brief.

A *Cartesian mesh* is a mesh produced by repeatedly splitting a square into four half-size squares or a cube into eight half-size cubes, as in a quadtree or an octree. Squares or cubes are repeatedly subdivided until they are small to enough to capture the desired level of detail in the domain geometry and the expected solution. Typically a grading constraint is imposed, such as no square is next to one more than twice its own size.

Cartesian meshes do not attempt to conform to domain geometry, thereby transferring some burdens from the mesh generator to the solver. 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].

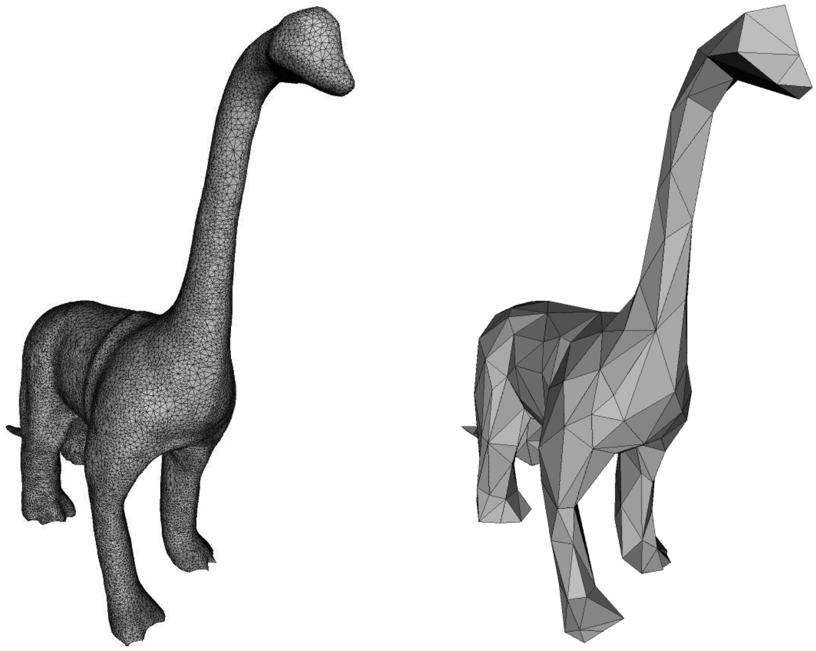

Fig. 31. A 47,904-face brontosaurus model (a) along with a 500-face quadric-based simplification. Notice how the triangles conform to the geometry. (M. Garland and P. Heckbert)

Recent fluid dynamics simulations have used Cartesian meshes quite successfully in both finite element and finite volume formulations of Euler equations [40, 41, 121]. The approach can be adapted even to very difficult meshing problems. For example, Berger and Oliger [19] and Berger and Colella [18] have developed adaptive Cartesian-based methods for rotational flows and flows with strong shocks. Cartesian meshes are not suitable for viscous flows with boundary layers, which require precisely conforming meshes.

A *subdivision surface* is a surface produced by starting from an initial very simple surface mesh and then repeatedly splitting facets and displacing vertices in space. One can generate a smooth curved surface by displacing each vertex to a weighted average of neighboring vertices; this process is the 3D analogue of converging to a cubic spline curve by repeatedly cutting off corners of a control polygon. There are a number of choices of subdivision patterns and averaging stencils; subdivision surfaces popular in computer graphics include Catmull-Clark surfaces [32] and Loop surfaces [72]. Subdivision surfaces have numerous advantages over other sorts of surfaces such as NURBS patches, including multiresolution editing [122] and signal-processing-style filtering and smoothing [59].

At a given stage of refinement, a subdivision surface is a *semi-regular* surface mesh: the initial unstructured mesh, uniformly refined by a subdivision pattern such as the regular 1:4 refinement shown in Figure 27(b). Because a surface mesh does not usually

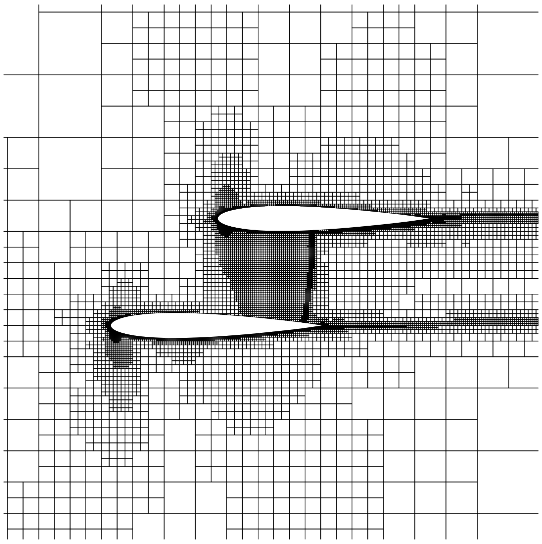

Fig. 32. A Cartesian mesh is a graded quadtree or octree that need not conform to the boundary of the domain. (W. Coirier)

need to conform to a complicated one-dimensional boundary, subdivision surfaces can be more pure than 2D finite element meshes produced by similar techniques (for example, Figure 22).

Lee et al. [68] gave a nice algorithm called MAPS (for "multiresolution adaptive parameterization of surfaces") for remeshing an arbitrary unstructured triangular surface mesh into such a semi-regular surface mesh. The algorithm starts by simplifying the unstructured mesh by iteratively removing maximal independent sets of vertices. When a vertex v is removed, the *star* of faces around v is flattened into the plane by a piecewise-linear approximation of conformal mapping, and the resulting flat polygon is triangulated and folded to fit into the hole produced by the removal of the star of v. The conformal mapping takes the removed vertex to a point in the new triangulation; we can think of the removed point and its incident edges as drawn on the new triangulation. "Drawings" are carried along from triangle to triangle as the mesh simplifies. At the end of the simplification process, we have reduced the initial surface mesh to a very simple base domain such as an octahedron or a polyhedral torus.

We now switch from simplification to refinement. We subdivide the mesh on the base domain using regular 1:4 refinement; newly added midpoints are displaced back into space according to the drawings on the adjacent triangles. The parameterization of the subdivision surface is not smooth across triangles of the base domain. This could be an advantage, for example if the original surface includes creases [66], or it could be an undesirable artifact, in which case the parameterization can be smoothed by weighted averaging [68]. MAPS does not have a provable guarantee; if we are unlucky drawings could fold over on themselves and spoil the bijective parameterization. A provably good version of MAPS would be of interest; a related problem for which there are theoretically justified algorithms [58] asks for a piecewise-linear homeomorphism between two polygonal domains.

9 Geometry Compression

There has been a flurry of recent work in computer graphics on *geometry compression*, meaning data compression of meshes [2, 42, 56, 60, 66, 99, 110, 113]. Since meshes can be quite large, geometry compression has a number of potential applications, for example, interactive gaming over the Web. One can think of the problem as having two parts: encoding the topology (connectivity) of the mesh, and encoding the coordinates of the vertices. A completely uncompressed encoding might specify the topology by adjacency lists and coordinates simply by 16-bit integers. (For computer graphics models and all but the most detailed finite element simulations, 16 bits are sufficient.) In such an encoding, an average vertex would require a total of about $9 \times 16 = 144$ bits, because it would use about six 16-bit integers for its neighbors and three more for its coordinates.

One idea for compression [113] uses a spanning tree on the vertices of the mesh, along with the dual tree on the faces that would result from cutting along all the edges of the vertex-spanning tree. The vertex tree can be used to predict (and hence compress) the coordinates of vertices from previous vertex coordinates along an in-order traversal of the tree. Thus this method prefers a long skinny vertex tree with many degree-2 nodes. The face tree can be used to encode the topology of the tree; a few bits per face suffice to encode the adjacency (across which edge is the next face adjacent) of the child faces in the tree. Using this method, the topology encoding drops to only about 2 bits per vertex and the coordinate encoding to about 10–16 bits per vertex, rather than 16 bits per coordinate. Another factor of almost two in the compression can be achieved by predicting vertex coordinates using a more two-dimensional predictor, such as incident triangles in the face tree, or such as layers of triangles determined by a breadth-first traversal from a starting vertex [56, 99]. Rossignac has released software called EDGEBREAKER for lossless compression of surface meshes [99].

A recent celebrated paper [66] demonstrated still another four-fold or more improvement in compression, down to about 2–4 bits per vertex for both topology and coordinates. This method, by Khodakovsky, Schröder and Sweldens, remeshes the arbitrary unstructured surface mesh into a semi-regular subdivision surface mesh using the MAPS algorithm of Lee et al. [68] described above. The remeshing is "almost lossless" in the sense that the new surface is visually indistinguishable from the old, even though triangle shapes and orientations may have changed. Due to the change in triangle shapes, this method is probably more appropriate for graphics applications than for physical simulation. Subdivision meshes compress extremely well: the topology after the initial unstructured mesh is automatic in the subdivision process, and the vertex coordinates are small deviations from predictions given by stencil-weighted averages of vertices one level up. (In fact, it is possible to give a general solid while constraining the deviations to be normal to the surface [60] and this observation gives a further improvement in compression [65].) Subdivision meshes are closely related to wavelet representations of functions, and well-developed wavelet compression methods such as zero-trees can be used to give very efficient encodings of coordinates. This compression scheme is also naturally progressive, meaning that initial sequences of the compressed data stream give better and better approximations to the surface.

10 Future Directions

We have mentioned future directions at various places in this survey. Here we gather together some of the previously mentioned directions along with some other possibilities.

1. It should be possible to prove efficiency bounds for solution-adaptive meshes for a given error tolerance, analogous to the "provably good" efficiency bounds for conforming to domain geometry, mentioned in Section 5.1.
2. Is there a tetrahedral meshing algorithm that will guarantee an M-matrix for elliptical PDEs? See Section 2 for a statement of sufficient geometric conditions for both finite element and finite volume formulations.
3. The geometry of cubical meshes (quads and hexes) is not very well understood compared to the geometry of simplicial meshes (triangles and tetrahedra). There are numerous open questions about cubical meshes. For examples: Can the flip moves for quad and hex meshes (Section 5.5) help to generate and improve cubical meshes? Is there a simple parity-switching "flip" for hex meshes? Is there an algorithm for no-small-angle quad meshing with a polynomial bound on the number of quadrilaterals? For no-large-angle quad meshing, there is such an algorithm, in fact one using only $O(n)$ quadrilaterals for an n-vertex polygonal domain (Section 5.2).
4. Related to unstructured quad and hex meshes is the problem of automatic multiblock meshing. Blocks in a multiblock mesh are topological cubes, typically with curved sides, each of which are meshed with a warped rectangular grid of cubes. One difference between multiblock meshing and unstructured quad or hex meshing is that it is often advantageous to curve blocks into horseshoe shapes, for example at leading and trailing edges of a wing, in order to confine small or anisotropic elements in the region where they are needed [76].
5. The stretching approach to generating anisotropic meshes (Section 5.3) suffers from the drawback of producing flat angles as easily as sharp angles. Are there better ways to produce *a priori* or *a posteriori* anisotropy? Section 5.3 mentions two possible approaches: square or rectangle packing and stiffly connected strings of bubbles.
6. So far there has been very little work on hybrid meshes or boundary layers in the computational geometry community. We would be expecially interested in hybrid meshes in which each boundary element has all of its faces either nearly parallel or nearly perpendicular to the boundary. For this to be possible, the domain boundary would have to satisfy certain angle bounds. It may be more convenient, at least in the initial theoretical studies, to assume a smooth curved domain boundary.
7. Adaptive mesh refinement is currently quite easy with unstructured triangular meshes, somewhat more difficult with tetrahedral meshes, and very difficult with quad or hex meshes. Are there better ways to refine tetrahedra, quads, and hexes?
8. So far there has been only a little work on subdivision volume meshes [11]. It would be nice to produce subdivision volume meshes whose boundaries are subdivision surface meshes, but this problem appears to be quite hard for general geometry. We are not aware of any work on geometry compression of 3D volume meshes.

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. P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proc. SIGGRAPH*, 2001.
3. 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.
4. N. Amenta, M. W. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proc. SIGGRAPH*, pages 415–421, 1998.
5. 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.
6. D. N. Arnold, A. Mukherjee, and L. Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM J. Sci. Computing*, 22:431–448, 2000.
7. D. N. Arnold and A. Mukherjee. Tetrahedral bisection and adaptive finite elements. In M. Bern et al., editor, *Grid Generation and Adaptive Algorithms*, pages 29–42. IMA Volumes in Mathematics and its Applications, Springer, 1999.
8. F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.
9. I. Babuška and A. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Analysis*, 13:214–227, 1976.
10. I. Babuška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM J. Numer. Analysis*, 15:736–754, 1978.
11. C. Bajaj, J. Warren, and G. Xu. A smooth subdivision scheme for hexahedral meshes. http://www.cs.rice.edu/∼jwarren/papers/hexahedral.pdf
12. T. J. Baker. Resolution of boundary layers on triangular meshes. In M. Bern et al., editor, *Grid Generation and Adaptive Algorithms*, pages 43–51. IMA Volumes in Mathematics and its Applications, Springer, 1999.
13. R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0.* SIAM Publications, Philadelphia, Penn., 1990.
14. 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.
15. 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.
16. E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *Impact of Computing in Science and Engineering*, 3:181–191, 1991.
17. 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.
18. M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.
19. M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
20. M. Bern, L.P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proc. 6th ACM Symp. Disc. Algorithms*, pages 189–196, 1995.
21. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry, 2nd Edition*, pages 47–123. World Scientific, Singapore, 1995.

22. M. Bern and D. Eppstein. Flipping cubical meshes. In *Proc. 10th International Meshing Roundtable*, 2001.
23. M. Bern and D. Eppstein. Optimal möbius transformations for information visualization and meshing. In *Proc. Workshop on Algorithms and Data Structures*, 2001.
24. M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. *J. Comp. System Sciences*, 48:384–409, 1994.
25. M. Bern, D. Eppstein, and J.R. Gilbert. Quadrilateral meshing by circle packing. *Int. J. Comp. Geometry and Applications*, 10:347–360, 2000.
26. M. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. *Disc. and Comp. Geometry*, 14:411–428, 1995.
27. M. Bern and P. Plassmann. Mesh generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 291–332. North-Holland, 2000.
28. M. Berzins. A solution based H^1 norm triangular mesh quality indicator. In M. Bern et al., editor, *Grid Generation and Adaptive Algorithms*, pages 77–89. IMA Volumes in Mathematics and its Applications, Springer, 1999.
29. F. J. Bossen and P. S. Heckbert. A pliant method for anisotropic mesh generation. In *Proc. 5th International Meshing Roundtable*, 1996.
30. 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.
31. 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.
32. E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design* 10(6):350–355, 1978.
33. Centaur Grid Generator. http://www.centaursoft.com/
34. S. Cheng and T. Dey. To appear in *Proc. 13th ACM-SIAM Symp. Disc. Algorithms*, 2002.
35. S.W. Cheng, T.K. Dey, H. Edelsbrunner, M.A. Facello, and S. Teng. Sliver exudation. In *Proc. 15th ACM Symp. Comp. Geometry*, pages 1–13, 1999. Also *J. Assoc. Comput. Mach.* 47:883–904, 2000.
36. L.P. Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Comp. Science Dept., Cornell University, 1989.
37. L.P. Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proc. 13th ACM Symp. Comp. Geometry*, pages 391–393, 1997.
38. P.G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, 1978.
39. 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.
40. W. J. Coirier. An adaptively-refined, Cartesian cell-based scheme for the Euler and Navier-Stokes equations. NASA Technical Memorandum 106754, NASA, October 1994.
41. 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.
42. M. Deering. Geometry compression. *Computer Graphics*, pages 13–20, 1995.
43. B. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.
44. H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.
45. H. Edelsbrunner and D. Guoy. Sink-insertion for mesh improvement. In *Proc. 17th ACM Symp. Comp. Geometry*, pages 115–123, 2001.
46. H. Edelsbrunner and D. Guoy. An experimental study of sliver exudation. In *Proc. 10th International Meshing Roundtable*, 2001.

47. H. Edelsbrunner, X.-L. Li, G. L. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng, A. Üngör, and N. Walkington. Smoothing and cleaning up slivers. In *Proc. ACM Symp. Theory of Computing*, 2000.
48. H. Edelsbrunner and T.-S. Tan. An upper bound for conforming Delaunay triangulations. *Disc. and Comp. Geometry*, 10:197–213, 1993.
49. D. A. Field. Laplacian smoothing and Delaunay triangulations. *Communications and Applied Numerical Methods*, 4:709–712, 1988.
50. P. Fleischmann. *Mesh Generation for Technology CAD in Three Dimensions* PhD Thesis, Technical University of Vienna, 1999.
51. On the road with the Opel CFD group. http://www.fluent.com/about/news/newsletters/01v10i1/a1.htm
52. 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.
53. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH*, pages 209–216, 1997. http://graphics.cs.uiuc.edu/∼garland/papers.html
54. A. S. Glassner. The perils of problematic parameterization. *IEEE Computer Graphics and Applications*, 2:78–83, September/October 1997.
55. L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992. Also in *Proc. 17th Int. Colloq. Automata, Languages and Programming*, Springer-Verlag, LNCS 443, 1990, pp. 414–431.
56. S. Gumhold and W. Straßer. Real time compression of triangle mesh connectivity. In *Proc. SIGGRAPH*, pages 133–140, 1998.
57. D. Guoy. *Tetrahedral mesh improvement, algorithms and experiments* PhD Thesis and Tech. Report UIUCDCS-R-2001-2225, University of Illinois, Computer Science Department, 2001.
58. H. Gupta and R. Wenger. Constructing piecewise linear homeomorphisms of simple polygons. *J. Algorithms*, 22(1):142–157, 1997.
59. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proc. SIGGRAPH*, pages 325–334, 1999. http://multires.caltech.edu/pubs/
60. I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In *Proc. SIGGRAPH*, 2000. http://multires.caltech.edu/pubs/
61. P. Heckbert. Collection of mesh generation links. http://www-2.cs.cmu.edu/afs/cs/user/ph/www/mesh.html
62. H. Hoppe. Progressive meshes. In *Proc. SIGGRAPH*, pages 99–108, 1996.
63. B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comput.*, 10:718–741, 1989.
64. 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.
65. A. Khodakovsky and I. Guskov. Normal mesh compression. http://multires.caltech.edu/pubs/
66. A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proc. SIGGRAPH*, 2000. http://multires.caltech.edu/pubs/
67. C. L. Lawson. Software for C^1 surface interpolation. In J. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.
68. A. W. F. Lee, W. Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *Proc. SIGGRAPH*, pages 95–104, 1998.
69. X. Li, S. Teng, and A. Üngör. Simultaneous refinement and coarsening: adaptive meshing with moving boundaries. In *Proc. 7th International Meshing Roundtable*, pages 201–210, 1998.

70. X. Li, S. Teng, and A. Üngör. Biting: advancing front meets sphere packing. In *Proc. 2nd Symp. Trends in Unstructured Mesh Generation (Boulder, Colorado)*, 1999.
71. X.-Y. Li and S. Teng. Generating well-shaped Delaunay meshes in 3D. In *Proc. 12th ACM-SIAM Symp. Disc. Algorithms*, pages 28–37, 2001.
72. C. Loop. *Smooth Subdivision based on Triangles* Master's Thesis, University of Utah, Dept. of Mathematics, 1987.
73. A. Liu and B. Joe. On the shape of tetrahedra from bisection. *Mathematics of Computation*, 63(207):141–154, July 1994.
74. 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.
75. S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *Int. J. Numer. Meth. Eng.*, 21:1403–1426, 1985.
76. N. Lu and P. R. Eisman. Interactive high quality grid generation. In *Proc. NASA Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions*, NASA-CP-3291, pages 833–843, May 1995.
77. I. MacPhedran and R. Young. Internet Finite Element Resources. http://www.engr.usask.ca/~macphed/finite/fe_resources/fe_resources.html
78. D. L. Marcum and N. P. Weatherill. Aerospace applications of solution adaptive finite element method. *Computer Aided Geometric Design*, 12:709–731, 1995.
79. J. M. Maubach. The number of similarity classes created by local n-simplicial bisection refinement. Manuscript, 1996.
80. D. J. Mavriplis. Adaptive mesh generation for viscous flows using Delaunay triangulation. *J. of Computational Physics*, 90(2):271–291, Oct. 1990.
81. G.L. Miller, D. Talmor, and S.-H. Teng. Optimal coarsening of unstructured meshes. In *Proc. 8th ACM-SIAM Symp. Disc. Algorithms*, pages 538–547, 1997.
82. G.L. Miller, D. Talmor, and S.-H. Teng. Data generation for geometric algorithms on nonuniform distributions. *Int. J. Computational Geometry and Applications*, 1998.
83. 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.
84. S. A. Mitchell and T. J. Tautges. Pillowing doublets: refining a mesh to ensure that faces share at most one edge. In *Proc. 4th International Meshing Roundtable*, pages 231–240. Sandia National Laboratories, 1995.
85. M. Murphy, D. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralization. In *Symp. on Discrete Algorithms*, pages 67–74, 2000.
86. C. F. Ollivier-Gooch. Multigrid acceleration of an upwind Euler solver on unstructured meshes. *AIAA Journal*, 33(10):1822–1827, 1995.
87. S. J. Owen. Meshing Research Corner. http://www.andrew.cmu.edu/user/sowen/mesh.html
88. 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.
89. J. Peraire, J. Peiró, and K. Morgan. Adaptive remeshing for 3-D compressible flow computations. *J. Comput. Physics*, 103:269–285, 1992.
90. M. Putti and C. Cordes. Finite element approximation of the diffusion operator on tetrahedra. *SIAM J. Sci. Comput.*, 19:1154–1168, 1998.
91. J.F. Randolph. *Calculus and Analytic Geometry*. Wadsworth, 1961.
92. S. Rippa. Minimal roughness property of the Delaunay triangulation. *Computer Aided Geometric Design*, 7:489–497, 1990.
93. S. Rippa. Long and thin triangles can be good for linear interpolation. *SIAM J. Numer. Analysis*, 29:257–270, 1992.
94. M.-C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Int. J. Numer. Meth. Eng.*, 20:745–756, 1984.

95. 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.
96. M.-C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM J. Numer. Analysis*, 21(3):604–613, June 1984.
97. 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.
98. I. G. Rosenberg and F. Stenger. A lower bound on the angles of triangles constructed by bisecting the longest side. *Math. Comp.*, 29:390–395, 1975.
99. J. Rossignac. Edgebreaker: connectivity compression for triangle meshes. Technical Report GIT-GVU-98-35, Georgia Tech, 1998.
100. J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, May 1995.
101. A. Saalfeld. Delaunay edge refinements. In *Proc. 3rd Candadian Conf. Comp. Geometry*, pages 33–36, 1991.
102. R. Schneiders. Mesh Generation and Grid Generation on the Web. http://www-users.informatik.rwth-aachen.de/∼roberts/meshgeneration.html
103. E. Schönhardt. Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Math. Annalen*, 98:309–312, 1928.
104. J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th ACM Symp. Comp. Geometry*, pages 76–85, 1998.
105. J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proc. 14th ACM Symp. Comp. Geometry*, pages 86–95, 1998.
106. J. R. Shewchuk. Mesh generation for domains with small angles. In *Proc. 16th ACM Symp. Comp. Geometry*, pages 350–359, 2000.
107. J. R. Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proc. 16th ACM Symp. Comp. Geometry*, pages 350–359, 2000.
108. K. Shimada. *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing*. PhD thesis, ME Dept., MIT, 1993.
109. 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.
110. J. Snoeyink and M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications. In *Proc. European Symp. on Algorithms*, 1997.
111. K. Stephenson. http://www.math.utk.edu/∼kens/
112. D. Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, School of Computer Science, Carnegie-Mellon University, 1997.
113. G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Trans. on Graphics*, 17:84–115, 1998.
114. Anisotropic adaptive unstructured mesh generation. Von Karman Institute for Fluid Dynamics. http://www.vki.ac.be/research/themes/cfd99/1.html
115. 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.
116. N. P. Weatherill, O. Hassan, D. L. Marcum, and M. J. Marchant. Grid generation by the delaunay triangulation. *von Karman Institute for Fluid Dynamics, Lecture Series*, 1994.
117. 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.
118. 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.

119. J. Xu and L. Zikatanov. A monotone finite element scheme for convection diffiusion equations. *Math. of Computation*, to appear.
120. S. Yamakawa and K. Shimada. High quality anisotropic tetrahedral mesh generation via ellipsoidal bubble packing. In *Proc. 9th International Meshing Roundtable*, pages 263–273, 2000.
121. 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.
122. D. Zorin, P. Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *Proc. SIGGRAPH*, pages 259–268, 1997.