Guaranteed-Quality Mesh Generation for Curved Surfaces *

L. Paul Chew
Department of Computer Science
Cornell University
Ithaca, NY 14853

ABSTRACT

For several commonly-used solution techniques for partial differential equations, the first step is to divide the problem region into simply-shaped elements, creating a mesh. We present a technique for creating high-quality triangular meshes for regions on curved surfaces. This technique is an extension of previous methods we developed for regions in the plane. For both flat and curved surfaces, the resulting meshes are guaranteed to exhibit the following properties: (1) internal and external boundaries are respected, (2) element shapes are guaranteed – all elements are triangles with angles between 30 and 120 degrees (with the exception of badly shaped elements that may be required by the specified boundary), and (3) element density can be controlled, producing small elements in "interesting" areas and large elements elsewhere. An additional contribution of this paper is the development of a practical generalization of *Delaunay triangulation* to curved surfaces.

1 INTRODUCTION

This paper includes three main contributions to the subject of mesh generation:

1. We show that a relatively simple mesh generating technique, based on earlier work by the author [Che89], can be used to create meshes in which (1) region boundaries are respected (including internal boundaries) and (2) all triangles are guaranteed to be both *well-shaped* and *well-sized*. Well-shaped triangles have angles that are between 30 and 120 degrees (smaller angles may appear if small angles appear as part of the boundary). *Well-sized* is defined by the user and can mean virtually anything as long as it can be achieved by shrinking the triangles. Typically, a triangle might be considered well-sized if its estimated error is small based on the user's intuition or on an earlier, presumably-cruder, numerical solution.
2. We extend this meshing technique to curved surfaces. Just as in the 2D meshing technique, triangles in the meshes produced by the curved-surface technique are guaranteed to respect boundaries and to be both well-shaped and well-sized. For curved surfaces, a user is likely to want *well-sized* to include the criterion that a triangle is a good approximation to the corresponding surface.
3. We develop a practical definition of *Delaunay triangulation* for curved surfaces. Both the 2D and the curved-surface techniques are based on Delaunay triangulations.

There is a large literature on the subject of mesh generation with most of the material emanating from the applications community. One good information source for the interested reader is the recent survey by Bern and Eppstein [BE92]; this paper also includes a very good bibliography. See [BEG90, MS92, MV92, Rup92] for recent work related to our results.

We assume the reader is familiar with Delaunay triangulations and with the incremental algorithm for producing them. We use a variation of the Delaunay triangulation called the *constrained Delaunay triangulation* (CDT). Basically, the CDT, like the Delaunay triangulation, can be built by edge-flipping, but for the CDT there are certain edges that cannot be flipped. The CDT is useful for mesh generation since it respects boundaries as well as having many of the properties of the Delaunay triangulation. The necessary background material is contained in, for instance, [BE92].

We develop the 2D meshing algorithm and prove some of its properties in Section 2. In Section 3 we describe the curved-surface Delaunay triangulation, which is used in Section 4 to create an algorithm for meshing regions on curved surfaces. The final section discuss practical improvements to the algorithms and some conclusions.

2 MESHING IN THE PLANE

We assume that our region-to-be-meshed has boundaries that consist of line segments (curved boundaries can be handled as well, but complicate the presentation). We use an implementation of CDT that supports the following operations:

- *Build:* given an initial set of sources (edges and vertices), build the CDT for this set.
- *Insert:* insert the given vertex into the CDT and update the CDT accordingly.

*This work was supported by the Advanced Research Projects Agency of the Department of Defense under ONR Contract N00014-92-J-1989, and by ONR Contract N00014-92-J-1839, NSF Contract IRI-9006137, and AFOSR Contract AFOSR-91-0328.

9th Annual Comptutational Geometry,5/93/CA,USA
© 1993 ACM 0-89791-583-6/93/0005/0274...$1.50

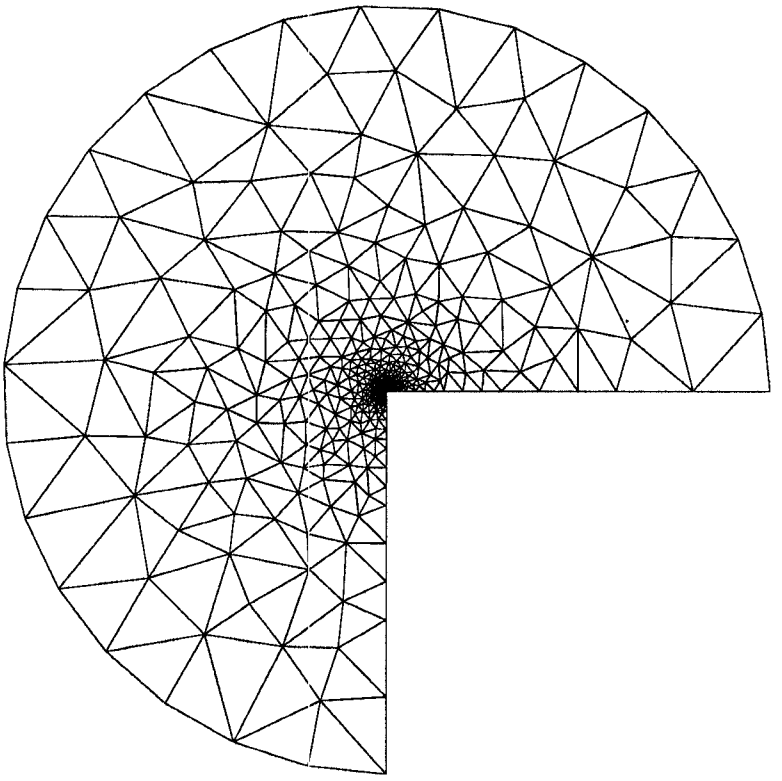

Figure 1: A mesh created using a size function to control element density.

- *Delete:* delete the given vertex from the CDT and update the CDT accordingly. (Note: endpoints of source-edges are never deleted.)
- *Split:* given a source-edge, split it in half by adding a new vertex and replacing the source edge by two new edges. In addition, we assume that *Split* has an option where the given source edge can be split into thirds instead of halves.
- *Travel:* Given a source vertex s (part of the CDT) and a destination vertex d (not necessarily part of the CDT), determine if s and d can "see" each other (i.e., determine if the segment from s to d intersects a source-edge at a non-endpoint). This is implemented by walking across the CDT from triangle to triangle. The following information is returned:
  - If s and d can see each other then return the triangle of the CDT that contains d;
  - Otherwise return the source-edge that blocks s from "seeing" d (i.e., the first source edge that is intersected on the way from s toward d).

During the meshing algorithm, triangles are graded according to two criteria: shape and size.

Definition 1 *A triangle is* well-shaped *if all its angles are greater than or equal to 30 degrees. A triangle is* well-sized *if it satisfies a user-supplied grading function. This function can use any user-chosen criteria at all as long as there exists a value $\delta > 0$ such that any well-shaped triangle that fits within a circle of radius δ would satisfy the grading function.*

For example, the user could specify a *size* function over the entire problem region; a triangle is well-sized if its circumradius is less than or equal to the value of the size function evaluated at the circumcenter. Figure 1 shows an example of a mesh generated using such a size function. This kind of function works as long as there is lower bound $\delta > 0$ on the sizes that can appear within the problem region. The user may want to use a size function derived from an error analysis of an earlier, presumably cruder, numerical solution to the problem. Another possible user-choice for well-sized triangles is to leave interior triangles completely unconstrained, but to require triangles with boundary edges to have boundary edges smaller than a chosen constant.

There are two kinds of vertices: (1) required vertices: those that are part of the boundary or those that are specifically chosen by the user, and (2) circumcenter vertices: additional vertices that are introduced during the meshing algorithm. During the meshing process it is sometimes necessary to eliminate vertices. Note that only circumcenter vertices can be eliminated; the other vertices must be retained.

To show that the following algorithm does not keep meshing forever, making smaller and smaller triangles, we show that no edge introduced by the algorithm will have length less than h, where h is defined as the minimum among (1) the smallest line-of-sight distance between two nonintersecting sources (vertices or edges) in the initial set of sources (note that objects on opposite sides of a source edge cannot "see" each other and are thus infinitely far apart), (2) one half the length of the smallest source edge that appears in the initial set of sources, and (3) the value δ associated with the user-supplied size grading function (by definition, δ is such that any well-shaped triangle that fits within a circle

of radius δ is also a well-sized triangle). The quantity h is used in the algorithm only in step 6.

To simplify our presentation, we assume that our initial problem region contains no angles that are less than 60 degrees. In fact, angles down to 30 degrees can be handled without significant modification to the algorithm. Angles less than 30 degrees can be removed from the problem region and triangulated in a postprocessing step; this type of "lopping off" technique is used in [BEG90, MS92, Rup92]

ALGORITHM A

1. Build the CDT for the initial set of sources (edges and vertices).
2. Grade any triangles that are currently ungraded. A triangle passes only if it is both (1) well-shaped: its smallest angle is greater than 30 degrees, and (2) well-sized: it satisfies a user-defined grading function as described in Definition 1.
3. If all triangles pass then Halt. Otherwise choose the largest triangle that fails (call it Δ) and determine its circumcenter (call it c).
4. Travel across the CDT from any vertex of Δ toward c until we either run into a source-edge or find the triangle containing c.
5. If we found the triangle containing c then Insert c into the CDT. Go to step 2.
6. If we ran into a source-edge then Split the edge and update the CDT. This is always a one-half split unless the length of the edge is between $2\sqrt{3}h$ and $4h$, in which case we use a one-third split. Let l be the length of the new edges and consider the one or two new vertices produced by the Split operation. Delete each circumcenter-vertex of the CDT that is closer than l (line-of-sight distance: an intervening source edge implies infinite distance) to a new vertex. Go to step 2.

In step 3, "largest triangle" means the triangle with the largest circumcircle. We could actually choose any failing triangle rather than the largest and the algorithm will produce a reasonable mesh. It might even take a bit less time, since choosing the largest requires maintaining a priority queue of some sort. However, experience has shown that the resulting mesh is more pleasing to the eye if larger triangles are done first. It would be useful to discover how well "pleasing to the eye" correlates with "better for finite element analysis."

It does not matter which triangle vertex we start at in Step 4, since the circumcenter we are looking for is the center of an *empty* circle. (Note: by definition, each triangle of the CDT has a circumcircle that is *empty* in the sense that no other vertices can be "seen" from the interior of the triangle.) Thus, during the Travel operation, we cross only edges whose endpoints are outside the circumcircle.

The one-third Split in Step 6 is used to significantly simplify the proof of Lemma 2. A more complicated proof can be used to show that such one-third splits are actually unnecessary (although some smaller edges might be produced). Also in Step 6, nearby vertices are deleted in order to avoid introducing new edges that are short. This too is important in the proof of Lemma 2.

Lemma 2 *At Step 2 of Algorithm A, no two vertices ever have line-of-sight distance less than h.*

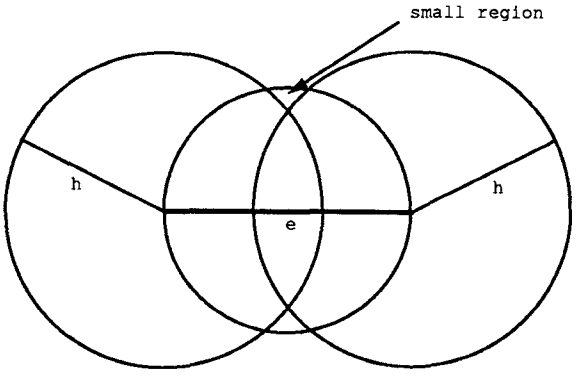

Figure 2: When e has length less than $\sqrt{3}h$, the triangle's vertex must lie within this small region.

Proof: The proof is by induction. Certainly the lemma holds initially. We assume the lemma holds at cycle i and show that the lemma must continue to hold for the next cycle. We need to check steps 5 and 6, the only steps where new vertices are introduced, to see that the lemma continues to hold.

In Step 5, the circumcenter of a failing triangle Δ is inserted into the CDT. Note that by definition of the CDT, Δ's circumcircle must be empty; thus, the new vertex is no closer to any other vertex than the radius of this circumcircle. We claim that this radius is greater than or equal to h. To see this, note that triangle Δ must have failed during the grading process. There are 2 ways for a triangle to fail:

1. Triangle Δ has a small angle. By simple geometry, a small angle has a small side opposite it; in fact, an angle is less than 30 degrees iff the opposite side has length less than the circumcircle radius. Thus, when there is a small angle, the radius is larger than the length of some edge that already exists in the CDT, which by the induction hypothesis has length greater than or equal to h.
2. Triangle Δ is not well-sized according to the user's size-grading function. To produce an edge of length less than h, the radius of Δ's circumcircle must be less than h. But when the circumcircle is less than h, Δ is automatically well-sized due to the definition of well-sized and the way h was chosen.

In Step 6, a source-edge is split into either two or three pieces and nearby vertices are eliminated. A source edge e is split only when there is a triangle on one side of e with the triangle's circumcenter on the other side of e. Assume that splitting e produces new edges that are too small. Let d be the length of e. Length d must be between h (by the induction hypothesis) and $2h$. We claim that if d is less than $\sqrt{3}h$ then no splitting can occur.

Too see this, observe that, with e so short, the locations of the vertices of triangle Δ, the triangle that caused the split, are very constrained. The endpoints of e are possible vertices of Δ. Any other vertex (or vertices) of Δ must (1) lie in a region within or on the circle that has e as its diameter (this must be true to make the circumcenter fall on the correct side of e) and (2) must lie outside circles of radius h about each endpoint of e by the induction hypothesis.

The resulting small region (see Figure 2) is too small to fit more than single vertex; more vertices in the region would contradict the induction hypothesis. Thus, the vertices of Δ are the endpoints of e plus an additional vertex v that lies within the small region described above. This small region is so small, in fact, that all the possible triangles that use v and the endpoints of e are well-shaped and have circumcircles with radius less than h; thus, by the definition of well-sized, the triangles are also well-sized. Any such triangle would pass in our grading step and could not possibly lead to edge splitting.

At this point, we have shown that our source-edge e, that is assumed to split into edges that are too small, must have length between $\sqrt{3}h$ and $2h$. We refer to an edge with length in this range as a *bad edge*. We claim that bad edges can never occur. To see this, we first note that, by the way h was defined, we initially start with all edges having length greater than or equal to $2h$. Thus, a bad edge appears only if it is the result of an earlier Split operation, either a half-split or a third-split. To be the result of a half-split, the unsplit edge must have had length between $2\sqrt{3}h$ and $4h$, but edges with length in this range are never split in half; such edges are split into thirds, producing edges with lengths in the range $1.1547h$ and $\frac{4}{3}h$ (outside the bad range and still of length greater than h). To be the result of a third-split, the unsplit edge must have had length between $3\sqrt{3}h$ and $6h$, but edges with lengths in this range are never split into thirds. Thus, bad edges never occur, so new source-edges can never have length less than h.

We still need to show that the new splitting-vertex is not too close to other vertices of our mesh. This is taken care of by the the elimination of nearby vertices that is done in the remainder of step 6. Our definition of h and our restriction that no angles less than 60 degrees occur in our original region ensure that required vertices cannot be too close. □

This lemma is just what we need to show that Algorithm A halts. Note that each cycle of the algorithm introduces either a new vertex along the boundary (which may eliminate some vertices in the interior) or a new vertex in the interior. The boundary has finite length and adjacent vertices on the boundary can be no closer than h, so points are added to the boundary at most a finite number of times. The interior of the region has finite area and, since vertices are no closer than h, there are only finitely many vertices that can be fit into the interior of the region. Thus, we have the following theorem.

Theorem 3 *Algorithm A halts.*

Since Algorithm A halts, all the triangles of the mesh it produces must be triangles that pass during the grading step. Thus meshes produced by Algorithm A have the following properties:

- Boundaries and vertices specified by the user are respected. Triangles do not cross boundaries. User-specified vertices are never eliminated.
- All triangles have angles greater than 30 degrees. (Smaller angles are required if we allow boundary angles less than 30 degrees.)
- The user can control triangle size. Any kind of size criterion can be chosen as long as it can be achieved by making triangles smaller.

It would be nice if we could show an additional desirable property: that the number of triangles produced is, in some sense, optimal. For variable-density meshing, all of the optimality results of which we are aware [BEG90, MS92, MV92, Rup92] prove their resulting mesh optimal within a constant factor in the sense that the number of triangles they produce is within a constant factor of the number of triangles in the best possible mesh that achieves the same triangle-quality. Unfortunately, the constant-factors in the proofs are extremely large; thus, the optimality results are not very helpful in terms of choosing which algorithm produces the smallest number of triangles in practice. As in Ruppert's work [Rup92], Algorithm A is derived from the techniques presented in [Che89]; thus, an optimality result for Algorithm A (or a slightly modified version of Algorithm A) can probably be derived using some of Ruppert's techniques. In practice, it appears that Algorithm A does not produce many more triangles than those necessary to achieve the 30 degree bound and reach the user's size demands.

3 CIRCLES ON CURVED SURFACES

We wish to extend Algorithm A to deal with meshes on curved surfaces. Algorithm A used two basic ideas to create a mesh: (1) the mesh is a Delaunay triangulation and (2) new vertices should be circumcenters of Delaunay triangles. To make these ideas work for curved surfaces we have to determine reasonable definitions for a *Delaunay triangulation* on a curved surface and for the *circumcenter* of a triangle on a curved surface. Both of these things come down to determining what is meant by a *circle* on a curved surface.

One possible definition for a *circle*, perhaps the most obvious such definition, is to use geodesic distance along the surface. Under this plan, a circle about a given center is the set of points on the surface that are equidistant (taking the shortest route along the surface) from the center. This has two major disadvantages: (1) for most surfaces, it is very difficult to calculate geodesic distances and (2) odd behaviors are possible (for instance, consider a circle-center on the side of a very steep hill – the circle can reach all the way around the hill, intersecting itself, without including the top of the hill).

It does not seem possible to completely eliminate this kind of odd behavior, but we can at least attack the first difficulty by using a different definition of *circle*.

Definition 4 *Given three vertices on a curved surface, consider the infinite set of spheres through the three vertices. The centers of all the spheres lie on a single line. We choose the sphere whose center is on the surface and define the* circumcircle *of the three vertices to be the set of points where this sphere intersects the surface.*

This definition has some advantages:

- It is easy to determine if a given point on the surface is inside or outside of a given circumcircle – we simply calculate distances in 3-space. The operation Point-in-Circumcircle is a fundamental operation for building Delaunay triangulations.
- A *circumcenter* is always a point that is on the curved surface. This property is important since the vertices of a curved-surface mesh should lie upon the surface and, in our meshing algorithm, new vertices are always circumcenters.
- Finding a circumcenter is basically equivalent to the operation Intersect-Line-with-Surface. This operation

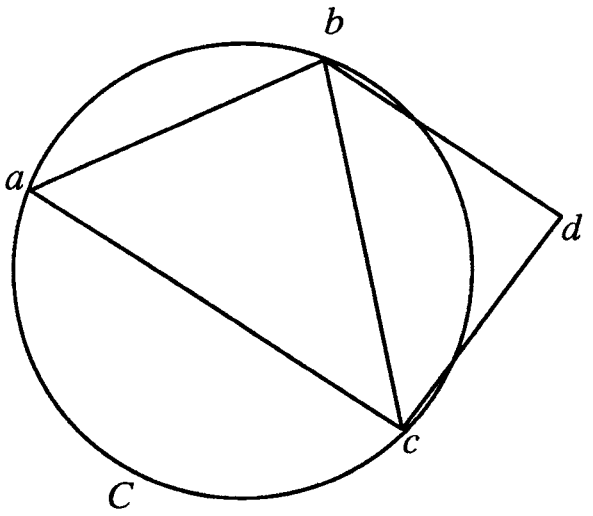

Figure 3: In a standard 2D Delaunay triangulation, circumcircles are consistent.

is available for all reasonable surface representation systems. (Unfortunately, this operation, although always available, is not necessarily an easy operation. For instance, when a surface is represented by parametric patches, intersecting the surface with a line is likely to require the use of an iterative method.)

This definition of circumcircle still has some difficulties. In particular, the line on which the sphere-centers lie may intersect the surface more than once or it may not intersect the surface at all. We claim that for triangulations that are reasonable – reasonable in the sense that they are somewhat close to approximating their curved surface – these difficulties either do not occur or can be easily resolved. Thus, this definition of circumcircle is useful if we have some way of creating a relatively crude, initial triangulation. This can be done by, for instance, intersecting the surface with a small-enough grid, or, in the case of surfaces represented by parametric patches, creating an initial triangulation by triangulating in the parameter space.

At this point, we still need to show that the definition of circumcircle that we have suggested is an appropriate definition for building Delaunay triangulations. In particular, we need to prove a property that usually remains unstated for the regular 2D Delaunay triangulation: we need to show that adjacent triangles have *consistent* circumcircles. That is, given edge bc with adjacent triangles abc and bcd, we need to show that d is within the circumcircle of abc iff a is within the circumcircle of bcd. In other words, we need to show that when we check the empty-circle property for a prospective Delaunay edge, we get the same answer regardless of which side we start from.

As an example, consider the 2D version of this problem (see Figure 3). Let C be the circumcircle of triangle abc and suppose d is outside of C. Now move C toward d while keeping b and c on its boundary (this will require that C change size). By the time C touches d it is clear that a is no longer within C; thus, the circumcircle of bcd does not contain a and the two triangles have consistent circumcircles.

For our 3D version, we let S represent the sphere through triangle abc that has its center on our curved surface. Call the sphere's center p. Suppose d is outside of S. Now move p, the center of S, along the surface toward d while keeping b and c on the boundary of S. Consider a plane through bc and perpendicular to the motion of the center. This plane cuts S into two pieces: on one side the sphere grows as the the center moves; on the other side the sphere shrinks. We need to ensure that (1) one side only shrinks and the other side only grows and (2) a is always on the side of the cutting plane where the sphere is shrinking. Both of these requirements can be met if we constrain the surface so that it does not bend too much in the region of our moving sphere. Thus, we have the following lemma.

Lemma 5 *Given a surface and given vertices a, b, c, and d on the surface where triangles abc, bcd, abd, and acd all have well-defined circumcenters, if the surface normals on the portion of the surface that is within the union of the circumcircles vary by less than $\frac{\pi}{2}$ then the triangles have consistent circumcircles.*

Assuming we can start with an initial, relatively crude triangulation with the property that the surface does not bend too much near any pair of adjacent triangles, we can build something that can reasonably be called a *Delaunay triangulation.* Once we have a Delaunay triangulation, we also have a CDT since, intuitively, a CDT is a Delaunay triangulation in which certain edges cannot be flipped.

4 THE CURVED-SURFACE MESHING ALGORITHM

Just as in Algorithm A, triangles are graded according to two criteria: shape and size. Triangles are well-shaped if angles are greater than or equal to 30 degrees. It is important to note that our goal is to make meshes with triangles that are well-shaped as triangles in 3-space; thus, angles are measured in the plane of the triangle's vertices. Triangles are well-sized if they pass a user-defined grading function, where the function is such that a small-enough, well-shaped triangle is always well-sized (see definition 1). For curved-surface meshing, it is expected that the user's size-grading function will take into account some measure of how well a given triangle matches the curved surface. This type of criterion is easily seen to satisfy Definition 1 as long as surface curvature is bounded away from infinity.

ALGORITHM B: Meshing for a region on a curved surface.

1. Build an initial crude triangulation with the property that the surface normals in a region around each adjacent pair of triangles vary by less than $\frac{\pi}{2}$. Use edge flips to make the initial triangulation into a curved-surface CDT. Some subdivision of triangles may be necessary.
2. Grade any triangles that are currently ungraded. A triangle passes only if it is both well-shaped and well-sized.
3. If all triangles pass then Halt. Otherwise choose the largest triangle that fails (call it Δ) and determine its surface-circumcenter (call it c).
4. Travel across the CDT from any vertex of Δ toward c (any path will work as long as we stay within the circumcircle of Δ) until we either run into a source-edge or find the triangle containing c.

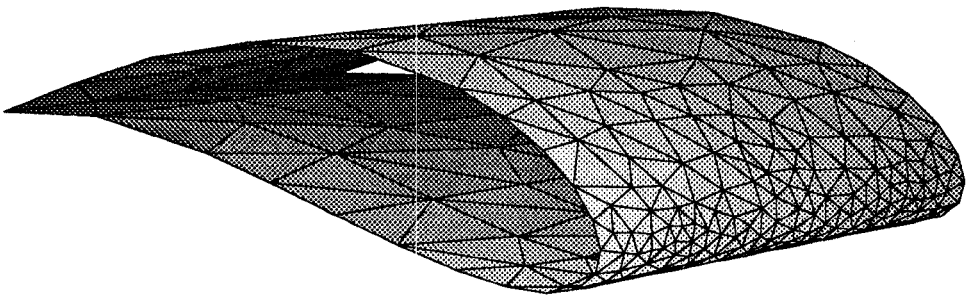

Figure 4: A curved surface mesh of a portion of a wing.

5. If we found the triangle containing c then Insert c into the CDT. Go to step 2.
6. If we ran into a source-edge then Split the edge and update the CDT. As in the 2D version of this algorithm, edges with lengths within a particular range can be split in thirds to simplify some proofs. In practice, splitting in half appears to always work. Note that, since boundaries may be curved, splitting in half here means finding a point on the boundary (and on the surface) that is equidistant from each endpoint. Let l be the (straight-line) length of the new edges and consider the one or two new vertices produced by the Split operation. Delete each circumcenter-vertex of the CDT that is closer than l (line-of-sight distance: an intervening source edge implies infinite distance) to a new vertex. Go to step 2.

As in the 2D case, if the algorithm halts then the resulting mesh consists entirely of triangles with desirable properties. For our curved surface version these properties can include that each triangle is a good approximation to the surface. See Figure 4 for an example surface mesh.

To show the algorithm halts, we need to show that very-short edges are not produced. In a sense, this is easier for curved surface meshes than for flat meshes, since the circumcenter of a triangle is closest to the triangle's vertices for flat circumcenters (i.e., when the center of the sphere is in the same plane as the vertices); a curved-surface circumcenter is usually farther away from the vertices. In the same way, splitting a curved boundary produces a "midpoint" that is farther from the endpoints than the midpoint of a straight-line boundary.

Boundary splitting can be a bit complicated, since the new boundary may change the status of some nearby vertices, moving them from inside the region to outside the region or *vice versa*. This is resolved by extending the surface-normal requirement to boundary-normals (vectors that are normal to the boundary and tangent to the surface). If the normals along a boundary do not vary by more than $\frac{\pi}{2}$ then any nearby vertices that change status are close enough that they are eliminated later in the splitting step.

5 SOME PRACTICAL IMPROVEMENTS AND CONCLUSIONS

Our meshing technique is heavily based on our ability to maintain a Delaunay triangulation, flipping edges as necessary to recover Delaunay properties after the insertion of a vertex. This entails checking adjacent triangles to see if the opposite vertex of one triangle is within the circumcircle of the other. Even though we have created reasonable definitions of *circumcircle* and *circumcenter*, it is still nontrivial to use these definitions in practice, since finding them may require iterative techniques (e.g., Newton's Method). Fortunately, experiment has shown our meshing technique to be extremely robust in the sense that, in practice, it appears we can

- Ask for better-shaped triangles. Angles between 35 and 100 degrees have caused no problems except in pathological cases specifically constructed to cause difficulties.
- Use approximate Delaunay triangulations. Instead of empty circumcircles, we can make-do with nearly-empty circumcircles – nearly-empty in the sense that any vertices inside a nearly-empty circle are close to the circle's boundary.
- Use approximate circumcenters. If we use "centers" that are within distance fR of the true centers, where R is the circle's radius and f is a fixed fraction less than 1, then we can modify the meshing algorithm and prove that all angles in the resulting mesh are greater than $\arcsin \frac{1-f}{2}$. For $f = 0.5$ this is about 14.5 degrees. In practice, it appears that the angle bound of 30 degrees can be maintained even when approximate circumcenters are used. The use of approximate circumcenters is particularly effective at speeding up the generation of meshes for curved surfaces.

We have developed 2D and curved-surface meshing techniques and shown that these techniques create meshes with the following properties:

- Boundaries and vertices specified by the user are respected. Triangles do not cross boundaries. User-specified vertices are never eliminated.
- All triangles have angles greater than 30 degrees. (Smaller angles are required if we allow boundary angles less than 30 degrees.) Note that a triangle's angles are always measured in the plane of the triangle.
- The user can control triangle size. Any kind of size criterion can be chosen as long as it can be achieved by making triangles smaller. Typical size criteria are based on error estimates (either intuitive or calculated) and on surface curvature.

The curved-surface meshing algorithm is based on our development of a curved-surface Delaunay triangulation. Such a Delaunay triangulation is well-defined only if the given

surface is smooth (surface normals vary by less than $\frac{\pi}{2}$) in a region about each triangle. Fortunately, this restriction is not a severe handicap – simple techniques can be used to generate a crude triangulation that is sufficiently dense for Delaunay operations (and our meshing technique) to be valid. For examples that we have tried, the initial crude triangulation typically consists of two to a few dozen triangles.

REFERENCES

[Rup92] J. Ruppert, *A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation*, Report UCB/CSD 92/694, University of California, Berkeley, 1992.

[BE92] M. Bern and D. Eppstein, Mesh Generation and Optimal Triangulation, *Computing in Euclidean Geometry*, edited by F. K. Hwang and D.-Z. Du, World Scientific, 1992, to appear. Also appears as Tech Report CSL-92-1, Xerox PARC, March 1992.

[BEG90] M. Bern, D. Eppstein, and J. R. Gilbert, Provably Good Mesh Generation, *Proceedings of the 31st IEEE Symposium on the Foundations of Computer Science*, 231-241, 1990. To appear in *JCSS*.

[Che89] L. P. Chew, *Guaranteed-Quality Triangular Meshes*, Department of Computer Science Tech Report TR 89-983, Cornell University, 1989.

[MV92] S. A. Mitchell and S. A. Vavasis, Quality Mesh Generation in Three Dimensions, *Proceedings of the Eighth Annual Symposium on Computational Geometry*, 212-221, ACM Press, 1992. Full version in Department of Computer Science Tech Report TR 92-1267, Cornell University, 1992.

[MS92] E. A. Melissaratos and D. L. Souvaine, Coping with Inconsistencies: A New Approach to Produce Quality Triangulations of Polygons with Holes, *Proceedings of the Eighth Annual Symposium on Computational Geometry*, 202-211, ACM Press, 1992.