Mesh Generation for Domains with Small Angles

Jonathan Richard Shewchuk
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720
jrs@cs.berkeley.edu

Abstract

Nonmanifold geometric domains having small angles present special problems for triangular and tetrahedral mesh generators. Although small angles inherent in the input geometry cannot be removed, one would like to find a way to triangulate a domain without creating any *new* small angles. Unfortunately, this problem is not always soluble. I discuss how mesh generation algorithms based on Delaunay refinement can be modified to ensure that they always produce a mesh. A two-dimensional algorithm presented here creates a mesh with no new angle smaller than $\arcsin[\sin(\phi/2)/\sqrt{2}]$, where $\phi \le 60^\circ$ is the smallest angle separating two segments of the input domain. Furthermore, new angles smaller than 20.7° appear only near input angles smaller than 60°. In practice, the algorithm's performance is better than these bounds suggest. A three-dimensional algorithm presented here creates a mesh in which all tetrahedra have circumradius-to-shortest edge ratios no greater than two, except near acute input angles (angles separating segments and/or facets of the input domain).

1. Introduction

The Delaunay refinement algorithms for triangular mesh generation introduced by Jim Ruppert [4] and Paul Chew [1] are largely satisfying in theory and in practice. However, one unresolved problem has limited their applicability: they do not always mesh domains with small angles well—or at all—especially if these domains are nonmanifold.

This problem is not just true of Delaunay refinement algorithms; it stems from a difficulty inherent to triangular mesh generation. Of course, a meshing algorithm must respect the input domain; small input angles cannot be removed. However, one would like to believe that it is possible to triangulate a domain without creating any small angles that aren't already present in the input. Unfortunately, no algorithm can make this guarantee for all straight-line domains, as Section 2 shows.

Supported in part by the National Science Foundation under Awards ACI-9875170, CMS-9980063, and EIA-9802069, and by gifts from the Okawa Foundation and Intel.

Therefore, to be universally applicable, a mesh generation algorithm must make decisions about where to create triangles that have small angles. The first result of this paper, presented in Section 6, is a modification to Ruppert's algorithm that is guaranteed to terminate and produce a mesh that has poor-quality triangles only in the vicinity of small input angles. Specifically, if the worst nearby input angle is ϕ, where $\phi \le 60^\circ$, no new angle smaller than $\arcsin[\sin(\phi/2)/\sqrt{2}]$ is generated. Section 6 formalizes exactly what "nearby" means. An implementation demonstrates that the algorithm typically far outperforms this worst-case bound. Ruppert's algorithm and its analysis are presented in Sections 3 and 4 as a necessary precursor of my results. The second result, sketched in Section 8, is a similar approach to the notably more difficult problem of meshing three-dimensional domains with small angles.

The input to Ruppert's or Chew's algorithm is a *planar straight line graph* (PSLG), which is a set of vertices and segments as illustrated in Figure 1(a). A segment is an edge that must be represented by a sequence of contiguous edges in the final mesh. A PSLG is required to contain both endpoints of every segment it contains, and a segment may intersect vertices and other segments only at its endpoints. The input to three-dimensional Delaunay refinement is a *piecewise linear complex* (PLC), which is a set of vertices, segments, and facets as illustrated in Figure 1(c). A facet is a planar surface that must be represented as a union of triangular faces in the final mesh. A facet can be quite complicated in shape; it is a polygon (not necessarily convex), possibly augmented by polygonal holes, slits, and vertices in its interior.

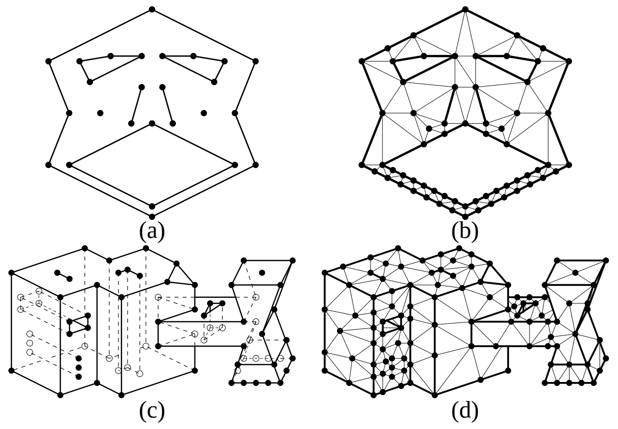

Figure 1: Top: A PSLG and a mesh generated by Ruppert's Delaunay refinement algorithm. Bottom: A PLC and a mesh thereof.

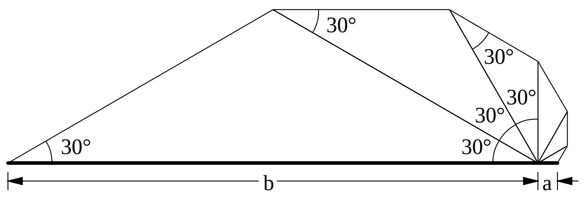

Figure 2: In any triangulation with no angle smaller than $30°$, the ratio b/a cannot exceed 27.

The process of tetrahedral mesh generation necessarily divides each segment into smaller edges called *subsegments*. The bold edges in Figures 1(b) and 1(d) are subsegments; other edges are not. Similarly, each facet is subdivided into triangular faces called *subfacets*. All of the triangular faces visible in Figure 1(d) are subfacets, but most of the faces in the interior of the tetrahedralization are not.

The *triangulation domain* is the region that a user wishes to triangulate. For the purposes of mesh generation, a PSLG must be *segment-bounded*, meaning that segments of the PSLG entirely cover the boundary separating the triangulation domain from its complement, the *exterior domain*. Similarly, a PLC must be *facet-bounded*.

2. A Negative Result on Quality Triangulations of PSLGs that Have Small Angles

For any angle bound $\theta > 0$, there exists a PSLG X such that it is not possible to triangulate X without creating a new corner (not present in X) whose angle is smaller than θ. This statement applies to any triangulation algorithm, and not just those discussed herein.

The result holds for certain PSLGs that have an angle much smaller than θ. Suppose two collinear subsegments of lengths a and b share a common endpoint, as illustrated in Figure 2. Mitchell [3] proves that if the triangles incident on the common endpoint have no angle smaller than θ, then the ratio b/a has an upper bound of $(2\cos\theta)^{180°/\theta}$. (This bound is tight if $180°/\theta$ is an integer. Figure 2 offers an example where the bound is obtained.) Hence, any bound on the smallest angle of a triangulation imposes a limit on the gradation of triangle sizes.

A problem can arise if three segments intersect at a vertex o, with two segments separated by a tiny angle ϕ and two separated by a much larger angle. Figure 3 (top) illustrates this circumstance. Suppose that the middle segment of the three is split by a vertex p, which may be present in the input or may have been inserted to help achieve the angle constraint elsewhere in the triangulation. The insertion of p forces the narrow wedge between the first two segments to be triangulated (Figure 3, center), which may necessitate the insertion of a new vertex q on the middle segment. Let $a = |pq|$ and $b = |op|$ as illustrated. If the angle bound is respected, the length a cannot be large; the ratio a/b cannot exceed

$$\frac{\sin\phi}{\sin\theta}\left(\cos(\theta+\phi) + \frac{\sin(\theta+\phi)}{\tan\theta}\right).$$

If the upper region (above the wedge) is part of the interior of the PSLG, the fan effect demonstrated in Figure 2 may necessitate the insertion of another vertex r between o and p (Figure 3, bottom); this circumstance is unavoidable if the product of the bounds on

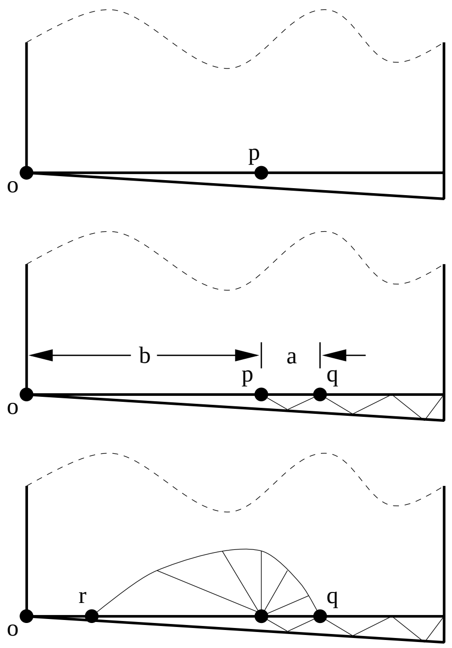

Figure 3: Top: A difficult PSLG with a small interior angle ϕ. Center: The vertex p and the angle constraint necessitate the insertion of the vertex q. Bottom: The vertex q and the angle constraint necessitate the insertion of the vertex r. The process repeats eternally.

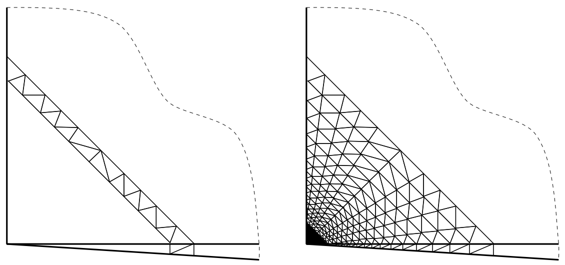

Figure 4: How to create a quality triangulation of infinite cardinality around the apex of a very small angle. The method employs a thin strip of well-shaped triangles about the vertex (left). Ever-smaller copies of the strip fill the gap between the vertex and the outer strip (right).

b/a and a/b given above is less than one. For an angle constraint of $\theta = 30°$, this condition occurs when ϕ is about six tenths of a degree. Unfortunately, the new vertex r creates the same conditions as the vertex p, but is closer to o; the process will cascade, eternally necessitating smaller and smaller triangles to satisfy the angle constraint. No algorithm can produce a finite triangulation of such a PSLG without violating the angle constraint. Delaunay refinement often fails in practice to achieve a $30°$ angle bound for $\phi = 5°$.

Oddly, it is straightforward to triangulate this PSLG using an infinite number of well-shaped triangles. A vertex at the apex of a small angle can be shielded with a thin strip of well-shaped triangles, as Figure 4 illustrates. (This idea is related to Ruppert's technique of using *shield edges* [4]. Ruppert mistakenly claims that the region concealed behind shield edges always has a finite good-quality triangulation.) The strip is narrow enough to admit

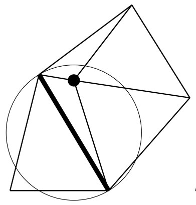

Figure 5: Segments are split recursively (while the constrained Delaunay property is maintained) until no subsegment is encroached.

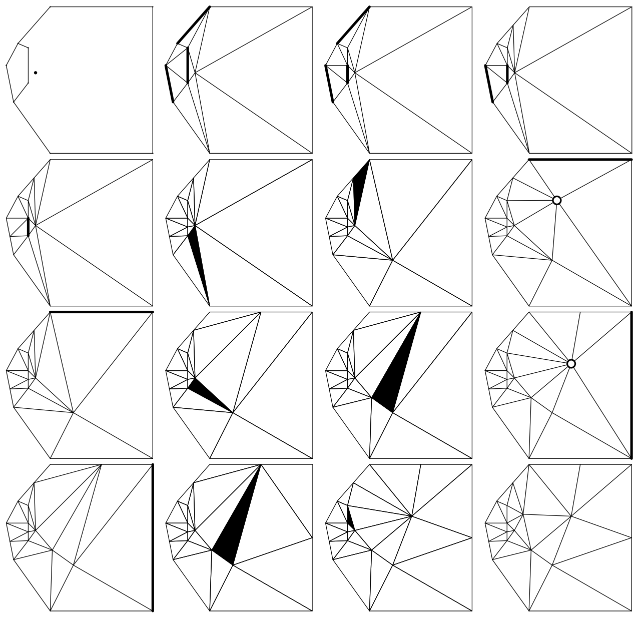

Figure 6: A complete run of Ruppert's algorithm with the angle bound 21.8°. The first two images are the input PSLG and the constrained Delaunay triangulation of its vertices. In each image, highlighted subsegments or triangles are about to be split, and open vertices are rejected because they encroach upon a subsegment.

a quality triangulation in the wedge that bears the smallest input angle. Its shape is chosen so that no acute angle appears outside the shield, and the region outside the shield can be triangulated by Delaunay refinement. The region inside the shield is triangulated by an infinite sequence of similar strips, with each successive strip smaller than the previous strip by a constant factor close to one.

3. Ruppert's Delaunay Refinement Algorithm

Jim Ruppert's and Paul Chew's algorithms for two-dimensional quality mesh generation [4, 1] are perhaps the first theoretically guaranteed meshing algorithms to be satisfactory in practice. In theory and practice, they produce meshes whose triangles are nicely shaped, are few in number, and have sizes that can grade from small to large over a short distance.

Ruppert's algorithm is presented here with a few modifications. The most significant change is that the algorithm here (like Chew's algorithm) begins with the constrained Delaunay triangulation [2], rather than an ordinary Delaunay triangulation, of the segment-bounded PSLG provided as input. The algorithm refines the triangulation by inserting additional vertices (while maintaining the constrained Delaunay property) until all triangles satisfy the quality constraint. Initially, each segment comprises one subsegment. Vertex insertion is governed by two rules.

- The *diametral circle* of a subsegment is the (unique) smallest circle that encloses the subsegment. A subsegment is said to be *encroached* if a vertex other than its endpoints lies on or inside its diametral circle, and the encroaching vertex is visible from the subsegment's interior. (Visibility is obstructed only by other segments.) Any encroached subsegment that arises is immediately split into two subsegments by inserting a vertex at its midpoint, as illustrated in Figure 5. These subsegments have smaller diametral circles, and may or may not be encroached themselves; splitting continues until no subsegment is encroached.
- A triangle is *skinny* if its smallest angle is less than some specified bound θ. Each skinny triangle is normally split by inserting a vertex at the triangle's *circumcenter*—the center of its *circumcircle*, which circumscribes the triangle—thus eliminating the triangle. However, if the new vertex would encroach upon any subsegment, then it is not inserted; instead, all the subsegments it would encroach upon are split.

Encroached subsegments are given priority over skinny triangles. The order in which subsegments are split, or skinny triangles are split, is arbitrary. Figure 6 illustrates the generation of a mesh by Ruppert's algorithm from start to finish.

4. Proof of Termination

Ruppert's algorithm can eliminate any skinny triangle by inserting a vertex, but new skinny triangles might take its place. How can we be sure the process will ever stop? The proof given here that Ruppert's algorithm terminates is related to Ruppert's own proof, but is rewritten in a more intuitive form that helps to clarify the extensions that handle small input angles. The quality of a triangle is measured by its *circumradius-to-shortest edge ratio*: the radius of its circumcircle divided by the length of its shortest edge. This metric is naturally improved by Delaunay refinement algorithms. A triangle's circumradius-to-shortest edge ratio r/ℓ is related to its smallest angle $\theta_{\min}$ by the formula $r/\ell = 1/(2\sin\theta_{\min})$. One would like the ratio to be as small as possible, so that the triangle's smallest angle is maximized.

Ruppert's algorithm takes a bound B as a parameter, and attempts to eliminate any triangle whose circumradius-to-shortest edge ratio exceeds B. Ruppert's algorithm is guaranteed to accomplish this goal and terminate if $B \geq \sqrt{2}$ and any two incident segments (segments that share an endpoint) in the input PSLG are separated by an angle of 90° or greater. (Removing the latter requirement is the main goal of this paper.) If $B = \sqrt{2}$, all angles of the final mesh range between 20.7° and 138.6°.

A *mesh vertex* is any vertex that has been successfully inserted into the mesh (including the input vertices). A *rejected vertex* is any vertex that is considered for insertion but rejected because it encroaches upon a subsegment. With each mesh vertex or rejected vertex v, associate an *insertion radius* r_v, equal to the length of the shortest edge connected to v immediately after v is introduced into the triangulation. (The insertion radius of a rejected vertex is defined as if it had been inserted, even though it is not actually inserted.)

Consider what this means in four different cases. If v is an input vertex, then r_v is the Euclidean distance between v and the near-

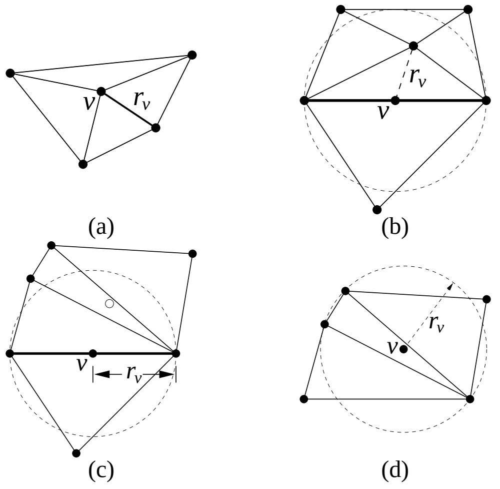

Figure 7: The insertion radius r_v of a vertex v.

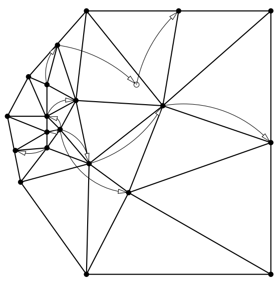

Figure 8: Trees of vertices for the example of Figure 6. Arrows are directed from parents to their children. Children include all inserted vertices and one rejected vertex.

est input vertex visible from v; see Figure 7(a). If v is a vertex inserted at the midpoint of an encroached subsegment, then r_v is the distance between v and the nearest encroaching mesh vertex; see Figure 7(b). If there is no encroaching mesh vertex (some triangle's circumcenter was considered for insertion but rejected as encroaching), then r_v is the radius of the diametral circle of the encroached subsegment; see Figure 7(c). If v is a vertex inserted at the circumcenter of a skinny triangle, then r_v is the circumradius of the triangle; see Figure 7(d).

Each vertex v, including any rejected vertex, has a *parent* vertex $p(v)$, unless v is an input vertex. Intuitively, $p(v)$ is the vertex that is "responsible" for the insertion of v. If v is a vertex inserted at the midpoint of an encroached subsegment, then $p(v)$ is the (nearest) encroaching vertex. (Note that $p(v)$ might be a rejected vertex; a parent need not be a mesh vertex.) If v is a vertex inserted (or rejected) at the circumcenter of a skinny triangle, then $p(v)$ is the most recently inserted endpoint of the shortest edge of that triangle.

Figure 8 illustrates the parents of all vertices inserted or considered for insertion during the sample execution of Ruppert's algorithm in Figure 6.

For any input vertex v, the *local feature size* $\text{lfs}(v)$ is the distance from v to the nearest input vertex or segment that is not incident to v. For any point p lying on an input segment s, $\text{lfs}(p)$ is the distance from p to the nearest input vertex or segment that is not incident to s. For a point p that does not intersect the input PSLG, leave

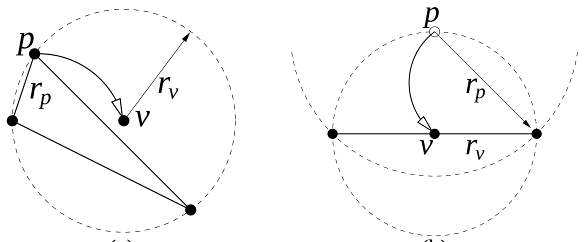

Figure 9: The relationship between the insertion radii of a child v and its parent p. (a) When a skinny triangle is split, $r_v \geq Br_p$. (b) When a subsegment is encroached upon by the circumcenter of a skinny triangle, $r_v \geq r_p/\sqrt{2}$.

$\text{lfs}(p)$ undefined. The key insight to why Ruppert's algorithm terminates is that no descendant of a mesh vertex has an insertion radius smaller than the minimum of the vertex's own insertion radius and (if it is defined) the descendent's local feature size. Therefore, no edge will ever appear that is shorter than the smallest feature of the input PSLG. To prove these facts, consider the relationship between the insertion radii of a vertex and its parent.

LEMMA 1. *Suppose the input PSLG has no acute angles. Let v be a vertex of the triangulation, and let $p = p(v)$ be its parent, if one exists. Then one of three possibilities holds: $r_v \geq Br_p$, if v is the circumcenter of a skinny triangle; $r_v \geq r_p/\sqrt{2}$, if v is the midpoint of an encroached subsegment and p is the (rejected) circumcenter of a skinny triangle; or $r_v \geq \text{lfs}(v)$, if the first two possibilities do not hold.*

Proof: If v is an input vertex, there is another input vertex a distance of r_v from v, so $\text{lfs}(v) \leq r_v$, and the theorem holds.

If v is inserted at the circumcenter of a skinny triangle, then its parent p is the most recently inserted endpoint of the shortest edge of the triangle; see Figure 9(a). Hence, the length of the shortest edge of the triangle is at least r_p. Because the triangle is skinny, its circumradius-to-shortest edge ratio is at least B, so its circumradius is $r_v \geq Br_p$.

If v is inserted at the midpoint of an encroached subsegment s, there are two cases to consider.

- If the parent p is an input vertex, or was inserted in a segment not incident to the segment containing s, then by definition, $\text{lfs}(v) \leq r_v$. (p cannot lie in a segment incident to the segment containing s, because incident segments are separated by angles of at least 90°.)
- If p is a circumcenter that was considered for insertion but rejected because it encroaches upon s, then p lies inside the diametral circle of s. Because the mesh is constrained Delaunay, one can show that the circumcircle centered at p contains neither endpoint of s. Hence, $r_v \geq r_p/\sqrt{2}$; see Figure 9(b) for an example where the relation is equality. ■

Lemma 1 limits how quickly the insertion radius can decrease as one traverses a sequence of descendants of a vertex. If vertices with ever-smaller insertion radii cannot be generated, then edges shorter than existing features cannot be introduced, and Delaunay refinement is guaranteed to terminate.

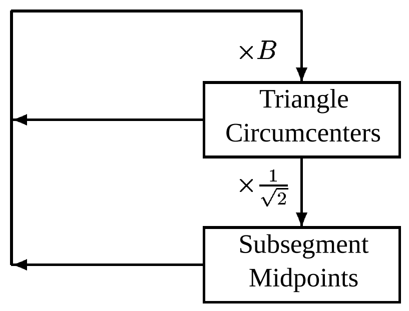

Figure 10: Dataflow graph illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycle has a product smaller than one, Ruppert's Delaunay refinement algorithm will terminate.

Figure 10 expresses this notion as a dataflow graph. Vertices are divided into three classes: input vertices (which are omitted from the figure because they cannot participate in cycles), *free vertices* inserted at circumcenters of triangles, and *segment vertices* inserted at midpoints of subsegments. Labeled arrows indicate how a vertex can cause the insertion of a child whose insertion radius is some factor times that of its parent. If the graph contains no cycle whose product is less than one, termination is guaranteed. This goal is achieved by choosing B to be at least $\sqrt{2}$. The following theorem formalizes these ideas.

THEOREM 2. *Let* $\text{lfs}_{\min}$ *be the shortest distance between two nonincident entities (vertices or segments) of the input PSLG. (Equivalently,* $\text{lfs}_{\min} = \min_u \text{lfs}(u)$*, where* u *is chosen from among the input vertices.)*

Suppose that any two incident segments are separated by an angle of at least $90°$*, and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is greater than* B*, where* $B \geq \sqrt{2}$*. Ruppert's algorithm will terminate, with no triangulation edge shorter than* $\text{lfs}_{\min}$.

Proof: Suppose for the sake of contradiction that the algorithm introduces an edge shorter than $\text{lfs}_{\min}$ into the mesh. Let e be the first such edge introduced. Clearly, the endpoints of e cannot both be input vertices, nor can they lie on nonincident segments. Let v be the most recently inserted endpoint of e.

By assumption, no edge shorter than $\text{lfs}_{\min}$ existed before v was inserted. Hence, for any ancestor a of v that is a mesh vertex, $r_a \geq \text{lfs}_{\min}$. Let $p = p(v)$ be the parent of v, and let $g = p(p)$ be the grandparent of v (if one exists). Consider the following cases.

- If v is the circumcenter of a skinny triangle, then by Lemma 1, $r_v \geq Br_p \geq \sqrt{2}r_p$.
- If v is the midpoint of an encroached subsegment and p is the circumcenter of a skinny triangle, then p has a parent g and by Lemma 1, $r_v \geq \frac{1}{\sqrt{2}}r_p \geq \frac{B}{\sqrt{2}}r_g \geq r_g$. (Recall that p is rejected.)

In both cases, $r_p \geq r_a$ for some ancestor a of p in the mesh. It follows that $r_p \geq \text{lfs}_{\min}$, contradicting the assumption that e has length less than $\text{lfs}_{\min}$. It also follows that no edge shorter than $\text{lfs}_{\min}$ is ever introduced, so the algorithm must terminate, because it will eventually run out of places to put new vertices. ■

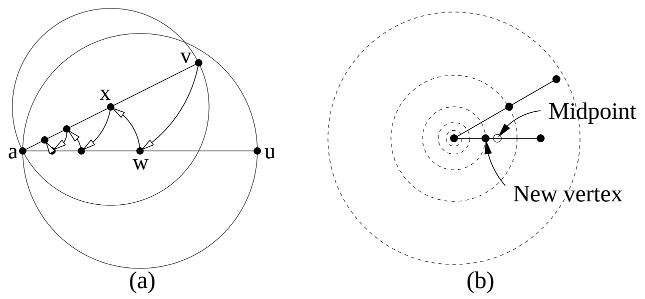

Figure 11: (a) A problem caused by a small input angle. Vertex v encroaches upon au, which is split at w. Vertex w encroaches upon av, which is split at x. Vertex x encroaches upon aw, and so on. (b) If an encroached subsegment has a shared input vertex for an endpoint, the subsegment is split at its intersection with a circular shell whose radius is a power of two.

Ruppert's algorithm terminates only when all triangles in the mesh have a circumradius-to-shortest edge ratio of B or smaller; hence, at termination, there is no angle smaller than $\arcsin \frac{1}{2B}$. If $B = \sqrt{2}$, the smallest value for which termination is guaranteed, no angle is smaller than $20.7°$.

5. Practical Handling of Small Input Angles

Ruppert's algorithm fails to terminate on PSLGs like that of Figure 3, even if the algorithm is modified so that it does not try to split any skinny triangle that bears a small input angle. As Section 2 demonstrates, any mesh of such a PSLG has a small angle that is removable, but another small angle invariably takes its place. A practical mesh generator should always terminate, even if it must leave small angles behind. How can one detect this circumstance, and ensure termination of the algorithm while still generating good-quality triangles wherever possible?

Figure 11(a) demonstrates one of the difficulties caused by small input angles. If two incident segments have unmatched lengths, an endless cycle of mutual encroachment may produce ever-smaller subsegments incident to the apex of the small angle. This phenomenon is only observed with angles smaller than $45°$. In effect, acute angles add a new cycle to Figure 10, whereby subsegment midpoints can sire other subsegment midpoints.

To solve this problem, Ruppert [4] suggests "modified segment splitting using concentric circular shells." Imagine that each input vertex is encircled by concentric circles whose radii are all the powers of two (that is, 2^i for all integers i), as illustrated in Figure 11(b). When an encroached subsegment has an endpoint that is an input vertex shared with another segment, the subsegment is split not at its midpoint, but at one of the circular shells, so that one of resulting subsegments has a power-of-two length. (Clearly, the final mesh will depend on the unit of measurement chosen to represent lengths.) If both endpoints are shared input vertices, choose one endpoint's shells arbitrarily. The shell that gives the best-balanced split is chosen; in the worst case, the smaller resulting subsegment is one-third the length of the split subsegment. Each input segment may undergo up to two unbalanced splits—one for each end. All other subsegment splits are bisections.

Concentric shell segment splitting prevents the runaway cycle of ever-smaller subsegments portrayed in Figure 11(a), because incident subsegments of equal length do not encroach upon each other.

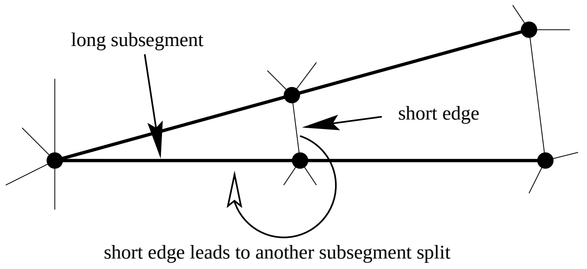

Figure 12: The short edge opposite a small angle can cause other short edges to be created as the algorithm attempts to remove skinny triangles. If the small insertion radii propagate around an endpoint and cause the supporting subsegments to be split, a shorter edge is created, and the cycle may repeat.

It is generally effective in practice for PSLGs that have small angles greater than 10°, and often for smaller angles. It is always effective for polygons (with holes). As Section 2 hints, difficulties are only likely to occur when a small angle is adjacent to a much larger angle. The negative result of Section 2 arises not because of the mutual encroachment problem illustrated in Figure 11(a), but because the free edge opposite a small angle is shorter than the subsegments that define the angle, as Figure 12 illustrates.

The two subsegments of Figure 12 are coupled, in the sense that if one is bisected then so is the other, because the midpoint of one encroaches upon the other. This holds true for any two segments of equal length separated by less than 60°. Each time such a dual bisection occurs, a new edge is created that is shorter than the subsegments produced by the bisection; the free edge can be arbitrarily short if the angle is arbitrarily small. One of the endpoints of the free edge has a small insertion radius, though that endpoint's parent (usually the other endpoint) might have a large insertion radius. Hence, a small angle acts as an "insertion radius reducer." The new short edge will engender other short edges as the algorithm attempts to remove skinny triangles. If small insertion radii propagate around an endpoint of the short edge, the incident subsegments may be split again, commencing an infinite sequence of shorter and shorter edges. This cycle does not occur if the PSLG is a simple polygon with polygonal holes.

For general PSLGs, how may one diagnose and cure diminishing cycles of edges? A sure-fire way to guarantee termination is to never insert a vertex whose insertion radius is smaller than the insertion radius of its most recently inserted ancestor (its parent if the parent was inserted; its grandparent if the parent was rejected), unless its parent is an input vertex or lies on a nonincident segment.

A proof that this cure works is best understood by modifying the dataflow graph of Figure 10. Acute angles are a problem because they allow subsegment midpoints to have other subsegment midpoints for children, thereby adding a new cycle to the dataflow graph. So long as the cycle's multiplier is not smaller than one, Ruppert's algorithm will still terminate. Cycles of diminishing insertion radii can be prevented as follows: if a subsegment is encroached upon by a subsegment midpoint, and the midpoint of the former would have a smaller insertion radius than the encroaching midpoint, do not split the encroached subsegment.

This restriction is undesirably conservative for two reasons. First, if a Delaunay triangulation is desired, the restriction might prevent

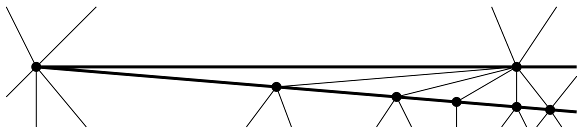

Figure 13: The simplest method of ensuring termination when small input angles are present has undesirable properties, including the production of large angles and many small angles.

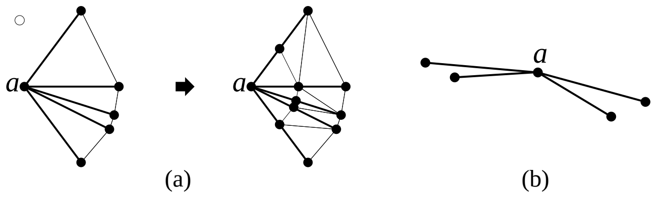

Figure 14: (a) Example of a subsegment cluster. If all the subsegments of a cluster have power-of-two lengths, then they all have the same length and are effectively split as a unit because of mutual encroachment. (b) Several independent subsegment clusters may share the same apex.

us from obtaining one, because segments may remain encroached. A second, more serious problem is demonstrated in Figure 13. Two subsegments are separated by a small input angle, and one of the two is bisected. The other subsegment is encroached, but is not bisected because its midpoint would have a small insertion radius. One unfortunate result is that the triangle bearing the small input angle also has a large angle of almost 180°. Large angles can be worse than small angles, because they jeopardize interpolation accuracy and convergence of the finite element method in ways that small angles do not. Another unfortunate result is that many skinny triangles may form. The triangles in the figure cannot be improved without splitting the upper subsegment.

6. A Better Algorithm

As an alternative, I suggest a Delaunay refinement algorithm called *the Terminator*, which is also guaranteed to terminate but produces fewer large angles in practice, and has a guaranteed minimum angle in theory. The Terminator is based on Delaunay refinement with concentric circular shells. When a subsegment s is encroached upon by the circumcenter of a skinny triangle, a decision is made whether to split s with a vertex v, or to leave s whole. (In either case, the circumcenter is rejected.) The decision process is somewhat elaborate.

If neither endpoint of s bears an input angle less than 60°, or if both endpoints do, then s is split. (If there is no input angle smaller than 60°, the Terminator acts no differently than Ruppert's algorithm.) Otherwise, let a be the apex of the small angle. Define the *subsegment cluster* of s to be the set of subsegments incident to a that are separated from s, or from some other member of the subsegment cluster of s, by less than 60°. Once all the subsegments of a cluster have been split to power-of-two lengths, they must all be the same length to avoid encroaching upon each other. If one is bisected, the others follow suit, as illustrated in Figure 14(a).

Not all subsegments incident to an input vertex are part of the same cluster. For instance, Figure 14(b) shows two independent subsegment clusters sharing one apex, separated from each other by angles of at least 60°.

If the subsegments in the subsegment cluster of s do not all have the same length, then s is split. Otherwise, to decide whether s should be split with a vertex v, the Terminator determines the insertion radius r_g of v's grandparent g (which is the parent of the encroaching circumcenter), and the minimum insertion radius $r_{\min}$ of all the midpoint vertices (including v) that will be introduced into the subsegment cluster of s if all the subsegments in the cluster are split. Because all the subsegments in the cluster have the same length, $r_{\min}$ depends upon the smallest angle in the subsegment cluster.

The vertex v is inserted, splitting s, if and only if one or more of the following three conditions hold.

- $r_{\min} \geq r_g$;
- the subsegments in the subsegment cluster of s are not of the same length as s (or s has two subsegment clusters, one at each endpoint); or
- no ancestor of v lies in the relative interior of the segment containing s. (It is not relevant whether an endpoint of the segment is an ancestor of v.)

The first condition permits subsegment splits that do not engender smaller edges than the existing edges. The second condition helps to create isosceles triangles (which are the best we can hope for) at small input angles. The third condition, which is optional, attempts to distinguish between the case where a subsegment is encroached because of small input features, and the case where a subsegment is encroached because it bears a small input angle.

If $r_{\min} < r_g$, but v is inserted in s anyway because one of the other two conditions hold, v is called a *trigger vertex*, because it (usually) triggers the splitting of all the subsegments in a cluster and thereby creates a new edge whose length is less than r_g. A *Type A trigger vertex* is inserted because the subsegments are not all of the same length (or a subsegment takes part in two clusters), and a *Type B trigger vertex* is inserted because it has no ancestor in the relative interior of the same segment. Recall that the parent of any trigger vertex is a rejected circumcenter of a skinny triangle, and skinny triangles have lower priority than encroached subsegments, so a trigger vertex is only generated when no subsegment in the mesh is encroached upon by a mesh vertex.

THEOREM 3. *The Terminator always terminates.*

Proof: If two incident subsegments separated by less than $60°$ have lengths that differ by a factor of two or more, the endpoint of the shorter one must encroach upon the longer one, so the longer one is split before a trigger vertex can be generated. Hence, a Type A trigger vertex is inserted into a cluster only if some subsegment in that cluster has a length not of the form 2^i for some integer i. Therefore, the Terminator inserts at most two Type A trigger vertices per input segment—one for each end of each segment. (Note that the subsegment in which a trigger vertex is inserted is not necessarily the subsegment whose length is not a power of two.)

No Type B trigger vertex is ever inserted in the relative interior of the same segment as any of its ancestors, so each input segment may contain at most one Type B trigger vertex for each input vertex. It follows that the number of trigger vertices the Terminator inserts is finite.

Say that a vertex v is a *diminishing vertex* if its insertion radius r_v is less than that of all its ancestors. It is a consequence of Lemma 1 that if v is a diminishing vertex, v must have been inserted in a subsegment s in one of the following circumstances.

- v is not the midpoint of s, because s was split using concentric circular shells;
- s is encroached upon by an input vertex, or by a vertex lying on a segment not incident to the segment that contains s; or
- s is encroached upon by a vertex p that lies on a segment incident to s at an angle less than $60°$.

Only a finite number of diminishing vertices can be inserted in these circumstances. The first circumstance can occur only twice for each input segment (after which the subsegment at either end has a power-of-two length). Only a finite number of vertex insertions of the second type are possible as well, because any subsegment shorter than $\mathrm{lfs}_{\min}$ cannot be encroached upon by a nonincident feature.

Only a finite number of diminishing vertices can be inserted in the third circumstance, too. Why? Segment splitting with concentric shells prevents a cluster of incident subsegments from engaging in a chain reaction of ever-diminishing mutual encroachment. Specifically, let 2^i be the largest power of two (with i an integer) less than or equal to the length of the shortest subsegment in the cluster. No subsegment of the cluster can be split to a length less than 2^{i-1} through the mechanism of encroachment alone. The subsegments will only be split to shorter lengths if one of them is split by a trigger vertex, or if one of them is encroached in the second circumstance listed above. However, the Terminator inserts only a finite number of trigger vertices and only a finite number of vertices under the second circumstance.

It follows that only a finite number of diminishing vertices are inserted. No edge ever appears whose length is less than the minimum insertion radius of all the diminishing vertices and input vertices. Because there is a lower bound on edge lengths, the Terminator terminates; it must eventually run out of places to put new vertices. ■

THEOREM 4. *Let ϕ be the smallest angle of a PSLG, with $\phi \leq 60°$. The Terminator produces a mesh of the PSLG wherein every triangle has a circumradius-to-shortest edge ratio no greater than $1/[\sqrt{2}\sin(\phi/2)]$. (Note that for $\phi = 60°$, this bound is $\sqrt{2}$, matching the bound for PSLGs free of acute angles.) Hence, no angle of the final mesh is less than $\arcsin[\sin(\phi/2)/\sqrt{2}]$.*

Proof: Let t be a skinny triangle that is not eliminated by the Terminator, as illustrated in Figure 15. t is not split because its circumcenter p encroaches upon a subsegment s whose splitting would have split a subsegment cluster, yielding a small edge of length $r_{\min}$ opposite the apex of the cluster. Let v be the midpoint of s; p would be the parent of v if v had been inserted. Let g be the parent of p (and the grandparent of v). Let d be the length of t's shortest edge. Recall that r_p is the circumradius of t, so r_p/d is t's circumradius-to-shortest edge ratio.

Because g is the most recently inserted endpoint of t's shortest edge (by the definition of "parent"), $r_g \leq d$. Because the Terminator chose not to insert v, $r_{\min} < r_g$. By Lemma 1, $r_p \leq \sqrt{2}r_v$. If the subsegment cluster of s had been split, every resulting subsegment would have length r_v. If the smallest angle of the subseg-

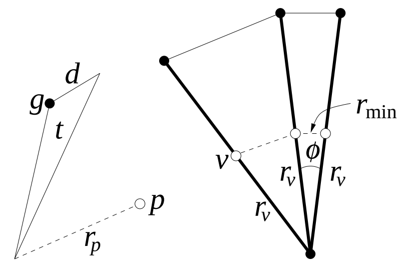

Figure 15: If the Terminator does not split t, the smallest angle of t is not much smaller than ϕ.

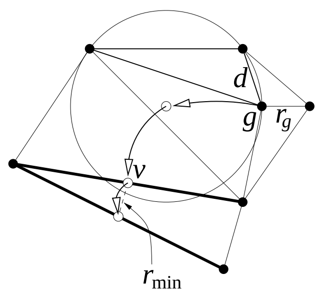

Figure 16: The length d of the shortest edge of a skinny triangle is an upper bound on the insertion radius r_g of the most recently inserted endpoint of that edge.

ment cluster of s is ϕ, basic trigonometry shows that $\sin(\phi/2) = r_{\min}/(2r_v)$. Putting these inequalities together, the circumradius-to-shortest edge ratio of t is $r_p/d < 1/[\sqrt{2}\sin(\phi/2)]$. ■

The Terminator eliminates all encroached subsegments, so there is no danger that the final mesh will not be Delaunay. Because subsegments are not encroached, an angle near $180°$ cannot appear immediately opposite a subsegment (as in Figure 13), although large angles can appear near subsegment clusters. Skinny triangles in the final mesh occur only near input angles less than (in practice, *much* less than) $60°$.

7. A Reduced-Memory Variant

The Terminator has the unfortunate characteristic that it demands more memory than Ruppert's algorithm, because each mesh vertex must store its insertion radius and a pointer to its parent (or, if its parent was rejected, its grandparent). Here I suggest possible modifications that avoid these requirements, although they may lessen the quality of the final mesh somewhat.

The Terminator needs to know the insertion radius of a vertex only when it considers inserting a vertex v into a subsegment cluster. It is straightforward to compute the insertion radii of v and the other vertices that will be inserted into the cluster. However, the insertion radius of v's grandparent g cannot be determined by inspecting the current mesh, because other vertices may have been inserted near g since g was inserted. Nevertheless, a good substitute for r_g is the length d of the shortest edge of the skinny triangle whose circumcenter is v's parent, illustrated in Figure 16. The length d is an upper bound on r_g, so its use will not jeopardize the Terminator's termination guarantee; the modified algorithm is strictly more conservative in its decision about whether to insert v. The Terminator's minimum angle bound still holds as well. With this modification, there is no need to store the insertion radius of each vertex.

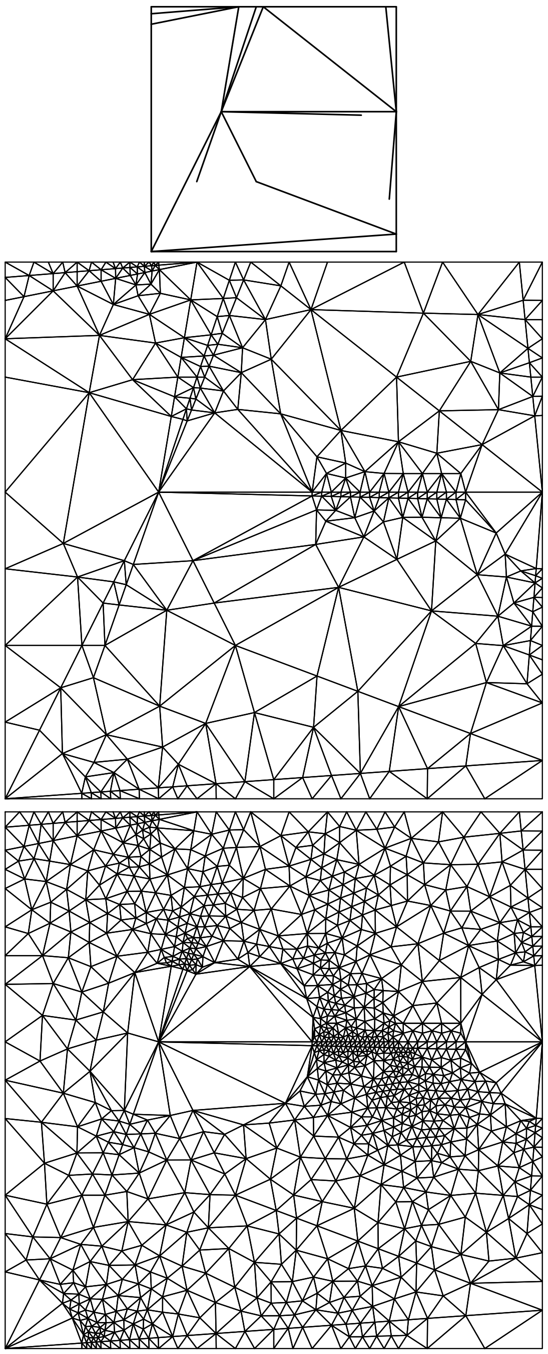

Figure 17: Two meshes of a PSLG (top) produced by the simplified Terminator. The upper mesh was produced with an angle bound of $20.7°$. The lower mesh used an angle bound of $33°$. Angles smaller than these occur only in triangles whose circumcenters would encroach upon a subsegment cluster.

The only apparent way to avoid storing a pointer from each vertex to its nearest inserted ancestor is to eliminate Type B trigger vertices entirely. The possible disadvantage is that a small input feature might fail to cause a nearby subsegment to be split even though it ought to have the privilege, and thus skinny triangles will unnecessarily remain in the mesh. Theorem 4 still holds, though.

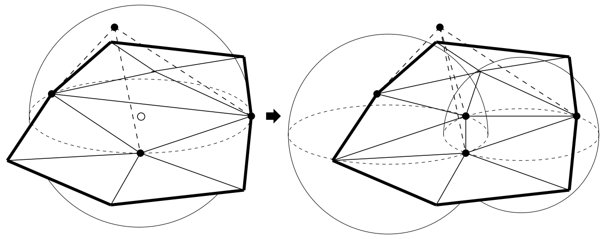

Figure 18: Splitting an encroached subfacet. The triangular faces shown are subfacets of a larger facet, with tetrahedra (not shown) atop them. In this example, all equatorial spheres (included the two illustrated) are empty after the split.

Figure 17 depicts a PSLG with nine small angles ranging from $1.43°$ to $8.97°$, and two meshes thereof produced by the Terminator with both the simplifications described above. The upper mesh was produced with an angle bound of $20.7°$—the greatest bound for which termination is guaranteed—and the lower mesh with an angle bound of $33°$. The upper mesh has 11 new angles less than $20.7°$, of which three are less than $10°$ and the smallest is $5.83°$. The upper mesh has 101 new angles less than $33°$, of which nine are less than $10°$ and the smallest is $7.02°$. The original version of Ruppert's algorithm fails with angle bounds larger than about $12°$ (and terminates for a $12°$ angle bound only if it uses concentric circular shells to split subsegments.)

8. Three-Dimensional Delaunay Refinement

Delaunay refinement has been generalized to three dimensions [7], albeit with complications. The biggest difficulty is that not every PLC has a constrained Delaunay tetrahedralization (CDT). Indeed, there are simple polyhedra for which no tetrahedralization of any sort exists [5]. As a result, small input angles are a more challenging foe in three dimensions than in two. An approach to meshing difficult PLCs is outlined here; space precludes a complete treatment.

Because an input PLC X may have no CDT, Delaunay refinement begins with the ordinary Delaunay tetrahedralization of the vertices of X. Some segments and facets of X may be missing from the tetrahedralization, in which case their recovery is aided by the insertion of additional vertices (while maintaining the Delaunay property). Vertex insertion is governed by three rules.

- A subsegment is encroached if a vertex other than its endpoints lies inside or on its diametral sphere. A subsegment may be encroached whether or not it actually appears as an edge of the tetrahedralization. As in Ruppert's algorithm, any encroached subsegment that arises is immediately split.
- The *equatorial sphere* of a triangular subfacet is the (unique) smallest sphere that passes through the three vertices of the subfacet. A subfacet is encroached if a non-coplanar vertex lies inside or on its equatorial sphere. Each encroached subfacet is normally split by inserting a vertex at its circumcenter; see Figure 18. However, if the new vertex would encroach upon any subsegment, it is not inserted; instead, all the subsegments it would encroach upon are split.
- A tetrahedron is *skinny* if its circumradius-to-shortest edge ratio is greater than some bound B. Each skinny tetrahedron is normally split by inserting a vertex at its circumcenter. However, if the new vertex would encroach upon any subsegment or subfacet, then it is not inserted; instead, all the subsegments it would encroach upon are split. If the skinny tetrahedron is not eliminated as a result, then all the subfacets its circumcenter would encroach upon are split.

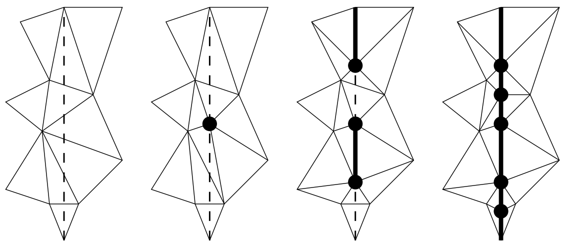

Figure 19: Missing segments are recovered through the same recursive splitting procedure used for encroached subsegments that aren't missing. In this sequence of illustrations, the dashed line represents a segment missing from the triangulation. Segment recovery is illustrated here in two dimensions for clarity, but operates no differently in three.

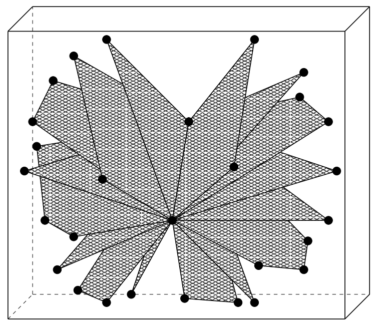

Figure 20: It is difficult to mesh the interior of this box with Delaunay tetrahedra that conform to all the facets.

Encroached subsegments are given priority over encroached subfacets, which have priority over skinny tetrahedra. It is a property of Delaunay tetrahedralizations that if a subsegment is missing from the tetrahedralization, then it is encroached. Hence, the first rule above, in conjunction with concentric shell segment splitting, is sufficient to recover all the missing segments in the manner illustrated in Figure 19.

Once every missing segment is represented by a contiguous linear sequence of edges of the tetrahedralization, the missing facets can be recovered by a combination of the first two rules under suitable conditions, as documented elsewhere [7]. However, if segments or facets of X are separated by angles of less than $90°$, the conditions might be quite unsuitable. Consider, for instance, the box illustrated in Figure 20. In the interior of the box, many oddly shaped facets are arrayed about a single common segment. The attempt to recover one facet—so that it is represented as a union of triangular faces of the Delaunay tetrahedralization—is likely to knock out subfacets of the adjacent facets.

Fortunately, if there are no encroached or missing subsegments in the tetrahedralization, it is possible to recover all of the missing facets by constructing a CDT (without inserting any additional vertices). Specifically, let Y be the modified PLC obtained by adding to X all of the vertices that were inserted into encroached subsegments. It can be shown that if Y contains no subsegment en-

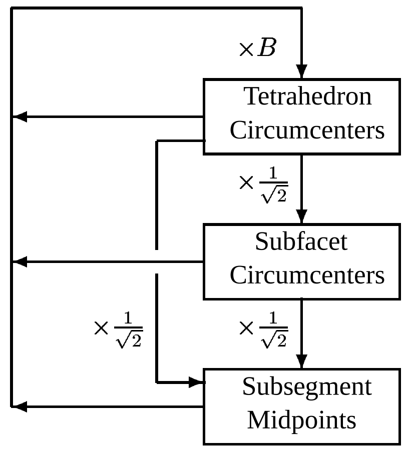
$\times B$
Tetrahedron Circumcenters
$\times \frac{1}{\sqrt{2}}$
Subfacet Circumcenters
$\times \frac{1}{\sqrt{2}}$
$\times \frac{1}{\sqrt{2}}$
Subsegment Midpoints

Figure 21: Dataflow graph illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets.

croached upon by vertices of Y, then a CDT of Y exists [6]. Hence, the algorithm proposed here always uses CDTs to recover missing facets. Some subfacets may remain encroached in the final mesh, however.

After all missing segments and facets have been recovered, the third rule comes into play, operating in conjunction with the first two. The constrained Delaunay property of the tetrahedralization is maintained throughout all the remaining vertex insertions. Under the right conditions—for instance, if all input angles are at least 90°, and all facets are convex—Theorem 2 can be extended to cover three-dimensional Delaunay refinement [7]. Details are omitted here, but the important relations appear in Figure 21. The dataflow graph reveals that termination is guaranteed if B is chosen to be greater than or equal to two.

If a vertex lying in a segment or facet F encroaches upon a subsegment or subfacet lying in a segment or facet F', where F and F' are incident, then additional cycles are introduced into Figure 21. How may one prevent diminishing cycles of ever-smaller edge lengths? For two-dimensional meshing, I proposed in Section 5 that the algorithm never insert a vertex whose insertion radius is smaller than the insertion radius of its most recently inserted ancestor, unless its parent is an input vertex or lies on a nonincident segment. Unfortunately, this solution doesn't always work in three dimensions, because every subsegment encroached upon by a mesh vertex *must* be split to guarantee the existence of a CDT.

Specifically, suppose that s and s' are two subsegments. If s is split, its midpoint v might encroach upon s', with the result that a CDT might not exist after v is inserted. So that the algorithm can maintain a CDT, s and s' should be split simultaneously. However, v and the midpoint v' of s' might each encroach upon other subsegments as well, and the ensuing chain reaction may necessitate the simultaneous insertion of many vertices. Hence, whereas subsegment clusters bring an aesthetic improvement to the two-dimensional algorithm, their use is mandatory in the three-dimensional algorithm. Note that a single chain reaction may involve more than one subsegment cluster.

Hence, the modified algorithm is as follows.

- If a subfacet f of a facet F is encroached, and f's circumcenter v will not encroach upon any subsegment, then f is split only if v's insertion radius is at least as large as the insertion radius of its most recently inserted ancestor, or if v's parent is an input vertex or lies in a segment or facet not incident to F. If v encroaches upon some subsegment, then v is rejected, and the subsegment may or may not be split according to the following rule.
- If a subsegment s is encroached, and the encroaching vertex p is a mesh vertex, then s must be split to ensure that a CDT exists. However, if p is a tetrahedron circumcenter or subfacet circumcenter (and is thus rejected), then determine the set of all subsegments that must be split if s is split by a vertex v, and determine the set of all new vertices that will be inserted into those subsegments. If the minimum insertion radius of all the new vertices is at least as large as the insertion radius of v's most recently inserted ancestor, split s. Otherwise, don't—but be forewarned that some skinny tetrahedra might remain in the mesh.

This modified algorithm is guaranteed to terminate. Upon termination, tetrahedra whose circumradius-to-shortest edge ratios are greater (worse) than B appear only near input angles less than 90°. Some subfacets may remain encroached, so the final tetrahedralization is not guaranteed to be Delaunay, although it is constrained Delaunay. It is an open question whether the algorithm can be modified to produce Delaunay meshes for all imaginable PLCs. More importantly, it is an open problem to generalize to three dimensions the aesthetic improvements offered by the Terminator in two dimensions.

9. References

[1] L. Paul Chew. *Guaranteed-Quality Mesh Generation for Curved Surfaces*. Proceedings of the Ninth Annual Symposium on Computational Geometry (San Diego, California), pages 274–280. Association for Computing Machinery, May 1993.

[2] D.-T. Lee and A. K. Lin. *Generalized Delaunay Triangulations for Planar Graphs*. Discrete & Computational Geometry **1**:201–217, 1986.

[3] Scott A. Mitchell. *Cardinality Bounds for Triangulations with Bounded Minimum Angle*. Proceedings of the Sixth Canadian Conference on Computational Geometry (Saskatoon, Saskatchewan, Canada), pages 326–331, August 1994.

[4] Jim Ruppert. *A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation*. Journal of Algorithms **18**(3):548–585, May 1995.

[5] E. Schönhardt. *Über die Zerlegung von Dreieckspolyedern in Tetraeder*. Mathematische Annalen **98**:309–312, 1928.

[6] Jonathan Richard Shewchuk. *A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 76–85. Association for Computing Machinery, June 1998.

[7] ______. *Tetrahedral Mesh Generation by Delaunay Refinement*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 86–95. Association for Computing Machinery, June 1998.