Delaunay Refinement Algorithms for Triangular Mesh Generation

Jonathan Richard Shewchuk
jrs@cs.berkeley.edu

May 21, 2001

Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720

Supported in part by the National Science Foundation under Awards ACI-9875170, CMS-9980063, CMS-9318163, and EIA-9802069, in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF under agreement number F30602-96-1-0287, in part by the Natural Sciences and Engineering Research Council of Canada under a 1967 Science and Engineering Scholarship, and in part by gifts from the Okawa Foundation and Intel. The views and conclusions contained in this document are those of the author. They are not endorsed by, and do not necessarily reflect the position or policies of, the U. S. Government or other sponsors.

Abstract

Delaunay refinement is a technique for generating unstructured meshes of triangles for use in interpolation, the finite element method, and the finite volume method. In theory and practice, meshes produced by Delaunay refinement satisfy guaranteed bounds on angles, edge lengths, the number of triangles, and the grading of triangles from small to large sizes. This article presents an intuitive framework for analyzing Delaunay refinement algorithms that unifies the pioneering mesh generation algorithms of L. Paul Chew and Jim Ruppert, improves the algorithms in several minor ways, and most importantly, helps to solve the difficult problem of meshing nonmanifold domains with small angles.

Although small angles inherent in the input geometry cannot be removed, one would like to triangulate a domain without creating any *new* small angles. Unfortunately, this problem is not always soluble. A compromise is necessary. A Delaunay refinement algorithm is presented that can create a mesh in which most angles are 30° or greater and no angle is smaller than $\arcsin[(\sqrt{3}/2)\sin(\phi/2)] \sim (\sqrt{3}/4)\phi$, where $\phi \leq 60^\circ$ is the smallest angle separating two segments of the input domain. New angles smaller than 30° appear only near input angles smaller than 60°. In practice, the algorithm's performance is better than these bounds suggest.

Another new result is that Ruppert's analysis technique can be used to reanalyze one of Chew's algorithms. Chew proved that his algorithm produces no angle smaller than 30° (barring small input angles), but without any guarantees on grading or number of triangles. He conjectures that his algorithm offers such guarantees. His conjecture is conditionally confirmed here: if the angle bound is relaxed to less than 26.5°, Chew's algorithm produces meshes (of domains without small input angles) that are nicely graded and size-optimal.

Keywords: triangular mesh generation, Delaunay triangulation, constrained Delaunay triangulation, Delaunay refinement, computational geometry

Contents

1 Introduction

Delaunay refinement is a technique for generating triangular meshes suitable for use in interpolation, the finite element method, and the finite volume method. The problem is to find a triangulation that covers a specified domain, and contains only triangles whose shapes and sizes satisfy constraints: the angles should not be too small or too large, and the triangles should not be much smaller than necessary, nor larger than desired. Delaunay refinement algorithms offer mathematical guarantees that such constraints can be met. They also perform excellently in practice.

This article has three purposes. First, it offers a theoretical framework for Delaunay refinement algorithms that makes it easy to understand why different variations of Delaunay refinement are successful. This framework is used to clarify the performance of an algorithm by Ruppert, to reanalyze an algorithm by Chew, and to generate several extensions of Delaunay refinement. Second, this article exploits the framework to help find a practical solution to the difficult problem of meshing domains with small angles that Delaunay refinement algorithms proposed to date cannot mesh. Third, it presents almost everything algorithmic a programmer needs to know to implement a state-of-the-art triangular mesh generator for straight-line domains. (Curved boundaries and surfaces, however, are not treated here. Thorough treatments of data structures and Delaunay triangulation algorithms are available elsewhere [17, 29, 7].)

A full description of the mesh generation problem begins with the domain to be meshed. Most theoretical treatments of meshing take as their input a *planar straight line graph* (PSLG). A PSLG is a set of vertices and segments, like that illustrated in Figure 1(a). A segment is an edge that must be represented by a sequence of contiguous edges in the final mesh, as Figure 1(b) shows. By definition, 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. (A set of segments that does not satisfy this condition can be converted into a set of segments that does. Run a segment intersection algorithm [11, 28, 12], then divide each segment into smaller segments at the points where it intersects other segments or vertices.)

The process of mesh generation necessarily divides each segment into smaller edges called *subsegments*. The bold edges in Figure 1(b) are subsegments; other edges are not. The *triangulation domain* is the region that a user wishes to triangulate. For 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*. A triangulation domain need not be convex, and it may enclose untriangulated holes, but the holes must also be bounded by segments. A segment must lie anywhere a triangulated region of the plane meets an untriangulated region.

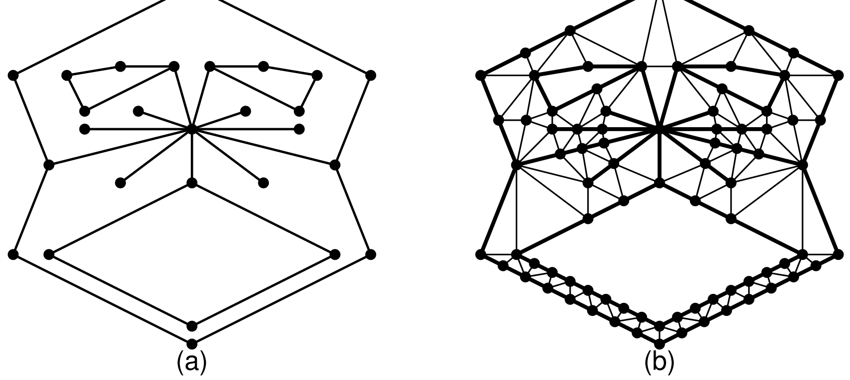


Figure 1: A PSLG and a mesh generated by Ruppert's Delaunay refinement algorithm.

A mesh generator produces a triangulation that attempts to satisfy three goals. First, the union of the triangles is the triangulation domain, and the triangulation *respects* the segments—each segment is a union of triangulation edges.

Second, the triangles should be relatively “round” in shape, because triangles with large or small angles can degrade the quality of the numerical solution to a finite element problem. In interpolation, triangles with large angles can cause large errors in the gradients of the interpolated surface. In the finite element method, large angles can cause a large *discretization error* [1]; the solution may be less accurate than the method would normally promise. Small angles can cause the coupled systems of algebraic equations that the finite element method yields to be ill-conditioned [6].

A lower bound on the smallest angle of a triangulation implicitly bounds the largest angle. If no angle is smaller than θ, no angle is larger than $180^\circ - 2\theta$. Hence, many mesh generation algorithms, including the Delaunay refinement algorithms studied here, take the approach of attempting to bound the smallest angle.

A third goal is to offer as much control as possible over the sizes of triangles in the mesh. Some meshing algorithms, including algorithms by Baker, Grosse, and Rafferty [2] and Chew [8], produce only *uniform meshes*, in which all triangles have roughly the same size. Other algorithms offer rapid *grading*—the ability to grade from small to large triangles over a relatively short distance. Small, densely packed triangles offer more accuracy than larger, sparsely packed triangles; but the computation time required to solve a problem is proportional to the number of triangles. Hence, choosing a triangle size entails trading off speed and accuracy. In the finite element method, the triangle size required to attain a given amount of accuracy depends upon the behavior of the physical phenomena being modeled, and may vary throughout the problem domain.

Given a *coarse* mesh—one with relatively few triangles—it is not difficult to *refine* it to produce another mesh having a larger number of smaller triangles [18]. The reverse process is not so easy [22]. Hence, mesh generation algorithms often set themselves the goal of being able, in principle, to generate a mesh with as few triangles as possible. They typically offer users the option to refine triangles that are not small enough to yield the required accuracy. The shape of a domain and the requirement for good-quality (round) triangles may necessitate the use of smaller triangles than desired in some portions of the mesh. The three goals are sometimes at odds with each other.

Delaunay refinement algorithms operate by maintaining a Delaunay or constrained Delaunay triangulation, which is refined by inserting carefully placed vertices until the mesh meets constraints on triangle quality and size. This article assumes that the reader is familiar with constrained Delaunay triangulations; consult Lee and Lin [20] and Chew [7] for treatments. Here, the most important property of a Delaunay triangulation is that it has the *empty circumcircle property*. The *circumcircle* of a triangle is the unique triangle that passes through its three vertices. The Delaunay triangulation of a set of vertices is the triangulation (usually, but not always, unique) in which every triangle has an *empty* circumcircle—meaning that the circle encloses no vertex of the triangulation.

A constrained Delaunay triangulation is similar, but respects the input segments as well as the vertices. Two points p and q are *visible* to each other if the open line segment pq (leaving out p and q) does not intersect any segment of the input PSLG. The constrained Delaunay triangulation of a PSLG has two properties. First, no segment intersects the interior of a triangle, because the triangulation must respect the segments. Second, each triangle's circumcircle encloses no vertex that is visible from the interior of the triangle, though it may enclose vertices hidden behind segments.

A few applications, such as some finite volume methods, may have an extra requirement: the mesh must be truly Delaunay (not just constrained Delaunay), and the center of each triangle's circumcircle must lie within the triangulation. This requirement arises because the Voronoi diagram, found by dualizing the Delaunay triangulation, must intersect "nicely" with the triangulation. Only a minority of applications have this requirement, but it is easily met by some of the mesh generation algorithms herein.

Delaunay refinement algorithms are successful because they exploit several favorable characteristics of Delaunay triangulations. One such characteristic is a result by Lawson [19] that a Delaunay triangulation maximizes the minimum angle among all possible triangulations of a point set. This result extends to the constrained Delaunay triangulation, which is optimal among all possible triangulations of a PSLG [20]. Another feature is that inserting a vertex is a local operation, and hence is inexpensive except in unusual cases. The act of inserting a vertex to improve poor-quality triangles in one part of a mesh will not unnecessarily perturb a distant part of the mesh that has no bad triangles. Furthermore, Delaunay triangulations have been extensively studied, and good algorithms for their construction are available [21, 17, 15, 7].

The greatest advantage of Delaunay triangulations is less obvious. The central question of any Delaunay refinement algorithm is, "Where should the next vertex be inserted?" As Section 2 will demonstrate, a reasonable answer is, "As far from other vertices as possible." If a new vertex is inserted too close to another vertex, the resulting small edge will engender thin triangles.

Because a Delaunay triangle has no vertices in its circumcircle, a Delaunay triangulation is an ideal search structure for finding points that are far from other vertices. (It's no coincidence that the circumcenter of each triangle of a Delaunay triangulation is a vertex of the corresponding Voronoi diagram.)

The first provably good Delaunay refinement algorithms were introduced by L. Paul Chew and Jim Ruppert. Ruppert [27] proves that his algorithm produces nicely graded, *size-optimal* meshes with no angle smaller than about 20.7°, if the triangulation domain has no two segments separated by an acute angle. Size optimality means that, for a given bound on the minimum angle, the algorithm produces a mesh whose cardinality (number of triangles) is at most a constant factor larger than the cardinality of the smallest-cardinality mesh that meets the same angle bound. The constant depends upon the angle bound, but is independent of the input PSLG. Alas, the constant is too large to be useful in practice, and the size optimality results are of theoretical interest only. Happily, there are lower bounds on edge lengths whose constants are small enough to be meaningful; these show that the meshes are nicely graded in a formal sense.

Chew [8, 9] proves that his algorithms can produce meshes with no angle smaller than 30°, albeit without any guarantees of grading or size optimality. He conjectures that the second of these algorithms offers the same guarantees as Ruppert's algorithm [9].

The foundation of this article is a new framework for analyzing Delaunay refinement—using simple, intuitive flow graphs—that unites Chew's and Ruppert's algorithms and points the way to a variety of improvements. The most important of these is a method that meshes domains with small angles that Chew's and Ruppert's original algorithms cannot mesh.

I describe Ruppert's algorithm, and reprise its analysis using the flow graph framework, in Section 3. In Section 4 I use the framework to conditionally confirm Chew's conjecture: his second published Delaunay refinement algorithm produces nicely graded, size-optimal meshes, if the bound on the smallest allowable angle is relaxed to 26.5°. The difference between this bound and the 20.7° bound of Ruppert's algorithm arises from Chew's method of deciding when to split segments. Examples show that Chew's algorithm behaves better in practice as well. However, the rare applications that require truly Delaunay meshes must still rely upon Ruppert's algorithm.

The algorithms of Ruppert and Chew 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 (e.g., if there are segments that extend into the interior of the triangulation domain). For example, Ruppert's algorithm sometimes fails to terminate. 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 triangulation domain—small input angles cannot be removed. However, one would like to triangulate a domain without creating any small angles that aren't already present in the input. Unfortunately, no algorithm can achieve this goal for all triangulation domains, as Section 5 demonstrates.

Therefore, to be universally applicable, a mesh generation algorithm must make decisions about where to create triangles that have small angles. The most important result of this paper, presented in Section 6, is a new Delaunay refinement 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 smallest input angle near a triangle is ϕ, where $\phi \leq 60^\circ$, the triangle cannot have an angle smaller than $\arcsin[\sin(\phi/2)/\sqrt{2}]$. Section 6.3 formalizes exactly what "near" means. Ideas related to Chew's algorithm help improve this bound to $\arcsin[(\sqrt{3}/2)\sin(\phi/2)] \sim (\sqrt{3}/4)\phi$ in Section 6.4. A 30° lower bound for all other angles can be established using an idea described in Section 7.3.

Several other extensions of Delaunay refinement are described in Section 7. Pseudocode and details on how to implement the algorithms are presented in Appendix A.

A mesh generator called *Triangle* [29], which implements most of the ideas in this article, is publicly available on the Web at http://www.cs.cmu.edu/~quake/triangle.html. Most of the meshes illustrated in this article were generated by Triangle. These examples demonstrate that Delaunay refinement algorithms often strongly outperform their worst-case bounds.

The engineering literature contains many mesh generation algorithms—too many to survey here. Some of these algorithms work well in practice for many applications, but it is likely that nearly all fail on some difficult triangulation domains, because there are no mathematical guarantees. The theoretical literature includes several provably good mesh generation algorithms based on grids or quadtrees, rather than Delaunay refinement. Baker, Grosse, and Rafferty [2] give the first provably good meshing algorithm, which produces triangulations whose angles are bounded between 13° and 90°. The triangles it produces are of uniform size. This shortcoming was addressed by the quadtree-based algorithm of Bern, Eppstein, and Gilbert [4], which is the first size-optimal meshing algorithm. It triangulates polygons so that no angle (except small input angles) is smaller than 18.4°. (This bound cannot be extended to general PSLGs.) Unlike Delaunay refinement algorithms, the Bern et al. algorithm is impractical because it generates far more triangles than are needed in practice; see Section 3 for a visual comparison. For a survey of provably good mesh generation algorithms, see Bern and Eppstein [3].

2 The Key Idea Behind Delaunay Refinement

In the finite element community, there are a wide variety of measures in use for the quality of a triangle, the most obvious being the smallest and largest angles of the triangle. Miller, Talmor, Teng, and Walkington [23] have pointed out that the most natural and elegant measure for analyzing Delaunay refinement algorithms is the *circumradius-to-shortest edge ratio* of a triangle or tetrahedron. The *circumcenter* and *circumradius* of a triangle are the center and radius of its circumcircle, respectively. The quotient of a triangle's circumradius r and the length ℓ of its shortest edge is the metric that is naturally improved by Delaunay refinement algorithms. One would like this ratio to be as small as possible.

Does optimizing this metric aid practical applications? Yes. A triangle's circumradius-to-shortest edge ratio r/ℓ is related to its smallest angle $\theta_{\min}$ by the formula $r/\ell = 1/(2\sin\theta_{\min})$. The smaller a triangle's

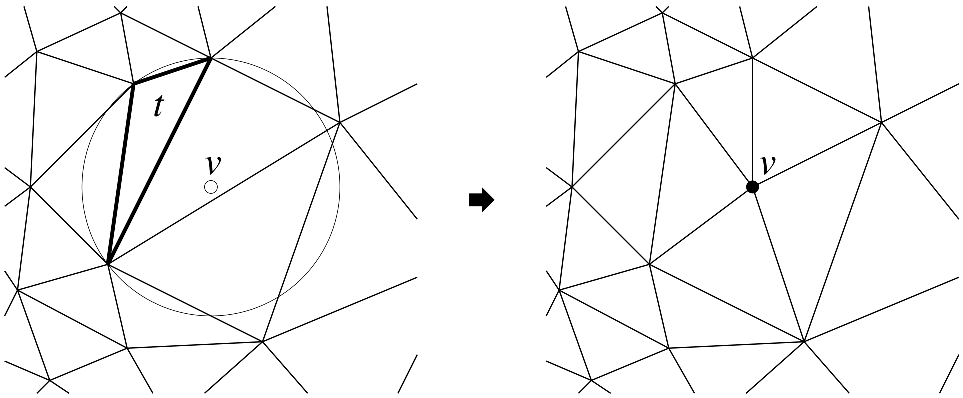


Figure 2: Any triangle whose circumradius-to-shortest edge ratio is larger than some bound B is split by inserting a vertex at its circumcenter. The Delaunay property is maintained, and the triangle is thus eliminated. Every new edge has length at least B times that of shortest edge of the poor triangle.

ratio, the larger its smallest angle. If B is an upper bound on the circumradius-to-shortest edge ratio of all triangles in a mesh, then there is no angle smaller than $\arcsin \frac{1}{2B}$ (and vice versa). A triangular mesh generator is wise to make B as small as possible.

The central operation of Chew's and Ruppert's Delaunay refinement algorithms is the insertion of a vertex at the circumcenter of a triangle of poor quality. The Delaunay property is maintained, using Lawson's algorithm [19] or the Bowyer–Watson algorithm [5, 32] for the incremental update of Delaunay triangulations. The poor-quality triangle cannot survive, because its circumcircle is no longer empty. For brevity, I refer to the act of inserting a vertex at a triangle's circumcenter as *splitting* a triangle. The idea dates back at least to the engineering literature of the mid-1980s [16]. If poor triangles are split one by one, either all will eventually be eliminated, or the algorithm will run forever.

The main insight behind the Delaunay refinement algorithms (including Chew's, Ruppert's, and the new ones in this article) is that the refinement loop is guaranteed to terminate if the notion of "poor quality" includes only triangles that have a circumradius-to-shortest edge ratio larger than some appropriate bound B. The only new edges created by the Delaunay insertion of a vertex v are edges connected to v (see Figure 2). Because v is the circumcenter of some Delaunay triangle t, and there were no vertices inside the circumcircle of t before v was inserted, no new edge can be shorter than the circumradius of t. Because t has a circumradius-to-shortest edge ratio larger than B, every new edge has length at least B times that of the shortest edge of t.

Ruppert's algorithm [27] employs a bound of $B = \sqrt{2}$, and Chew's second Delaunay refinement algorithm [9] employs a bound of $B = 1$. Chew's first Delaunay refinement algorithm [8] splits any triangle whose circumradius is greater than the length of the shortest edge in the entire mesh, thus achieving a bound of $B = 1$, but forcing all triangles to have uniform size. With these bounds, every new edge created is at least as long as some other edge already in the mesh. Hence, no vertex is ever inserted closer to another vertex than the length of the shortest edge in the initial triangulation. Delaunay refinement must eventually terminate, because the augmented triangulation will run out of places to put vertices. When it does, all angles are bounded between 20.7° and 138.6° for Ruppert's algorithm, and between 30° and 120° for Chew's.

Henceforth, a triangle whose circumradius-to-shortest edge ratio is greater than B is said to be *skinny*. Figure 3 provides a different intuition for why all skinny triangles are eventually eliminated by Delaunay refinement. The new vertices that are inserted into a triangulation (grey dots) are spaced roughly according

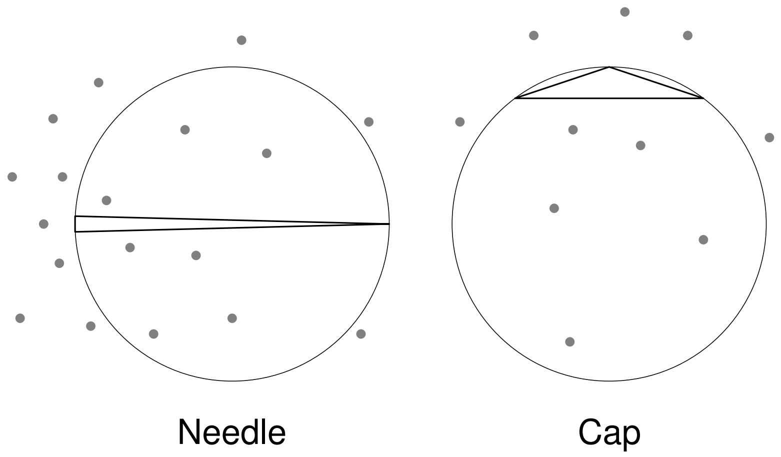


Figure 3: Skinny triangles have circumcircles larger than their shortest edges. Each skinny triangle may be classified as a needle, whose longest edge is much longer than its shortest edge, or a cap, which has an angle close to $180°$. (The classifications are not mutually exclusive.)

to the length of the shortest nearby edge. Because skinny triangles have relatively large circumradii, their circumcircles are inevitably popped. When enough vertices are introduced that the spacing of vertices is somewhat uniform, large empty circumcircles cannot adjoin small edges, and no skinny triangles can remain in the Delaunay triangulation. Fortunately, the spacing of vertices does not need to be so uniform that the mesh is poorly graded; this fact is formalized in Section 4.3.

These ideas generalize without change to higher dimensions, and are used in several three-dimensional Delaunay refinement algorithms [14, 10, 30]. Imagine a triangulation that has no boundaries—perhaps it has infinite extent, or perhaps it lies in a periodic space that "wraps around" at the boundaries. Regardless of the dimensionality, Delaunay refinement can eliminate all simplices having a circumradius-to-shortest edge ratio greater than one, without creating any edge shorter than the shortest edge already present.

Unfortunately, my description of Delaunay refinement thus far has a gaping hole: mesh boundaries have not been accounted for. The flaw in the procedure described above is that the circumcenter of a poor triangle might not lie in the mesh at all. Delaunay refinement algorithms, including the algorithms of Chew and Ruppert, are distinguished primarily by how they handle boundaries. Boundaries complicate mesh generation immensely.

3 Ruppert's Delaunay Refinement Algorithm

Jim Ruppert's algorithm for two-dimensional quality mesh generation [27] is perhaps the first theoretically guaranteed meshing algorithm to be truly satisfactory in practice. It extends an earlier Delaunay refinement algorithm of Chew [8]. Whereas Chew's first Delaunay refinement algorithm produces uniform meshes, Ruppert's allows the density of triangles to vary quickly over short distances, as illustrated in Figure 4. The number of triangles produced is typically smaller than the number produced either by Chew's algorithm or the Bern–Eppstein–Gilbert quadtree algorithm [4], as Figure 5 shows.

Chew independently developed a second Delaunay refinement algorithm quite similar to Ruppert's [9]. I present Ruppert's algorithm first in part because Ruppert's earliest publications of his results [25, 26] slightly predate Chew's, and mainly because the algorithm is accompanied by a proof that it produces meshes that

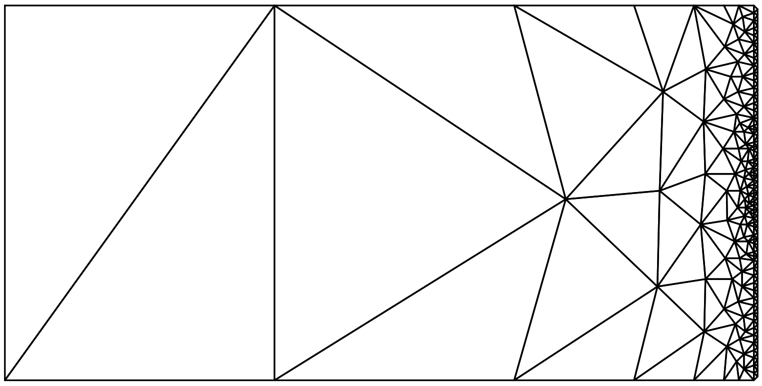


Figure 4: A demonstration of the ability of Delaunay refinement to achieve large gradations in triangle size while constraining angles. No angle is smaller than 24°.

are both nicely graded and size-optimal. Sections 4.2 and 4.3 apply Ruppert's analysis method to Chew's second algorithm, which yields better bounds on triangle quality than Ruppert's.

3.1 Description of the Algorithm

Ruppert's algorithm is presented here with a few modifications from Ruppert's original presentation. The most significant change is that the algorithm here begins with the constrained Delaunay triangulation of the segment-bounded PSLG provided as input. In contrast, Ruppert's presentation begins with a Delaunay triangulation, and the missing segments are recovered by inserting vertices at the midpoints of the missing segments. Most of the improvements discussed in this article depend on constrained Delaunay triangulations (see Sections 4, 6, and 7.1), so this treatment analyzes the constrained case.

The triangulation must remain constrained Delaunay after each vertex insertion. Fortunately, Lawson's algorithm and the Bowyer–Watson algorithm for incremental vertex insertion can easily be adapted to constrained Delaunay triangulations, simply by never flipping or removing any subsegment. A new vertex cannot cause a triangle to be deleted if a subsegment occludes the visibility between the vertex and the triangle.

Ruppert's algorithm inserts additional vertices until all triangles satisfy the constraints on quality and size set by the user. Some vertices are inserted at triangle circumcenters, and some vertices are inserted to divide segments into subsegments. The algorithm interleaves segment splitting with triangle splitting. 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 interior of the subsegment. (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 6. These subsegments have smaller diametral circles, and may or may not be encroached themselves; splitting continues until no subsegment is encroached.
- Each skinny triangle (having a circumradius-to-shortest edge ratio greater than some bound B) is normally split by inserting a vertex at its circumcenter, thus eliminating the triangle. However, if the

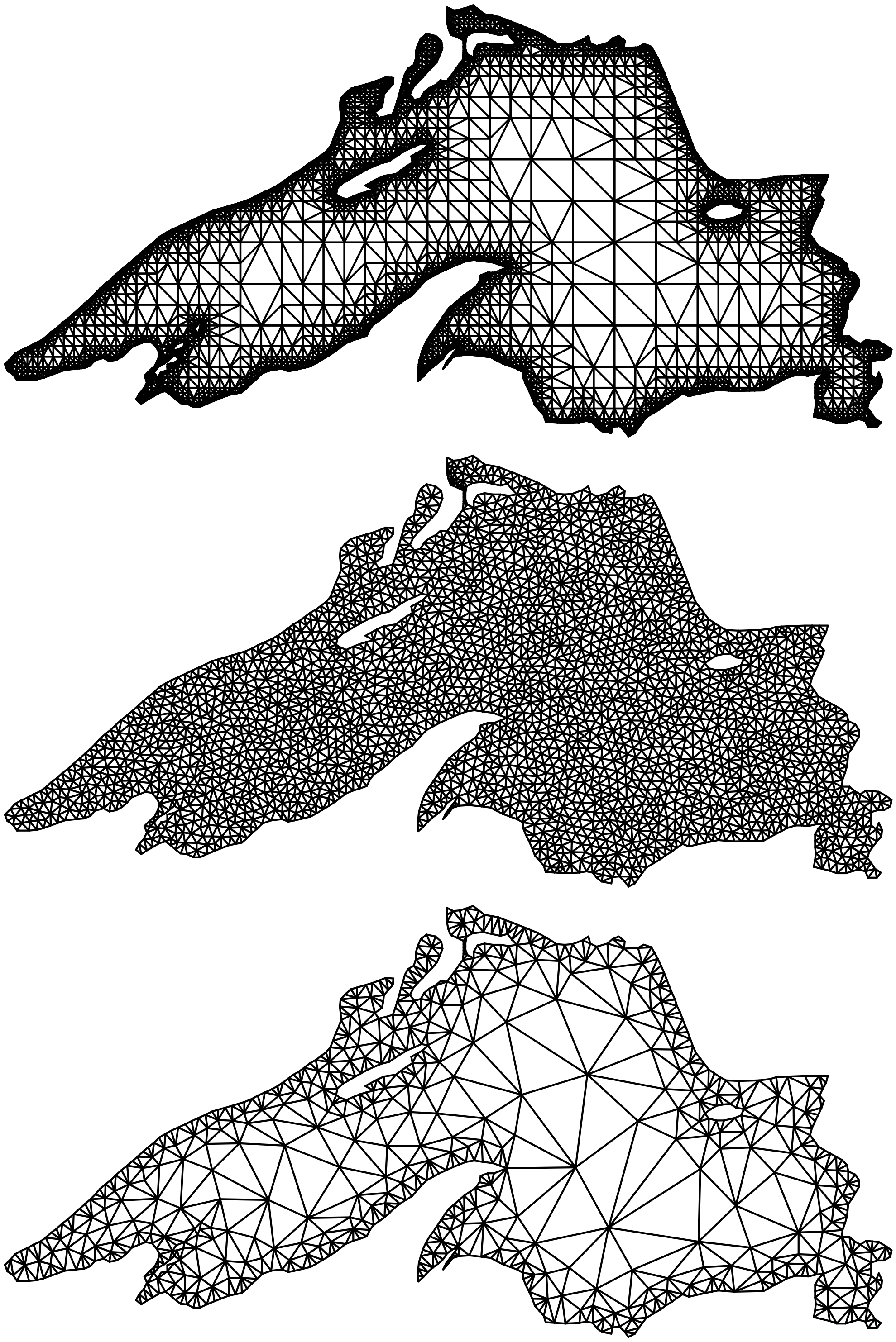


Figure 5: Meshes generated by the Bern–Eppstein–Gilbert quadtree-based algorithm (top), Chew's first Delaunay refinement algorithm (center), and Ruppert's Delaunay refinement algorithm (bottom). For this polygon, Chew's second Delaunay refinement algorithm produces nearly the same mesh as Ruppert's. (The first mesh was produced by the program `tripoint`, courtesy Scott Mitchell.)

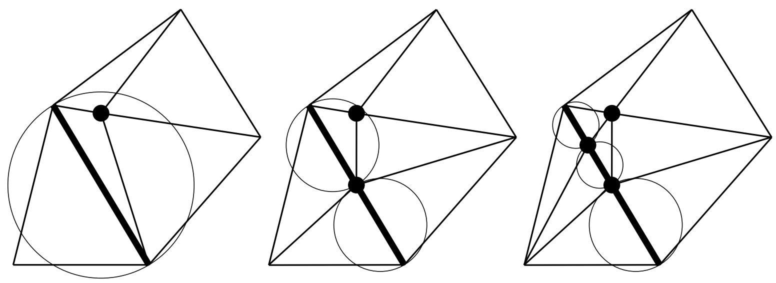


Figure 6: Segments are split recursively (while maintaining the constrained Delaunay property) until no subsegment is encroached.

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.

When no encroached subsegments remain, all triangles and edges of the triangulation are Delaunay. A mesh produced by Ruppert's algorithm is Delaunay, and not just constrained Delaunay.

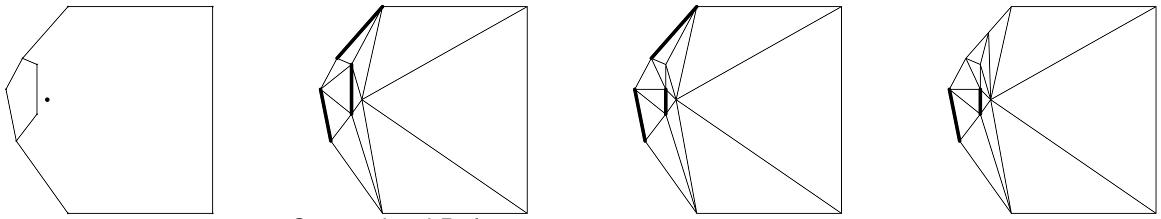
Figure 7 illustrates the generation of a mesh by Ruppert's algorithm from start to finish. Several characteristics of the algorithm are worth noting. First, if the circumcenter of a skinny triangle is considered for insertion and rejected, it may still be successfully inserted later, after the subsegments it encroaches upon have been split. On the other hand, the act of splitting those subsegments is sometimes enough to eliminate the skinny triangle. Second, the smaller features at the left end of the mesh lead to the insertion of some vertices to the right, but the size of the triangles on the right remains larger than the size of the triangles on the left. The smallest angle in the final mesh is $21.8°$.

There is a loose end to tie up. What should happen if the circumcenter of a skinny triangle falls outside the triangulation? Fortunately, the following lemma shows the question is moot.

Lemma 1 *Let T be a segment-bounded Delaunay triangulation. (Hence, any edge of T that belongs to only one triangle is a subsegment). Suppose that T has no encroached subsegments. Let v be the circumcenter of some triangle t of T. Then v lies in T.*

Proof: Suppose for the sake of contradiction that v lies outside T. Let c be the centroid of t; c clearly lies inside T. Because the triangulation is segment-bounded, the line segment cv must cross some subsegment s, as Figure 8 illustrates. (If there are several such subsegments, let s be the subsegment nearest c.) Because cv is entirely enclosed by the circumcircle of t, the circumcircle must enclose a portion of s; but the constrained Delaunay property requires that the circumcircle enclose no vertex visible from c, so the circumcircle cannot enclose the endpoints of s.

Because c and the center of t's circumcircle lie on opposite sides of s, the portion of the circumcircle that lies strictly on the same side of s as c (the bold arc in the illustration) is entirely enclosed by the diametral circle of s. Each vertex of t lies on t's circumcircle and either is an endpoint of s, or lies on the same side of s as c. Up to two of the vertices of t may be endpoints of s, but at least one vertex of t must lie strictly inside the diametral circle of s. But T has no encroached subsegments by assumption; the result follows by contradiction. ■

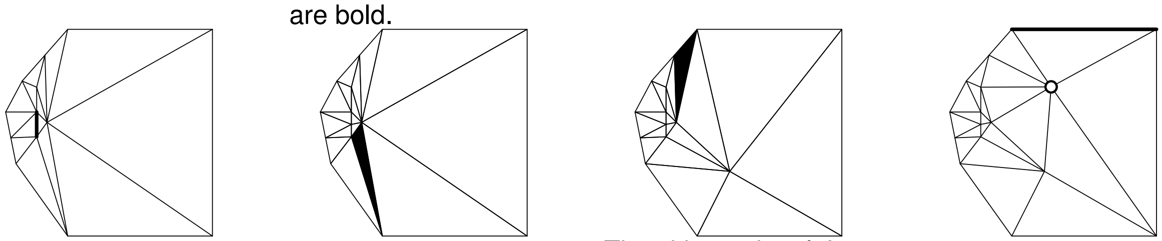


A sample input PSLG.

Constrained Delaunay triangulation of the input vertices. Encroached segments are bold.

One encroached subsegment is bisected.

And another.

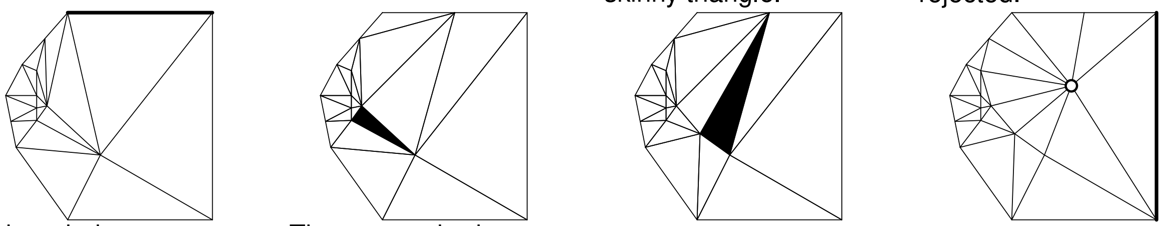


A third encroached subsegment is split.

The last encroached subsegment is split. Find a skinny triangle.

The skinny triangle's circumcenter is inserted. Find another skinny triangle.

This circumcenter encroaches upon a segment, and is rejected.

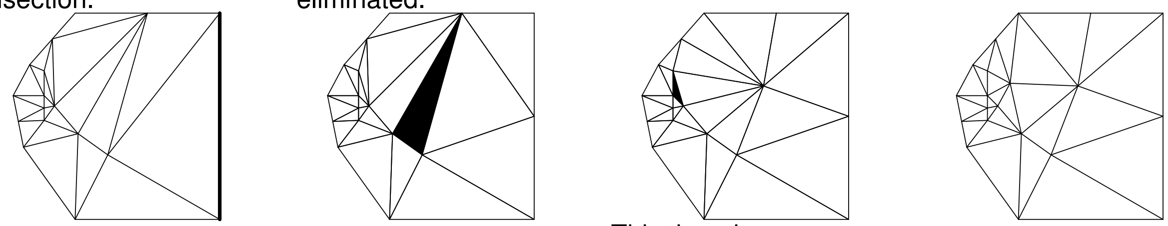


Although the vertex was rejected, the segment it encroached upon is still marked for bisection.

The encroached segment is split, and the skinny triangle that led to its bisection is eliminated.

A circumcenter is successfully inserted, creating another skinny triangle.

The triangle's circumcenter is rejected.



The encroached segment will be split.

The skinny triangle was not eliminated. Attempt to insert its circumcenter again.

This time, its circumcenter is inserted successfully. There's only one skinny triangle left.

The final mesh. No angle is smaller than $20.7°$.

Figure 7: A complete run of Ruppert's algorithm with an upper bound of $B = \sqrt{2}$ on circumradius-to-shortest edge ratios. 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.

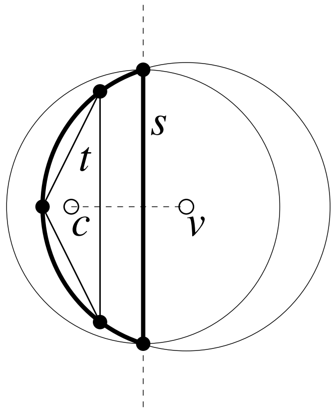


Figure 8: If the circumcenter v of a triangle t lies outside the triangulation, then some subsegment s is encroached.

Lemma 1 offers the best reason why encroached subsegments are given priority over skinny triangles. Because a circumcenter is inserted only when there are no encroached subsegments, one is assured that the circumcenter will be within the triangulation. The act of splitting encroached subsegments rids the mesh of triangles whose circumcircles lie outside it. The lemma is also reassuring to applications (like some finite volume methods) that require all triangle circumcenters to lie within the triangulation.

In addition to being required to satisfy a quality criterion, triangles can also be required to satisfy a maximum size criterion. In a finite element problem, the triangles must be small enough to ensure that the finite element solution accurately approximates the true solution of some partial differential equation. Ruppert's algorithm can allow the user to specify an upper bound on allowable triangle areas or edge lengths, and the bound may be a function of each triangle's location. Triangles that exceed the local upper bound are split, whether they are skinny or not. So long as the function bounding the sizes of triangles is itself everywhere greater than some positive constant, there is no threat to the algorithm's termination guarantee.

3.2 Local Feature Sizes of Planar Straight Line Graphs

The claim that Ruppert's algorithm produces nicely graded meshes is based on the fact that the spacing of vertices at any location in the mesh is within a constant factor of the sparsest possible spacing. To formalize the idea of "sparsest possible spacing," Ruppert introduces a function called the *local feature size*, which is defined over the plane relative to a specific PSLG.

Given a PSLG X, the local feature size $\mathrm{lfs}(p)$ at any point p is the radius of the smallest disk centered at p that intersects two nonincident vertices or segments of X. (Two distinct features, each a vertex or segment, are said to be *incident* if they intersect.) Figure 9 illustrates the notion by giving examples of such disks for a variety of points.

The local feature size of a point is proportional to the sparsest possible spacing of vertices in the neighborhood of that point in any triangulation that respects the segments and has no skinny triangles. The function $\mathrm{lfs}(\cdot)$ is continuous and has the property that its directional derivatives (where they exist) are bounded in the range $[-1, 1]$. This property leads to a lower bound (within a constant factor to be derived in Section 4.3) on the rate at which edge lengths grade from small to large as one moves away from a small feature. Formally, this is what it means for a mesh to be "nicely graded."

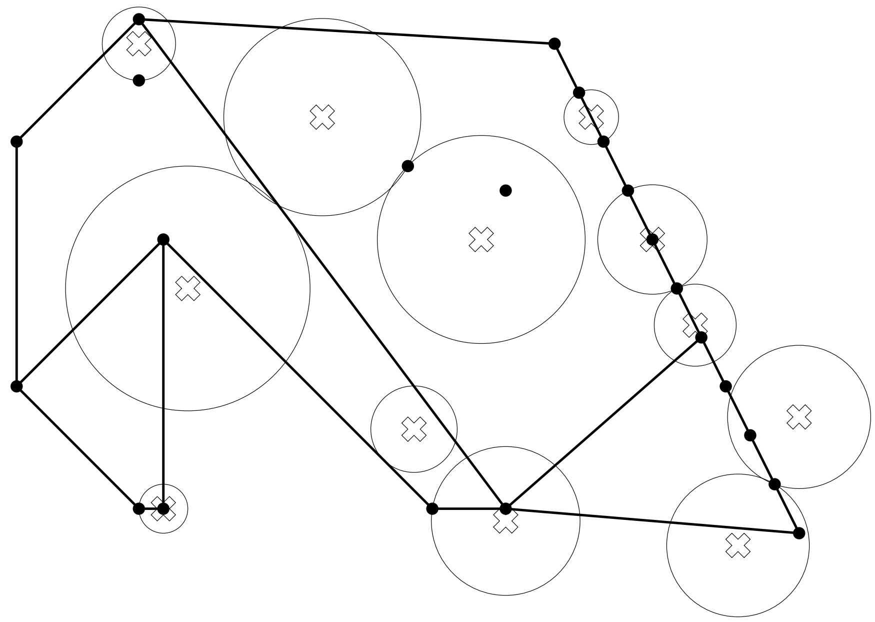


Figure 9: The radius of each disk illustrated here is the local feature size of the point at its center.

Lemma 2 (Ruppert [27]) *For any PSLG X, and any two points u and v in the plane,*

$$\mathrm{lfs}(v) \leq \mathrm{lfs}(u) + |uv|.$$

Proof: The disk having radius $\mathrm{lfs}(u)$ centered at u intersects two nonincident features of X. The disk having radius $\mathrm{lfs}(u) + |uv|$ centered at v contains the prior disk, and thus also intersects the same two features. Hence, the smallest disk centered at v that intersects two nonincident features of X has radius no larger than $\mathrm{lfs}(u) + |uv|$. ■

If the triangulation domain is nonconvex or nonplanar, this lemma can be generalized to use geodesic distances—lengths of shortest paths that are constrained to lie within the triangulation domain—instead of straight-line distances. The proof relies only on the triangle inequality: if u is within a distance of $\mathrm{lfs}(u)$ of each of two nonincident features, then v is within a distance of $\mathrm{lfs}(u) + |uv|$ of each of those same two features. The use of geodesic distances is discussed in detail in Section 7.1.

3.3 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? In this section and in Section 4.3, I present two proofs of the termination of Ruppert's algorithm. The first is included for its intuitive value, and because it offers the best bound on the lengths of the shortest edges. The second proof, adapted from Ruppert, offers better bounds on the lengths of the longer edges of a graded mesh, and thus shows that the algorithm produces meshes that are nicely graded and size-optimal. The presentation here uses a flow graph to expose the intuition behind Ruppert's proof and its natural tendency to bound the circumradius-to-shortest edge ratio.

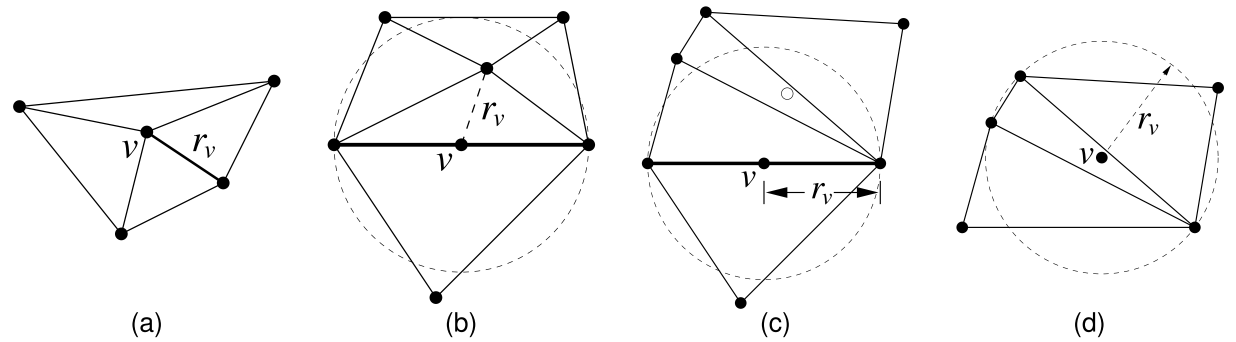


Figure 10: The insertion radius r_v of a vertex v is the distance to the nearest vertex when v first appears in the mesh. (a) If v is an input vertex, r_v is the distance to the nearest other input vertex. (b) If v is the midpoint of a subsegment encroached upon by a mesh vertex, r_v is the distance to that vertex. (c) If v is the midpoint of a subsegment encroached upon only by a rejected vertex, r_v is the radius of the subsegment's diametral circle. (d) If v is the circumcenter of a skinny triangle, r_v is the radius of the circumcircle.

Both proofs require that $B \geq \sqrt{2}$, and that any two incident segments (segments that share an endpoint) in the input PSLG are separated by an angle of 60° or greater. (Ruppert asks for angles of at least 90°, but the weaker bound suffices.) For the second proof, these inequalities must be strict.

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. Consider what this means in three different cases.

- If v is an input vertex, then r_v is the Euclidean distance between v and the nearest input vertex visible from v. See Figure 10(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 10(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, and hence the length of each of the two subsegments thus produced; see Figure 10(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 10(d).

If a vertex is considered for insertion but rejected because of an encroachment, its insertion radius is defined the same way—as if it had been inserted, even though it is not actually inserted.

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. The parent is defined as follows.

- If v is an input vertex, it has no parent.
- If v is a vertex inserted at the midpoint of an encroached subsegment, then $p(v)$ is the encroaching vertex. (Note that $p(v)$ might be a rejected vertex; a parent need not be a mesh vertex.) If there are several encroaching vertices, choose the one nearest v.

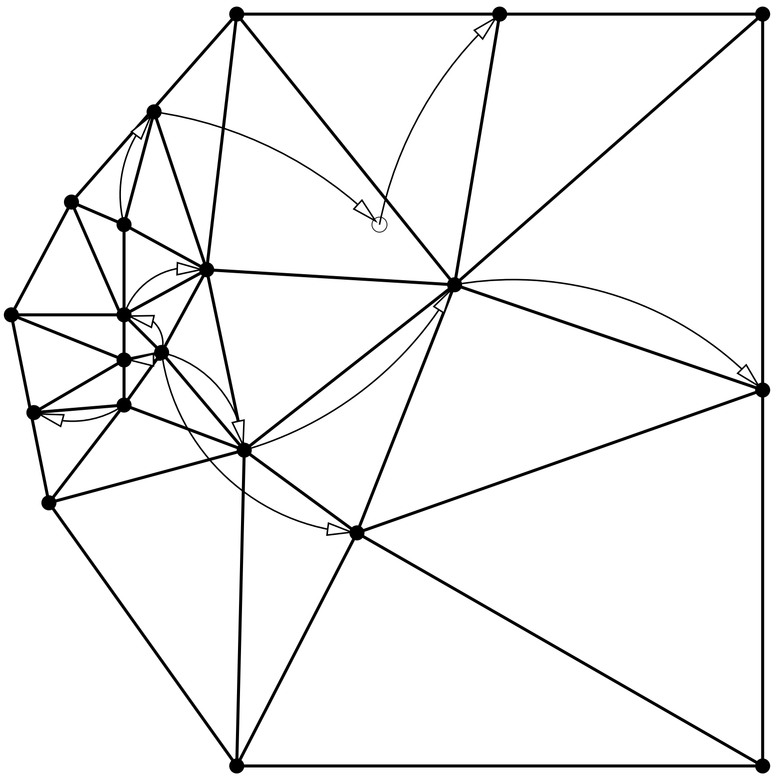


Figure 11: Trees of vertices for the example of Figure 7. Arrows are directed from parents to their children. Children include all inserted vertices and one rejected vertex.

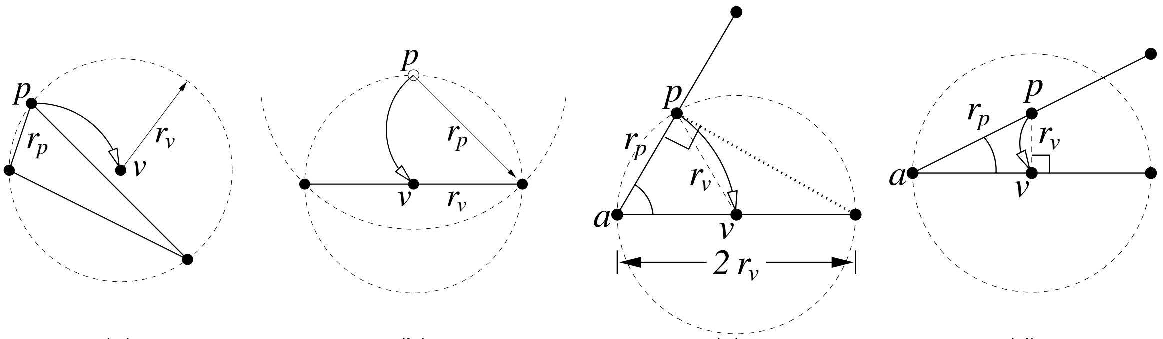
- If v is a vertex inserted (or rejected) at the circumcenter of a skinny triangle, then $p(v)$ is the most recently inserted endpoint of the shortest edge of that triangle. If both endpoints of the shortest edge are input vertices, choose one arbitrarily.

Each input vertex is the root of a tree of vertices. However, what is interesting is not each tree as a whole, but the sequence of ancestors of any given vertex, which forms a sort of history of the events leading to the insertion of that vertex. Figure 11 illustrates the parents of all vertices inserted or considered for insertion during the sample execution of Ruppert's algorithm in Figure 7.

I will use these definitions to show why Ruppert's algorithm terminates. The key insight is that no descendant of a mesh vertex has an insertion radius smaller than the vertex's own insertion radius—unless the descendant's local feature size is even smaller. Therefore, no edge will ever appear that is shorter than the smallest feature in the input PSLG. To prove these facts, consider the relationship between the insertion radii of a vertex and its parent.

Lemma 3 *Let v be a vertex, and let $p = p(v)$ be its parent, if one exists. Then either $r_v \geq \mathrm{lfs}(v)$, or $r_v \geq C r_p$, where*

- *$C = B$ if v is the circumcenter of a skinny triangle;*
- *$C = 1/\sqrt{2}$ if v is the midpoint of an encroached subsegment and p is the (rejected) circumcenter of a skinny triangle;*



(a) (b) (c) (d)

Figure 12: The relationship between the insertion radii of a child and its parent. (a) When a skinny triangle is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by the circumcenter of a skinny triangle, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than the parent's, as this worst-case example shows. (c, d) When a subsegment is encroached upon by a vertex in an incident segment, the relationship depends upon the angle α separating the two segments.

- $C = 1/(2\cos\alpha)$ *if v and p lie on incident segments separated by an angle of α (with p encroaching upon the subsegment whose midpoint is v), where $45^\circ \leq \alpha < 90^\circ$; and*
- $C = \sin\alpha$ *if v and p lie on incident segments separated by an angle of $\alpha \leq 45^\circ$.*

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 lemma 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 12(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 four cases to consider. The first two are all that is needed to prove the termination of Ruppert's algorithm if no angle smaller than 90° is present in the input. The last two cases consider the effects of acute angles.

- If the parent p is an input vertex, or was inserted in a segment not incident to the segment containing s, then by definition, $\text{lfs}(v) \leq r_v$.
- If p is a circumcenter that was considered for insertion but rejected because it encroaches upon s, then p lies on or 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 12(b) for an example where the relation is equality.
- If v and p lie on incident segments separated by an angle α where $45^\circ \leq \alpha < 90^\circ$, the vertex a (for "apex") where the two segments meet obviously cannot lie inside the diametral circle of s; see Figure 12(c). Because s is encroached upon by p, p lies on or inside its diametral circle. To find the worst-case (smallest) value of r_v/r_p, imagine that r_p and α are fixed; then $r_v = |vp|$ is minimized by making the subsegment s as short as possible, subject to the constraint that p cannot fall outside its diametral circle. The minimum is achieved when $|s| = 2r_v$. Basic trigonometry shows that $|s| \geq r_p/\cos\alpha$, and therefore $r_v \geq r_p/(2\cos\alpha)$.

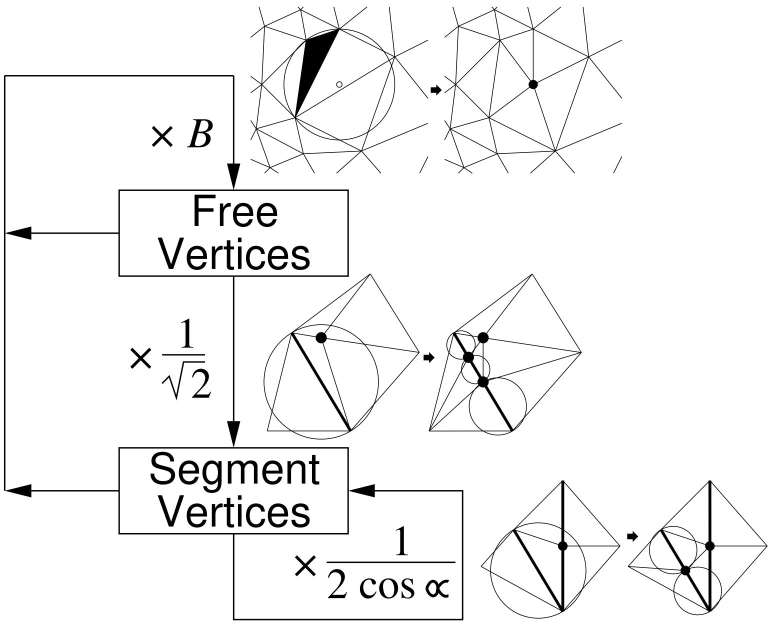


Figure 13: Flow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycles have a product smaller than one, Ruppert's Delaunay refinement algorithm will terminate. Input vertices are omitted from the diagram because they cannot contribute to cycles.

- If v and p lie on incident segments separated by an angle α where $\alpha \leq 45^\circ$, then r_v/r_p is minimized not when p lies on the diametral circle, but when v is the orthogonal projection of p onto s, as illustrated in Figure 12(d). Hence, $r_v \geq r_p \sin \alpha$. ■

Lemma 3 limits how quickly the insertion radii can decrease through 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 13 expresses this notion as a flow 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 $\sqrt{2}$, and ensuring that the minimum angle between input segments is at least 60°. The following theorem formalizes these ideas.

Theorem 4 *Let* $\mathrm{lfs}_{\min}$ *be the shortest distance between two nonincident entities (vertices or segments) of the input PSLG*[1].

[1]Equivalently, $\mathrm{lfs}_{\min} = \min_u \mathrm{lfs}(u)$, where u is chosen from among the input vertices. The proof that both definitions are equivalent is omitted, but it relies on the recognition that if two points lying on nonincident segments are separated by a distance d, then at least one of the endpoints of one of the two segments is separated from the other segment by a distance of d or less. Note that $\mathrm{lfs}_{\min}$ is not a lower bound for $\mathrm{lfs}(\cdot)$ over the entire domain; for instance, a segment may have length $\mathrm{lfs}_{\min}$, in which case the local feature size at its midpoint is $\mathrm{lfs}_{\min}/2$.

Suppose that any two incident segments are separated by an angle of at least $60°$*, and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is larger than* B*, where* $B \geq \sqrt{2}$*. Ruppert's algorithm will terminate, with no triangulation edge shorter than* $\mathrm{lfs}_{\min}$.

Proof: Suppose for the sake of contradiction that the algorithm introduces an edge shorter than $\mathrm{lfs}_{\min}$ 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 $\mathrm{lfs}_{\min}$ existed before v was inserted. Hence, for any ancestor a of v that is a mesh vertex, $r_a \geq \mathrm{lfs}_{\min}$. 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 3, $r_v \geq Br_p \geq \sqrt{2}r_p$.
- If v is the midpoint of an encroached subsegment and p is the circumcenter of a skinny triangle, then by Lemma 3, $r_v \geq r_p/\sqrt{2} \geq Br_g/\sqrt{2} \geq r_g$. (Recall that p is rejected.)
- If v and p lie on incident segments, then by Lemma 3, $r_v \geq r_p/(2\cos\alpha)$. Because $\alpha \geq 60°$, $r_v \geq r_p$.

In all three cases, $r_p \geq r_a$ for some ancestor a of p in the mesh. It follows that $r_p \geq \mathrm{lfs}_{\min}$, contradicting the assumption that e has length less than $\mathrm{lfs}_{\min}$. It also follows that no edge shorter than $\mathrm{lfs}_{\min}$ is ever introduced, so the algorithm must terminate. ■

By design, Ruppert's algorithm terminates only when all triangles in the mesh have a circumradius-to-shortest edge ratio of B or better; 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 $20.7°$. Sections 4 and 7 describe several ways to improve this bound.

What about running time? A constrained Delaunay triangulation can be constructed in $\mathcal{O}(n \log n)$ time [7], where n is the size of the input PSLG. Once the initial triangulation is complete, well-implemented Delaunay refinement algorithms invariably take time linear in the number of additional vertices that are inserted. See Appendix A for advice on how to achieve this speed. Ruppert (personal communication) exhibits a PSLG on which his algorithm takes $\Theta(h^2)$ time, where h is the size of the final mesh, but the example is contrived and such pathological examples do not arise in practice.

4 Chew's Second Delaunay Refinement Algorithm

Paul Chew has published at least two Delaunay refinement algorithms of great interest. The first, not described in this article, produces triangulations of uniform density by dividing segments into subsegments of nearly-uniform length before applying Delaunay refinement [8]. (See Figure 5, center.) The second, which can produce graded meshes, is discussed here.

Compared to Ruppert's algorithm, Chew's second Delaunay refinement algorithm [9] offers an improved guarantee of good grading in theory, and splits fewer subsegments in practice. This section shows that the algorithm exhibits good grading and size optimality for angle bounds of up to $26.5°$ (compared with $20.7°$ for Ruppert's algorithm). Chew shows that his algorithm terminates for an angle bound of up to $30°$, albeit with no guarantee of good grading or size optimality. The means by which he obtains this bound is discussed in Section 7.3.

Chew's paper also discusses triangular meshing of curved surfaces in three dimensions, but I consider the algorithm only in its planar context.

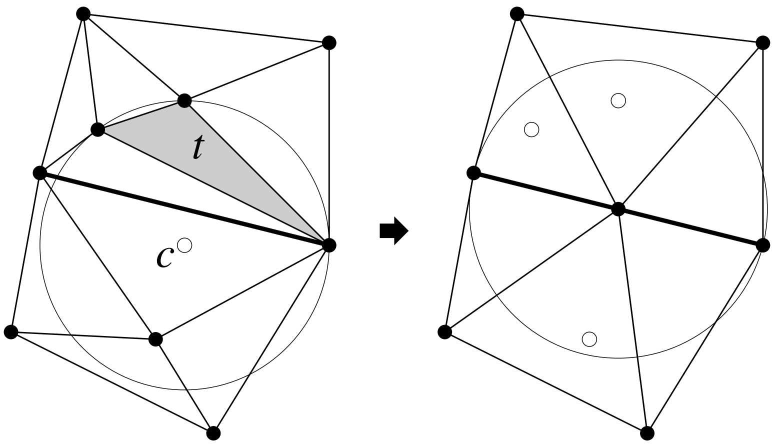


Figure 14: At left, a skinny triangle and its circumcenter lie on opposite sides of a subsegment. At right, all vertices in the subsegment's diametral circle have been deleted, and a new vertex has been inserted at the subsegment's midpoint.

4.1 Description of the Algorithm

Chew's second Delaunay refinement algorithm begins with the constrained Delaunay triangulation of a segment-bounded PSLG, and eliminates skinny triangles through Delaunay refinement, but Chew does not use diametral circles to determine if subsegments are encroached. Instead, it may arise that a skinny triangle t cannot be split because t and its circumcenter c lie on opposite sides of a subsegment s. (Lemma 1 does not apply to Chew's algorithm, so c may even lie outside the triangulation.) Although Chew does not use the word, let us say that s is *encroached* when this circumstance occurs.

Because s is a subsegment, inserting a vertex at c will not remove t from the mesh. Instead, c is rejected, and all free vertices that lie inside the diametral circle of s and are visible from the midpoint of s are deleted from the triangulation. (Input vertices and segment vertices are not deleted.) Then, a new vertex is inserted at the midpoint of s. The constrained Delaunay property is maintained throughout all vertex deletions and insertions. Figure 14 illustrates a subsegment split in Chew's algorithm.

If several subsegments lie between t and c, only the subsegment nearest t is split. If no subsegment lies between t and c, but c lies precisely on a subsegment, then that subsegment is considered encroached and split at its midpoint.

Chew's second algorithm produces a mesh that is not guaranteed to be Delaunay (only constrained Delaunay). For the few applications that require truly Delaunay triangles, Ruppert's algorithm is preferable. For the majority of applications, however, Chew has two advantages. First, some subsegment splits are avoided that would otherwise have occurred, so the final mesh may have fewer triangles, depending on the edge lengths of the PSLG. Consider two contrasting examples. In Figure 5 (bottom), the segments are so short that few are ever encroached, so Ruppert and Chew generate virtually the same mesh. In Figure 15, the segments are long compared to their local feature sizes, and Chew produces many fewer triangles.

The second advantage is that when a subsegment is split by a vertex v with parent p, a better bound can be found for the ratio between r_v and r_p than Lemma 3's bound. This improvement leads to better bounds on the minimum angle, the edge lengths, and the mesh cardinality.

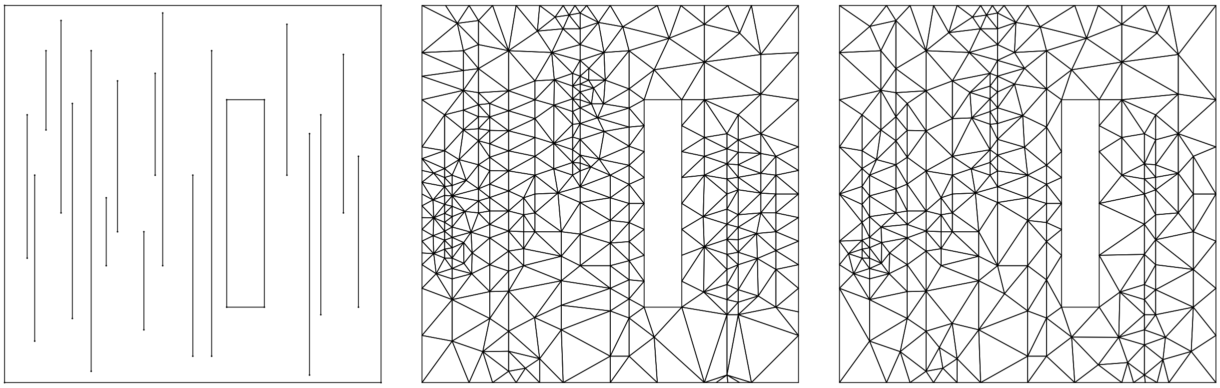


Figure 15: A PSLG, a 559-triangle mesh produced by Ruppert's algorithm, and a 423-triangle mesh produced by Chew's second algorithm. No angle in either mesh is smaller than $25°$.

4.2 Proof of Termination

If no input angle is less than $60°$, Chew's algorithm terminates for any bound on circumradius-to-shortest edge ratio B such that $B \geq \sqrt{5}/2 \doteq 1.12$. Therefore, the smallest angle can be bounded by up to $\arcsin(1/\sqrt{5}) \doteq 26.56°$. Section 7.3 discusses how Chew improves this bound to $30°$, but the weaker result is discussed here because it is the first step to proving (in Section 4.3) that Chew's algorithm offers guaranteed good grading and size optimality for angle bounds less than $26.56°$. PSLGs with angles less than $60°$ are treated in Section 6.4.

By the reasoning of Lemma 1, if a triangle and its circumcenter lie on opposite sides of a subsegment, or if the circumcenter lies on the subsegment, then some vertex of the triangle (other than the subsegment's endpoints) lies on or inside the subsegment's diametral circle. Hence, Chew's algorithm never splits a subsegment that Ruppert's algorithm would not split. It follows that the inequalities in Lemma 3 are as true for Chew's algorithm as they are for Ruppert's algorithm. However, Chew will often decline to split a subsegment that Ruppert would split, and thus splits fewer subsegments overall. A consequence is that the relationship between the insertion radii of a subsegment midpoint and its parent can be tightened.

Lemma 5 *Let $\theta = \arcsin \frac{1}{2B}$ be the angle bound below which a triangle is considered skinny. Let s be a subsegment that is encroached because some skinny triangle t and its circumcenter c lie on opposite sides of s (or c lies on s). Let v be the vertex inserted at the midpoint of s. Then one of the following four statements is true. (Only the fourth differs from Lemma 3.)*

- $r_v \geq \mathrm{lfs}(v)$;
- $r_v \geq r_p/(2\cos\alpha)$, *where p is a vertex that encroaches upon s and lies in a segment separated by an angle of $\alpha \geq 45°$ from the segment containing s;*
- $r_v \geq r_p \sin\alpha$, *where p is as above, with $\alpha \leq 45°$; or*
- *there is some vertex p (which is deleted from inside the diametral circle of s or lies precisely on the diametral circle) such that $r_v \geq r_p \cos\theta$.*

Proof: Chew's algorithm deletes all free vertices inside the diametral circle of s that are visible from v. If any vertex remains visible from v inside the diametral circle, it is an input vertex or a segment vertex. Define

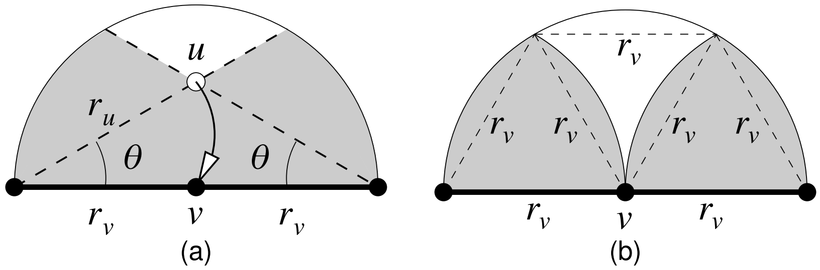


Figure 16: (a) The case where exactly one vertex is in the semicircle. (b) The case where more than one vertex is in the semicircle.

the parent p of v to be the closest such vertex. If p is an input vertex or lies on a segment not incident to the segment that contains s, then $\mathrm{lfs}(v) \leq r_v$ and the lemma holds. If p lies on an incident segment, then $r_v \geq r_p/(2\cos\alpha)$ for $\alpha \geq 45^\circ$ or $r_v \geq r_p \cos\theta$ for $\alpha \leq 45^\circ$ as in Lemma 3.

Otherwise, no vertex inside the diametral circle of s is visible after the deletions, so r_v is equal to the radius of the diametral circle. This is the reason why Chew's algorithm deletes the vertices: when v is inserted, the nearest visible vertices are the subsegment endpoints, and no short edge appears.

Mentally jump back in time to just before the vertex deletions. Assume without loss of generality that t lies above s, with c below. Following Lemma 1, at least one vertex of t lies on or inside the upper half of the diametral circle of s. There are two cases, depending on the total number of vertices on or inside this semicircle.

If the upper semicircle encloses only one vertex u visible from v, then t is the triangle whose vertices are u and the endpoints of s. Because t is skinny, u must lie in the shaded region of Figure 16(a). The insertion radius r_u cannot be greater than the distance from u to the nearest endpoint of s, so $r_v \geq r_u \cos\theta$. (For a fixed r_v, r_u is maximized when u lies at the apex of the isosceles triangle whose base is s and whose base angles are θ.) Define the parent of v to be u.

If the upper semicircle encloses more than one vertex visible from v, consider Figure 16(b), in which the shaded region represents points within a distance of r_v from an endpoint of s. If some vertex u lies in the shaded region, then $r_u \leq r_v$; define the parent of v to be u. If no vertex lies in the shaded region, then there are at least two vertices visible from v in the white region of the upper semicircle. Let u be the most recently inserted of these vertices. The vertex u is at a distance of at most r_v from any other vertex in the white region, so $r_u \leq r_v$; define the parent of v to be u. ■

Lemma 5 extends the definition of parent to accommodate the new type of encroachment defined in Chew's algorithm. When a subsegment s is encroached, the parent p of its newly inserted midpoint v is defined to be a vertex on or inside the diametral circle of s, just as in Ruppert's algorithm.

Chew's algorithm can be shown to terminate in the same manner as Ruppert's. Do the differences between Chew's and Ruppert's algorithms invalidate any of the assumptions used in Theorem 4 to prove termination? The most important difference is that vertices may be deleted from the mesh. When a vertex is deleted from a constrained Delaunay triangulation, no surviving vertex finds itself adjoining a shorter edge than the shortest edge it adjoined before the deletion. (This fact follows because a constrained Delaunay triangulation connects every vertex to its nearest visible neighbor.) Hence, each vertex's insertion radius still serves as a lower bound on the lengths of all edges that connect the vertex to vertices older than itself.

If vertices can be deleted, are we certain that the algorithm will run out of places to put new vertices? Observe that vertex deletions only occur when a subsegment is split, and vertices are never deleted from

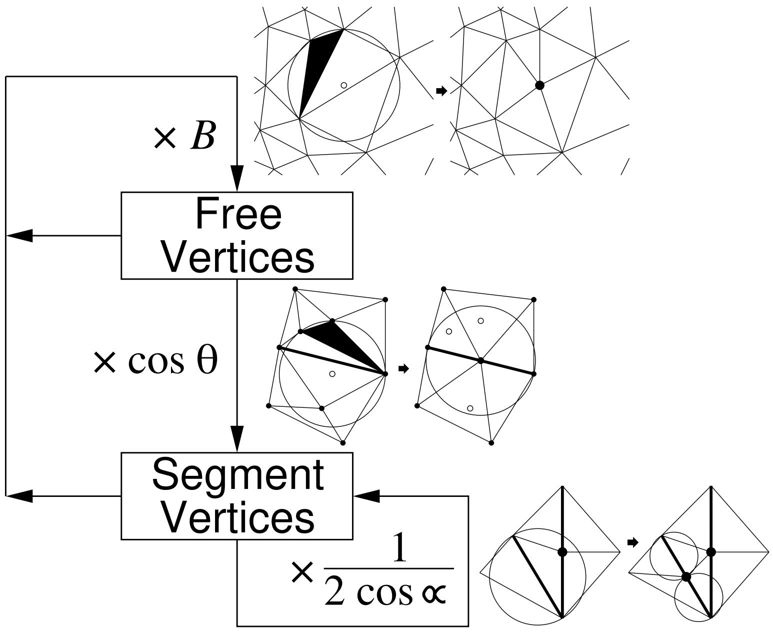


Figure 17: Flow diagram for Chew's algorithm.

segments. Theorem 4 sets a lower bound on the length of each subsegment, so only a finite number of subsegment splits can occur. After the last subsegment split, no more vertex deletions occur, and eventually there will be no space left for new vertices. Therefore, Theorem 4 holds for Chew's algorithm as well as Ruppert's.

The consequence of the bound proven by Lemma 5 is illustrated in the flow graph of Figure 17. Recall that termination is guaranteed if no cycle has a product less than one. Hence, a condition of termination is that $B \cos\theta \geq 1$. As $\theta = \arcsin \frac{1}{2B}$, the best bound that satisfies this criterion is $B = \sqrt{5}/2 \doteq 1.12$, which corresponds to an angle bound of $\arcsin(1/\sqrt{5}) \doteq 26.56^\circ$.

4.3 Good Grading and Size Optimality in Ruppert's and Chew's Algorithms

Theorem 4 guarantees that no edge of a mesh produced by Ruppert's algorithm is shorter than $\mathrm{lfs}_{\min}$, and the proof extends to Chew's algorithm. This guarantee may be satisfying for a user who desires a uniform mesh, but it is not satisfying for a user who requires a spatially graded mesh. What follows is a proof that each edge of the output mesh has length proportional to the local feature sizes of its endpoints. Hence, edge lengths are determined by local considerations; features lying outside the disk that defines the local feature size of a point can only weakly influence the lengths of edges that contain that point. Triangle sizes vary quickly over short distances where such variation is desirable to help reduce the number of triangles in the mesh.

The main point of this section is to demonstrate that Chew's algorithm offers better theoretical guarantees about triangle quality, edge lengths, and good grading than Ruppert's. (We should not forget, though, that it is Ruppert's analysis technique that allows us to draw this conclusion.) Whereas Ruppert only guarantees good grading and size optimality for angle bounds less than about 20.7°, Chew can make these promises for angle bounds less than about 26.5°, and offer better bounds on edge lengths for the angle bounds where Ruppert's guarantees do hold. However, the differences are not as pronounced in practice as in theory, and readers whose interests are purely practical may skip this section without affecting their understanding of the rest of the article.

Lemmata 3 and 5 were concerned with the relationship between the insertion radii of a child and its parent. The next lemma is concerned with the relationship between $\text{lfs}(v)/r_v$ and $\text{lfs}(p)/r_p$. For any vertex v, define $D_v = \text{lfs}(v)/r_v$. Think of D_v as the one-dimensional density of vertices near v when v is inserted, weighted by the local feature size. One would like this density to be as small as possible. $D_v \leq 1$ for any input vertex, but D_v tends to be larger for a vertex inserted late.

Lemma 6 *Let v be a vertex with parent $p = p(v)$. Suppose that $r_v \geq Cr_p$ (following Lemmata 3 and 5). Then $D_v \leq 1 + D_p/C$.*

Proof: By Lemma 2, $\text{lfs}(v) \leq \text{lfs}(p) + |vp|$. By definition, the insertion radius r_v is $|vp|$ if p is a mesh vertex, whereas if p is a rejected circumcenter, then $r_v \geq |vp|$. Hence, we have

$$\begin{aligned}\text{lfs}(v) &\leq \text{lfs}(p) + r_v \\ &= D_p r_p + r_v \\ &\leq \frac{D_p}{C} r_v + r_v.\end{aligned}$$

The result follows by dividing both sides by r_v. ■

Let's consider Ruppert's algorithm first, and then compare Chew's.

Lemma 7 (Ruppert [27]) *Consider a mesh produced by Ruppert's algorithm. Suppose the quality bound B is strictly larger than $\sqrt{2}$, and the smallest angle between two incident segments in the input PSLG is strictly greater than $60°$. There exist fixed constants $D_T \geq 1$ and $D_S \geq 1$ such that, for any vertex v inserted (or considered for insertion and rejected) at the circumcenter of a skinny triangle, $D_v \leq D_T$, and for any vertex v inserted at the midpoint of an encroached subsegment, $D_v \leq D_S$. Hence, the insertion radius of every vertex has a lower bound proportional to its local feature size.*

Proof: Consider any non-input vertex v with parent $p = p(v)$. If p is an input vertex, then $D_p = \text{lfs}(p)/r_p \leq 1$. Otherwise, assume for the sake of induction that the lemma is true for p, so that $D_p \leq D_T$ if p is a circumcenter, and $D_p \leq D_S$ if p is a midpoint. Hence, $D_p \leq \max\{D_T, D_S\}$.

First, suppose v is inserted or considered for insertion at the circumcenter of a skinny triangle. By Lemma 3, $r_v \geq Br_p$. Thus, by Lemma 6, $D_v \leq 1 + \max\{D_T, D_S\}/B$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen so that

$$1 + \frac{\max\{D_T, D_S\}}{B} \leq D_T. \tag{1}$$

Second, suppose v is inserted at the midpoint of a subsegment s. If its parent p is an input vertex or lies on a segment not incident to s, then $\text{lfs}(v) \leq r_v$, and the theorem holds. If p is the circumcenter of a skinny triangle (considered for insertion but rejected because it encroaches upon s), $r_v \geq r_p/\sqrt{2}$ by Lemma 3, so by Lemma 6, $D_v \leq 1 + \sqrt{2}D_T$.

Alternatively, if p, like v, is a segment vertex, and p and v lie on incident segments, then $r_v \geq r_p/(2\cos\alpha)$ by Lemma 3, and thus by Lemma 6, $D_v \leq 1 + 2D_S\cos\alpha$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen so that

$$1 + \sqrt{2}D_T \leq D_S, \quad \text{and} \tag{2}$$

$$1 + 2D_S\cos\alpha \leq D_S. \tag{3}$$

If the quality bound B is strictly larger than $\sqrt{2}$, Inequalities (1) and (2) are simultaneously satisfied by choosing

$$D_T = \frac{B+1}{B-\sqrt{2}}, \qquad D_S = \frac{(1+\sqrt{2})B}{B-\sqrt{2}}.$$

If the smallest input angle $\alpha_{\min}$ is strictly greater than 60°, Inequalities (3) and (1) are satisfied by choosing

$$D_S = \frac{1}{1-2\cos\alpha_{\min}}, \qquad D_T = 1 + \frac{D_S}{B}.$$

One of these choices will dominate, depending on the values of B and $\alpha_{\min}$. In either case, if $B > \sqrt{2}$ and $\alpha_{\min} > 60°$, there are values of D_T and D_S that satisfy all the inequalities. ∎

Note that as B approaches $\sqrt{2}$ or α approaches 60°, D_T and D_S approach infinity. In practice, the algorithm is better behaved than the theoretical bound suggests; the vertex density approaches infinity only after B drops below one.

Theorem 8 (Ruppert [27]) *For any vertex v of the output mesh, the distance to its nearest neighbor w is at least* $\text{lfs}(v)/(D_S+1)$.

Proof: Inequality (2) indicates that $D_S > D_T$, so Lemma 7 shows that $\text{lfs}(v)/r_v \le D_S$ for any vertex v. If v was added after w, then the distance between the two vertices is $r_v \ge \text{lfs}(v)/D_S$, and the theorem holds. If w was added after v, apply the lemma to w, yielding

$$|vw| \ge r_w \ge \frac{\text{lfs}(w)}{D_S}.$$

By Lemma 2, $\text{lfs}(w) + |vw| \ge \text{lfs}(v)$, so

$$|vw| \ge \frac{\text{lfs}(v) - |vw|}{D_S}.$$

It follows that $|vw| \ge \text{lfs}(v)/(D_S+1)$. ∎

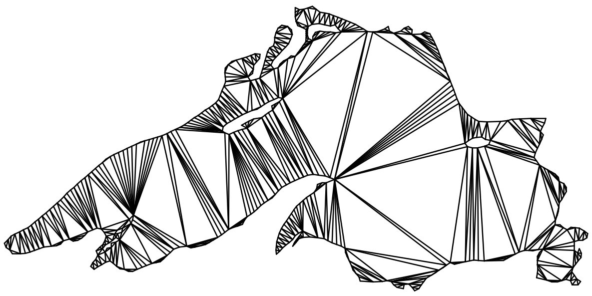
To give a specific example, consider triangulating a PSLG (having no acute input angles) so that no angle of the output mesh is smaller than 15°; hence $B \doteq 1.93$. For this choice of B, $D_T \doteq 5.66$ and $D_S \doteq 9.01$. Hence, the spacing of vertices is at worst about ten times smaller than the local feature size. Away from boundaries, the spacing of vertices is at worst seven times smaller than the local feature size.

Figure 18 illustrates the algorithm's grading for a variety of angle bounds. Ruppert's algorithm typically terminates for angle bounds much higher than the theoretically guaranteed 20.7°, and typically exhibits much better vertex spacing than the provable worst-case bounds imply.

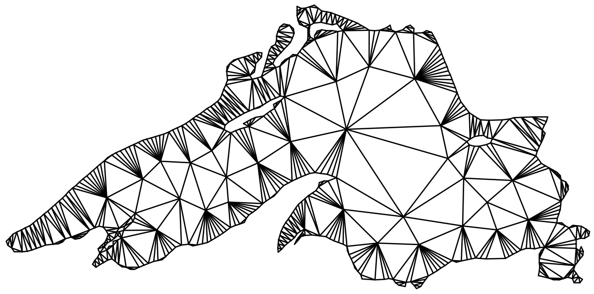
Let's compare Chew's algorithm.

Lemma 9 *Consider a mesh produced by Chew's algorithm. Suppose the quality bound B is strictly larger than $\sqrt{5}/2$, and the smallest angle between two incident segments in the input PSLG is strictly greater than 60°. There exist fixed constants $D_T \ge 1$ and $D_S \ge 1$ such that, for any vertex v inserted at the circumcenter of a skinny triangle, $D_v \le D_T$, and for any vertex v inserted at the midpoint of an encroached subsegment, $D_v \le D_S$.*

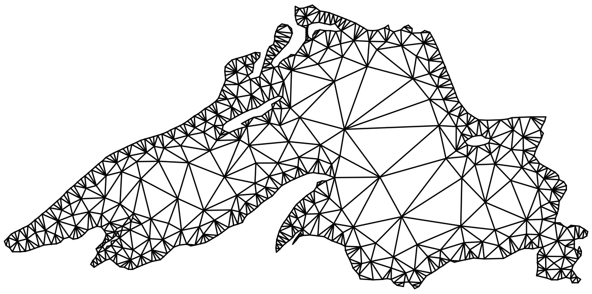
Lake Superior PSLG.



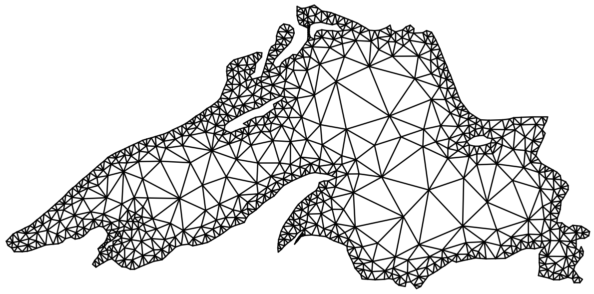
Triangulated with no minimum angle.



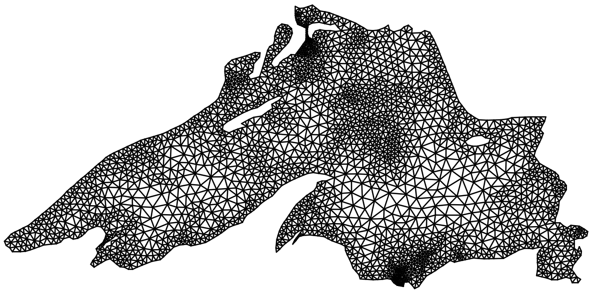
Triangulated with 5° minimum angle.



Triangulated with 15° minimum angle.



Triangulated with 25° minimum angle.



Triangulated with 34.2° minimum angle.

Figure 18: Meshes generated with Ruppert's algorithm for several different angle bounds. The algorithm does not terminate for angle bounds of 34.3° or higher on this PSLG.

Proof: Essentially the same as the proof of Lemma 7, except that Lemma 5 makes it possible to replace Inequality (2) with

$$\begin{aligned} D_S &\geq 1 + \frac{D_T}{\cos\theta} \\ &\geq 1 + \frac{2BD_T}{\sqrt{4B^2-1}} \end{aligned} \tag{4}$$

If the quality bound B is strictly larger than $\sqrt{5}/2$, Inequalities (1) and (4) are simultaneously satisfied by choosing

$$D_T = \frac{\left(1+\frac{1}{B}\right)\sqrt{4B^2-1}}{\sqrt{4B^2-1}-2}, \qquad D_S = \frac{\sqrt{4B^2-1}+2B}{\sqrt{4B^2-1}-2}.$$

D_T and D_S must also satisfy Inequality (3), so larger values of D_T and D_S may be needed, as in Lemma 7. However, If $B > \sqrt{5}/2$ and $\alpha_{\min} > 60^\circ$, there are values of D_T and D_S that satisfy all the inequalities. ■

Theorem 8 applies to Chew's algorithm as well as Ruppert's, but the values of D_T and D_S are different. Again consider triangulating a PSLG (free of acute angles) so that no angle of the output mesh is smaller than 15°. Then $D_T \doteq 3.27$, and $D_S \doteq 4.39$, compared to the corresponding values of 5.66 and 9.01 for Ruppert's algorithm. Hence, the spacing of vertices is at worst a little more than five times the local feature size, and a little more than four times the local feature size away from segments.

Ruppert [27] uses Theorem 8 to prove the size optimality of the meshes his algorithm generates, and his result has been improved by Mitchell [24]. Mitchell's theorem is stated below, but the lengthy proof is omitted.

Theorem 10 (Mitchell [24]) *Let $\mathrm{lfs}_T(p)$ be the local feature size at p with respect to a triangulation T (treating T as a PSLG), whereas $\mathrm{lfs}(p)$ remains the local feature size at p with respect to the input PSLG. Suppose a triangulation T with smallest angle θ has the property that there is some constant $k_1 \geq 1$ such that for every point p, $k_1 \mathrm{lfs}_T(p) \geq \mathrm{lfs}(p)$. Then the cardinality (number of triangles) of T is less than k_2 times the cardinality of any other triangulation of the input PSLG with smallest angle θ, where $k_2 \in \mathcal{O}(k_1^2/\theta)$.* ■

Theorem 8 can be used to show that the precondition of Theorem 10 is satisfied by meshes generated by Ruppert's and Chew's algorithms (with $k_1 \propto D_S$). Hence, the cardinality of a mesh generated by either algorithm is within a constant factor of the cardinality of the best possible mesh satisfying the angle bound. However, the constant factor hidden in Mitchell's theorem is much too large to be a meaningful guarantee in practice.

Because the worst-case number of triangles is proportional to the square of D_S, Chew's algorithm is size-optimal with a constant of optimality almost four times better than Ruppert's algorithm. However, worst-case behavior is never seen in practice, and the observed difference between the two algorithms is less dramatic.

5 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 imposes a fundamental limitation on any triangular mesh generation algorithm.

The result holds for certain PSLGs that have an angle much smaller than θ. Of course, one must respect the PSLG; small input angles cannot be removed. However, one would like to believe that it is possible to triangulate a PSLG without creating any small angles that aren't already present in the input. Unfortunately, no algorithm can make this guarantee for all PSLGs.

The reasoning behind the claim is as follows. Suppose two collinear subsegments of lengths a and b share a common endpoint, as illustrated in Figure 19. Mitchell [24] 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 $(2\cos\theta)^{180^\circ/\theta}$. (This bound is tight if $180^\circ/\theta$ is an integer. Figure 19 offers an example where the bound is obtained.) Hence, any bound on the smallest angle of a triangulation imposes a limit on the grading of triangle sizes.

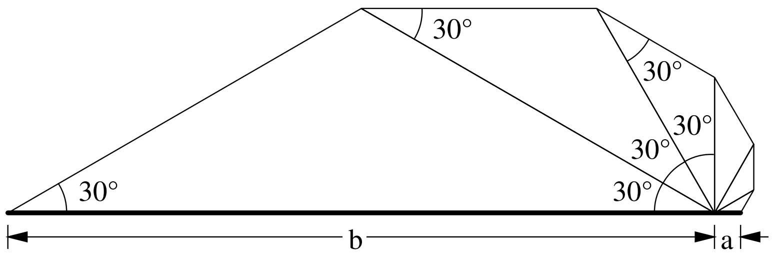


Figure 19: In any triangulation with no angle smaller than $30°$, the ratio b/a cannot exceed 27.

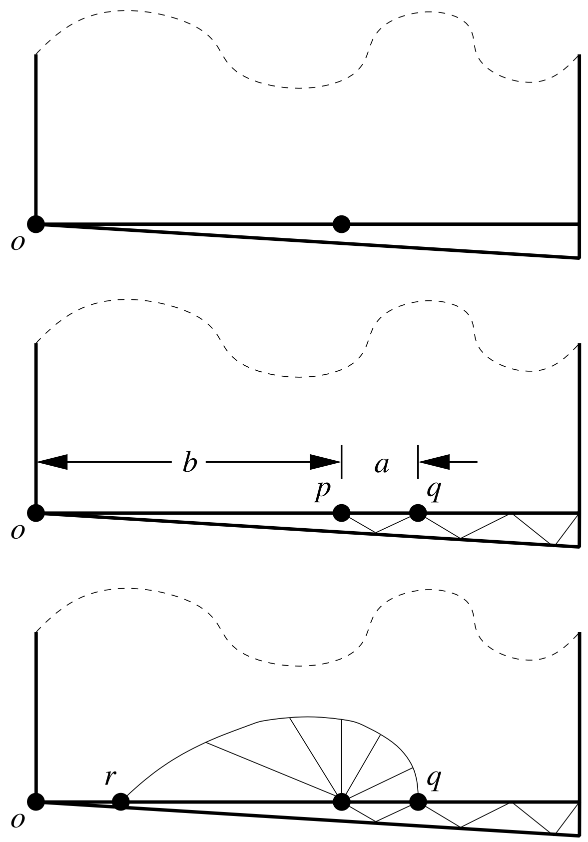


Figure 20: Top: A difficult PSLG with a small interior angle ϕ. Center: The vertex p and the angle constraint necessitate the placement of the vertex q. Bottom: The vertex q and the angle constraint necessitate the placement of the vertex r. The process repeats eternally.

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 20 (top) illustrates this circumstance. Suppose that the middle segment of the three has been split into two segments by a vertex in the middle.

Assume for the sake of contradiction that there is a triangulation T that respects this domain and has no angle smaller than θ, except for the one small input angle at o. Let p be the vertex of T that lies on the middle segment closest to o, so that po is an edge of T. The placement of p forces the narrow wedge between the first two segments to be triangulated (Figure 20, center), which may necessitate the placement of another vertex q on the middle segment. Let $a = |pq|$ and $b = |op|$ as illustrated. If the angle bound is

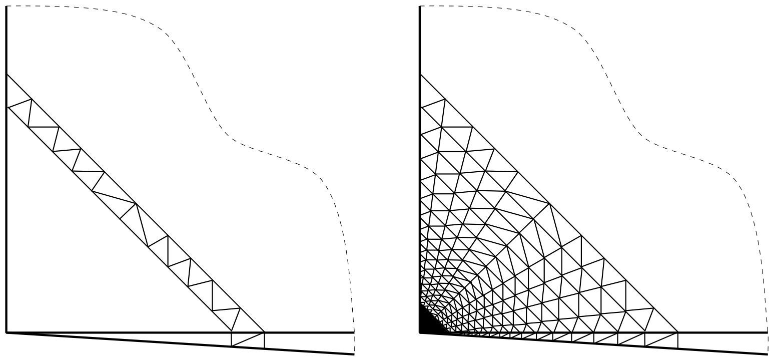


Figure 21: How to create a quality triangulation of infinite cardinality around the apex of a very small angle. The method employs a thin strip of good-quality triangles about the vertex (left). Ever-smaller copies of the strip fill the gap between the vertex and the outer strip (right).

respected, the length a is small; one can show that

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

If the upper region (above the wedge) is part of the triangulation domain, then it too must be triangulated, perhaps with the fan-like triangulation in Figure 19. Because T has no angle less than θ in the upper region, $b/a \leq (2\cos\theta)^{180°/\theta}$. However, if the product of these bounds on b/a and a/b is less than one, there are no values of a and b that satisfy both inequalities. For any choice of $\theta > 0$, there is a choice of $\phi > 0$ for which the product of these bounds is less than one. Hence, by contradiction, there is no triangulation of such a domain in which no new angle is smaller than θ.

For an angle constraint of $\theta = 30°$, the bounds are incompatible when ϕ is about six tenths of a degree or smaller. In practice, Delaunay refinement often fails to achieve a 30° angle bound for $\phi = 5°$. A Delaunay refinement algorithm, attempting to triangulate the upper region, will introduce another vertex r between o and p (Figure 20, bottom). Unfortunately, the vertex r creates the same conditions as the vertex p in the lower region, but r is closer to o. The process will cascade, eternally creating smaller and smaller triangles in an attempt to satisfy the angle constraint, and the algorithm will never terminate.

Oddly, it is straightforward to triangulate this PSLG using an infinite number of good-quality triangles. A vertex at the apex of a small angle can be shielded with a thin strip of good-quality triangles, as Figure 21 illustrates. (This idea is one form of corner-lopping, discussed in the next section.) The strip is narrow enough to admit 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.

6 Practical Handling of Small Input Angles

Ruppert's algorithm fails to terminate on PSLGs like that of Figure 20, even if the algorithm is modified so that it does not try to split any skinny triangle that bears a small input angle. As the previous section demonstrates, any mesh of such a PSLG has a small angle that is removable, but another small angle invariably takes its place. The negative result quashes all hope of finding a magic pill that will make it possible

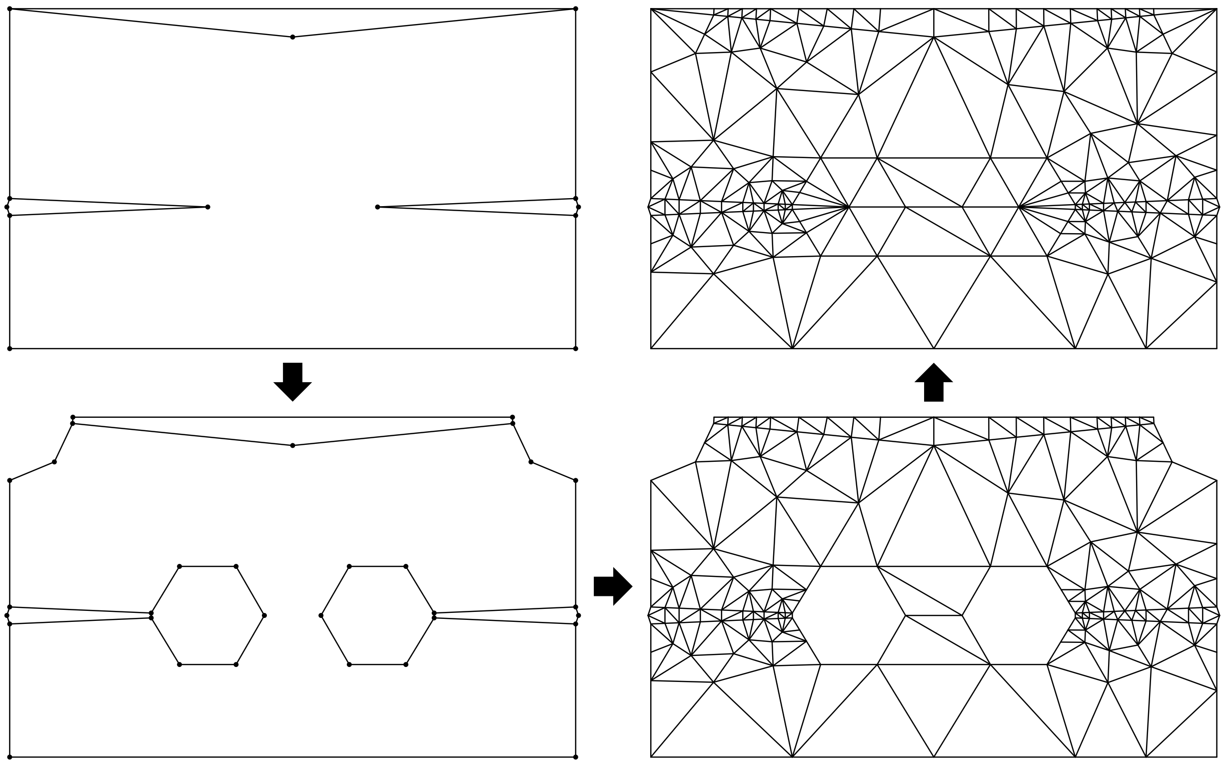


Figure 22: A PSLG with small angles (upper left) can be triangulated by lopping off the corners where small angles appear (lower left), triangulating the modified PSLG (lower right), and filling the lopped corners with fans of triangles (upper right).

to triangulate any PSLG without introducing additional small angles. However, 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?

The following sections compare a traditional solution, corner-lopping, with a new algorithm that generates better meshes in practice.

6.1 Corner-Lopping

Corner-lopping is an approach proposed by Bern, Eppstein, and Gilbert [4] and Ruppert [27]. The input PSLG X is replaced by a modified PSLG X', as Figure 22 illustrates.

X' is created by lopping off corners of X where small angles lie. Any apex a of a small angle (less than $60°$) in X is "shielded" as follows. Imagine a circle of some suitable radius centered at a. A radius of $\text{lfs}(a)/3$, where $\text{lfs}(\cdot)$ is defined relative to the original PSLG X, is a reasonable choice. Vertices are inserted wherever the circle intersects a segment, thereby dividing the circle into arcs. If any of the arcs subtend an angle over $60°$ in the interior of the triangulation domain, these arcs are further divided by the insertion of additional vertices. These vertices define a convex polygon (or portion thereof) around a. The edges of this polygon become segments of X' (which Ruppert calls *shield edges*), and the contents of this polygon are removed from the triangulation domain. Hence, X' does not contain a, and the segments that were incident on a in X are shortened in X'.

The modified PSLG X' has no small interior angles and can be triangulated using Ruppert's algorithm (as in the lower right corner of Figure 22) or Chew's algorithm. This mesh is converted into a triangulation of the original PSLG X by filling each lopped corner with a fan of triangles around each apex (adding what Ruppert calls *spoke edges*), as illustrated in the upper right corner of Figure 22. This approach has the advantage that a new small angle can appear only at the apex of a small input angle. Ruppert [27] outlines some clever suggestions for triangulating the lopped corners with fewer new small angles. However, the negative result of Section 5 states that new small angles cannot always be avoided entirely.

An advantage of this approach is that if the minimum angle bound is $\theta \leq 30^\circ$, no angle larger than $180^\circ - 2\theta$ appears in the final mesh. Another advantage is that, if the radii of the circles are chosen carefully, corner-lopping can be implemented so that no new angle of the final mesh is smaller than the smallest input angle. (A proof of this fact would exploit the lower bounds on edge lengths given by Theorems 4 and 8, but would probably require the lopping circles to have smaller radii than $\mathrm{lfs}/3$.)

Corner-lopping is simple in concept, but somewhat troublesome to implement well. One complication is choosing the radius of the circle around each apex so as to produce the best possible mesh. A second complication is how best to triangulate the lopped corners. To mesh them using only spoke edges gives inferior results.

The main disadvantage of corner-lopping, however, is its tendency to reduce features to a third of their original lengths. The next several sections consider an alternative approach. It is conceptually more complicated than corner-lopping, but it is no more difficult to implement, it is just as theoretically sound, and as Figure 23 illustrates, it generally yields better meshes in practice. Observe, for instance, the shorter edges at the center of the corner-lopped meshes (left), and the larger number of small angles in the upper left mesh. In PSLGs where angles are in an intermediate range (about 10°–60°), corner-lopping lops corners that might not have caused any difficulty if they had not been lopped. As we will see, the alternative approach works well in these ambiguous cases, often leaving behind no new small angles, and giving up gracefully without producing unreasonably small edges when it is not successful in eliminating all new small angles.

6.2 Guaranteed Termination Without Corner-Lopping

Figure 24 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-shorter subsegments incident to the apex of the small angle. This phenomenon is only observed with angles of 45° or less. In these cases, the lower right cycle in the flow graph has a multiplier of $1/\sqrt{2}$ or less, and a single pair of segments causes the cycle to realize its threat of an infinite loop in the algorithm.

To solve this problem, Ruppert [27] 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 25. When an encroached subsegment has an endpoint that is 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, choose one endpoint's shells arbitrarily. The shell that gives the best-balanced split is chosen; in the worst case, the shorter 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-shorter subsegments portrayed in Figure 24, because incident subsegments of equal length do not encroach upon each other. It tends to simulate the effects of corner-lopping, but in a lazy manner so that corners are not lopped until necessary.

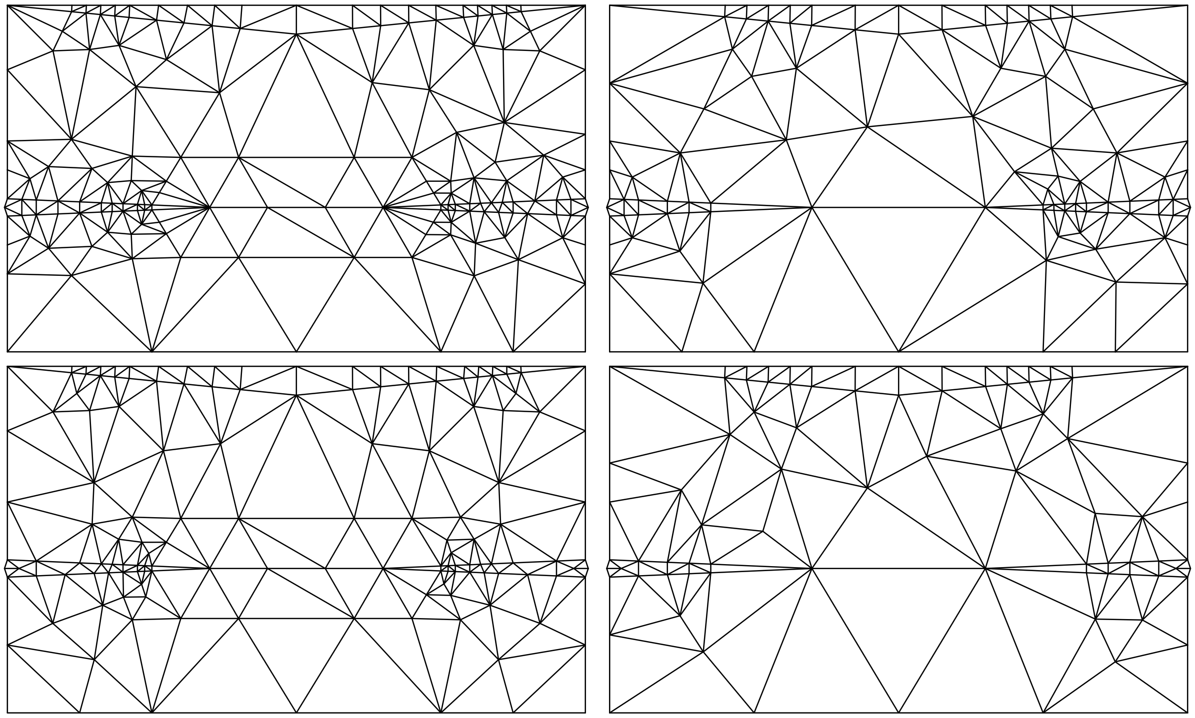


Figure 23: Upper left: a 291-triangle mesh created by Ruppert's algorithm with corner-lopping. Upper right: a 180-triangle mesh created by the Terminator, discussed in Section 6.3. Lower left: a 246-triangle mesh created by Chew's algorithm with corner-lopping. Lower right: a 136-triangle mesh created by the Terminator with Chew-inspired diametral lenses, discussed in Section 6.4.

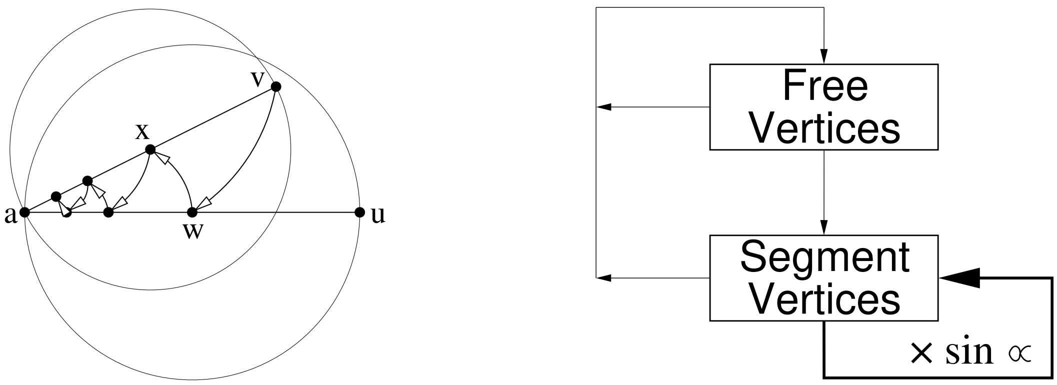


Figure 24: Left: A problem caused by a small input angle. Vertex v encroaches upon au, which is split at w. Vertex w encroaches upon av, which is split at x. Vertex x encroaches upon aw, and so on. Right: The corresponding cycle in the flow graph is shown in bold.

If Ruppert's algorithm is modified slightly so that it does not attempt to split a skinny triangle that bears a small input angle (and thus cannot be improved), concentric shell segment splitting is generally effective in practice for PSLGs that have small angles greater than about 10°, and often for smaller angles. (See Figures 1 and 25 for examples.) It is always effective for polygons (with polygonal holes). In effect, the lower right cycle in the flow graph (Figure 24, right) can no longer, *by itself*, cause the algorithm to fail to terminate.

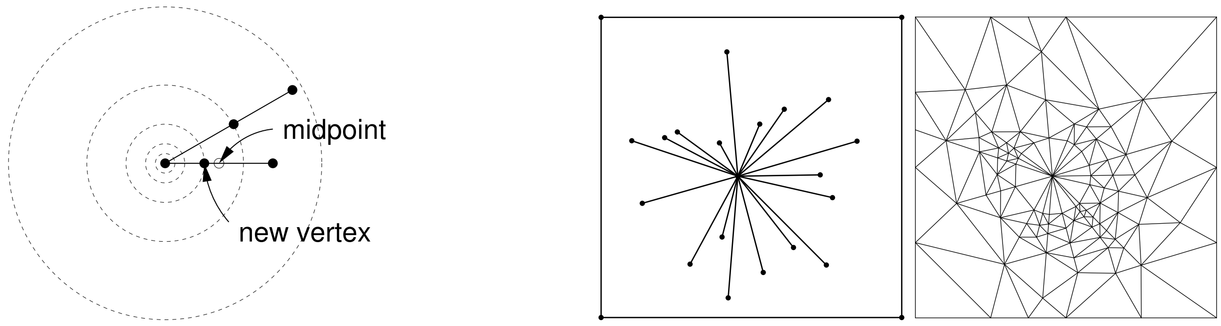


Figure 25: Left: "Modified segment splitting using concentric circular shells." If an encroached subsegment has a shared input vertex for an endpoint, the subsegment is split at its intersection with a circular shell whose radius is a power of two. Right: Sample input and output of Ruppert's algorithm with concentric shell segment splitting.

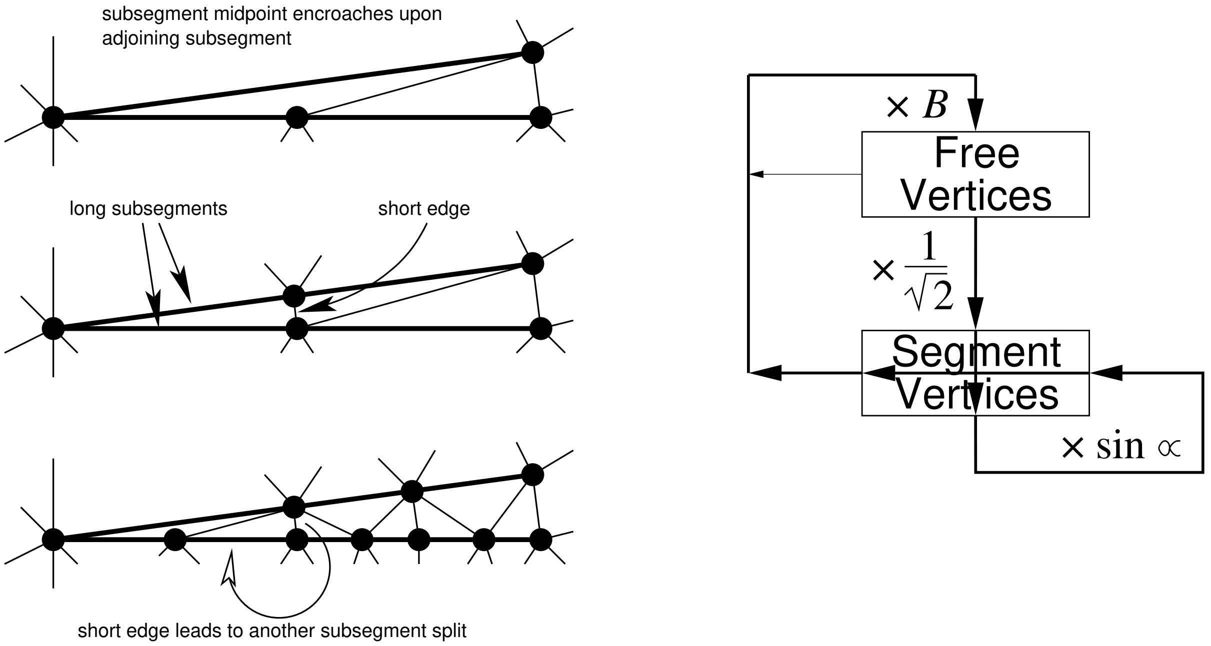


Figure 26: Left: The short edge created opposite a small angle can cause other short edges to be created as the algorithm attempts to remove skinny triangles. If the small insertion radii propagate around an endpoint and cause the supporting subsegments to be split, a shorter edge is created, and the cycle may repeat. Right: A corresponding cycle in the flow graph is shown in bold.

However, the flow graph may contain another cycle whose product is less than one: a figure-eight loop through the flow graph (Figure 26, right) in which a segment vertex sires another segment vertex (via a small input angle), which sires a free vertex (at the circumcenter of a skinny triangle), which sires another segment vertex. As Section 5 demonstrates, difficulties occur when a small angle is adjacent to a much larger angle. The negative result of Section 5 arises not because of the mutual encroachment problem illustrated in Figure 24, but because the free edge opposite a small angle is much shorter than the subsegments that define the angle, as Figure 26 (left) illustrates.

The two subsegments in Figure 26 are coupled, in the sense that if one is bisected then so is the other, because the midpoint of one encroaches upon the other. In Ruppert's algorithm, this holds true for any two

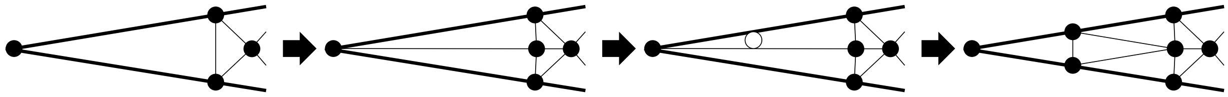


Figure 27: Concentric shell segment splitting ensures that polygons (with holes) can be triangulated, because it causes small angles to be clipped off. This sequence of illustrations demonstrate that if a clipped triangle's interior edge is breached, a smaller clipped triangle will result.

subsegments of equal length separated by $60°$ or less. 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 coupled subsegments may be split again, continuing an infinite sequence of shorter and shorter edges.

If the PSLG is a polygon (optionally with polygonal holes), small insertion radii cannot propagate around the short edge, because the short edge partitions the polygon into a skinny triangle (which the algorithm does not attempt to split) and everything else. The short edge is itself flipped or penetrated only if there is an even smaller feature elsewhere in the mesh. If the short edge is thus removed, the algorithm will attempt to fix the two skinny triangles that result, thereby causing the subsegments to be split again, thus creating a new shorter edge (Figure 27).

For general PSLGs, how may one diagnose and cure cycles of diminishing edge lengths? A sure-fire way to guarantee termination is to never create an edge shorter than $\mathrm{lfs}_{\min}$. Here's an example of a strategy that achieves this goal while still accounting for mesh grading: 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.

This cure works by directly attacking the lower right cycle in the flow graph of Figure 26. When the cycle's multiplier is smaller than one, the child vertex is simply not inserted. The precise rule is that if a subsegment is encroached upon by a segment vertex, and the new vertex inserted in the encroached subsegment would have a smaller insertion radius than the encroaching vertex, do not split the encroached subsegment.

This restriction is undesirably conservative for two reasons. The first problem is demonstrated in Figure 28. 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 other skinny triangles may form. The skinny triangles in the figure cannot be improved without splitting the upper subsegment.

Second, if subsegments remain encroached, the mesh is unsuitable for applications that require the final triangulation to be truly Delaunay, or that (like some finite volume methods) require all triangle circumcenters to lie in the triangulation.

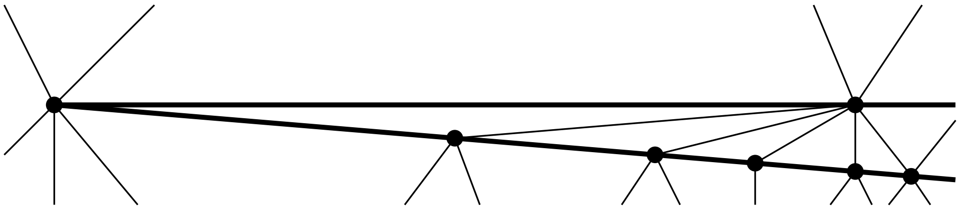


Figure 28: The simplest method of ensuring termination when small input angles are present has undesirable properties, including the production of large angles and many small angles.

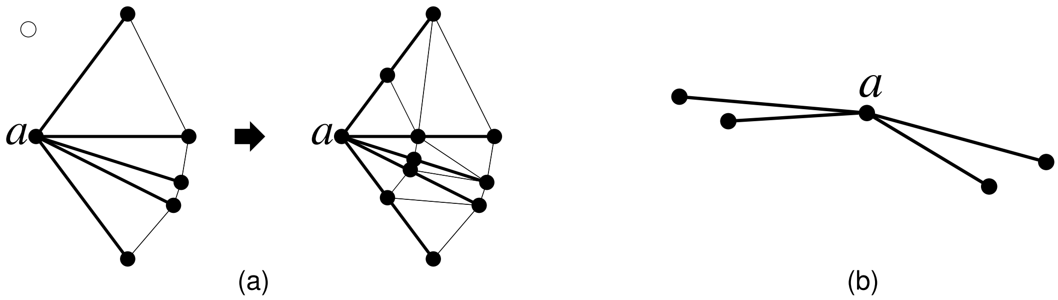


Figure 29: (a) Example of a subsegment cluster. If all the subsegments of a cluster have power-of-two lengths, then they all have the same length and are effectively split as a unit because of mutual encroachment. (b) Several independent subsegment clusters may share the same apex.

6.3 A Better Algorithm: The Terminator

As an alternative, I suggest a Delaunay refinement algorithm called the *Terminator*, which produces fewer large angles in practice than the simple solution described above, and offers guaranteed termination and a guaranteed minimum angle.

The Terminator is based on Delaunay refinement with concentric shell segment splitting and a look-ahead procedure that prevents unsafe subsegment splits. 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 of 60° or less, or if both endpoints do, then s is split. (If every input angle is greater 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 60° or less. (However, "exterior" small angles are ignored; two subsegments are not necessarily part of the same cluster if the region between them is not part of the triangulation domain.) 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 29(a).

Not all subsegments incident to an input vertex are necessarily part of the same cluster. For instance, Figure 29(b) shows two independent subsegment clusters sharing one apex, separated from each other by angles over 60°.

Let t be a skinny or oversized triangle whose circumcenter encroaches upon a subsegment s that belongs to a subsegment cluster. Let v be the vertex that will be introduced if s is split. Let g be v's grandparent (and the parent of t's circumcenter). The Terminator determines the insertion radius r_g of g and the minimum insertion radius $r_{\min}$ of all the subsegment vertices (including v) that will be introduced into the subsegment cluster of s if all the subsegments in the cluster are split to a length of $|s|/2$. This radius is the length of the shortest free edge that will appear opposite a small angle of the subsegment cluster, so $r_{\min} = |s|\sin(\phi_{\min}/2)$, where $\phi_{\min}$ is the smallest angle separating two subsegments of the cluster.

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

A. $r_{\min} \geq r_g$;

B. the triangle t is too large (according to some user-defined criterion);

C. the length of s is not a power of two;

D. s has two subsegment clusters—one at each endpoint;

E. some subsegment in the subsegment cluster of s is shorter than s; or

F. 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.)

Condition A permits subsegment splits that do not engender shorter edges than the existing edges. The remaining five conditions are optional heuristics that help improve the mesh by allowing shorter edges to be created in specific circumstances, without threatening the termination guarantee. Condition B ensures that all oversized triangles are eliminated. Conditions C, D, and E help to create isosceles triangles (which are the best we can hope for) at small input angles. Condition F 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. Any of the last five conditions can be omitted, but C, D, and E are strongly recommended. Be aware that Condition F may significantly slow down the algorithm if the chains of vertex ancestors grow long.

If the Terminator declines to insert v, and no subsegment is split because of encroachment by the circumcenter of t, the Terminator never again tries to split t. See Appendix A for details.

Theorem 11 *The Terminator always terminates.*

Proof: If $r_{\min} < r_g$, but v is inserted in s anyway because one of the other five 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 $r_{\min}$ is less than r_g. A trigger vertex is named according to the condition that allows its insertion; for example, a *Type C trigger vertex* is inserted because the length of s is not a power of two. Recall that the parent of any trigger vertex is a rejected circumcenter of a skinny or oversized triangle, and bad 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.

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 3 that if v is a diminishing vertex, v must have been inserted in a subsegment s in one of the following circumstances.

1. v is not the midpoint of s, because s was split using concentric circular shells;
2. 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

3. s is encroached upon by a vertex p that lies on a segment incident to s at an angle less than $60°$. Clearly, the insertion of v was preceded by the insertion of either a trigger vertex or one of the above two types of diminishing vertices in the same subsegment cluster.

Call these Type 1, 2, and 3 diminishing vertices, respectively.

The rest of the proof will show that only a finite number of Types B–F trigger vertices and Types 1–3 diminishing vertices can be inserted. After the last diminishing vertex is inserted, no edge ever appears whose length is less than the minimum insertion radius of all the diminishing vertices and input vertices. Because this establishes a lower bound on edge lengths, the Terminator terminates; it must eventually run out of places to put new vertices.

The total number of diminishing vertices of Type 1, plus trigger vertices of Type C or D, is at most twice the number of segments in the PSLG. After a segment undergoes two off-center splits, the subsegments at the ends of the segment have power-of-two lengths.

Only a finite number of Type 2 diminishing vertices can be inserted, because any subsegment shorter than $\mathrm{lfs}_{\min}$ cannot be encroached upon by a nonincident feature.

By Lemma 3, a Type B trigger vertex cannot split a subsegment to a length shorter than the circumradius of the bad triangle divided by $\sqrt{2}$. Any reasonable definition of "oversized" applies only to triangles whose circumradii are above some positive threshold. Hence, once the subsegments in every cluster are sufficiently short, no more Type B trigger vertices can be inserted.

No Type F 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 F trigger vertex for each input vertex.

Finally, we come to Type 3 diminishing vertices and Type E trigger vertices, which are inserted into a subsegment only if there is some shorter subsegment in the same cluster. How did that subsegment get to be shorter? Consider a subsegment cluster whose shortest original segment has a length of at least 2^{i+2} for some integer i. Thanks to concentric shell segment splitting, once a subsegment in this cluster has been split to a length of 2^i or smaller, its length is a power of two. If a subsegment of this cluster whose length is $\ell \leq 2^i$ is split by the insertion of a Type 3 or Type E vertex, the cluster must have already contained a subsegment of length $\ell/2$. Therefore, a subsegment that is shorter than the shortest subsegment in the same cluster (and shorter than 2^i) cannot be created by inserting a Type 3 or Type E vertex.

It follows that only a finite number of Type 3 diminishing vertices and Type E trigger vertices can be inserted for each diminishing vertex or trigger vertex of another type inserted. Therefore, the Terminator inserts only a finite number of diminishing vertices, and it is guaranteed to terminate. ∎

Theorem 12 *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 30. t is not split because its circumcenter p encroaches upon a subsegment s whose splitting would have split a subsegment cluster, yielding a short 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

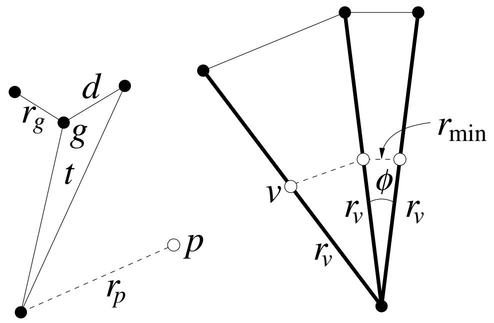


Figure 30: If the Terminator does not split t, the smallest angle of t is not much smaller than ϕ.

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 3, $r_p \leq \sqrt{2}r_v$. If the subsegment cluster of s had been split, every resulting subsegment would have length r_v. If the smallest angle of the subsegment 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, and is suitable for applications that require the final mesh to be Delaunay and have every triangle's circumcenter in the mesh. Because subsegments are not encroached, an angle near 180° cannot appear immediately opposite a subsegment (as in Figure 28), 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°. What does "near" mean? Every skinny triangle's circumcenter encroaches upon a subsegment cluster bearing a small input angle. Because $\arcsin[\sin(\phi/2)/\sqrt{2}] \sim \phi/(2\sqrt{2})$ for small ϕ, the smallest angle in the final mesh cannot be smaller than the smallest input angle by a factor of much more than $2\sqrt{2}$. This bound is not quite as good as the bound of ϕ that corner-lopping, carefully implemented, can achieve. However, the Terminator performs better than corner-lopping in practice, and angles smaller than ϕ, while theoretically possible, have not been observed.

6.4 Terminator Chew

Chew's algorithm and the Terminator can be combined straightforwardly, as each of them can be viewed as a modification to Ruppert's algorithm. However, the combined algorithm is unsatisfactory. Suppose that a segment s is encroached, but the Chew–Terminator combination declines to split s, following the rules of Section 6.3. Should the free vertices inside the diametral circle of s be deleted anyway? If they are deleted, the combination algorithm may fail to terminate, because vertices are eternally created and deleted in the diametral circle; see Figure 31 for an example. If they are not deleted, arbitrarily skinny triangles may survive, like t in Figure 31. The bound on the smallest angle established by Theorem 12 does not apply to the Chew–Terminator because it is not possible to bound the circumradius of the worst triangle. For example, the circumradius of t in Figure 31 can be arbitrarily large. (The Ruppert–Terminator's bound on circumradius, from Theorem 12, does not transfer to Chew's algorithm because Lemma 5 changes the definition of "parent.")

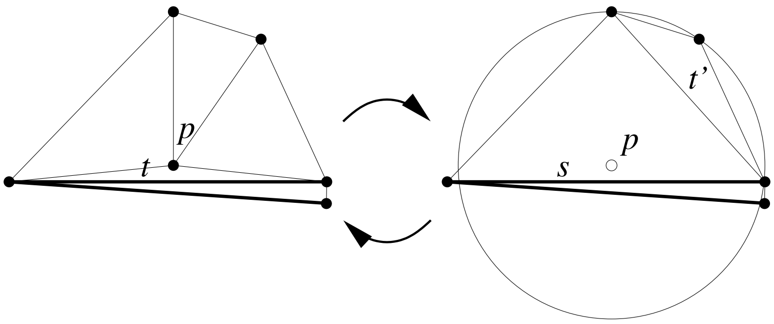


Figure 31: Chew's algorithm wants to split the skinny triangle t, but the circumcenter of t lies on the other side of s, so s is encroached. The Terminator declines to split s, so that a shorter edge is not created. Should vertices inside the diametral circle of s be deleted? If not, an arbitrarily bad triangle t will survive. If the vertices are deleted, the algorithm may fail to terminate. In this example, when t' is split, the triangle t is recreated, and an infinite loop occurs.

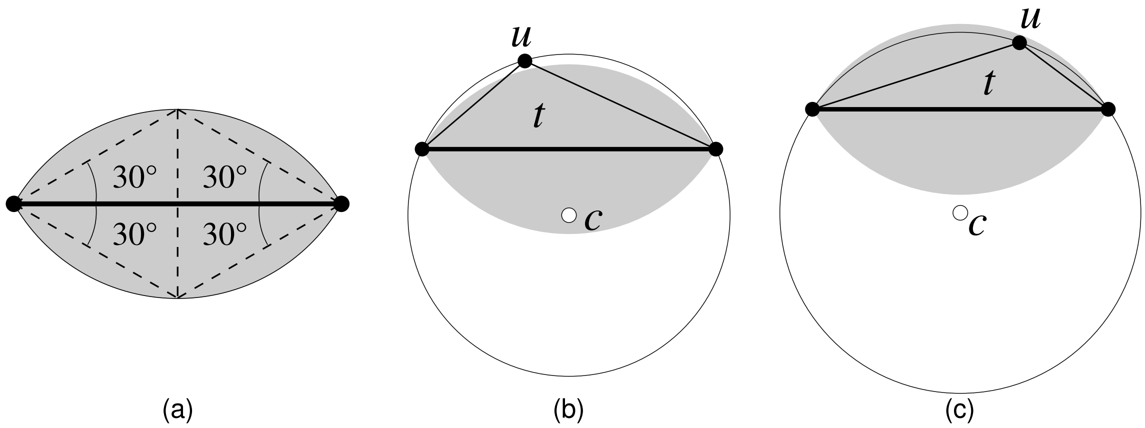


Figure 32: (a) A hybrid Chew–Ruppert algorithm uses diametral lenses. Any vertex in the shaded region encroaches upon this subsegment. (b, c) If a skinny triangle and its circumcenter lie on opposite sides of a subsegment, then either the circumcenter of or a vertex of the triangle lies in the subsegment's diametral lens.

To rescue the combination algorithm, change the definition of encroachment. Replace Ruppert's diametral circles with *diametral lenses*, illustrated in Figure 32(a). The diametral lens of a subsegment s is the intersection of two disks whose centers lie on the bisector of s, with one center on each side of s. Each disk's center is at a distance of $|s|/(2\sqrt{3})$ from s, where $|s|$ is the length of s. Each disk has radius $|s|/\sqrt{3}$, so each disk's boundary passes through the endpoints of s and the center of the other disk. As Figure 32(a) shows, a diametral lens is just big enough to accommodate an isosceles triangle whose base is s and whose angles at the base are $30°$.

Much like in Ruppert's algorithm, a subsegment s is encroached if there is a vertex, or if a vertex is rejected for insertion, in the diametral lens of s (including its boundary), unless another subsegment obstructs the visibility of the encroaching vertex from the interior of s. A vertex considered for insertion at the circumcenter of a skinny triangle is rejected if it lies in the diametral lens of a subsegment, and the subsegment is split instead. As in Chew's algorithm, all free vertices that are inside the diametral circle of s and visible from the midpoint of s are deleted before s is bisected.

Any segment encroached in Chew's algorithm also has an encroached diametral lens (but the reverse is not necessarily true); and any segment whose diametral lens is encroached also has an encroached diametral circle (but the reverse is not necessarily true). The latter statement is obvious, but let's examine the former one. Let t be a skinny triangle whose circumcenter c lies on the opposite side of a subsegment s. Let u be any vertex of t that is not an endpoint of s. Obviously, u is on the circumcircle of t. There are two possibilities: either c encroaches upon the diametral lens of s, as in Figure 32(b), or u encroaches upon the diametral lens, as in Figure 32(c).

Because lens-based encroachment lies between Ruppert's and Chew's algorithms in its aggressiveness in splitting subsegments, it is not surprising that its angle bound lies between their bounds as well. Let p be a circumcenter that is rejected because it encroaches upon the diametral lens of a subsegment s, and let v be the vertex inserted at the midpoint of s. If "parent" is defined as for Ruppert's algorithm, p is the parent of v. Because diametral lenses are narrower than diametral circles, a better bound is available for the insertion radius of v than in Ruppert's algorithm. The bound is $r_v \geq r_p \cos 30^\circ = (\sqrt{3}/2)r_p$, and the proof is much like that of the related bound in Lemma 3. The limiting case, where $r_v = (\sqrt{3}/2)r_p$, occurs when p lies on the topmost or bottommost point of the lens depicted in Figure 32(a).

In Figure 17, replace the multiplier "$\times \cos\theta$" with "$\times\sqrt{3}/2$." If no input angle is smaller than 60°, this Ruppert-Chew hybrid terminates if $B \geq 2/\sqrt{3}$, giving an angle bound of up to $\arcsin(\sqrt{3}/4) \doteq 25.65^\circ$, which is not quite as good as the 26.56° bound of Chew's algorithm, but is still better than the 20.7° bound of Ruppert's algorithm.

If a circumcenter p encroaches upon the diametral lens of a subsegment s, but the Terminator declines to split s, then p is rejected (not inserted), but other vertices inside the diametral circle of s are *not* deleted. This way, termination can still be guaranteed, but s cannot take part in an arbitrarily skinny triangle. In Figure 31, the right-hand configuration is preferred.

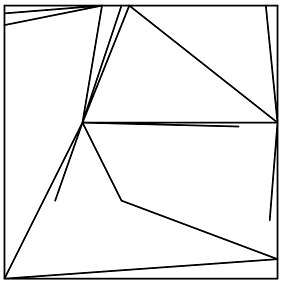
Several other ways in which the Terminator interacts with diametral lenses merit some scrutiny. The definition of "subsegment cluster" in Section 6.3 is motivated in part by the fact that if all the subsegments in a cluster have equal length, and one is split, then all the subsegments are split by mutual encroachment. When diametral lenses replace diametral circles, this statement holds true only if the subsegments are separated by angles of $\arctan\sqrt{13/3} - 30^\circ \doteq 34.34^\circ$ or less. However, subsegment clusters are still defined by a 60° angle, because any angle less than 60° still functions as an "insertion radius reducer."

Theorem 11 (which guarantees that the Terminator terminates on any PSLG) also applies when diametral lenses are used. The bound on the smallest new angle, derived in Theorem 12, can be improved by using the bound $r_v \geq (\sqrt{3}/2)r_p$. If the smallest angle in the input PSLG is $\phi \leq 60^\circ$, the mesh produced by the Terminator with diametral lenses has no angle smaller than $\arcsin[(\sqrt{3}/2)\sin(\phi/2)] \sim (\sqrt{3}/4)\phi$.

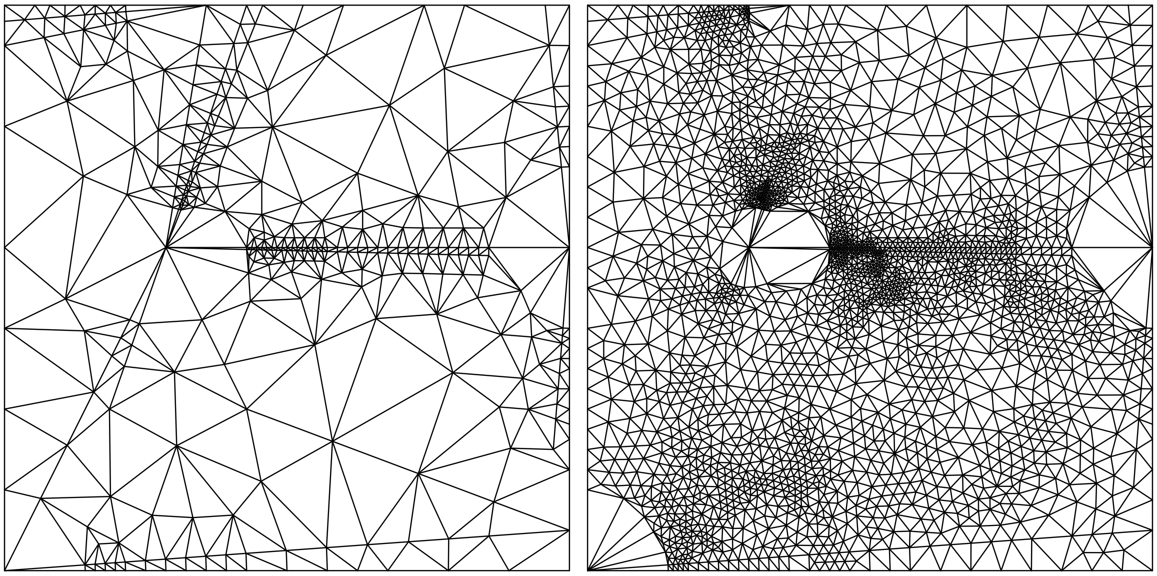
6.5 Sample Meshes

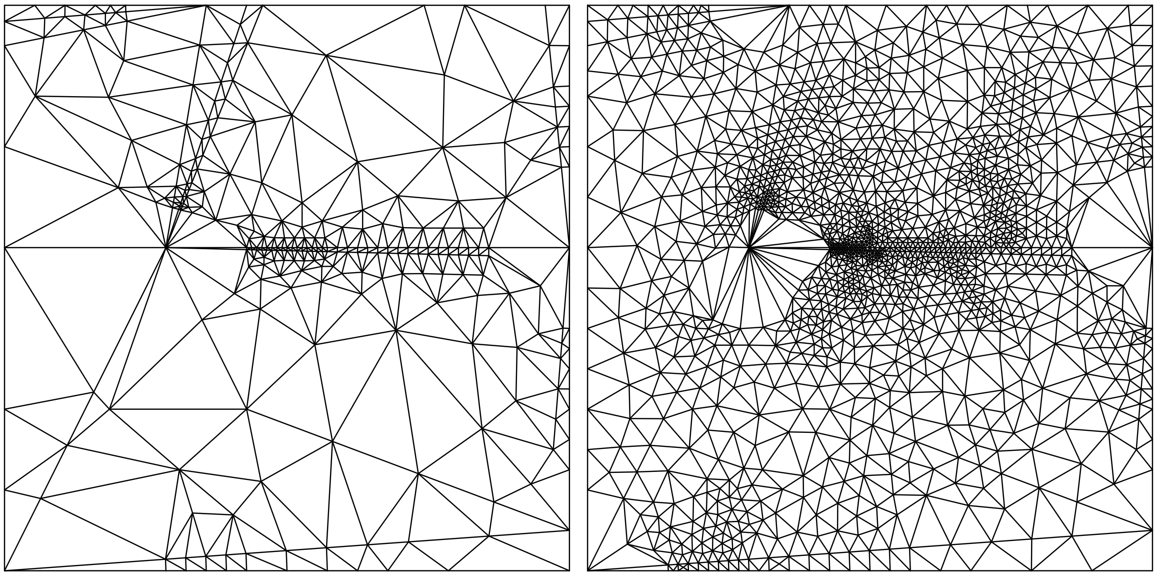
Figure 33 depicts a PSLG with nine small angles ranging from 1.43° to 8.97°, two meshes thereof produced by the Terminator using diametral circles (top), and two meshes produced by the Terminator using diametral lenses (bottom). The left meshes were produced with an angle bound of 20.7°, and the right meshes with an angle bound of 33°.

The upper left mesh has nine new angles less than 20.7°, of which two are less than 10° and the smallest is 5.83°. The lower left mesh has five new angles less than 20.7°, of which only the smallest, 6.24°, is less than 10°. The upper right mesh has 111 new angles less than 33°, of which twelve are less than 10° and the smallest is 5.83°. The lower right mesh has 86 new angles less than 33°, of which nine are less than 10° and the smallest is 4.49°. The original version of Ruppert's algorithm fails on this PSLG with any angle bound larger than about 11.5°, and Chew's algorithm fails for angle bounds larger than about 13°. (They terminate for smaller angle bounds only if they use concentric circular shells to split subsegments.)



circles, 20.7° angle bound, 643 triangles　　circles, 33° angle bound, 4744 triangles





lenses, 20.7° angle bound, 522 triangles　　lenses, 33° angle bound, 2965 triangles

Figure 33: Four meshes of a PSLG (top) produced by the Terminator. The top two meshes were created using diametral circles to define encroachment, and the bottom two were created using diametral lenses. The left meshes were produced with an angle bound of 20.7°; the right meshes with a bound of 33°. Angles smaller than these bounds occur only in triangles whose circumcenters would encroach upon a subsegment cluster.

The advantages of diametral lenses over diametral circles are most boldly realized when there are many small input angles or long, closely-spaced parallel segments, because lenses cause adjacent segments to encroach upon each other to a lesser degree than diametral circles do. As a consequence, fewer triangles are produced.

The program that produced these meshes deviates from the algorithm discussed in Sections 6.3 and 6.4 in two ways. First, because of time limitations (and to save memory), Condition F was not implemented.

The second difference is a slight simplification of Condition A, which saves a little bit of memory and programming effort. 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 30. 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 more conservative in its decision about whether to insert v. The Terminator's minimum angle bound (Theorem 12) still holds as well. With this modification, there is no need to store the insertion radius of each vertex, nor to store the information needed to determine which of the endpoints of an edge was most recently inserted.

7 Extensions and Improvements

Here, I describe several twists to the Delaunay refinement algorithms. The first extends them to flat interlocking surfaces embedded in three-dimensional space. The second improves the quality of triangles away from the boundary of the mesh. The third, which generalizes an idea of Chew, improves the quality of triangles everywhere.

7.1 Meshing Adjoining Planar Surfaces in Three Dimensions

Applications such as global illumination and radiosity use surface meshes of three-dimensional objects for lighting calculations and interpolation. A Delaunay refinement algorithm can mesh each planar surface independently, but what if the surfaces adjoin each other at shared edges, and the triangulations are required to conform to each other—that is, match triangle edge to triangle edge—along their shared boundaries?

Surprisingly, the Delaunay refinement algorithms need no conceptual changes to tackle this problem, so long as all the surfaces can be meshed simultaneously. The key is to redefine the notion of local feature size to use geodesic distances, rather than straight-line distances.

As a prelude, consider the significance of geodesic distances for triangulations of nonconvex two-dimensional domains. Delaunay refinement algorithms begin by forming a constrained Delaunay triangulation of the input PSLG. As Figure 34 (left) shows, if Delaunay refinement is performed before exterior triangles (those that are not part of the triangulation domain) are removed, overrefinement can occur near small exterior features. Overrefinement is prevented if exterior triangles are removed before refinement begins, and the encroachment of a subsegment by a vertex on its exterior side is ignored, as illustrated in Figure 34, right.

Ruppert [27] points out that the difference between these two cases can be formalized by redefining "local feature size." Let the *geodesic distance* between two points be the length of the shortest path between

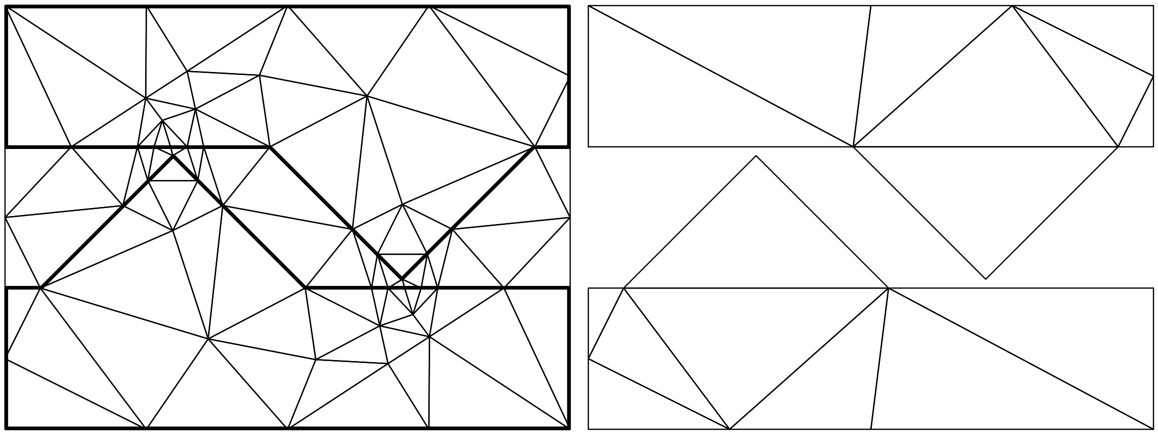


Figure 34: Two variations of Ruppert's Delaunay refinement algorithm with a 20° minimum angle. Left: If Delaunay refinement is performed before exterior triangles are removed, overrefinement can occur. Right: Mesh created by removing exterior triangles before refinement.

them that lies entirely in the triangulation domain. In other words, any geodesic path must go around holes and concavities. Given a PSLG X and a point p in the triangulation domain of X, define the local feature size $\mathrm{lfs}(p)$ to be the smallest value such that there exist two points u and v within a geodesic distance of $\mathrm{lfs}(p)$ from p that lie on features (vertices or segments) of X that are not incident to each other. This is essentially the same as the definition of local feature size given in Section 3.2, except that Euclidean distances are replaced with geodesic distances.

All of the proofs in this article can be made to work with geodesic distances. Lemma 2 depends only upon the triangle inequality, which holds for geodesic distances as well as Euclidean distances. Lemmata 3 and 5 hold because a child and its parent cannot be separated by an untriangulated region of the plane; they are visible to each other. The geodesic definition of local feature size prevents small exterior feature separations from influencing local feature sizes. Theorem 8's bounds on edge lengths are thus sometimes improved: in the left mesh of Figure 34, edge lengths are proportional to Euclidean local features sizes, whereas in the right mesh, edge lengths are proportional to geodesic feature sizes.

These observations about geodesic distance extend directly to planar surfaces embedded in three (or more) dimensions, which may join each other along shared segments. The triangulation domain is the union of all the planar surfaces. The geodesic distance between two points is the length of the shortest path between them that lies entirely within the triangulation domain. Where surfaces meet at shared segments, features in one surface may affect the local feature size in another, reflecting the fact that the refinement of one surface can affect another surface by splitting the shared segments.

The Delaunay refinement algorithms discussed in this article may be applied in this context with essentially no conceptual changes. There are some interesting implications, however; for instance, subsegment clusters need not be planar. An arbitrary number of planar surfaces might intersect at a single segment, and in each of these surfaces, the segment might participate in a small angle. However, the termination proofs require no change to account for this.

7.2 Improving the Quality Bound in the Interior of the Mesh

A simple improvement arises easily from the discussion in Section 3.3. So long as no cycle having a product less than one appears in the insertion radius flow graph (Figure 13 or 17), termination is assured. The barrier

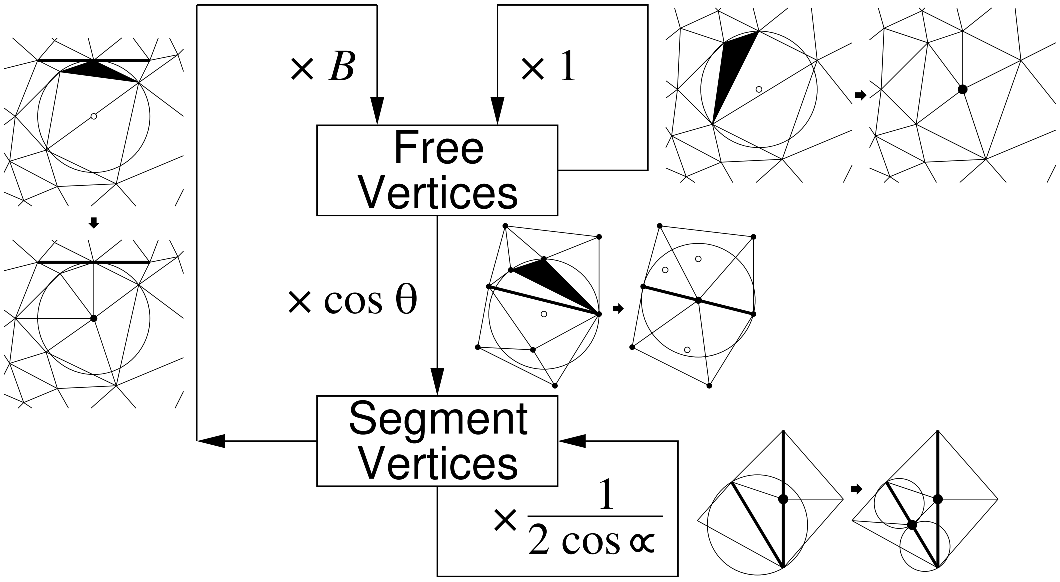


Figure 35: This flow diagram demonstrates how a simple modification to Chew's (or Ruppert's) algorithm can improve the quality of triangles away from mesh boundaries.

to reducing the quality bound B below $\sqrt{5}/2$ is the fact that, when an encroached segment is split, the child's insertion radius may be a factor of $\sqrt{5}/2$ smaller than its parent's. However, not every segment bisection is a worst-case example, and it is easy to explicitly measure the insertion radii of a parent and its potential progeny before deciding to take action. One can take advantage of these facts with either of the following strategies.

- Use a quality bound of $B = 1$ for triangles that are not in contact with segment interiors, and a quality bound of $B = \sqrt{2}$ (for Ruppert's algorithm) or $B = \sqrt{5}/2$ (for Chew's second algorithm) for any triangle having a vertex that lies in the interior of a segment.

  This strategy is easily understood from Figure 35. Because segment vertices may have smaller insertion radii than free vertices, segment vertices are only allowed to father free vertices whose insertion radii are larger than their own by a factor of $\sqrt{2}$ or $\sqrt{5}/2$, as appropriate. Hence, no diminishing cycles are possible.

- Attempt to insert the circumcenter of any triangle whose circumradius-to-shortest edge ratio is larger than one. If any subsegments would be encroached, the circumcenter is rejected as usual, and each encroached subsegment is checked to determine the insertion radius of the new vertex that might be inserted at its midpoint (which is half the length of the subsegment). The only midpoints inserted are those whose insertion radii are at least as large as the length of the shortest edge of the skinny triangle.

  This strategy works for the same reason the Terminator works: mesh vertices are expressly forbidden from creating descendants having insertion radii smaller than their own.

The first strategy differs from the second in its tendency to space segment vertices more closely than free vertices. The second strategy tends to space segment vertices and free vertices somewhat more equally. The first strategy interrupts the propagation of reduced insertion radii from segment vertices to the free vertices, whereas the second interrupts the process by which free vertices create segment vertices with

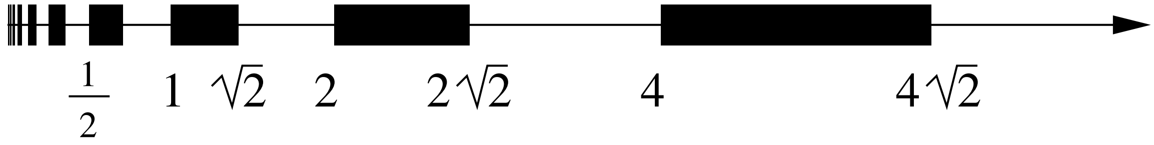


Figure 36: Legal subsegment lengths are of the form $c2^x$, where $c \in [1, \sqrt{2})$ and x is an integer.

smaller insertion radii. The effect of the first strategy is easily stated: upon termination, all angles are better than $20.7°$ (for Ruppert's algorithm) or $26.5°$ (for Chew's), and all triangles whose vertices do not lie in segment interiors have angles of $30°$ or better. For the second strategy, the delineation between $26.5°$ triangles and $30°$ triangles is not so simply stated, although the former only occur near segments.

If the angle bounds of the first strategy are loosened, the algorithm still exhibits good grading and size optimality. Bounds similar to those derived in Section 4.3 can be obtained, but the exercise is not especially enlightening, so details are omitted here.

7.3 Range-Restricted Segment Splitting

Chew's second Delaunay refinement algorithm, as he originally presented it [9], applies a quality bound of $B = 1$ to all triangles of the mesh, so no angle is smaller than $30°$. He achieves this bound through the occasional use of off-center segment splits. This section generalizes Chew's idea so that it can be applied to Ruppert's algorithm and both versions of the Terminator as well. When the angle bound is $30°$, however, there is no accompanying guarantee of good grading or size optimality. The modification is completely unnecessary in practice—Ruppert's and Chew's algorithms consistently achieve angle bounds better than $30°$ without it—but it is of theoretical interest, and it completes the presentation of Chew's second Delaunay refinement algorithm.

Consider Ruppert's algorithm first. Observe that the only mechanism by which a vertex can have a child with a smaller insertion radius than its own is by encroaching upon a subsegment. By Lemma 3, an encroaching circumcenter v cannot have a child whose insertion radius is smaller than $r_v/\sqrt{2}$ (if all subsegment splits are bisections), and hence it cannot cause the splitting of a segment whose length is less than $\sqrt{2}r_v$. Hence, a vertex v can only produce a child whose insertion radius is less than r_v if a segment is present whose length is between $\sqrt{2}r_v$ and $2r_v$. If no such segment exists, the cycle of diminishing edge lengths is broken.

Thus the motivation for *range-restricted segment splitting*. Whenever possible, the length of each subsegment is restricted to be of the form $c2^x$, where $c \in [1, \sqrt{2})$ and x is an integer. This restriction is illustrated in Figure 36, wherein darkened boxes on the number line represent legal subsegment lengths. The positive reals are partitioned into contiguous sets, each having a geometric width of $\sqrt{2}$, and alternate sets are made illegal. With this choice of partition, the bisection of any legal subsegment will produce legal subsegments.

Of course, input segments might not have legal lengths. However, when an encroached segment of illegal length is split, rather than place a new vertex at its midpoint, one may place the vertex so that the resulting subsegments fall within the legal range.

How does this restriction help? Assume no input angle is less than $45°$. Then a vertex whose insertion radius is 2^x or greater for some integer x cannot have a descendant whose insertion radius is less than 2^x unless a subsegment having illegal length is split. However, each illegal subsegment can be split only once, yielding subsegments of legal length; hence, the fuel for diminishing edge lengths is in limited supply.

An illegal segment of the form $c2^x$, where $c \in [\sqrt{2}, 2)$ and x is an integer, is split into subsegments of lengths $c_1 2^x$ and $c_2 2^x$ as follows.

- If $c \in [\sqrt{2}, \frac{3}{2})$, then $c_1 = \frac{\sqrt{2}}{4} - \epsilon$ and $c_2 = c - c_1$. (Here, ϵ is a minuscule value used because $\frac{\sqrt{2}}{4}$ is technically not in the legal range. In practice, let c_1 be the largest floating-point number less than $\frac{\sqrt{2}}{4}$.)
- If $c \in [\frac{3}{2}, 1 + \frac{\sqrt{2}}{2})$, then $c_1 = 1$ and $c_2 = c - c_1$.
- If $c \in [1 + \frac{\sqrt{2}}{2}, 2)$, then $c_1 = \frac{\sqrt{2}}{2} - \epsilon$ and $c_2 = c - c_1$.

The most unbalanced split occurs if c is just slightly less than $3/2$. Then, the ratio between c_1 and c is at least $\sqrt{2}/6 \doteq 0.2357$, so one subsegment is slightly less than a quarter as long as the original segment.

Theorem 13 *Suppose that any two incident segments are separated by an angle of at least 60°, and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is larger than $B \geq 1$. Ruppert's algorithm with range-restricted segment splitting will terminate, with no triangulation edge shorter than* $\mathrm{lfs}_{\min}/6$.

Proof: Omitted. ■

Chew's algorithm makes it possible to use narrower illegal ranges. If the input PSLG has no acute angles, a subsegment midpoint with parent p cannot have an insertion radius smaller than $r_p \cos\theta$ (Lemma 5), so the geometric width of each illegal range only needs to be $1/\cos\theta$. For instance, if one uses Chew's algorithm with $\theta = 30^\circ$, or the Ruppert–Chew hybrid algorithm with diametral lenses, one may use illegal ranges having a geometric width of $2/\sqrt{3} \doteq 1.15$ instead of $\sqrt{2} \doteq 1.41$. In this case, legal segment lengths are of the form $c2^x$, where $c \in [1, \sqrt{3})$, and x is an integer.

Because the legal range is wider, and the illegal range narrower, splitting an illegal segment to yield legal subsegments is easier. Chew handles illegal segments by trisecting them. One can do better by splitting them into two pieces, just unevenly enough to ensure that both subsegment lengths are legal. The following recipe is suggested.

- If $c \in [\sqrt{3}, 1 + \frac{\sqrt{3}}{2})$, then $c_1 = 1$ and $c_2 = c - c_1$.
- If $c \in [1 + \frac{\sqrt{3}}{2}, 2)$, then $c_1 = \frac{\sqrt{3}}{2} - \epsilon$ and $c_2 = c - c_1$.

The most unbalanced split occurs if c is $\sqrt{3}$. In this case, the ratio between c_2 and c is approximately $1 - 1/\sqrt{3} \doteq 0.4226$, which isn't much worse than bisection.

For comparison, consider how Chew [9] guarantees that his algorithm terminates for an angle bound of 30°. Chew employs range-restricted segment splitting, but uses only one illegal range instead of an infinite sequence of ranges. For an appropriate value h (see Chew for details, but one could use $h = \mathrm{lfs}_{\min}/2$), Chew declares the range $(\sqrt{3}h, 2h)$ illegal; subsegments with lengths between $2\sqrt{3}h$ and $4h$ are trisected rather than bisected. Hence, no edge shorter than h ever appears.

There are two disadvantages of using a single illegal range, rather than an infinite series of illegal ranges. The first is the inconvenience of computing h in advance. The second and more fundamental problem is that

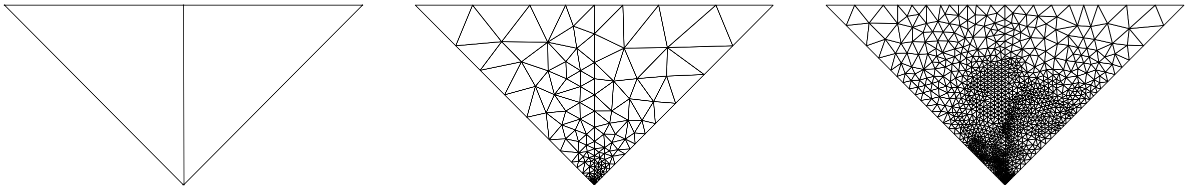


Figure 37: (a) A PSLG with a tiny segment at the bottom tip. (b) With a minimum angle of 32° ($B \doteq 0.944$), Ruppert's algorithm creates a well-graded mesh, despite the lack of theoretical guarantees. (c) With a minimum angle of 33.5° ($B \doteq 0.906$), grading is poor.

if small angles are present in the input PSLG, edges shorter than h may arise anyway. The Terminator and range-restricted segment splitting mix well, but the Terminator requires an infinite series of illegal ranges.

How may the Terminator use range-restricted segment splitting? Modified segment splitting using concentric circular shells has priority over range-restricted segment splitting. Cluster subsegments are split to power-of-two lengths, and are thus legal. If a subsegment is not part of a subsegment cluster, it is split using the rules of range-restricted segment splitting. Hence, every segment may undergo up to three unbalanced splits: one for each end of the segment (using concentric shells), and one more if the middle subsegment has an illegal length. The only modification that must be made to Theorem 11 to prove that the Terminator still terminates is to account for the third unbalanced split (which can only happen a finite number of times). The Terminator's bound on the smallest angle (Theorem 12) still applies, with no change to the proof. Angles not near small input angles are guaranteed to be between 30° and 120°.

It does not appear to be possible to prove that Delaunay refinement with range-restricted segment splitting produces nicely graded or size-optimal meshes with circumradius-to-shortest edge ratios that are very close to one. The theoretical difficulty is that if a mesh contains a long segment with a small local feature size at one end, the small feature size could (in principle) propagate along the whole length of the segment. A short subsegment at one end of the segment might indirectly cause its neighboring subsegment to be split until the neighbor is the same length. The neighboring subsegment might then cause its neighbor to be split, and so on down the length of the segment.

As Figure 37 demonstrates, however, even if Ruppert's algorithm with diametral circles is used, a chain reaction severe enough to compromise the grading of the mesh only seems to occur in practice if the quality bound is less than about 0.92 (corresponding to an angle of about 33°)!

The meshes in this figure were generated without range-restricted segment splitting, which is useful as a theoretical construct but unnecessary in practice. There is a good deal of "slack" in the inequalities that underlie the proofs of termination, because not every vertex has the smallest possible insertion radius relative to its parent. As a result of this slack, any Delaunay refinement algorithm that handles boundaries in a reasonable way seems to achieve angle bounds higher than 30°.

8 Conclusions

The intuition governing Delaunay refinement arises from an understanding of the relationship between the insertion radii of parents and their children. Hence, I use flow graphs to demonstrate these relationships. This analysis technique brings clarity to ideas that otherwise might be hidden within proofs. For instance, Figure 13 provides an immediate explanation for why Ruppert's algorithm achieves an angle bound of up to

Proven bounds on the minimum angle			
	Diametral Circles	Diametral Lenses	Chew's Algorithm
nicely graded & size-optimal	$20.70°$	$25.65°$	$26.56°$
grading not guaranteed	$30°$	$30°$	$30°$
Terminator	$\arcsin[\sin(\phi/2)/\sqrt{2}]$	$\arcsin[(\sqrt{3}/2)\sin(\phi/2)]$	n/a

Table 1: Angle bounds for the Delaunay refinement algorithms described in this article. The bounds in the first two rows apply only sufficiently far from small input angles. ϕ is the smallest input angle. The $30°$ bounds in the second row depend on the use of range-restricted segment splitting.

$20.7°$ (which corresponds to a circumradius-to-shortest edge ratio of $\sqrt{2}$). The same analysis can be found in Ruppert's paper, but the underlying intuition is obscured by the mathematics.

The flow graphs pointed my way to a variety of improvements to Delaunay refinement and its analysis.

- The Terminator solves the problem of small input angles by preventing mesh vertices from having descendants with smaller insertion radii.
- A new analysis of Chew's algorithm arose from my efforts to understand the relationship between segment midpoints and their parents, a relationship reflected in the flow graph of Figure 17.
- The flow graphs sparked my recognition of the fact that a better quality bound can be applied in the interior of the mesh, as illustrated in Figure 35.
- Range-restricted segment splitting is a generalization of Chew's method for weakening the spiral of diminishing insertion radii.

Simple as the flow graphs are, they provided the clues that helped to unearth most of the new results in this article. Even more interestingly, they illuminate the way to extending most of the ideas discussed in this article to three-dimensional mesh generation [30, 31]. Table 1 summarizes the angle bounds of the two-dimensional Delaunay refinement algorithms.

The problem of meshing PSLGs with tiny angles is harder than most researchers recognized. The negative result on quality triangulations in Section 5 comes as a surprise; many of us have been laboring under the false assumption that arbitrary PSLGs could be triangulated without creating new small angles. The recognition that this feat is impossible helps point the way to strategies for minimizing the unavoidable damage that small input angles can cause. The method suggested in Sections 6.3 and 6.4 is rather elaborate, but rewards the effort with good triangulations.

Acknowledgments

Thanks to Omar Ghattas, Thomas Gross, Gary Miller, David O'Hallaron, Jim Ruppert, James Stichnoth, Dafna Talmor, and Daniel Tunkelang for comments and conversations that greatly improved this document.

A Implementing Delaunay Refinement

Pseudocode for the Terminator is provided in Figures 38 and 39. This pseudocode implements both the Terminator with diametral circles from Section 6.3, and the Terminator with diametral lenses from Section 6.4.

```
DelaunayTerminator(X, θ, δ)
{ X is a segment-bounded PSLG. θ is the minimum allowable angle. δ(t) is a boolean function }
{ that returns true if triangle t is too large (as determined by the user), false otherwise. }
    Construct T, the constrained Delaunay triangulation of X
    Remove from T all triangles not in the triangulation domain
    for each segment s ∈ X
        if s is encroached
            Enqueue(s, Q_s)      { Q_s is the queue of encroached subsegments }
    SplitEncroachedSubsegments(0, false)
    for each triangle t ∈ T
        if δ(t) or t has an angle less than θ (but not separating two subsegments)
            Enqueue(t, Q_t)      { Q_t is the queue of bad triangles }
    while queue Q_t is not empty
        t ← Dequeue(Q_t)
        if t still exists in T
            c ← the circumcenter of t
            E ← the set of subsegments that c encroaches upon
            if E = ∅
                Insert c into T, maintaining the constrained Delaunay property
                NewVertex(c, θ, δ)      { check subsegments and triangles }
            else
                d ← r_g, where g is youngest endpoint of shortest edge of t
                        { alternatively, d ← length of shortest edge of t }
                for each s ∈ E
                    if δ(t) or SplitPermitted(s, d)
                        Enqueue(s, Q_s)
                if queue Q_s is not empty
                    Enqueue(t, Q_t)      { Put t back in queue for another try }
                    SplitEncroachedSubsegments(θ, δ)
    return T

SplitPermitted(s, d)
    if s belongs to no subsegment cluster, s belongs to two subsegment clusters,
                        or the length of s is not a power of two (within some tolerance)
        return true
    S ← the subsegment cluster that includes s
    if S contains a subsegment shorter than s (within some tolerance)
        return true
    r_min ← |s| sin(φ_min/2), where φ_min is the smallest angle separating two subsegments in S.
                        { r_min is the length of the shortest edge that will be created if
                        all subsegments in S are split to a length of |s|/2. }
    if r_min ≥ d
        return true      { No new edge will be shorter than the shortest existing edge }
    return false
```

Figure 38: Code for the Terminator, the Delaunay refinement algorithm recommended in this article. The subroutines SplitEncroachedSubsegments and NewVertex appear in Figure 39. Subsegment encroachment can be defined using either diametral circles or diametral lenses.

```
SplitEncroachedSubsegments(θ, δ)
36   while queue Q_s is not empty
37       s ← Dequeue(Q_s)
38       if s is still an edge of T
39           Let v be a splitting vertex in s (chosen using concentric circular shells)
{ Lines 40–43 needed only when using diametral lenses }
40           Delete from T all free vertices visible from v inside the diametral circle of s,
                     maintaining the constrained Delaunay property
41           for each triangle t created by Line 40
42               if δ(t) or t has an angle less than θ (but not separating two subsegments)
43                   Enqueue(t, Q_t)
44           Insert v into T, splitting s into s_1 and s_2 and maintaining the
                     constrained Delaunay property
45           NewVertex(v, θ, δ)
46           if s_1 is encroached
47               Enqueue(s_1, Q_s)
48           if s_2 is encroached
49               Enqueue(s_2, Q_s)

NewVertex(v, θ, δ)
50   for each triangle t ∈ T that has v as a vertex
51       e ← the edge of t opposite v
52       if e is a subsegment and v encroaches upon e
53           Enqueue(e, Q_s)
54       else if δ(t) or t has an angle less than θ (but not separating two subsegments)
55           Enqueue(t, Q_t)
```

Figure 39: Subroutines for the Terminator.

After the initial triangulation is constructed, Line 2 removes all exterior triangles (those not in the triangulation domain) before Delaunay refinement begins. By contrast, in Ruppert's presentation [27], Delaunay refinement proceeds first. One of the reasons for removing exterior triangles first is discussed in Section 7.1: so that small exterior features do not bring forth unnecessarily small triangles. Figure 40 illustrates another reason. If the entire convex hull of the triangulation domain is triangulated, the boundary edges of the convex hull must be treated as segments, and spurious small angles and features may form. If an input vertex lies just inside the convex hull, then the local feature size near the vertex may be artificially reduced to an arbitrarily small distance. An arbitrarily small angle may also appear with dastardly stealth, as the figure shows. If the Terminator is not used, the user might be mystified why Ruppert's or Chew's algorithm fails to terminate on what appears to be an easy PSLG. In fact, it was this example that first made the negative result of Section 5 evident to me.

Early removal of extraneous triangles solves this problem, as Figure 40(d) shows. It also ensures that if two segments are separated by a small angle within a hole or concavity, where the region between the segments is not part of the triangulation domain, the small angle will have no effect on meshing.

During Delaunay refinement, the pseudocode maintains a queue Q_t of skinny and oversized triangles and a queue Q_s of encroached subsegments. (Chew's algorithm does not require the latter queue, as only one subsegment may be encroached at a time.) After the initial constrained Delaunay triangulation is con-

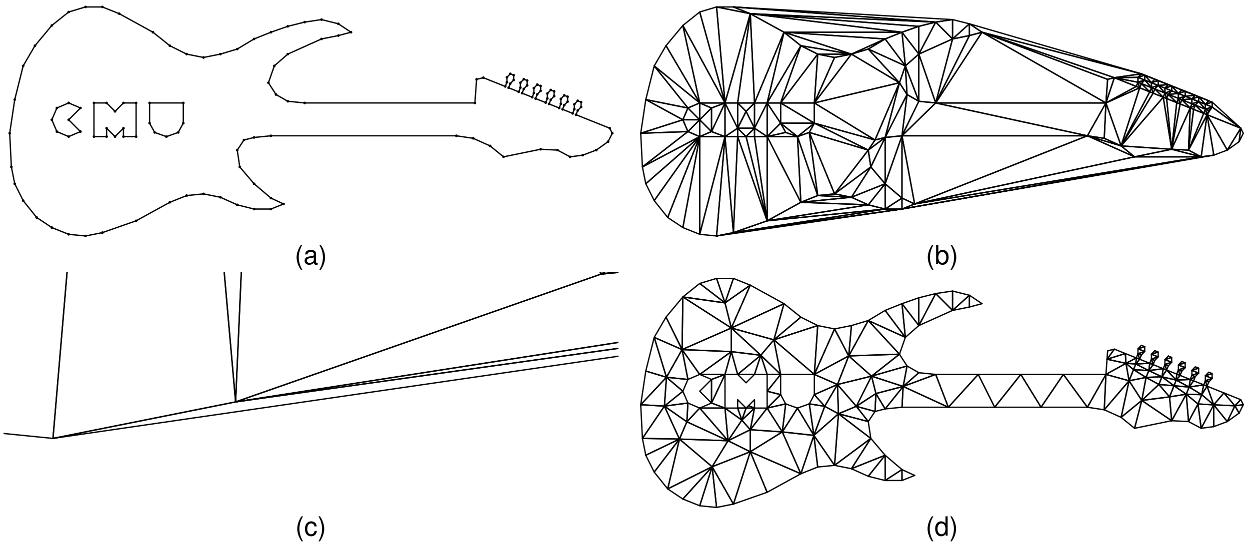


Figure 40: (a) Guitar PSLG. (b) Its constrained Delaunay triangulation. (c) Close-up of a small angle formed at the bottom of the guitar between a PSLG segment and a convex hull edge. (d) Mesh with 20° minimum angle. Exterior triangles were removed before applying Delaunay refinement.

structed, the queue Q_s is initialized by testing each segment of the initial triangulation for encroachment. After all the encroached segments are split, the queue Q_t is initialized by testing the size and angles of each triangle in the triangulation. Both queues are maintained throughout Delaunay refinement. Q_s rarely contains more than a few items, except at the beginning of Delaunay refinement (Lines 3–6), when it may contain many.

Each vertex insertion may add new members to either queue. After a new vertex v is inserted, The subroutine NEWVERTEX identifies and enqueues all the bad triangles thus created, and all the subsegments encroached upon by v. If v was inserted into a subsegment, the two subsegments resulting from the split must also be tested for encroachment (subroutine SPLITENCROACHEDSUBSEGMENTS, Lines 46–49).

Fortunately, testing a subsegment for encroachment (Lines 4, 46, 48, 52) is a quick local operation. The algorithm must inspect only the one or two vertices that form triangles in conjunction with the subsegment. To see why, consider Figure 41(a). In the illustration, both vertices (v and w) opposite the subsegment s lie outside the diametral circle of s. Because the mesh is constrained Delaunay, each triangle's circumcircle encloses no vertex visible from the triangle's interior, and therefore the diametral circle of s encloses no vertex visible from the relative interior of s. In general, if s is encroached, then either v or w must lie on or inside its diametral circle. As Figure 41(b) shows, the same argument is true if diametral lenses are used instead of diametral circles, because a diametral lens is defined by circular arcs passing through a segment's endpoints.

The test for encroached diametral circles is quite simple. Let t be a triangle formed by a subsegment s and a vertex w opposite s. Let x and y be the endpoints of s. If the angle at w is greater than or equal to 90°, then w encroaches upon s; this condition is true if $(x - w) \cdot (y - w) \leq 0$. A diametral lens is encroached if the angle at w is 120° or more, which is true if $(x - w) \cdot (y - w) \leq -|xw||yw|/2$.

Line 16 performs a Delaunay insertion of c, the circumcenter of a triangle t, into the triangulation. Before c can be inserted, the triangle containing c must be located. The potential difficulty is that general-purpose point location in a dynamically changing nonconvex triangulation is complicated. Fortunately, locating the circumcenter of a triangle is much easier and quicker than general point location. The circumcenter c of t is

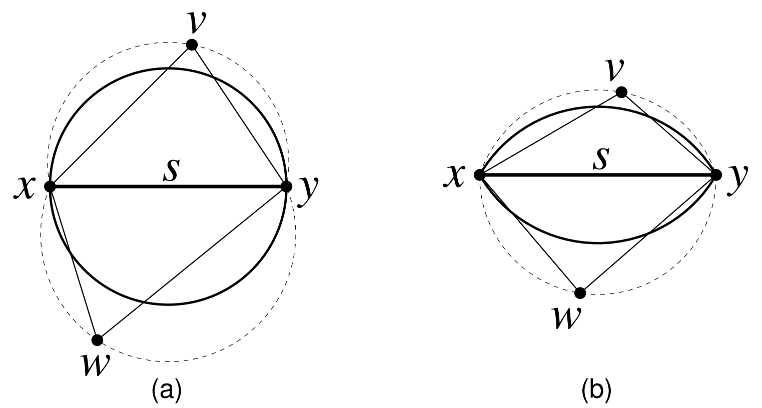


Figure 41: (a) If the apices (here, v and w) of the triangles that contain a subsegment s are outside the diametral circle of s, then no visible vertex lies in the diametral circle of s, because the triangles are constrained Delaunay. (b) The same statement is true for the diametral lens of s.

located by walking from the centroid of t toward c. If the path is blocked by a subsegment, then c encroaches upon the subsegment, and the search may be abandoned. (The subsegment is encroached whether diametral circles or diametral lenses are used.) Because the mesh is segment-bounded, each search must either succeed or be foiled by an encroached subsegment.

To help in deciding whether to insert the circumcenter c, Line 14 finds the subsegments encroached upon by c. The simplest way to accomplish this is to insert c into the triangulation, then check each edge that appears opposite c in some triangle to see if it is a subsegment and if it is encroached upon by c (just like the NEWVERTEX subroutine does, but without enqueueing the encroached subsegments). If c does encroach upon one or more subsegments, c must be deleted, restoring the mesh to its original form. The most efficient way to accomplish this is to record the sequence of edge flips performed by Lawson's algorithm when c is inserted, then reverse the sequence if c must be deleted. One caveat in using this approach: it is important that any "bad" triangles (too skinny or too large) that were in the queue Q_t just before c was inserted are still in the queue after c is deleted.

If c is rejected because it encroaches upon a subsegment, and at least one subsegment is split as a result, then t is put back in the queue Q_t (Line 24). Recall from the example of Figure 7 that even if c is rejected, it may be inserted later when Delaunay refinement attempts to split t again. However, if the Terminator declines to split every segment c encroaches upon, t must *not* be put back in the queue, because an infinite loop might occur as the algorithm tries repeatedly to split t. For the same reason, it is also important that any triangles that were *not* in Q_t just before c was inserted are still *not* in Q_t after c is deleted, even if they are skinny.

Two different options are given for Line 19. These options are discussed briefly in Section 6.5. It takes more programming effort to determine r_g than to determine the length of the shortest edge of t. The latter choice is slightly more conservative about splitting subsegment clusters, so it may produce a mesh with fewer triangles, but it may also leave more skinny triangles behind. I have not implemented the former choice, so this comparison is speculative.

Line 40 asks that a free vertex be deleted from the mesh while the constrained Delaunay property is preserved. The simple $\mathcal{O}(m \log m)$-time vertex deletion algorithm of Devillers [13] is suitable (where m is the degree of the vertex), but the typical vertex degree is usually so small in practice that even a simple $\mathcal{O}(m^2)$-time algorithm will suffice, and perhaps run faster.

The number of triangles in the final mesh depends on the order in which skinny triangles are split,

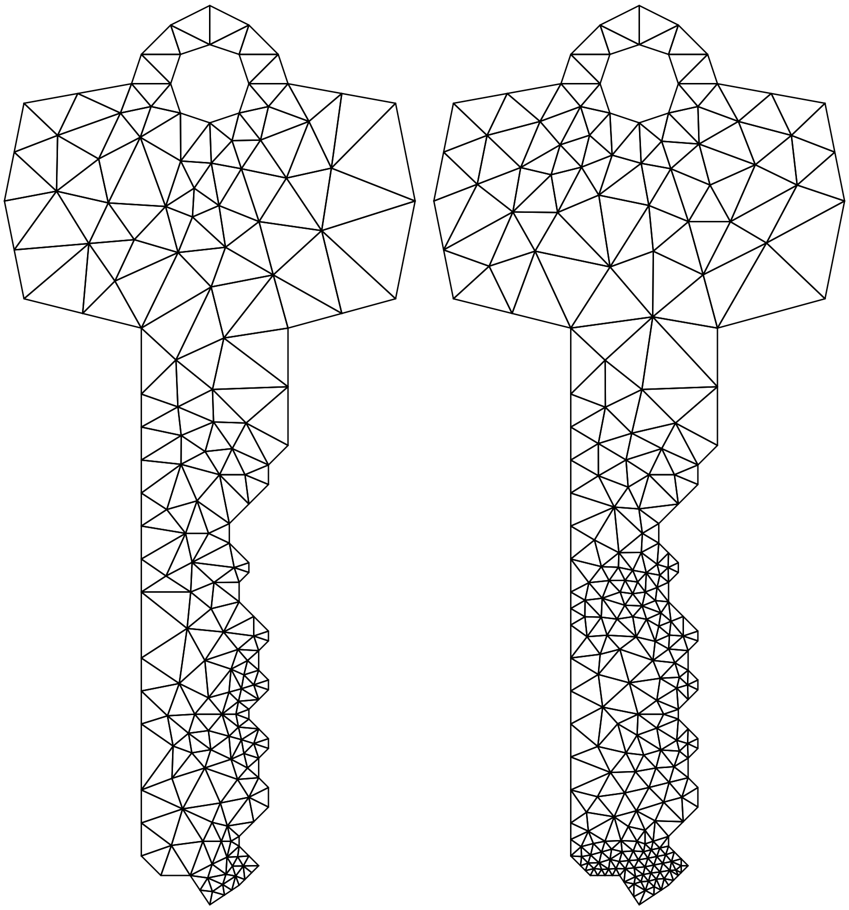


Figure 42: Two meshes with a 33° minimum angle. The left mesh, with 290 triangles, was formed by always splitting the worst existing triangle. The right mesh, with 450 triangles, was formed by using a first-come first-split queue of skinny triangles.

especially when a strong angle bound is used. Figure 42 demonstrates how sensitive Ruppert's algorithm is to the order. For this example with a 33° minimum angle, a heap of skinny triangles indexed by their smallest angle confers a 35% reduction in mesh cardinality over a first-in first-out (FIFO) queue. (This difference is typical for strong angle bounds, but thankfully seems to disappear for small angle bounds.) This improvement probably occurs because circumcenters of very skinny triangles are likely to eliminate more skinny triangles than circumcenters of mildly skinny triangles. Unfortunately, a heap is slow for large-cardinality meshes, especially when bounds on the maximum triangle size force all of the triangles into the heap. Delaunay refinement usually takes $\mathcal{O}(n)$ time in practice, where n is the number of vertices inserted, but the use of a heap increases the complexity to $\mathcal{O}(n \log n)$.

A solution that seems to work well in practice is to use 64 FIFO queues, each representing a different interval of circumradius-to-shortest edge ratios. Oddly, it is counterproductive in practice to order good-quality triangles, so just one queue is used for good-quality but too-large triangles whose ratios are all roughly smaller than 0.8, corresponding to a minimum angle of about 39°. Triangles with larger quality ratios are partitioned among the remaining queues. When a skinny triangle is chosen for splitting, it is taken from the "worst" nonempty queue. An ordered queue of nonempty queues is maintained so that a skinny triangle may be chosen quickly. This method yields meshes comparable with those generated using a heap, but is only slightly slower than using a single queue.

References

[1] Ivo Babuška and A. K. Aziz. *On the Angle Condition in the Finite Element Method*. SIAM Journal on Numerical Analysis **13**(2):214–226, April 1976.

[2] Brenda S. Baker, Eric Grosse, and C. S. Rafferty. *Nonobtuse Triangulation of Polygons*. Discrete and Computational Geometry **3**(2):147–168, 1988.

[3] Marshall Bern and David Eppstein. *Mesh Generation and Optimal Triangulation*. Computing in Euclidean Geometry (Ding-Zhu Du and Frank Hwang, editors), Lecture Notes Series on Computing, volume 1, pages 23–90. World Scientific, Singapore, 1992.

[4] Marshall Bern, David Eppstein, and John R. Gilbert. *Provably Good Mesh Generation*. Journal of Computer and System Sciences **48**(3):384–409, June 1994.

[5] Adrian Bowyer. *Computing Dirichlet Tessellations*. Computer Journal **24**(2):162–166, 1981.

[6] Graham F. Carey and John Tinsley Oden. *Finite Elements: Computational Aspects*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[7] L. Paul Chew. *Constrained Delaunay Triangulations*. Algorithmica **4**(1):97–108, 1989.

[8] ———. *Guaranteed-Quality Triangular Meshes*. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[9] ———. *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.

[10] ———. *Guaranteed-Quality Delaunay Meshing in 3D*. Proceedings of the Thirteenth Annual Symposium on Computational Geometry, pages 391–393. Association for Computing Machinery, June 1997.

[11] Kenneth L. Clarkson and Peter W. Shor. *Applications of Random Sampling in Computational Geometry, II*. Discrete & Computational Geometry **4**(1):387–421, 1989.

[12] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[13] Olivier Devillers. *On Deletion in Delaunay Triangulations*. Proceedings of the Fifteenth Annual Symposium on Computational Geometry, pages 181–188. Association for Computing Machinery, June 1999.

[14] Tamal Krishna Dey, Chanderjit L. Bajaj, and Kokichi Sugihara. *On Good Triangulations in Three Dimensions*. International Journal of Computational Geometry & Applications **2**(1):75–95, 1992.

[15] Rex A. Dwyer. *A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations*. Algorithmica **2**(2):137–151, 1987.

[16] William H. Frey. *Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes*. International Journal for Numerical Methods in Engineering **24**(11):2183–2200, November 1987.

[17] Leonidas J. Guibas and Jorge Stolfi. *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*. ACM Transactions on Graphics **4**(2):74–123, April 1985.

[18] Mark T. Jones and Paul E. Plassmann. *Adaptive Refinement of Unstructured Finite-Element Meshes*. Finite Elements in Analysis and Design **25**:41–60, 1997.

[19] Charles L. Lawson. *Software for C^1 Surface Interpolation*. Mathematical Software III (John R. Rice, editor), pages 161–194. Academic Press, New York, 1977.

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

[21] Der-Tsai Lee and Bruce J. Schachter. *Two Algorithms for Constructing a Delaunay Triangulation*. International Journal of Computer and Information Sciences **9**(3):219–242, 1980.

[22] Gary L. Miller, Dafna Talmor, and Shang-Hua Teng. *Optimal Good-Aspect-Ratio Coarsening for Unstructured Meshes*. Proceedings of the Eighth Annual Symposium on Discrete Algorithms (New Orleans, Louisiana), pages 538–547. Association for Computing Machinery, January 1997.

[23] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. *A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition*. Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing (Las Vegas, Nevada), pages 683–692, May 1995.

[24] 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.

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

[26] ——— . *A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation*. Proceedings of the Fourth Annual Symposium on Discrete Algorithms, pages 83–92. Association for Computing Machinery, January 1993.

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

[28] Raimund Seidel. *Backwards Analysis of Randomized Geometric Algorithms*. Technical Report TR-92-014, International Computer Science Institute, University of California at Berkeley, Berkeley, California, February 1992.

[29] Jonathan Richard Shewchuk. *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*. Applied Computational Geometry: Towards Geometric Engineering (Ming C. Lin and Dinesh Manocha, editors), Lecture Notes in Computer Science, volume 1148, pages 203–222. Springer-Verlag, Berlin, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[30] ______. *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.

[31] ______. *Mesh Generation for Domains with Small Angles*. Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong), pages 1–10. Association for Computing Machinery, June 2000.

[32] David F. Watson. *Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes*. Computer Journal **24**(2):167–172, 1981.