Chapter 10
A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation*

Jim Ruppert†

Abstract

We present a simple new algorithm for triangulating polygons and planar straightline graphs. It provides "shape" and "size" guarantees:

- All triangles have a bounded aspect ratio.
- The number of triangles is within a constant factor of optimal.

Such "quality" triangulations are desirable as meshes for the finite element method, in which the running time generally increases with the number of triangles, and where the convergence and stability may be hurt by very skinny triangles. The technique we use—successive refinement of the Delaunay triangulation—extends a mesh generation technique of Chew by allowing triangles that vary in size. Previous algorithms with shape and size bounds have all been based on quadtrees. The Delaunay refinement algorithm matches their theoretical bounds, but uses a fundamentally different approach. It is much simpler, and hence easier to implement, and it generally produces smaller meshes in practice.

1 Introduction

Many applications in computational geometry, graphics, solid modeling, numerical simulation and other areas require complicated geometric objects to be decomposed into simpler pieces for further processing. For instance, in the finite element method, a planar domain is divided into a mesh of elements, typically triangles. Differential equations representing some physical property such as heat distribution or airflow are then approximated using functions that are piecewise polynomial within each triangle. The running time and accuracy of these algorithms often depends on properties of the decomposition, such as its *size* (the number of pieces), and its *shape* (whether the pieces are "long and skinny").

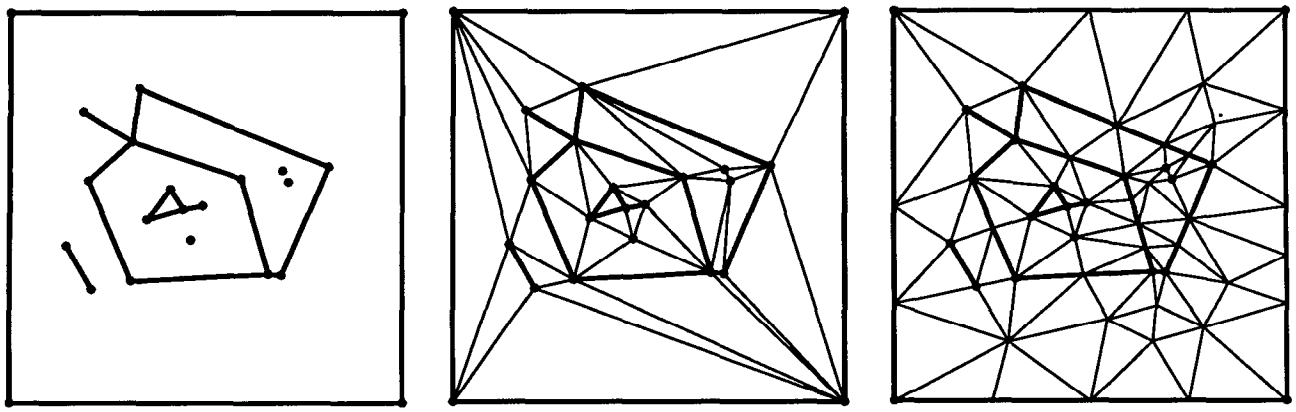
Our interest in this paper is the decomposition of 2-dimensional objects such as polygons into triangles. We will refer to this problem both as *mesh generation* and *triangulation*. *Quality mesh generation* describes techniques that offer a guarantee on some measure of shape, such as all triangles non-obtuse, or all with bounded aspect ratio. The *aspect ratio* of a triangle is length of the longest edge divided by the length of the shortest altitude. A fairly general measure of triangle shape is the minimum angle α, since this gives a bound of $\pi - 2\alpha$ on maximum angle and guarantees an aspect ratio between $|\frac{1}{\sin\alpha}|$ and $|\frac{2}{\sin\alpha}|$. We allow triangulations to contain *Steiner points*—vertices of the mesh that are not vertices of the input—because in general they are necessary for achieving shape bounds (see Figure 1, for example). A mesh satisfying a certain shape bound is said to be *size-optimal* if the number of triangles is within a constant factor of the minimum number possible in any triangulation of the given input that meets the same shape bound.

The first algorithm to give a shape guarantee was due to Baker, Grosse and Rafferty [1]. They gave a technique for producing a non-obtuse triangulation of polygons, in which all angles are at most 90°. In addition, the smallest angle is at least 13°. (Of course, this is only possible if all angles in the input are at least 13°.) Together, these bounds guarantee an aspect ratio of at most 4.6. Their algorithm places a uniform square grid over the polygon, with grid spacing determined by the smallest feature present in the polygon. (Roughly speaking, the *smallest feature* is determined either by the pair of closest vertices, or by the closest vertex-edge pair, where the edge does not contain the vertex.) Since the smallest feature determines the mesh density throughout the polygon, the number of triangles can be very large.

Bern, Eppstein and Gilbert gave the first mesh generation algorithm with both shape and size guarantees in [3]. They show how to triangulate polygons so that every triangle has aspect ratio at most 5. In addition, their analysis shows that the mesh is size-optimal. One of the key ideas in their algorithm is to replace the uniform grid of [1] with a *quadtree*, which is a recursive subdivision into squares of varying sizes. This yields large triangles in areas of large features. By keeping the quadtree *balanced*, aspect ratios are bounded in the output. Melissaratos and Souvaine [9] give some extensions to the quadtree algorithm. Mitchell and Vavasis show in [10] an extension of the quadtree technique to

*Supported by funds from NSF PYI Grant CCR-90-58840.

†Computer Science Division, University of California at Berkeley, Berkeley, CA 94720, USA. E-mail: ruppert@cs.berkeley.edu. A portion of this work was done while the author was at Hewlett-Packard Laboratories, Palo Alto, CA 94303-0969, USA.



(a) Typical input PSLG and bounding box

(b) Typical triangulation without Steiner points

(c) Output of Delaunay refinement algorithm with minimum angle 20 degrees

Figure 1: Sample input planar straightline graph (PSLG), typical (non-Steiner) triangulation of PSLG and bounding box, and Delaunay refinement algorithm's output.

3D. They give an algorithm that uses *octrees* to produce size-optimal, bounded aspect ratio triangulations of polyhedra.

All the above techniques use grids or quadtrees. A quite different technique for quality mesh generation is *Delaunay refinement*, so-called because a Delaunay triangulation is maintained, and some criterion is used to successively pick new points to add to it. Chew [5] presented a Delaunay refinement algorithm that triangulates a given polygon into a mesh in which all angles are between 30 and 120 degrees. The algorithm produces *uniform* meshes, meaning that all triangles are roughly the same size. The output mesh is size-optimal (to within a constant factor) amongst all uniform meshes. However, as was the case for the algorithm of [1], a uniform mesh may have many more triangles than are necessary.

In this paper, we extend Chew's work by giving an algorithm to triangulate planar straightline graphs (PSLGs) such that all triangles in the output have angles between α and $\pi - 2\alpha$. Here α is a parameter that can be chosen between 0 and 20 degrees. The triangles will vary in size, and the mesh will be size optimal to within a constant factor (the constant depends on the choice of α). PSLGs include polygons, polygons with holes, and complexes (objects made of multiple polygons); dangling edges and isolated vertices are also allowed (see Figure 1(a)).

Theoretically speaking, our algorithm essentially matches the PSLG algorithms of [9] and [3](modified as mentioned in [2]), but it is distinguished from them in a number of ways: (1) The Delaunay refinement approach is fundamentally different from the quadtree techniques. (2) It is much simpler. With fewer special case constructions, it is easier to implement. (3) It generally produces fewer triangles in practice. (4) It is "parameterized": the user can ask for the "best" with a given number of triangles. In this way, the algorithm takes advantage of the inherent mesh size/shape tradeoff. (5) The output mesh is more "intrinsic" to the input. For instance, quadtree meshes produce a sort of "scaffold" of mesh edges aligned with the coordinate axes. Such alignment may affect subsequent computation. (6) Delaunay refinement produces a unique mesh, independent of the orientation of the input. (Strictly speaking, this requires careful handling of input degeneracies such as co-circular points, as well as elimination of the bounding box, see § 5.)

A few words about the input to the algorithm: The input can be any planar straightline graph (PSLG), with dangling edges and isolated points allowed (see Figure 1(a)). As shown in the figure, the algorithm will triangulate a larger region, out to an enclosing box. To get a triangulation of a particular region, say the interior of a polygon, exterior triangles can be removed. (To maintain the size optimality guarantee in this case, the algorithm must be modified slightly, as discussed in § 5.)

The remainder of this paper is organized as follows. In the next section we present our algorithm. Then we show that it halts and outputs a valid triangulation satisfying the minimum angle bound. We define the *local feature size* at each point in the input, and bound the output size in terms of it. Then we show that every triangle is within a constant factor of the largest possible at that point, which proves size-optimality. We

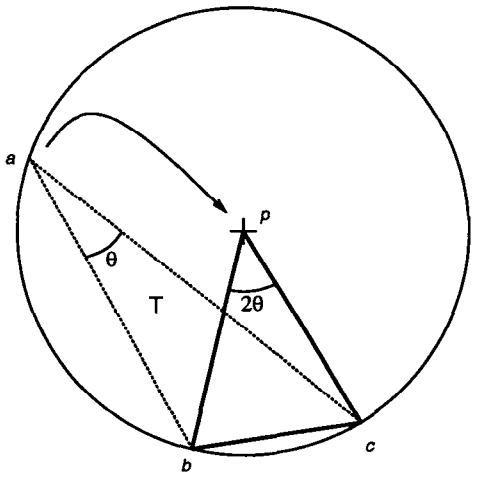


Figure 2: Moving a to circumcenter doubles its angle.

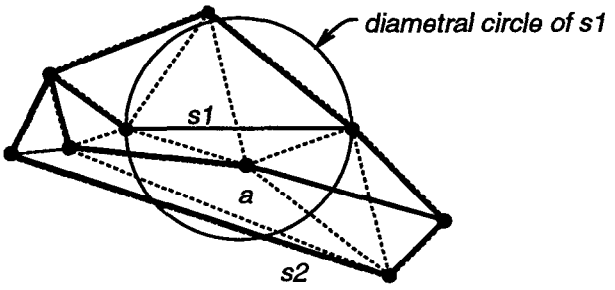


Figure 3: Input PSLG in bold, Delaunay triangulation of its vertices shown dotted. This is not a valid triangulation of the PSLG because s_1 is not present as a Delaunay edge. Vertex a "encroaches upon" both segments s_1 and s_2.

also show sample outputs, discuss some implementation issues and give directions for further work. Due to space constraints, some details are omitted here. The full version is available as [12].

2 The Delaunay Refinement Algorithm

The basic idea of the algorithm is to maintain a triangulation, making local improvements in order to remove the skinny triangles. Each improvement involves adding a new vertex to the triangulation and retriangulating. To pick good locations for these new vertices, we use the following fact of elementary geometry:

FACT 2.1. *If triangle $T = abc$ has $\angle bac = \theta$, and p is the circumcenter of T, then $\angle bpc = 2\theta$. (See Figure 2.)*

As described below, we will generally be adding vertices that are circumcenters, though when such locations are unsuitable, we will instead place new vertices on the input segments.

The particular triangulation we maintain is a Delaunay triangulation, which has been extensively discussed in the literature (see, e.g., [11] or [6]). (At this point, we mention the *constrained* Delaunay triangulation [8],[4], which takes into account segments as well as vertices of the input. Its usefulness as an alternative to Delaunay triangulation will be discussed in a later section.)

Edges of the input PSLG will be referred to as *segments* to distinguish them from the *edges* of the Delaunay triangulation that is maintained. Also, a *vertex* is a vertex of the input or of the growing Delaunay triangulation, whereas a *point* is any point in the plane. During the course of the algorithm, we will maintain a set V of vertices (initialized to the vertices in the input) and a set S of segments (initially those in the input). Vertices are added to the triangulation for two reasons: to improve triangle shape, and to insure that all input segments are present in the Delaunay triangulation (as the union of one or more Delaunay edges).

The two basic operations in the algorithm are to *split* a segment by adding a vertex at its midpoint, and to *split* a triangle with a vertex at its circumcenter. In each case, the new vertex is added to V; and when a segment is split, it is replaced in S by its two subsegments.

For a segment s, the circle with s as a diameter is referred to as its *diametral circle*, and we say that a vertex *encroaches upon* segment s if it lies within the diametral circle of s. Figure 3 illustrates this: the vertex a encroaches upon both segments s_1 and s_2 (only s_1's diametral circle is shown). It is easy to show that a segment not present in the Delaunay triangulation is encroached upon by some vertex.

To simplify the description and analysis of the algorithm, we assume for now that all angles of the input PSLG are at least 90 degrees. In § 5, this restriction will be removed.

Any triangle with an angle below α is called *skinny*. In a nutshell, the algorithm says to split skinny triangles, unless the triangle's circumcenter would encroach upon some input segment, in which case split the segment instead. Here is the algorithm in detail, including subroutines for the two basic operations:

```
subroutine SplitTri(triangle t )
  Add circumcenter of t to V, updating DT(V)

subroutine SplitSeg(segment s )
  Add midpoint of s to V, updating DT(V)
  Remove s from S, add its
      two halves s1 and s2 to S
```

Algorithm *DelaunayRefine*
INPUT: planar straightline graph X;
desired minimum angle bound α
OUTPUT: triangulation of X,
with all angles $\geq \alpha$.
Initialize:
add a bounding square B to X:
compute extremes of X:
xmin, ymin, xmax, ymax
let *span(X)* = max(*xmax-xmin, ymax-ymin*)
let B be the square of side $3 \times span(X)$,
centered on X
add the four boundary segments of B to X
let segment list S = edges of X
let vertex list V = vertices of X
compute initial Delaunay triangulation $\mathcal{DT}(V)$
repeat:
while any segment s is encroached upon:
SplitSeg(s)
let t be (any) skinny triangle (min angle $< \alpha$)
let p be t's circumcenter
if p encroaches upon any
segments $s_1, \ldots, s_k$ **then**
for $i = 1$ to k:
SplitSeg(s_i)
else
SplitTri(t) ($\star$ adds p to V $\star$)
endif
until no segments encroached upon,
and no angles $< \alpha$
output current Delaunay triangulation $\mathcal{DT}(V)$

The execution of the algorithm on a simple polygonal example is described in Figures 4-6. (The observant reader might notice a slight enhancement in the algorithm used in the example: if a segment s is encroached upon by a vertex on another segment, s does not have to be split as long as it appears in the triangulation, and no skinny triangles are present. For instance, the vertex added between (b) and (c) encroaches upon two segments that are never split.)

In the next section, we show that the algorithm halts for any $\alpha < 20°$. (In practice, larger values can be chosen, up to $\alpha \approx 30°$.) Upon termination, all triangles will have aspect ratios at most $|\frac{2}{\sin \alpha}|$, since all angles smaller than α will have been removed. Furthermore, all input segments will be present in the output (as the union of one or more Delaunay edges), since any segments missing from the Delaunay triangulation are encroached upon, and hence get split until they are present. Note that the algorithm specifies no order for splitting skinny triangles. This and other implementation issues will be discussed in § 6.

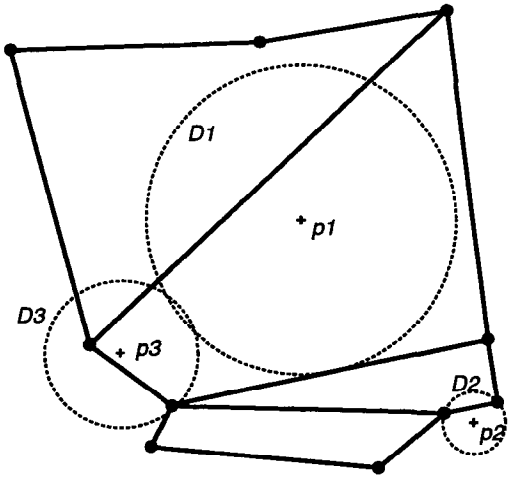


Figure 7: Local feature size at several points. Radius of disk D_i is $lfs(p_i)$.

3 Output Size

In this section we give an upper bound on the number of triangles in the output. The bound depends upon the *local feature size* of the input. At every point in the mesh, the vertex spacing will be close to the local feature size. In the next section, we will show that the local feature size is indeed the desired spacing, since it yields meshes within a constant factor of the optimal size.

DEFINITION 3.1. *Given a PSLG X, The* **local feature size** *at a point p,* $\mathbf{lfs}_X(p)$*, or simply* $\mathbf{lfs}(p)$*, is the radius of the smallest disk centered at p that intersects 2 non-incident vertices or segments of X.*

Figure 7 illustrates the definition of *lfs*( ), the radius of the disk D_i being $lfs(p_i)$. Note D_3 in particular: a smaller disk would intersect 2 segments, but they are incident to each other.

For a given input X, $lfs(p)$ is defined for all points p in the plane, and the entire function, which we refer to as $lfs(X)$, is continuous. If $lfs(p)$ is interpreted as an elevation at p, then $lfs(X)$ is a "not-too-steep" surface above the plane. The following Lemma shows that it has a Lipschitz condition of 1, i.e. the slope in any direction is at most 1.

LEMMA 3.1. *Given any PSLG X, and any two points p and q in the plane,*

$$lfs(q) \leq lfs(p) + \text{dist}(p, q),$$

where $\text{dist}(p, q)$ *is the distance between p and q.*

Proof. See Figure 8. The disk D of radius $r = lfs(p)$ centered at p intersects 2 non-incident portions of X.

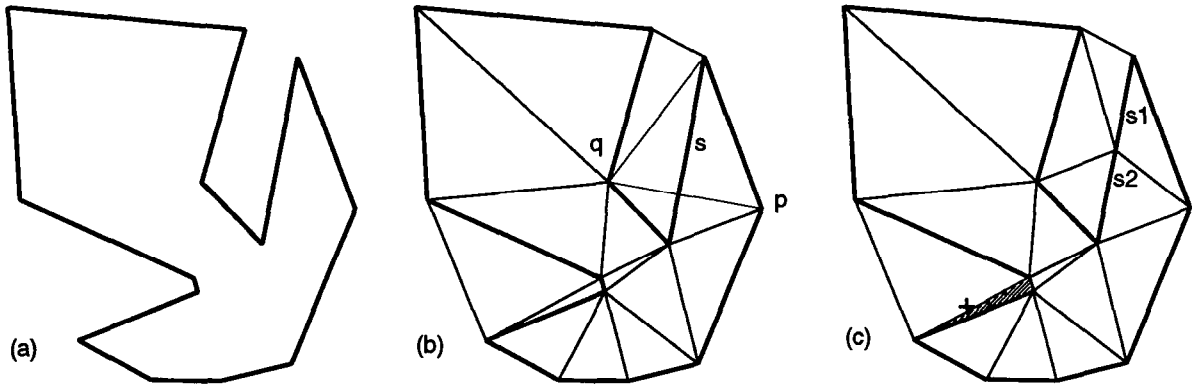


Figure 4: Execution of the algorithm on a simple example. For clarity, no bounding box is used. In each picture, the input is shown in thick lines, the current Delaunay triangulation is overlayed in thin lines. The initial Delaunay triangulation is shown in (b). Note that input segment s is not a Delaunay edge. This is because s is encroached upon by vertices p and q, so s is split at its midpoint into two segments $s1$ and $s2$, yielding the updated Delaunay triangulation shown in (c). Now we choose skinny triangles to be split. The shaded triangle has the smallest angle, 5.9 degrees. A cross indicates its circumcenter.

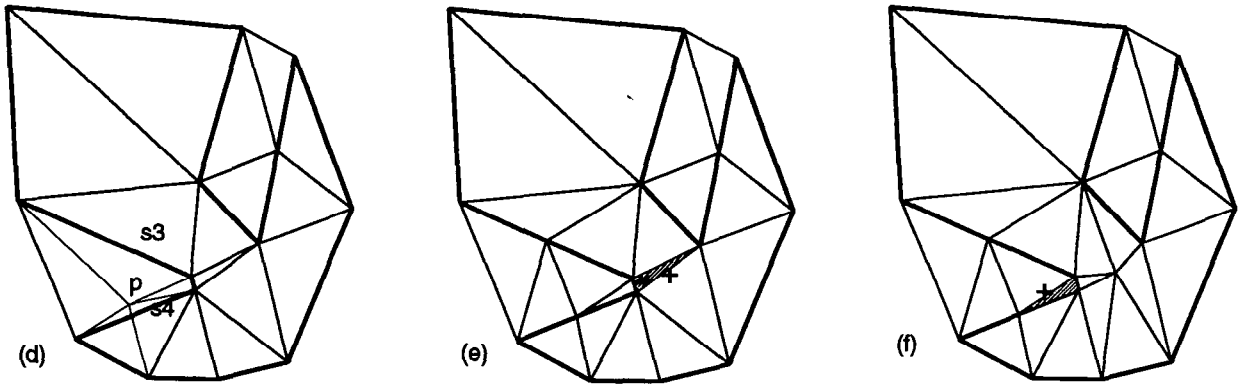


Figure 5: In (d), we see what would happen if the skinny triangle's circumcenter p were added to the triangulation: it would encroach upon two segments $s3$ and $s4$. These segments are split instead of adding p, yielding the triangulation shown in (e). The shaded triangle has the smallest angle, 9.8 degrees. Splitting that triangle yields (f), minimum angle now 11.8 degrees.

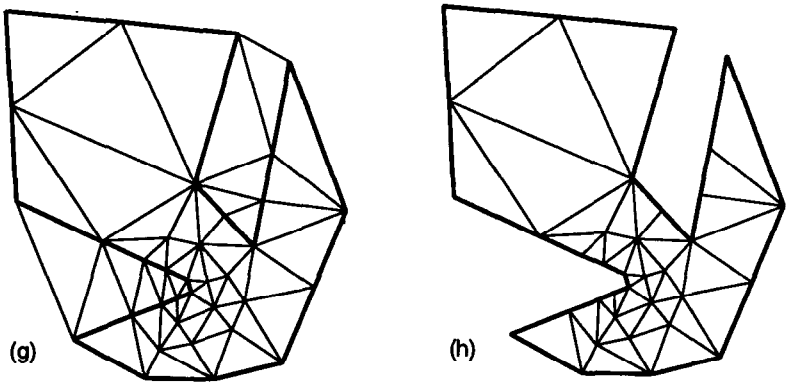


Figure 6: Allowing the execution to continue until all angles are at least 25 degrees yields (g), and optionally, external triangles can be removed, as shown in (h).

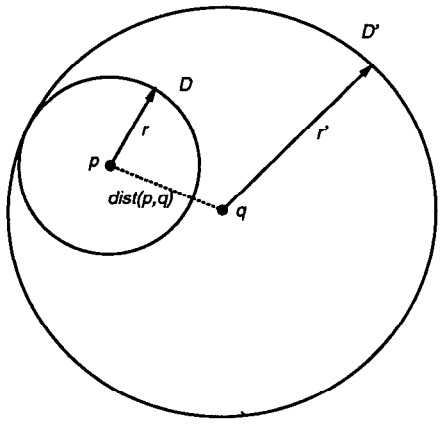


Figure 8: Lemma 3.1: local feature size is not too "steep".

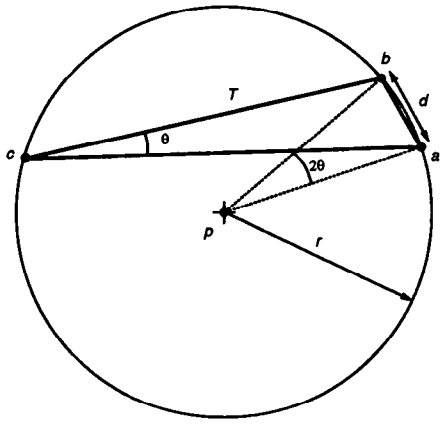


Figure 9: Lemma 3.2 Case 1: p added as circumcenter of triangle T with small angle $\theta < \alpha$.

The disk D' of radius $r' = r + \text{dist}(p,q)$ centered at q contains D and hence intersects the same portions of X. So $lfs(q) \leq r'$. Putting this together, we have

$$lfs(q) \leq r' = r + \text{dist}(p,q) = lfs(p) + \text{dist}(p,q).$$

The next lemma is the crux of the mesh size analysis. It shows that as each vertex is added, it is at the center of a "vertex-free" circle of radius at least a constant fraction of the local feature size. Thus the density of added vertices is bounded by the geometry of the input. We emphasize that adding vertices does not change the $lfs(\,)$ function, since it is determined by the input.

LEMMA 3.2. *For fixed constants C_T and C_S, specified below, the following statements hold:*

- *At initialization, for each input vertex p, the distance to its nearest neighbor vertex is at least $lfs(p)$.*
- *When a point p is chosen as the circumcenter of a skinny triangle, the distance to the nearest vertex is at least $\frac{lfs(p)}{C_T}$. (p may be added to the triangulation, or may be rejected because it encroaches upon some segment.)*
- *When a vertex p is added as the midpoint of a split segment, the distance to its nearest neighbor vertex is at least $\frac{lfs(p)}{C_S}$.*

Proof. For any input vertex p, the distance to its nearest neighbor vertex is at least $lfs(p)$, by definition of the $lfs(\,)$ function. This is the base case of the lemma. For vertices added later, we assume the lemma is true for all previous vertices.

Case 1: We first consider the case where p is the circumcenter of a skinny triangle T. Since p is at the center of T's Delaunay circle, its nearest neighbors are the vertices of T (see Figure 9), at a distance of r. Assume the vertices of T are a, b, c, with the smallest angle θ at c. Then the shortest edge of T is from a to b. Call its length d. Without loss of generality, assume a was added after b (or that both were in the input). We will use the fact that a and b are close together to bound $lfs(a)$ in each of several cases, which in turn will bound $lfs(p)$.

Case 1(a): a was a vertex of the input. Then so was b, so $lfs(a) \leq d$.

Case 1(b): a was added as a circumcenter of some triangle with circumradius $r' \leq d$ (since b was outside that triangle's circumcircle). We can apply this lemma to a, yielding $lfs(a) \leq r'C_T \leq dC_T$.

Case 1(c): a was the midpoint of a segment that was split. Applying this lemma to a now yields $lfs(a) \leq dC_S$, since b was outside a's vertex-free circle.

So we have $lfs(a) \leq dC_S$, assuming we have the condition $\boxed{C_S \geq C_T \geq 1}$, which we will be able to satisfy below. By Fact 2.1, $\angle apb = 2\theta$, so simple geometry gives $\sin\theta = \frac{d}{2r}$. Putting this all together, Lemma 3.1 gives

$$lfs(p) \leq lfs(a) + r$$

using our bound for $lfs(a)$ we have

$$lfs(p) \leq dC_S + r$$

$$= 2rC_S \sin\theta + r$$

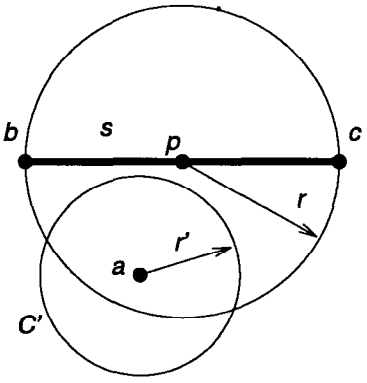


Figure 10: Lemma 3.2 Case 2: p added to split segment s which is encroached upon by a.

or, since $\theta < \alpha$,

$$r \geq \frac{lfs(p)}{1 + 2C_S \sin\alpha}$$

So we get the desired bound on r as long as we can satisfy the condition $\boxed{C_T \geq 1 + 2C_S \sin\alpha}$.

Case 2: We now consider the case where a vertex p is added to split a segment s. Segment s is split because some vertex or circumcenter a is inside s's diametral circle, which has radius r. (See Figure 10.) We have two cases for a:

Case 2(a): a lies on some segment t, which cannot be incident to s, since we are assuming that all angles in the input PSLG are at least 90°. (Any segment incident to s makes a larger angle, and hence would be completely outside the diametral circle.) So there are two non-incident segments, one containing p, the other containing a, within a distance r of each other. Thus $lfs(p) \leq r$. Above, we have assumed the condition $C_S \geq 1$, so this case is done.

Case 2(b): a was a circumcenter, proposed for addition to the Delaunay triangulation, but rejected because it lay inside the diametral circle of s. Suppose it was the center of circle C' with radius r'. By this applying this lemma to a, we know that $r' \geq \frac{lfs(a)}{C_T}$. Also, b and c, the endpoints of S, must be outside the Delaunay circle C', so $r' \leq \sqrt{2}r$. Lemma 3.1 gives

$$\begin{aligned} lfs(p) &\leq lfs(a) + r \\ &\leq r'C_T + r \\ &\leq \sqrt{2}rC_T + r \end{aligned}$$

or

$$r \geq \frac{lfs(p)}{1 + \sqrt{2}C_T}$$

This yields the correct bound on r, provided that $\boxed{C_S \geq 1 + \sqrt{2}C_T}$.

It can be checked that the 3 boxed conditions can be simultaneously satisfied for any $\alpha \leq 20°$. For instance, $C_T = \frac{1+2\sin\alpha}{1-2\sqrt{2}\sin\alpha}$, $C_S = \frac{1+\sqrt{2}}{1-2\sqrt{2}\sin\alpha}$ will work. For $\alpha = 10°$, we can choose $C_T = 2.8$, and $C_S = 5$.

Since $C_T \leq C_S$, the lemma shows that when a vertex p is added, no other vertex is within a distance $\frac{lfs(p)}{C_S}$ of p. The following theorem shows that vertices added later cannot get much closer to p.

THEOREM 3.1. *Given a vertex p of the output mesh, its nearest neighbor vertex q is at a distance at least $\frac{lfs(p)}{C_S+1}$.*

Proof. The previous lemma handles all but the case when q was added after p, in which case we can apply the lemma to q and get

$$\text{dist}(p,q) \geq \frac{lfs(q)}{C_S}$$

Lemma 3.1 gives a bound for $lfs(q)$ in terms of $lfs(p)$ and q's distance from p, so

$$\text{dist}(p,q) \geq \frac{lfs(p) - \text{dist}(p,q)}{C_S}$$

rearranging finishes the proof: $\text{dist}(p,q) \geq \frac{lfs(p)}{C_S+1}$

The next theorem uses an area argument to yield a bound on the number of vertices. Intuitively, a region of small local feature size requires small triangles, i.e. the vertex spacing should be proportional to the local feature size. Thus the triangle density in the mesh is proportional to the inverse of the square of the local feature size. So we will "charge" the cost for each vertex to the local feature size around it.

THEOREM 3.2. *The number of vertices in the output mesh is at most*

$$C_1 \int_B \frac{1}{lfs^2(x)} dx,$$

where B is the region enclosed by the bounding square, and C_1 is a constant to be specified.

Proof. The previous theorem says that each vertex p in the mesh is at the center of an open disk of radius $\frac{lfs(p)}{C_S+1}$ that contains no other vertex. Halving the radii gives non-intersecting disks: let D_p be the open disk of radius $r_p = \frac{lfs(p)}{2(C_S+1)}$ centered on p. Since at least one-fourth of each D_p is contained in the bounding square B, we can lower bound the integral by summing its value in the disks D_p for every p in the vertex set V:

$$\int_B \frac{1}{lfs^2(x)} dx \geq \frac{1}{4} \sum_{p \in V} \int_{D_p} \frac{1}{lfs^2(x)} dx$$

By Lemma 3.1, the maximum $lfs(\ )$ attainable in D_p is $lfs(p) + r_p$, which gives a bound for $\int_{D_p}$:

$$\int_{D_p} \frac{1}{lfs^2(x)} dx \geq \text{area}(D_p) \frac{1}{\max_{x \in D_p}\{lfs^2(x)\}}$$

$$\geq \text{area}(D_p) \frac{1}{(lfs(p) + r_p)^2}$$

Using $\text{area}(D_p) = \pi {r_p}^2$, plugging in for r_p, and cancelling yields

$$\int_{D_p} \frac{1}{lfs^2(x)} dx \geq \frac{\pi}{(2C_S + 3)^2}$$

Substituting back in for the entire integral,

$$\int_B \frac{1}{lfs^2(x)} dx \geq \frac{1}{4} \sum_{p \in V} \frac{\pi}{(2C_S + 3)^2}$$

$$= \frac{\pi}{4(2C_S + 3)^2} \sum_{p \in V} 1$$

Since the summation merely counts the number of vertices in the output mesh, the theorem holds if we choose the constant $C_1 \geq \frac{4(2C_S+3)^2}{\pi}$.

4 Size-Optimality

We omit this section except to state the main result. The details are given in [12]. The analysis is similar to that given by Mitchell and Vavasis in [10] for their 3D algorithm. A series of technical lemmas are proved, with the final lemma stating that for any point q within any bounded aspect ratio triangulation $\mathcal{T}$ of a given input PSLG X, the "triangle size" at q is within a constant factor of $lfs(q)$. This is what we showed in the previous section for the triangulation output by our Delaunay refinement algorithm and hence we have the following theorem.

THEOREM 4.1. *Given $\alpha \leq 20°$, and input X, suppose $\mathcal{T}$ is any triangulation of X with minimum angle bound α. There is a constant C_α such that if $\mathcal{T}$ has N triangles, then our triangulation has at most $C_\alpha \cdot N$ triangles. Letting $\mathcal{T}$ be the triangulation with fewest possible triangles shows that our triangulation is within a factor C_α of optimal.*

The constant factor C_α depends on the choice of α, but not on X, i.e. our algorithm is optimal on every input, not just in the worst case. We discuss C_α more in § 6.

5 Corner-Lopping and Riemann Sheets

In this section we discuss two issues that must be resolved so that our algorithm produces optimal bounded aspect ratio triangulations for general 2-dimensional inputs. These problems have been tackled by previous researchers, and here we briefly sketch how to adapt their solutions to our algorithm. More details are given in [12].

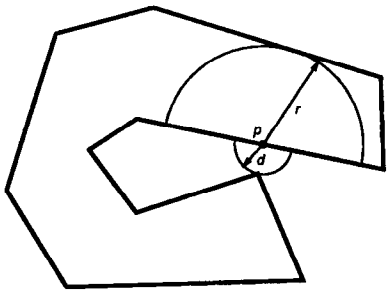


Figure 11: Do the polygon's two "arms" determine a small feature at p?

First, we must deal with small input angles reasonably (recall that we unreasonably assumed all angles were at least 90°!). Also, though input angles smaller than our minimum angle bound of α cannot be removed, they should be dealt with gracefully. We deal with small angles by "lopping off" the sharp corners as follows. The input is preprocessed so as to "shield" any vertex p with a small angle by committing in advance to a specific triangulation around p. This was previously done in [3] with a circle around p, and in the 3D algorithm of [10], a cube was used. We can adapt the technique of [3], using circles of radius $\frac{lfs(p)}{3}$, so that the size-optimality property will still hold for the output.

The second issue relates to our definition of local feature size in non-convex polygons: in Figure 11, do the two "arms" of the polygon generate a small feature at p? Our definition says they do, and produces small triangles around p accordingly. This could be suboptimal if only an interior triangulation of p is desired. In particular, the local feature size at p should be r, rather than d, as our definition states. As in [10], we modify the definition to use the *geodesic distance* to the 2 nearest non-incident portions of the input. The geodesic distance is measured along the shortest path that stays within the region to be triangulated (e.g. the interior of the polygon). We modify our algorithm to work using a *constrained Delaunay triangulation*, say by using *Riemann sheet* techniques, similar to [10].

6 Implementation and Discussion

The basic algorithm of § 2 leaves unspecified some issues concerning its implementation. We now discuss these issues in general, and describe our own implementation.

An incremental Delaunay triangulation algorithm

is ideal as a basis for our algorithm, Guibas and Stolfi [7] give a useful implementation. The algorithm allows skinny triangles to be split in any order; by always splitting the one with the smallest angle, the algorithm trades off nicely between mesh size and shape: the overall minimum improves as the algorithm continues to run (see Figure 13).

The detection of "encroached upon" segments (those containing a point in their diametral circle) can be done efficiently by checking some local criteria during each update of the Delaunay triangulation. A segment is encroached upon if either: (1) It is not present as a Delaunay edge (e.g. s_1 in Figure 3); or (2) It is present, but opposite an obtuse angle in a Delaunay triangle (e.g. s_2 in Figure 3).

We have implemented our algorithm, except for the corner-lopping and Riemann sheet modifications. Though the corner-lopping was required for the optimality analysis in the presence of input angles below 90°, in practice, redefinition of a *skinny* triangle to exclude small input angles works well in most cases. Some configurations with input angles below 10° can cause too many points to be added near that angle.

Figure 12 shows the output on two examples, each with a minimum output angle of 20°. The example on the right has several input angles near 15°, which remain in the output.

We now turn to the question of how size-optimal the algorithm is. The examples of Figure 12 seem to be within a factor of 2–4 times the minimum possible size for the given angle bound, so the "true" optimality constant lies somewhere between there and the value of C_α of § 4. For a minimum angle bound of 20°, the best explicit value we were able to prove for the optimality constant C_α was $C_\alpha \approx 2.1 \times 10^{25}$. Though this is the first explicitly stated optimality constant for a bounded aspect ratio triangulation algorithm, the value is clearly meaningless as a practical guarantee. Examination of the analysis shows much slack that might be tightened up, for example a constant of A^{2K+6}, with $A \approx 6$, $K \approx 4$, that we suspect can be replaced by 2^K or A^2, but even shaving off 10 or 15 orders of magnitude would not yield a useful value for C_α. One would really like a stronger proof technique.

We can make a non-rigorous argument about output size using the constant C_S of § 4. It bounds the density of points along input segments, and its value indicates that at most 5 "layers" of triangles will appear between 2 nearby input vertices. In Figure 12, we see that short segments are not broken up at all, and so there is usually only 1 layer. This contrasts with the algorithm in [3], in which each input vertex must be isolated within a 5-by-5 grid of quadtree squares, yielding at least 2–3 layers of triangles between any two vertices.

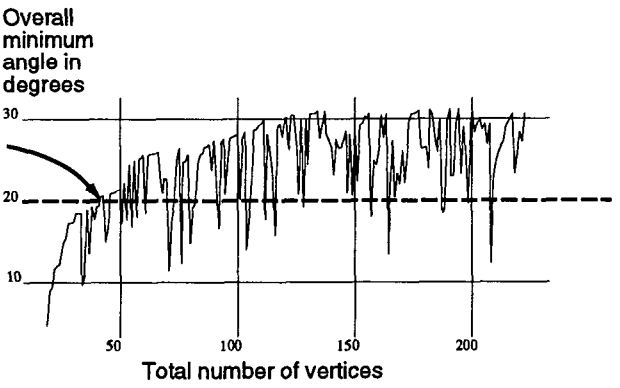


Figure 13: Progress of minimum angle, during a typical run. Arrow shows when a minimum angle of 20° is first reached.

Additional evidence concerning the behavior of the algorithm comes from Figure 13, which charts the overall minimum angle during a lengthy run on a simple input with about 15 vertices. We see the minimum rise to about 30° and then level off, except for frequent downward spikes when a small angle gets divided in two, then quickly improved. The optimality proof says that eventually, no spike will drop below the dotted line (here, for $\alpha = 20°$), which would be far to the right of the plotted portion of the graph. The arrow points out when the algorithm would actually halt for this case.

We have not analyzed the running time of the Delaunay refinement algorithm in detail. The worst-case running time for incremental Delaunay triangulation is $O(M^2)$, where M is the output size. In practice, such algorithms usually run much faster [7]. Much of the time is typically taken up locating the triangle containing the new point. For non-input vertices, this is simplified in our algorithm by starting at the skinny triangle or encroached upon segment being split.

7 Conclusion

We have presented a new Delaunay refinement algorithm for bounded aspect ratio triangulation of planar straightline graphs. The algorithm is very simple, and quite different from previous techniques.

There are many opportunities for further work. Can the corner-lopping preprocessing step be avoided by some technique that splits edges more cleverly than always at the midpoints? The algorithm is well-suited applications involving adaptive analyses that increase mesh density in regions of large error. For problems with a solution that changes, mesh *reduction* is also useful—is there a Delaunay based criterion that indi-

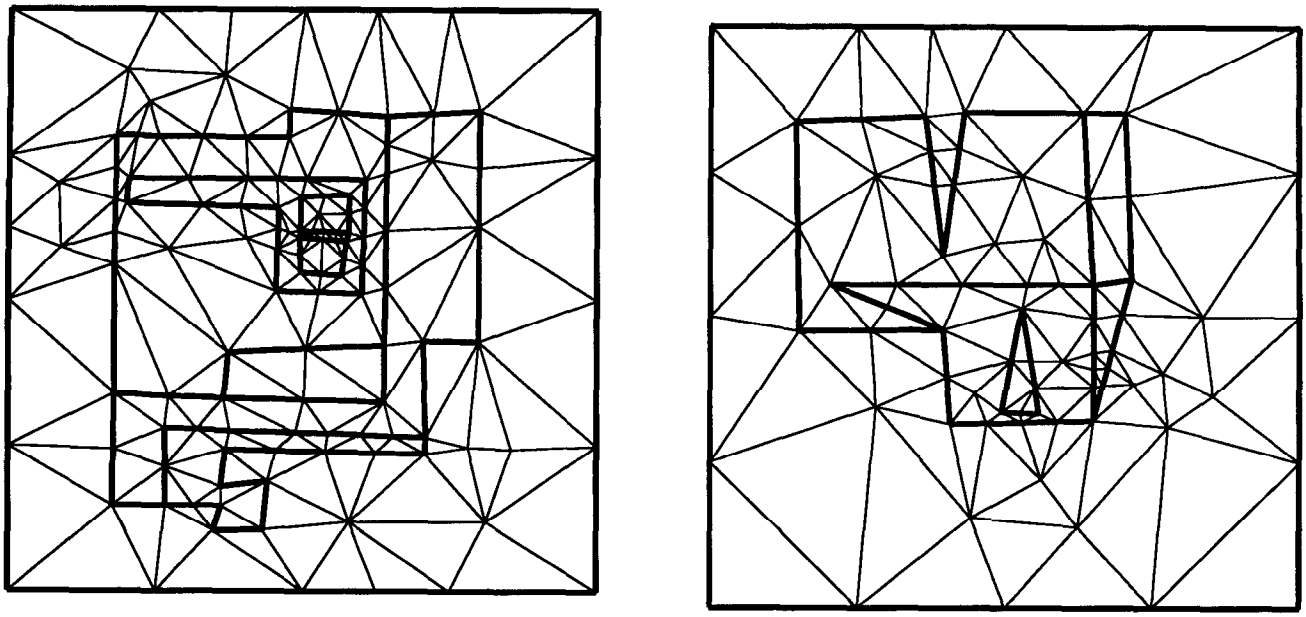


Figure 12: Output on two sample PSLGs, minimum angle $\approx 20°$.

cates good vertices to delete from the mesh? There are several questions regarding the size-optimality constants: Can the analysis be significantly improved? Are there lower bounds for bounded-aspect ratio triangulation, even for specific inputs like two ϵ-separated points centered in the unit square? Finally, can the Delaunay refinement algorithm be generalized to work for 3D triangulation of polyhedra?

8 Acknowledgements

I would particularly like to thank Raimund Seidel, for many productive discussions. Helpful suggestions were provided by Balas Natarajan and Marshall Bern. The development of the algorithm was aided by the Voronoi diagram implementation of Steve Fortune (available via *netlib*).

References

[1] T.J. Baker, E. Grosse, and C.S. Rafferty. Nonobtuse triangulation of polygons. *Disc. and Comput. Geom.*, 3:147–168, 1988.

[2] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.Z. Du and F.K. Hwang, editors, *Euclidean Geometry and the Computer (to appear)*. World Scientific, (1992?).

[3] M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 231–241. IEEE, 1990. To appear in *J. Comp. System Science*.

[4] L.P. Chew. Constrained Delaunay triangulation. *Algorithmica*, 4:97–108, 1989.

[5] L.P. Chew. Guaranteed-quality triangular meshes. Technical report, Cornell University, 1989. No. TR-89-983.

[6] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[7] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4:74–123, 1985.

[8] D.T. Lee and A. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, 1:201–217, 1986.

[9] E. Melissaratos and D. Souvaine. Coping with inconsistencies: A new approach to produce quality triangulations of polygonal domains with holes. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 202–211. ACM, 1992.

[10] S.A. Mitchell and S.A. Vavasis. Quality mesh generation in three dimensions. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 212–221. ACM, 1992. Full version in Cornell Tech. Report TR 92-1267, Feb. 1992.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry – an Introduction*. Springer-Verlag, New York, 1985.

[12] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. Technical Report UCB/CSD 92/694, Computer Science Division, 570 Evans Hall, University of California, Berkeley, CA 94720, June 1992.