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

Jim Ruppert
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720, USA

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 "Steiner points" added 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 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").

*This work was supported by an NSF Presidential Young Investigator Grant CCR-90-58840.

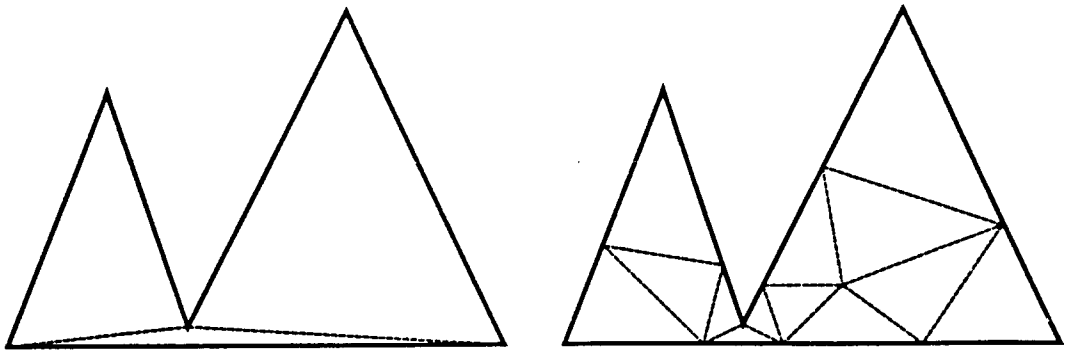


Figure 1: A triangulation without Steiner points may require skinny triangles.

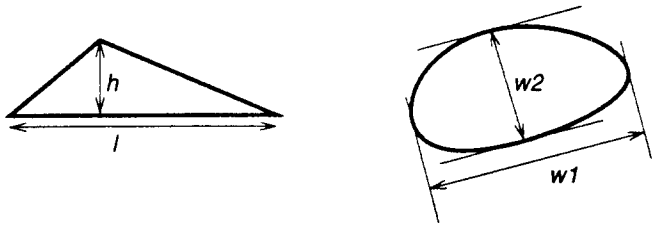


Figure 2: Aspect ratio of a triangle is $\frac{l}{h}$, where l is the length of the longest edge and h is the shortest altitude. For a general convex object, use the longest and shortest width: $\frac{w_1}{w_2}$.

For instance, skinny triangles are known to slow the convergence of finite element computations. A second example is the solution of systems of linear equations defined by the vertices and edges of a triangulation. If all the triangles have bounded aspect ratio, then the fast algorithm of [17] can be used to produce small "separators", which yield efficient orderings for nested dissection and other solution techniques. Other applications ask for a *non-obtuse* triangulation, in which all angles are $\leq 90°$. Such triangles contain their circumcenters, and by connecting adjacent circumcenters, we obtain a *planar dual* graph in which the dual edges are perpendicular to the triangulation edges, simplifying certain calculations.

Our interest in this paper is the decomposition of 2-dimensional objects into triangles. This is called *mesh generation* in most engineering literature, since the triangles form a mesh used for the finite element method. (Specifically, we are referring to *unstructured* mesh or grid generation.) In computational geometry, the same problem is called *triangulation* or *Steiner triangulation*, since we will allow *Steiner points*: vertices of the mesh that are not vertices of the input. We will use the terms "mesh generation" and "triangulation" interchangeably. A triangulation must be a *simplicial complex*, that is, any two triangles may intersect only in a common edge, a common vertex, or not at all. Figure 1 shows two triangulations of a polygon, one using Steiner points. *Quality mesh generation* has been used to describe techniques that offer a guarantee on some measure of the triangulation's shape.

Typically the *shape* of a triangulation is determined by its worst-shaped triangle. Various measures of triangle shape have been studied, including minimum angle, maximum angle and aspect ratio. The *aspect ratio* of a triangle is the length of the longest edge divided by the

length of the shortest altitude (see Figure 2). The aspect ratio is nice because it generalizes to any convex object in any dimension: it is the ratio of the longest width to the shortest width, where a *width* is the distance measured between two parallel supporting hyperplanes (lines in our 2-dimensional figure). We will generally be concerned with bounding the minimum angle α in any triangle; this guarantees a maximum angle $\leq \pi - 2\alpha$ and an aspect ratio between $|\frac{1}{\sin\alpha}|$ and $|\frac{2}{\sin\alpha}|$.

We give an algorithm for the problem of triangulating planar straightline graphs (PSLGs) such that all triangles have bounded aspect ratio. PSLGs include polygons, polygons with holes, and complexes (objects made of multiple polygons); dangling edges and isolated vertices are also allowed. Steiner points may be used, but not too many: we desire a triangulation that is *size-optimal*: the number of triangles is within a constant factor of the minimum number possible in any bounded aspect ratio triangulation of the given PSLG. Algorithms with both size and shape guarantees have been previously shown for triangulating point sets [5], polygons [5], polygons with holes [15], PSLGs [15, 4], and even polyhedra in 3 dimensions [18]. All of these techniques are based on grids or quadtrees, whereas we use a "Delaunay refinement" approach, similar to [7]. Theoretically speaking, our algorithm matches the PSLG algorithms of [15] and [5](modified as mentioned in [4]), but the Delaunay refinement approach offers several advantages, as discussed in the next section. Our algorithm is very simple; essentially it boils down to the following: Start with the Delaunay triangulation of the vertices of the input. Repeatedly add circumcenters of "skinny" triangles to the triangulation, unless they are "too close" to an input edge, in which case divide that edge at its midpoint.

2 Relation to Previous Work

The literature includes quite a few papers on the subject of mesh generation, many describing heuristics intended to produce nicely-shaped triangulations. Ho-Le gives a survey of some practical approaches in [14], see also the references in [4]. Here we mention only those algorithms that carry guarantees on their output size or shape, or both. A more detailed review of theoretical results in mesh generation is given in the excellent survey by Bern and Eppstein in [4].

We group the work on guaranteed quality mesh generation into 3 categories:

1. *No Steiner points allowed.* Since no shape bounds can be guaranteed in this case, we only touch on such techniques.

2. *Rectangular decomposition based.* The space including the input is first subdivided into boxes, which are then triangulated. This includes grid and quadtree techniques.

3. *Delaunay triangulation based.* This includes our "Delaunay refinement" approach, which generalizes an earlier technique due to Chew.

No Steiner Points Allowed

First we note that a minimum size triangulation can be generated by polygon triangulation, since an n-vertex polygon can always be divided into $n-2$ triangles without using any Steiner points. Many algorithms are known for polygon triangulation, with the most asymptotically efficient being the linear time algorithm of Chazelle [6]. Other algorithms are discussed in the book by Preparata and Shamos [20]. Since a given polygon might have many different triangulations, one might wish to optimize some shape criterion. For instance, an algorithm for minimizing the maximum angle was given by Edelsbrunner, Tan and Waupotitsch in [10]. Their *edge insertion* technique optimizes other criteria as well, as was later shown by Bern, Edelsbrunner, Eppstein, Mitchell and Tan [3]. Several other optimal non-Steiner triangulation algorithms are discussed by Bern and Eppstein in their survey paper [4].

Since a non-Steiner triangulation may use only the vertices of the input, there are cases in which arbitrarily skinny triangles are unavoidable. For example, the polygon in Figure 1 has only one triangulation, and the largest aspect ratio is determined by the small distance from the base edge to the bottom of the notch. On the right we see that a careful choice of Steiner points (here picked manually) can yield better-shaped triangles.

Techniques Using Rectangular Decompositions

There are two basic types of rectangular decompositions used for guaranteed quality mesh generation: a *uniform grid*, in which all squares are the same size, and a *quadtree*, where they are recursively subdivided into different sizes. The basic mesh generation steps are the same in both cases:

1. **Refine** the decomposition until each box contains a small enough portion of the input.
2. **Warp** boxes to align with the input's boundaries.
3. **Triangulate** boxes, using a variety of special cases.

The first algorithm to give a shape guarantee was due to Baker, Grosse and Rafferty [2]. They give a technique for producing a non-obtuse triangulation of polygons, in which all angles are $\leq 90°$. In addition, the smallest angle is $\geq 13°$. (Of course, this is only possible if all angles in the input are $\geq 13°$.) Together, these bounds guarantee an aspect ratio of ≤ 4.6. Their algorithm requires a uniform square grid over the polygon, with spacing roughly equal to the smallest feature present in the polygon. 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 [5] they give a method for triangulating polygons so that every triangle has aspect ratio ≤ 5. In addition, they give a clever analysis to show that the number of triangles produced is within a constant multiplicative factor of optimal. By *optimal* we mean the minimum number of triangles in *any* triangulation of the given input achieving the same aspect ratio bound. One of the key ideas in their algorithm is to replace the uniform grid of [2] 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

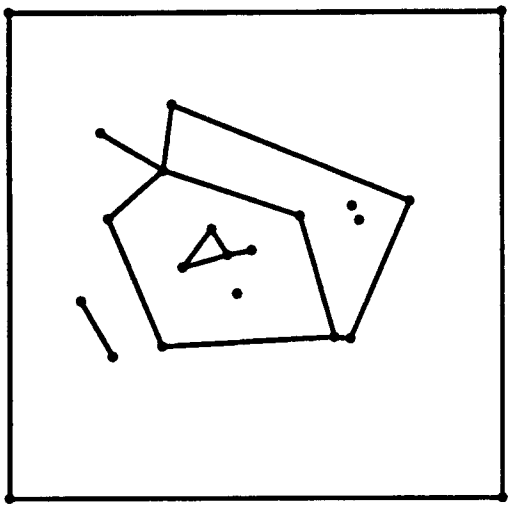
bounded in the output. Melissaratos and Souvaine [15] give some extensions to the quadtree algorithm. Mitchell and Vavasis [18] have shown how to extend the quadtree technique of [5] to *octrees* in 3 dimensions. They give an algorithm for size-optimal bounded aspect ratio triangulation of polyhedra.

Delaunay-Based Techniques

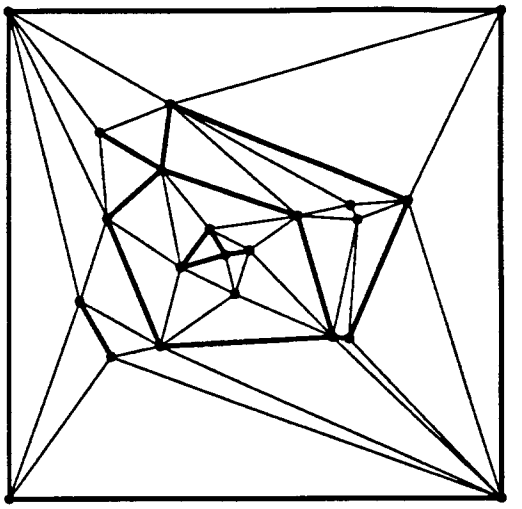
The well-known Delaunay triangulation (defined in Section 3, see also [20]) has been used for many different problems. Its application to mesh generation has been discussed in [1], [7], [11], [16], [8], and many others. This is in spite of a fundamental shortcoming: because a Delaunay triangulation is defined only for a set of points, it does not take into account any edges (or faces in 3D) of the object to be meshed. As a result, the Delaunay triangulation of the input vertices alone may not contain portions of the boundary which are essential in a mesh (see Figure 5 for an example). What is needed is a *conforming Delaunay triangulation*: one in which all input edges are present, as the union of one or more Delaunay edges. The general strategy for producing a conforming triangulation is to place additional Steiner points on the edges of the input. Edelsbrunner and Tan [9] gave an algorithm which uses $O(n^3)$ Steiner points (n is the number of vertices in the input). They also show a lower bound of $\Omega(n^2)$. These are *combinatorial* bounds because they depend only on the combinatorial complexity of the input; algorithms with size bounds dependent upon *geometric* factors have been given in Nackman and Srinivasan [19] and Chew [7]. A conforming triangulation is necessary for mesh generation, but is not sufficient to guarantee well-shaped triangles.

The first mesh generation algorithm with both size and shape bounds was based on Delaunay triangulation, given by L.P. Chew in [7]. Chew's algorithm triangulates polygons such that all angles are between 30 and 120 degrees. Since our algorithm is an extension of Chew's, we review it here. First, a parameter h is chosen. It must be smaller than the smallest feature present in the input. A preprocessing step subdivides edges of the input to lengths between h and $\sqrt{3}h$. Next, a Delaunay triangulation of the vertices is computed. Then, if any triangle has a circumradius $> h$, its circumcenter is added to the triangulation. This process of adding a point and updating the Delaunay triangulation is repeated until all circumradii are less than h. Chew shows that in the final triangulation, all edges are between h and $2h$, and all angles are between 30° and 120°. This guarantees aspect ratios of $\leq 2\sqrt{3} \approx 3.5$. 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 in [2], a uniform mesh may have many more triangles than are necessary. See Figure 3(c) for an example of a uniform mesh. This mesh was produced by a modified version of our algorithm, but is not too different from meshes produced by the algorithm given in [7].

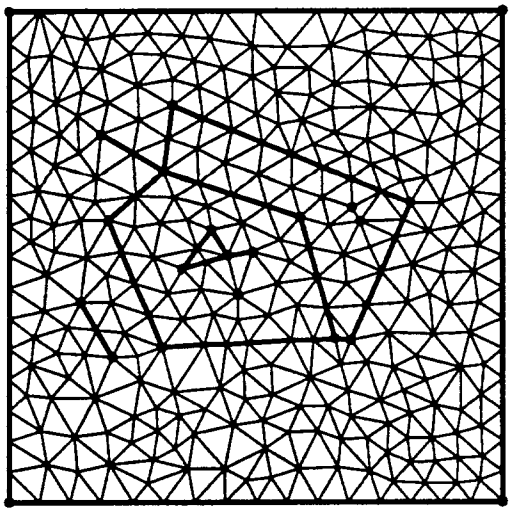
The algorithm we describe in the next section extends Chew's work. We adopt the idea of placing points at Delaunay circumcenters, but according to a different criterion which allows triangles of varying sizes. An example output mesh is shown in Figure 3(d). The algorithm triangulates planar straightline graphs (PSLGs) such that all triangles in the output have angles between α and $\pi - 2\alpha$, where α is a parameter that can be chosen by the user. In practice, the algorithm succeeds for $\alpha \leq 30°$ (assuming all input angles are at least α). For



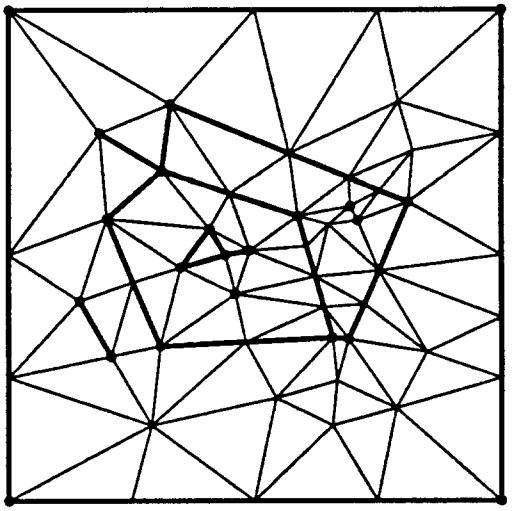
(a) Typical input PSLG and bounding box



(b) Typical triangulation without Steiner points



(c) Uniform mesh with minimum angle 22.5 degrees



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

Figure 3: Sample input planar straightline graph (PSLG), and several triangulations of it.

$\alpha \leq 20°$, we can show that the output 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 3(a)). Theoretically speaking, our algorithm essentially matches the PSLG algorithms of [15] and [5](modified as mentioned in [4]), but it is distinguished from them in the following respects:

- The Delaunay refinement approach is fundamentally different from the quadtree techniques.
- It is much simpler. With fewer special case constructions, it is easier to implement.
- It generally produces fewer triangles.
- It is "parameterized": the user can ask for the "best mesh" with a given number of triangles. In this way, the algorithm takes advantage of the inherent mesh size/shape tradeoff.
- 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. Edges in a Delaunay refinement mesh have no preferred orientation.
- It 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 Section 6.)

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 3(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 Section 6.)

3 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 1 If triangle $T = abc$ has $\angle bac = \theta$, and p is the circumcenter of T, then $\angle bpc = 2\theta$. (See Figure 4.)

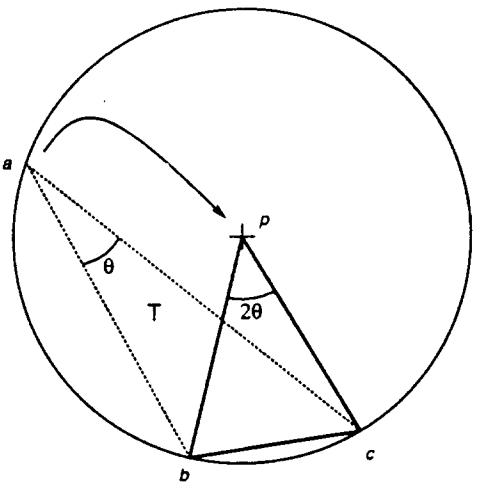


Figure 4: Moving a to circumcenter doubles its angle.

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., [20] or [12]). Several equivalent definitions of the Delaunay triangulation can be given, for our purposes the "circumcircle" characterization is most useful:

Definition 1 *Given a set V of points in the plane, the* **Delaunay triangulation** $\mathcal{DT}(V)$ *consists of all triangles through 3 points whose circumcircle contains no other point of V.*

Assuming the point set is non-degenerate (i.e. no 4 points are co-circular), the Delaunay triangulation is unique and is a triangulation of the entire convex hull of the point set. The degenerate case can be handled by triangulating co-circular points arbitrarily.

(At this point, we note that the use of the *constrained* Delaunay triangulation, which takes into account segments as well as vertices of the input, 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 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 (i.e. so that we have a *conforming* Delaunay triangulation).

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.

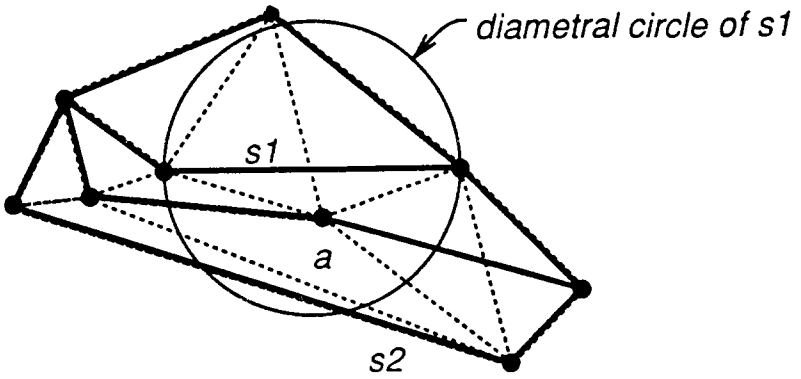


Figure 5: Input PSLG in bold, Delaunay triangulation of its vertices shown dotted. The triangulation is non-conforming because s_1 is not present as a Delaunay edge. Vertex a "encroaches upon" both segments s_1 and s_2.

Figure 5 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 menaced 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 Section 6, 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 3 subroutines it uses:

```
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

subroutine repair( )
    repeat while any segment s of S is encroached upon in DT(V):
       SplitSeg(s)
```

Algorithm *DelaunayRefine*

INPUT: planar straightline graph X;
desired minimum angle bound α

OUTPUT: triangulation conforming to X, with all angles $\geq \alpha$.

1. Initialize:
   add a bounding square to X: 3 times as large and concentric with X
   let segment list S = edges of X
   let vertex list V = vertices of X

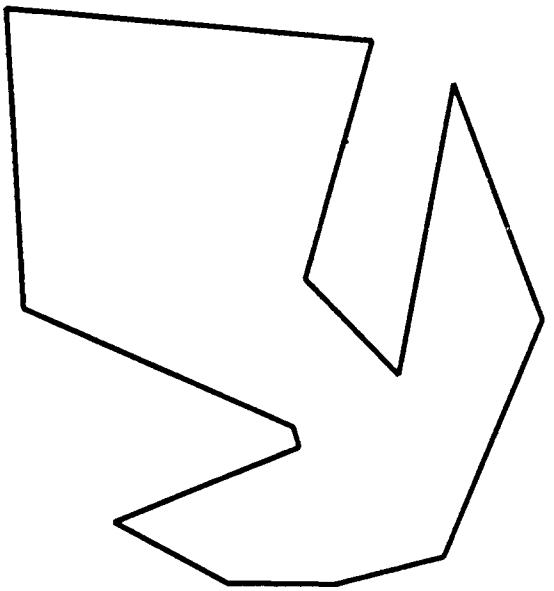
```
2.  repair( )
3.  repeat until all angles ≥ α:
        let t be (any) skinny triangle (min angle < α)
        let p be t's circumcenter
        SplitTri(t) (* adds p to V *)
        if p encroaches upon any segments s_1,...,s_k
            remove p from V
            SplitSeg( s_1,...,s_k )
            repair( )
4.  output DT(V)
```

Figures 6 and 7 show the execution of the algorithm on a simple polygonal 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 causes s to be split, during the first call to the `repair( )` subroutine. The updated Delaunay triangulation is shown in (c). Next, we enter the **repeat** loop to split skinny triangles. At each iteration, we can choose any triangle with a small angle to be split; our choice is the triangle with the smallest angle. In Figures 6 and 7, the triangle about to be split is shown shaded, with a cross indicating its circumcenter. In (d), we have attempted to add a vertex at the circumcenter p, but it is discovered to encroach upon two segments: segment s_4 is not a Delaunay edge, and the obtuse angle at p opposite s_3 indicates that p is within s_3's diametral circle. In (e), the encroached upon segments have been split and the triangle with the new minimum angle is shown. The circumcenter of this triangle does not encroach upon any segments, so we retain it, yielding the Delaunay triangulation shown in (f). Allowing the execution to continue until all angles are at least 25° yields (g), and optionally we can remove triangles outside the polygon, as shown in (h). (The observant reader might have noticed a slight enhancement to the algorithm in the figure: if a segment s is encroached upon by a vertex on another segment, s doesn't have to be split as long as it appears in the triangulation, and no skinny triangles are created. For instance, the vertex added in (c) encroaches upon two segments that aren't ever split.)

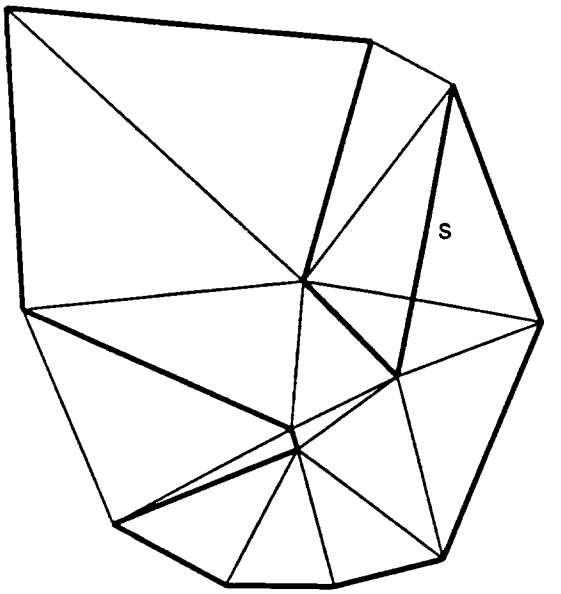
In the next section, we show that the algorithm halts for any $\alpha < 20°$. (In practice, larger values can be chosen, up to $\alpha \approx 30°$.) Upon termination, all triangles will have aspect ratios $\leq |\frac{2}{\sin\alpha}|$, since all angles smaller than α will have been removed. Furthermore, it will be a conforming triangulation, 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 Section 7.

4 Output Size

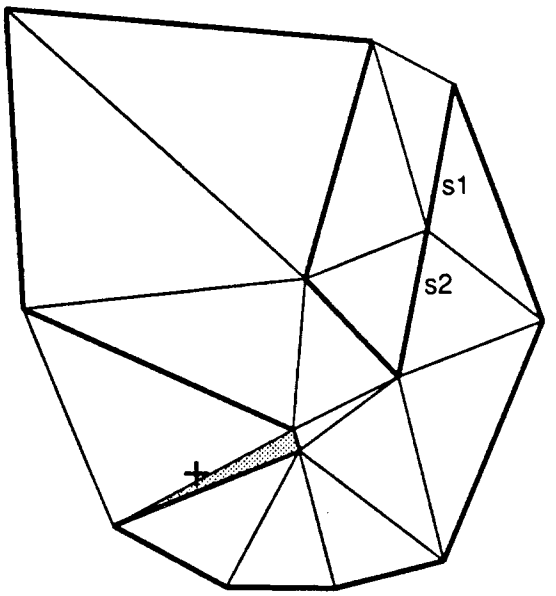
In this section we give an upper bound on the number of triangles in the output. The bound depends upon the *local feature size* of the input. At every point in the mesh, the vertex



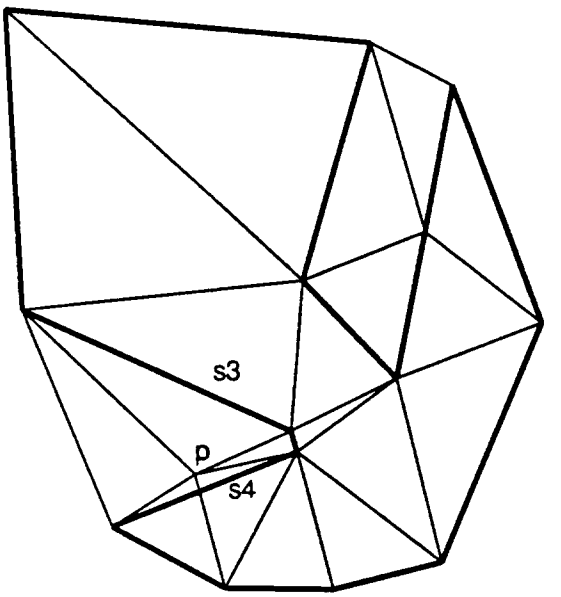
(a) Input polygon
(bounding box not used
in this example)



(b) Delaunay triangulation of input vertices.
Note that segment s is not a Delaunay edge.

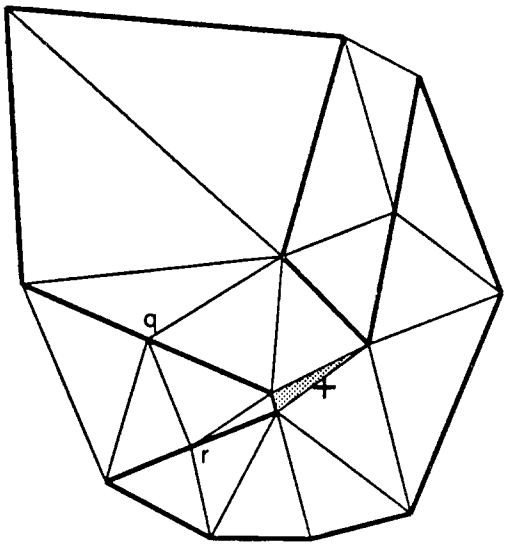


(c) Segment s "split" at midpoint, into s1 and s2.
Shaded triangle has smallest angle,
5.9 degrees. Cross indicates its circumcenter.

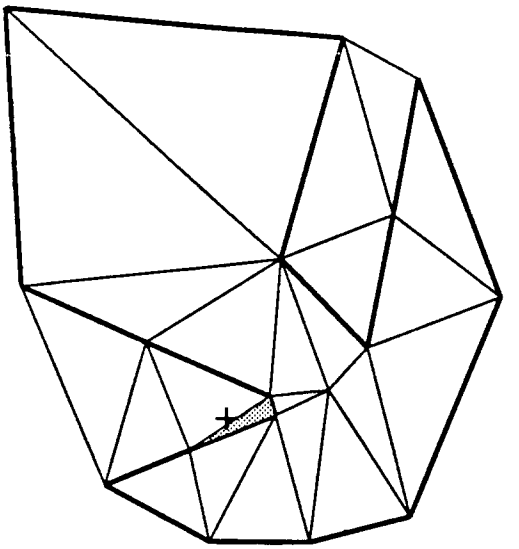


(d) Vertex p was added, but will be removed
since it encroaches upon segments s3 and s4

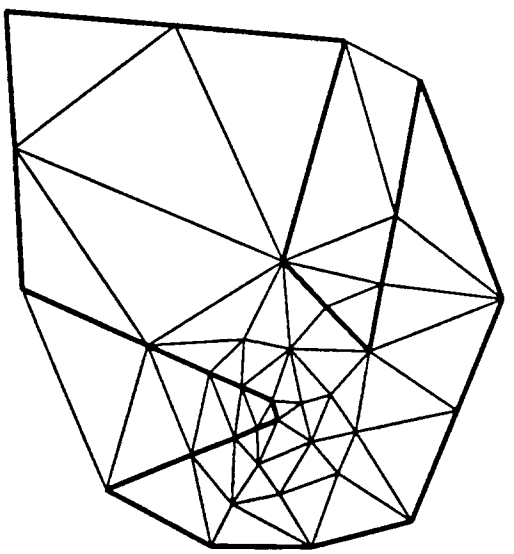
Figure 6: Illustration of the algorithm on a simple example. For clarity, bounding box not used.



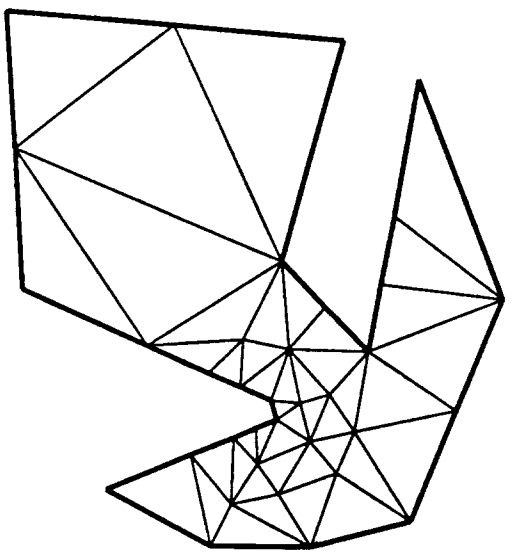
(e) 2 segments split at q and r, minimum angle now 9.8 degrees.



(f) New minimum angle 11.6 degrees.



(g) Total of 22 Steiner points added, minimum angle 25 degrees.



(h) Optionally, external triangles can be removed.

Figure 7: Continuation of example. In this case, minimum angle $\alpha = 25°$.

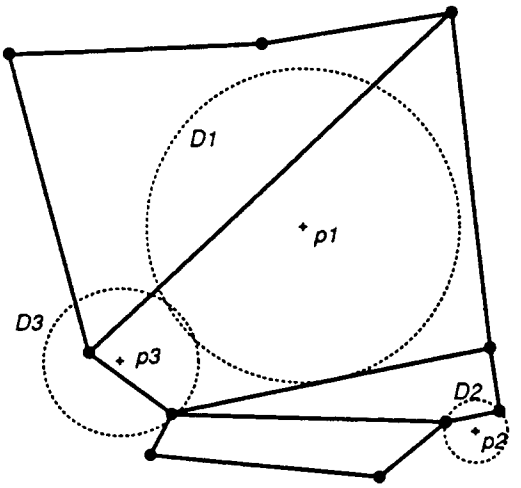


Figure 8: Local feature size at several points. Radius of disk D_i is $lfs(p_i)$.

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

Figure 8 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 1 *Given any PSLG X, and any two points p and q in the plane,*

$$|lfs(q) - lfs(p)| \leq \mathrm{dist}(p, q),$$

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

Proof: See Figure 9. Assume without loss of generality that $lfs(p) \leq lfs(q)$. The disk D of radius $r = lfs(p)$ centered at p intersects 2 non-incident portions of X. The disk D' of radius $r' = r + \mathrm{dist}(p, q)$ centered at q contains D and hence intersects the same portions of X. So $lfs(q) \leq r'$. Putting this together, we have

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

■

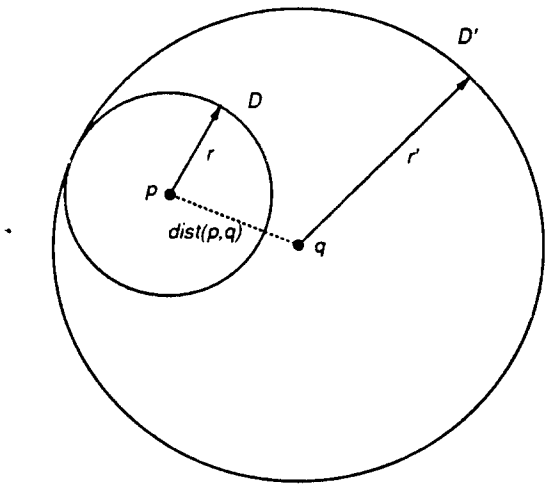


Figure 9: Lemma 1: local feature size is not too "steep".

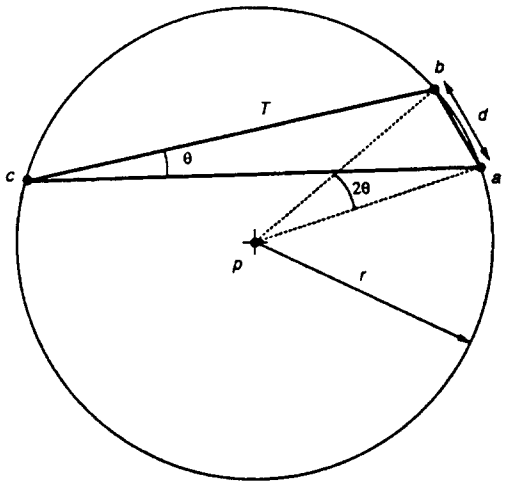


Figure 10: Lemma 2 Case 1: p added as circumcenter of triangle T with small angle $\theta < \alpha$.

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.

First, we observe that for an input vertex p, the distance to the nearest vertex is at least $lfs(p)$, by definition.

Lemma 2 *At the time vertex p is added to the triangulation, the distance to its nearest neighbor vertex is at least*

- $\frac{lfs(p)}{C_T}$ *if p was added as the circumcenter of a skinny triangle.*
- $\frac{lfs(p)}{C_S}$ *if p was added as the midpoint of an input segment.*

C_T and C_S are fixed constants we will specify below.

Proof: Assume the lemma is true for all previously added 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 10), 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. 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 radius $r' \leq d$ (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 1, $\angle apb = 2\theta$, so simple geometry gives $\sin\theta = \frac{d}{2r}$. Putting this all together, Lemma 1 gives

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

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

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

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 the newly added vertex p is added to split a segment s. Segment s is split because some vertex or circumcenter a is inside the diametral circle of s. (See Figure 11.) We have two cases for a:

Case 2(a): a lies on some segment t, which can't be incident to s, since we're assuming that all angles in the input PSLG are $\geq 90°$. (Any segment incident to s makes a larger angle, and hence would be completely outside the diametral circle.) So by definition, $lfs(p) \leq r$. Above we've 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 1 gives

$$lfs(p) \leq lfs(a) + r$$
$$\leq r'C_T + r$$
$$\leq \sqrt{2}\,rC_T + r$$

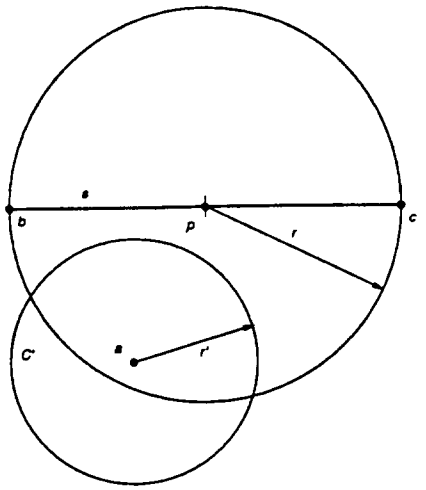


Figure 11: Lemma 2 Case 2: p added to split segment s which is encroached upon by a.

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 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 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 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 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 $\text{area}(D_p) = \pi {r_p}^2$, and plugging in for r_p, we get

$$\int_{D_p} \frac{1}{lfs^2(x)} dx \geq \left(\pi \frac{lfs^2(p)}{4(C_S+1)^2}\right)\left(\frac{4(C_S+1)^2}{(2(C_S+1)+1)^2 lfs^2(p)}\right)$$

$$\geq \frac{\pi}{(2C_S+3)^2}$$

Substituting back in for the entire integral,

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

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

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

The choice of $\alpha = 10°$ mentioned above yields $C_1 = 216$. This quite large value seems to result from the proof techniques rather than the inherent behavior of the algorithm. This issue will be discussed below.

5 Size-Optimality

In this section we state and prove some properties that any bounded aspect ratio triangulation must have, and then use these properties to show that even the optimal triangulation isn't too much better than the output of the Delaunay refinement algorithm. The following properties of bounded aspect ratio triangulations are seemingly obvious, but a number of technical details are required to state and prove them precisely:

- Small input features will be surrounded by proportionally small triangles.
- Nearby triangles have similar sizes.
- The size variation between distant triangles depends on their distance.

The basic idea would be to show that triangle sizes in an "optimal" mesh satisfy a Lipschitz smoothness condition, and hence they are proportional to our local feature size measure, yielding optimality within a constant factor for our meshes. The difficulty in doing this directly is that triangle size is a step function, with potentially large discontinuities between triangles, especially near mesh vertices of high degree. As a result, we must define precisely what we mean by "triangle size", and show that though it is a step function, it is reasonably well-behaved. With a series of lemmas, we bound the "maximum triangle size" at an arbitrary point, and show that triangle sizes within a Delaunay refinement mesh are within a constant factor of the largest possible.

The analysis in this section is similar to that given by Mitchell and Vavasis in [18] for their 3D algorithm. A basic notion in their proof is that of a "characteristic length function", which defines the "triangle size" at every point within the triangulation:

Definition 3 *If a point p is contained in a triangulation $\mathcal{T}$ of input PSLG X, then we say the* **edge length** *at p, $el_{\mathcal{T},X}(p)$, or simply $el(p)$, is the length of the longest edge among all triangles of $\mathcal{T}$ containing p.*

We shall now prove some properties about this function within bounded aspect ratio triangulations. For the remainder of this section, we assume all triangles have a minimum angle bound of α, which guarantees all aspect ratios are at most $A = \frac{2}{\sin\alpha}$. By considering the shared edge between two triangles, we have the following:

Fact 2 If p and q lie in the interiors of distinct triangles T_p and T_q, which share an edge, then $\frac{el(q)}{el(p)} \leq A$.

Repeated use of this fact gives the following lemma about points in arbitrary triangles:

Lemma 3 *If p and q lie in the interiors of triangles T_p and T_q, respectively, then*

$$el(q) - el(p) \leq C_2 \cdot el(p) + C_3 \cdot dist(p, q)$$

where C_2 and C_3 are constants to be specified.

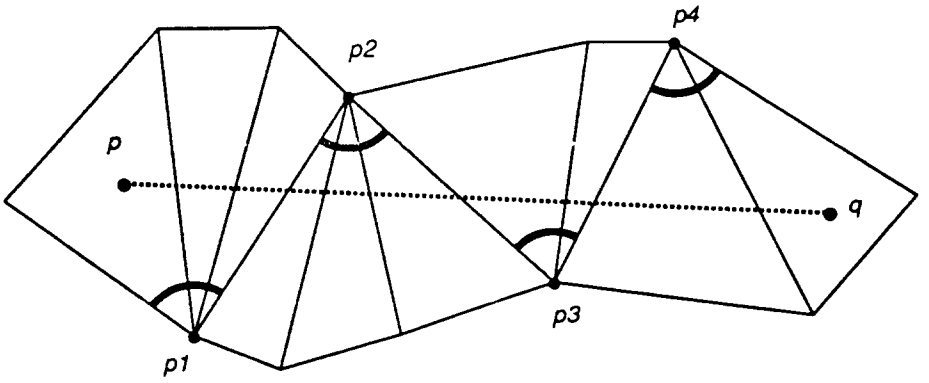


Figure 12: Triangles crossed by the segment from p to q are divided into "fans".

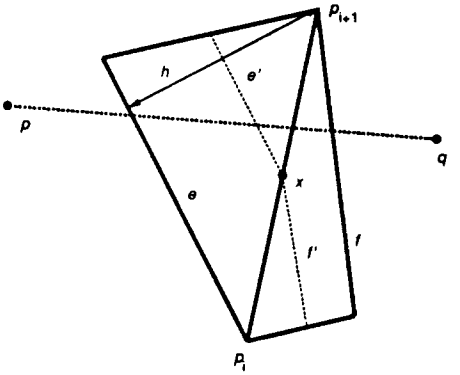


Figure 13: Since line segment pq crosses edges e and f, it must cross e' or f'.

Proof: Consider the sequence of triangles crossed by the line segment from p to q, as shown in Figure 12. (Here we are assuming that the triangulation fills a convex region, so that the segment stays within the triangulation. More on this in the next section.) Any vertex on the segment is treated as though it were to the "right" of the directed segment from p to q. Label any vertices shared by more than two consecutive triangles $p_1, p_2, \cdots$. The "zigzag" edges connecting successive p_i's divide the triangles into "fans" around each p_i, indicated by the bold arcs in Figure 12. Since at most $\lfloor \frac{2\pi}{\alpha} \rfloor$ triangles fit around a vertex, each fan contains at most $K = \lfloor \frac{\pi}{\alpha} \rfloor$ triangles, except the first and last, which may contain $K+1$. We consider two different cases, depending upon the number k of triangles between p and q, including T_p and T_q.

Case 1: $k \leq K+3$: Using k applications of Fact 2, we see that the lemma holds as long as $C_2 \geq A^{K+3}$ and $C_3 \geq 0$.

Case 2: $k > K+3$: Since zigzag edges are separated by at most k triangles, there exists some zigzag edge $p_i p_{i+1}$ that is flanked by two triangles, neither of which contains p or q. Consider the closest such edge to q. In Figure 12, this is the edge $p_3 p_4$. Figure 13 shows edge $p_i p_{i+1}$ and its two flanking triangles. We now show that the length of the segment pq is at least half of an altitude of one of these triangles. Let e and f be the two outer edges crossed by pq, as shown. Let x be the midpoint of $p_i p_{i+1}$, and construct e' and f' through

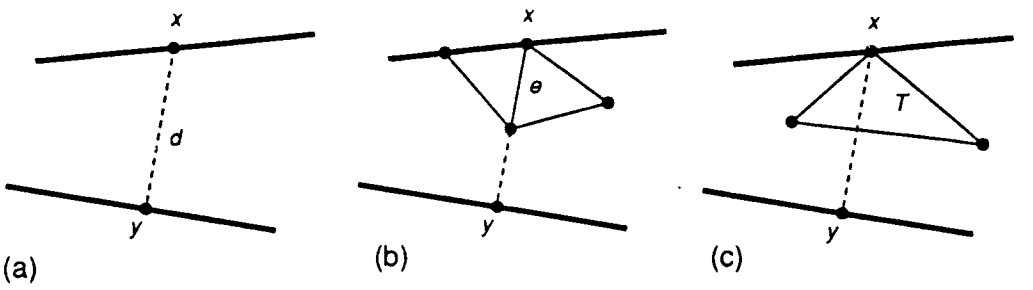


Figure 14: Triangle size along xy if x is a mesh vertex.

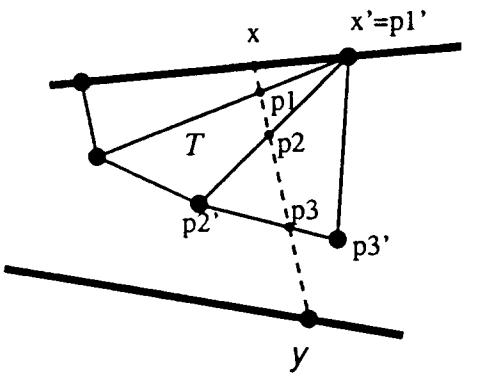


Figure 15: Triangle size along xy if x is not a mesh vertex.

x and parallel to e and f, as shown. Since pq crosses e and f, it must cross either e' or f'. If it crosses e', then it is longer than half the altitude h from edge e to vertex p_{i+1}, i.e. $\text{dist}(p,q) \geq \frac{h}{2}$. By the definition of aspect ratio, the edge p_ip_{i+1} has length at most $A \cdot h$, and with $K+2$ applications of Fact 2, we see that $el(q) \leq A^{K+3}h \leq 2A^{K+3}\text{dist}(p,q)$. The lemma holds for $C_3 = 2A^{K+3}$, since $el(p) > 0$, $C_2 > 0$. The case where f' is crossed by pq is handled similarly, within the same bound. The choice of $C_2 \geq A^{K+3}$, $C_3 \geq 2A^{K+3}$ satisfies all the conditions. ∎

Lemma 3 gives a bound on how fast edge lengths can change in a bounded aspect ratio triangulation. The following lemma shows that there must be small triangles near small input features.

Lemma 4 *Let x and y be points (not necessarily endpoints) of non-incident input segments. Let d be the distance between x and y. Then, in any triangulation with aspect ratios bounded by A, there is a point p on the line segment connecting x and y with $el(p) \leq 2d \cdot A$.*

Proof: See Figure 14(a).

Case 1: The easy case is when x or y is a vertex of the triangulation. Without loss of generality, suppose x is a vertex. If there is a triangulation edge at x along the line segment xy (see Figure 14(b)), then that edge has length $\leq d$. For any point p in the interior of the

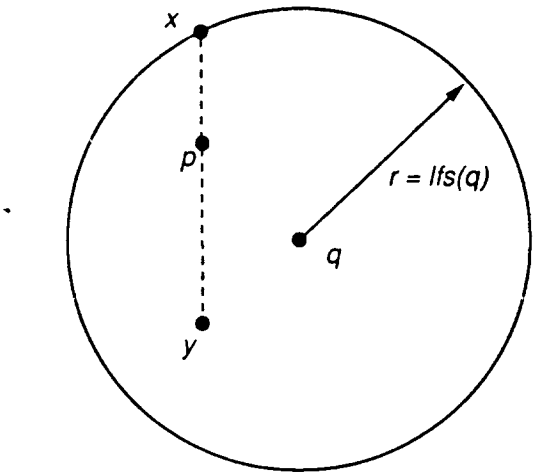


Figure 16: Points x and y on input segments determine edge length $el(p)$ at some point p along xy, and local feature size $lfs(q)$.

edge, $el(p) \leq d \cdot A$. If the line segment xy is in the interior of some triangle T near x (see Figure 14(c)), then the minimum altitude of T is no longer than T's intersection with xy, so for any point p on xy interior to T, $el(p) \leq d \cdot A$.

Case 2: This case, where x is in the interior of an edge of the triangulation, is illustrated in Figure 15. Let x' be the endpoint nearest x, and consider all triangles incident to x' that intersect the line segment xy. Let $p_1, p_2, \ldots$ be the intersections of xy with the edges of these triangles, as shown. Finally, on the edge containing p_i, label the endpoint closest to p_i as p_i'. Let j be the smallest index such that $p_j' \neq x'$. Such a j exists, because eventually xy will reach an edge not incident to x', for instance the edge containing y. By an argument similar to that used in Case 2 of the previous lemma, we see that the minimum altitude of T, the triangle containing p_{j-1} and p_j is at most $2d$. Then for any point p in the interior of the line segment $p_{j-1}p_j$, $el(p) \leq 2d \cdot A$. ∎

Next, we use the preceding lemmas to relate the edge length function $el(\ )$ to our local feature size measure $lfs(\ )$. Recall that we assume $\mathcal{T}$ is any triangulation in which all angles are at least α.

Lemma 5 *At any point q in the interior of a triangle of $\mathcal{T}$, $el(q) \leq C_4 lfs(q)$, where C_4 is a constant to be specified.*

Proof: By definition, $lfs(q)$ is the radius r, determined by two points x and y on non-incident segments of the input (see Figure 16). From Lemma 4, there must be some point p along x, y with $el(p) \leq 2 \cdot \text{dist}(x, y) \cdot A$. Since we always have $\text{dist}(x, y) \leq 2r$, $el(p) \leq 4r \cdot A$. Using $\text{dist}(p, q) \leq r$ and Lemma 3,

$$el(q) \leq el(p) + C_2 \cdot el(p) + C_3 \cdot \text{dist}(p, q)$$

$$\leq (C_2 + 1) \cdot el(p) + C_3 \cdot r$$

$$\leq (C_2+1)\cdot 4r\cdot A + C_3\cdot r$$
$$\leq [(C_2+1)\cdot 4A + C_3]\cdot r$$
$$\leq [(C_2+1)\cdot 4A + C_3]\cdot lfs(q)$$

Choosing $C_4 \geq (C_2+1)\cdot 4A + C_3$ concludes the proof. ∎

We can now state and prove the major result of this section: that the mesh output by the Delaunay refinement algorithm is size-optimal to within a constant factor. First we recall the situation: the input is a planar straightline graph X with all angles at least 90 degrees, $\alpha \leq 20°$ is the minimum angle bound for the output, which guarantees all triangles have aspect ratio $\leq \frac{2}{\sin\alpha}$. The algorithm triangulates the region inside $B(X)$, a larger bounding box of X, and the optimality is with respect to any triangulation of $B(X)$ with minimum angle bound α. The constant factor C_α depends on the choice of α, but not on X, i.e. the algorithm is optimal on every input, not just in the worst case. (The 90° input restriction, and the requirement that the mesh triangulates $B(X)$, will be eliminated in the next section.)

Theorem 3 *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 the Delaunay refinement 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.*

Proof: Theorem 2 bounds the number of vertices in the Delaunay refinement triangulation. In any triangulation, the number of triangles is at most twice the number of vertices (true by Euler's relation, see [20], p. 19). Thus the triangulation has

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

triangles. By Lemma 5 this is

$$\leq C_1 \cdot {C_4}^2 \int_B \frac{1}{el^2(x)}dx$$

where the edge-length function $el(\ )$ is with respect to $\mathcal{T}$. (Strictly speaking, Lemma 5 does not apply to edges of the triangulation, but since they have measure 0, they don't contribute to the integral.) We can instead sum the integrals over each triangle $T \in \mathcal{T}$:

$$= C_1 \cdot {C_4}^2 \sum_{T\in\mathcal{T}} \int_T \frac{1}{el^2(x)}dx$$

In each triangle T, $el(\ )$ is constant, just the length of the longest edge. The area of T is at most $\frac{\sqrt{3}}{4} el^2(\ )$, which occurs if T is equilateral. So for T we have

$$\int_T \frac{1}{el^2(x)}dx \leq \frac{\frac{\sqrt{3}}{4}el^2(x)}{el^2(x)} = \frac{\sqrt{3}}{4}$$

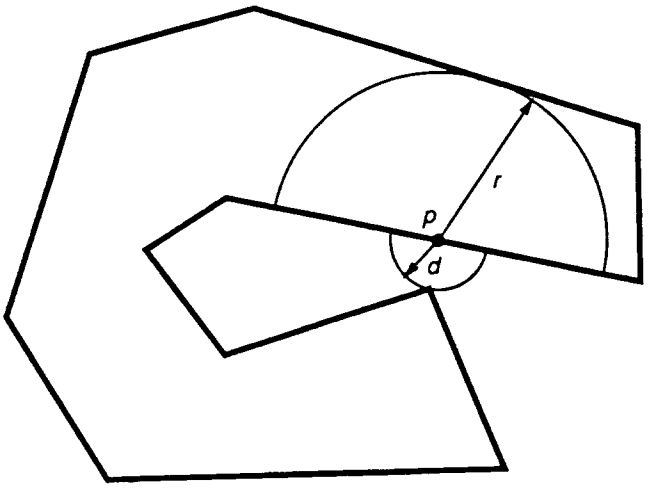


Figure 17: Do the polygon's two "arms" determine a small feature at p?

Substituting back in,

$$M \leq C_1 \cdot {C_4}^2 \frac{\sqrt{3}}{4} \sum_{T \in \mathcal{T}} 1$$

The summation counts the number of triangles in $\mathcal{T}$, so the theorem holds for $C_\alpha = C_1 \cdot {C_4}^2 \frac{\sqrt{3}}{4}$. ■

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 Section 7.

6 Corner-Lopping and Riemann Sheets

Two issues must be resolved so that our algorithm produces optimal bounded aspect ratio triangulations for general 2-dimensional inputs. First, we must deal with small input angles reasonably (recall that we unreasonably assumed all angles were at least 90°!). The second issue is subtler, and relates to our definition of local feature size in non-convex polygons: in Figure 17, 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. Fortunately, previous researchers have dealt with both of these concerns, and we can adapt their solutions to our algorithm. These modifications may increase the size of the mesh, but by at most a constant factor.

We handle small angles by "lopping off" the sharp corners during a preprocessing step. Any input vertex p with a small angle is "shielded" by committing in advance to a specific triangulation around p. This was previously done in [6] with a circle around p, and in the 3D algorithm of [18], a cube was used. At any vertex p with an angle smaller than 90°, we will intersect a circle with the input edges, as shown in Figure 18. The radius of the circle will be $\frac{lfs(p)}{3}$, so that circles around different vertices do not intersect or get too close. Angles at p that are $> \alpha$ are divided so as to be between α and 2α, introducing *shield edges* around the

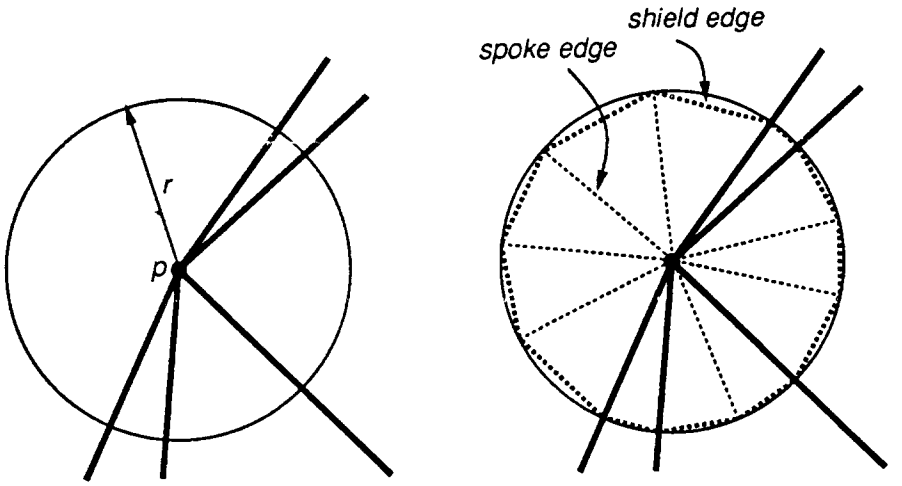


Figure 18: "Lopping off" sharp corners with a shielding circle (input segments in bold).

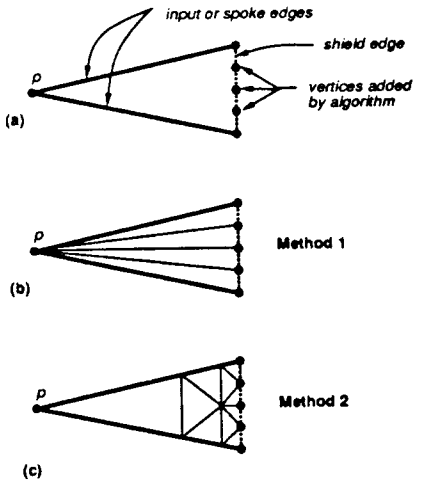


Figure 19: Fixing shield edges that get split.

circle, and *spoke edges* from p to the circle. These edges, and new vertices, are henceforth considered as part of a modified input PSLG X', and will appear in the output mesh. This reduces the local feature size in X' compared with X, but only by a constant factor that depends on α.

Outside the shielding circles, all angles will be greater than 90°. Within the shielding circles, our hope is to use the triangles shown in Figure 18 as the output triangles around p. If we disallow such triangles to be split, then no vertices will be added within the shielding circle, but still the shield edges may get split, as shown in Figure 19(a). When the algorithm terminates, each shield edge will be split into at most a constant number of pieces, since the local feature size along the edge is proportional to the edge length.

We now have two ways of dealing with split shield edges. Following [18], we place edges between the split vertices and p, as shown in Figure 19(b). (In fact, these will be present as the Delaunay edges.) As shown for the 3D case in [18], this means the minimum angle in

the output mesh will be within a constant factor of optimal (depending on how many times the small shield edges get split), while retaining the constant factor optimality for the mesh size.

A more complicated construction can avoid splitting the smallest input angle at all. As shown in [6], a construction like that in Figure 19(c) will work for polygon inputs. This doesn't work directly for PSLG inputs, since the vertices added to the spoke edges incident to p may conflict with those used for another shield edge. This can be handled by applying the construction to the smallest angle incident to p. The two spokes bounding this angle become subdivided with new vertices, and we subdivide all other spokes identically.

Some practical issues concerning small input angles are discussed in Section 7.

The second issue is illustrated in Figure 17. We assumed our input was a planar straight-line graph, and produced a triangulation that extended out to a larger surrounding box. If only the interior of a polygon is to be triangulated, then we would not consider the clearance between the two "arms" of the polygon in Figure 17 as a small feature. In particular, the local feature size at p should be r, rather than d, as our definition states. As in [18], 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). The algorithm is modified to work using a *constrained Delaunay triangulation*, say by using the *Riemann sheet* technique of Seidel [21]. This corresponds to Mitchell and Vavasis' use of Riemann volumes with octrees [18].

7 Implementation and Discussion

The basic algorithm of Section 3 leaves unspecified some issues concerning its implementation. We now discuss these issues in general, and describe our own implementation, which does not include the corner-lopping or Riemann sheet modifications of the previous section. Figure 20 shows the output of our implementation on two examples.

An incremental Delaunay triangulation algorithm is ideal as a basis for the Delaunay refinement algorithm. The simplest such algorithm works by swapping diagonals within a quadrilateral formed by two triangles. Guibas and Stolfi [13] give pseudo-code for this, as well as a useful data structure for manipulating triangulations.

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 5), or

2. It is present, but opposite an obtuse angle in a Delaunay triangle (e.g. s_2 in Figure 5).

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 angle improves as the algorithm continues to run (see Figure 22).

The algorithm specifies a square bounding box 3 times as large as the input. In fact, any constant multiple will work. For clarity in our examples, we have used a smaller bounding box. The bounding box has both a theoretical and a practical purpose. Though a polygon

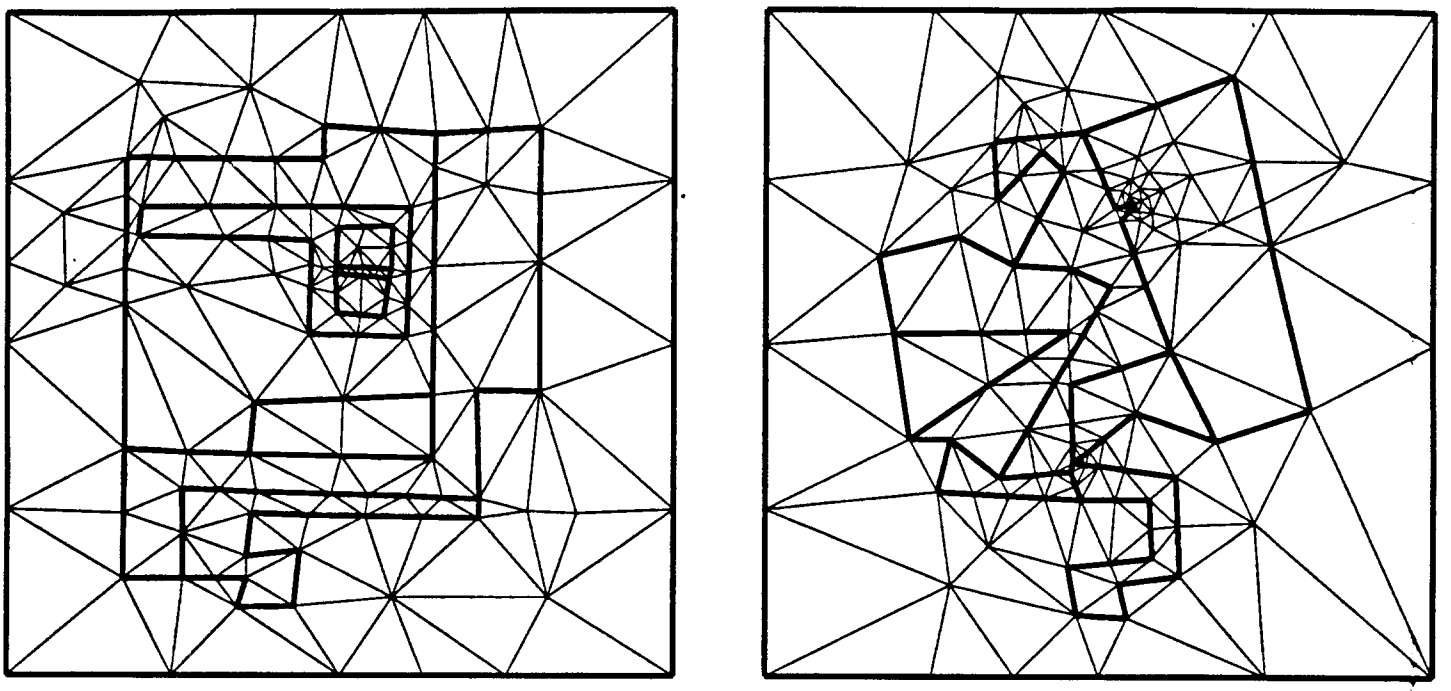


Figure 20: Output on two sample PSLGs, minimum angle $\approx 20°$.

clearly has an interior, a PSLG input may have dangling edges, and it is not always clear exactly what region is to be triangulated. The convex hull of the PSLG is a logical candidate, but then an input vertex just inside the hull could generate a "small feature" that isn't really present in the input. The bounding box gives an unambiguous region to be triangulated, without reducing the local feature size by more than a constant factor. An axis-aligned bounding square also improves the algorithm's stability, since splitting an edge of the box gives a midpoint which is precisely collinear with the endpoints. Otherwise, if roundoff were to occur, then the midpoint could fall inside the edge, causing a very skinny Delaunay triangle to form between the midpoint and the endpoints.

Though the corner-lopping was required for the optimality analysis in the presence of input angles below 90°, it is not generally necessary in practice. By redefining a *skinny* triangle to exclude those with small input angles, the algorithm works well in most cases. Figure 21 shows the output on an example with several input angles near 15°. The desired minimum output angle is 20°, which is achieved everywhere except at the small input angles. Input angles below 10° may present a problem, causing a large number of vertices to be added near that angle. This occurs because the incident segments are split many times. A reliable practical implementation would need to incorporate some technique, such as corner-lopping, to handle all small input angles. Pre-processing steps such as corner-lopping tend to greatly increase the size of the output mesh, so we have tried to avoid them. The following technique for handling small angles is currently under investigation, and seems promising: First, choose a parameter d near the smallest feature size. Then, when splitting segments incident to a vertex p with a small input angle, don't choose the segment's midpoint, but choose a nearby point on the segment at a distance $2^k \cdot d$ from p, for some integer k. The

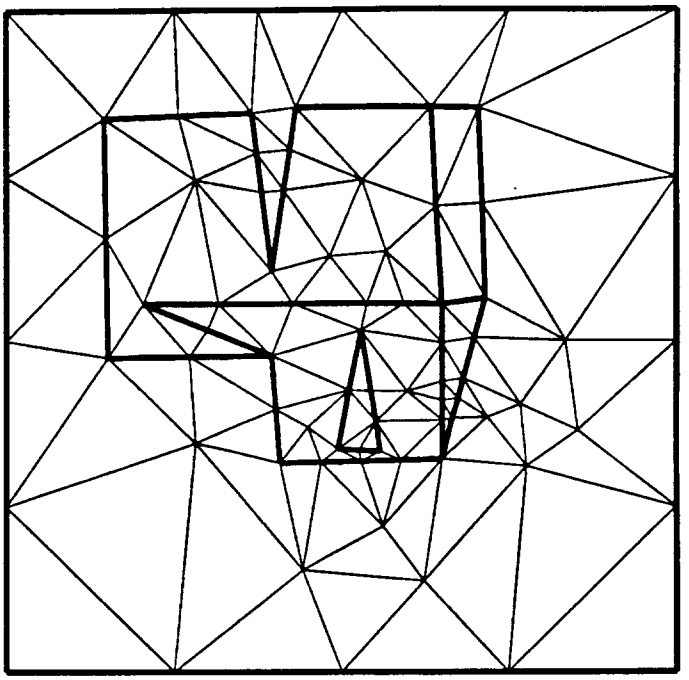


Figure 21: An input that contains angles smaller than α.

goal is to create circular "rings" of vertices around p, similar to those in the shielding circles used in corner-lopping.

We now turn to the question of how size-optimal the algorithm is. The examples of Figure 20 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 that and the value of C_α of Section 5. 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, C_α includes a factor of A^{2K+6}, with $A \approx 6$, $K \approx 4$, for $\alpha = 20°$. We suspect that with a careful analysis of triangles packed around a vertex, this factor 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 Section 5. 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 20, 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 [6], 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.

Additional evidence concerning the behavior of the Delaunay refinement algorithm comes from Figure 22, 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,

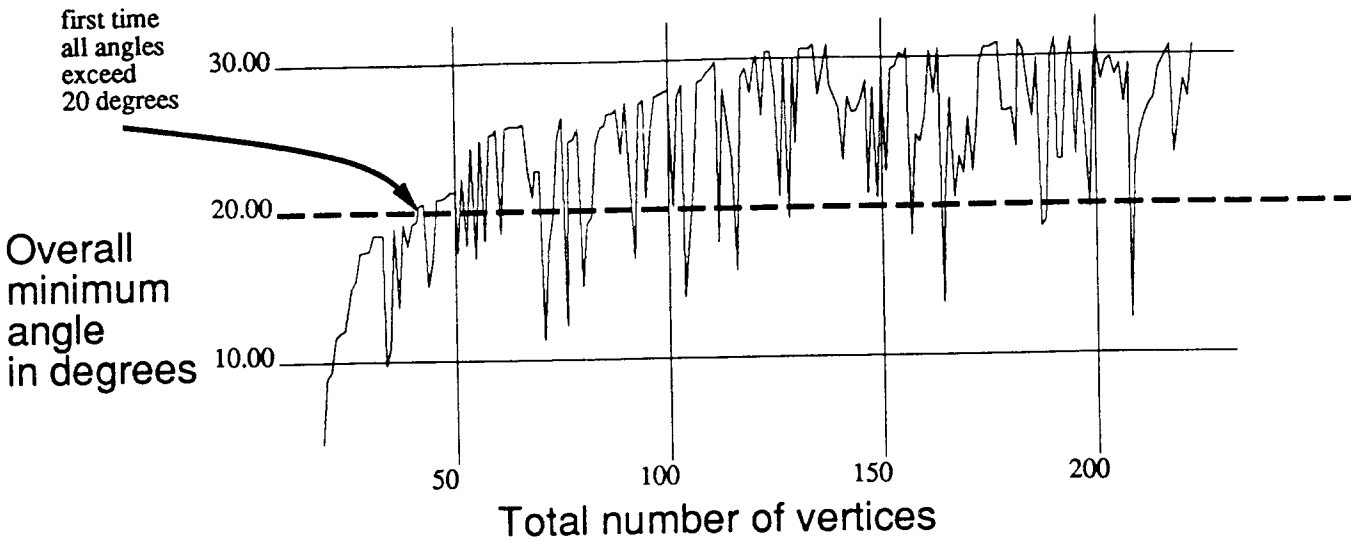


Figure 22: Progress of minimum angle, during a typical run.

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 running time is dominated by the time needed to maintain the incremental Delaunay triangulation. The determination of encroached upon segments and skinny triangles can be absorbed into this using the techniques described above. The worst-case running time for incremental Delaunay triangulation is $O(M^2)$, where M is the output size. The algorithm of [13] is one that achieves this bound. In practice, such algorithms usually run much faster. 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.

8 Generalization to Three Dimensions

Our greatest hope would be to generalize our 2D algorithm to perform 3D tetrahedral meshing of polyhedra and polyhedral complexes. In this regard, we are somewhat pessimistic: the Delaunay refinement algorithm extends fairly readily to three dimensions, but its bounded aspect ratio guarantee does not. It seems that significant new ideas are necessary in order to get bounded aspect ratio tetrahedra using a Delaunay triangulation based approach.

The discrepancy between 2D and 3D boils down to the following: in 2D, a set of "evenly spaced" points will have a Delaunay triangulation with no skinny triangles, but this does not hold true in 3D. Figure 23 shows why skinny triangles, or "slivers", are ruled out in 2D: a sliver either has widely varying edge lengths, or a circumcircle much larger than any of its edges. Such a circumcircle forms a large "gap" not containing any points. In 3D, however,

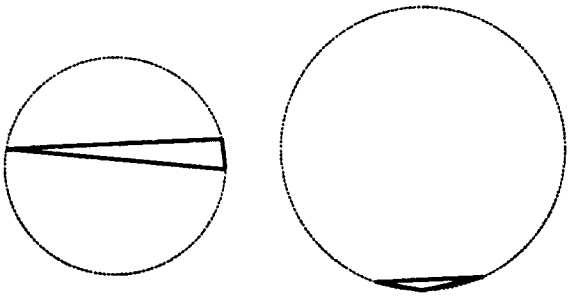


Figure 23: A "sliver" triangle in 2D must have a circumcircle much larger than its shortest edge.

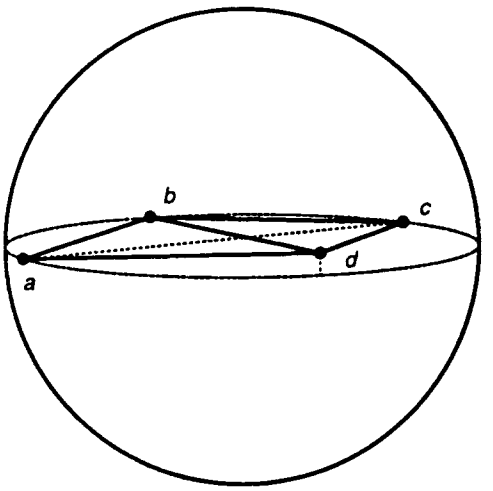


Figure 24: "Sliver" tetrahedron: 4 points spaced around equator of a sphere, with d raised slightly.

tetrahedra can have roughly equal-length edges, a reasonable sized circumsphere, and yet be arbitrarily skinny, as shown in Figure 24: four vertices spaced equally around the equator of a sphere, with d raised slightly to a latitude of ϵ above the equator.

These flat sliver tetrahedra appear quite often in 3-dimensional Delaunay triangulations. The difficulties of avoiding them or removing them have been discussed in [11], [16], [1], amongst others.

9 Conclusion

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

There are many opportunities for further work. Can small input angles be handled without the corner-lopping preprocessing step? How useful would it be to incorporate the constrained Delaunay triangulation? The algorithm is well-suited to 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 to indicate 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?

10 Acknowledgements

Particular thanks go to Raimund Seidel for many encouraging discussions. Thanks also to Balas Natarajan and Marshall Bern for helpful suggestions, and to Steve Fortune, whose Voronoi diagram implementation (available via *netlib*) proved very useful in developing our algorithm.

References

[1] C. Bajaj, T. Dey, and K. Sugihara. On good triangulations in three dimensions. In *Proceedings of the ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, 1991.

[2] T.J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Eng. with Computers*, 5:161–175, 1989.

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

[4] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T.S. Tan. Edge-insertion for optimal triangulation. In *Proc. Latin American Theoretical Informatics (to appear)*, 1992.

[5] 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 (?).

[6] 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*.

[7] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 5:485–524, 1991.

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

[9] H. Edelsbrunner and T.S. Tan. A upper bound for conforming Delaunay triangulations. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 53–62. ACM, 1992.

[10] H. Edelsbrunner, T.S. Tan, and R. Waupotitsch. A polynomial time algorithm for the minmax angle triangulation. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 44–52. ACM, 1990.

[11] D. Field. Implementing Watson's algorithm in three dimensions. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 246–259. ACM, 1986.

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

[13] 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.

[14] K. Ho-Le. Finite element mesh generation methods: a review and classification. *Computer-Aided Design*, 20:27–38, 1988.

[15] 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.

[16] S. Meshkat, J. Ruppert, and H. Li. Three-dimensional unstructured grid generation based on Delaunay tetrahedrization. In *Proceedings of the 3rd International Conference on Numerical Grid Generation*, pages 841–851, June 1991.

[17] G.L. Miller, S.H. Teng, and S.A. Vavasis. A unified geometric approach to graph separators. In *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 538–547, 1991.

[18] 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.

[19] L. Nackman and V. Srinivasan. Point placement for Delaunay triangulation. In *Third Canadian Conference on Computational Geometry*, pages 37–40, Vancouver, 1991.

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

[21] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, Inst. for Information Processing, Graz, Austria, 1988.