# TRIANGULAR MESH GENERATION IN $\mathbb{R}^2$

HANG SI

## Contents



## Introduction

In this chapter, we consider the triangular mesh generation problem. The input is no longer only a set of vertices, but typically an 2d polygonal (possible non-convex) domain. Moreover, it may contains holes and subdomains (separated by internal boundaries).

We first consider a constrained triangulation problem in which a set of points plus a set of line segments are given, and how to construct a triangulation of this point set which contains the set of line segments. Although there are efficient algorithms for constructing constrained triangulations, but incremental algorithm is easy to implement and is thus attractive for practical use. We introduce a practical incremental algorithm which uses a flip-based edge recovery algorithm to construct it.

We then introduces a special type of constrained triangulation that is closet, in some sense, to the (unconstrained) Delaunay triangulation. Such triangulation can be constructed by the same incremental algorithm combined with the Lawson's flip algorithm.

For many applications, the mesh quality is of the utmost importance. We will consider how to generate "good quality" mesh with respect to geometric and numerical criteria.

For this purpose, the mesh vertices are no longer part of the input but need to be placed by the algorithm itself. We introduce the classical Delaunay refinement method that adds vertices at the circumcenters of Delaunay triangles.

## 1. Constrained triangulations

1.1. **Planar straight line graphs.** Consider now the input is a finite set of points $S \subset \mathbb{R}^2$, together with a finite set of line segments, $L$, each connecting two points in $S$. We require that any two line segment of $L$ are disjoint or meet at most in a common endpoint. $G = (S, L)$ is a *planar straight line graph* (PSLG), see Figure 1.1 Left for an example.

A *constrained triangulation* (or shortly CT) of a PSLG $(S, L)$ is a triangulation of $S$ that contains all line segments of $L$ as edges, see Figure 1.1 Right.

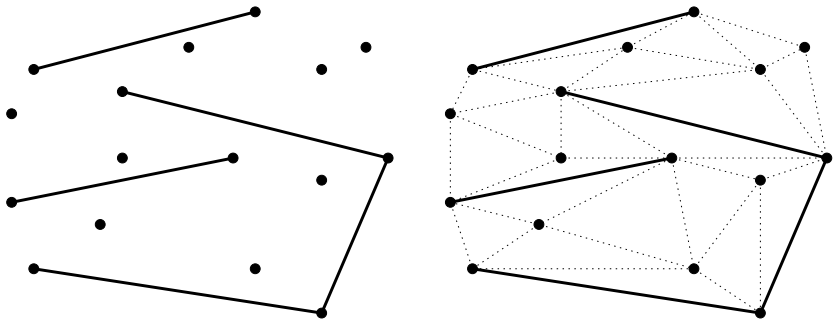

Figure 1. A set of vertices and line segments (Left) and a constrained triangulation (Right).

By this definition, a CT of a $(S, L)$ is nothing else a special triangulation of $S$ which includes $L$ in its edge set. Hence it covers the cover hull of $S$. While it is not necessarily the Delaunay triangulation of $S$.

Since the flip-graph of a 2d point set is connected, This implies that such a constrained triangulation of $S$ exists. Moreover, it needs no additional point. This is the crucial property which ensures the termination of constrained triangulation algorithms.

1.2. **Insert an edge.** We first discuss how to enforce a given edge into a constrained triangulation of $(S, L)$ with the assumption that this edge does not intersect with any line segments of $L$. We assume that both of the endpoints of the edge already exist in a triangulation $\mathcal{T}$. Otherwise, we can use the vertex insertion subroutine in Section **??** to insert them. The algorithm of insertion of an edge is given below, it consists of two subroutines, *edge location* and *edge recovery*.

We discuss these two subroutines in the following subsections, respectively.

1.2.1. *Edge location.* This is similar to point location. Given an edge $AB$ to be inserted, it first locates one of the endpoints, say $A$ (e.g., use the simple straight line walk approach), it then searches the other endpoint $B$ from the located triangle whose origin is $A$. This is a simple rotary traversal of the adjacent triangles of $A$. If $AB$ already appears in $\mathcal{T}$, then it is inserted. Otherwise, it is a *missing* edge in $\mathcal{T}$. We use an edge recovery algorithm to insert this edge.

**Algorithm**: InsertEdge($AB$, $\mathcal{T}$)
**Input**: $AB$ is a given edge, $\mathcal{T}$ is a CT of $(S, L)$;
**Output**: $\mathcal{T}$ is a CT of $(S, L)$ and $AB \in \mathcal{T}$;
1 **if** AB is not an edge of $\mathcal{T}$ **then**
2 RecoverEdge($AB$, $\mathcal{T}$);
3 **endif**

FIGURE 2. Insert an edge $AB$ into a constrained triangulation $\mathcal{T}$ of $(S, L)$. Assume that $A, B \in S$ and $AB$ does not intersect with any line segment in $L$.

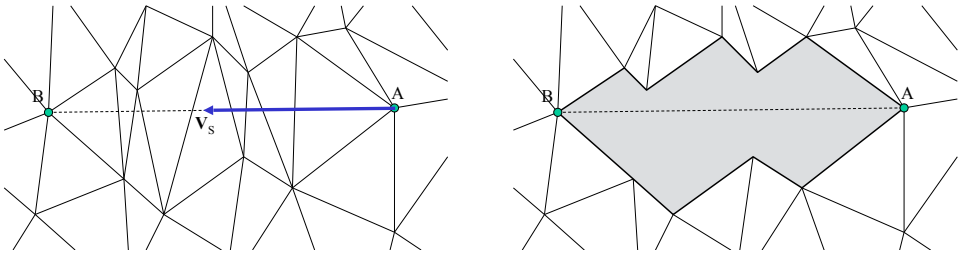


FIGURE 3. Left: search an edge $AB$ in a constrained triangulation. Right: the cavity of the missing edge $AB$. (Figures from S. Owen).

1.2.2. *Recover edge by flips.* A missing edge must intersect a set of edges and triangles of $\mathcal{T}$. The union of the set of all intersecting elements forms a *cavity* inside the triangulation. It is a simple polygon which is not necessarily convex, see Figure 3. Moreover, the interior of this polygon may contain vertex (vertices) of $S$, see an example in Figure 4.

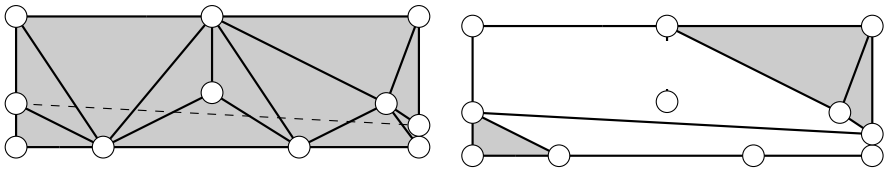

FIGURE 4. An example of a cavity which contains an interior vertex (Figures from J. Shewchuk [9]).

To recover the segment, the triangles in the cavity will be removed and replaced by a set of new triangles which do not intersect the segment. Moreover, any vertex (if there exists) inside the cavity needs to be preserved.

Let $AB$ be an edge which does not in a constrained triangulation $\mathcal{T}$. In this section we describe an algorithm to recover $AB$ in $\mathcal{T}$ by a sequence of edge flips. This approach is simple and easy to implement. There is no need to explicitly create the cavity of $AB$. Moreover, those vertices of $\mathcal{T}$ which lie inside $P$ are preserved automatically.

Call an edge in $\mathcal{T}$ a *crossing edge* if it intersects $AB$ in its interior. This algorithm maintains a list $Q$ which contains all crossing edges. Then it tries to flip them. Once there is no crossing edge, the edge $AB$ is recovered.

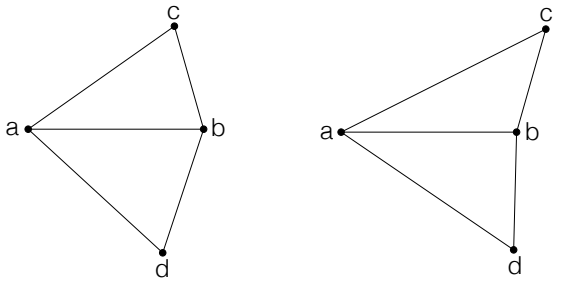


FIGURE 5. Left: edge $ab$ is flippable. Right: edge $ab$ is unflippable.

Let $ab$ be one of the crossing edges, let $abc, abd \in \mathcal{T}$ be the two triangles sharing at $ab$. This algorithm does edge flip on $ab$ based on the following two rules:

(1) We say that $ab$ is *flippable* if the vertices $a$, $b$, $c$, and $d$ are in strictly convex position. Otherwise, it is *unflippable*, see Figure 5. We only do flip if $ab$ is flippable. Otherwise, one of $cda$ and $cdb$ is inverted such that it intersects other triangles of $\mathcal{T}$ in its interior, see Figure 1.2.2 (4). In this case, we put $ab$ back into $Q$, to try it later.
(2) Another possible case is that the edge $cd$ was a previously flipped edge. To avoid an endless loop, we also do not flip the edge $ab$.

If we have successfully flipped the edge $ab$. It is possible that the new edge $cd$ also be a crossing edge, see Figure 1.2.2 (5). If it is the case, $cd$ is put into the $Q$, and it will be flipped at a later time. The algorithm is shown below.

**Algorithm**: Recover_Edge_by_Flips($\mathcal{T}$, $AB$)
**Input**: A triangulation $\mathcal{T}$, and
an edge $AB \notin \mathcal{T}$;
**Output**: A constrained triangulation $\mathcal{T} \ni AB$;

```
1   let Q be a queue of all crossing edges;
2   while Q ≠ ∅ do
3       pop a crossing edge ab from Q;
4       if ab is flippable and cd was not a flipped edge then
5           flip ab to cd;
6           if cd is a crossing edge then
7               add cd to Q;
8           endif
9       else
10          add ab to Q;
11      endif
12  endwhile
```

FIGURE 6. The algorithm to recover an edge by flips.

Figure 1.2.2 illustrates this edge recovery algorithm by different steps of a triangulation $\mathcal{T}$ modified in this way.

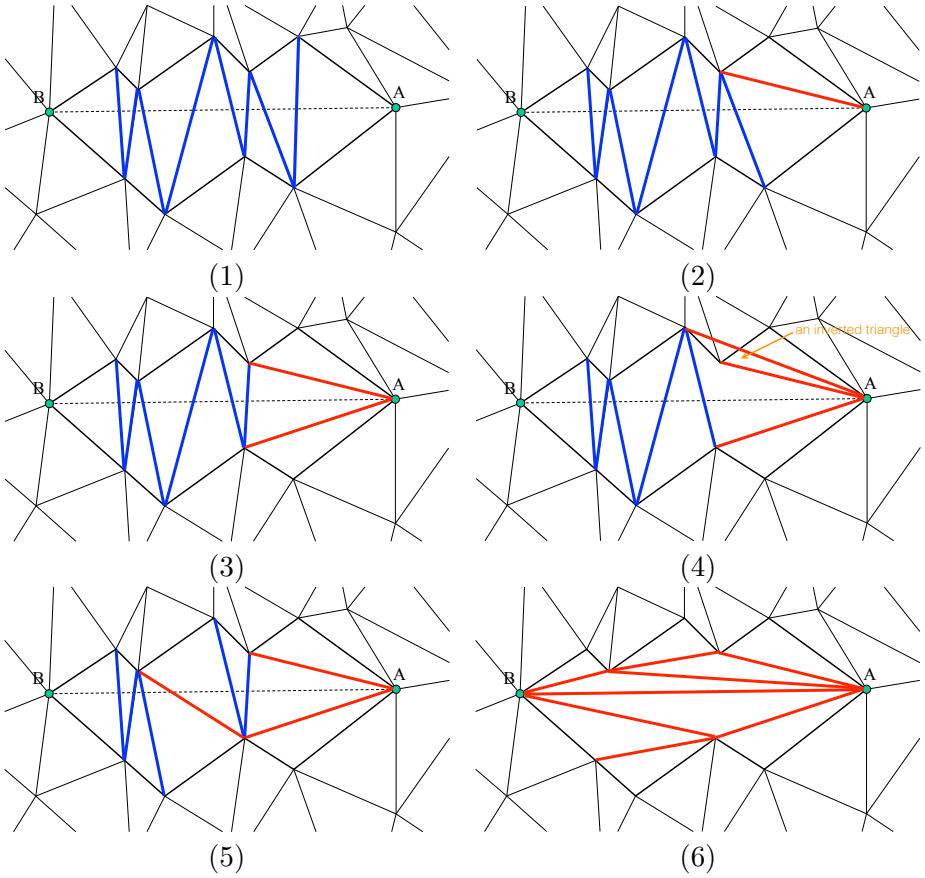


FIGURE 7. An illustration of the algorithm to recover an edge by flips. (Figures from S. Owen).

1.2.3. *Termination and runtime.* The termination of this algorithm is guaranteed by the connectedness of the flip-graph of the point set $S$. We can show that the following claim always hold:

**Lemma 1.1.** *At any step within each edge recovery procedure, there always exists an edge such that (i) it intersects the missing segment, (ii) it is flippable, and (iii) it has not being flipped before.*

*Proof.* (i) must be true, otherwise this edge must be recovered. (ii) ... (iii) ... □

The above Lemma guarantees that the incremental algorithm must terminate.

In our case, the minimum flip distance (i.e., the number of required flips) between these two triangulations are determined by the number of crossing edges. It is shown that in the worst case, each edge insertion may take $O(m)$ time where $m$ is the number of triangles whose interiors intersect the segment. It is possible that $m = O(n)$.

1.2.4. *Incremental construction.* Although there are many efficient algorithms to construct constrained triangulations. Incremental algorithm is the one that commonly used in practice due to its simplicity and efficiency.

The algorithm begins by constructing an arbitrary triangulation of the point set $S$, then inserts the segments of $L$ into the triangulation one by one. The basic scheme of the algorithm is given below.

**Algorithm**: IncrementalCT($S$, $L$)
**Input**: A PSLG $(S, L)$, $k := |L|$;
**Output**: A constrained triangulation $\mathcal{T}$ of $(S, L)$;
1 construct an initial CT $\mathcal{T}_0$ of $S$;
2 **for** $i = 1$ to $k$ **do**
3 InsertEdge($s_i$, $\mathcal{T}_{i-1}$);
4 **endfor**

FIGURE 8. The incremental constrained triangulation algorithm. $L_i$ is the subset of $L$ containing the first $i$ segments.

Hence the incremental algorithm has a $\Theta(kn^2)$ worst-case runtime, where $n$ is the number of input vertices and $k$ is the number of input segments.

## 2. Constrained Delaunay triangulations

There are many constrained triangulations for the same point set $S$ and $L$. We would like to have one that has similar properties as those of the Delaunay triangulation of $S$. This section introduces such triangulations, called constrained Delaunay triangulations. They were independently developed by Lee and Lin [5] and Chew [2]. After introducing the basic definitions and properties of them, we show how to adapt the incremental construction algorithm to construct them.

2.1. **Definitions and properties.** We use a notion of visibility between points to introduce a special type of constrained triangulation. Points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ are *visible* from each other if the line segment $\mathbf{xy}$ contains no point of $S$ in its interior, and it shares no interior point with a line segment of $L$, i.e., $\text{int}(\mathbf{xy}) \cap S = \emptyset$ and $\mathbf{xy} \cap \mathbf{uv} = \emptyset$, for all $\mathbf{uv} \in L$.

A triangle $\tau$ in a constrained triangulation of $(S, L)$ is *constrained Delaunay* if it has a circumcircle that contains no point in $S$ that is visible from the interior of $\tau$.

A constrained Delaunay triangle is a Delaunay triangle if it does not contain any edge of $L$. Otherwise, it may not be globally Delaunay, since its circumcircle can be non-empty. However, it remains Delaunay if we only consider those points in $S$ that are visible from its interior.

Assume $S$ is in general position, a constrained triangulation $\mathcal{T}$ of $(S, L)$ is the *constrained Delaunay triangulation* (or CDT) of $(S, L)$ if all triangles of $\mathcal{T}$ are constrained Delaunay.

Note that if $L = \emptyset$, then the constrained Delaunay triangulation is just the Delaunay triangulation of $S$. However, it is still unclear whether such a triangulation exists or not. For example, why is it true the collection of constrained Delaunay triangles forms

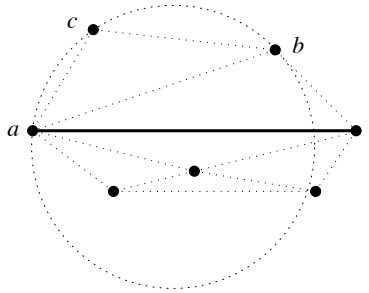


FIGURE 9. Constrained Delaunay triangulation for 7 points and one constraining line segment. The circumcircle of *abc* encloses only points that are invisible from all points of the interior of *abc* (Figure from [4]).

a triangulation? We generalise the concept of being locally Delaunay for edges, and use it to prove the above definition make sense.

2.1.1. *Constrained Delaunay Lemma.* Let $\mathcal{K}$ be any constrained triangulation of $(S, L)$. An edge $\mathbf{ab} \in \mathcal{K}$ is *locally Delaunay* if either:

(i) $\mathbf{ab} \in L$, or
(ii) $\mathbf{ab}$ is on the convex hull of $S$, or
(iii) $\mathbf{d}$ lies outside the circumcircle of $\mathbf{abc}$ where $\mathbf{abc}, \mathbf{abd} \in \mathcal{K}$.

**Theorem 2.1** (Constrained Delaunay Lemma)**.** *If every edge of $\mathcal{K}$ is locally Delaunay, then $\mathcal{K}$ is the constrained Delaunay triangulation of $(S, L)$.*

The proof of the above theorem is similar to the proof of the Delaunay Lemma. The key is to show when all three edges of a triangle is locally Delaunay, then this triangle must be a constrained Delaunay triangle.

2.1.2. *Lawson's flip algorithm.* The above theorem suggests we can use Lawson's edge flip algorithm (in Section **??**) to construct constrained Delaunay triangulations by starting with an arbitrary constrained triangulation of $S, L)$. The only difference to the original algorithm is that the edges in $L$ are not flipped. The algorithm terminates with at most $\binom{n}{2}$ flips, i.e., its runtime is $O(n^2)$.

The analysis of angle changes in each edge flip operation implies that the MaxMin Lemma also holds in the constrained case.

**Theorem 2.2** (The MaxMin Angle Property)**.** *Assume $S$ is in general position. Among all constrained triangulation of $S$ and $L$, the constrained Delaunay triangulation of maximise the minimum angle.*

2.2. **Incremental CDT construction.** Constrained Delaunay triangulations can also be constructed by the incremental algorithm, i.e., start with an initial Delaunay triangulation of the point set, insert the line segments one by one. The difference to the previous algorithm is that after the insertion of each line segment, one needs to re-construct a constrained Delaunay triangulation which includes it.

In this section, we will show how to incrementally construct a CDT by using the incremental algorithm introduced in Section **??**. We will first use the simple edge recovery by flips algorithm to recover a missing edge. Then we immediately use the Lawson's flip algorithm to recover the CDT.

**Algorithm**: IncrementalCDT($S$, $L$)
**Input**: A PSLG $(S, L)$, $k := |L|$;
**Output**: the CDT $\mathcal{T}$ of $(S, L)$;
1 construct an initial CDT $\mathcal{T}_0$ of $S$;
2 **for** $i = 1$ to $k$ **do**
3  **if** $s_i$ is not an edge of $\mathcal{T}_{i-1}$ **then**
4  RecoverEdge($s_i$, $\mathcal{T}_{i-1}$);
5  Let $L$ be the set of new edges in $\mathcal{T}_{i-1}$;
6  ConstrainedLawsonFlip($L$);
7  **endif**
8 **endfor**

FIGURE 10. The incremental constrained Delaunay triangulation algorithm.

Runtime. The runtime of this algorithm is certainly not better than the IncrementalCT algorithm, since it needs an extra step to perform Lawson's flip algorithm. An advantage of this algorithm is simple and easy to implement.

2.3. **Recover edge by weighted Delaunay flips.** In this section, we introduce a simple and efficient approach to insert a segment into the constrained Delaunay triangulation by performing a sequence of edge flips. It is developed by of Shewchuk [8].

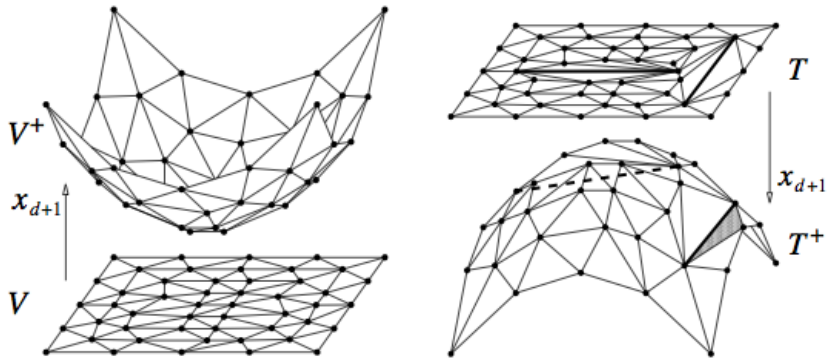


FIGURE 11. Left: The lifting map of a point set and its Delaunay triangulation. Right: The constrained Delaunay triangulation with the lifting map inverted to more clearly show its non-convexity. (Figures from [8]).

2.3.1. *Order flips via the lifting map.* In the previous flip-based edge recovery algorithm, when we want to flip a crossing edge, it may be either unflippable, or it was a previously flipped edge. This is due to randomness of selection of edges. Moreover, the resulting triangulation may be arbitrary which is far from to be constrained Delaunay. By using

the lifting map, it is indeed possible to sort the edges and perform flips following the sorted edges. This way, one can guarantee every edge to be flipped must be flippable. Moreover, the result will be the constrained Delaunay triangulation.

Recall that the Delaunay triangulation is the projection of the convex hull of its lifted point set. The constrained Delaunay triangulation is also related to a such a surface in $\mathbb{R}^2$ which is not necessarily convex. The non-convexity of this surface is due to the constrained line segments, see Figure 11 Right.

Consider every intersecting edge of the missing segment as an flip event. One can use the lifting map to arrange them into a uniquely determined flip sequence. Recall the lifting map $\mathbf{p}'$ of a point $\mathbf{p} = (p_x, p_y) \in \mathbb{R}^2$ is:

$$\mathbf{p}' := (p_x, p_y, p_z) \in \mathbb{R}^3,$$

where $p_z := p_x^2 + p_y^2$, and $\mathbf{p}'$ is point on the paraboloid in $\mathbb{R}^3$. We call $p_z$ is the *height* of the point $\mathbf{p}$. Let $\mathbf{a}$ and $\mathbf{b}$ be the two endpoints of the missing segment $s_i$. The lifted segment $s_i'$ (whose endpoints are $\mathbf{a}'$ and $\mathbf{b}'$) must lie above the lower faces of the convex hull of the lifted point set $S'$, otherwise it must appear in the Delaunay triangulation of $S$.

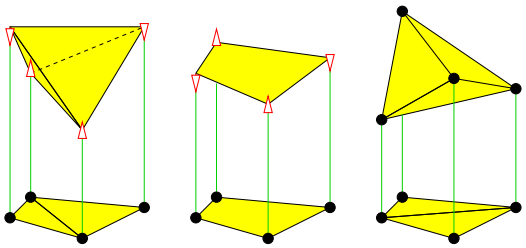

FIGURE 12. An edge flip in the plane corresponds to dynamically changing the heights of their lifted points in $\mathbb{R}^3$ (Figure from [8]).

Now imaging that the heights of $\mathbf{a}$ and $\mathbf{b}$ are dynamically decreasing, from $a_x^2+a_y^2 \to 0$ and $b_x^2 + b_y^2 \to 0$. At certain time $\tau$, a lifted companion, $\mathbf{c}'\mathbf{d}'$ of a crossing edge $\mathbf{cd}$ will eventually become coplanar with $\mathbf{a}'\mathbf{b}'$ in $\mathbb{R}^3$. If the heights of $\mathbf{a}$ and $\mathbf{b}$ decrease further, the edge $\mathbf{c}'\mathbf{d}'$ will not on the convex hull of $S'$ anymore, that is, the edge $\mathbf{cd}$ gets flipped, and is replaced by another edge $\mathbf{ef}$ whose lifted companion $\mathbf{e}'\mathbf{f}'$ lies on the lower face of the convex hull of $S'$. Note that $\mathbf{e}'\mathbf{f}'$ may be just $\mathbf{a}'\mathbf{b}'$, or may correspond to another crossing edge $ef$ of $ab$. This process will eventually make the lifted edge $\mathbf{a}'\mathbf{b}'$ lies on the lower face of the convex hull of $S'$, which implies, the edge $\mathbf{ab}$ is inserted into the current constrained Delaunay triangulation.

2.3.2. *The algorithm.* The algorithm is given below. An example is shown in Figure 14.

The subroutine `add_to_queue` in above is used to sort the sequence of edge flips to be performed. Assume the general position of $S$. Every flip event corresponds to a unique time $t > 0$. It can be directly used as the value to modify the heights of $\mathbf{a}$ and $\mathbf{b}$, i,e,

$$a_z := a_x^2 + a_y^2 - t; \quad b_z := b_x^2 + b_y^2 - t.$$

**Algorithm**: Recover_Edge_by_Flips_WDT($\mathcal{T}_P$, $AB$)
**Input**: A triangulation $\mathcal{T}_P$ of a cavity $P$, and an edge $AB \notin \mathcal{T}_P$;
**Output**: A constrained Delaunay triangulation $\mathcal{T}_P \ni AB$;

```
Initialise an empty priority queue Q;
for each crossing edge xy in D_i;
   add_to_queue(Q, xy, ab);
endfor
while Q is not empty do
   remove the top edge xy from Q;
   if xy is still an edge in D_i then
      flip xy;
      for each crossing edge uv in D_i do
         add_to_queue(Q, uv, ab);
      endfor
   endif
endwhile
```

FIGURE 13. The incremental CDT algorithm to recover line segments by flips ordered by weighted Delaunay.

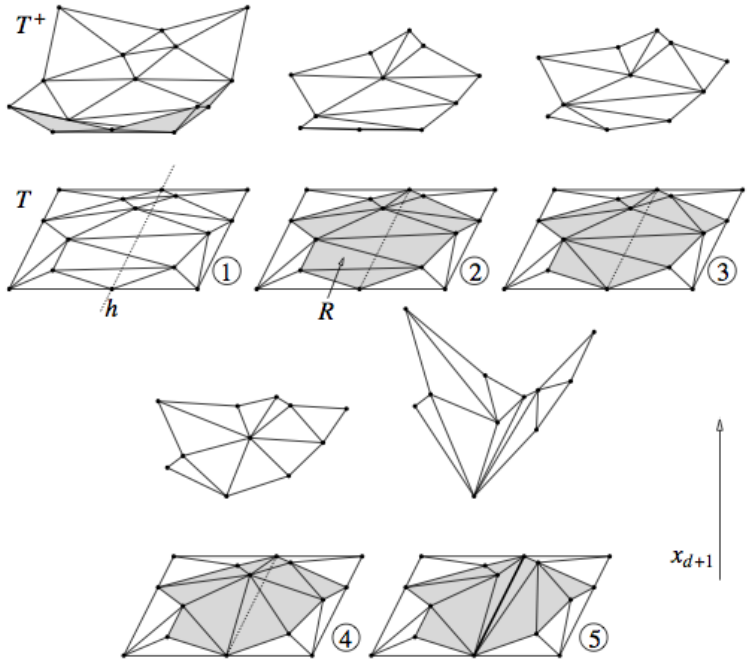


FIGURE 14. Recover an edge in a CDT by flips. (Figures from [8]).

Then the four lifted points $\mathbf{a}'$, $\mathbf{b}'$, $\mathbf{c}'$, and $\mathbf{d}'$ are coplanar in $\mathbb{R}^3$ when the following equation is satisfied:

$$\det \begin{pmatrix} a_x & a_y & a_x^2 + a_y^2 - t & 1 \\ b_x & b_y & b_x^2 + b_y^2 - t & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{pmatrix} = 0 \tag{1}$$

2.3.3. *Runtime.* Shewchuk [8] proved that this incremental edge insertion algorithm has expected runtime $O(n \log^2 k)$, where $n$ is the number of vertices in $S$ and $k$ is the number of segments in $L$. It thus improves the $O(n^2)$ runtime of the generalised Lawson's flip algorithm.

## 3. Delaunay refinement

This section deals with a simple meshing problem in the plane. The input object is a two-dimensional polygonal domain, the goal is to obtain a discretisation of this domain with a triangular mesh. Moreover, it is desired that the shape of triangles are "good", and the total number of triangles are not large. Such basic problem has many applications, for example, function interpolations and finite element simulations. In this section, we discuss a concrete version of two-dimensional mesh generation problem.

3.1. **The meshing problem.** We start with the definitions of the input and output objects. The input $\Omega$ is a polygonal region in the plane, possibly with holes and with constraining edges and vertices inside the domain. The *boundary* $\partial\Omega$ is a set of vertices, edges which separates the interior of $\Omega$ from its exterior. $\partial\Omega$ is a planar straight line graph (PSLG), see Figure 3.1 Left for an example.

The general objective in mesh generation is to decompose a mesh domain bounded by a PSLG. The elements are restricted in type and shape, the number of elements should not be too big. The output of our problem is a triangular mesh with the following properties:

(a) **Conformity**: The output collectively forms a simplical complex $\mathcal{T}$ whose underlying space equal to the given polygonal domain. This means, for every line segment $s \in \partial\Omega$, $s$ is the union of edges of $\mathcal{T}$. We call $\mathcal{T}$ is a *mesh* of $\Omega$.
(b) **Quality**: There are few or no "poor-quality" triangles in $\mathcal{T}$. This means, most of triangles are good with respect to certain quality measures of triangles.
(c) **Cardinality**: It is necessary for $\mathcal{T}$ to include additional points, called *Steiner points*, vertices of the mesh that are not vertices of the input PSLG. Because they are needed to achieve the good shape of elements. We want fewer Steiner points have been added, i.e., the number of triangles is small.

Figure 3.1 Right shows an example of such an output.

Note that the requirements of having good shaped elements and a small number of elements are contradicting to each other. 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 possible in any triangulation of the given input that meets the same shape bound.

Various approaches have been developed for this purpose, such as advancing-front methods, quadtree methods, Delaunay-based methods, and the combinations of them. Most of them works well in practice, but come with no guarantee on quality and size of the generated mesh.

We will demonstrates the use constrained Delaunay triangulations in constructing such meshes. A simple technique – Delaunay refinement – is introduced to solve this basic problem.

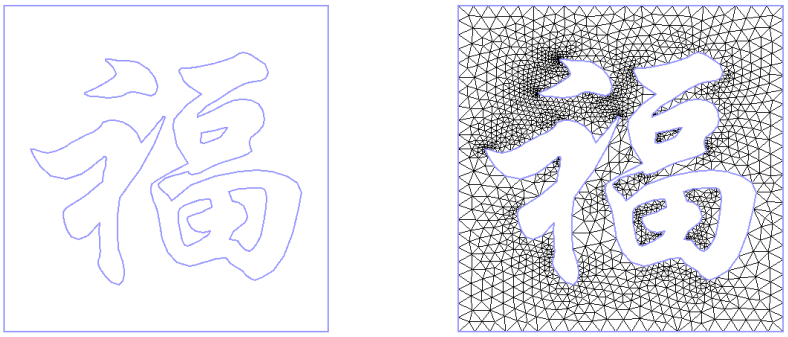

FIGURE 15. Left: an input mesh domain. Right: an output triangular mesh of the input.

3.2. **Quality measures for triangles.** Quality mesh generation describes techniques that offer some guarantees on some measure of shape and size of triangles.

A generally used shape measure for simplices is called *aspect ratio*, which measures the roundness of the element. The aspect ratio of a triangle is the length of the longest edge divided by the length of the shortest altitude.

A fairly general shape measure for triangle is the *minimum angle* $\theta$, since this gives a bound of $\pi - 2\alpha$ on maximum angle and guarantees a lower bound for aspect ratio. Let the longest edge of a triangle $abc$ be $ac$, and assume the smallest angle occurs at $a$. Then $\|b - x\| = \|b - a\| \sin\theta$, where $x$ is the orthogonal projection of $b$ onto $ac$. The edge $ab$ is at least at long as $cb$, and therefore $\|b - a\| \geq \|c - a\|/2$. It follows that

$$\frac{1}{\sin\theta} \leq \frac{\|c - a\|}{\|b - x\|} \leq \frac{2}{\sin\theta}.$$

In word, the aspect ratio (here is $\frac{\|c-a\|}{\|b-x\|}$) of the triangle $abc$ is bounded between $|\frac{1}{\sin\theta}|$ and $|\frac{2}{\sin\theta}|$.

3.3. **Description of the algorithm.** This section presents an incremental point insertion approach extended from a classical Delaunay refinement scheme proposed by Chew [3] and Ruppert [6]. The idea is to add new vertices until the triangulation forms a satisfying mesh. It makes use of many nice geometric properties of Delaunay triangulations.

Given a two-dimensional polygonal domain $\Omega$, we first construct a CDT of $\partial\Omega$. We then add vertices one by one to improve the mesh quality.

We call a triangle *skinny* if it has a minimum angle less than a given limit, which can be a parameter supplied by the users.

Suppose a triangle $abc$ in current CDT is skinny. We add a new point which is the circumcenter of $abc$ into this CDT, see Figure 3.3. Since the circumcircle of $abc$ is no longer empty, triangle $abc$ is guaranteed to be removed by one of the edge flips used to repair the constrained Delaunay triangulation.

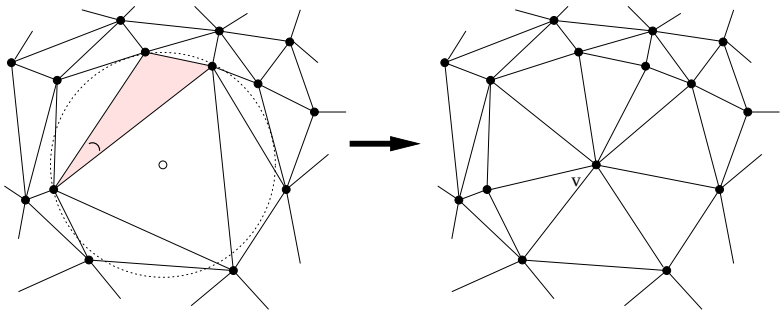


FIGURE 16. Split a bad-quality triangle by adding its circumcenter.

It is possible that a circumcenter of a skinny triangle may lie outside the domain. This can happen since a skinny triangle near the boundary could have an arbitrary big empty circumcircle. A simple way to fix this problem is based on the following fact.

Recall an edge can have arbitrary many circumscribed circles. The smallest circumscribed circle of an edge is the *diametrical circumcircle*. We say an edge is *Gabriel* if its diametrical circumcircle is empty.

**Lemma 3.1.** *If all line segments of $\partial\Omega$ satisfy the Gabriel property, then no circumcenter of triangles of a CDT of $\partial\Omega$ lies outside of $\Omega$.*

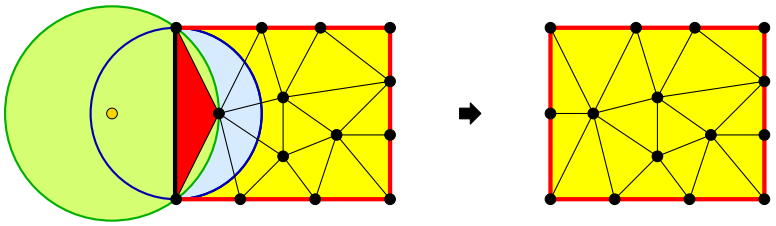

FIGURE 17. Split a boundary segment if its diametrical circumcircle is not empty.

**Algorithm**. After the creation of a CDT of the input PSLG $\partial\Omega$. We use two rules to add new vertices into this CDT until we cannot add any new vertex.

- **R1** The first rule is to ensure that there is no boundary segment will cause a circumcenter of any skinny triangle lies outside of the domain. Call a vertex *encroaches upon* a segment if it lies inside its diametrical circumcircle. If a segment is encroached, we split it by adding its midpoint.
- **R2** The second rule is to improve the mesh quality by removing skinny triangle. If a triangle is skinny, we add its circumcenter $c$. However, if $c$ encroaches upon some boundary segments, do not add $c$, and split one of the encroached segment instead.

The algorithm is given in the following:

A possible run of the above Delaunay refinement algorithm on a simple input is shown in Figure 3.3. In (b)-(e), segments are split due to the encroachment reasons. In (f) a

**Algorithm**: DelaunayRefinement($\Omega$, $\theta_{min}$)
**Input**: A 2d polygonal domain $\Omega$;
$\theta_{min}$ is a desired minimum angle of output triangles.
**Output**: A mesh $\mathcal{T}$ of $\Omega$;

```
construct an initial CDT T of ∂Ω;
while ∃τ ∈ T and MinAngle(τ) > θ_min do
    let c be the circumcenter of τ;
    if c encroaches upon any segment of T then
        split an encroached segment;
    else
        insert c into the CDT T;
    endif
endwhile
```

FIGURE 18. The Delaunay refinement algorithm.

circumcenter is added in a (Quality) operation. In (g) a circumcenter is considered, but it encroaches a segment, as shown in (h); the segment is split instead. In (i) another segment is split. A circumcenter is considered in (j), but it encroaches on two segments, both of which are split. In (l) the same circumcenter is considered again, but instead another segment is split. A circumcenter is committed in (m). In (n) the final vertex and segment sets are shown, in (o) the Constrained Delaunay Triangulation is shown, with segments in bold.

3.4. **Proof of termination.** We provide an analysis the Delaunay refinement algorithm to show that this algorithm will terminate, and it provides a lower bound on the minimum angle of the triangles (based on some assumptions on the input), and it proves an upper bound on the number of triangles of the mesh.

Some preliminary definitions and results are essential to the exposition. We understand the Delaunay refinement algorithm through relating its actions to the *local feature size* at a point $x \in \mathbb{R}^2$, relative to an input PSLG $\partial\Omega$, is defined as a map $f : \mathbb{R}^2 \to \mathbb{R}$, such that $f(x)$ is the smallest radius $r$ of the closed disc with center $x$ and radius $r$ either

(i) contains two vertices of $\partial\Omega$,
(ii) intersects one edge of $\partial\Omega$ and contains one vertex of $\partial\Omega$ that is not endpoint of that edge, or
(iii) intersects two vertex disjoint edges of $\partial\Omega$.

The definition of local feature size is illustrated in Figure 3.4.

The local feature size is a Lipschitz function, i.e., $lfs(x) \leq |x - y| + lfs(y)$. This implies that the local feature size function is continuous.

The *radius-edge-ratio* $\rho(\tau)$ of a triangle $\tau$ is defined as the ratio between the radius $R$ of the circumscribed circle of $\tau$ and the shortest edge length $L$ of $\tau$, i.e., $\rho(\tau) = \frac{R}{L}$.

The radius edge ratio and the minimum angle measure are equivalent. There is a nice relation between the radius edge ratio and the minimum angle of a triangle, which is

$$\rho(\tau) = \frac{R}{L} = \frac{1}{2\sin\theta},$$

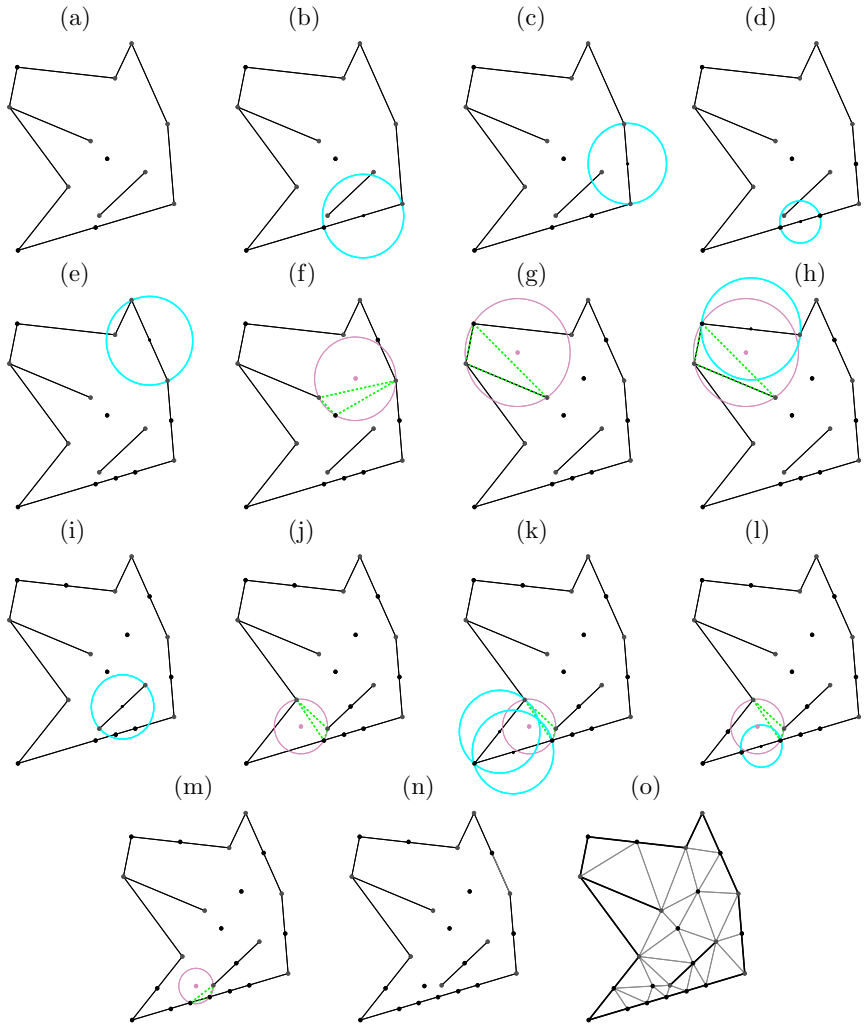


FIGURE 19. An example of the Delaunay refinement algorithm (Courtesy of Steven Pav).

where $\theta$ is the smallest angle of $\tau$. This can be proven from a basic fact of elementary geometry which relates the radius of the circumcircle and its minimum angle, see Figure 3.4, If triangle $abc$ has $\angle bra = \theta$, and $p$ is the circumcenter of $abc$, then $\angle bpa = 2\theta$. By this relation, for a triangle, an upper bound for $\rho(\tau)$ gives a lower bound for the minimum angle of $\theta$.

Let $\rho_0$ be the value of the radius-edge ratio corresponds to the given minimum angle bound $\theta_{min}$.

The algorithm starts with the vertices of $\partial\Omega$ and generates all other vertices in sequence. We show that, when a new vertex is added, its distance to already present vertices is not much smaller than the local feature size.

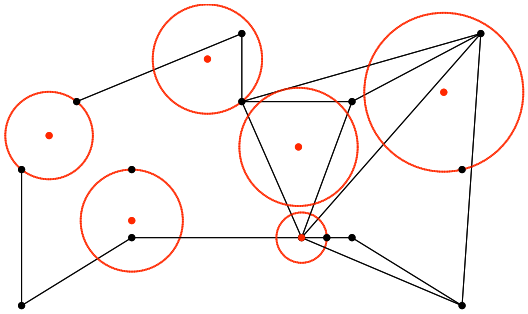

FIGURE 20. For number of points in the plane, the local feature size with respect to a PSLG is shown. About each of the points is a circle whose radius is the local feature size of the centre point.

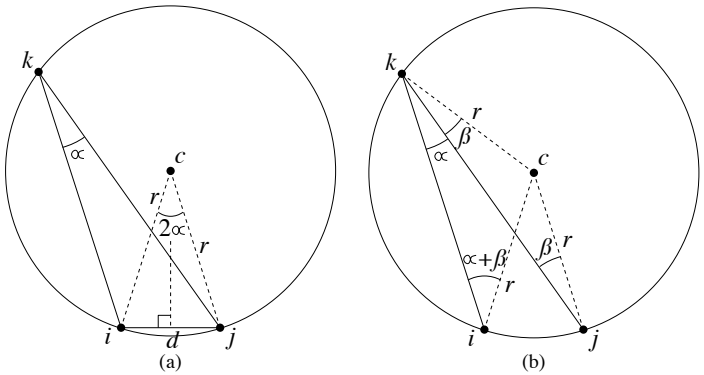


Figure 3.1: (a) Diagram for proof that $d = 2r\sin\alpha$. (b) Diagram for proof that $\angle icj = 2\angle ikj$.

FIGURE 21. The relation between angle of a triangle and its rads-edge ratio.

The following lemma shows the relation of the shortest edge length at a new vertex and an existing length of an edge immediately after it is inserted.

**Lemma 3.2** ([7]). *Let $v$ be a newly inserted vertex, and let $p$ be the closet vertex to $v$. Then one of the following two holds:*

(i) $\|v-p\| \geq lfs(v)$, *or*
(ii) $\|v-p\| \geq \beta\|p-q\|$, *where $q$ is the nearest vertex to $p$ before $v$ is inserted, $p$ may be rejected, where*
   (a) $\beta = \rho_0$, *if $v$ is the circumcenter of a skinny triangle,*
   (b) $\beta = \frac{1}{\sqrt{2}}$, *if $v$ is the midpoint of an encroached segment ($p$ must be a rejected vertex),*
   (c) $\beta = \frac{1}{2\cos\alpha}$ *if $v$ and $p$ lie on incident segments with an angle $\alpha$, with $p$ encroaches upon the subsegment contains $v$, where $45^\circ \leq \alpha < 90^\circ$, and*
   (d) $\beta = \sin\alpha$ *if if $v$ and $p$ lie on incident segments with an angle $\alpha < 45^\circ$.*

The four cases of different constant $\beta$ is shown in the figure 3.4.

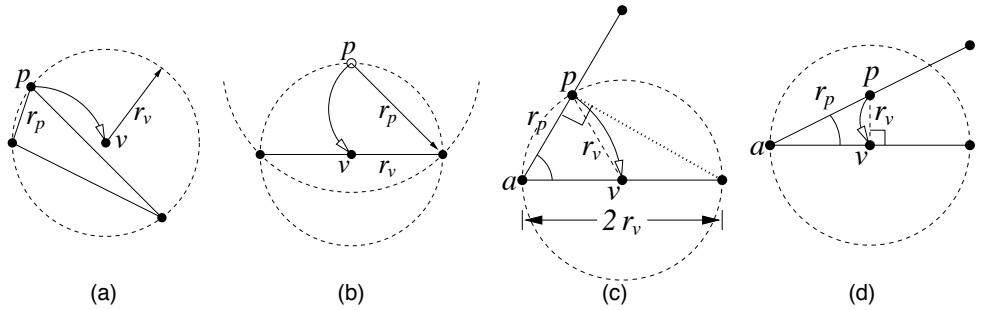


FIGURE 22. The relationship between the shortest edge length and the length of an existing edge.

The above lemma shows how quickly the shortest edge length between newly inserted vertices can increase or decrease. Lets consider a sequence of vertices $v_0, v_1, \cdots, v_i, \cdots$ such that $v_{i+1}$ is inserted strictly after $v_i$. Figure 3.4 shows a flow graph, which shows the worst-case increase/decrease of the shortest edge length in this sequence.

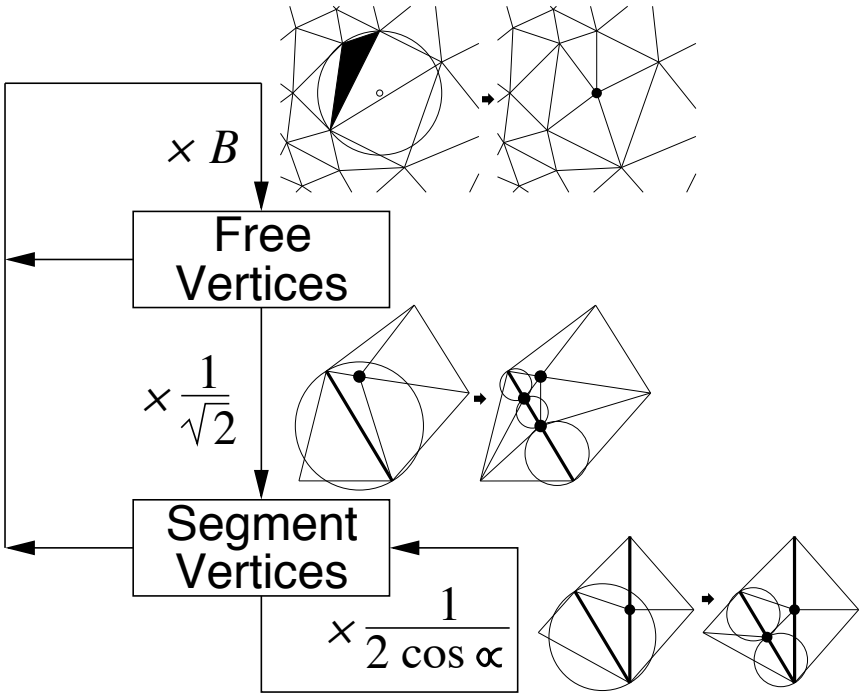


FIGURE 23. Proof of termination. The flow diagram illustrating the worst-case relation of the sequence of shortest edge lengths produced by the Delaunay refinement algorithm.

Hence we have the following theorem which shows under which condition that this algorithm must terminate.

**Theorem 3.1.** *The* DelaunayRefinementAlgorithm *terminates if the following two conditions hold at the same time:*

(i) *the smallest angle between any two incident segments in* $\partial\Omega$ *is not smaller than* $60°$*, and*
(ii) *the minimum angle parameter is not larger than* $\arcsin \frac{1}{\sqrt{2}} \approx 20.7°$.

The condition (ii) provides a default minimum angle guarantee on the output mesh.

3.5. **Output mesh size.** In this section, we show that the output triangulation produced by this algorithm is size-optimal, meaning that the number of triangles is within a constant factor of the minimum number possible. In practice, this means that the output mesh has the following nice properties:

- Small input features will be surrounded by small triangles in order to have good quality.
- Nearby triangles have similar size.
- The size variation between distant triangles depends on their distance.

These properties ensure that the resulting mesh does not contain an unnecessarily large number of triangles while have a good quality. The mesh is *graded*, which means the edge lengths changing slowly from small to large triangles, see Figure **??** for an example.

The basic idea would be to show that in an optimal mesh, triangle sizes must vary slowly, proportional to the local feature size of the input. What we need to find is the relation of every edge length of the mesh to the local feature size of its endpoints. If this length is always bounded by a constant times the local feature size of its endpoints, then this mesh must have the above desired property. The following lemma shows this property.

**Lemma 3.3.** *Let* $x$ *be a vertex in the output mesh, and* $p$ *be the closet vertex of* $x$*. Let* $C_1$ *and* $C_2$ *be two constants. Then*

(A) $\|x - p\| \geq lfs(x)$ *if* $x$ *is a vertex of the input PSLG* $X$*,*
(B) $\|x - p\| \geq lfs(x)/C_1$ *if* $x$ *is added at the midpoint of a segment of* $X$*; or*
(C) $\|x - p\| \geq lfs(x)/C_2$*, if* $x$ *is the circumcenter of some skinny triangle.*

3.6. **Failures of Delaunay refinement.**

## 4. Mesh Adaptation

## References


[1] L. P Chew. Building voronoi diagrams for convex polygons in linear expected time. Technical report, Hanover, NH, USA, 1990.
[2] L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.
[3] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Dept. of Comp. Sci., Cornell University, 1989.
[4] Herbert Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge University Press, Cambridge, England, 2001.
[5] D. T. Lee and A. K. Lin. Generalized Delaunay triangulations for planar graphs. *Discrete and Computational Geometry*, 1:201–217, 1986.
[6] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[7] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry Theory and applications*, 22:21–74, 2002.

[8] Jonathan R. Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. 19th Ann. Symp. on Comput. Geom.*, pages 86–95, 2003.

[9] Jonathan Richard Shewchuk and Brielin C. Brown. Fast segment insertion and incremental construction of constrained delaunay triangulations. *Computational Geometry*, 48(8):554–574, 2015.

[10] Hang Si and Jonathan Richard Shewchuk. Incrementally constructing and updating constrained delaunay tetrahedralizations with finite-precision coordinates. *Engineering with Computers*, 30(2):253–269, 2014.