# 3D Delaunay Refinement of Sharp Domains without a Local Feature Size Oracle

Alexander Rand and Noel Walkington

Carnegie Mellon University

**Summary.** A practical incremental refinement algorithm for computing a quality, conforming Delaunay mesh of an arbitrary 3D piecewise linear complex is given. The algorithm allows small input angles and does not require the local feature size of the input to be computed before performing the refinement. Small input angles are protected with a new type of collar which is simpler to implement and analyze than previous approaches. The algorithm has been implemented and several computational examples are given.

## 1 Introduction

Ruppert's algorithm [9] is an elegant method for computing a quality conforming Delaunay mesh of a two-dimensional piecewise linear complex (PLC) which contain no acute angles. Based on Ruppert's original techniques of corner lopping and concentric shell splitting, the algorithm has been rigorously extended to 2D PLCs containing sharp input angles. Ruppert's algorithm has also been extended to handle three-dimensional PLCs without acute input angles [4]. Handling sharp angles in 3D leads to a variety of issues not seen in 2D. Subsequent algorithms have been complicated (in some cases to the point of prohibiting implementation) and lacked certain desirable properties seen in the original method.

Early algorithms for computing conforming Delaunay tetrahedralizations (without considering radius-edge quality) in 3D [6, 3] involved computing local feature size on all segments of the input through a brute force search before inserting Steiner points. Several extensions of these methods to the problem of quality mesh generation [2, 1] have also required the computation of local feature size as a preprocess. An algorithm of Pav and Walkington [7] gives an alternative view: Delaunay refinement should be considered an algorithm for computing local feature size and thus local feature size should not be a necessary input for the algorithm. They gave a 3D Delaunay refinement algorithm for which all steps require only "local" information about the mesh in the Delaunay tetrehedralization. Still, this algorithm is not local in the same way as Ruppert's algorithm: steps in the algorithm require searching the Delaunay triangulation over a prescribed distance rather than only querying the immediate Delaunay neighbors. This issue of finding a local algorithm for computing constrained Delaunay

tetrahedralizations has been resolved [12, 11], but has not been addressed in the case of (usual) Delaunay tetrahedralization.

We present a Delaunay refinement algorithm for general PLCs which attempts to preserve as many features of Ruppert's algorithm as possible. The algorithm does not require local feature size information to be precomputed. Each operation of the algorithm is local to the current Delaunay tetrahedralization in the sense that encroachment can be determined considering only a point's immediate Delaunay neighbors. The result of the algorithm is a conforming Delaunay tetrahedralization consisting of tetrahedra with bounded radius-edge ratio away from sharp input angles. The strategy for protecting sharp angles is designed for simplicity both in implementation and analysis. The authors are unaware of any algorithm for quality, conforming Delaunay refinement in 3D which has been implemented. Existing software packages which handle small input angles are based on constrained Delaunay tetrahedralization [12, 11] and weighted Delaunay tetrehedralization [1].

To highlight the differences between the algorithm and other extensions of Ruppert's algorithm to three dimensions, an analogous two dimensional version is developed and analyzed. In both cases, a Delaunay refinement algorithm is used to prove estimates on the local feature size of the mesh at input points (in 2D) or on input segments (in 3D). These are the reverse inequalities of those usually seen in the analysis of Delaunay refinement algorithms; while typically local feature size is shown to be a lower bound for the distance between points in the mesh, we show that local feature size bounds this distance (in the form of segment lengths in 3D) from above. With these estimates, a "collar" can be inserted near these sharp angles which ensures that the resulting mesh conforms to the input at termination. An example of the collar used to protect sharp input features can be seen in Figure 6 in Section 4.

In Section 2, we define the problem and a number of feature size functions used in the analysis. Sections 3 and 4 outline the algorithms in two and three dimensions, respectively, and justify termination and correctness. Several resulting meshes from the 3D implementation are given in Section 5.

## 2 Problem Description and Notation

We describe algorithms for meshing a general PLC given by sets of input points and input segments in 2D, $\mathcal{C} = (\mathcal{P}, \mathcal{S})$, or sets of input points, input segments and input faces in 3D, $\mathcal{C} = (\mathcal{P}, \mathcal{S}, \mathcal{F})$. In 2D, a PLC $\mathcal{C}' = (\mathcal{P}', \mathcal{S}')$ is a refinement of the PLC $(\mathcal{P}, \mathcal{S})$ if $\mathcal{P}' \subset \mathcal{P}$ and each segment in $\mathcal{S}$ is the union of segments in $\mathcal{S}'$. In 3D, a PLC $(\mathcal{P}', \mathcal{S}', \mathcal{F}')$ refining $(\mathcal{P}, \mathcal{S}, \mathcal{F})$ must satisfy the additional condition that each face in $\mathcal{F}$ must be the union of faces in $\mathcal{F}'$.

An important parameter to the algorithm is a raduis-edge ratio threshold, $\kappa$, which must be greater than 2. The algorithm eliminates all tetrahedra which have radius-edge ratio greater than $\kappa$ except for some which are near the sharp angles in the input.

The algorithms described involve incrementally refining the input PLC until the Delaunay triangulation/tetrahedralization of the point set $\mathcal{P}'$ conforms to the input PLC and satisfies the radius-edge quality criteria. Throughout this refinement, it is necessary to identify some important simplices based on their relation to the input complex.

**Definition 1.** *Consider a refinement* $(\mathcal{P}', \mathcal{S}', \mathcal{F}')$ *of an input PLC* $(\mathcal{P}, \mathcal{S}, \mathcal{F})$.

- *An* **end segment** *is a segment in* $\mathcal{S}'$ *for which at least one endpoint is an input point in* $\mathcal{P}$.
- *An* **end facet** *is a facet in* $\mathcal{F}'$ *for which at least one vertex lies on an input segment in* $\mathcal{S}$.

The smallest angle in the input PLC will be denoted by $\alpha$. In 2D, this is the smallest angle between adjacent input segments. In 3D, this is the smallest angle between any pair of adjacent input features: faces or segments. Let $\mathbf{C}_{d,\alpha}$ denote the set of $d$-dimensional PLCs for which no two features meet at an angle of less than $\alpha$.

An appropriate notion of feature size is essential to the analysis of Delaunay refinement algorithms. The standard definition of local feature size is given below as well as another sizing function (mesh feature size defined below) which is used throughout the arguments.

**Definition 2.** *Let* $\mathcal{C}$ *be a PLC.*

- *The* $i$**-local feature size** *at point* $x$ *with respect to* $\mathcal{C}$, $\mathrm{lfs}_i(x, \mathcal{C})$, *is the radius of the smallest circle centered at* $x$ *which touches two* disjoint *features of* $\mathcal{C}$ *of dimension no greater than* $i$.
- *The* $i$**-mesh feature size** *at point* $x$ *with respect to* $\mathcal{C}$, $\mathrm{mfs}_i(x, \mathcal{C})$, *is the radius of the smallest circle centered at* $x$ *which touches two features of* $\mathcal{C}$ *of dimension no greater than* $i$.
- *The* **current feature size** *with respect to* $\mathcal{C}'$, $N(x, \mathcal{C}') := \mathrm{lfs}_0(x, \mathcal{C}')$, *is the distance from* $x$ *to its second nearest neighbor in* $\mathcal{P}'$.

Each of these feature size functions is Lipschitz (with constant 1). For a fixed PLC, local and current feature size each have a strictly positive minimum value while mesh feature size can equal zero.

If the argument supplied to any of the above feature size functions is a set of points, rather than a point, then the result is defined to the be infimum of the function over the set. The dimension argument will be omitted when considering the $d-1$ dimensional feature sizes. For instance $\mathrm{lfs}_i(s, \mathcal{C}) := \inf_{x \in s} \mathrm{lfs}_i(x, \mathcal{C})$ and, in 3D, $\mathrm{lfs}(x, \mathcal{C}) := \mathrm{lfs}_2(x, \mathcal{C})$.

The distinction in notation between local/mesh feature size and current feature size is deliberate: these functions will be used differently.

**Convention:** Throughout the analysis, local/mesh feature size will always be evaluated with respect to the input PLC, while current feature size will always involve the current refined PLC. By adhering to this usage, the PLC argument

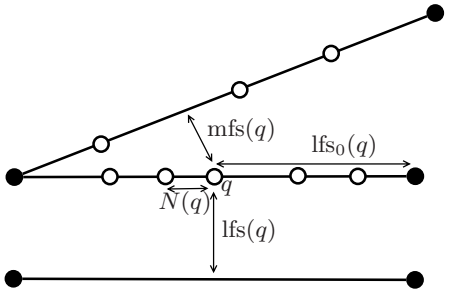


**Fig. 1.** Example of sizing functions in Definition 2 for a 2D PLC. The black points represent input points while the white points represent points added by the algorithm.

can be omitted. See Figure 1 for an example the different feature size functions for a simple PLC.

Finally, much of the analysis involves giving identical estimates for the local feature size of end segments and the mesh feature size of non-end segments. It is useful to refer to these two cases with the same notation.

**Definition 3.** *The* $i$**-feature size** *of segment* $s$ *is defined as follows.*

$$\mathrm{fs}_i(s) = \begin{cases} \mathrm{lfs}_i(s) & \text{if } s \text{ is an end segment,} \\ \mathrm{mfs}_i(s) & \text{if } s \text{ is a non-end segment} \end{cases}$$

*Given a segment or triangle* $s$ *in* $\mathcal{C}'$*, point* $x$ *is called an* $i$**-feature size witness** *for* $s$ *if* $x$ *is contained in a feature of* $\mathcal{C}$ *of dimension at most* $i$ *which is disjoint from* $s$*. For segment or triangle* $s$ *in* $\mathcal{C}'$*, point* $x$ *is called a* **local feature size witness** *for* $s$ *if* $x$ *is contained in a feature of* $\mathcal{C}$ *which is disjoint from another feature of* $\mathcal{C}$ *containing* $s$*.*

This definition of feature size is similar to the local gap size used by Chen and Poon [2]. The notion of $i$-feature size witness is used by recognizing that if $x$ is an $i$-feature size witness of segment $s$ then

$$\mathrm{fs}_i(s) \leq \mathrm{dist}(x, s).$$

## 3 Delaunay Refinement Algorithm in 2D

The following algorithm can be used to mesh any 2D PLC and extends many of the properties of Ruppert's Delaunay refinement algorithm for PLCs in $\mathbf{C}_{2,90}$. Several effective approaches to protecting acute angles from cascading encroachment sequences have already been developed for this problem [9, 10, 5, 8]. However, we present a different corner-lopping algorithm which mirrors the 3D algorithm given in Section 4. In what follows, small angles are protected by splitting all acutely adjacent segments at an equal distance. Then, these segments are

protected by rejecting any circumcenters proposed for insertion which encroach the diametral disk of an end segment. This algorithm differs from previous versions as we compute an acceptable length for the protected end segments in the output mesh before refining any triangles for quality.

*Algorithm Overview*

The algorithm consists of the following steps.

- (Step 0) Compute the Delaunay triangulation of the input point set.
- (Step 1a) Estimate local feature size at all input points.
- (Step 1b) Protect small input angles with a "collar".
- (Step 2) Perform standard Delaunay refinement away from the collar.

Step 0 is a standard procedure and common in many Deluanay refinement algorithms. Steps 1a and 2 are Delaunay refinement algorithms. Step 1b involves a single pass over the set of input points to insert points. Properties of the mesh generated by this algorithm are summarized in two theorems.

**Theorem 1 (Termination/Grading).** *The algorithm terminates. Moreover, there exists $c_\alpha > 0$ such that for each point $p$ added by the algorithm,*

$$N(p) \geq c_\alpha \text{lfs}(p)$$

Verifying Theorem 1 holds for each step in the algorithm uses the standard arguments; the insertion radius of each point is inductively bounded below by feature size.

Ruppert's analysis of the standard Delaunay refinement algorithm for non-acute domains showed that this condition is sufficient to ensure a size-competitive output mesh. In our case, $c_\alpha = O(\frac{1}{\sin \alpha})$ and thus, this condition does not guarantee a competitive output size over the class of all 2D PLCs but does ensure that the output is size competitive on $\mathbf{C}_{2,\alpha}$ for any fixed $\alpha$.

When considering the standard Delaunay refinement algorithm for non-acute domains, termination implies that the output conforms to the input PLC and that no poor quality triangles remain. A guarantee of this type requires more attention when dealing with acute angles.

**Theorem 2 (Conformity/Quality).** *At termination, the resulting Deluanay triangulation conforms to the input PLC. Moreover, any remaining poor quality triangles are near the collar region.*

Ensuring that the conformity/quality theorem holds requires a series of lemmas related to each step. With the lemmas, the notion that a triangle is near the collar will be made precise.

*Step 1a*

This step is a Delaunay refinement algorithm. Segments are queued to be split based on the following encroachment criteria and no triangles are split for quality during this step.

- (Segment Encroachment) Segment $s$ is encroached if it has an endpoint $q$ with a Delaunay neighbor $p$ which lies in the diametral disk of $s$ and either $p$ is a feature size witness for $s$ or $s$ is an end segment and is more than twice as long as its shortest adjacent end segment.

This is the usual definition of encroachment with one modification: an end segment cannot be encroached by a point on a sufficiently long adjacent end segment. This ensures that adjacent end segments do not alternately encroach each other and prevent termination of the algorithm. Thus, this definition of encroachment is strictly weaker than the standard diametral disk notion, yet shares the property that encroachment of a segment can be checked by only considering the Delaunay neighbors of its endpoints.

The triangulation resulting from this step does not necessarily conform to the input PLC and may contain poor quality elements. However, using this triangulation, it is possible to bound the local feature size at input points in terms of a local quantity (in the Delaunay triangulation): the distance from the input point to its nearest neighbor.

**Lemma 1.** *After Step 1a terminates, the following inequality holds for all input points $q_0 \in \mathcal{P}$.*

$$N(q_0) \leq \sqrt{2}\mathrm{lfs}(q_0).$$

This inequality is in the opposite direction of those in the standard analysis of Delaunay refinement algorithms and Theorem 1. Rather than guaranteeing points are far enough apart, it ensures that the distance from each input point to its nearest neighbor is not much greater than the local feature size.

While the constant in Theorem 1 depends on $\alpha$ and can be large, the constant in Lemma 1 is sharp and independent of $\alpha$. This estimate gives an appropriate length at which segments can be split to protect acutely adjacent segments.

While the bound in Lemma 1 can be shown for any point in the mesh, $q \in \mathcal{P}'$, the statement is deliberately only made for input points. In order to protect sharp angles, it is necessary to estimate the feature size on all $d-2$ dimensional input features. The same principle will be applied in the 3D version.

*Step 1b*

End segments are split in this step to form a "collar" around each input point which ensures end segments conform to the input, even near skinny input angles. For each input point $q_0$, all segments containing $q_0$ are split at a distance of $\frac{N(q_0)}{3}$ away from $q_0$.

Any point added during this step will be called a **collar point**. All end segments are considered to be **collar simplices**. (This distinction is not particularly useful in the 2D algorithm but it will be necessary to consider the analogous simplices in the 3D version.) The **collar region** is defined to be the set of collar simplices.

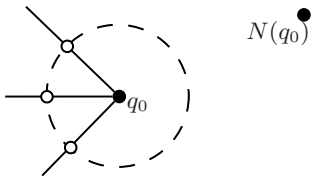


**Fig. 2.** Adjacent segments are split at equal length in Step 1b

**Lemma 2.** *Following Step 1b, the following properties hold.*

- *The diametral disk of each collar simplex contains no points of* $\mathcal{P}'$.
- *The diametral disk of each collar simplex does not intersect any non-collar segment.*

Lemma 1 is used to show that the ball of radius $\frac{N(q_0)}{3}$ centered at $q_0$ does not intersect any segments which are disjoint from $q_0$. The results of Lemma 2 will ensure that these segments will not need to be split again in the following Delaunay refinement step.

*Step 2*

With the collar in place, it is now safe to perform a slightly modified version of Ruppert's algorithm. In this step, segments are queued for conformity based on the standard diametral disk encroachment criteria, and triangles are queued based on the radius edge-quality criteria. As the input may contain small angles, an additional rule is needed to ensure the algorithm terminates despite some poor quality triangles near these small input angles. When processing poor quality triangles from the queue, the following policy is considered before inserting the circumcenter.

- Let $c$ be a circumcenter proposed for insertion to split a poor quality triangle. If $c$ lies in the diametral disk of a collar simplex, reject $c$.

**Lemma 3.** *No point is inserted in the diametral disk of a collar segment.*

With this lemma, standard techniques can be used to verify Theorem 1 and Theorem 2. Any remaining poor quality triangles must have circumcenters which lie in the diametral disk of some collar simplex (i.e. end segment). This makes precise what it means in Theorem 2 for a triangle to be "near the collar."

## 4 Delaunay Refinement Algorithm in 3D

A similar approach to that given in 2D can be applied to 3D PLCs. It is important to note that *input points* in the two dimensional problem play the role of *segments* in the 3D case. In 2D, we used the distance to the nearest neighbor to estimate the local feature size at an input point, while, in 3D, each segment's length will be used to estimate its feature size.

*Algorithm Overview*

The first steps of the algorithm provide an estimate of the feature size on each segment of the input. Then a collar is inserted around the 1-skeleton to ensure the mesh conforms to the input near skinny input angles. Finally, the usual Delaunay refinement algorithm is performed with a rejection policy to ensure both that the algorithm terminates and that the resulting mesh conforms to the input PLC.

- (Step 0) Compute the Delaunay triangulation of the set of input points.
- (Step 1a) Protect small angles between segments by splitting based on 0-local feature size.
- (Step 1b) Estimate the 1-feature size on all segments.
- (Step 2a) Split segments to improve the 1-feature size estimate.
- (Step 2b) Estimate the local feature size on all segments.
- (Step 2c) Protect the 1-skeleton with a collar.
- (Step 3) Perform standard Delaunay refinement away from the collar.

Steps 1a, 2a, and 2c occur in one pass over the mesh: they loop through all the segments and add an appropriate number of points for each segment. Steps 1b, 2b and 3 are Delaunay refinement algorithms. Each of these steps has its own notion of encroachment by which simplices are queued and split according to some priority. Only in Step 3 are tetrahedra split based on radius-edge quality. Figure 9 in Section 5 shows a face of an example mesh produced at each step of the algorithm.

The output guarantees of the algorithm are split into two theorems.

**Theorem 3 (Termination/Grading).** *The algorithm terminates. Moreover, there exists $c_\alpha > 0$ such that for any point $p$ in the output mesh,*

$$N(p) \geq c_\alpha \mathrm{lfs}(p).$$

This theorem is shown via induction using similar techniques to algorithms for large angle input. In the case of inputs with small angles, the next theorem is also essential.

**Theorem 4 (Conformity/Quality).** *The resulting mesh is a conforming Delaunay refinement of the input PLC. Moreover, all resulting tetrahedra with radius-edge ratio greater than $\kappa$ are near the collar region.*

Throughout the refinement, independent Delaunay meshes for each of the input features (segments and faces) are maintained in additional to the complete tetrahedralization. (This also occurs in 2D case, but the meshes of segments are simple and no hierarchy needs to be considered.) To ensure correctness of the algorithm, simplex splits and point insertions must be processed following a partial order given in [4]. Segments and triangles in the refinement refer to simplices in the separate meshes of the input features. However, when discussing Delaunay neighbors of the endpoint of a segment or vertex of a triangle, we are referring

to the neighbors in the 3D mesh. To ensure that this makes sense, we further restrict the required partial order to always handle forced point insertions before handling simplex splits. For example, if a segment is split, then its midpoint is immediately inserted into all meshes containing the segment before another simplex is split in any mesh. This ensures that for any point in a lower dimensional mesh, its Delaunay neighbors in the 3D mesh can still be considered.

In order to prove Theorem 4, conditions on the mesh following each step are shown.

*Step 1a*

Once the Delaunay triangulation of the point set has been computed, each segment is split at one-third the distance from an input point to its nearest neighbor. This is a typical "grooming" step in two dimensional algorithms [5]. This ensures that each end segment in the mesh conforms to the input.

*Step 1b*

This is the first Delaunay refinement step. In this case, segments are queued based on the following encroachment criteria.

- (Segment Encroachment) Segment $s$ is encroached if it has an endpoint $q$ with a Delaunay neighbor $p$ such that $p$ is a 1-feature size witness for $s$ and $|p - q| < |s|$.
- (End Segment Balance) Non-end segment $s_n$ is encroached if it is adjacent to an end segment $s_e$ and $|s_e| < |s_n|$.

The encroachment criteria only depend on the Delaunay neighbors of the endpoints of the segment in question in the current Delaunay triangulation.

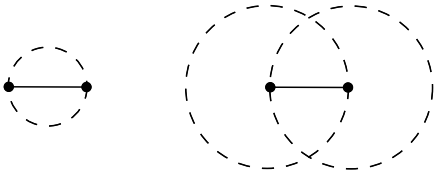

**Fig. 3.** (Left) Encroachment region for a segment in Ruppert's algorithm. (Right) Encroachment region for a segment in Step 1b.

Also, observe that the end segment balance criterion is necessary in order to ensure that the feature size bound in the next lemma holds. This ensures that a long non-end segment segment will be split even if a nearby feature is hidden from the non-end segment by an input point. See Figure 4.

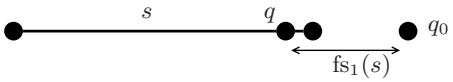


**Fig. 4.** End segment balance is essential. Without this, segment $s$ may be much longer than the local feature size on the segment.

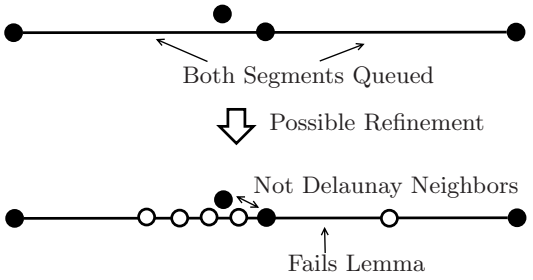


**Fig. 5.** Poor refinement order can lead to a failure of Lemma 4

**Lemma 4.** *After Step 1b, for any segment $s$ in the mesh, $|s| \leq \sqrt{2}\,\mathrm{fs}_1(s)$.*

Ideally, this lemma would follow from the algorithm without further restriction. However, for the algorithm as specified this may not be true! Consider a situation as seen in Figure 5. If the algorithm always processes segments on the left first, once the segment on the right splits for the first time, it may no longer have a 1-feature size witness which is on its Delaunay cavity. At this point, though, the segment will not have been split enough to satisfy Lemma 4.

To prevent this issue we prioritize the queue of segments to be split according to the following rule.

- Segments are prioritized by length. Longer segments are always split first.

We choose this ordering to get a sharp constant in the lemma. This can be seen from the following example. Consider the segment between $(-1, 0, 0)$ and $(1, 0, 0)$ and the segment between $(0, -1, \sqrt{2})$ and $(0, 1, \sqrt{2})$. Each segment has length two and the distance between the two segments is $\sqrt{2}$. The distance between any pair of endpoints is 2, and thus neither segment is encroached according to this step of the algorithm. The following proposition shows that this example gives the nearest two segments which are not encroached.

**Proposition 1.** *Let $s$ and $\bar{s}$ be segments in $\mathbb{R}^3$. If $|s| \geq |\bar{s}|$ and $\mathrm{dist}(s, \bar{s}) < \frac{|s|}{\sqrt{2}}$, then there are endpoints $q$ and $\bar{q}$ of $s$ and $\bar{s}$, respectively, such that $|q - \bar{q}| < |s|$.*

*Step 2a*

The constant in the bound of Step 1b is not strong enough for the upcoming argument in Step 2b. Thus, in this step, all segments are split into fourths, giving the needed bound.

**Lemma 5.** *Following Step 2a, for any segment in the mesh,*

$$|s| \leq \frac{1}{2\sqrt{2}}\text{lfs}_1(s).$$

The estimate $|s| \leq \frac{1}{2\sqrt{2}}\text{fs}_1(s)$ does not hold after this step. This is due to the fact that when an end segment is split, the resulting non-end segment can have a 1-mesh feature size which is much smaller than its 1-local feature size.

*Step 2b*

This step is similar to Step 1b with an additional encroachment rule for triangles in the current Delaunay triangulation of the faces. Segments and faces are queued for encroachment according to the following rules.

- (Segment Encroachment) Segment $s$ is encroached if it has an endpoint $q$ with a Delaunay neighbor $p$ such that $p$ is a 1-feature size witness for $s$ or a local feature size witness for $s$ and $|p - q| < |s|$.
- (End Segment Balance) Non-end segment $s_n$ is encroached if it is adjacent to an end segment $s_e$ and $|s_e| < |s_n|$.
- (Face Encroachment) A triangle $t$ is encroached if it has a vertex $q$ with Delaunay neighbor $p$ such that $p$ is not in the face containing $q$ and $|p - q| < 2r_t$. (The circumradius of $t$ is denoted $r_t$.)

Without a rejection policy, the above encroachment rules can (and do) lead to non-termination. This can occur when two acutely adjacent faces alternately cause each other to split. This is prevented by the following rule.

- Suppose a circumcenter of a triangle, $c_t$, is proposed for insertion into a face $F$ and has a prospective Delaunay neighbor, $p$, which is the end point of a segment, $s$, contained in $F$, and $|c_t - p| < |s|$. Then $c_t$ is not inserted into the mesh.

As in Step 1b, an ordering of the queue is necessary to get the proper bounds on segment lengths. The queue priority is as follows.

1. Triangles.
2. Segments by length. Longer segments are split first.

Using this algorithm, it is now possible to extend the result in Lemma 4 to include a bound based on local feature size.

**Lemma 6.** *After Step 2b, for any segment $s$ in the mesh, $|s| \leq \frac{5}{3}\text{lfs}(s)$ and $|s| \leq \sqrt{2}\text{fs}_1(s)$.*

*Step 2c*

Lemma 6 allows for the insertion of points to form a collar region around the segments in each face. This will ensure both that the mesh conforms to the input

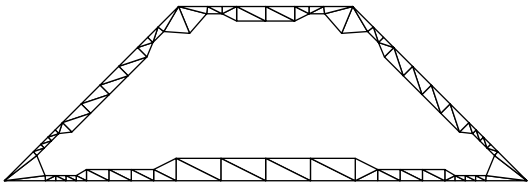

**Fig. 6.** Collar simplices in a face

near skinny angles and that Ruppert's algorithm will terminate when applied away from these regions.

The collar ensures conformity near the boundary by placing points in each face based only on the segments on the boundary of the face. This ensures that end triangles in adjacent faces "line up" so they do not encroach each other.

In order to prevent the collar from tangling, we need a slightly stronger bound on the feature size of segments: $|s| < \mathrm{fs}_1(s)$. This can be attained by splitting all non-end segments, as all end segments satisfy this bound following Step 2a.

The collar is formed by inserting points according to the following rules.

- If $s$ and $s'$ are adjacent non-end segments containing point $q$, then a point $p$ is inserted at distance $\frac{\max(|s|,|s'|)}{2}$ from $q$, in the direction into the face perpendicular to $s$.
- If $s$ is an end segment and $s'$ is an adjacent non-end segment, both containing $q$, then insert $p$ at the intersection of the line parallel to $s$ in the face at distance $\frac{|s'|}{2}$ away from $s$ and on the circle of radius $|s|$ around the input point on $s$.
- Given any input point $q_0$ in the face, insert a point $p$ in the face such that the circle of radius $N(q_0)$ around $q_0$ has no arcs of angle greater than 90 degrees.

Let each point added during this step be called a **collar point**. For each segment, let the segment between the two collar points corresponding to the end points of the segment be called a **collar segment**. Similarly, the segments which connect collar points on the disk in a face around an input point are also considered collar segments. **Collar simplices** are both subsegments of input segments and triangles which are between a segment or input point and its associated collar segment. While some collar segments may be split in Step 3 of the algorithm, the collar region never changes. This is analogous to the 2D version: once adjacent end segments are split at an equal distance, they are never split again. See Figure 6 for an example of the points added to a face to form the collar.

**Lemma 7.** *After Step 2c, the following properties hold.*

- *All adjacent collar segments meet at non-acute angles.*
- *In each face, the diametral disk of each collar segment contains no points of* $\mathcal{P}'$.
- *The circumball of any collar simplex contains no points of* $\mathcal{P}'$.

- *The circumball of any collar simplex does not intersect any disjoint faces or segments.*

Each of the above properties is important. Since the circumball of each collar simplex is empty, this ensures that the collar simplices conform to the input. Collar segments meeting non-acutely ensures that the complement of the collar region in each face is well-suited for Ruppert's algorithm. The final property is a result of the feature size bounds in the Step 2b, and is needed to guarantee that subsequent points inserted in the mesh for conformity will not encroach disjoint collar simplices.

*Step 3*

In the final step, the volume mesh is refined based on both quality and conformity criteria using Ruppert's algorithm. Similar to the non-acute case, any maximum radius-edge threshold $\kappa > 2$ can be selected for determining poor quality tetrahedra. As seen in the 2D case, a rejection policy must be followed to ensure that adjacent faces do not lead to cascading encroachment.

- Let $c$ be a circumcenter proposed for insertion to split a skinny tetrahedron. If $c$ lies in the circumball of a collar simplex, reject $c$.

During the algorithm, it is important to ensure that the properties of the collar in Lemma 7 continue to hold while allowing refinement of the non-collar region of each face to create a conforming mesh. As in 2D, the "collar region" does not change, however the set of collar simplices does change. This occurs when the standard Delaunay refinement algorithm seeks to insert a point in a face that encroaches upon a collar segment. Instead of adding this encroaching point, this collar segment is split. This new point is considered a collar point and the collar segment is broken into two new collar segments. The collar region has not changed but there are new collar simplices. This new point may encroach upon the circumball of another collar element in an adjacent face. In this face, the collar segment associated with this encroached circumball is also split so that the collar simplices on adjacent faces again "line up."

**Proposition 2.** *Let $A$ be the set of vertices on the boundary of some face $F$ and let $A'$ be a set of points inside $F$. Suppose that for each boundary segment of $F$ there is a circle through the end points of the segment which does not contain any points of $A \cup A'$ in its interior. Then the Delaunay triangulation of $A \cup A'$ conforms to $F$. Moreover, for any Delaunay triangle $t$ in the interior of $F$, the circumcenter of $t$ either lies inside $F$ or inside the empty disk associated with a boundary segment.*

A face with the collar simplices removed satisfies the hypothesis of the lemma (as the circumdisk of each collar segment is empty). This ensures that whenever a circumcenter of a triangle in a face is proposed for insertion, it either lies in the face or encroaches a collar segment. Thus whenever a triangle is processed

to be split, some action is taken and encroached faces are never forgotten. At termination, all faces must conform to the input.

Another proposition used in this step comes from [4].

**Proposition 3.** *[4, Lemma 4.5] Let $\mathcal{T}$ be the Delaunay triangulation of a planar face $F$ such that the circumball of each bounding segment is empty. Let $\mathcal{B}_1$ be the union of the circumballs of all bounding segments of $F$ and $\mathcal{B}_2$ be the union of the circumballs of all triangles in $F$.*

*Let $p \in F \setminus \mathcal{B}_1$. If $\mathcal{T}'$ is the Delaunay triangulation of the face resulting from the addition of $p$, define $\mathcal{B}'_1$ and $\mathcal{B}'_2$ to be the union of the circumballs of bounding segments and triangles, respectively. Then,*

$$\mathcal{B}'_1 \cup \mathcal{B}'_2 \subset \mathcal{B}_1 \cup \mathcal{B}_2$$

During this step of the algorithm, when a circumcenter is inserted it does not encroach any collar simplices. The above proposition ensures that as the collar is split the new collar simplices are not encroached by previously added circumcenters. Observe that once the collar is added the circumball of each segment always remains empty. Thus the conditions of Proposition 3 hold for every face in the mesh.

**Lemma 8.** *Whenever the queue of collar segments to be split is empty, the following properties hold.*

- *The circumball of any collar element contains no points in $\mathcal{P}'$.*
- *Adjacent collar segments meet at non-acute angles.*
- *In each face, the protecting disk of each collar segment contains no points in $\mathcal{P}'$.*

In the final property, the term protecting disk is used, rather than diametral disk. For the initial collar segments this protecting disk is the diametral disk. When a collar segment is split, the protecting disk for a new collar segment is the smallest disk containing the new collar segments and contained in the old protecting disk. This disk is not the diametral disk when collar segments corresponding to input points are split. In this case, the split point is added on the circle around the associated input point, and the diametral disks of the new collar segments are not contained in the old protecting disk. See Figure 7.

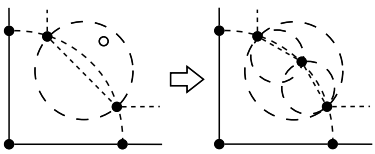

**Fig. 7.** A collar segment associated with an input point is split. When a proposed point (denotd by the empty dot) encroaches a collar segment, the segment is split by adding the midpoint of the arc of a circle around the input point. The new collar segments are protected with disks which are contained in the old protecting disk.

Also, Lemma 8 only applies when the queue of collar segments to be split is empty. Collar segments can be queued when a collar point on an adjacent face encroaches some collar simplex. When this occurs, the circumball of some collar simplices are not empty. However, once all collar segments have been split on adjacent faces, the circumball of each collar simplex is empty and thus the collar simplices conform to the input.

## 5 Implementation Details and Examples

When implementing the algorithm, several minor enhancements have been made which greatly reduce the output mesh size. Since the collar is needed to protect sharp input angles, it only needs to be inserted near segments and input points which violate the standard requirements for the termination of Ruppert's algorithm. (Any acute angle between adjacent faces is protected with a collar. For angles involving segments, this bound can be weakened as stated in [4].) In some examples, this greatly reduces the number of points which must be added to the mesh in Step 2c and thus the size of the final result.

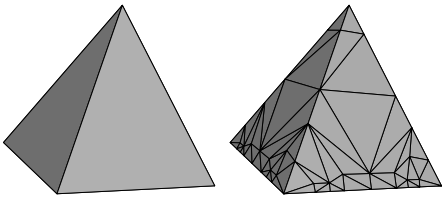

**Fig. 8.** Initial PLC input and final refined mesh for the pyramid example

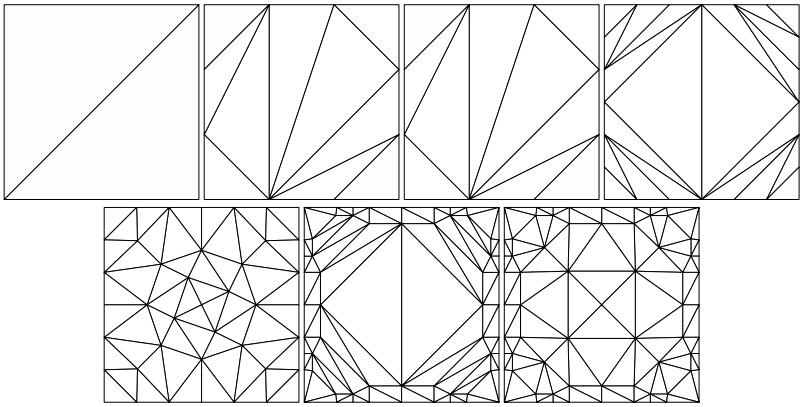

**Fig. 9.** Refinement of the base of the pyramid after each step, showing the mesh of the face after Steps 0, 1a, 1b, 2a, 2b, 2c, 3

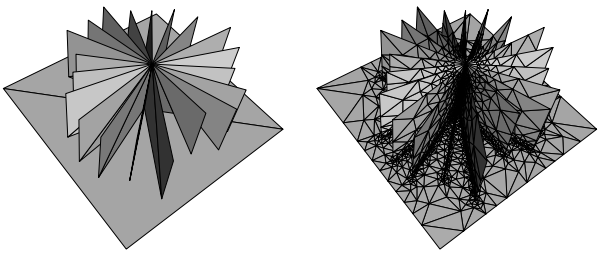

**Fig. 10.** Initial PLC input and final refined mesh for the wheel example. The base face is not connected to the other faces.

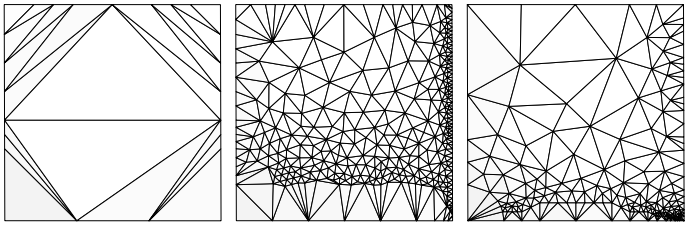

**Fig. 11.** One spoke in the wheel example is shown following Step 1b (left), Step 2b (center), and Step 3 (right). The center of the wheel is at the bottom of the face and the disjoint, nearby plane is very close to the right side of the face.

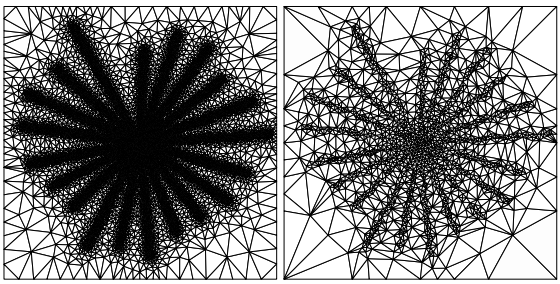

**Fig. 12.** Base plane in the wheel example is shown following Step 2b (left) and Step 3 (right)

Next, instead of adding the collar to the mesh produced by Step 2b, it is instead added to a blank copy of the input PLC. Since the encroachment property in Step 2b is stronger than that in Step 3, this often results in a mesh which is less refined on the faces than the mesh produced in Step 2b. This is especially effective when using a large radius-edge threshold.

Finally, while the Lemma 5 requires each segment in the mesh to be divided in to fourths, the implementation has been seen to terminate when splitting only in

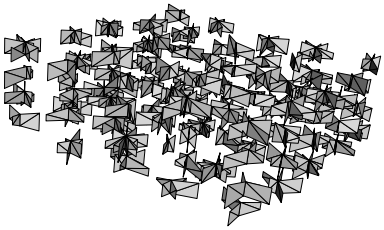

**Fig. 13.** A PLC with many small angles

**Table 1.** Number of vertices in the mesh following each step. Following Step 0, the mesh contains the 1556 input points plus 26 points for the initial bounding box. Step 3 was performed with several maximum radius-edge parameters and the final mesh size is given for each of these cases with the parameter in parentheses.

| Step | 0 | 1a | 1b | 2a | 2b | 2c | 3 (2) | 3 (4) | 3 (8) |
|---|---|---|---|---|---|---|---|---|---|
| Vertices | 1,582 | 5,850 | 13,590 | 26,628 | 105,781 | 13,695 | 76,522 | 54,392 | 46,776 |

half. We continue to seek a example causing non-termination or stronger proof which justifies this practical modification.

The first example is a mesh of a pyramid with a square base. The angles between the base face and each of the side faces is less than 90°, so each of the edges around the base is protected with a collar. See Figures 8 and 9.

Another example involves a "wheel" of twenty rectangular faces which meet at a single segment. This wheel of faces lies very close to an additional disjoint face. In this example, the collar is only added around the segment in the center of the wheel. Away from this center segment all of the faces and segments meet at non-acute angles and do not need to be protected. See Figures 10, 11, and 12.

In the final example, depicted in Figure 13, 100 "wheel" complexes similar to the previous example are meshed together. Each wheel contains in this complex contains between 3 and 16 spokes. The input contains 1556 vertices, 2134 segments and 678 faces. Table 1 contains a list of mesh sizes after each step of the algorithm. Note that the final mesh is substantially smaller than the mesh created in Step 2b. Removing unneeded points before the encroachment criteria is relaxed in Step 3 greatly reduces the output size.

## References

1. Cheng, S.-W., Dey, T.K., Levine, J.A.: A practical Delaunay meshing algorithm for a large class of domains. In: Proceedings of the 16th International Meshing Roundtable (October 2007)
2. Cheng, S.-W., Poon, S.-H.: Three-dimensional Delaunay mesh generation. Discrete Comput. Geom. 36(3), 419–456 (2006)
3. Cohen-Steiner, D., de Verdière, E.C., Yvinec, M.: Conforming Delaunay triangulations in 3D. Comput. Geom. Theory Appl. 28(2-3), 217–233 (2004)

4. Miller, G.L., Pav, S.E., Walkington, N.J.: An incremental Delaunay meshing algorithm. Technical Report 02-CNA-023, Center for Nonlinear Analysis, Carnegie Mellon University, Pittsburgh, Pennsylvania (2002)
5. Miller, G.L., Pav, S.E., Walkington, N.J.: When and why Ruppert's algorithm works. In: Proceedings of the 12th International Meshing Roundtable, pp. 91–102 (September 2003)
6. Murphy, M., Mount, D.M., Gable, C.W.: A point-placement strategy for conforming Delaunay tetrahedralization. International Journal of Computational Geometry and Applications 11(6), 669–682 (2001)
7. Pav, S.E., Walkington, N.J.: Robust three dimensional Delaunay refinement. In: Proceedings of the 13th International Meshing Roundtable (September 2004)
8. Pav, S.E., Walkington, N.J.: Delaunay refinement by corner lopping. In: Proceedings of the 14th International Meshing Roundtable (September 2005)
9. Ruppert, J.: A Delaunay refinement algorithm for quality 2-dimensional mesh generation. J. Algorithms 18(3), 548–585 (1995)
10. Shewchuk, J.R.: Mesh generation for domains with small angles. In: Symposium on Computational Geometry, pp. 1–10 (2000)
11. Si, H.: On refinement of constrained Delaunay tetrahedralizations. In: Proceedings of the 15th International Meshing Roundtable (September 2006)
12. Si, H., Gartner, K.: Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In: Proceedings of the 14th International Meshing Roundtable (September 2005)