Tetrahedral Mesh Generation by Delaunay Refinement

Jonathan Richard Shewchuk
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
jrs@cs.cmu.edu

Abstract

Given a complex of vertices, constraining segments, and planar straight-line constraining facets in E^3, with no input angle less than 90°, an algorithm presented herein can generate a conforming mesh of Delaunay tetrahedra whose circumradius-to-shortest edge ratios are no greater than two. The sizes of the tetrahedra can provably grade from small to large over a relatively short distance. An implementation demonstrates that the algorithm generates excellent meshes, generally surpassing the theoretical bounds, and is effective in eliminating tetrahedra with small or large dihedral angles, although they are not all covered by the theoretical guarantee.

1 Introduction

Meshes of triangles or tetrahedra have many applications, including interpolation, rendering, and numerical methods such as the finite element method. Most such applications demand more than just a triangulation of the object or domain being rendered or simulated. To ensure accurate results, the triangles or tetrahedra must be "well-shaped," having small aspect ratios or bounds on their smallest and largest angles.

Mesh generation algorithms based on *Delaunay refinement* are effective both in theory and in practice. Delaunay refinement algorithms operate by maintaining a Delaunay or constrained Delaunay triangulation, which is refined by inserting carefully placed vertices until the mesh meets constraints on triangle quality and size.

It is difficult to trace who first used Delaunay triangulations for finite element meshing, and equally difficult to tell where the suggestion arose to use the triangulation to guide vertex creation. These ideas have been intensively studied in the engineering community since the mid-1980s; for instance, Frey [7] eliminates poorly shaped triangles from a triangulation by inserting new vertices at their circumcenters (defined in Section 2), whereas Weatherill [17] inserts new vertices at their centroids.

These ideas bore vital theoretical fruit when the problem of mesh generation began to attract interest from the computational geometry community in the early 1990s. The first provably good Delaunay refinement algorithm is due to Paul Chew [2], and takes as its input a set of vertices and segments that define the region to be meshed. By inserting additional vertices, Chew's algorithm generates a two-dimensional constrained Delaunay triangulation whose angles are bounded between 30° and 120°. Dey, Bajaj, and Sugihara [5] and Chew [4] generalize Chew's algorithm to three dimensions, but only for unconstrained point set inputs. All three algorithms produce *uniform* meshes, whose triangles or tetrahedra are of roughly the same size.

Uniform meshes sometimes have many more triangles or tetrahedra than are necessary, and thus impose an excessive computational load upon the applications that make use of them. Jim Ruppert [13] and Paul Chew [3] have each proposed two-dimensional Delaunay refinement algorithms that produce meshes of well-shaped triangles whose sizes are graded, and Ruppert has furthermore proven that his algorithm produces meshes that are nicely graded in a theoretical sense described in Section 7.

These algorithms are successful because they exploit several favorable characteristics of Delaunay triangulations. Delaunay triangulations have been extensively studied, and good algorithms are available. Inserting a vertex is a local operation, and is inexpensive except in unusual cases. In two dimensions, Delaunay triangulations maximize the minimum angle, compared with all other triangulations of the same vertex set [8].

The greatest advantage of Delaunay triangulations is less obvious. The central question of any Delaunay refinement algorithm is "where should the next vertex be inserted?" As Section 3 will demonstrate, a reasonable answer is "as far from other vertices as possible." If a new vertex is inserted too close to another vertex, the resulting short edge will engender thin triangles or tetrahedra.

Because a Delaunay triangle has no vertices in its circumcircle (and a Delaunay tetrahedron has no vertices in its circumsphere), a Delaunay triangulation is an ideal search structure for finding points that are far from other vertices.

Herein, I build upon the algorithmic and analytical framework of Ruppert to design a new tetrahedral Delaunay refinement algorithm. This algorithm generates meshes whose tetrahedra have circumradius-to-shortest edge ratios (defined shortly) no greater than the bound $B = 2$. Upon relaxing B to be greater than two, one can also guarantee good grading. My algorithm is distinguished from those of Dey et al. and Chew by this guarantee of good grading (as opposed to uniform meshes), and by its ability to handle the rather general constrained inputs described in Section 4 (as opposed to unconstrained sets of vertices). The main theoretical deficiency of the algorithm is its requirement that incident input segments and facets be separated by angles of at least 90°. (The requirements are less onerous in practice.) This deficiency can be partly ameliorated; Section 9 includes a brief summary of the steps involved.

Supported in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF under agreement number F30602-96-1-0287, in part by the National Science Foundation under Grant CMS-9318163, and in part by a grant from the Intel Corporation.

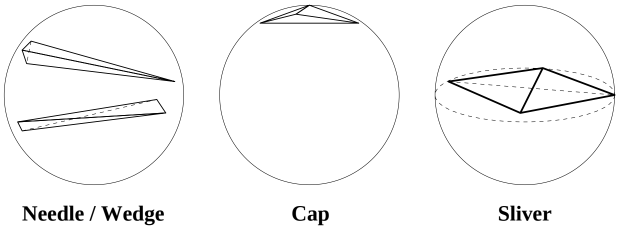


Figure 1: Tetrahedra with poor angles. Needles and wedges have edges of greatly disparate length; caps have a large solid angle; slivers have neither, but can have good circumradius-to-shortest edge ratios.

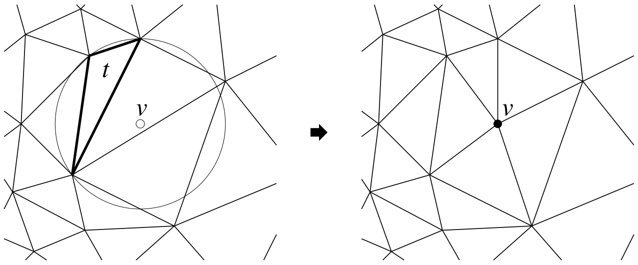


Figure 2: Any triangle whose circumradius-to-shortest edge ratio is larger than some bound B is split by inserting a vertex at its circumcenter. The Delaunay property is maintained, and the triangle is thus eliminated. Every new edge has length at least B times that of shortest edge of the poor triangle.

Two other tetrahedral mesh generation algorithms (not based on Delaunay refinement) have provable bounds. The octree-based algorithm of Mitchell and Vavasis [11, 12] is a theoretical *tour de force*, obtaining provable bounds on dihedral angles and grading, but its bounds are too weak to offer any practical reassurance (and have not been explicitly stated). An algorithm by Miller, Talmor, Teng, Walkington, and Wang [10] generates its final vertex set before triangulating it. Their algorithm has provable bounds similar to those of the algorithm described herein, but the algorithm herein has the opportunity to stop early if fewer vertices are needed than the theory suggests. As a result, the present algorithm typically uses fewer vertices by several orders of magnitude; details are provided elsewhere [14].

2 A Quality Measure for Simplices

Miller, Talmor, Teng, and Walkington [9] have pointed out that the most natural and elegant measure for analyzing Delaunay refinement algorithms is the *circumradius-to-shortest edge ratio* of a triangle or tetrahedron. The *circumsphere* of a simplex is the unique circle or sphere that passes through all its vertices. The *circumcenter* and *circumradius* of a simplex are the center and radius of its circumsphere, respectively. The quotient of a simplex's circumradius and the length of its shortest edge is the metric that is naturally optimized by Delaunay refinement algorithms. One would like this ratio to be as small as possible.

But does optimizing this metric aid practical applications? In two dimensions, the answer is yes. A triangle's circumradius-to-shortest edge ratio r/ℓ is related to its smallest angle $\theta_{\min}$ by the formula $r/\ell = 1/(2\sin\theta_{\min})$. The smaller a triangle's ratio, the larger its smallest angle. Chew's two-dimensional algorithms produce meshes whose triangles' circumradius-to-shortest edge ratios are bounded below one, and hence their angles are bounded between 30° and 120°. Ruppert's algorithm can produce meshes whose triangles' ratios are bounded below $\sqrt{2}$, and hence their angles range between 20.7° and 138.6°.

In three dimensions, however, a mesh of tetrahedra whose circumradius-to-shortest edge ratios are bounded is not entirely adequate for the needs of interpolation. As Dey et al. illustrate, most tetrahedra with poor angles have circumcircles much larger than their shortest edges, including the needle, wedge, and cap illustrated in Figure 1, but there is one type called a *sliver* or *kite* tetrahedron that does not. The canonical sliver is formed by arranging four vertices, equally spaced, around the equator of a sphere, then perturbing one of the vertices slightly off the equator. A sliver can have a circumradius-to-shortest edge ratio as low as $1/\sqrt{2}$, yet be considered awful by most other measures, because its volume and its shortest altitude can be arbitrarily close to zero, and its dihedral angles can be arbitrarily close to 0° and 180°.

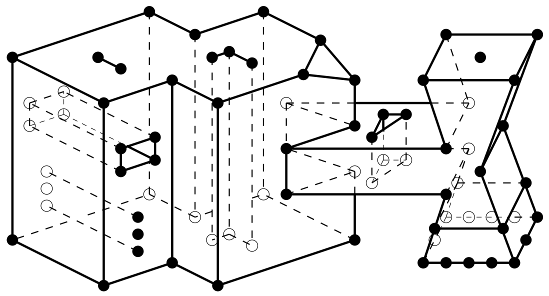
Despite slivers, Delaunay refinement methods are valuable for generating three-dimensional meshes. Slivers having good circumradius-to-shortest edge ratios typically arise in small numbers in practice. As Section 8 will demonstrate, the worst slivers can often be removed by Delaunay refinement, even if there is no theoretical guarantee. Meshes with bounds on the circumradius-to-shortest edge ratios of their tetrahedra are an excellent starting point for mesh smoothing and optimization methods that remove slivers and otherwise improve the quality of an existing mesh [6]. Even if slivers are not removed, the Voronoi dual of a tetrahedralization with bounded circumradius-to-shortest edge ratios has nicely rounded cells, and is sometimes ideal for use in the control volume method [9].

3 The Key Idea Behind Delaunay Refinement

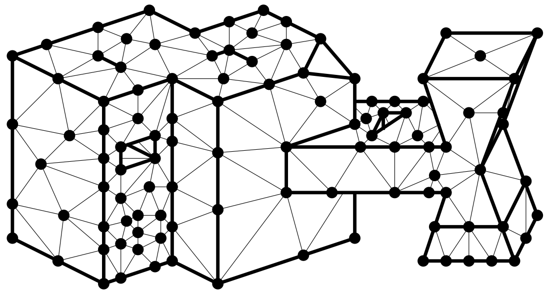
The central operation of Chew's, Ruppert's, Dey's, and my own Delaunay refinement algorithms is the insertion of a vertex at the circumcenter of a triangle or tetrahedron of poor quality. The Delaunay property is maintained, perhaps by the Bowyer/Watson algorithm for the incremental update of Delaunay triangulations [1, 16]. The poor simplex cannot survive, because its circumsphere is no longer empty. For brevity, the act of inserting a vertex at a simplex's circumcenter is called *splitting* a simplex. If poor simplices are split one by one, either all will eventually be eliminated, or the algorithm will run forever.

The main insight of all these Delaunay refinement algorithms is that Delaunay refinement is guaranteed to terminate if the notion of "poor quality" includes only simplices that have a circumradius-to-shortest edge ratio larger than some appropriate bound B. The only new edges created by the Delaunay insertion of a vertex v are edges connected to v (see Figure 2). Because v is the circumcenter of some Delaunay simplex t, and there were no vertices inside the circumsphere of t before v was inserted, no new edge can be shorter than the circumradius of t. Because t has a circumradius-to-shortest edge ratio larger than B, every new edge has length at least B times that of the shortest edge of t.

Ruppert's Delaunay refinement algorithm [13] employs a bound of $B = \sqrt{2}$, and Chew's second Delaunay refinement algorithm [3] employs a bound of $B = 1$. Chew's first Delaunay refinement algorithm [2] splits any triangle whose circumradius is greater than the length of the shortest edge in the entire mesh, thus achieving a bound of $B = 1$, but forcing all triangles to have uniform size. In the same manner, the three-dimensional algorithms of Dey et al. and Chew achieve a bound of $B = 2$. With these bounds, every new edge created is at least as long as some other edge already in the mesh. Hence, no vertex is ever inserted closer to another vertex than the length of the shortest edge in the initial triangulation. Delaunay refinement must eventually terminate, because the augmented triangulation will run out of places to put vertices.



(a)



(b)

Figure 3: (a) Any facet of a PLC may contain holes, slits, and vertices; these may support intersections with other facets and segments or allow a user of the finite element method to apply boundary conditions. (b) When a PLC is tetrahedralized, each facet of the PLC is partitioned into triangular subfacets, which respect the holes, slits, and vertices.

This idea generalizes without change to higher dimensions. Unfortunately, my description of Delaunay refinement thus far has a gaping hole: mesh boundaries have not been accounted for. The flaw in the procedure presented above is that the circumcenter of a poor simplex might not lie in the mesh at all. Delaunay refinement algorithms, including those of Chew, Ruppert, and Dey et al., are distinguished primarily by how they handle boundaries.

4 Piecewise Linear Complexes

The input upon which my three-dimensional Delaunay refinement algorithm operates is called a *piecewise linear complex* (PLC). See Miller, Talmor, Teng, Walkington, and Wang [10] for a definition in E^d. In three dimensions, a PLC is a set of vertices, segments, and facets. A segment is a constraining edge that must be represented by a sequence of contiguous edges in the final mesh. A facet is a constraining planar surface that must be represented by a set of triangular faces in the final mesh. A facet can be quite complicated in shape; it is a polygon (not necessarily convex), possibly augmented by holes, slits, and vertices in its interior. Figure 3(a) demonstrates some of the possibilities.

A piecewise linear complex X is required to have the following properties. First, X contains both endpoints of each segment of X. Similarly, facets of X are *segment-bounded*: for any facet in X, every edge and vertex of the facet must be a segment or vertex of X.

Second, X is closed under intersection. For example, if two facets of X intersect at a line segment, that line segment must be a segment of X. Third, if a segment of X intersects a facet of X at a point in the segment's interior, then the segment must be entirely contained in the facet.

The process of tetrahedral mesh generation necessarily divides each segment into smaller edges called *subsegments*. (Each segment is initially represented by one subsegment, until it is subdivided.) The bold edges of the tetrahedralization illustrated in Figure 3(b) are subsegments; other edges are not. Similarly, each facet is subdivided into triangular faces called *subfacets*. All of the triangular faces visible in Figure 3(b) are subfacets, but most of the faces in the interior of the tetrahedralization are not.

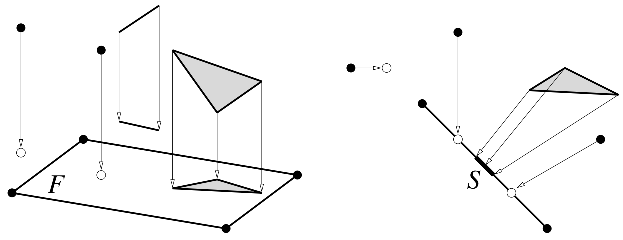


Figure 4: The orthogonal projections of points and sets of points onto facets and segments.

Any vertex inserted into a segment or facet during Delaunay refinement remains there permanently. However, keep in mind that the edges that partition a facet into subfacets are *not* permanent, are *not* treated like subsegments, and are subject to flipping according to the Delaunay criterion.

Many approaches to tetrahedral mesh generation permanently triangulate the input facets as a separate step prior to tetrahedralizing the interior of a region. The problem with this approach is that these independent facet triangulations may not be collectively ideal for forming a good tetrahedralization. The algorithm discussed herein uses an alternative approach, wherein facet triangulations are refined in conjunction with the tetrahedralization. The tetrahedralization process is not beholden to poor decisions made earlier.

5 Orthogonal Projections

The following sections use the notion of the *orthogonal projection* of a geometric entity onto a line or plane. Given a facet or subfacet F and a point p, the orthogonal projection $\text{proj}_F(p)$ of p onto F is the point that is coplanar with F and lies in the line that passes through p orthogonally to F, as illustrated in Figure 4. The projection exists whether or not it falls in F.

Similarly, the orthogonal projection $\text{proj}_S(p)$ of p onto a segment or subsegment S is the point that is collinear with S and lies in the plane through p orthogonal to S.

Sets of points may be projected as well. If F and G are facets, then $\text{proj}_F(G)$ is the set $\{\text{proj}_F(p) : p \in G\}$.

6 A 3D Delaunay Refinement Algorithm

In this section, I describe a three-dimensional Delaunay refinement algorithm that produces well-graded tetrahedral meshes satisfying any circumradius-to-shortest edge ratio bound greater than two.

The algorithm takes a *facet-bounded* PLC as its input. Tetrahedralized and untetrahedralized regions of space must be separated by facets so that, in the final mesh, any triangular face not shared by two tetrahedra is a subfacet.

The first step is to form a Delaunay tetrahedralization of the input vertices. Some input segments and facets might be missing (or partly missing) from this mesh. The tetrahedralization is refined by inserting additional vertices into the mesh, using the Bowyer/Watson algorithm to maintain the Delaunay property, until all segments and facets are respected and all constraints on tetrahedron quality are met. Vertex insertion is governed by three rules.

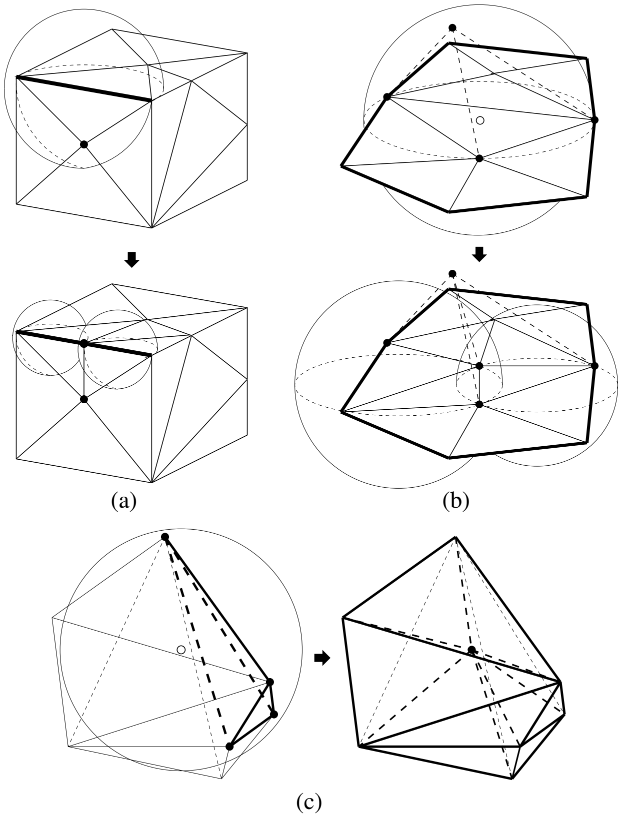


Figure 5: Three operations for three-dimensional Delaunay refinement. (a) Splitting an encroached subsegment. The original subsegment is encroached because there is a vertex in its diametral sphere. In this example, the two subsegments created by bisecting the original subsegment are not encroached. (b) Splitting an encroached subfacet. The triangular faces shown are subfacets of a larger facet, with tetrahedra (not shown) atop them. In this example, all equatorial spheres (included the two illustrated) are empty after the split. (c) Splitting a skinny tetrahedron.

- The *diametral sphere* of a subsegment is the (unique) smallest sphere that encloses the subsegment. Following Ruppert [13], a subsegment is said to be *encroached* if a vertex other than its endpoints lies inside or on its diametral sphere. A subsegment may be encroached whether or not it actually appears as an edge of the tetrahedralization. It is a property of Delaunay tetrahedralizations that if a subsegment is missing from the tetrahedralization, then it is encroached. Any encroached subsegment that arises is immediately split into two subsegments by inserting a vertex at its midpoint, as illustrated in Figure 5(a). These subsegments may or may not be encroached themselves; splitting continues until no subsegment is encroached.

- The *equatorial sphere* of a triangular subfacet is the (unique) smallest sphere that passes through the three vertices of the subfacet. A subfacet is encroached if a non-coplanar vertex lies inside or on its equatorial sphere. It is a property of Delaunay tetrahedralizations that if a subfacet does not appear in the tetrahedralization, and it is not covered by other faces that share the same equatorial sphere, then it is encroached. (The question of what subfacets should not be missing from the mesh will be considered shortly.) Each encroached subfacet is normally split by inserting a vertex at its circumcenter; see Figure 5(b). However, if the new vertex would encroach upon any subsegment, it is not inserted; instead, all the subsegments it would encroach upon are split.

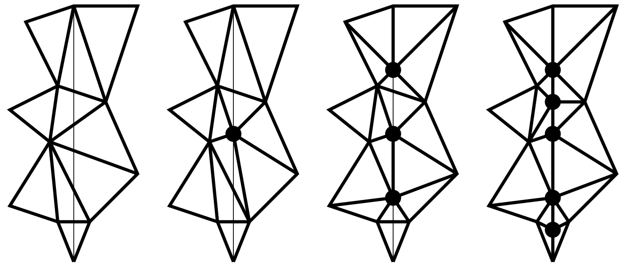


Figure 6: Missing segments are recovered through the same recursive splitting procedure used for encroached subsegments that aren't missing. In this sequence of illustrations, the thin line represents a segment missing from the triangulation. Segment recovery is illustrated here in two dimensions for clarity, but operates no differently in three.

- A tetrahedron is said to be *skinny* if its circumradius-to-shortest edge ratio is larger than some bound B. Each skinny tetrahedron is normally split by inserting a vertex at its circumcenter, thus eliminating the tetrahedron; see Figure 5(c). However, if the new vertex would encroach upon any subsegment or subfacet, then it is not inserted; instead, all the subsegments it would encroach upon are split. If the skinny tetrahedron is not eliminated as a result, then all the subfacets its circumcenter would encroach upon are split.

Encroached subsegments are given priority over encroached subfacets, which have priority over skinny tetrahedra. These encroachment rules are intended to recover missing segments and facets, and to ensure that all vertex insertions are valid. Although the proof is omitted here, one can show that if there are no encroached subsegments, then each subfacet circumcenter lies in the containing facet; and if there are no encroached subfacets, then each tetrahedron circumcenter lies in the mesh.

Because encroached subsegments have priority, the algorithm begins by recovering all the segments that are missing from the initial tetrahedralization. Each missing segment is bisected by inserting a vertex into the mesh at the midpoint of the segment (more accurately, at the midpoint of the place where the segment should be). After the mesh is adjusted to maintain the Delaunay property, the two resulting subsegments may appear in the mesh. If not, the procedure is repeated recursively for each missing subsegment until the original segment is represented by a contiguous linear sequence of edges of the mesh, as illustrated (in two dimensions) in Figure 6. We are assured of eventual success because the Delaunay triangulation always connects a vertex to its nearest neighbor; once the spacing of vertices along a segment is sufficiently small, its entire length will be represented. In the engineering literature, this process is sometimes called *stitching*.

When no encroached subsegment remains, missing facets are recovered in an analogous manner. The main complication is that if a facet is missing from the mesh, it is difficult to say what its subfacets are. With segments there is no such problem; if a segment is missing from the mesh, and a vertex is inserted at its midpoint, one knows unambiguously where the two resulting subsegments are. But how may we identify subfacets that do not yet exist?

The solution is straightforward. For each facet, it is necessary to maintain a two-dimensional Delaunay triangulation of its vertices, independently from the tetrahedralization in which we hope its subfacets will eventually appear. By comparing the triangles of a facet's triangulation against the faces of the tetrahedralization, one can identify subfacets that need help in forcing their way into

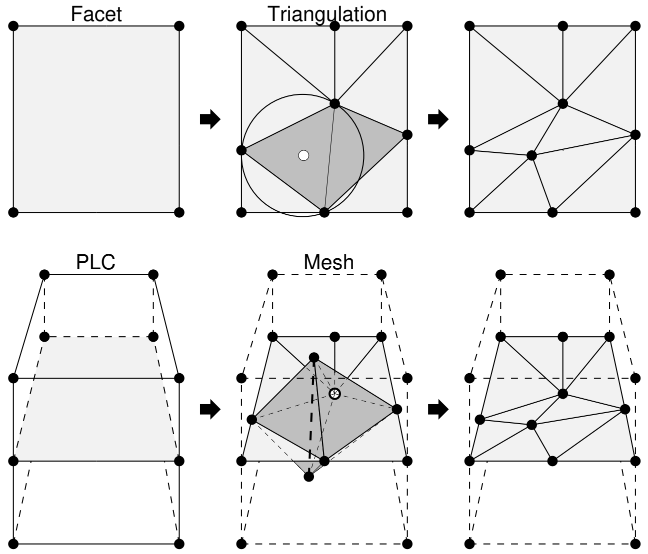


Figure 7: The top illustrations depict a rectangular facet and its triangulation. The bottom illustrations depict the facet's position as an interior boundary of a PLC, and its progress as it is inserted into the tetrahedralization. Most of the vertices and tetrahedra of the mesh are omitted for clarity. The facet triangulation and the tetrahedralization are maintained separately. Shaded triangular subfacets in the facet triangulation (top center) are missing from the tetrahedralization (bottom center). The bold dashed line (bottom center) represents a tetrahedralization edge that passes through the facet. A missing subfacet is forced into the mesh by inserting a vertex at its circumcenter (top right and bottom right). The vertex is independently inserted into both the triangulation and the tetrahedralization.

the mesh. For each triangular subfacet in a facet triangulation, look for a matching face in the tetrahedralization; if the latter is missing, insert a vertex at the circumcenter of the subfacet (subject to rejection if subsegments are encroached), as illustrated in Figure 7. The new vertex is independently inserted into both the facet triangulation and the tetrahedralization. Similarly, the midpoint of an encroached subsegment is independently inserted into the tetrahedralization and into *each* facet triangulation that contains the subsegment.

In essence, the same procedure is used to recover missing segments. However, the process of forming a one-dimensional triangulation is so simple that it passes unnoticed.

Which vertices of the tetrahedralization need to be considered in a facet triangulation? If a facet appears in a Delaunay tetrahedralization as a union of faces, then the triangulation of the facet is determined solely by the vertices of the tetrahedralization that lie in the plane of the facet. If a vertex lies near a facet, but is not coplanar with the facet, it may cause a subfacet to be missing (as in Figure 7, bottom center), but it cannot otherwise affect the shape of the triangulation. If a facet appears as a union of faces in a Delaunay tetrahedralization, then those faces form a two-dimensional Delaunay triangulation of the facet.

Furthermore, because each facet is segment-bounded, and segments are recovered (in the tetrahedralization) before facets, each facet triangulation can safely ignore vertices that lie outside the facet (coplanar though they may be). The requirements set forth in Section 4 ensure that all of the vertices and segments of a facet must be explicitly identified in the input PLC. The only additional vertices to be considered are those that were inserted in segments to help recover those segments and other facets. The algorithm maintains a list of the vertices on each segment, ready to be called upon when a facet triangulation is initially formed.

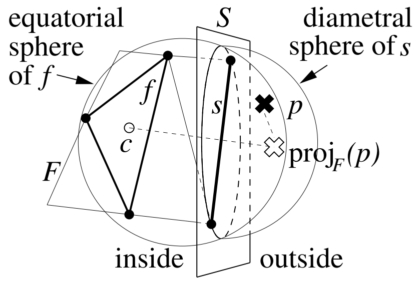


Figure 8: If a vertex p encroaches upon a Delaunay subfacet f of a facet F, but its projection into the plane containing F lies outside F, then p encroaches upon some subsegment s of F as well.

Unfortunately, if a facet's Delaunay triangulation is not unique because of cocircularity degeneracies, then the facet might be represented in the tetrahedralization by faces that do not match the independent facet triangulation. An implementation must detect these cases and correct the triangulation so that it matches the tetrahedralization.

When no encroached subsegment or subfacet remains, every input segment and facet is represented by a union of edges or faces of the mesh. The first time the mesh reaches this state, all external tetrahedra (lying in the convex hull of the input vertices, but outside the region enclosed by the facet-bounded PLC) are removed prior to splitting any skinny tetrahedra. This measure prevents problems that might arise if superfluous skinny tetrahedra are split, such as overrefinement and failure to terminate because of spurious small angles formed between the PLC and its convex hull.

One further amendment to the algorithm is necessary to obtain the best possible bound on the circumradius-to-shortest edge ratios of the tetrahedra. When several encroached subfacets exist, they should not be split in arbitrary order. If a vertex p encroaches upon a subfacet f of a facet F, but the projected point $\mathrm{proj}_F(p)$ does not lie in f, then splitting f is not the best choice. One can show (with the following lemma) that there is some other subfacet g of F that is encroached upon by p and contains $\mathrm{proj}_F(p)$. (The lemma assumes that there are no encroached subsegments in the mesh, as they have priority.) A better bound is achieved if the algorithm splits g first and delays the splitting of f indefinitely.

Lemma 1 (Projection Lemma) *Let f be a subfacet of the Delaunay triangulated facet F. Suppose that f is encroached upon by some vertex p, but p does not encroach upon any subsegment of F. Then $\mathrm{proj}_F(p)$ lies in the facet F, and p encroaches upon a subfacet of F that contains $\mathrm{proj}_F(p)$.*

Proof: First, I prove that $\mathrm{proj}_F(p)$ lies in F. Suppose for the sake of contradiction that $\mathrm{proj}_F(p)$ lies outside the facet F. Let c be the centroid of f; c clearly lies inside F. Because all facets are segment-bounded, the line segment connecting c to $\mathrm{proj}_F(p)$ must intersect some subsegment s in the boundary of F. Let $\mathcal{S}$ be the plane that contains s and is orthogonal to F, as illustrated in Figure 8.

Because f is a Delaunay subfacet of F, its circumcircle (in the plane containing F) encloses no vertex of F. However, its equatorial sphere may enclose vertices—including p—and f might not appear in the tetrahedralization.

It is apparent that p and $\mathrm{proj}_F(p)$ lie on the same side of $\mathcal{S}$, because the projection is defined orthogonally to F. Say that a point is *inside* $\mathcal{S}$ if it is on the same side of $\mathcal{S}$ as c, and *outside* $\mathcal{S}$ if it is on the same side as p and $\mathrm{proj}_F(p)$. The circumcircle of f cannot enclose the endpoints of s, because f is Delaunay in F. Furthermore, the circumcenter of f lies in F; otherwise, a vertex of f would encroach upon some boundary segment of F. It follows that

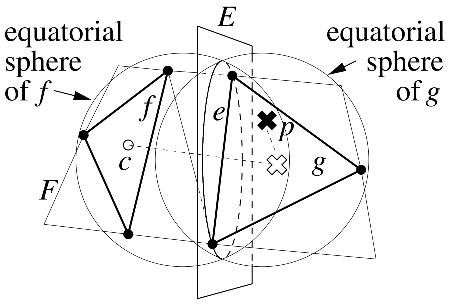


Figure 9: If a vertex p encroaches upon a subfacet f of a Delaunay triangulated facet F, but does not encroach upon any subsegment of F, then p encroaches upon the subfacet(s) g of F that contains $\mathrm{proj}_F(p)$.

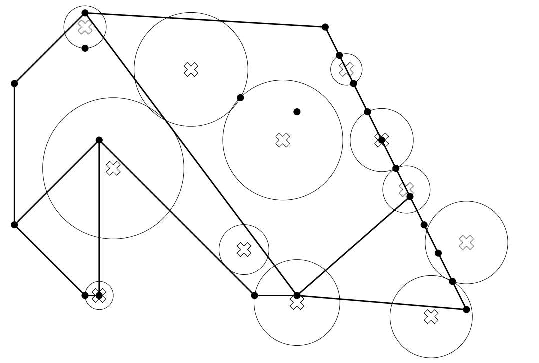


Figure 10: The radius of each disk illustrated is the local feature size of the point at its center.

the portion of f's equatorial sphere outside $\mathcal{S}$ lies entirely inside or on the diametral sphere of s (as the figure demonstrates). Because p is inside or on the equatorial sphere of f, p also lies inside or on the diametral sphere of s, contradicting the assumption that p encroaches upon no subsegment of F.

It follows that $\mathrm{proj}_F(p)$ must be contained in some subfacet g of F. (The containment is not necessarily strict; $\mathrm{proj}_F(p)$ may fall on an edge interior to F, and be contained in two subfacets.) To complete the proof of the lemma, I shall show that p encroaches upon g. If $f = g$ the result follows immediately, so assume that $f \neq g$.

Again, let c be the centroid of f. The line segment connecting c to $\mathrm{proj}_F(p)$ must intersect some edge e of the subfacet g, as illustrated in Figure 9. Let $\mathcal{E}$ be the plane that contains e and is orthogonal to F. Say that a point is on the *g-side* if it is on the same side of $\mathcal{E}$ as g. Because the triangulation of F is Delaunay, the portion of f's equatorial sphere on the g-side lies entirely inside or on the equatorial sphere of g. The point p lies on the g-side or in $\mathcal{E}$ (because $\mathrm{proj}_F(p)$ lies in g), and p lies inside or on the equatorial sphere of f, so it must also lie inside or on the equatorial sphere of g, and hence encroaches upon g. ■

If p encroaches upon some subfacet of F but no subfacet of F contains $\mathrm{proj}_F(p)$, then p also encroaches upon some boundary subsegment of F. Because encroached subsegments have priority, subsegments encroached upon by p are split until none remains. The Projection Lemma guarantees that any subfacets of F that were encroached upon by p are eliminated in the process.

On the other hand, if $\mathrm{proj}_F(p)$ lies in a subfacet g of F, and no subsegment is encroached, then splitting g is a good choice. As a result, several new subfacets will appear, at least one of which contains $\mathrm{proj}_F(p)$; if this subfacet is encroached, then it is split as well, and so forth until the subfacet containing $\mathrm{proj}_F(p)$ is not encroached. The Projection Lemma guarantees that any other subfacets of F that were encroached upon by p are eliminated in the process.

7 Proof of Termination and Good Grading

The proof of termination for this algorithm is related to that of Ruppert's. Ruppert's analysis requires that any two incident input segments be separated by an angle of at least $90°$. Similarly, the three-dimensional analysis requires that the input PLC satisfy the following *projection condition*: if two constraints (each being a segment or facet) in X intersect, then the orthogonal projection of either one onto the other cannot intersect the interior of the other. (Here, "interior" is defined to exclude all boundaries, including isolated slits and input vertices in the interior of the facet.) For example, if two convex facets intersect along a segment, they must be separated by a dihedral angle of at least $90°$. Thanks to the Projection Lemma, this rule ensures that no vertex in the interior of one segment or facet can cause the splitting of a subsegment or subfacet in an incident segment or facet, except that a vertex in the interior of a facet may encroach upon a subsegment of that facet. In theory (and more so in practice), the projection condition can be relaxed somewhat, although the analysis becomes too complicated to present here.

Ruppert's algorithm and the three-dimensional algorithm described here produce nicely graded meshes, in the sense that a small feature in one part of a mesh does not unreasonably reduce the edge lengths at other, distant parts of the mesh. To formalize this idea, Ruppert introduces a function called the *local feature size*, defined on all points.

Given a PLC X, the local feature size $\mathrm{lfs}(p)$ at any point p is the radius of the smallest ball centered at p that intersects two nonincident features of X (where each of the two features might be a vertex, segment, or facet). Figure 10 illustrates the notion in two dimensions (with no facets) by giving examples of such balls for a variety of points.

The function $\mathrm{lfs}(\cdot)$ is continuous and has the property that its directional derivatives (where they exist) are bounded in the range $[-1, 1]$, as the following lemma shows. This property sets a lower bound (within a constant factor to be derived shortly) on the rate at which edge lengths grade from small to large as one moves away from a small feature.

Lemma 2 (Ruppert [13]) *For any PLC X, and any two points u and v, $\mathrm{lfs}(v) \leq \mathrm{lfs}(u) + |uv|$.*

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

With each vertex v, associate an *insertion radius* r_v equal to the length of the shortest edge connected to v immediately after v is introduced into the tetrahedralization. Consider what this means in each possible circumstance. If v is an input vertex, then r_v is the Euclidean distance between v and the input vertex nearest v. If v is a vertex inserted to split a subsegment or subfacet, then r_v is the distance between v and the nearest encroaching mesh vertex. If there is no encroaching vertex in the mesh (a vertex was considered for insertion but rejected as encroaching), then r_v is the radius of the diametral/equatorial sphere of the encroached subsegment/subfacet. If v is a vertex inserted at the circumcenter of a skinny tetrahedron, then r_v is the circumradius of the tetrahedron. If a vertex is considered for insertion but rejected because of an encroachment, its insertion radius is defined the same way, even though it is never actually inserted.

Each vertex v, whether inserted or rejected, has a *parent* vertex $p(v)$, unless v is an input vertex. Intuitively, for any non-input vertex v, $p(v)$ is the vertex that is "responsible" for the insertion of v.

If v is a vertex inserted to split a subsegment or subfacet, then $p(v)$ is the encroaching vertex, whether that vertex is inserted or rejected. If there are several encroaching vertices, choose the one nearest v. If v is a vertex inserted (or rejected) at the circumcenter of a skinny tetrahedron, then $p(v)$ is the most recently inserted endpoint of the shortest edge of that tetrahedron.

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

- *$C = B$ if v is the circumcenter of a skinny tetrahedron;*
- *$C = \frac{1}{\sqrt{2}}$ if v is the midpoint of an encroached subsegment or the circumcenter of an encroached subfacet.*

Proof: If v is an input vertex, there is another input vertex a distance of r_v from v, so $\text{lfs}(v) \leq r_v$, and the theorem holds.

If v is inserted in an encroached subsegment or subfacet, and its parent p is an input vertex or lies on another subsegment or subfacet, then there are two cases to consider. The case in which v lies in a boundary segment of a facet that contains p will be considered shortly. Otherwise, v and p must lie on nonincident features (because X satisfies the projection condition), so $\text{lfs}(v) \leq r_v$.

If v is inserted at the circumcenter of a skinny tetrahedron, then its parent $p = p(v)$ is the most recently inserted endpoint of the shortest edge of the tetrahedron; see Figure 11(a). Hence, the length of the shortest edge of the tetrahedron is at least r_p. Because the tetrahedron is skinny, its circumradius-to-shortest edge ratio exceeds B, so its circumradius is $r_v > Br_p$.

If v is inserted at the midpoint of an encroached subsegment s, and its parent p is a tetrahedron/subfacet circumcenter that was considered for insertion but rejected because it encroaches upon s, then p lies inside or on the diametral sphere of s. Because the tetrahedralization/facet triangulation is Delaunay, the circumsphere/circumcircle centered at p encloses no vertices, and in particular does not enclose the endpoints of s. Hence, $r_p \leq \sqrt{2}r_v$; see Figure 11(b) for an example where the relation is equality.

If v is inserted at the circumcenter of an encroached subfacet f, and its parent p is a tetrahedron circumcenter that was considered for insertion but rejected because it encroaches upon f, then p lies inside or on the equatorial sphere of f. Because the tetrahedralization is Delaunay, the circumsphere centered at p encloses no vertices, including the vertices of f. Recall that the algorithm splits f only if f contains $\text{proj}_f(p)$. Given these facts, one can show that $r_p \leq \sqrt{2}r_v$; see Figure 11(c) for an example where the relation is equality. ∎

Lemma 3 limits how quickly the insertion radius can decrease as one traverses a sequence of descendants of a vertex. If vertices with ever-smaller insertion radii cannot be generated, then edges shorter than existing ones cannot be introduced, and Delaunay refinement is guaranteed to terminate. Figure 12 expresses this notion as a dataflow graph. Mesh vertices are divided into four classes: input vertices (which are omitted from the figure because they cannot contribute to cycles), vertices inserted into segments, vertices inserted into facet interiors, and vertices inserted at circumcenters of tetrahedra. Labeled arrows indicate how a vertex can cause the insertion of a child whose insertion radius is some factor times that of its parent. If the graph has no cycle whose product is less than one, termination is guaranteed. This goal is achieved by choosing B to be at least 2.

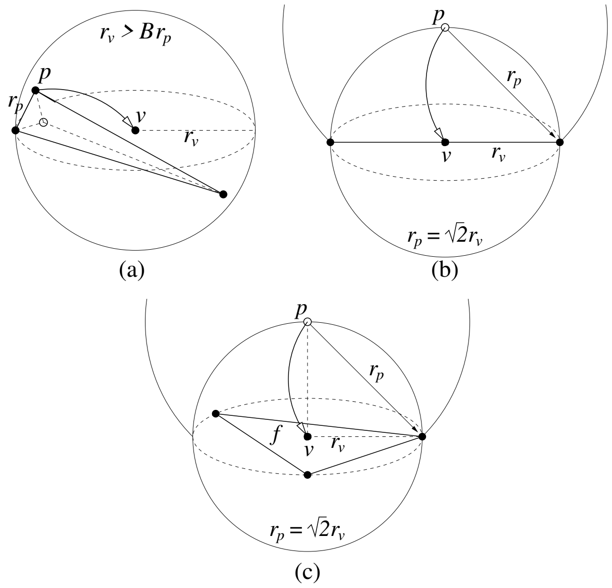


Figure 11: The relationship between the insertion radii of a child and its parent. (a) When a skinny tetrahedron is split, the child's insertion radius is at least B times larger than that of its parent. (b) When a subsegment is encroached upon by a circumcenter, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than its parent's. (c) When a subfacet is encroached upon by the circumcenter of a skinny tetrahedron, the child's insertion radius may be a factor of $\sqrt{2}$ smaller than its parent's.

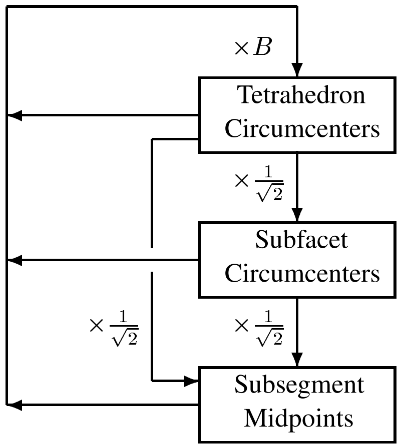


Figure 12: Dataflow diagram illustrating the worst-case relation between a vertex's insertion radius and the insertion radii of the children it begets. If no cycle has a product smaller than one, Delaunay refinement will terminate.

A guarantee of termination alone is not satisfying when a graded mesh is required. What follows is a proof that each edge of the output mesh has length proportional to the local feature sizes of its endpoints. Hence, edge lengths are determined by local considerations; features lying outside the ball that defines the local feature size of a point can only weakly influence the sizes of edges that contain that point.

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

Lemma 4 *Let v be a vertex with parent $p = p(v)$. Suppose that $r_v \geq Cr_p$ (following Lemma 3). Then $D_v \leq 1 + \frac{D_p}{C}$.*

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

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

The result follows by dividing both sides by r_v. ■

Theorem 5 *Suppose the quality bound B is strictly larger than 2, and the input PLC satisfies the projection condition. Then there exist fixed constants $D_T \geq 1$, $D_F \geq 1$, and $D_S \geq 1$ such that, for any vertex v inserted (or rejected) at the circumcenter of a skinny tetrahedron, $D_v \leq D_T$; for any vertex v inserted (or rejected) at the circumcenter of an encroached subfacet, $D_v \leq D_F$; and for any vertex v inserted at the midpoint of an encroached subsegment, $D_v \leq D_S$. Hence, the insertion radius of every vertex has a lower bound proportional to its local feature size.*

Proof: Consider any non-input vertex v with parent $p = p(v)$. If p is an input vertex, then $D_p = \frac{\text{lfs}(p)}{r_p} \leq 1$ by Lemma 3. Otherwise, assume for the sake of induction that the lemma is true for p. Hence, $D_p \leq \max\{D_T, D_F, D_S\}$.

First, suppose v is inserted or considered for insertion at the circumcenter of a skinny tetrahedron. By Lemma 3, $r_v \geq Br_p$. Therefore, by Lemma 4, $D_v \leq 1 + \frac{\max\{D_T, D_F, D_S\}}{B}$. It follows that one can prove that $D_v \leq D_T$ if D_T is chosen sufficiently large that

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

Second, suppose v is inserted or considered for insertion at the circumcenter of a subfacet f. If its parent p is an input vertex or lies on a segment or facet not incident to the facet containing f, then $\text{lfs}(v) \leq r_v$, so $D_v \leq 1$ and the theorem holds. If p is the circumcenter of a skinny tetrahedron (rejected for insertion because it encroaches upon f), $r_v \geq \frac{r_p}{\sqrt{2}}$ by Lemma 3, so by Lemma 4, $D_v \leq 1 + \sqrt{2} D_T$. It follows that one can prove that $D_v \leq D_F$ if D_F is chosen so that

$$1 + \sqrt{2} D_T \leq D_F. \tag{2}$$

Third, suppose v is inserted at the midpoint of a subsegment s. If its parent p is an input vertex or lies on a segment or facet not incident to the segment containing s, then $\text{lfs}(v) \leq r_v$, and the theorem holds. If p is the circumcenter of a skinny tetrahedron or encroached subfacet (rejected for insertion because it encroaches upon s), $r_v \geq \frac{r_p}{\sqrt{2}}$ by Lemma 3, so by Lemma 4, $D_v \leq 1 + \sqrt{2}\max\{D_T, D_F\}$. It follows that one can prove that $D_v \leq D_S$ if D_S is chosen so that

$$1 + \sqrt{2}\max\{D_T, D_F\} \leq D_S. \tag{3}$$

If the quality bound B is strictly larger than 2, Inequalities 1, 2, and 3 are simultaneously satisfied by choosing

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

$$D_S = \frac{(3 + \sqrt{2})B}{B - 2}.$$ ■

Theorem 6 (Ruppert [13]) *For any vertex v of the output mesh, the distance to its nearest neighbor w is at least $\frac{\text{lfs}(v)}{D_S + 1}$.*

Proof: Inequality 3 indicates that D_S is larger than D_T and D_F, so Theorem 5 shows that $\frac{\text{lfs}(v)}{r_v} \leq D_S$ for any vertex v. If v was added after w, then $|vw| = r_v \geq \frac{\text{lfs}(v)}{D_S}$, and the theorem holds.

If w was added after v, apply the theorem to w, yielding

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

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

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

It follows that $|vw| \geq \frac{\text{lfs}(v)}{D_S + 1}$. ■

To provide an example, suppose $B = 2.5$. Then $D_T \doteq 9.8$, $D_F \doteq 14.9$, and $D_S \doteq 22.1$. Hence, the spacing of vertices is at worst about 23 times smaller than the local feature size. Note that as B approaches 2, the values of D_T, D_F, and D_S approach infinity.

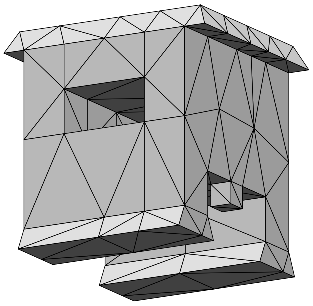
As Figure 13 shows, the algorithm performs much better in practice. In this example, as soon as all encroached subsegments and subfacets have been eliminated (upper left), the largest circumradius-to-shortest edge ratio is already less than 2.1. The shortest edge length is 1, and $\text{lfs}_{\max} = \sqrt{5}$, so the spectre of edge lengths 23 times smaller than the local feature size has not materialized. As the quality bound B decreases, the number of tetrahedra in the final mesh increases gracefully until B drops below 1.05. With $B = 1.04$, the algorithm fails to terminate on this example.

Figure 14 offers a demonstration of the grading of a tetrahedralization generated by Delaunay refinement. A cube has been truncated at one corner, cutting off a portion whose width is one-millionth that of the cube. Although this mesh satisfies a bound on circumradius-to-shortest edge ratio of $B = 1.2$, reasonably good grading is apparent. For this bound there is no theoretical guarantee, but the worst edge is 73 times shorter than the local feature size at one of its endpoints. If a bound of $B = 2.5$ is applied, the worst edge is 9 (rather than 23) times smaller than the local feature size at one of its endpoints.

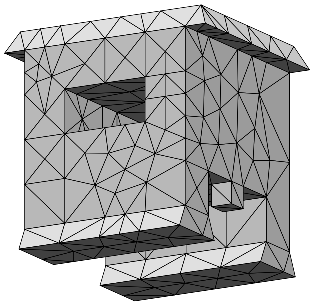
After the initial Delaunay tetrahedralization has been formed, my implementation runs in constant time per vertex insertion—in practice. The theoretical worst-case running time is surely much worse, and may be $\mathcal{O}(n^2)$ time per vertex insertion, even after amortization, in pathological cases. I do not know how to devise such a case, however. In practice, the first few vertex insertions usually banish any large vertex-free regions, after which further vertex insertions tend to be well-localized and thus execute in constant time.

8 Sliver Removal by Delaunay Refinement

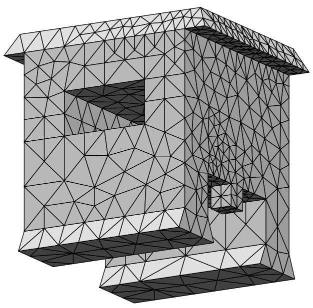
The only known theoretical guarantee about Delaunay refinement's ability to remove sliver tetrahedra is provided by Chew [4], whose bound on tetrahedron quality is too weak to provide any practical assurance. Nonetheless, it is natural to wonder whether Delaunay refinement might be effective in practice. If one inserts a vertex at the circumcenter of any tetrahedron with a small dihedral angle, will the algorithm fail to terminate?



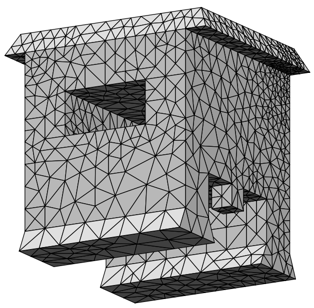
$B = 2.095$, $\theta_{\min} = 1.96°$, $\theta_{\max} = 176.02°$, $h_{\min} = 1$, 143 vertices, 346 tetrahedra.



$B = 1.2$, $\theta_{\min} = 1.20°$, $\theta_{\max} = 178.01°$, $h_{\min} = 0.743$, 334 vertices, 1009 tetrahedra.



$B = 1.07$, $\theta_{\min} = 1.90°$, $\theta_{\max} = 177.11°$, $h_{\min} = 0.369$, 1397 vertices, 5596 tetrahedra.



$B = 1.041$, $\theta_{\min} = 0.93°$, $\theta_{\max} = 178.40°$, $h_{\min} = 0.192$, 3144 vertices, 13969 tetrahedra.

Figure 13: Several meshes of a $10 \times 10 \times 10$ PLC generated with different bounds (B) on circumradius-to-shortest edge ratio. Below each mesh is listed the smallest dihedral angle $\theta_{\min}$, the largest dihedral angle $\theta_{\max}$, and the shortest edge length $h_{\min}$. The algorithm does not terminate on this PLC for the bound $B = 1.04$.

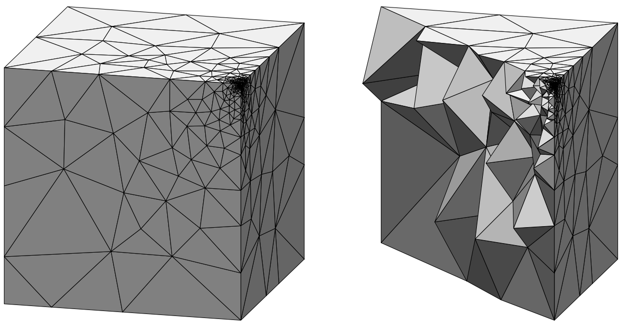
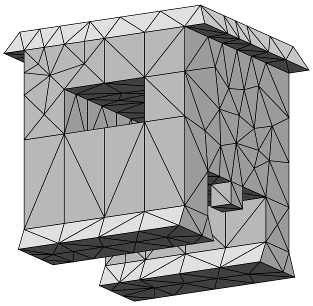


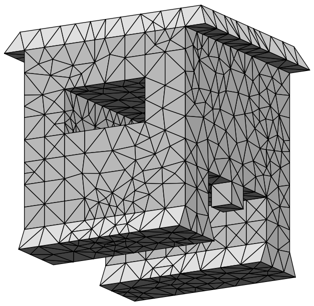
Figure 14: At left, a mesh of a truncated cube. At right, a cross-section through a diagonal of the top face.

As Figure 15 demonstrates, Delaunay refinement can succeed for useful dihedral angle bounds. Each of the meshes illustrated was generated by enforcing a lower bound $\theta_{\min}$ on dihedral angles, rather than a circumradius-to-shortest edge ratio bound. However, the implementation prioritizes poor tetrahedra according to their ratios, and thus slivers are split last. I suspect that the program generates meshes with fewer tetrahedra this way, and that the likelihood of termination is greater. Intuitively, one expects that a vertex inserted at the circumcenter of the tetrahedron with the largest ratio is more likely to eliminate more bad tetrahedra.

Both meshes illustrated have dihedral angles bounded between



$B = 1.88$, $\theta_{\min} = 23.5°$, $\theta_{\max} = 144.8°$, $h_{\min} = 0.765$, 307 vertices, 891 tetrahedra.



$B = 2.02$, $\theta_{\min} = 21.3°$, $\theta_{\max} = 148.8°$, $h_{\min} = 0.185$, 1761 vertices, 7383 tetrahedra.

Figure 15: Meshes created by Delaunay refinement with bounds on the smallest dihedral angle $\theta_{\min}$.

$21°$ and $149°$. The mesh on the right was generated with bounds on both tetrahedron volume and dihedral angle, so that enough tetrahedra were generated to ensure that the mesh on the left wasn't merely a fluke. (The best attainable lower bound drops by $2.2°$ as a result.) Experiments with very large meshes suggest that a minimum angle of $19°$ can be obtained reliably. See Chew [4] for some intimations as to why slivers might be eliminated so readily. One of his useful observations is that the region in which a tetrahedron's apex must lie so that the tetrahedron can simultaneously have poor dihedral angles and a good circumradius-to-shortest edge ratio is relatively small.

Unfortunately, my success in removing slivers is probably due in part to the severe restrictions on input angles I have imposed. Practitioners report that they have the most difficulty removing slivers at the boundary of a mesh, especially near small angles. Mesh improvement techniques such as optimization-based smoothing and topological transformations can likely remove some of the imperfections that cannot be removed directly by Delaunay refinement.

9 Conclusions

Delaunay refinement is an effective technique for three-dimensional mesh generation. Its theoretical guarantees on tetrahedron quality and grading make it attractive. However, these guarantees are not entirely satisfying. There is no guarantee that slivers can be eliminated. The bounds on edge length are reassuring, but might not be strong enough for all practical purposes, as the number of tetrahedra is inversely proportional to the cube of the edge length. Although Ruppert uses Theorem 6 to prove the size-optimality of the triangular meshes his algorithm produces, an analogous result is not possible in three dimensions.

Fortunately, Delaunay refinement algorithms outperform their worst-case bounds. There is a great deal of "slack" in the analysis; the relationship between the insertion radii of a child and its parent is usually looser than in the worst case. This slack accumulates as one traces a sequence of descendants from an input vertex, and makes it possible to achieve better bounds on tetrahedron quality and edge length than the theory promises. (Miller et al. [10] cannot exploit this slack, and thus require about 8,000 times as many vertices to guarantee a bound of $B = 2.5$ on the PLC in Figure 13).

The main outstanding problem in three-dimensional Delaunay refinement is the question of how best to handle small input angles. The difficulty is that if vertices lying in segments or facets can encroach upon incident subsegments and subfacets, then additional arcs appear in the dataflow diagram of Figure 12, creating cycles of ever-diminishing insertion radii. Fortunately, it is possible to modify the algorithm so that it always terminates, even when the

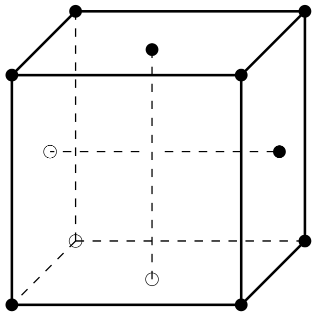


Figure 16: A counterexample demonstrating that the three-dimensional Delaunay refinement algorithm is not size-optimal.

projection condition is not satisfied. The insertion radius of each vertex can be explicitly computed. A vertex should be rejected if its insertion radius is too small compared to its parent's. As a result, some skinny tetrahedra may remain in the final mesh, but only in the vicinity of small input angles. A few other tricks are needed to ensure that all segments and facets are recovered: segments are recovered by using a technique based on concentric spherical shells suggested by Ruppert [13], then facets are recovered by forming a constrained Delaunay tetrahedralization [15].

The reason why a size-optimality result cannot be established is worth exploring. Gary Miller and Dafna Talmor (private communication) have pointed out the counterexample depicted in Figure 16. Inside this PLC, two segments pass very close to each other without intersecting. The PLC might reasonably be tetrahedralized with a few dozen tetrahedra having bounded circumradius-to-shortest edge ratios, if these tetrahedra include a sliver tetrahedron whose four vertices are the endpoints of the two internal segments. However, the best the present Delaunay refinement algorithm can promise is to fill the region with tetrahedra whose edge lengths are proportional to the distance between the two segments. Because this distance may be arbitrarily small, the algorithm is not size-optimal. However, if guaranteed bounds could be established for the dihedral angles, and not merely the circumradius-to-shortest edge ratios, then size-optimality might be proven using ideas like those with which Mitchell and Vavasis [11, 12] demonstrate the size-optimality of their octree-based algorithms.

Many theoretical improvements can be made to the core algorithm described here. The bound on circumradius-to-shortest edge ratio can be improved to 1.63 by replacing equatorial spheres with narrower protective shells called *equatorial lenses*. (This technique also requires the use of constrained Delaunay tetrahedralizations.) The bound can be further improved to 1.15 with a technique called *range-restricted segment splitting*, albeit by sacrificing good grading in theory (not in practice). With or without these improvements, a bound as small as 1 can be applied to any tetrahedron that does not intersect the interior of a segment or facet, and thus only boundary tetrahedra are subject to the weaker bound of 2, 1.63, or 1.15. The projection condition can be somewhat weakened; for example, a 60° separation between incident segments suffices. All these improvements are described in detail elsewhere [14]. The improvements in circumradius-to-shortest edge ratio, in particular, are significant and further propel the approach described here beyond the pioneering tetrahedral mesh generation algorithms of Dey et al., Chew, and Miller et al.

References

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

[2] L. Paul Chew. *Guaranteed-Quality Triangular Meshes*. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[3] ———. *Guaranteed-Quality Mesh Generation for Curved Surfaces*. Proceedings of the Ninth Annual Symposium on Computational Geometry, pages 274–280. ACM, May 1993.

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

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

[6] Lori A. Freitag and Carl Ollivier-Gooch. *A Comparison of Tetrahedral Mesh Improvement Techniques*. Fifth International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 87–100. Sandia National Laboratories, October 1996.

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

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

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

[10] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. *Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening*. Fifth International Meshing Roundtable, pages 47–61, October 1996.

[11] Scott A. Mitchell and Stephen A. Vavasis. *Quality Mesh Generation in Three Dimensions*. Proceedings of the Eighth Annual Symposium on Computational Geometry, pages 212–221, 1992.

[12] ———. *Quality Mesh Generation in Higher Dimensions*. Submitted to SIAM Journal on Computing, February 1997.

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

[14] Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997. Available as Technical Report CMU-CS-97-137.

[15] ———. *A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations*. This proceedings, June 1998.

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

[17] Nigel P. Weatherill. *Delaunay Triangulation in Computational Fluid Dynamics*. Computers and Mathematics with Applications **24**(5/6):129–150, September 1992.