Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery

Jonathan Richard Shewchuk

University of California at Berkeley, Berkeley, CA, U.S.A. jrs@cs.berkeley.edu

Abstract

In two dimensions, a *constrained Delaunay triangulation* (CDT) respects a set of segments that constrain the edges of the triangulation, while still maintaining most of the favorable properties of ordinary Delaunay triangulations (such as maximizing the minimum angle). CDTs solve the problem of enforcing *boundary conformity*—ensuring that triangulation edges cover the boundaries (both interior and exterior) of the domain being modeled. This paper discusses the three-dimensional analogue, *constrained Delaunay tetrahedralizations* (also called CDTs), and their advantages in mesh generation. CDTs maintain most of the favorable properties of ordinary Delaunay tetrahedralizations, but they are more difficult to work with, because some sets of constraining segments and facets simply do not have CDTs. However, boundary conformity can always be enforced by judicious insertion of additional vertices, combined with CDTs. This approach has three advantages over other methods for boundary recovery: it usually requires fewer additional vertices to be inserted, it yields provably good bounds on edge lengths (i.e. edges are not made unnecessarily short), and it interacts well with provably good Delaunay refinement methods for tetrahedral mesh generation.

Keywords: finite element mesh generation, tetrahedral mesh generation, constrained Delaunay tetrahedralization, boundary conformity, boundary recovery

1 Introduction

Delaunay tetrahedra have desirable properties that make them popular for finite element meshes. These properties include their tendency to favor "round" tetrahedra over "skinny" (high aspect ratio) tetrahedra, their suitability for interpolation [25, 24], and their mathematical properties that allow Delaunay refinement algorithms [6, 8, 14, 21, 22] to generate meshes that have provably good properties.

However, Delaunay tetrahedralizations are convex. The domains modeled with finite element methods usually are not. Domains have boundaries that must be respected by the mesh, including both exterior boundaries that bound the mesh, and interior boundaries used to separate regions having different material properties (e.g. the interface between two metals in a heat conduction problem), to represent known discontinuities in the solution, or to establish vertices, edges, and faces of the mesh where boundary conditions may be applied effectively.

Supported in part by the National Science Foundation under Awards ACI-9875170, CMS-9980063, and EIA-9802069, and in part by a gift from the Okawa Foundation. The views and conclusions in this document are those of the author. They are not endorsed by, and do not necessarily reflect the position or policies of, the Okawa Foundation or the U. S. Government.

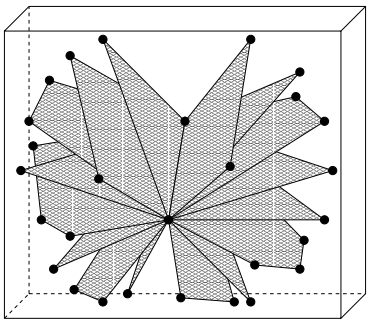

Figure 1: It is difficult to mesh the interior of this box with Delaunay tetrahedra that conform to all the facets.

The problem is most difficult for domains where boundaries are separated by small dihedral angles, as in Figure 1. Vertices inserted to recover one boundary—so that it is represented as a union of triangular faces of the Delaunay tetrahedralization—are likely to knock faces of the adjacent

boundaries out of the Delaunay tetrahedralization.

There are three established approaches to this problem. One is the *conforming Delaunay* approach, in which additional vertices are inserted into the mesh—while the Delaunay property of the tetrahedralization is maintained—until it *conforms* to the boundaries, meaning that each boundary is represented by a union of triangular faces of the mesh. The question of *where* to insert additional vertices to obtain boundary conformity is difficult. Murphy, Mount, and Gable [16] and Cohen-Steiner, de Verdière, and Yvinec [4] offer algorithms that work even with difficult examples like the box in Figure 1. However, their tetrahedralizations may have very short edges, which engender very small tetrahedra, and the numbers of extra vertices can be large. These algorithms might not even produce meshes whose complexity is polynomial in the complexity of the description of the input domain. (The approach described in this paper does not have polynomial bounds either, but it does reduce the number of extra vertices.) And nobody knows how to reconcile the need for good-quality elements with the need for boundary conformity in a purely Delaunay mesh.

The second approach, and the most common one in the engineering literature [11, 12, 29], might be called the *almost Delaunay* approach. Missing domain boundaries are recovered by inserting additional vertices where the boundaries intersect the faces or edges of the tetrahedralization. However, the Delaunay property is not maintained during these vertex insertions (at least not fully), so vertex insertions do not knock recovered boundaries out of the mesh. The tetrahedra are subdivided in ways that introduce new vertices without eliminating existing faces and edges that represent domain boundaries. Once all the boundaries are recovered, a mesh generator might attempt to regain the Delaunay property of the mesh by using topological flips, but because the flips are not permitted to disturb the domain boundaries, the mesh is usually not entirely Delaunay. The deviations occur at the boundaries—where obtaining high-quality elements is most difficult. One consequence is that provably good Delaunay refinement procedures cannot be guaranteed to work correctly. "Almost Delaunay" mesh generation programs do work well in many circumstances, but they are unlikely to be able to control the quality of the elements in circumstances like Figure 1. Furthermore, they are also in danger of creating very short edges, or inserting many more vertices than necessary, unless they are carefully designed.

The third approach uses *constrained Delaunay tetrahedralizations* (CDTs). CDTs are composed entirely of *constrained Delaunay* tetrahedra, which are not always Delaunay, but nevertheless retain two of the favorable properties of Delaunay tetrahedra: they help to control interpolation error [24], and they help to prove that Delaunay refinement algorithms reliably generate good meshes [22]. CDTs have the advantage of requiring fewer vertices than conforming Delaunay tetrahedralizations, and as this paper will show, it is straightforward to form CDTs without creating unnecessarily short edges.

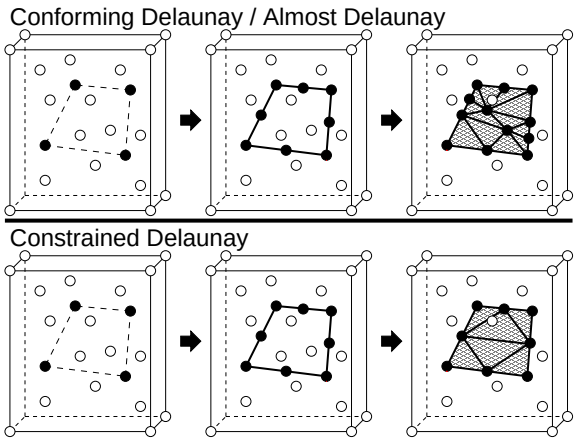

Figure 2: Two different methods for recovering a two-dimensional interior boundary inside a cubical domain. The initial Delaunay tetrahedralization does not respect the rectangular boundary. (For clarity, the tetrahedra are not shown.) Both approaches insert additional vertices to recover the missing domain edges. Next, the standard approaches insert more vertices to recover missing facets (top), but no additional vertices are needed if constrained Delaunay tetrahedra are used (bottom).

It takes some care to construct a CDT that conforms to a domain. This paper details the following approach. First, form the Delaunay tetrahedralization of the vertices of the domain. This tetrahedralization covers the entire convex hull of the domain, and probably does not respect all the domain boundaries. Second, identify the domain edges and determine which of them fail to appear in the tetrahedralization (see Section 4). Third, insert additional vertices into the tetrahedralization (while maintaining the Delaunay property), with the aim of recovering the missing domain edges. An algorithm for placing these vertices is given in Section 5.

So far, the approach is identical to the construction of a conforming Delaunay tetrahedralization, and to some of the "almost Delaunay" approaches as well. The approaches part ways completely when all the domain edges have been recovered, and it is time to recover the missing domain faces. Figure 2 illustrates the difference. The standard approaches insert more vertices to recover the missing faces. The approach expounded here simply replaces the Delaunay tetrahedra with constrained Delaunay tetrahedra, thereby recovering the missing faces without inserting another vertex. Algorithms for constructing CDTs are discussed in Sections 6 and 7.

This technique may yield a large, even asymptotic, savings in the number of vertices required to obtain domain conformity. Three-dimensional CDTs also make it possible for Delaunay refinement algorithms to mesh any PLC, even for domains with small angles [22], and to establish provable bounds on the lengths of the edges and shapes of the tetrahedra they produce.

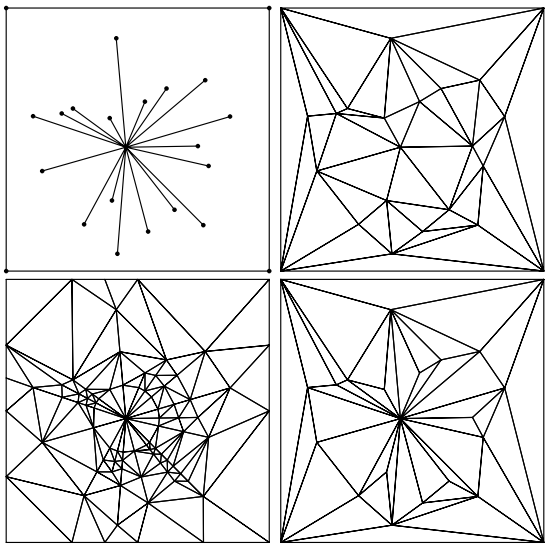

Figure 3: The Delaunay triangulation (upper right) of the vertices of a PSLG (upper left) might not respect the segments of the PSLG. These segments can be incorporated by adding vertices to obtain a conforming Delaunay triangulation (lower left), or by forgoing Delaunay triangles in favor of constrained Delaunay triangles (lower right).

2 Definitions: PLCs and CDTs

The phrase "constrained Delaunay triangulation" appears often in the meshing literature, but there appears to be little agreement on exactly what it means, especially in three dimensions. In contrast, "constrained Delaunay" and "conforming Delaunay" are rigorously defined in the computational geometry literature; this section explains those definitions. My apologies in advance to anyone who wishes "constrained Delaunay triangulation" to mean something else.

I illustrate with two-dimensional examples first. Both conforming and constrained Delaunay triangulations are defined on the assumption that the input is a *planar straight line graph* (PSLG) X, which is a set of vertices and *segments* (constraining edges) illustrated in Figure 3 (upper left). A triangulation is sought that contains the vertices of X and respects the segments of X.

In a *conforming Delaunay triangulation* (Figure 3, lower left), every simplex (triangle, edge, or vertex) is *Delaunay*. A simplex is Delaunay if there exists a *circumcircle* of the simplex—a circle that passes through all its vertices—that encloses no vertex (although any number of vertices is permitted on the circle). The vertices of X are augmented by additional vertices (sometimes called *Steiner points*) carefully chosen so that the Delaunay triangulation of the augmented vertex set *respects* all the segments—in other words, so that each segment is the union of a contiguous linear sequence of edges of the triangulation. Edelsbrunner and Tan [10] show that any PSLG can be triangulated with the addition of $\mathcal{O}(m^2 n)$ augmenting vertices, where m is the number of segments in X, and n is the number of vertices. Few PSLGs require this many augmenting vertices, but PSLGs are known for which $\Theta(mn)$ augmenting vertices are needed. Closing the gap between the $\mathcal{O}(m^2 n)$ and $\Omega(mn)$ bounds remains an open problem.

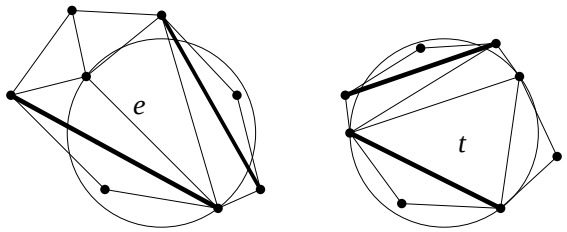

Figure 4: The edge e and the triangle t are each constrained Delaunay. Bold lines represent segments.

A *constrained Delaunay triangulation* (CDT) [13, 2] of X has no vertices not in X, and every segment of X is a single edge of the CDT (Figure 3, lower right). However, a CDT, despite its name, is not a Delaunay triangulation, because the simplices are not required to be Delaunay. Instead, every simplex must either be a segment specified in X or be *constrained Delaunay*. A simplex is constrained Delaunay if it has a circumcircle that encloses no vertex of X that is *visible* from any point in the relative interior of the simplex; and furthermore, the relative interior of the simplex does not intersect any segment. Visibility is occluded only by segments of X.

Figure 4 demonstrates examples of a constrained Delaunay edge e and a constrained Delaunay triangle t. Segments in X appear as bold lines. Although there is no empty circle that encloses e, the depicted circumcircle of e encloses no vertex that is visible from the relative interior of e. There are two vertices inside the circle, but both are hidden behind segments. Hence, e is constrained Delaunay. Similarly, the circumcircle of t encloses two vertices, but both are hidden from the interior of t by segments, so t is constrained Delaunay.

The advantage of a CDT over a conforming Delaunay triangulation is that it has no vertices other than those in X. The disadvantage is that its triangles are not Delaunay. However, a CDT retains many of the desirable properties of Delaunay triangulations. For instance, a two-dimensional CDT maximizes the minimum angle in the triangulation, compared with all other constrained triangulations of X [13].

CDTs generalize to three or more dimensions [20], but whereas every PSLG has a CDT, not every polyhedron has one. One of the reasons why three-dimensional boundary recovery is difficult is that there are polyhedra that cannot be tetrahedralized at all without additional vertices. Schönhardt [18] furnishes a three-dimensional example depicted in Figure 5 (right). The easiest way to envision this polyhedron is to begin with a triangular prism. Imagine grasping the prism so that one of its two triangular faces cannot move, while the opposite triangular face is rotated

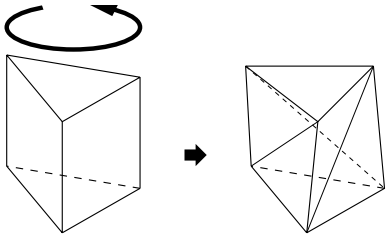

Figure 5: Schönhardt's untetrahedralizable polyhedron (right) is formed by rotating one end of a triangular prism (left), thereby creating three diagonal reflex edges.

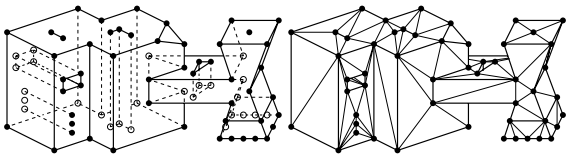

Figure 6: Each facet of a PLC (left) may have holes, slits, and interior vertices, which may be used to enforce the presence of specific faces (perhaps so that boundary conditions may be applied) or to support intersections with other facets. The right illustration is the constrained Delaunay tetrahedralization of the PLC.

slightly about its center without moving out of its plane. As a result, each of the three square faces is broken along a diagonal *reflex edge* (an edge at which the polyhedron is locally nonconvex) into two triangular faces. After this transformation, the upper left corner and lower right corner of each (formerly) square face are separated by a reflex edge and are no longer visible to each other within the polyhedron. Any four vertices of the polyhedron include two separated by a reflex edge; thus, any tetrahedron whose vertices are vertices of the polyhedron will not lie entirely within the polyhedron. Therefore, Schönhardt's polyhedron cannot be tetrahedralized without an additional vertex. (One extra vertex in the center will do.)

Realistic three-dimensional domains are often more complicated than polyhedra, so consider a more general input called a *piecewise linear complex* (PLC), following Miller, Talmor, Teng, Walkington, and Wang [15].[1] A PLC X is a set of vertices, segments, and *facets*, as illustrated in Figure 6. Each facet is a polygon (roughly speaking), possibly with holes, slits, and isolated vertices in it. As the figure shows, a facet may have any number of sides and may be nonconvex. Just as a segment imposes a one-dimensional constraint on a triangulation, a facet imposes a two-dimensional constraint: for a tetrahedralization T to be a CDT of X, each facet of X must be a union of triangular faces of T.

PLCs have restrictions like those of any other type of complex. If X contains a facet f, then X must contain every

[1] Miller et al. call it a *piecewise linear system*, but their construction is so obviously a complex that a change in name seems obligatory.

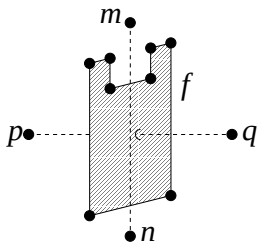

Figure 7: In this example, the facet f occludes the visibility between p and q. However, m and n can see each other. Their view is not blocked by f, because m and n lie in the same plane as f; nor is it blocked by any boundary segment of f, because segments do not affect visibility.

segment and vertex of f. A segment and a facet may intersect only at a shared vertex, unless the segment lies in the boundary of the facet. Any two facets of a PLC may intersect only at a shared segment or vertex, or a union of shared segments and vertices. (Because facets are nonconvex, two facets may intersect in several places.)

The *triangulation domain* is the region of space a user wishes to tetrahedralize. We could choose the convex hull by default, but sometimes it pays to be more specific, because there are PLCs for which a CDT of the triangulation domain exists but a CDT of its convex hull does not. For example, it is easy to tetrahedralize the domain sandwiched between Schönhardt's polyhedron and the boundary of its convex hull, even though the interior of the polyhedron is not tetrahedralizable.

The triangulation domain is required to be *facet-bounded*, meaning that facets of X entirely cover the boundary that separates the triangulation domain from its complement, the *exterior domain*. The exterior domain includes any hollow cavities enclosed by the triangulation domain, as well as outer space. Some facets, known as *interior facets*, may have the triangulation domain on both sides. Such facets allow PLCs to represent non-manifold and multiple-component domains—for instance, by serving as interfaces between two different materials.

To define what a CDT is in three dimensions, more definitions are needed. Say that the visibility between two points p and q is *occluded* if there is a constraining facet f of X such that p and q lie on opposite sides of the plane that includes f, and the line segment pq intersects f. If either p or q lies in the plane that includes f, then f does not occlude the visibility between them. See Figure 7. Segments in X do not occlude visibility. The points p and q are *visible* from each other (equivalently, can *see* each other) if there is no occluding facet of X.

Let s be any simplex (tetrahedron, triangle, edge, or vertex) whose vertices are in X (but s is not necessarily in X). Let S be a sphere; S is a *circumsphere* of s if S passes through all the vertices of s. If s is a tetrahedron, then s has a unique circumsphere; otherwise, s has infinitely many circumspheres.

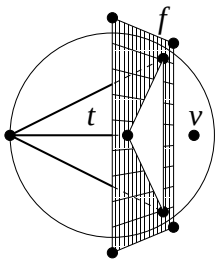

Figure 8: A constrained Delaunay tetrahedron t.

The simplex s is *Delaunay* if there is a circumsphere S of s that encloses no vertex of X (although any number of vertices is permitted on the sphere itself). The simplex s is *strongly Delaunay* if there is a circumsphere S of s such that no vertex of X lies inside *or on* S, except the vertices of s. Every vertex is strongly Delaunay.

One reason for the distinction between Delaunay and strongly Delaunay simplices is because if a vertex set has five or more vertices that lie on a common empty sphere, the vertex set has more than one (unconstrained) Delaunay tetrahedralization. Every Delaunay simplex appears in at least one of those tetrahedralizations, but a strongly Delaunay simplex appears in *every* Delaunay tetrahedralization.

Loosely speaking, the simplex s *respects* X if no segment is cut in two by s, and s does not penetrate from one side of a facet to the other. Formally, s respects X if s lies in the triangulation domain *and* the intersection of s and any segment or facet of X is a union of faces of s. This union may be the empty set, s itself, or the melding of several faces of s, possibly of mixed dimension. For example, a nonconvex facet might intersect two or even three edges of a triangle without including the triangle's interior.

The simplex s is *constrained Delaunay* if

- s respects X,[2] and
- there is a circumsphere S of s such that no vertex of X inside S is visible from any point in the relative interior of s.

Figure 8 depicts a constrained Delaunay tetrahedron t. The intersection of t with the facet f is a face of t, so t respects X. The circumsphere of t encloses one vertex v, but v is not visible from any point in the interior of t (even though v is visible from some points on the boundary of t).

A tetrahedralization T is a *constrained tetrahedralization* of X if T and X have the same vertices (no more, no less), the tetrahedra in T respect X, and the tetrahedra in T entirely cover the triangulation domain. This definition implies that each facet in X is a union of triangular faces of T.

A *constrained Delaunay tetrahedralization* of X is a constrained tetrahedralization of X in which each tetrahedron is constrained Delaunay.

[2] This definition of "constrained Delaunay" differs slightly from the definition I have used in previous publications [20, 23]. The present definition is more sound.

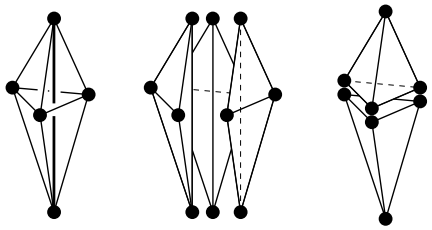

Figure 9: Left: A PLC with no CDT. Center: The sole constrained tetrahedralization of this PLC. Its three tetrahedra are not constrained Delaunay. Right: The two Delaunay tetrahedra do not respect the central segment.

The difficulty of incorporating segments is the main impediment to the existence of CDTs. Figure 9 offers an example of a PLC with no CDT. There is one segment that runs through the interior of the PLC's convex hull. There is only one constrained tetrahedralization of this PLC—composed of three tetrahedra bordering the central segment—and its tetrahedra are not constrained Delaunay, because each of them has a vertex inside its circumsphere. If the central segment is removed, the PLC has a CDT made up of two tetrahedra.

By convention, to say that T *is the CDT of* X is to say that T has no vertices other than those in X. However, Delaunay mesh generators insert additional vertices to improve the quality of the elements. Moreover, although a given PLC X might not have a CDT, additional vertices can be inserted into X to yield a new PLC Y that does have a CDT T. (Section 5 explains how to choose the extra vertices.) T is not a CDT of X, because it has vertices that X lacks, but T is what I call a *conforming constrained Delaunay tetrahedralization* (CCDT) of X: "conforming" because additional vertices are permitted, and "constrained" because its tetrahedra (and faces and edges) are all constrained Delaunay.

One advantage of a CCDT over a conforming Delaunay tetrahedralization of X is that the number of additional vertices inserted is generally smaller. The remainder of this paper describes how to construct a CCDT of X.

3 Edge Protection

The theorem that makes three-dimensional CDTs useful states that there is a simple condition that guarantees the existence of a CDT. A PLC X is *edge-protected* if every segment in X is strongly Delaunay.

Theorem 1 ([20]) *If X is edge-protected, then X has a CDT.* ■

It is not sufficient for every segment to be Delaunay. If Schönhardt's polyhedron is specified so that all six of its vertices lie on a common sphere, then all of its edges (and

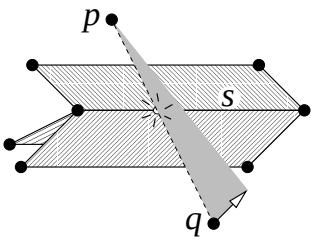

Figure 10: Example of a grazeable segment s. Note that p cannot see q, but if q is perturbed by an infinitesimal amount, then p and q can see each other. There is a line of sight that grazes s, hence the term *grazeable*.

its faces as well) are Delaunay, but it still does not have a tetrahedralization. It is not possible to place the vertices of Schönhardt's polyhedron so that all three of its reflex edges are strongly Delaunay (though any two may be).

Although the boundary segments of a facet must be strongly Delaunay for the guarantee to hold, no such restriction applies to the edges introduced into the facet by meshing. For example, consider finding a tetrahedralization of a cube. Each square face must be partitioned by a diagonal edge to yield a triangulation. The diagonals are not strongly Delaunay, and if they were segments in the PLC, a constrained tetrahedralization might not exist (depending on the choice of diagonals). If the diagonals are not elements of the PLC, the existence of a CDT is guaranteed by Theorem 1, and a CDT construction algorithm can choose a compatible set of diagonals.

Next, a stronger and even more useful version of Theorem 1. A PLC segment often serves as a boundary to several facets, which can be ordered as they appear in rotary order around the segment. A segment is *grazeable* if two consecutive facets in the rotary order are separated by an interior angle of 180° or more (as Figure 10 shows), or if the segment is included in fewer than two facets. (An *interior angle* subtends the interior of the triangulation domain. Exterior angles of 180° or more are not an impediment to a CDT.) Only the grazeable segments need to be strongly Delaunay to guarantee a CDT. A three-dimensional PLC X is *weakly edge-protected* if each grazeable segment in X is strongly Delaunay. If X is weakly edge-protected, then X has a CDT.

Segments that are not grazeable occur commonly in practice. For instance, in a regular complex of cubical cells, no segment is grazeable. The stronger result excludes the non-grazeable segments from the need to be strongly Delaunay.

But what if X is not weakly edge-protected, and does not have a CDT? Any PLC can be made edge-protected (weakly or fully) by the insertion of additional vertices that split the segments into smaller segments, as described in Section 5.

4 Testing Edge Protection

Testing whether a PLC X is edge-protected is straightforward. Form the Delaunay tetrahedralization T of the vertices of X using any standard algorithm [1, 3, 28]. T covers the entire convex hull of X. If a segment s is missing from T, then s is not strongly Delaunay.

If s is an edge of T, then s is Delaunay but might not be strongly Delaunay. The recommended solution is to use the symbolic perturbation technique discussed in Section 9 while constructing the Delaunay tetrahedralization. The perturbations ensure that every edge (and triangle and tetrahedron) that is Delaunay is also strongly Delaunay. Then, if s is an edge of T, s is strongly Delaunay.

If the perturbation technique is not used, there is a test that checks whether a Delaunay edge s is strongly Delaunay by examining only the tetrahedra of T that have s for an edge. Details are omitted because the perturbation technique has many advantages.

5 Provably Good Edge Protection

Given a set of vertices and segments in three-dimensional space, where should additional vertices be inserted (splitting segments into smaller segments) to ensure that all of the segments are strongly Delaunay, so that all the segments are guaranteed to appear in a Delaunay tetrahedralization?

This problem is either easy or difficult, depending on what constraints you set on the new vertices. If you ask that the number of added vertices be polynomial in the number of input vertices and segments, the problem is unsolved (even though the problem is easier than constructing a conforming Delaunay tetrahedralization, because there is no need to recover facets). Edelsbrunner and Tan's $\mathcal{O}(m^2 n)$-vertex solution for the two-dimensional case [10] has not been generalized to three dimensions, and even it is rather complicated.

However, mesh generation has different needs. The PLCs that require the most additional vertices have segments spaced very closely together or meeting at small angles. However, high-quality meshes of such PLCs might need a superpolynomial number of vertices anyway. For example, two parallel segments that are 10,000 units long and spaced one unit apart will require perhaps 20,000 tetrahedra to mesh if the tetrahedra are required to have good aspect ratios.

A more meaningful goal for provably good boundary recovery is to ensure that no two vertices are spaced much more closely together (say, by a small constant factor) than necessary to form a high-quality tetrahedral mesh of the domain. Provably good Delaunay refinement algorithms create meshes in which each edge's length is proportional to the *local feature size* at that edge. A boundary recovery algorithm should do the same. If smaller tetrahedra are needed for accuracy, they can easily be refined later; but if boundary recovery creates too-small tetrahedra, it might not be possible to fix them.

For a fixed PLC X, the local feature size $\mathrm{lfs}(p)$ of any point p in space is the radius of the smallest ball centered at p that intersects two segments or vertices in X that do not intersect each other. Delaunay refinement algorithms typically bound

the sizes of the elements they create in terms of this or a similar definition of lfs [17, 21]. lfs is a continuous function that is positive everywhere and suggests a rough upper bound on how large high-quality elements can be. It can vary widely over the domain, reflecting the impact of domain geometry on element size. The function lfs is defined in terms of the input PLC and does not change as new vertices are inserted.

Observe that this definition of local feature size does not take facets into account. This is good; it means that facets have no effect on the edge lengths in the final triangulation (though the segments that bound each facet do). Perhaps later, small distances between facets and vertices may force a Delaunay refinement algorithm to create much smaller edges (to eliminate poor-quality elements), but it will not be the fault of the boundary recovery algorithm.

The edge protection algorithm proceeds in two steps, illustrated in Figure 11. The first step uses protecting spheres centered at input vertices to choose locations to insert new vertices. Let V be the set of vertices in X where at least two segments meet at an angle less than 90°. Imagine that each vertex v_i in V is the center of a sphere S_i with some appropriate radius r_i, which may be different for each vertex. (Let us momentarily put off the question of how to choose these radii.) For each segment s that meets some other segment at an angle of less than 90°, let v_i be the vertex where they meet. The algorithm inserts a new vertex at the point $s \cap S_i$, as illustrated. Insert all the new vertices generated this way into the Delaunay tetrahedralization constructed in Section 4 (using the Bowyer–Watson [1, 28] algorithm, so the mesh remains Delaunay).

The protecting spheres cut off the ends of some segments. Each of these "ends" is guaranteed to be strongly Delaunay because no vertex lies on or inside its diametral sphere (the smallest sphere that contains the end), except its endpoints. Therefore, all the ends appear as edges in the updated Delaunay tetrahedralization. The only segments that can pierce the diametral spheres of the ends are other ends. The second step does not insert a vertex into any of the ends, so the ends remain strongly Delaunay.

The second step recovers the segments that are not ends by recursive bisection. Any segment that is not strongly Delaunay is split in two with a new vertex at its midpoint. The Delaunay tetrahedralization of the vertices is maintained throughout, so subsegments that are not strongly Delaunay can be diagnosed as described in Section 4. When every subsegment is strongly Delaunay, the PLC is edge-protected and the algorithm terminates.

This procedure can be used to make a PLC weakly or fully edge-protected. To obtain full edge protection, the procedure considers all segments—except edges of the convex hull, which are always strongly Delaunay. (This means true edges of the convex hull—an edge that lies in a face of the convex hull, but not in the boundary of the face, must be considered. Convex hull edges can be discovered by searching for dihedral angles greater than 180° on the boundary of the initial tetrahedralization.) To obtain weak edge protection, the procedure considers only the grazeable segments, and ignores the others as if they did not exist. The advantage of weak edge protection is that fewer new vertices are inserted, but the incremental facet insertion algorithm of Section 7 can construct the CDT only of a fully edge-protected PLC. Weak edge protection suffices for the CDT construction algorithms discussed in Section 6.

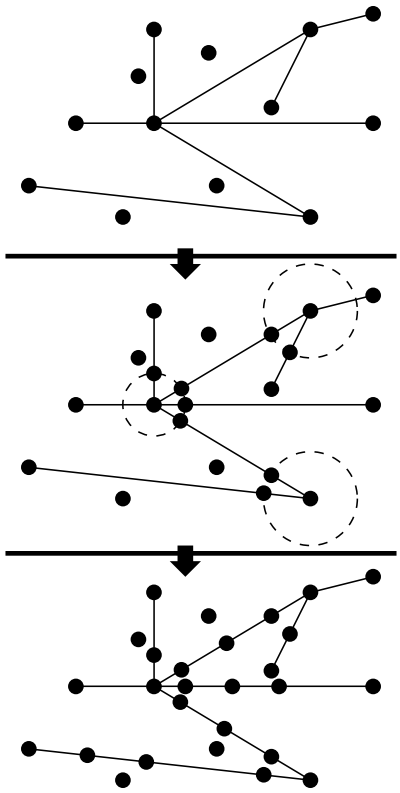

Figure 11: Making a PLC edge-protected. (Here illustrated in two dimensions, but the principle is the same in three.) First, use spheres centered at input vertices to subdivide segments that meet other segments at angles less than 90°. Second, bisect any segment that is not strongly Delaunay. Observe that the second step does not split the "ends" created by the first step because no new vertices can appear in their diametral spheres.

How are the sphere radii r_i chosen? The radii should be as large as possible while accomplishing two goals. First, if both ends of a segment are clipped, the middle subsegment should not be too short. This goal is accomplished by choosing r_i no larger than $\ell_i/3$, where ℓ_i is the length of the shortest segment that is clipped by S_i. Second, the diametral spheres of the ends cannot enclose any vertices, nor any subsegments that are not ends. This goal is accomplished by choosing r_i no larger than $\mathrm{lfs}(v_i)$, and no larger than $2d_i/3$, where d_i is the length of the shortest segment adjoining v_i whose *other* end is clipped. (Note that d_i is sometimes less than ℓ_i, because a segment whose far end is clipped and

whose near end is not clipped—because only one end adjoins another segment at an angle less than 90°—figures into d_i but not ℓ_i.) Hence, set $r_i = \min\{\text{lfs}(v_i), \ell_i/3, 2d_i/3\}$.

The algorithm must explicitly compute the local feature size at some vertices. The local feature size $\text{lfs}(v_i)$ is simply the distance from v_i to the nearest vertex or segment that doesn't intersect v_i. A Delaunay tetrahedralization connects each vertex to its nearest neighbor (via an edge), so the initial tetrahedralization helps find the nearest vertex quickly. The nearest segment may be found by checking every segment of the PLC.

Theorem 2 *This algorithm for edge-protecting a PLC creates no segment shorter than one-quarter the local feature size of any point in that segment.*

Proof: Observe that an end e adjoining vertex v_i has a length of at least $\text{lfs}(v_i)/3$. A well-known property [17, 21] of the lfs function is that for any two points u and v, $\text{lfs}(u) \leq \text{lfs}(v) + |uv|$. So at any point u in the end e, $\text{lfs}(u) \leq \text{lfs}(v_i) + |e| \leq 4|e|$, where $|e|$ is the length of e. It follows that for any point u in any end e, $|e| \geq \text{lfs}(u)/4$.

Let s be the segment left over after one or both ends of a segment t are cut off. For any point u in s, $|s| \geq |t|/3 \geq \text{lfs}(u)/3$.

During the second step, a segment s is produced only when a segment of length $2|s|$ has a vertex v on or inside its diametral sphere, and v does not intersect the original segment of which s is a subsegment. For any point u in s, the ball of radius $2|s|$ centered at u intersects both v and s, so $\text{lfs}(u) \leq 2|s|$. It follows that for any point u in any segment s produced during the second step, $|s| \geq \text{lfs}(u)/2$. ■

The same result applies to every edge of the CDT, except where two segments meet at an angle less than 60°. Unfortunately, small angles sometimes unavoidably engender short edges, but the edge lengths are still bounded in terms of the angles.

Theorem 3 *Let T be the CCDT of a PLC X yielded by applying the edge-protection algorithm described above, then constructing the CDT of the augmented PLC. Let e be any edge of T, and let u be any point in e. If the endpoints of e lie in two different segments of X that meet at an angle of $\phi < 60^\circ$, then $|e| \geq \text{lfs}(u)\sin(\phi/2)/2$. For any other edge, $|e| \geq \text{lfs}(u)/4$.*

Proof: If e is a subsegment, the result follows from Theorem 2. Otherwise, let w and x be the endpoints of e. If w and x lie on nonintersecting features (vertices or segments) of X, then $\text{lfs}(u) \leq |e|$ by the definition of local feature size.

The only other case is where w and x lie in two segments of X that meet at some vertex v at some angle ϕ. The triangle $\triangle vwx$ has side lengths $b = |wv|$, $c = |xv|$, and $|e|$. Assume without loss of generality that $b \leq c$. The point u lies in e, so $|uv| \leq c$ and $\text{lfs}(u) \leq \text{lfs}(v) + |uv| \leq 3b + c$.

By the law of cosines, $|e|^2 = b^2 + c^2 - 2bc\cos\phi$. If b is held fixed, calculus shows that the ratio $|e|/(3b+c)$ is minimized for a value of c that is (for any ϕ) less than b. But by assumption, $c \geq b$, so the worst (smallest) ratio is attained with $c = b$. For this case, $\triangle vwx$ is isosceles and basic trigonometry shows that $|e| = 2b\sin(\phi/2)$. Therefore, $|e|/\text{lfs}(u) \geq |e|/(3b + c) \geq 2b\sin(\phi/2)/(4b) = \sin(\phi/2)/2$. ■

6 Constructing CDTs

The first step of constructing the CDT of a PLC is to construct the two-dimensional CDTs (triangulations) of all the facets of the PLC. This can be done using Chew's $\mathcal{O}(n \log n)$ algorithm [2] (where n is the number of vertices in a facet), by using a two-dimensional version of the gift-wrapping algorithm described below, or by constructing the Delaunay triangulation of the vertices of a facet, then inserting the segments one by one. Call each triangle of each facet CDT a *constraining triangle*, because these triangles are constrained to appear as faces of the tetrahedralization.

There are two known algorithms that can construct the CDT of any PLC X that has a CDT, including any weakly edge-protected PLC—if no five vertices of X are cospherical. (If the latter condition is not satisfied, see Section 9.) These are naïve gift wrapping, and a sweep algorithm [23]. Naïve gift wrapping is easier to implement, but the more complicated sweep algorithm is faster. The running time of gift wrapping is $\mathcal{O}(n_v n_f n_s)$, where n_v is the number of vertices, n_f is the number of constraining triangles, and n_s is the number of tetrahedra in the CDT. For large PLCs, this is too slow. The sweep algorithm runs in $\mathcal{O}(n_v n_s)$ time, but in practice the running time is likely to be $\mathcal{O}(n_v^2 + n_s \log n_v)$ or better in all but the most pathological cases. Hence, the sweep algorithm is fast enough for most circumstances.

Nevertheless, the gift-wrapping algorithm is usually fast enough to be used for incremental facet insertion (described in the next section), so it is described here. (Space prevents my describing the sweep algorithm here.) It is a variation on the gift-wrapping algorithms for constructing convex hulls [26] and ordinary Delaunay triangulations [27, 7].

Gift-wrapping begins by choosing one constraining triangle, which serves as a *seed* upon which the constrained Delaunay tetrahedra crystallize one by one. Both sides of this triangle, plus each face of every crystallized tetrahedron, is used as a base from which to search for the vertex that serves as the apex of an adjacent tetrahedron.

Gift-wrapping is based on a straightforward procedure for "growing" a tetrahedron from a triangle, illustrated in Figure 12. Let f be either a constraining triangle or a face of a constrained Delaunay tetrahedron. Assume without loss of generality that f is oriented horizontally, and the constrained Delaunay tetrahedron immediately above f is sought. Suppose that at least one vertex of X lies above f. Let S be a

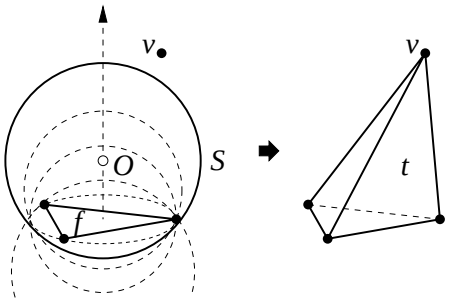

Figure 12: A sphere S that circumscribes f, expanding in search of a vertex v to form a new tetrahedron $t = \mathrm{conv}(f \cup v)$.

sphere that can shrink or expand, but always circumscribes f. Suppose the center O of S is initially infinitely far below f, so that the "inside" of S is the open halfspace below f; then O moves up until the portion of S above f touches the first vertex v that is visible from some point (any fixed point will do) in the interior of f.

Let t be the tetrahedron $\mathrm{conv}(f \cup v)$. If a CDT of X exists, then t is constrained Delaunay. If no eligible vertex can be found, or if t is not constrained Delaunay, then X has no CDT.

The gift-wrapping algorithm is as follows. Let f be the seed triangle. Say that both sides of f are *unfinished*. Thereafter, say that a triangular face of the growing tetrahedralization is *unfinished* if the algorithm has not yet identified the second constrained Delaunay tetrahedron that shares the face. To *finish* a face is to construct the second tetrahedron, or to determine that the face adjoins the exterior domain and there is no second tetrahedron.

The gift-wrapping algorithm maintains a dictionary (i.e. a hash table) of unfinished faces, which initially contains both sides of f. Repeat the following steps: remove an arbitrary unfinished face f from the dictionary and search for a vertex v that finishes f. If no vertex is above f, then f lies on the boundary of the convex hull. If f does not lie on the convex hull boundary and does not bear a notation that indicates that it adjoins the exterior domain, find v through the growth procedure and add $t = \mathrm{conv}(f \cup v)$ to the growing tetrahedralization. Check each face of t, except f, against the dictionary. If a face is already present in the dictionary, then the face is now finished, so remove it from the dictionary. Otherwise, the face is new and has one unfinished side, so insert it into the dictionary.

Finishing a single face takes $\mathcal{O}(n_v n_f)$ time, because it may be necessary to test the visibility of each vertex from the face, which is done by testing each vertex against every constraining triangle in X. (Clearly, it is the visibility testing that makes gift wrapping CDTs so slow.) Gift wrapping thus runs in $\mathcal{O}(n_v n_f n_s)$ time. This leaves much room for improvement, which can be realized by using the sweep algorithm instead. For practical purposes it seems likely that CDT construction by gift wrapping can be sped up by using it in concert with incremental facet insertion.

7 Incremental Facet Insertion

Because the naïve gift wrapping algorithm is slow, and the faster sweep algorithm takes a good deal of effort to implement and is difficult to make robust, I suggest a third CDT construction algorithm. The algorithm begins with a Delaunay tetrahedralization of the vertices of a fully edge-protected PLC—recall that we have constructed both an edge-protected PLC and its tetrahedralization in Section 5—and incrementally recovers the missing facets one by one.

The incremental facet insertion algorithm seems likely to strike the best trade-off between ease of implementation and speed in many practical applications. Another advantage of the algorithm is that there is no need to precompute the two-dimensional CDTs of the facets; these are produced automatically. Hence, this may be the easiest option to implement.

The disadvantage of incremental facet insertion is that the algorithm does not work on weakly edge-protected PLCs; they must be fully edge-protected. A facet can only be inserted if all the segments that bound it already appear as edges in the tetrahedralization (including segments that are not grazeable). This means that more new vertices must be inserted (by the algorithm of Section 5) to construct a CCDT than for the gift-wrapping or sweep algorithms. However, because many of the extra vertices might be inserted later to achieve high-quality elements, the difference is usually small in the final mesh.

The best way to reason about incremental CDT operations, such as inserting or deleting facets or vertices, is to imagine that both the underlying PLC and the CDT are incrementally modified at the same time. If the underlying PLC remains always edge-protected after each operation, then a CDT exists after each step as well, and the incremental approach is viable.

Imagine a PLC X whose CDT T has been constructed. Let f be a facet we would like to insert into the PLC (and recover in the tetrahedralization). The segments that bound f must already be present in X, and they must appear as edges in T. Let $X^f = X \cup \{f\}$. If X is weakly edge-protected, then so is X^f, because X^f has exactly the same segments and vertices as X. (Facets have no effect on whether a segment is strongly Delaunay.) Therefore, X^f also has a CDT T^f. The goal of a facet insertion algorithm is to convert T into T^f. Such an algorithm is described shortly.

Incremental facet insertion takes advantage of this fact to construct a CDT from an ordinary Delaunay tetrahedralization. To find the CDT of an edge-protected PLC Y, let X be a PLC containing all the vertices and segments of Y, but no facets. (Note that there is no need for the algorithm to actually store X; X is just a mathematical construction for our understanding.) Let T be the Delaunay tetrahedralization of the vertices in X. Because Y is fully edge-protected, every

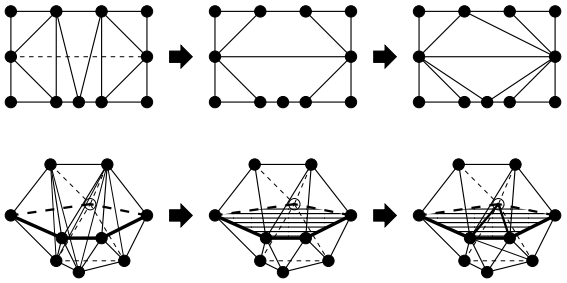

Figure 13: Two- and three-dimensional examples of inserting a facet into a CDT.

segment in X is strongly Delaunay, so T is the CDT of X.

Next, insert the facets of Y into X, one by one. With each facet insertion, update T so it is still the CDT of X. The key observation is that because X always remains edge-protected, this update is always possible. When all the facets have been inserted, $X = Y$ and T is the CDT of Y—or would be if T didn't cover the entire convex hull of the triangulation domain. The final step is to remove from T any tetrahedra that do not lie in the triangulation domain.

Let us consider an algorithm for recovering a facet f. How does T transform into T^f? First, find all the tetrahedra in T that intersect the relative interior of f. It may be that f is already represented as a union of triangular faces, in which case there is nothing to do. Otherwise, the next step is to delete from T each tetrahedron whose interior intersects f, as Figure 13 illustrates. (Tetrahedra that intersect f only on their boundaries stay put.) It is easy to verify that every other tetrahedron is still constrained Delaunay, and so must be present in the new CDT.

Next, use the naïve gift-wrapping algorithm to retriangulate the polygonal cavities created on each side of f. Be forewarned that there may be more than one polygonal cavity on each side of f, because some triangular faces of the tetrahedralization might already conform to f before f is inserted.

For most domains, the polygonal cavities will be bounded by a small number of triangular faces, so the poor time complexity of naïve gift wrapping is unlikely to be a burden. Of course, it is possible to design examples where the cavities have many faces, so the incremental facet insertion algorithm may be even slower than gift wrapping the whole PLC from scratch. Such examples are likely to be the exception in practice.

To retriangulate a cavity, let Z be a PLC consisting only of the triangular faces that bound one cavity, plus their edges and vertices. Happily, the naïve gift-wrapping algorithm works correctly even if some or all of the facets on the convex hull of the region being triangulated are left unspecified. So when a polygonal cavity is triangulated, f may be omitted from the description of the cavity. Therefore, there is no need to precompute the two-dimensional CDT of f before inserting it. When the cavity is tetrahedralized, the two-dimensional CDT of f appears automatically on the surface of the cavity tetrahedralization.

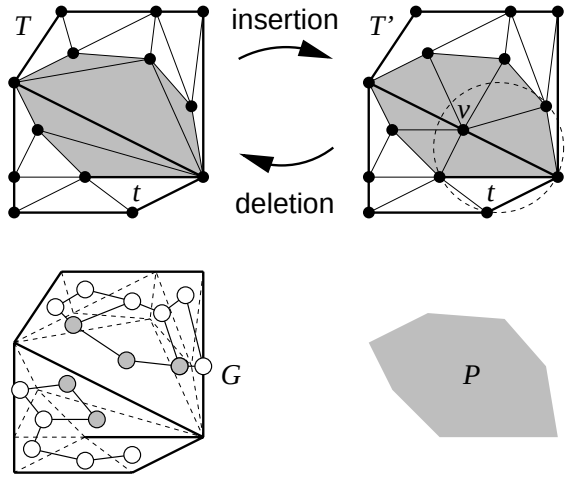

Figure 14: Inserting or deleting a vertex v. This example is two-dimensional for clarity, but the same principles operate in three dimensions. Bold edges are segments. The polygon P is the union of the triangles adjoining v. Triangles outside P are constrained Delaunay in both triangulations. The graph G is used as a search structure to identify P when v is inserted. Note that although v lies inside the circumcircle of t, t is not deleted when v is inserted because G does not connect it to any triangle in P.

8 Vertex Insertion and Deletion

Once a domain has been tetrahedralized with constrained Delaunay elements (collectively forming a CCDT of the input PLC), some of these elements will be of poor quality and need to be improved. Delaunay meshing algorithms [6, 11, 21, 22, 29] insert additional vertices to replace bad elements with better ones, while maintaining the Delaunay (or "almost Delaunay") property throughout the insertions. Fortunately, inserting a vertex into, or deleting a vertex from, a constrained Delaunay tetrahedralization is almost as easy as in an ordinary Delaunay tetrahedralization.

Consider vertex deletion first. Suppose we have the CDT T^v of a PLC X^v. Let X be the PLC obtained by deleting a vertex v from X^v. (Assume that v is not the endpoint of any segment, because endpoints cannot be deleted.) We wish to transform T^v into the CDT T of X.

The first step is to delete every tetrahedron that has v for a vertex. Let P be the union of these tetrahedra. As Figure 14 shows, P is a star-shaped polyhedron whose points are all visible from v. No other tetrahedron is deleted.

The polytope is retriangulated by a simple sweep algorithm for constructing CDTs of star-shaped polyhedra, described elsewhere [23]. The algorithm, based on a similar algorithm of Devillers [5], runs in $\mathcal{O}(n_s \log n_v)$ time, where n_s is the

number of tetrahedra in the new tetrahedralization of P, and n_v is the number of vertices in P.

There is a danger though. What if P has no CDT? Perhaps P is Schönhardt's polyhedron, or another difficult polyhedron. Sometimes vertices cannot be deleted, and it is the responsibility of the application program to ensure that a vertex is safe to delete before deleting it.

The best way to accomplish that is to ensure that the underlying PLC always remains weakly edge-protected after each vertex deletion or insertion. A simple guideline is that if X^v is weakly edge-protected, then so long as v does not lie in a segment, X is also weakly edge-protected, and thus has a CDT. However, if v is deleted from the interior of a segment, the deletion merges two segments into one. The deletion is safe only if the new, larger segment is known to be strongly Delaunay. For example, let X be the PLC in Figure 9, let v be a vertex in the middle of the central segment, and let $X^v = X \cup \{v\}$. X^v has a CDT T^v from which v cannot be safely deleted.

Vertex insertion is easier to implement than vertex deletion, because no sweep algorithm is needed. The insertion algorithm is just a slight variation of the Bowyer–Watson algorithm for vertex insertion in Delaunay triangulations [1, 28].

Vertex insertion is the reverse of vertex deletion. Given a CDT T of a PLC X and a vertex v, the goal is to construct the CDT T^v of $X^v = X \cup \{v\}$, if it exists. The first task is to identify P, the polyhedron to be retriangulated. Unfortunately, while it is easy to determine P from T^v, it is harder to determine P from T.

First, find the tetrahedra that contain v. (Typically there is just one, but there are more if v lies on a face or edge of T.) These tetrahedra are no longer constrained Delaunay, so they must be deleted. P is connected, so the other tetrahedra in P can be found by a depth-first search from the tetrahedra that contain v. The search does not walk through constraining triangles, and it backtracks whenever it encounters a tetrahedron whose circumsphere does not enclose v. The search finds every simplex that is no longer constrained Delaunay.

On what graph is the search performed? Let G be a graph with one node for each tetrahedron. An edge connects two nodes if the corresponding tetrahedra share a triangular face that is *not* a constraining triangle.

After all the tetrahedra in P are identified and deleted, P is retetrahedralized to yield T^v. The new tetrahedralization is easy to generate: for each triangular face s in the boundary of P, construct a new tetrahedron $\mathrm{conv}(s \cup v)$, as Figure 14 illustrates. The total running time is $\mathcal{O}(n_s)$, where n_s is the number of old tetrahedra deleted. (The number of new tetrahedra created is never larger than $2n_s + 2$, though it could be asymptotically smaller.) This running time does not include the time needed to find a tetrahedron that contains v.

Like vertex deletion, vertex insertion is not always safe. A new vertex may cause a segment to no longer be strongly Delaunay, and knock it out of the tetrahedralization. A Delaunay mesh generation program should test for this possibility, and back out when vertex insertion cannot succeed.

A three-dimensional Delaunay refinement algorithm I describe elsewhere [21] handles this difficulty by considering the diametral sphere of each segment (the smallest sphere that contains the segment) to be a protected region. Any effort to insert a vertex on or inside the sphere is thwarted; instead, the segment is split at its center, and the two resulting subsegments have smaller diametral spheres. Thus, every subsegment is kept strongly Delaunay at all times, and the mathematical integrity of the CDT is never broached.

Unfortunately, there is one circumstance where this strategy does not work. A vertex inserted on one grazeable segment might knock a second grazeable segment out of the mesh, and a vertex on the second segment might knock the first one out. This is a chicken-and-egg problem where both vertices—perhaps many vertices—must be inserted simultaneously to keep the mesh weakly edge-protected. Simultaneous vertex insertions can be performed by identifying and removing all the tetrahedra that would be removed by individual vertex insertions, then gift wrapping the cavities.

9 Degeneracies and Robustness

Unfortunately, CDT construction algorithms are even more sensitive to numerical error than most geometric algorithms. Furthermore, PLCs in which five or more vertices lie on a common sphere can cause gift wrapping to fail if degeneracies are not handled carefully.

Therefore, I recommend that exact arithmetic [19] be used to implement the numerical predicates (i.e. orientation and insphere tests), and that all the algorithms use a modified insphere test that employs a simple symbolic perturbation technique [9, 23] to simulate the circumstance where no five vertices are cospherical. Symbolic perturbation ensures that the gift-wrapping and sweep algorithms correctly produce a CDT of any weakly-edge protected PLC, and simplifies the test for strongly Delaunay segments (as discussed in Section 4).

References

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

[2] L. Paul Chew. *Constrained Delaunay Triangulations*. Algorithmica **4**(1):97–108, 1989.

[3] Kenneth L. Clarkson and Peter W. Shor. *Applications of Random Sampling in Computational Geometry, II*. Discrete & Computational Geometry **4**(1):387–421, 1989.

[4] David Cohen-Steiner, Éric Colin de Verdière, and Mariette Yvinec. *Conforming Delaunay Triangulations in 3D*. Proceedings of the Eighteenth Annual Symposium on Computational Geometry (Barcelona, Spain), pages 199–208, June 2002.

[5] Olivier Devillers. *On Deletion in Delaunay Triangulations*. Proceedings of the Fifteenth Annual Symposium on Computational Geometry, pages 181–188. Association for Computing Machinery, June 1999.

[6] 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.

[7] Rex A. Dwyer. *Higher-Dimensional Voronoi Diagrams in Linear Expected Time*. Discrete & Computational Geometry **6**(4):343–367, 1991.

[8] Herbert Edelsbrunner, Xiang-Yang Li, Gary Miller, Andreas Stathopoulos, Dafna Talmor, Shang-Hua Teng, Alper Ungor, and Noel Walkington. *Smoothing and Cleaning Up Slivers*. Proceedings of the 32nd Annual Symposium on the Theory of Computing (Portland, Oregon), pages 273–278. Association for Computing Machinery, May 2000.

[9] Herbert Edelsbrunner and Ernst Peter Mücke. *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms*. ACM Transactions on Graphics **9**(1):66–104, 1990.

[10] Herbert Edelsbrunner and Tiow Seng Tan. *An Upper Bound for Conforming Delaunay Triangulations*. Discrete & Computational Geometry **10**(2):197–213, 1993.

[11] Paul-Louis George and Houman Borouchaki. *Delaunay Triangulation and Meshing: Application to Finite Elements*. Hermès, Paris, 1998.

[12] Carol Hazlewood. *Approximating Constrained Tetrahedrizations*. Computer Aided Geometric Design **10**:67–87, 1993.

[13] Der-Tsai Lee and A. K. Lin. *Generalized Delaunay Triangulations for Planar Graphs*. Discrete & Computational Geometry **1**:201–217, 1986.

[14] 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 (Las Vegas, Nevada), pages 683–692, May 1995.

[15] 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 (Pittsburgh, Pennsylvania), pages 47–61, October 1996.

[16] Michael Murphy, David M. Mount, and Carl W. Gable. *A Point-Placement Strategy for Conforming Delaunay Tetrahedralization*. Proceedings of the Eleventh Annual Symposium on Discrete Algorithms, pages 67–74. Association for Computing Machinery, January 2000.

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

[18] E. Schönhardt. *Über die Zerlegung von Dreieckspolyedern in Tetraeder*. Mathematische Annalen **98**:309–312, 1928.

[19] Jonathan Richard Shewchuk. *Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates*. Discrete & Computational Geometry **18**(3):305–363, October 1997.

[20] ———. *A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 76–85. Association for Computing Machinery, June 1998.

[21] ———. *Tetrahedral Mesh Generation by Delaunay Refinement*. Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pages 86–95. Association for Computing Machinery, June 1998.

[22] ———. *Mesh Generation for Domains with Small Angles*. Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong), pages 1–10. Association for Computing Machinery, June 2000.

[23] ———. *Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations*. Proceedings of the Sixteenth Annual Symposium on Computational Geometry (Hong Kong), pages 350–359. Association for Computing Machinery, June 2000.

[24] ———. *What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures*. This proceedings, 2002.

[25] Robin Sibson. *A Brief Description of Natural Neighbor Interpolation*. Interpreting Multivariate Data (V. Barnett, editor), pages 22–36. John Wiley, 1981.

[26] G. Swart. *Finding the Convex Hull Facet by Facet*. Journal of Algorithms **6**:17–48, 1985.

[27] M. Tanemura, T. Ogawa, and N. Ogita. *A New Algorithm for Three-Dimensional Voronoi Tessellation*. Journal of Computational Physics **51**:191–207, 1983.

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

[29] Nigel P. Weatherill and O. Hassan. *Efficient Three-Dimensional Grid Generation using the Delaunay Triangulation*. Proceedings of the First European Computational Fluid Dynamics Conference (Brussels, Belgium) (Ch. Hirsch, J. Périaux, and W. Kordulla, editors), pages 961–968, September 1992.