Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations

Jonathan Richard Shewchuk
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720
jrs@cs.berkeley.edu

Abstract

I discuss algorithms for constructing constrained Delaunay triangulations (CDTs) in dimensions higher than two. If the CDT of a set of vertices and constraining simplices exists, it can be constructed in $\mathcal{O}(n_v n_s)$ time, where n_v is the number of input vertices and n_s is the number of output d-simplices. In practice, the running time is likely to be $\mathcal{O}(n_v^2 + n_s \log n_v)$ in all but the most pathological cases. The CDT of a star-shaped polytope can be constructed in $\mathcal{O}(n_s \log n_v)$ time, yielding an efficient way to delete a vertex from a CDT.

1. Introduction

Mesh generation and interpolation can benefit from triangulations that have properties similar to Delaunay triangulations, but are constrained to contain specified faces. These constraints may arise because a mesh must conform to the shape of an object, or because of the desire to interpolate a discontinuous function.

The *constrained Delaunay triangulation* (CDT) [6, 1, 12] is a Delaunay-like triangulation that conforms to constraints. In two dimensions, the input is a *planar straight line graph* (PSLG) X, which is a set of vertices and segments (constraining edges) illustrated in Figure 1 (left). A PSLG is required to contain both endpoints of every segment it contains, and a segment may intersect vertices and other segments only at its endpoints. The CDT of X (Figure 1, right) is a specific triangulation whose vertices are the vertices in X, and whose edges include the segments in X.

CDTs have been generalized to dimensions higher than two in a companion paper to this one [12]. The input is a *piecewise linear complex* (PLC), following Miller, Talmor, Teng, Walkington, and Wang [7]. A PLC X is a set of vertices and *constraining facets*—polytopes that may be nonconvex and may have holes, slits, or dangling vertices inside them, as Figure 2 illustrates.

Let d be the dimensionality of the input. Constraining facets may be of any dimension from 1 to $d-1$, but for reasons explained in the companion paper, there is no need to consider PLCs in their full generality here. Instead, assume that X is a collection of vertices and $(d-1)$-dimensional simplices—henceforth, $(d-1)$-simplices—which are constrained to be faces of the CDT. These *constraining simplices* may be obtained by recursively finding the CDT of each $(d-1)$-dimensional constraining facet. A PLC is a complex: if X contains a $(d-1)$-simplex s, then X must contain every vertex of s. (A complex also contains faces of intermediate dimensionality, but we may ignore these.) Any two $(d-1)$-simplices of X may intersect only at a shared lower-dimensional face (possibly a vertex).

A CDT, despite its name, is not a Delaunay triangulation. In an ordinary Delaunay triangulation, every simplex (of any dimensionality) is *Delaunay*. In a CDT, this requirement is waived, and instead every simplex must either be a constraining simplex (or face thereof) specified in X, or be *constrained Delaunay*.

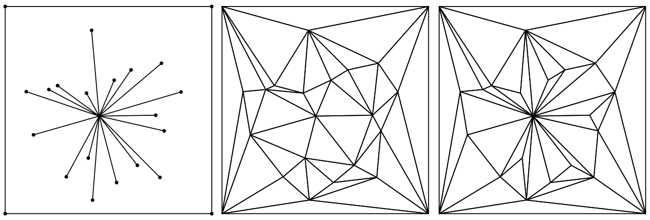

Figure 1: The Delaunay triangulation (center) of the vertices of a PSLG (left) might not include the segments of the PSLG. These segments can be incorporated by forgoing Delaunay triangles in favor of constrained Delaunay triangles (right).

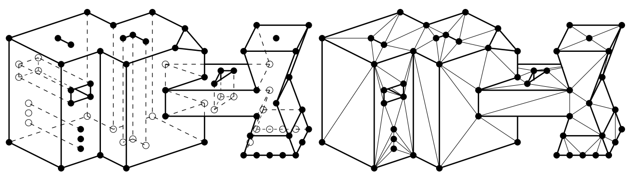

Figure 2: Any facet of a PLC (left) may contain holes, slits, and vertices, some of which may support intersections with other facets. The illustration at right is the constrained Delaunay tetrahedralization of the region bounded by the facets of the PLC at left.

Supported in part by the National Science Foundation under Awards ACI-9875170, CMS-9980063, and EIA-9802069, and by gifts from the Okawa Foundation and Intel. The information presented here is not endorsed by, and does not necessarily reflect the position or policies of, the U.S. Government or other sponsors.

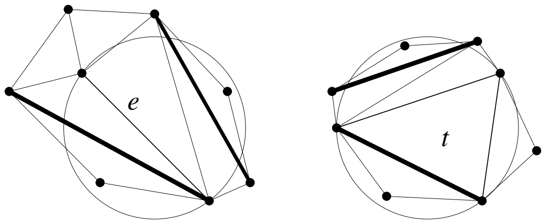

Figure 3: The edge e and the triangle t are each constrained Delaunay. Bold lines represent segments.

Algorithm	Running time
Naïve gift-wrapping	$\mathcal{O}(n_v n_f n_s)$
SweepCDT	$\mathcal{O}(n_v n_s)$
SweepCDT with linear programming	$\mathcal{O}(\min\{n_v n_s, n_v^2 n_f + n_s \log n_v\})$
CDT of a star-shaped polytope	$\mathcal{O}(n_s \log n_v)$

Figure 4: CDT construction algorithms discussed in this paper. n_v is the number of vertices, n_f is the number of constraining simplices, and n_s is the output size.

Let s be any k-simplex (for any k) whose vertices are in X (but s is not necessarily a constraining simplex in X). Let S be a (full-dimensional) sphere in E^d; S is a *circumsphere* of s if S passes through all the vertices of s. If $k = d$, then s has a unique circumsphere; otherwise, s has infinitely many circumspheres. 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). Every 0-simplex (vertex) is trivially Delaunay.

Say that the visibility between two points p and q in E^d is *occluded* if there is a constraining $(d-1)$-simplex t in X such that p and q lie on opposite sides of the hyperplane that contains t, and the line segment pq intersects t (either in the boundary or in the relative interior of t). If either p or q lies in the hyperplane containing t, then t does not occlude the visibility between them. The points p and q are *visible* from each other if there is no occluding $(d-1)$-simplex in X.

A simplex s is *constrained Delaunay* if

- no constraining simplex in X intersects the relative interior of s unless it contains s in its entirety, 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 3 demonstrates examples of a constrained Delaunay edge e and a constrained Delaunay triangle t in two dimensions. Input segments appear as bold lines. Although there is no circumcircle of e that encloses no vertex, 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 *triangulation domain* is the portion of E^d that a user wishes to triangulate. If the triangulation domain is the convex hull of X, a *constrained Delaunay triangulation* of X is any triangulation of the vertices of X whose d-simplices are all constrained Delaunay. Otherwise, X must be *facet-bounded*, meaning that $(d-1)$-simplices of X entirely cover the boundary separating the triangulation domain from its complement, the *exterior domain*. In this case, a *constrained Delaunay triangulation* of X is a triangulation composed of constrained Delaunay d-simplices that cover the triangulation domain. Each $(d-1)$-simplex that separates the triangulation domain from the exterior domain should bear a notation that indicates which side of it adjoins the exterior domain. (The ability to specify a triangulation domain can be crucial, because there are PLCs for which a CDT of the triangulation domain exists but a CDT of its convex hull does not.) Some constraining simplices may have the triangulation domain on both sides. Such simplices allow PLCs to represent non-manifold and multiple-component domains.

Although the d-simplices of a CDT are not Delaunay, 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 [6]. Three-dimensional CDTs can help improve the quality of meshes produced by tetrahedral mesh generation algorithms [11, 13].

Unfortunately, not every legal set of vertices and constraining simplices has a CDT. One reason is that in three or more dimensions, there are polytopes that cannot be triangulated at all without additional vertices. The first of these to be discovered is a three-dimensional example by Schönhardt [9]. Ruppert and Seidel [8] prove that it is NP-hard to determine whether a polyhedron is tetrahedralizable. This paper demonstrates that determining whether a d-dimensional polytope has a CDT is (probably) easier, because it can be done in polynomial time by attempting to construct the CDT. The companion paper to this one [12] offers another useful and more easily-tested condition that guarantees that a CDT exists. (The condition discussed therein is sufficient, but not necessary.)

The main result of this paper is a family of algorithms for finding the CDT of any PLC that has one. These algorithms, summarized in Figure 4, include a fast special-case algorithm for finding the CDT of a star-shaped polytope, which is useful for retriangulating the region evacuated when a vertex is deleted from a CDT. The only previously known algorithm for constructing CDTs is gift-wrapping, which was first applied to higher-dimensional CDTs in the companion paper [12] and is briefly revisited in Section 2. The running time of gift-wrapping is $\mathcal{O}(n_v n_f n_s)$, where n_v is the number of vertices in X, n_f is the number of constraining $(d-1)$-simplices in X, and n_s is the number of d-simplices in the output. A new sweep algorithm presented in Section 3, SweepCDT, constructs the CDT in $\mathcal{O}(n_v n_s)$ time, and is likely to exhibit $\mathcal{O}(n_v^2 + n_s \log n_v)$ or better behavior in practice. In two dimensions, Chew's $\mathcal{O}(n_v \log n_v)$ algorithm [1] is preferable, but SweepCDT is the best existing option in higher dimensions. The running time of SweepCDT becomes $\mathcal{O}(\min\{n_v n_s, n_v^2 n_f + n_s \log n_v\})$ if linear programming is used to construct a carefully chosen fraction of the constrained Delaunay d-simplices. This adjustment, described briefly in Section 7, is an improvement if the output size is unusually large ($n_s \gg n_v n_f$).

Throughout this paper, the terms "simplex," "triangle," and "convex hull" refer to closed, convex sets of points; hence, they include all the points on their boundaries and in their interiors. The notation $\mathrm{conv}(S)$ represents the convex hull of the point set S. A *face* of a k-simplex is the convex hull of any subset of the vertices of the simplex, and may be of any dimension less than or equal to k. Some faces of specific dimensions have their own names: a *hyperface* is a $(d-1)$-face, and a *ridge* is a $(d-2)$-face. A vertex is a 0-face, an edge is a 1-face, and a triangle is a 2-face.

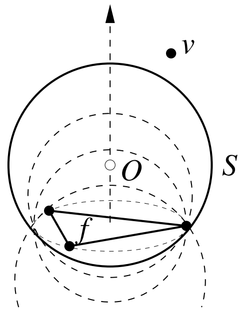

Figure 5: A sphere S that circumscribes f, expanding in search of another vertex v.

2. Constructing a CDT by Gift-Wrapping

Constrained Delaunay triangulations can be constructed by a modification of a well-known algorithm for computing ordinary Delaunay triangulations [14, 3], variously called *gift-wrapping*, *graph traversal*, *pivoting*, or *incremental search*. Naïvely gift-wrapping a CDT takes $\mathcal{O}(n_v n_f n_s)$ time. A more complicated algorithm in Section 3 improves the running time to $\mathcal{O}(n_v n_s)$. Both algorithms compute the CDT of any PLC X that has a CDT, if no $d+2$ vertices of X are cospherical. If the latter condition is not satisfied, perturbation methods outlined in Section 6 can usually ensure that a CDT is produced.

Gift-wrapping begins by constructing a single constrained Delaunay d-simplex Δ, which is used as a *seed* upon which the remaining constrained Delaunay d-simplices crystallize one by one. Each hyperface of a constrained Delaunay d-simplex is used as a base from which to search for the vertex that serves as the apex of an adjacent d-simplex.

Gift-wrapping is based on a straightforward procedure for "growing" a d-simplex from a $(d-1)$-simplex, illustrated in Figure 5. Let f be a constraining $(d-1)$-simplex or a hyperface of a constrained Delaunay d-simplex. Assume without loss of generality that f is oriented horizontally, and the constrained Delaunay d-simplex immediately above f is sought. Suppose that at least one vertex of X lies above f. Let S be a 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 a vertex v that finishes f.

Let v be the first vertex above f touched by the expanding sphere such that the interior of the d-simplex $s = \text{conv}(f \cup v)$ intersects no constraining facet of X. If a CDT of X exists, then s is constrained Delaunay. (The proof of this claim will appear in a longer version of this paper.) If no such vertex can be found, or if s is not constrained Delaunay, then X has no CDT.

The simplex growth procedure may be used both to construct the seed and to crystallize the remaining d-simplices. To find a seed Δ, let f be an arbitrary constraining $(d-1)$-simplex of X, and choose the direction of sphere movement so the triangulation domain is immediately "above" f. Thereafter, say that a hyperface of the CDT is *unfinished* if the algorithm has not yet identified the second constrained Delaunay d-simplex that shares the hyperface. To *finish* a hyperface is to construct the second simplex, or to determine that the hyperface adjoins the exterior domain and there is no second simplex.

The gift-wrapping algorithm maintains a dictionary of unfinished hyperfaces, which initially contains the $d+1$ hyperfaces of Δ. Repeat the following steps: remove an arbitrary unfinished hyperface 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, v is found through the growth procedure and $s = \text{conv}(f \cup v)$ is added to the growing triangulation. Check each hyperface of s, except f, against the dictionary. If a hyperface is already present in the dictionary, then the face is now finished, so remove it from the dictionary. Otherwise, the face is new, so insert it into the dictionary.

Finishing a single hyperface takes $\mathcal{O}(n_v n_f)$ time, because one must test the visibility of each vertex from the hyperface, which is done by testing each vertex against each constraining $(d-1)$-simplex in X. Gift-wrapping thus runs in $\mathcal{O}(n_v n_f n_s)$ time. This leaves much room for improvement, some of which will be realized in the next section.

3. Sweep Algorithms for Constructing CDTs

The $\mathcal{O}(n_v n_f n_s)$ running time of gift-wrapping can be improved to $\mathcal{O}(n_v n_s)$—or even to $\mathcal{O}(\min\{n_v n_s, n_v^2 n_f + n_s \log n_v\})$, which is faster if the output size n_s is significantly larger than $n_v n_f$. In the special case of triangulating a star-shaped polygon, gift-wrapping takes only $\mathcal{O}(n_s \log n_v)$ time. These running times are achieved by space-sweep algorithms closely related to a convex hull algorithm of Seidel [10]. The algorithm for vertex deletion was proposed by Devillers [2] for Delaunay triangulations; here I show that it is correct for CDTs as well.

The sweep algorithms come in two versions: one in which space is swept by a hyperplane, and one in which it is swept by a hypersphere. The versions are for the most part quite similar, and will be discussed simultaneously, but each has advantages of its own. The plane version is easier to understand and employs simpler geometric predicates, but the sphere version is best for inputs for which the boundary is unknown and for deleting a vertex from a CDT.

Let $x_1, x_2, \ldots, x_d$ be the coordinate axes. The sweep-plane version employs a moving hyperplane $x_1 = \gamma$, where γ sweeps from $-\infty$ to ∞. As Figure 6 illustrates, each d-simplex of the CDT is constructed when the hyperplane sweeps across its *circumcenter*—the center of its circumsphere. (Readers familiar with Fortune's sweepline algorithm for constructing two-dimensional Delaunay triangulations [5] will notice a key difference: Fortune's algorithm constructs a triangle only when the sweepline has passed entirely over its circumcircle.) Because several circumcenters may share the same x_1 coordinate, they are swept in *lexicographic order*. Lexicographic order is analogous to alphabetical order; it sorts points according to their x_1-coordinates, using x_2-coordinates to break ties, then x_3-coordinates and so on.

In an ordinary Delaunay triangulation, a consequence of constructing d-simplices in this order is that the growing triangulation is connected, but this is not necessarily true of a CDT. As Figure 6 reveals, temporarily isolated groups of simplices can form upon constraining $(d-1)$-simplices. Constraining simplices appear as bold lines in the illustration.

Let h be an arbitrary hyperplane, which partitions space into two halfspaces. If h cuts the x_1-axis into two rays, then a point p is said to lie *behind* h if it lies to the left of h—that is, on the same side

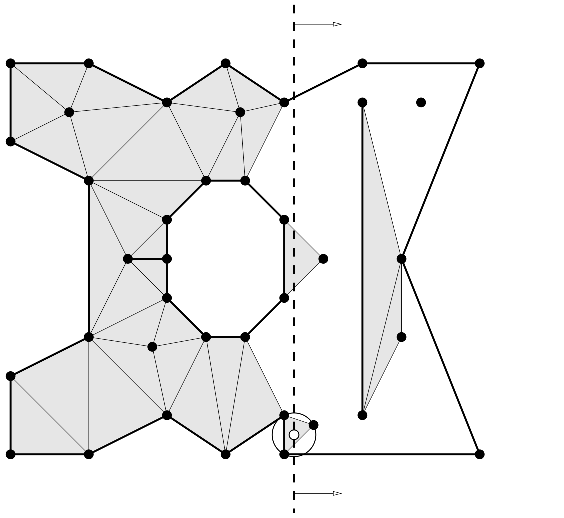

Figure 6: Illustration of the sweep hyperplane (dashed line) in two dimensions. Bold lines are constraining segments. Each d-simplex is created when the hyperplane sweeps over its circumcenter.

of h as the point $(-\infty, 0, \dots, 0)$—and p lies *in front of* h if it is on the same side of h as $(\infty, 0, \dots, 0)$. However, if h is parallel to the x_1-axis, use the x_2-axis to break ties: p is behind h if it lies on the same side of h as $(0, -\infty, 0, \dots, 0)$. If h does not cut the x_2-axis either, try the x_3-axis, and so on. Hence, every hyperplane has a behind and a front.

Let f be an arbitrary $(d-1)$-simplex. A point p is *behind* f if p is behind the hyperplane containing f, and *in front of* f if p is in front of that hyperplane.

There is a simple intuition governing the order in which adjacent d-simplices are constructed: if we shoot any ray parallel to the x_1-axis, directed from left to right, the d-simplices that the ray encounters are constructed in the order they are encountered, until the ray strikes a constraining $(d-1)$-simplex. The following theorem confirms this fact.

THEOREM 1 (FOR SWEEP HYPERPLANES). *For any hyperface f of the CDT that is not a constraining simplex, let s_1 and s_2 be the two constrained Delaunay d-simplices that contain f (if they both exist), with s_1 behind f and s_2 in front of f. Let c_1 and c_2 be the circumcenters of s_1 and s_2, respectively. Then c_1 is lexicographically less than c_2, so s_1 is constructed before s_2. (However, if f is a constraining simplex that is* not *constrained Delaunay, this result does not apply.)*

Proof: The idea is perhaps best understood by looking at the Voronoi diagram of the vertices of s_1 and s_2 (Figure 7). The Voronoi edge extending from c_1 to c_2 is orthogonal to f, because c_1 and c_2 are each equidistant from all the vertices of f. Because the Voronoi edge is orthogonal to f, the points increase lexicographically as one moves along the edge from behind f to the front. Let v_1 be the vertex of s_1 opposite f; v_1 is clearly behind f. Because s_1 and s_2 are constrained Delaunay, v_1 is not enclosed by the circumsphere of s_2. By assumption, no $d+2$ vertices of X are cospherical; hence, v_1 cannot lie on the circumsphere of s_2, either. It follows that the circumsphere of s_1 extends further behind f than the circumsphere of s_2, and therefore c_1 is lexicographically less than c_2. ■

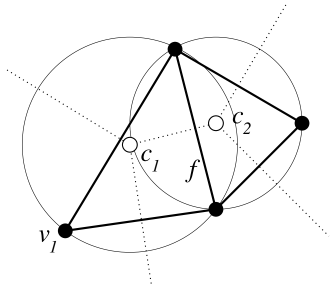

Figure 7: Two constrained Delaunay d-simplices and their Voronoi dual.

The sweep hyperplane algorithm is a good choice if the input PLC X is facet-bounded. However, it is inconvenient if X is a jumble of vertices and simplices with no airtight boundary, and the algorithm is asked to triangulate the convex hull of X. To get started, the sweep plane algorithm needs to know the leftward-facing boundary of the convex hull of X (more precisely, the set of hyperfaces of the convex hull whose exteriors are their behinds). Furthermore, the left boundary must be triangulated in a manner constrained by the constraining facets of X. If such a triangulation is not readily available, an expanding sweep hypersphere is preferable to a sweep hyperplane because the hypersphere computes the boundary of the convex hull automatically as a side effect of triangulating the interior, with no additional programming effort.

To seed the expanding sphere, begin by growing a single constrained Delaunay d-simplex Δ, as described in Section 2. Let $\Delta_0, \Delta_1, \dots, \Delta_d$ be the vertices of Δ.

For any d-simplex s, let S_s be the circumsphere of s, and let O_s and r_s be the center and radius of S_s, respectively. The d-simplices of the CDT (other than Δ) are constructed in the order dictated by the *power function* $\Psi_{\Delta_0}(s) = |\Delta_0 O_s|^2 - r_s^2$ defined for each d-simplex s relative to the vantage point Δ_0 (where $|pq|$ is the Euclidean distance between p and q).

Intuitively, the order of simplex construction is determined by an expanding hypersphere centered on Δ_0. For any d-simplex s of the CDT, let p be any point on the circumsphere of s such that the ray $\Delta_0 p$ is tangent to the circumsphere, as illustrated in Figure 8. The simplex s is constructed when the expanding sphere sweeps across p, which occurs when the square of the radius of the sphere is $\Psi_{\Delta_0}(s)$. (This intuition fails for constrained Delaunay d-simplices whose circumspheres enclose Δ_0; these have negative power.) As with the sweep plane variant, temporarily isolated groups of simplices can form upon constraining $(d-1)$-simplices.

Several simplices might have the same power. So that ties may be broken, each simplex s is assigned a power-tuple $\Psi_\Delta(s) = \langle \Psi_{\Delta_0}(s), \Psi_{\Delta_1}(s), \dots, \Psi_{\Delta_d}(s) \rangle$, and simplices are constructed in lexicographic order according to their power-tuples. (It is not really necessary to use power-tuples, and an implementation may safely use only Ψ_{Δ_0}. However, power-tuples simplify the proofs behind the algorithms by ensuring that pairs of d-simplices are unambiguously ordered.)

Naturally, a sweep sphere entails different definitions of "behind" and "in front of." Let f be a $(d-1)$-simplex, and let h be the hyperplane containing f. The halfspace (bounded by h) containing Δ_0 is said to be *behind* f (and h), and the other halfspace is said

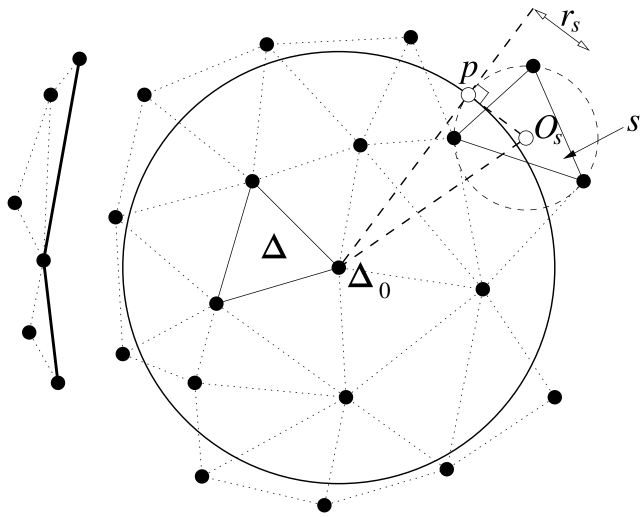

Figure 8: Illustration of the sweep sphere in two dimensions. Bold lines at left are constraining segments. When the sphere has expanded to the size depicted, all the illustrated triangles have been constructed. The simplex s is created when the expanding sphere sweeps over p, which occurs when the square of the radius of the sweep sphere is $|\Delta_0 O_s|^2 - r_s^2$.

to be *in front of* f. If Δ_0 lies in h, then the halfspace containing Δ_1 is said to be *behind* f. If Δ_1 also lies in h, ties are broken by consulting Δ_2, Δ_3, and so on. At least one vertex of Δ does not lie in h.

As with sweep hyperplanes, there is an intuition governing the order in which adjacent d-simplices are constructed: if we shoot any ray directed away from Δ_0, the d-simplices that the ray encounters are constructed in the order they are encountered, until the ray strikes a constraining $(d-1)$-simplex.

THEOREM 2 (FOR SWEEP HYPERSPHERES). *For any hyperface f of the CDT that is not a constraining simplex, let s_1 and s_2 be the two constrained Delaunay d-simplices that contain f (if they both exist), with s_1 behind f and s_2 in front of f. Then $\Psi_\Delta(s_1)$ lexicographically precedes $\Psi_\Delta(s_2)$, so s_1 is constructed before s_2.*

The proof of this theorem relies on a proof in the companion paper. Let p be an arbitrary vantage point in E^d (such as Δ_0). Let s and t be any two d-simplices. Say that s *overlaps* t from the viewpoint p if some point of s not shared by t lies directly between p and t. In other words, there exists a point p_s of s and a point p_t of t such that $p_s \notin t$ and p_s lies between p and p_t. Say that s *parallels* t from the viewpoint p if there exists a line that passes through p and at least two points of $s \cap t$. In other words, the affine hull of $s \cap t$ contains p.

LEMMA 3. *Let s and t be two d-simplices. Suppose that no vertex of t, except vertices common to s and t, lies on or inside the circumsphere of s, and no vertex of s not shared with t lies on or inside the circumsphere of t. If s overlaps t from the viewpoint p, then $\Psi_p(s) < \Psi_p(t)$. If s parallels t from the viewpoint p, then $\Psi_p(s) = \Psi_p(t)$.*

Proof: See Shewchuk [12], Lemma 3. ■

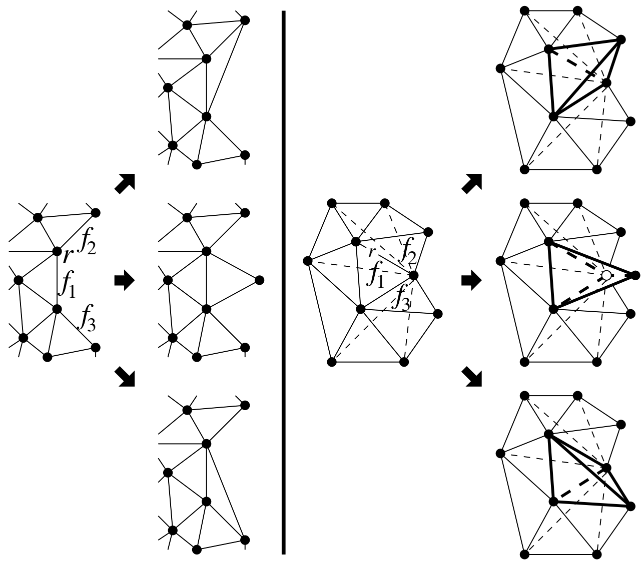

Figure 9: Two examples of finding the apex of a Delaunay d-simplex. In the two-dimensional example at left, ridges (such as r) are vertices. If the Delaunay triangle in front of f_1 is also the Delaunay triangle in front of f_2 (or f_3), the apical vertex of the triangle can be identified quickly using the structure of the incomplete triangulation. Of course, it is possible that the apical vertex is not yet part of the triangulation. In the three-dimensional example at right, ridges are edges. Because r is a reflex edge, the apex of the Delaunay tetrahedron atop f_1 might be identified by examining f_2.

Proof of Theorem 2: By the definition of "behind," Δ_0 is either behind f or coplanar with the hyperface f. If Δ_0 is behind f, then any line through Δ_0 and the interior of f is witness to the fact that s_1 overlaps s_2 from the viewpoint Δ_0, so $\Psi_{\Delta_0}(s_1) < \Psi_{\Delta_0}(s_2)$ by Lemma 3. If Δ_0 lies in the hyperplane containing f, then s_1 parallels s_2 from the viewpoint Δ_0, so $\Psi_{\Delta_0}(s_1) = \Psi_{\Delta_0}(s_2)$. In this case, we repeat the argument with Δ_1, and so on until we reach a vertex of Δ that is not coplanar with f, thereby verifying that $\Psi_\Delta(s_1)$ precedes $\Psi_\Delta(s_2)$ lexicographically. ■

The sweep plane and sweep sphere algorithms, like naïve gift-wrapping, maintain a set of unfinished faces, but only the front of a hyperface can be considered unfinished. The face f (in Theorems 1 and 2) is *finished* if s_2 has been constructed, or if f is adjoins the exterior domain and is notated to indicate that s_2 does not exist. f is *unfinished* otherwise. A consequence of Theorems 1 and 2 is that the face f is first constructed when s_1 is constructed, unless f is a constraining simplex (or a leftward-facing boundary face, in the sweep plane algorithm). If f is a constraining simplex, ignore the possibility that the behind of f is not finished; the simplex s_1 will take care of itself.

An observation made in different forms by Seidel [10] and Fortune [5] makes it possible to improve the speed of gift-wrapping. Let f_1 be an unfinished hyperface of the CDT, for which we seek the finishing d-simplex s. Let f_2 be another unfinished face that shares a common ridge r with f_1, as illustrated in Figure 9. If r is a reflex ridge—in other words, the front angle separating f_1 from f_2 is less than 180°—then s might also finish f_2, in which case the identities of all the vertices of s are available without the need for an expensive search step.

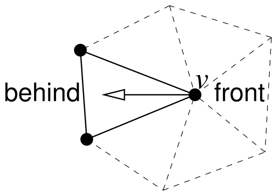

Figure 10: A triangle that is in front of one of its edges, and behind its two edges incident on v. A ray shot behind v passes through this triangle first. No other triangle incident on v (dashed edges) has these properties.

This observation can help identify any simplex s that is in front of at least two of its hyperfaces, because those faces will be constructed before s. However, the idea cannot be used to construct a d-simplex whose apex has not yet appeared in the growing triangulation. Every vertex makes a first appearance once, so there are up to n_v such simplices.

Let us characterize the simplices we seek. Consider that every d-simplex (except Δ) is in front of at least one of its hyperfaces; some face looks toward the point $(-\infty, 0, \ldots, 0)$ if a sweep plane is employed, or toward the point Δ_0 (or Δ_1 if Δ_0 is a vertex of the simplex, or $\Delta_2 \ldots$) if a sweep sphere is used. Any d-simplex in front of two or more of its hyperfaces can be identified as specified above. Hence, each hard-to-identify d-simplex is in front of exactly one of its hyperfaces, and is therefore behind its other d hyperfaces.

For each vertex $v \in X - \{\Delta_0, \Delta_1, \ldots, \Delta_d\}$, let s_v be the unique constrained Delaunay d-simplex that is behind all d of its hyperfaces incident on v. Intuitively, s_v is the first d-simplex encountered by a ray shot from v parallel to the negative x_1-axis (for a sweep plane) or toward Δ_0 (for a sweep sphere), as Figure 10 illustrates. (Ties must be broken lexicographically, so it is more precise to characterize s_v in terms of which faces it is behind.)

Let f_v be the hyperface opposite v in s_v; v and s_v are in front of f_v, and f_v is the only hyperface of s_v that looks left (for a sweep plane) or toward Δ_0 (for a sweep sphere). At some time, f_v will be constructed, either because the d-simplex immediately behind f_v is constructed, or because f_v is a constraining simplex. A successful sweep algorithm must recognize that f_v is the face that v finishes. A simple method for doing so is described here.

Let the *drop face* of a vertex v be the first hyperface of the triangulation data structure (in its current state) encountered by an open ray shot from v parallel to the negative x_1-axis (for a sweep plane) or toward Δ_0 (for a sweep sphere), as illustrated in Figure 11. Lexicographic rules are enforced: if the ray strikes the boundary of a hyperface f, the collision counts only if the d-simplex $\mathrm{conv}(f \cup v)$ is nondegenerate and is behind all d of its hyperfaces incident on v; otherwise, the ray continues on. In the sweep plane variant a vertex might not have a drop face if the vertex is on a left-facing exterior boundary. In the sweep sphere variant the vertices of Δ do not have drop faces. Every other vertex has a drop face, which is initially either a constraining simplex or a face of Δ. As the triangulation grows, the drop face of each vertex may change repeatedly, and the algorithm updates its identity (until the vertex enters the triangulation). However, the algorithm stores a mapping from unfinished faces to vertices, not vice versa.

Here is a description of the sweep algorithm. The first step—after constructing Δ, if a sweep sphere is used—is to build a *ridge dictionary* and use it to begin building the triangulation data structure. The triangulation data structure is composed of three simple record types: one that represents hyperfaces of the growing triangulation, one that represents ridges, and one that represents vertices, as illustrated in Figure 12. Each ridge maintains a circularly-, doubly-linked list of the hyperfaces incident to it that have been constructed thus far. The pointers for each linked list should be maintained in the hyperfaces themselves. Each hyperface is on d linked lists—one for each of its ridges—employing $2d$ pointers per face. Each hyperface also maintains pointers to its own vertices and a singly-linked list of the vertices for which it is the drop face, henceforth called its *drop list*. There is no need to directly represent d-simplices; these can be recovered easily from the final configuration of hyperfaces.

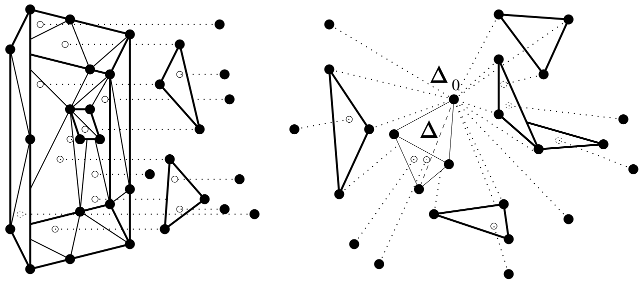

Figure 11: Finding the drop face of a vertex. If no face stands between a vertex and Δ_0, the vertex's drop face is a face of Δ.

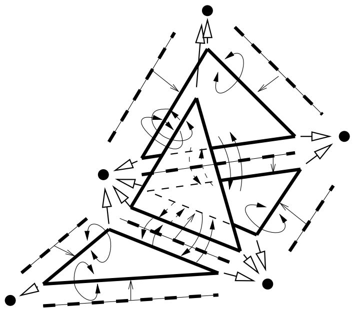

Figure 12: The suggested data structures are hyperfaces, ridges, and vertices. In this three-dimensional example, the hyperfaces appear as triangles and the ridges appear as dashed lines. Each hyperface is entered in several circularly-, doubly-linked lists—one for each of its ridges. Each ridge needs only one pointer to a hyperface to access the appropriate list.

The ridge dictionary (best implemented as a hash table) maps an unordered $(d-1)$-tuple of vertices to the corresponding ridge, if a ridge having those $d-1$ vertices has been constructed. (The appellation *unordered tuple* implies that the order of the vertices does not affect the ridge, or lack thereof, produced by the dictionary. This effect may be achieved by an appropriate choice of hash function, or by sorting the vertices before hashing.) Initialize the dictionary and the triangulation data structure as follows. For each constraining $(d-1)$-simplex f in X, enter each ridge of f in the ridge dictionary (if it's not already there), and place f on the circularly-linked lists associated with those ridges. Similarly, enter each ridge of Δ in the ridge dictionary, and place the hyperfaces of Δ on the lists associated with the ridges they contain.

Any given ridge may be contained in several constraining simplices (and perhaps in Δ as well), but each ridge should be entered into the dictionary only once. A ridge may have any number of hyperfaces associated with it when the triangulation data structure is initialized, and these faces should simply be appended to the ridge's linked list. When all entries have been recorded in the ridge dictionary, sort each ridge's linked list of hyperfaces according to their rotary order around the ridge. If a hyperface of Δ is also a constraining facet, it will have been entered twice; duplicate faces are eliminated after the sorting is done. (When eliminating a face, be sure to remove it from all d of its linked lists simultaneously.) Henceforth, the sorted order of all linked lists will be maintained whenever a new hyperface is inserted into the triangulation data structure.

For each vertex v, the drop face f of v is determined by checking every constraining $(d-1)$-simplex and the hyperfaces of Δ to see which is struck first. Once f is identified, v is appended to its drop list.

The next step is to create a priority queue that holds a collection of potential d-simplices (each of which may or may not be constrained Delaunay). Simplices on the queue are ordered lexicographically according to their circumcenters (for a sweep plane) or power-tuples (for a sweep sphere). Each item on the priority queue is either a pair of hyperfaces f and f' that share a common ridge, or a pair consisting of a vertex v and its drop face f. The key for each item is the circumcenter or power-tuple Ψ_Δ of the d-simplex $\text{conv}(f \cup f')$ or $\text{conv}(f \cup v)$.

The priority queue is initialized by identifying every pair f, f' of hyperfaces that satisfies three conditions: f and f' share a common ridge r, there is no intervening face sharing r, and each of f and f' is in front of the other (which implies that the dihedral angle between them at r, measured in front of either one, is less than $180°$). Where all three conditions are satisfied, the pair of faces is stored on the priority queue as the triple $\langle f, f', p\rangle$, where p is the circumcenter or power-tuple of $\text{conv}(f \cup f')$.

The drop list of each unfinished hyperface f is scanned to find one vertex v that is a candidate for finishing f (if f's drop list is not empty). Of all the vertices on f's drop list, v is the vertex that is inside the circumsphere of $\text{conv}(f \cup v')$ for every other vertex v' on the drop list. As this property is transitive (for vertices in front of f), a linear scan through the list will find v. The triple $\langle f, v, p\rangle$ is stored on the priority queue, where p is the circumcenter or power-tuple of $\text{conv}(f \cup v)$.

Finally, the sweeping step commences. The effect of the sweep plane or sweep sphere is simulated by successively removing potential d-simplices from the priority queue, and constructing them if they are constrained Delaunay. Repeatedly, remove the triple $\langle f, g, p\rangle$ (where g may be a hyperface or a vertex) with lexicographically minimum p from the priority queue. If g is a vertex, but the drop list of f is now empty, then f has been finished by some other vertex; discard the triple and move on to the next triple in the priority queue. If g is a hyperface, and f and g are separated by one or more intervening hyperfaces sharing the same common ridge, move on to the next item in the priority queue. (Clearly, the intervening face was constructed after $\langle f, g, p\rangle$ was enqueued.) Otherwise, $s = \text{conv}(f \cup g)$ is constrained Delaunay; construct any ridges and hyperfaces of s not yet present in the triangulation data structure, if any are missing.

Let F_{old} be the set of preexisting hyperfaces of s, let F_{new} be the set of hyperfaces of s that were not previously present in the triangulation data structure, and let r be any ridge of any face of F_{new}. If r is contained in one face f_{new} of F_{new}, and one preexisting face f_{old} of F_{old}, then insert f_{new} next to f_{old} in r's linked list. If r is contained in two faces of F_{new}, there are two possibilities. If r is a new ridge, not previously in the ridge dictionary, enter r into the dictionary with the two new faces on its linked list. If r is already in the dictionary (because it is a ridge of one or more constraining simplices), then insert the two new faces of s into r's linked list. This entails a linear search through the list.

Each face in F_{new} creates the opportunity to find additional potential constrained Delaunay d-simplices. For each face $f \in F_{\text{new}}$, examine the neighboring faces of f for possible pairings. The conditions are the same as when the priority queue is initialized: any pair $\langle f, f'\rangle$ is enqueued if f and f' share a common ridge r, each of f and f' is in front of the other, and there is no intervening face sharing r.

Construct the drop lists of the faces of F_{new} by redistributing the drop lists of the faces that were finished by the construction of s. (Note that this does not necessarily include all the faces of F_{old}, because some faces of F_{old} may be constraining simplices with s behind them.) Walk through the drop list of each face finished by s, and move each vertex to the drop list of its new drop face in F_{new}. For each face f in F_{new}, find the vertex v in its drop list that is a candidate for finishing f (if the list is not empty), and enqueue $\langle f, v\rangle$.

Once the ridges and hyperfaces of F_{new} have been processed, remove another item from the priority queue and process it likewise. When the queue is empty, the CDT is complete.

The algorithm SWEEPCDT is summarized in Figure 13. Its total running time is $\mathcal{O}(n_v n_f + n_s \log n_v + \sum_f \text{drop}(f))$, where the summation is over all hyperfaces created by the algorithm, and $\text{drop}(f)$ is the number of vertices on f's drop list after f is created. This expression simplifies to $\mathcal{O}(n_v n_s)$, so SWEEPCDT is at least as efficient as the gift-wrapping of unconstrained triangulations. Unfortunately, $\sum_f \text{drop}(f)$ really can achieve the worst case implied by its upper bound: it is possible to realize a sequence of $\Theta(n_s)$ faces such that each has most of the vertices in its drop list, even if $n_s \in \Theta(n_v^{\lceil d/2\rceil})$. However, examples that realize the worst case are difficult to devise, and $\sum_f \text{drop}(f)$ is unlikely to exceed $\mathcal{O}(n_v^2)$ in practical problems. It might be closer to $\mathcal{O}(n_v^{1+1/d})$ for "nicely distributed" input vertices. Hence, overall performance better than $\Theta(n_v n_s)$ is likely.

For the sake of simplicity, several inefficiencies appear in the algorithm as written. When the priority queue is initialized, each eligible pair of faces is entered into the queue twice (as $\langle f, f', p\rangle$ and $\langle f', f, p\rangle$). When NEWFACE investigates a new face f for possible participation in constrained Delaunay d-simplices, up to d potential triples of the form $\langle f, f', p\rangle$ and one triple of the form $\langle f, v, p\rangle$ may be entered into the queue; however, it is only necessary to enter the pair with the minimum key. Both these inefficiencies can and should be removed, though it makes the code less readable.

4. The Correctness of the Sweep Algorithms

The proof that SWEEPCDT constructs a CDT if one exists relies on the fact that constrained Delaunay d-simplices do not occupy shared volume.

```
SWEEPCDT(X)
  Construct one constrained Delaunay d-simplex Δ by
                naïve gift-wrapping
  for each constraining hyperface f ∈ X
    Enter each ridge r of f in the dictionary, with f
                appended to r's linked list
  Enter each ridge r of Δ in the dictionary, with the two
                hyperfaces of Δ that contain r
                appended to r's linked list
  for each ridge r in the dictionary
    Sort the list of hyperfaces around r; remove duplicates
  for each vertex v ∈ X − {Δ_0, Δ_1, ... , Δ_d}
    Find the drop face f of v; append v to f's drop list
  for each hyperface f in the triangulation data structure
                whose front is not exterior
    NEWFACE(f)
  while priority queue Q is not empty
    Remove ⟨f, g, p⟩ with lexicographically least p from Q
    if (g is a vertex and f's drop list is not empty) or
                f and g are adjacent hyperfaces in a linked list
      Let s be the d-simplex conv(f ∪ g), F_old the existing
                faces of s, and F_new the missing faces of s
      Create the faces F_new
      for each ridge r of the faces in F_new
        if r is a ridge of any face in F_old
          Insert the new face that contains r into r's list
        else if r is in the ridge dictionary
          Insert the two new faces that contain r into r's list
        else Enter r into the dictionary, linked to two new faces
      for each vertex v (except g) in the drop list of each
                face finished by s
        Find v's new drop face f among F_new
        Append v to f's drop list
      for each face f ∈ F_new
        NEWFACE(f)

NEWFACE(f)
  for each ridge r of f
    f' ⇐ the next face (in front of f) in r's linked list
    if r is a reflex ridge (the angle ff' < 180°) and
                f is in front of f'
      p ⇐ the circumcenter or power-tuple of conv(f ∪ f')
      Insert ⟨f, f', p⟩ into priority queue Q
  if f's drop list is not empty
    Choose a candidate vertex v from f's drop list
    p ⇐ the circumcenter or power-tuple of conv(f ∪ v)
    Insert ⟨f, v, p⟩ into priority queue Q
```

Figure 13: Sweep algorithm for constructing a CDT. The first and fourth lines are used in the sweep sphere version only.

THEOREM 4. *Suppose that no $d+2$ vertices of a PLC X lie on a common sphere. Then the constrained Delaunay d-simplices of X have disjoint interiors.*

Proof: Suppose for the sake of contradiction that some point p lies in the interior of two distinct constrained Delaunay d-simplices s and t. Because s and t are constrained Delaunay, every vertex of s and t is visible from p.

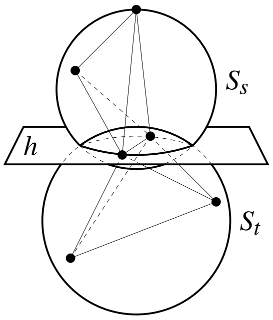

Figure 14: The constrained Delaunay d-simplices s and t intersect at a lower-dimensional shared face (in this illustration, an edge).

Let S_s and S_t be the circumspheres of s and t. S_s and S_t cannot be identical, because s and t each have at least one vertex not shared by the other; if S_s and S_t were the same, at least $d+2$ vertices would lie on S_s. S_s cannot enclose S_t, nor the converse, because S_s and S_t enclose no vertices visible from p. Hence, either S_s and S_t are entirely disjoint (and thus so are s and t), or their intersection is a $(d-1)$-dimensional circle or point and is contained in a $(d-1)$-dimensional hyperplane h, as Figure 14 illustrates. (If S_s and S_t intersect at a single point, h is chosen to be tangent to both spheres.) Without loss of generality, suppose h is oriented horizontally, with the center of S_s directly above the center of S_t. Because p lies in the interiors of s and t, either some vertex of s lies below h, or some vertex of t lies above h. In the former case, there is a vertex of s inside S_t that is visible from a point (p) inside t, so t is not constrained Delaunay. In the latter case, there is a vertex of t inside S_s that is visible from a point inside s, so s is not constrained Delaunay. Either case implies a contradiction, so s and t have disjoint interiors. ■

THEOREM 5. *Suppose that no $d+2$ vertices of a PLC X lie on a common sphere. If X has a CDT,* SWEEPCDT(X) *constructs it.*

Proof: Let s be a constrained Delaunay d-simplex of X. By Theorem 4, no other constrained Delaunay d-simplex intersects the interior of s, so any CDT of X must include s. Suppose for the sake of induction that SWEEPCDT constructs every constrained Delaunay d-simplex whose circumcenter or power-tuple precedes that of s. Then I shall show that SWEEPCDT constructs s as well.

Let f be a hyperface of s such that s is in front of f. If f is a constraining $(d-1)$-simplex of X or a hyperface of Δ, then f is constructed by the first six lines of SWEEPCDT. Otherwise, the assumption that a CDT exists implies that f is a constrained Delaunay hyperface of another constrained Delaunay d-simplex, which has a lexicographically lesser circumcenter (by Theorem 1) or power-tuple (by Theorem 2) than s, and hence has already been constructed. Therefore, SWEEPCDT constructs every hyperface of s that s is in front of before the sweep plane or sweep sphere reaches s.

If s is in front of at least two of its hyperfaces, s is inserted in the priority queue as soon as the second of these hyperfaces is constructed. If s is in front of exactly one hyperface f, where $s = \text{conv}(f \cup v)$, then f must become the drop face for v as soon as f is created (otherwise, s would not be constrained Delaunay), whereupon s is inserted in the priority queue. In either case, s enters the priority queue before the sweep reaches s.

When the sweep reaches s, s is created unless some hyperface f that s is in front of has already been finished, which never happens. Suppose for the sake of contradiction that f is finished by a d-simplex $t = \mathrm{conv}(f \cup w)$ before the sweep reaches s. (Theorem 4 states that two constrained Delaunay d-simplices cannot have intersecting interiors, so t is not constrained Delaunay.) Because t was swept first, and because no $d + 2$ vertices are cospherical, t's circumcenter or power-tuple lexicographically precedes that of s, which implies that w is inside the circumsphere of s. Because s is constrained Delaunay, w is not visible from any point in the interior of s, and therefore is not visible from any point in the interior of f. Hence, w cannot have appeared on f's drop list at any time.

It follows that t was entered on the priority queue because t is in front of at least two hyperfaces f and f', with w a vertex of f'. When t was entered, f and f' were adjacent in the linked list of their shared ridge $r = f \cap f'$. Because s is constrained Delaunay, for any point p that is in the interior of s and in a neighborhood of r, some constraining $(d - 1)$-simplex must prevent w from being visible from p. This constraining simplex cannot contain r (because it would interpose between f and f' in r's linked list), nor can it contain w (because it would not block w's visibility). It follows that some constraining $(d - 1)$-simplex intersects the interior of f' but does not contain f', so f' cannot be a face of the CDT.

However, f' must be a face of the CDT, because f' is either a hyperface of a constrained Delaunay d-simplex (behind f'), a constraining $(d - 1)$-simplex, or a face of the leftward-facing boundary of the CDT. This contradiction implies that no d-simplex can be built atop f before the sweep reaches s. Hence, SWEEPCDT constructs s.

By induction, every constrained Delaunay d-simplex of X is constructed by SWEEPCDT. No d-simplex that is not constrained Delaunay is constructed, because any candidate simplex t entered in the priority queue is in front of at least one of its hyperfaces f, and the constrained Delaunay d-simplex in front of f is constructed first, thereby preventing the construction of t if t is not constrained Delaunay. By Theorem 4, any two constrained Delaunay d-simplices have disjoint interiors, so the set of all constrained Delaunay d-simplices of X is the CDT of X if a CDT of X exists. ■

5. Constructing the CDT of a Star-Shaped Polytope

Let X be a PLC whose constraining $(d - 1)$-simplices bound a simple star-shaped polytope, and let p be a point inside the polytope that can see every point in the boundary of the polytope. When used to find the CDT of the polytope, the sweep sphere algorithm may be sped up and simplified with just a few changes. The most important change is that the order of simplex construction is reversed: a constrained Delaunay d-simplex s is constructed before another constrained Delaunay d-simplex t if $\Psi_p(s) > \Psi_p(t)$.

With this ordering, the drop face of a vertex is found by shooting a ray from the vertex *away* from p. Because the polytope is star-shaped, such a ray strikes no constraining simplex, and no vertex has a drop face. All code for computing, maintaining, and using drop faces may be expunged. Furthermore, there is no need to construct a seed Δ. The algorithm that results is conceptually the same as that presented by Devillers [2] for unconstrained Delaunay triangulations. With the overhead of drop faces gone, the running time is improved to $\mathcal{O}(n_s \log n_v)$. This algorithm can be used to delete a vertex v from a CDT by setting $p = v$ and letting X be the set of vertices adjacent to v and hyperfaces opposite v in the d-simplices that contain v. The algorithm works correctly even if v lies on the boundary of the CDT (and thus X does not represent a closed boundary). However, there are circumstances where deleting a vertex from a PLC yields a PLC that has no CDT. In these cases, the algorithm may not behave in any reasonable manner.

6. Symbolic Perturbation of Cospherical Vertices Made Totally Easy

A PLC containing $d + 2$ or more cospherical vertices may have constrained Delaunay d-simplices whose interiors are not disjoint. Because of these, a gift-wrapping algorithm may make decisions that are mutually inconsistent, and find itself unable to construct a valid triangulation.

This problem can be solved by perturbing the vertices by tiny displacements so that no cospherical subsets of $d + 2$ vertices remain. The perturbed vertices are triangulated, then the vertices are restored to their original positions.

A problem with most perturbation techniques (including symbolic perturbation methods) is that the perturbed vertices may admit extremely thin constrained Delaunay d-simplices that become degenerate when the perturbation is reversed. The method proposed below avoids this pitfall, because coplanar subsets of vertices remain coplanar when perturbed. Only geometric tests that involve spheres are affected by the perturbation.

Vertices are not perturbed in E^d at all. Instead, the present method takes advantage of the well-known *lifting map* of Edelsbrunner and Seidel [4]. The lifting map projects each input vertex to a vertex on a paraboloid in a space one dimension higher, as Figure 15 illustrates. Specifically, each vertex $(v_1, v_2, \ldots, v_d) \in X$ is mapped to a point $(v_1, v_2, \ldots, v_d, v_1^2 + v_2^2 + \cdots + v_d^2)$ in E^{d+1}. The lifting map is commonly used to convert a convex hull algorithm for E^{d+1} into a Delaunay triangulation algorithm for E^d. This connection cannot be directly extended to CDTs, because a lifted CDT does not generally bound a convex volume. However, my perturbation method exploits the fact that a test on spheres in E^d is a test on hyperplanes in E^{d+1}. For instance, testing whether a point is inside the sphere passing through $d + 1$ vertices in E^d is equivalent to testing whether a lifted point is under the hyperplane passing through $d + 1$ lifted vertices in E^{d+1}.

Abstractly, the vertices of X are perturbed sequentially in some arbitrary order (e.g., lexicographic). Each lifted vertex is perturbed downward—that is, by a negative displacement of its $(d + 1)$th coordinate. Intuitively, the first vertex is perturbed an infinitesimal distance, and each subsequent vertex a distance infinitesimally smaller than the previous vertex. For rigor's sake, the perturbations are better viewed as tiny, finite displacements chosen small enough that no lifted vertex ever enters or crosses a hyperplane defined by any other $d + 1$ lifted vertices.

These perturbations are not actually performed. Instead, their effect is simulated during any geometric test involving spheres. For instance, the standard test for whether a vertex lies inside a sphere (in E^d) computes the sign of an expression. If the expression is nonzero, the perturbations have no effect and need not be simulated. If the expression is zero, the $d + 2$ vertices lie on a com-

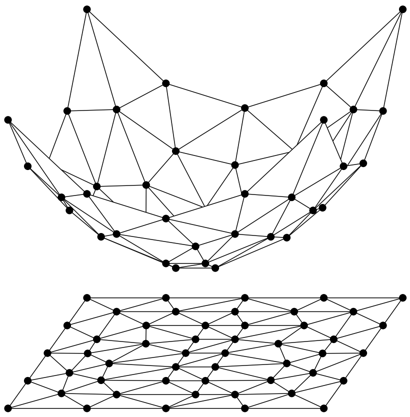

Figure 15: The parabolic lifting map. In this example, a two-dimensional Delaunay triangulation is lifted to a paraboloid in E^3. The lifted triangulation is the lower boundary of the convex hull of the lifted vertices.

mon sphere. To simulate the perturbations, choose whichever of the $d+2$ vertices comes first in the ordering, subtract an arbitrary positive constant from its $(d+1)$th coordinate (for this test only), and recompute the expression. The choice of constant is irrelevant; if the other $d+1$ lifted vertices are affinely independent, the perturbed vertex is moved beneath the hyperplane that contains them, and a nonzero result is certain. If the vertices are not affinely independent (the expression's value is still zero), continue through the lifted vertices in order, perturbing each (again by an arbitrary downward displacement) until the expression's value is nonzero.

For consistency, the same method and vertex order must be used in the tests that determine the order of circumcenters in the priority queue. Each circumcenter is defined by $d+1$ vertices; perturbing any one of these perturbs the circumcenter accordingly.

The tests performed by this perturbation method are mutually consistent, and (if exact arithmetic is used) ensure that the gift-wrapping and sweep algorithms produce the weighted constrained Delaunay triangulation of the perturbed points, if one exists. However, it is possible that such a triangulation does not exist, even if the unperturbed points have a CDT. The order in which the vertices are perturbed can determine whether or not a triangulation is discovered. It is an open question whether any PLC with a CDT can be perturbed in such a way that a CDT will be produced.

7. An Improved Sweep Algorithm

Recall from Section 3 that for each vertex v, we seek the constrained Delaunay d-simplex s_v that is behind all its hyperfaces incident on v. With an idea adapted from Seidel [10], s_v may be identified in $\mathcal{O}(n_v n_f)$ time by linear programming. This cost is high, but is worthwhile if a vertex threatens to be redistributed to a new drop face more than $\Theta(n_v n_f)$ times.

If a vertex v is redistributed $cn_v n_f$ times for some experimentally chosen constant c, the improved sweep algorithm constructs the simplex s_v using linear programming, adds it to the triangulation data structure, and checks every pending vertex to see if its drop face is now a face of s_v. Because no vertex is redistributed more than $cn_v n_f$ times, $\sum_f \text{drop}(f) \in \mathcal{O}(n_v^2 n_f)$, and the overall running time for CDT construction is $\mathcal{O}(\min\{n_v n_s, n_v^2 n_f + n_s \log n_v\})$.

In practice, it is difficult to construct examples where any vertex is redistributed more than $\Theta(n_v)$ times, so this algorithm is only of theoretical interest. The algorithm for finding s_v through linear programming is quite complicated, and is omitted here.

8. References

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

[2] 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.

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

[4] Herbert Edelsbrunner and Raimund Seidel. *Voronoi Diagrams and Arrangements*. Discrete & Computational Geometry **1**:25–44, 1986.

[5] Steven Fortune. *A Sweepline Algorithm for Voronoi Diagrams*. Algorithmica **2**(2):153–174, 1987.

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

[7] 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.

[8] Jim Ruppert and Raimund Seidel. *On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra*. Discrete & Computational Geometry **7**(3):227–254, 1992.

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

[10] Raimund Seidel. *Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face*. Proceedings of the 18th Annual ACM Symposium on the Theory of Computing, pages 404–413. Association for Computing Machinery, 1986.

[11] 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.

[12] ———. *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.

[13] ———. *Mesh Generation for Domains with Small Angles*. June 2000. This proceedings.

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