Chapter 4

Three-dimensional Delaunay triangulations

Three-dimensional triangulations are sometimes called tetrahedralizations. Delaunay tetrahedralizations are not quite as effective as planar Delaunay triangulations at producing elements of good quality, but they are nearly as popular in the mesh generation literature as their two-dimensional cousins. Many properties of Delaunay triangulations in the plane generalize to higher dimensions, but many of the optimality properties do not. Notably, Delaunay tetrahedralizations do not maximize the minimum angle (whether plane angle or dihedral angle). Figure 4.1 depicts a three-dimensional counterexample. The hexahedron at the top is the convex hull of its five vertices. The Delaunay triangulation of those vertices, to the left, includes a thin tetrahedron known as a *sliver* or *kite*, whose vertices are nearly coplanar and whose dihedral angles can be arbitrarily close to 0° and 180°. A triangulation of the same vertices that is not Delaunay, at lower right, has better quality.

This chapter surveys Delaunay triangulations and constrained Delaunay triangulations in three—and occasionally higher—dimensions. Constrained Delaunay triangulations generalize uneasily to three dimensions, because there are polyhedra that do not have any tetrahedralization at all.

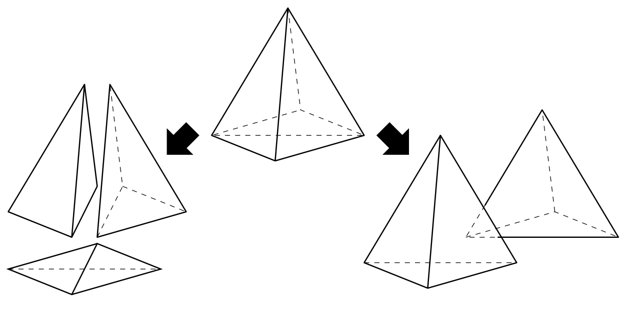

Figure 4.1: This hexahedron has two tetrahedralizations. The Delaunay tetrahedralization at left includes an arbitrarily thin sliver tetrahedron. The non-Delaunay tetrahedralization at right consists of two nicely shaped tetrahedra.

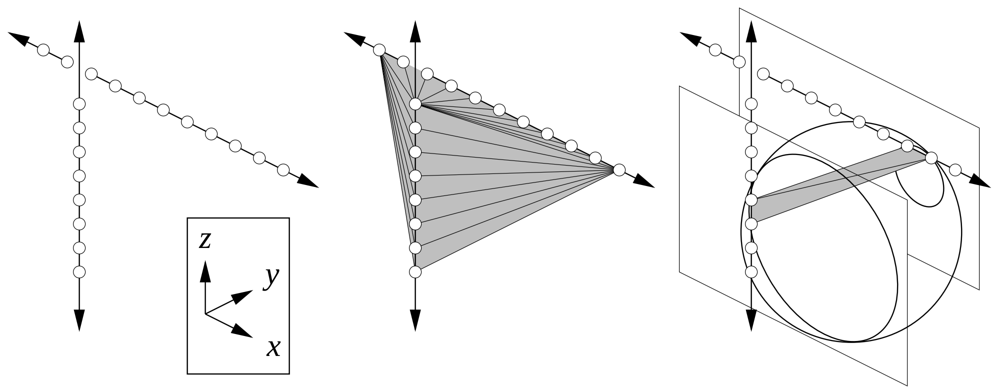

Figure 4.2: At center, the Delaunay tetrahedralization of the points at left. At right, the circumball of one Delaunay tetrahedron with two cross-sections showing it is empty.

4.1 Triangulations of a point set in $\mathbb{R}^d$

Definition 2.1 in Section 2.1 defines a triangulation of a set of points to be a simplicial complex whose vertices are the points and whose union is the convex hull of the points. With no change, the definition holds in any finite dimension d. Figures 4.1–4.4 illustrate triangulations of point sets in three dimensions. Every finite point set in $\mathbb{R}^d$ has a triangulation; for example, the lexicographic triangulation of Section 2.1 also generalizes to higher dimensions with no change.

Let S be a set of n points in $\mathbb{R}^d$. Recall from Section 2.1 that if all the points in S are collinear, they have one triangulation having n vertices and $n - 1$ collinear edges connecting them. This is true regardless of d; the triangulation is one-dimensional, although it is embedded in $\mathbb{R}^d$. More generally, if the affine hull of S is k-dimensional, then every triangulation of S is a k-dimensional triangulation embedded in $\mathbb{R}^d$: the simplicial complex has at least one k-simplex but no $(k + 1)$-simplex.

The *complexity* of a triangulation is its total number of simplices of all dimensions. Whereas a planar triangulation of n points has $O(n)$ triangles and edges, a surprising property of higher-dimensional triangulations is that they can have superlinear complexity. Figure 4.2 shows a triangulation of n points that has $\Theta(n^2)$ edges and tetrahedra. Every vertex lies on one of two non-intersecting lines, and there is one tetrahedron for each pairing of an edge on one line and an edge on the other. This is the *only* triangulation of these points, and it is Delaunay. In general, a triangulation of n vertices in $\mathbb{R}^3$ has at most $(n^2 - 3n - 2)/2$ tetrahedra, at most $n^2 - 3n$ triangles, and at most $(n^2 - n)/2$ edges. An n-vertex triangulation in $\mathbb{R}^d$ can have a maximum of $\Theta(n^{\lceil d/2 \rceil})$ d-simplices.

4.2 The Delaunay triangulation in $\mathbb{R}^d$

Delaunay triangulations generalize easily to higher dimensions. Let S be a finite set of points in $\mathbb{R}^d$, for $d \geq 1$. Let σ be a k-simplex (for any $k \leq d$) whose vertices are in S. The simplex σ is *Delaunay* if there exists an open circumball of σ that contains no point in S. Clearly, every face of a Delaunay simplex is Delaunay too. The simplex σ is *strongly Delaunay* if there exists

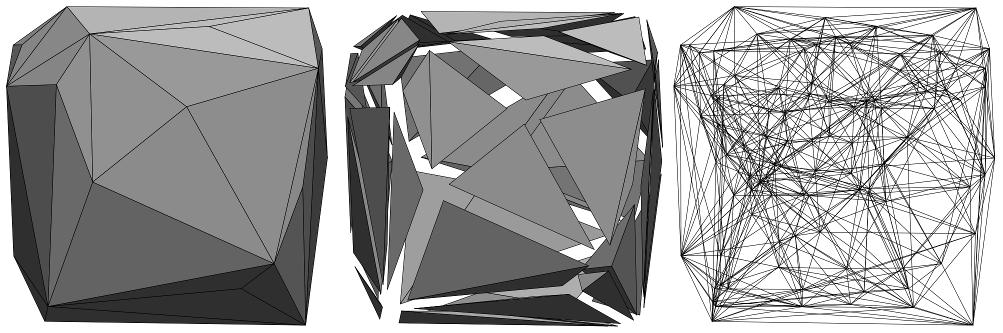

Figure 4.3: Three renderings of a Delaunay tetrahedralization.

a closed circumball of σ that contains no point in S except the vertices of σ. Every point in S is trivially a strongly Delaunay vertex.

Definition 4.1 (Delaunay triangulation)**.** Let S be a finite point set in $\mathbb{R}^d$, and let k be the dimension of its affine hull. A *Delaunay triangulation* $\operatorname{Del} S$ of S is a triangulation of S in which every k-simplex is Delaunay—and therefore, every simplex is Delaunay.

Figure 4.2 depicts a Delaunay tetrahedralization and the empty circumball of one of its tetrahedra. Figure 4.3 depicts a more typical Delaunay tetrahedralization, with complexity linear in the number of vertices.

The parabolic lifting map generalizes to higher dimensions too. It maps each point $p = (p_1, p_2, \ldots, p_d) \in \mathbb{R}^d$ to its *lifted companion*, the point $p^+ = (p_1, p_2, \ldots, p_d, p_1^2 + p_2^2 + \cdots + p_d^2)$ in $\mathbb{R}^{d+1}$. Consider the $(d+1)$-dimensional convex hull of the lifted points, $S^+ = \{v^+ : v \in S\}$. Projecting the downward-facing faces of $\operatorname{conv} S^+$ to $\mathbb{R}^d$ yields a polyhedral complex called the *Delaunay subdivision* of S. If S is *generic*, its Delaunay subdivision is simplicial and S has exactly one Delaunay triangulation.

Definition 4.2 (generic)**.** Let S be a point set in $\mathbb{R}^d$. Let k be the dimension of the affine hull of S. The set S is *generic* if no $k+2$ points in S lie on the boundary of a single d-ball.

If S if not generic, its Delaunay subdivision may have non-simplicial faces; recall Figure 2.3. In that case, S has multiple Delaunay triangulations, which differ according to how the non-simplicial faces are triangulated.

Whereas each non-simplicial face in a two-dimensional Delaunay subdivision can be triangulated independently, in higher dimensions the triangulations are not always independent. Figure 4.4 illustrates a set of twelve points in $\mathbb{R}^3$ whose Delaunay subdivision includes two cubic cells that share a square 2-face. The square face can be divided into two triangles in two different ways, and each cube can be divided into five or six tetrahedra in several ways, but they are not independent: the triangulation of the square face constrains how both cubes are triangulated.

A *least-vertex triangulation* provides one way to safely subdivide a polyhedral complex into a simplicial complex. To construct it, triangulate the 2-faces through the d-faces in order of increasing dimension. To triangulate a non-simplicial k-face f, subdivide it into k-simplices of the form $\operatorname{conv}(v \cup g)$, where v is the lexicographically minimum vertex of f, and g varies over the $(k-1)$-simplices on f's subdivided boundary that do not contain v. The choice of the lexicographically

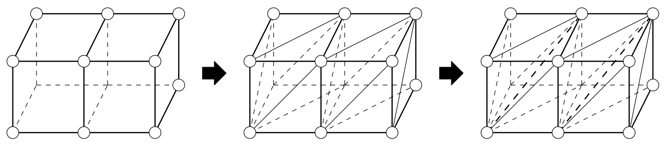

Figure 4.4: A Delaunay subdivision comprising two cubic cells and their faces. The least-vertex Delaunay triangulation subdivides each 2-face into triangles adjoining the face's lexicographically minimum vertex, and likewise subdivides each 3-face into tetrahedra.

minimum vertex of each face ensures that the face triangulations are compatible with each other. The least-vertex triangulation is consistent with the weight perturbations described in Section 2.9.

Many properties of planar Delaunay triangulations discussed in Chapter 2 generalize to higher dimensions. A few of them are summarized below. Proofs are omitted, but each of them is a straightforward extension of the corresponding proof for two dimensions.

Recall that a *facet* of a polyhedral complex is a $(d-1)$-face, and a facet of a triangulation is a $(d-1)$-simplex. The forthcoming Delaunay Lemma provides an alternative definition of a Delaunay triangulation: a triangulation of a point set in which every facet is locally Delaunay. A facet f in a triangulation $\mathcal{T}$ is said to be *locally Delaunay* if it is a face of fewer than two d-simplices in $\mathcal{T}$, or it is a face of exactly two d-simplices τ_1 and τ_2 and it has an open circumball that contains no vertex of τ_1 nor τ_2. Equivalently, the open circumball of τ_1 contains no vertex of τ_2. Equivalently, the open circumball of τ_2 contains no vertex of τ_1.

Lemma 4.1 (Delaunay Lemma)**.** *Let $\mathcal{T}$ be a triangulation of a finite, d-dimensional set S of points in $\mathbb{R}^d$. The following three statements are equivalent.*

- *Every d-simplex in $\mathcal{T}$ is Delaunay (i.e. $\mathcal{T}$ is Delaunay).*
- *Every facet in $\mathcal{T}$ is Delaunay.*
- *Every facet in $\mathcal{T}$ is locally Delaunay.* □

As in the plane, a generic point set has exactly one Delaunay triangulation, composed of every strongly Delaunay simplex. The following three propositions have essentially the same proofs as in Section 2.7.

Proposition 4.2. *Let σ be a strongly Delaunay simplex, and let τ be a Delaunay simplex. Then $\sigma \cap \tau$ is either empty or a shared face of both σ and τ.*

Proposition 4.3. *Every Delaunay triangulation of a point set contains every strongly Delaunay simplex.*

Theorem 4.4. *A generic point set has exactly one Delaunay triangulation.*

4.3 The optimality of the Delaunay triangulation in $\mathbb{R}^d$

Some optimality properties of Delaunay triangulations hold in any dimension. Consider the use of triangulations for piecewise linear interpolation of a quadratic multivariate function. If the function is isotropic—of the form $\alpha\|p\|^2 + \langle a, p\rangle + \beta$ for $p \in \mathbb{R}^d$—then the Delaunay triangulation minimizes the interpolation error measured in the L_q-norm for every $q \geq 1$, compared with all other triangulations of the same points. (If the function is not isotropic, but it is parabolic rather than hyperbolic, then the optimal triangulation is a weighted Delaunay triangulation in which the function determines the vertex heights.)

Delaunay triangulations also minimize the radius of the largest min-containment ball of their simplices (recall Definition 1.20). This result implies a third optimality result, also related to multivariate piecewise linear interpolation. Suppose one must choose a triangulation to interpolate an unknown function, and one wishes to minimize the largest pointwise error in the domain. After one chooses the triangulation, an adversary will choose the worst possible smooth function for the triangulation to interpolate, subject to a fixed upper bound on the absolute curvature (i.e. second directional derivative) of the function anywhere in the domain. The Delaunay triangulation is the optimal choice.

To better understand these three optimality properties, consider multivariate piecewise linear interpolation on a triangulation $\mathcal{T}$ of a point set S. Let $\mathcal{T}^+ = \{\sigma^+ : \sigma \in \mathcal{T}\}$ be the triangulation lifted by the parabolic lifting map; $\mathcal{T}^+$ is a simplicial complex embedded in $\mathbb{R}^{d+1}$. Think of $\mathcal{T}^+$ as inducing a continuous piecewise linear function $\mathcal{T}^+(p)$ that maps each point $p \in \operatorname{conv} S$ to a real value.

How well does $\mathcal{T}^+$ approximate the paraboloid? Let $e(p) = \mathcal{T}^+(p) - \|p\|^2$ be the error in the interpolated function $\mathcal{T}^+$ as an approximation of the paraboloid $\|p\|^2$. At each vertex $v \in S$, $e(v) = 0$. Because $\|p\|^2$ is convex, the error satisfies $e(p) \geq 0$ for all $p \in \operatorname{conv} S$.

Proposition 4.5. *At every point $p \in \operatorname{conv} S$, every Delaunay triangulation $\mathcal{T}$ of S minimizes $\mathcal{T}^+(p)$, and therefore minimizes the interpolation error $e(p)$, among all triangulations of S. Hence, every Delaunay triangulation of S minimizes $\|e\|_{L_q}$ for every Lebesgue norm L_q, and every other norm monotonic in e.*

P . If $\mathcal{T}$ is Delaunay, then $\mathcal{T}^+$ is the set of faces of the underside of the convex hull $\operatorname{conv} S^+$ of the lifted vertices (or a subdivision of those faces if some of them are not simplicial). No simplicial complex in $\mathbb{R}^{d+1}$ whose vertices are all in S^+ can pass through any point below $\operatorname{conv} S^+$. $\square$

Proposition 4.6. *Let o and r be the circumcenter and circumradius of a d-simplex σ. Let o_{mc} and r_{mc} be the center and radius of the min-containment ball of σ. Let q be the point in σ nearest o. Then $o_{\text{mc}} = q$ and $r_{\text{mc}}^2 = r^2 - d(o, q)^2$.*

P . Let τ be the face of σ whose relative interior contains q. The face τ is not a vertex, because the vertices of σ are σ's furthest points from o. Because q is the point in τ nearest o, and because q is in the relative interior of τ, the line segment oq is orthogonal to τ. (This is true even if $\tau = \sigma$, in which case $o - q$ is the zero vector.) This fact, plus the fact that o is equidistant from all the vertices of τ, implies that q is equidistant from all the vertices of τ (as Figure 4.5 demonstrates). Let r be the distance between q and any vertex of τ. As $q \in \tau$, there is no containment ball of τ (or σ) with radius less than r because q cannot move in any direction without moving away from some vertex of τ. Therefore, q and r are the center and radius of the min-containment ball of τ.

By the following reasoning, σ has the same min-containment ball as τ. If $q = o$, this conclusion is immediate. Otherwise, let h be the hyperplane through q orthogonal to oq. Observe that

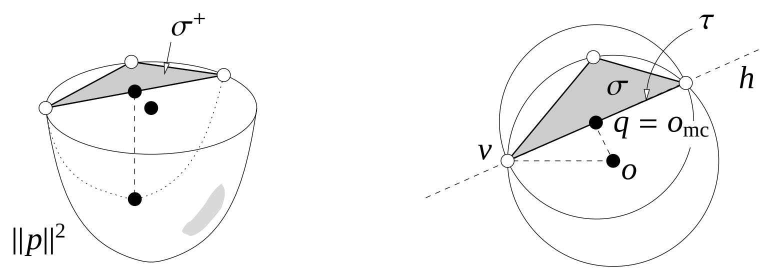

Figure 4.5: Left: within σ, the error $e(p)$ is maximized at the point nearest the circumcenter of σ. Right: top view of σ, its circumdisk, and its min-containment disk.

$\tau \subset h$. No point in σ is on the same side of h as o: if there were such a point w, there would be a point in σ (between w and q) closer to o than q, contradicting the fact that q is closest. The hyperplane h cuts the closed circumball of σ into two pieces, and the piece that includes σ is included in the min-containment ball of τ. Therefore, q and r are the center and radius of the min-containment ball of σ.

Let v be any vertex of τ. Pythagoras' Theorem on the triangle oqv (see Figure 4.5) yields $r_{\mathrm{mc}}^2 = r^2 - d(o,q)^2$. □

Proposition 4.7. *Every Delaunay triangulation of S minimizes the largest min-containment ball, compared with all other triangulations of S.*

P . Over any single d-simplex σ, there is an explicit expression for $e(p)$. Recall from the proof of the Lifting Lemma (Lemma 2.1) that the hyperplane h_σ that includes σ^+ is defined by the function $h_\sigma(p) = 2\langle o, p\rangle - \|o\|^2 + r^2$, where o and r are the circumcenter and circumradius of σ and $p \in \mathbb{R}^d$ varies freely. Hence, for all $p \in \sigma$,

$$\begin{aligned} e(p) &= \mathcal{T}^+(p) - \|p\|^2 \\ &= h_\sigma(p) - \|p\|^2 \\ &= 2\langle o, p\rangle - \|o\|^2 + r^2 - \|p\|^2 \\ &= r^2 - d(o,p)^2. \end{aligned}$$

Figure 4.5 (left) illustrates the functions $h_\sigma(p)$ and $\|p\|^2$ over a triangle σ. The error $e(p)$ is the vertical distance between the two functions. At which point p in σ is $e(p)$ largest? At the point nearest the circumcenter o, because $d(o,p)^2$ is smallest there. (The error is maximized at $p = o$ if o is in σ; Figure 4.5 gives an example where it is not.) Let o_{mc} and r_{mc} be the center and radius of the min-containment ball of σ, respectively. By Proposition 4.6, the point in σ nearest o is o_{mc}, and $e(o_{\mathrm{mc}}) = r^2 - d(o, o_{\mathrm{mc}})^2 = r_{\mathrm{mc}}^2$.

It follows that the square of the min-containment radius of σ is $r_{\mathrm{mc}}^2 = \max_{p\in\sigma} e(p)$, and thus $\max_{p\in\operatorname{conv} S} e(p)$ is the squared radius of the largest min-containment ball of the entire triangulation $\mathcal{T}$. By Proposition 4.5, the Delaunay triangulation minimizes this quantity among all triangulations of S. □

The optimality of the Delaunay triangulation for controlling the largest min-containment radius dovetails nicely with an error bound for piecewise linear interpolation derived by Waldron. Let $\mathcal{F}(c)$ be the space of scalar functions defined over $\operatorname{conv} S$ that are C^1-continuous and whose absolute curvature nowhere exceeds c. In other words, for every $f \in \mathcal{F}(c)$, every point $p \in \operatorname{conv} S$, and every unit direction vector $\vec{u}$, the magnitude of the second directional derivative $f''_{\vec{u}}(p)$ is at most c. This is a common starting point for analyses of piecewise linear interpolation error.

Let f be a function in $\mathcal{F}(c)$. Let $\sigma \subseteq \operatorname{conv} S$ be a simplex (of any dimensionality) with min-containment radius r_{mc}. Let h_σ be a linear function that interpolates f at the vertices of σ. Waldron shows that for all $p \in \sigma$, the absolute error $|e(p)| = |h_\sigma(p) - f(p)|$ is at most $cr_{\text{mc}}^2/2$. Furthermore, this bound is sharp: for every simplex σ with min-containment radius r_{mc}, there is a function $f \in \mathcal{F}(c)$ and a point $p \in \sigma$ such that $|e(p)| = cr_{\text{mc}}^2/2$. That function is $f(p) = c\|p\|^2/2$, as illustrated in Figure 4.5, and that point is $p = o_{\text{mc}}$.

Proposition 4.8. *Every Delaunay triangulation $\mathcal{T}$ of S minimizes* $\max_{f\in\mathcal{F}(c)} \max_{p\in\operatorname{conv} S} |\mathcal{T}^+(p) - f(p)|$*, the worst-case pointwise interpolation error, among all triangulations of S.*

P . Per Waldron, for any triangulation $\mathcal{T}$, $\max_{f\in\mathcal{F}(c)} \max_{p\in\operatorname{conv} S} |\mathcal{T}^+(p) - f(p)| = cr_{\max}^2/2$, where $r_{\max}$ is the largest min-containment radius among all the simplices in $\mathcal{T}$. The result follows immediately from Proposition 4.7. □

One of the reasons for the longstanding popularity of Delaunay triangulations is that, as Propositions 4.5 and 4.8 show, the Delaunay triangulation is an optimal piecewise linear interpolating surface. Of course, $e(p)$ is not the only criterion for the merit of a triangulation used for interpolation. Many applications require that the interpolant approximate the gradient, i.e., $\nabla\mathcal{T}^+(p)$ must approximate $\nabla f(p)$ well. For the goal of approximating $\nabla f(p)$ in three or more dimensions, the Delaunay triangulation is sometimes far from optimal even for simple functions like the paraboloid $f(p) = \|p\|^2$. This is why eliminating slivers is a crucial problem in Delaunay mesh generation.

4.4 Bistellar flips and the flip algorithm

The flip algorithm described in Section 2.5 extends to three or more dimensions, but unfortunately, it does not always produce a Delaunay triangulation. The natural generalizations of edge flips are *bistellar flips*, operations that replace one set of simplices with another set filling the same volume. Figure 4.6 illustrates the bistellar flips in one, two, and three dimensions. The three-dimensional analogs of edge flips are called *2-3 flips* and *3-2 flips*. The names specify the numbers of tetrahedra deleted and created, respectively.

The upper half of the figure depicts basic bistellar flips, which retriangulate the convex hull of $d+2$ vertices in $\mathbb{R}^d$ by replacing a collection of k d-simplices with $d+2-k$ different d-simplices. In three dimensions, there are four basic flips: 1-4, 2-3, 3-2, and 4-1. The 1-4 flip inserts a vertex, and the 4-1 flip deletes one.

The lower half of the figure depicts extended bistellar flips, in which a lower-dimensional basic flip transforms higher-dimensional simplices, possibly many of them. For example, consider bisecting an edge of a tetrahedralization, as illustrated at the lower right of the figure. In essence, this operation is a one-dimensional 1-2 flip, replacing one edge with two. But every tetrahedron

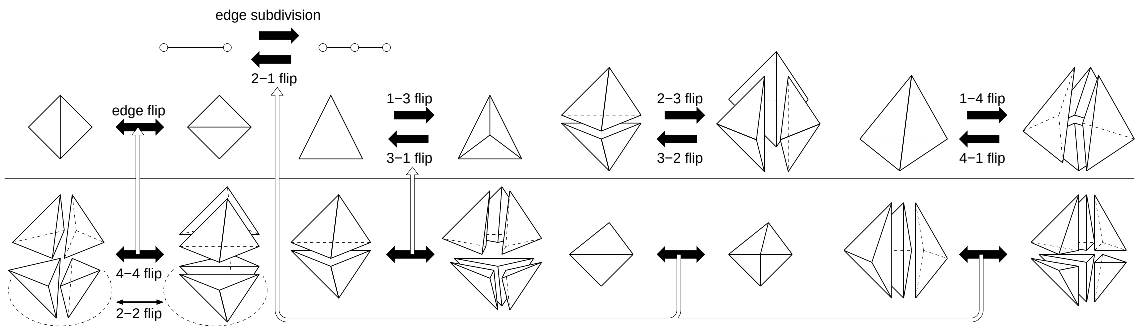

Figure 4.6: Basic bistellar flips in one, two, and three dimensions appear above the line. Extended bistellar flips appear below the line. White arrows connect extended flips to the lower-dimensional flips they are based on. The 2-2 flip at bottom left typically involves two coplanar triangular faces on the boundary of a domain, whereas the 4-4 flip occurs when the corresponding faces are in a domain interior. Edge subdivisions and their inverses can also occur on a domain boundary, as at bottom right, or in a domain interior.

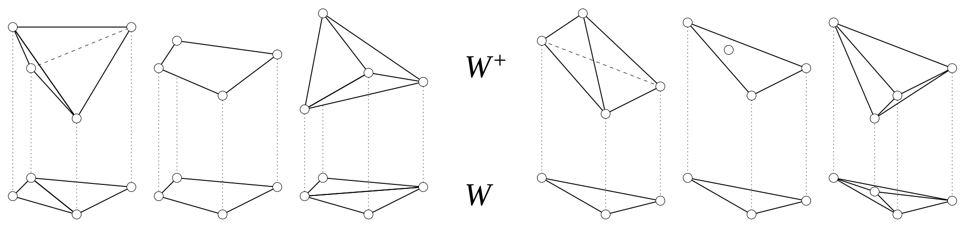

Figure 4.7: If the convex hull of four points in $\mathbb{R}^3$ is a tetrahedron, the facets on its underside determine one planar triangulation, and the facets on its upper side determine another.

that includes the subdivided edge is subdivided into two tetrahedra, and the number of tetrahedra that share the edge can be arbitrarily large. Therefore, some extended flips can delete and create arbitrarily many d-simplices.

An intuitive way to understand the basic bistellar flips in $\mathbb{R}^d$ is through the faces of a simplex in $\mathbb{R}^{d+1}$, as illustrated in Figure 4.7. Let W be a set of $d + 2$ points in $\mathbb{R}^d$, and let W^+ be the same points lifted by the parabolic lifting map in $\mathbb{R}^{d+1}$. Call the $(d + 1)$th coordinate axis (along which the points are lifted) the *vertical axis*. Assume the points in W are not cospherical. Then the points in W^+ are not cohyperplanar, and conv W^+ is a $(d + 1)$-simplex—call it the *W-simplex*. Each facet of the W-simplex can be placed in one of three classes: vertical (parallel to the vertical axis), lower (facing the negative end of the vertical axis), or upper (those that would get wet if rain were falling).

The W-simplex suggests two different triangulations of the region conv W: the Delaunay triangulation by projecting the lower facets of conv W^+ to $\mathbb{R}^d$, and a non-Delaunay triangulation by projecting the upper facets. (If the points in W^+ are cohyperplanar, W has two Delaunay triangulations.) The act of replacing one such triangulation with the other is a bistellar flip. If the W-simplex has no vertical facet, the flip is basic. These are the only two ways to triangulate conv W such that all the vertices are in W. One of the two triangulations might omit a vertex of

W—witness the 3-1 flip, which deletes a vertex.

An extended bistellar flip in $\mathbb{R}^d$ is built on a j-dimensional basic flip as follows. Consider a basic flip that replaces a set T_D of k j-simplices with a set T_C of $j+2-k$ j-simplices. Let the *join* $\tau * \zeta$ of two disjoint simplices τ and ζ be the simplex conv $(\tau \cup \zeta)$ having all the vertices of τ and ζ, and let the *join* of two sets T and Z of simplices be the set of simplices $\{\tau * \zeta : \tau \in T \text{ and } \zeta \in Z\}$. Let Z be a set of $(d - j - 1)$-simplices such that every member of $T_D * Z$ is a nondegenerate d-simplex and $T_D * Z$ contains every d-simplex in $\mathcal{T}$ with a face in T_D. If such a Z exists, the act of replacing the simplices $T_D * Z$ with the simplices $T_C * Z$ is an extended flip. For example, in the edge bisection illustrated at the lower right of Figure 4.6, T_D contains just the bisected edge, T_C contains the two edges the bisection yields, and Z contains one edge for each bisected tetrahedron.

The flip algorithm requires a solution to the following problem. A triangulation $\mathcal{T}$ has a facet f that is not locally Delaunay. What is the appropriate flip to eliminate f? Let W be the set containing the d vertices of f and the two additional vertices of the two d-simplices having f for a face. The fact that f is not locally Delaunay implies that f^+ lies on the upper surface of the W-simplex conv W^+. The upper facets of the W-simplex indicate which d-simplices the flip algorithm should delete from $\mathcal{T}$ (including the two adjoining f), and the lower facets indicate which d-simplices should replace them. The procedure F in Figure 4.8 identifies these tetrahedra for $d = 3$. If the W-simplex has vertical facets, F performs an extended flip.

The difficulty is that the simplices to be deleted might not all be in $\mathcal{T}$. Figure 4.9 illustrates circumstances where the flip algorithm wishes to perform a flip, but cannot. At left, the shaded triangle is not locally Delaunay. The right flip to remove it is a 3-2 flip, but the flip is possible only if the third tetrahedron is present. If four or more tetrahedra share the bold edge, the flip is blocked, at least until another flip creates the missing tetrahedron. The flip algorithm can get stuck in a configuration where *every* locally non-Delaunay triangle's removal is blocked, and the algorithm cannot make further progress toward the Delaunay triangulation. This is not a rare occurrence in practice.

If the flip algorithm is asked to compute a *weighted* Delaunay triangulation, it must sometimes perform a flip that deletes a vertex, such as the 4-1 flip illustrated at right in Figure 4.9. Such a flip is possible only if all the simplices the flip is meant to delete are present. Even in the plane, there are circumstances where every desired 3-1 flip is blocked and the flip algorithm is stuck.

Extended flips are even more delicate; they require not only that $T_D * Z \subseteq \mathcal{T}$, but also that $T_D * Z$ contains *every* tetrahedron in $\mathcal{T}$ that has a face in T_D.

Whether it succeeds or gets stuck, the running time of the flip algorithm is $O(n^{1+\lfloor d/2 \rfloor})$, by the same reasoning explained in Section 2.5: a flip can be modeled as the act of gluing a $(d+1)$-simplex to the underside of the lifted triangulation, and a triangulation in $\mathbb{R}^{d+1}$ has at most $O(n^{1+\lfloor d/2 \rfloor})$ simplices.

One of the most important open problems in combinatorial geometry asks whether the *flip graph* is connected. For a specified point set, the flip graph has one node for every triangulation of those points. Two nodes are connected by an edge if one triangulation can be transformed into the other by a single bistellar flip, excluding those flips that create or delete a vertex. For every planar point set, its flip graph is connected—in other words, any triangulation of the points can be transformed into any other triangulation of the same points by a sequence of edge flips. One way to see this is to recall that Proposition 2.5 states that every triangulation can be flipped to the Delaunay triangulation. However, there exist point sets in five or more dimensions whose flip

```
F    ($\mathcal{T}, uvw$)
1.      $x \leftarrow$ A          $(u, v, w)$
2.      $y \leftarrow$ A          $(w, v, u)$
3.      $T_D \leftarrow \{wvuy, uvwx\}$      { delete tetrahedra $wvuy$ and $uvwx$ }
4.      $T_C \leftarrow \emptyset$
5.      $V \leftarrow \emptyset$
6.      For $(a, b, c) \leftarrow (u, v, w)$, $(v, w, u)$, and $(w, u, v)$
7.          $\alpha \leftarrow$ O       3D$(a, b, x, y)$
8.          If $\alpha > 0$
9.              $T_D \leftarrow T_D \cup \{abxy\}$      { delete tetrahedron $abxy$ }
10.         else if $\alpha < 0$
11.             $T_C \leftarrow T_C \cup \{abyx\}$      { create tetrahedron $abyx$ }
12.         else $V \leftarrow V \cup \{c\}$      { $abxy$ is degenerate and $c$ is not part of the basic flip }
        { perform a flip that replaces $T_D$ with $T_C$ }
13.     If $V = \emptyset$      { basic flip: 2-3 flip or 3-2 flip or 4-1 flip }
            { note: if any simplex in $T_D$ is absent from $\mathcal{T}$, the flip is blocked }
14.         For each $\tau \in T_D$
15.             Call D        T                  $(\tau)$
16.         For each $\tau \in T_C$
17.             Call A    T                  $(\tau)$
18.     else      { extended flip: 2-2 flip or 4-4 flip or 3-1 flip or 6-2 flip or edge merge }
19.         $j \leftarrow 3 - |V|$
20.         Remove the vertices in $V$ from every simplex in $T_D$ and $T_C$, yielding $j$-simplices
21.         $\sigma \leftarrow$ a $j$-simplex in $T_D$
22.         For each $(d - j - 1)$-simplex $\zeta$ such that $\sigma * \zeta$ is a tetrahedron in $\mathcal{T}$
23.             For each $\tau \in T_D$
                    { note: if $\tau * \zeta$ is absent from $\mathcal{T}$, the flip is blocked }
24.                 Call D        T                  $(\tau * \zeta)$
25.             For each $\tau \in T_C$
26.                 Call A    T                  $(\tau * \zeta)$
```

Figure 4.8: Algorithm for performing a tetrahedral bistellar flip. The parameters to F are a triangulation $\mathcal{T}$ and a facet $uvw \in \mathcal{T}$ to flip. F assumes uvw can be flipped; comments identify places where this assumption could fail.

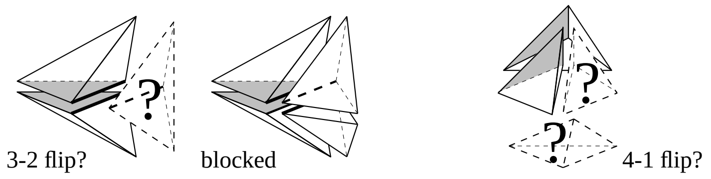

Figure 4.9: The shaded facet at left is not locally Delaunay. It can be removed by a 3-2 flip, but only if the third tetrahedron is present; the flip is blocked if more than three tetrahedra share the central edge (bold). The shaded facet at right can be removed by a 4-1 flip if the other two tetrahedra are present.

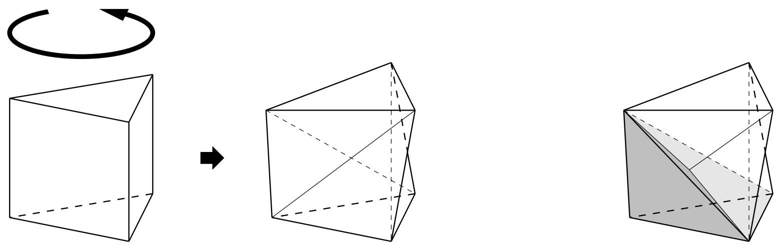

Figure 4.10: Schönhardt's untetrahedralizable polyhedron (center) is formed by rotating one end of a triangular prism (left), thereby creating three diagonal reflex edges. The convex hull of any four polyhedron vertices (right) sticks out.

graphs are not connected; they have triangulations that cannot be transformed to Delaunay by a sequence of bistellar flips. The question remains open in three and four dimensions. But even if all flip graphs for three-dimensional point sets are connected, flipping facets that are locally non-Delaunay does not suffice to find the Delaunay triangulation.

Despite the failure of the flip algorithm for three-dimensional Delaunay triangulations and weighted two-dimensional Delaunay triangulations, some Delaunay triangulation algorithms rely on bistellar flips, including several incremental vertex insertion algorithms and an algorithm for inserting a polygon into a CDT, the latter described in Section 5.8. In particular, if a new vertex is introduced into a Delaunay triangulation by a simple 1-4 flip (or by subdividing a facet or edge of the triangulation), and the flip algorithm is run before the triangulation is changed in any other way, the flip algorithm is guaranteed to restore the Delaunay property without getting stuck.

4.5 Three-dimensional constrained Delaunay triangulations

Constrained Delaunay triangulations generalize to three or more dimensions, but whereas every piecewise linear complex in the plane has a CDT, not every three-dimensional PLC has one. Worse yet, there exist simple polyhedra that do not have triangulations at all—that is, they cannot be subdivided into tetrahedra without creating new vertices (i.e. tetrahedron vertices that are not vertices of the polyhedron).

E. Schönhardt furnishes an example depicted in Figure 4.10. The easiest way to envision this polyhedron is to begin with a triangular prism. Imagine grasping the prism so that its bottom triangular face cannot move, while twisting the top triangular face so it rotates slightly about its center while remaining horizontal. This rotation breaks each of the three square faces into two triangular faces along a diagonal *reflex edge*—an edge at which the polyhedron is locally nonconvex. After this transformation, the upper left corner and lower right corner of each (former) 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 does not lie entirely within the polyhedron. Therefore, Schönhardt's polyhedron cannot be triangulated without additional vertices. It can be subdivided into tetrahedra with the addition of one vertex at its center.

Adding to the difficulty, it is NP-hard to determine whether a polyhedron has a triangulation, or whether it can be subdivided into tetrahedra with only k additional vertices for an arbitrary

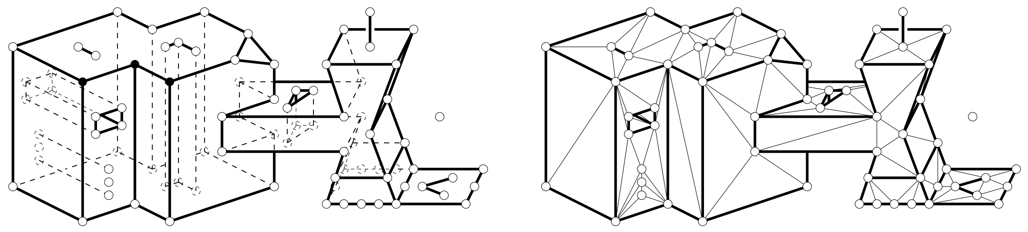

Figure 4.11: A three-dimensional piecewise linear complex and its constrained Delaunay triangulation. Each polygon and polyhedron may have holes, slits, and vertices in its relative interior. Each polyhedron may also have polygons in its interior.

constant *k*.

The following sections discuss triangulations and CDTs of polyhedra and PLCs in three dimensions. It is possible to refine any polyhedron or PLC by adding new vertices on its edges so that it has a constrained Delaunay triangulation. This fact makes CDTs useful in three dimensions.

4.5.1 Piecewise linear complexes and their triangulations in $\mathbb{R}^d$

The domain over which a general-dimensional CDT is defined is a general-dimensional piecewise linear complex, which is a set of linear cells—vertices, edges, polygons, and polyhedra—as illustrated in Figure 4.11. The linear cells constrain how the complex can be triangulated: each linear cell in the complex must be a union of simplices in the triangulation. The union of the linear cells specifies the region to be triangulated.

Definition 4.3 (piecewise linear complex)**.** A *piecewise linear complex* (PLC) $\mathcal{P}$ is a finite set of linear cells that satisfies the following properties.

- The vertices and edges in $\mathcal{P}$ form a simplicial complex.
- For each linear cell $f \in \mathcal{P}$, the boundary of f is a union of linear cells in $\mathcal{P}$.
- If two distinct linear cells $f, g \in \mathcal{P}$ intersect, their intersection is a union of linear cells in $\mathcal{P}$, all having lower dimension than at least one of f or g.

As in the plane, the edges in $\mathcal{P}$ are called *segments*. Its *underlying space* is $|\mathcal{P}| = \bigcup_{f \in \mathcal{P}} f$, which is usually the domain to be triangulated. The *faces* of a linear cell $f \in \mathcal{P}$ are the linear cells in $\mathcal{P}$ that are subsets of f, including f itself.

A triangulation of a PLC must cover every polyhedron, respect every polygon, and include every segment and vertex.

Definition 4.4 (triangulation of a PLC)**.** Let $\mathcal{P}$ be a PLC. A *triangulation of* $\mathcal{P}$ is a simplicial complex $\mathcal{T}$ such that $\mathcal{P}$ and $\mathcal{T}$ have the same vertices, every linear cell in $\mathcal{P}$ is a union of simplices in $\mathcal{T}$, and $|\mathcal{T}| = |\mathcal{P}|$.

Because this definition does not allow $\mathcal{T}$ to have new vertices absent from $\mathcal{P}$, every edge in $\mathcal{P}$ must appear in $\mathcal{T}$. However, the polygons in $\mathcal{P}$ may be subdivided into triangles in $\mathcal{T}$.

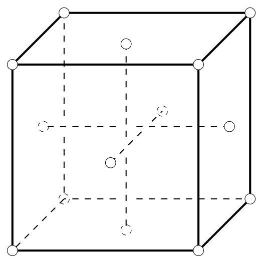

Figure 4.12: A convex PLC with no triangulation.

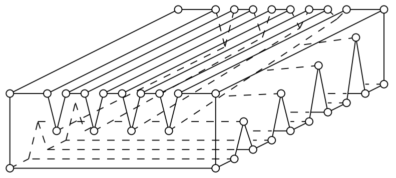

Figure 4.13: Chazelle's polyhedron.

Schönhardt's polyhedron shows that not every PLC has a triangulation. Every convex polyhedron has a triangulation; what about convex polyhedra with internal segments? Figure 4.12 illustrates a PLC with no triangulation, consisting of a cube inside which three orthogonal segments pass by each other but do not intersect. If any one of the segments is omitted, the PLC has a triangulation. This example shows that, unlike with planar triangulations, it is not always possible to insert a new edge into a tetrahedralization.

Because some polyhedra and PLCs do not have triangulations, Steiner triangulations are even more important in three dimensions than in the plane.

Definition 4.5 (Steiner triangulation of a PLC)**.** Let $\mathcal{P}$ be a PLC. A *Steiner triangulation of* $\mathcal{P}$, also known as a *conforming triangulation of* $\mathcal{P}$ or a *mesh of* $\mathcal{P}$, is a simplicial complex $\mathcal{T}$ such that $\mathcal{T}$ contains every vertex in $\mathcal{P}$ and possibly more, every linear cell in $\mathcal{P}$ is a union of simplices in $\mathcal{T}$, and $|\mathcal{T}| = |\mathcal{P}|$. The new vertices in $\mathcal{T}$, not present in $\mathcal{P}$, are called *Steiner points*. A *Steiner Delaunay triangulation* of $\mathcal{P}$, also known as a *conforming Delaunay triangulation* of $\mathcal{P}$, is a Steiner triangulation of $\mathcal{P}$ in which every simplex is Delaunay.

Every n-vertex polyhedron has a Steiner triangulation with at most $O(n^2)$ vertices, found by constructing a *vertical decomposition* of the polyhedron. The same is true for PLCs of complexity n. Unfortunately, there are polyhedra for which it is not possible to do better; Figure 4.13 depicts Chazelle's polyhedron, which has n vertices and $O(n)$ edges, but cannot be divided into fewer than $\Theta(n^2)$ convex bodies. The worst-case complexity of subdividing a polyhedron is related to its number of reflex edges: there is an algorithm that divides any polyhedron with r reflex edges into $O(n + r^2)$ tetrahedra, and some polyhedra with r reflex edges cannot be divided into fewer than $\Omega(n + r^2)$ convex bodies.

It appears likely, though it is proved only in two dimensions, that there exist PLCs whose smallest Steiner Delaunay triangulations are asymptotically larger than their smallest Steiner triangulations. There are algorithms that can find a Steiner Delaunay tetrahedralization of any three-dimensional polyhedron, but they might introduce a superpolynomial number of new vertices. No

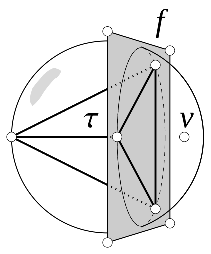

Figure 4.14: A constrained Delaunay tetrahedron τ.

known algorithm for finding Steiner Delaunay tetrahedralizations is guaranteed to introduce only a polynomial number of new vertices, and no algorithm of any complexity has been offered for four- or higher-dimensional Steiner Delaunay triangulations. Moreover, the existing algorithms all seem to introduce an unnecessarily large number of vertices near small domain angles. These problems can be partly remediated by Steiner CDTs.

4.5.2 The constrained Delaunay triangulation in $\mathbb{R}^3$

Three-dimensional constrained Delaunay triangulations aspire to retain most of the advantages of Delaunay triangulations while respecting constraints. But Figures 4.10, 4.12, and 4.13 demonstrate that some PLCs, even some polyhedra, have no triangulation at all. Moreover, some polyhedra that do have triangulations do not have CDTs. Nevertheless, CDTs are useful because, if we are willing to add new vertices, every three-dimensional PLC has a Steiner CDT, and a Steiner CDT might require many fewer vertices than a Steiner Delaunay triangulation.

As in the plane, there are several equivalent definitions of “constrained Delaunay triangulation” in three dimensions. The simplest is that a CDT is a triangulation of a PLC in which every facet not included in a PLC polygon is locally Delaunay. A CDT differs from a Delaunay triangulation in three ways: it is not necessarily convex, it is required to respect a PLC, and the facets of the CDT that are included in PLC polygons are exempt from being locally Delaunay.

Recall from Definition 2.11 that a simplex σ *respects* a PLC $\mathcal{P}$ if $\sigma \subseteq |\mathcal{P}|$ and for every $f \in \mathcal{P}$ that intersects σ, $f \cap \sigma$ is a union of faces of σ. By Definition 2.13, two points x and y are *visible* to each other if xy respects $\mathcal{P}$. A linear cell in $\mathcal{P}$ that intersects the relative interior of xy but does not include xy *occludes* the visibility between x and y. The primary definition of CDT specifies that every tetrahedron is constrained Delaunay, defined as follows.

Definition 4.6 (constrained Delaunay)**.** In the context of a PLC $\mathcal{P}$, a simplex σ is *constrained Delaunay* if $\mathcal{P}$ contains the vertices of σ, σ respects $\mathcal{P}$, and there is an open circumball of σ that contains no vertex in $\mathcal{P}$ that is visible from any point in the relative interior of σ.

Figure 4.14 depicts a constrained Delaunay tetrahedron τ. Every face of τ whose relative interior intersects the polygon f is included in f, so τ respects $\mathcal{P}$. The open circumball of τ contains one vertex v, but v is not visible from any point in the interior of τ, because f occludes its visibility.

Definition 4.7 (constrained Delaunay triangulation)**.** Let $\mathcal{P}$ be a three-dimensional PLC. A *constrained Delaunay triangulation* (CDT) of $\mathcal{P}$ is a triangulation of $\mathcal{P}$ in which every tetrahedron

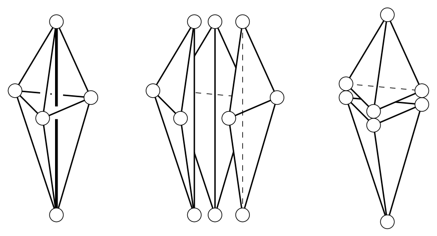

Figure 4.15: Left: a PLC with no CDT. Center: the sole tetrahedralization of this PLC. Its three tetrahedra are not constrained Delaunay. Right: the two Delaunay tetrahedra do not respect the central segment.

is constrained Delaunay, and every dangling triangle (i.e. not a face of any tetrahedron) is also constrained Delaunay.

Figure 4.11 illustrates a PLC and its CDT. Observe that the PLC has a polygon that is not a face of any polyhedron; this face is triangulated with constrained Delaunay triangles.

Figure 4.15 illustrates a PLC that has no CDT because of a segment that runs vertically through the domain interior. There is only one tetrahedralization of this PLC—composed of three tetrahedra encircling the central segment—and its tetrahedra are not constrained Delaunay, because each of them has a visible vertex in its open circumball. Whereas polygons usually block enough visibility to ensure their presence in a CDT, segments usually do not. But segments can dictate that a CDT does not exist at all. If the central segment in Figure 4.15 is removed, the PLC has a CDT made up of two tetrahedra.

A *Steiner CDT* or *conforming CDT* of $\mathcal{P}$ is a Steiner triangulation of $\mathcal{P}$ in which every tetrahedron is constrained Delaunay, and every dangling triangle (i.e. not a face of any tetrahedron) is also constrained Delaunay. A PLC with no CDT has a Steiner CDT, but one or more Steiner points must be added on its segments. For example, the PLC in Figure 4.15 has a Steiner CDT with one Steiner point on its central segment.

4.5.3 The CDT Theorem

Although not all piecewise linear complexes have constrained Delaunay triangulations, there is an easy-to-test, sufficient (but not necessary) condition that guarantees that a CDT exists. A three-dimensional PLC $\mathcal{P}$ is *edge-protected* if every edge in $\mathcal{P}$ is strongly Delaunay.

Theorem 4.9 (CDT Theorem)**.** *Every edge-protected PLC has a CDT.* □

It is not sufficient for every edge in $\mathcal{P}$ to be Delaunay. If all six vertices of Schönhardt's polyhedron lie on a common sphere, then all of its edges (and all its faces) are Delaunay, but it still has no 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.

What if a PLC that one wishes to triangulate is not edge-protected? One can make it edge-protected by adding vertices on its segments—a task that any Delaunay mesh generation algorithm must do anyway. The augmented PLC has a CDT, which is a Steiner CDT of the original PLC.

Figure 4.16 illustrates the difference between using a Delaunay triangulation and using a CDT for mesh generation. With a Delaunay triangulation, the mesh generator must insert new vertices

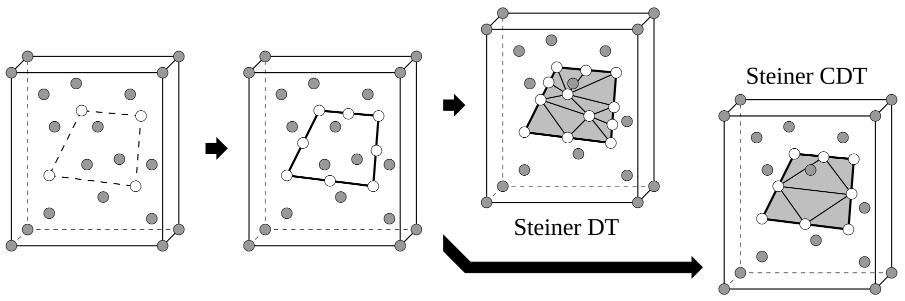

Figure 4.16: Comparison of Steiner Delaunay triangulations and Steiner CDTs. For clarity, vertices inside each box are shown, but tetrahedra are not. For both types of triangulation, missing segments are recovered by inserting new vertices until each segment is a union of strongly Delaunay edges. In a Steiner Delaunay triangulation, additional vertices are inserted until each polygon is a union of strongly Delaunay triangles. In a Steiner CDT, no additional vertices need be inserted; the polygons are recovered by computing a CDT.

that guarantee that every segment is a union of Delaunay (preferably strongly Delaunay) edges, and every polygon is a union of Delaunay (preferably strongly Delaunay) triangles. With a CDT, new vertices must be inserted that guarantee that every segment is a union of strongly Delaunay edges; but then the augmented PLC is edge-protected, and the CDT Theorem guarantees that the polygons can be recovered without inserting any additional vertices. The advantage of a CDT is that many fewer vertices might be required.

Testing whether a PLC $\mathcal{P}$ is edge-protected is straightforward. Form the Delaunay triangulation of the vertices in $\mathcal{P}$. If a segment $\sigma \in \mathcal{P}$ is missing from the triangulation, then σ is not strongly Delaunay, and $\mathcal{P}$ is not edge-protected. If σ is present, it is Delaunay. If the symbolic perturbations described in Section 2.9 are used to make the vertices in $\mathcal{P}$ generic, then every Delaunay edge is strongly Delaunay; so if every segment in $\mathcal{P}$ is present, $\mathcal{P}$ is edge-protected. (If symbolic perturbations are not used, then testing whether a Delaunay segment σ is strongly Delaunay is equivalent to determining whether the Voronoi polygon dual to σ is nondegenerate.)

4.5.4 Properties of the constrained Delaunay triangulation in $\mathbb{R}^3$

This section summarizes the properties of three-dimensional CDTs.

The Delaunay Lemma for three-dimensional CDTs provides an alternative definition of CDT: a triangulation of a PLC $\mathcal{P}$ is a CDT if and only if every one of its facets is locally Delaunay *or* is included in a polygon in $\mathcal{P}$.

Lemma 4.10 (Constrained Delaunay Lemma)**.** *Let $\mathcal{P}$ be a PLC in which every linear cell is a face of some polyhedron in $\mathcal{P}$, so there are no dangling polygons. Let $\mathcal{T}$ be a triangulation of $\mathcal{P}$. The following three statements are equivalent.*

(i) *Every tetrahedron in $\mathcal{T}$ is constrained Delaunay (i.e. $\mathcal{T}$ is constrained Delaunay).*

(ii) *Every facet in $\mathcal{T}$ not included in a polygon in $\mathcal{P}$ is constrained Delaunay.*

(iii) *Every facet in $\mathcal{T}$ not included in a polygon in $\mathcal{P}$ is locally Delaunay.* □

A constrained Delaunay triangulation $\mathcal{T}$ of $\mathcal{P}$ induces a two-dimensional triangulation of each polygon $f \in \mathcal{P}$, namely $\mathcal{T}|_f = \{\sigma \in \mathcal{T} : \sigma \subseteq f\}$. Statement (ii) above implies that the triangles in $\mathcal{T}|_f$ need not be constrained Delaunay with respect to $\mathcal{P}$—but they *are* constrained Delaunay with respect to the polygon f, in the following sense.

Proposition 4.11. *Let $\mathcal{T}$ be a CDT of a three-dimensional PLC $\mathcal{P}$. Let $f \in \mathcal{P}$ be a polygon. Let $\mathcal{T}|_f$ be the set of simplices in $\mathcal{T}$ that are included in f. Let $\mathcal{P}|_f$ be the set of faces of f (including f itself); $\mathcal{P}|_f$ is a two-dimensional PLC embedded in three-dimensional space. Then $\mathcal{T}|_f$ is a CDT of $\mathcal{P}|_f$.* □

A PLC is *generic* if its vertices are generic. A generic PLC has a unique CDT, if it has one at all.

Proposition 4.12. *A generic piecewise linear complex has at most one constrained Delaunay triangulation.* □

A consequence of Propositions 4.11 and 4.12 is that, if a PLC is generic, a CDT construction algorithm can begin by computing the two-dimensional CDTs of the polygons, then use them to help compute the three-dimensional CDT of the PLC, secure in the knowledge that the polygon triangulations will match the volume triangulation.

CDTs inherit the optimality properties of Delaunay triangulations described in Section 4.3, albeit with respect to a smaller set of triangulations, namely the triangulations of a PLC. However, if a PLC has no CDT, finding the optimal triangulation is an open problem.

Proposition 4.13. *Let $\mathcal{P}$ be a PLC. If $\mathcal{P}$ has a CDT, then every CDT of $\mathcal{P}$ minimizes the largest min-containment ball, compared with all other triangulations of $\mathcal{P}$. Every CDT of $\mathcal{P}$ also optimizes the criteria discussed in Propositions 4.5 and 4.8.* □

4.6 Notes and exercises

The upper bound of $\Theta(n^{\lceil d/2 \rceil})$ simplices in an n-vertex triangulation follows from McMullen's celebrated Upper Bound Theorem [145] of 1970. Seidel [191] gives a one-paragraph proof of the asymptotic bound.

Rajan [173] shows that the Delaunay triangulation minimizes the largest min-containment ball in any dimensionality, thereby generalizing the two-dimensional result of D'Azevedo and Simpson [67] and yielding Proposition 4.7. For an algebraic proof of Proposition 4.6 based on quadratic program duality, see Lemma 3 of Rajan [173]. Rippa [176] shows that the Delaunay triangulation in the plane minimizes the piecewise linear interpolation error for bivariate functions of the form $Ax^2 + Ay^2 + Bx + Cy + D$, measured in the L_q-norm for every $q \geq 1$, and Melissaratos [146] generalizes Rippa's result to higher dimensions, yielding Proposition 4.5. Shewchuk [204] extends all these optimality results to CDTs. The error bound for piecewise linear interpolation given in Section 4.3 is by Waldron [221].

Lawson [132] proves the claim from Section 4.4 that there are only two triangulations of the configuration of vertices involved in a basic bistellar flip. An earlier paper by Lawson [130] shows that for every planar point set, the flip graph is connected. Santos [184, 185] gives examples of point sets in five or more dimensions whose flip graphs are not connected. Joe [118] gives an example of a tetrahedralization for which the flip algorithm is stuck and can make no progress toward the Delaunay tetrahedralization. Edelsbrunner and Shah [91] give an example of a triangulation in the plane and a set of weights for which the flip algorithm is stuck and can make no progress toward the weighted Delaunay triangulation. The fact that the flip algorithm does not get stuck after a single vertex is introduced into a Delaunay triangulation by subdivision is proved by Joe [119], and by Edelsbrunner and Shah [91] for weighted Delaunay triangulations.

Schönhardt's polyhedron was discovered by Schönhardt [187], and Chazelle's polyhedron by Chazelle [44]. The NP-hardness of determining whether a polyhedron has a triangulation, cited in Section 4.5, is proved by Ruppert and Seidel [181]. Chazelle [44] proposes the vertical decomposition of a polyhedron, and Chazelle and Palios [45] give an algorithm that subdivides any n-vertex polyhedron with r reflex edges into $O(n + r^2)$ tetrahedra. This bound is optimal for the worst polyhedra.

The notion of a PLC was proposed by Miller, Talmor, Teng, Walkington, and Wang [149].[1]

Algorithms for computing a Steiner Delaunay triangulation of a PLC include those by Murphy, Mount, and Gable [156], Cohen-Steiner, Colin de Verdière, and Yvinec [65], Cheng and Poon [56], Cheng, Dey, Ramos, and Ray [52], and Rand and Walkington [174]. None has a polynomial bound on the number of new vertices.

CDTs were generalized to three or more dimensions by Shewchuk [204], whose paper includes proofs of the CDT Theorem and the properties of three-dimensional CDTs given in Section 4.5.4.

Exercises

1. Definition 4.3 of *piecewise linear complex* implies that if the interior of a segment intersects the interior of a polygon, the segment is entirely included in the polygon. Prove it.

2. Show that the edges and triangular faces of a strongly Delaunay tetrahedron are strongly Delaunay.

3. Prove Proposition 4.2. Consult Figure 2.10 for inspiration.

4. Prove Proposition 4.11.

5. Exercise 7 in Chapter 2 asks for a proof of a fact about constrained Delaunay triangles in the plane. Give a counterexample that demonstrates that the analogous fact is *not* true of constrained Delaunay tetrahedra in three dimensions.

6. Design an algorithm that adds vertices to a three-dimensional PLC so that the augmented PLC has a CDT.

[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. Our definition is different from that of Miller et al., but nearly equivalent, with one true difference: Miller et al. do not impose the restriction that the vertices and edges form a simplicial complex; they permit vertices to lie in the relative interior of an edge. Disallowing such vertices simplifies our presentation while entailing no essential loss of generality, because edges with vertices in their relative interiors can be subdivided into edges that obey the restriction.

Chapter 5

Algorithms for constructing Delaunay triangulations in $\mathbb{R}^3$

The most popular algorithms for constructing Delaunay triangulations in $\mathbb{R}^3$ are incremental insertion and gift-wrapping algorithms, both of which generalize to three or more dimensions with little difficulty. This chapter reprises those algorithms, with attention to the aspects that are different in three dimensions. In particular, the analysis of the running time of point location with a conflict graph is more complicated in three dimensions than in the plane. We use this gap as an opportunity to introduce a more sophisticated vertex ordering and its analysis. Instead of fully randomizing the order in which vertices are inserted, we recommend using a *biased randomized insertion order* that employs just enough randomness to ensure that the expected running time is the worst-case optimal $O(n^2)$—or better yet, $O(n \log n)$ time for the classes of point sets most commonly triangulated in practice—while maintaining enough spatial locality that implementations of the algorithm use the memory hierarchy more efficiently. This vertex ordering, combined with a simpler point location method, yields the fastest three-dimensional Delaunay triangulators in practice.

CDTs have received much less study in three dimensions than in two. There are two classes of algorithm available: gift-wrapping and incremental polygon insertion. Gift-wrapping is easier to implement; it is not much different in three dimensions than in two. It runs in $O(nh)$ time for Delaunay triangulations and $O(nmh)$ time for CDTs, where n is the number of vertices, m is the total complexity of the PLC's polygons, and h is the number of tetrahedra produced.

Perhaps the fastest three-dimensional CDT construction algorithm in practice is similar to the one we advocate in two dimensions. First, construct a Delaunay triangulation of the PLC's vertices, then insert its polygons one by one with a flip algorithm described in Section 5.8. This algorithm constructs a CDT in $O(n^2 \log n)$ time, though there are reasons to believe it will run in $O(n \log n)$ time on most PLCs in practice. Be forewarned, however, that this algorithm only works on edge-protected PLCs. This is rarely a fatal restriction, because a mesh generation algorithm that uses CDTs should probably insert vertices on the PLC's edges to make it edge-protected and ensure that it has a CDT.

Procedure	Purpose
A T (u, v, w, x)	Add a positively oriented tetrahedron *uvwx*
D T (u, v, w, x)	Delete a positively oriented tetrahedron *uvwx*
A (u, v, w)	Return a vertex x such that *uvwx* is a positively oriented tetrahedron
A 2V (u)	Return vertices v, w, x such that *uvwx* is a positively oriented tetrahedron

Figure 5.1: An interface for a three-dimensional triangulation data structure.

5.1 A dictionary data structure for tetrahedralizations

Figure 5.1 summarizes an interface for storing a tetrahedral complex, analogous to the interface for planar triangulations in Section 3.2. Two procedures, A T and D T - , specify a tetrahedron to be added or deleted by listing its vertices with a positive orientation, as described in Section 3.1. The procedure A recovers the tetrahedron adjoining a specified oriented triangular face, or returns $\emptyset$ if there is no such tetrahedron. The vertices of a tetrahedron may include the ghost vertex. The data structure enforces the invariant that only two tetrahedra may adjoin a triangular face, and only one on each side of the face.

The simplest fast implementation echoes the implementation described in Section 3.2. Store each tetrahedron *uvwx* four times in a hash table, keyed on the oriented faces *uvw*, *uxv*, *uwx*, and *vxw*. Then the first three procedures run in expected $O(1)$ time. To support A 2V queries, an array stores, for each vertex u, a triangle *uvw* such that the most recently added tetrahedron adjoining u has *uvw* for a face. As Section 3.2 discusses, these queries take expected $O(1)$ time in most circumstances, but not when the most recently added tetrahedron adjoining u has subsequently been deleted.

The interface and data structure extend easily to permit the storage of triangles or edges that are not part of any tetrahedron, but it does not support fast adjacency queries on edges.

5.2 Delaunay vertex insertion in $\mathbb{R}^3$

The Bowyer–Watson algorithm extends in a straightforward way to three (or more) dimensions. Recall that the algorithm inserts a vertex u into a Delaunay triangulation in four steps. First, find one tetrahedron whose open circumball contains u (point location). Second, a depth-first search in the triangulation finds all the other tetrahedra whose open circumballs contain u, in time proportional to their number. Third, delete these tetrahedra, as illustrated in Figure 5.2. The union of the deleted tetrahedra is a star-shaped polyhedral cavity. Fourth, for each triangular face of the cavity, create a new tetrahedron joining it with u, as illustrated.

To support inserting vertices that lie outside the triangulation, each triangular face on the boundary of the triangulation adjoins a *ghost tetrahedron* analogous to the ghost triangles of Section 3.4, having three solid vertices and a ghost vertex g. A tetrahedron that is not a ghost is called *solid*. Let *vwx* be a boundary triangle, oriented so its back adjoins a positively oriented solid tetrahedron *xwvy*. The incremental insertion algorithm stores a positively oriented ghost

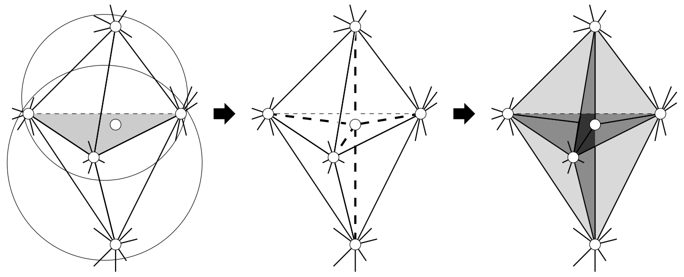

Figure 5.2: The Bowyer–Watson algorithm in three dimensions. A new vertex falls in the open circumballs of the two tetrahedra illustrated at left. These tetrahedra may be surrounded by other tetrahedra, which for clarity are not shown. The two tetrahedra and the face they share (shaded) are deleted. At center, the five new Delaunay edges. At right, the nine new Delaunay triangles—one for each edge of the cavity. Six new tetrahedra are created—one for each facet of the cavity.

tetrahedron *vwxg* in the triangulation data structure.

When a new vertex is inserted, there are two cases in which *vwxg* must be deleted, i.e. *vwx* is no longer a boundary triangle: if a vertex is inserted in the open halfspace in front of *vwx*, or if a newly inserted vertex lies in the open circumdisk of *vwx* (i.e. it is coplanar with *vwx* and in its open diametric ball). Call the union of these two regions the *outer halfspace* of *vwx*. It is the set of points in the open circumball of *vwxg* in the limit as *g* moves away from the triangulation.

The following pseudocode details the Bowyer–Watson algorithm in three dimensions, omitting point location. The parameters to I V 3D are a vertex *u* to insert and a positively oriented tetrahedron *vwxy* whose open circumball contains *u*. In this pseudocode, all the triangles and tetrahedra are oriented, so the vertex order matters.

```
I V 3D($u, vwxy$)

  1. Call D T ($v, w, x, y$).
  2. Call D C 3D($u, xwv$), D C 3D($u, yvw$), D C 3D($u, vyx$), and D -
     C 3D($u, wxy$) to identify the other deleted tetrahedra and insert new tetra-
     hedra.

D C 3D($u, wxy$)

  3. Let $z$ = A ($w, x, y$); $wxyz$ is the tetrahedron on the other side of facet $wxy$
     from $u$.
  4. If $z = \emptyset$, then return, because the tetrahedron has already been deleted.
  5. If I S ($u, w, x, y, z$) $> 0$, then $uwxy$ and $wxyz$ are not Delaunay, so call
     D T ($w, x, y, z$). Call D C 3D($u, wxz$), D C 3D($u, xyz$),
     and D C 3D($u, ywz$) to recursively identify more deleted tetrahedra and in-
     sert new tetrahedra, and return.
  6. Otherwise, $wxy$ is a face of the polyhedral cavity, so call A T ($u, w, x, y$).
```

The correctness of I V 3D depends on the use of a ghost vertex. In particular, step 4 should not confuse the ghost vertex with $\emptyset$; the former marks the triangulation exterior and the latter marks the cavity. Unlike with the planar algorithm I V in Section 3.4, Step 4 is necessary for both unweighted and weighted Delaunay triangulations.

Step 5 requires a modification to the I S test discussed in Section 3.1: if *wxyz* is a ghost tetrahedron, then replace the formula (3.5) with a test of whether *u* lies in the outer half-space of *wxyz*. To adapt the code for a weighted Delaunay triangulation, replace (3.5) with O 4D(u^+, w^+, x^+, y^+, z^+), which tests whether u^+ is below the witness plane of *wxyz*. The parameter *vwxy* of I V 3D must be a tetrahedron whose witness plane is above u^+.

How expensive is vertex insertion, leaving out the cost of point location? The insertion of a single vertex into an *n*-vertex Delaunay triangulation can delete $\Theta(n^2)$ tetrahedra if the triangulation is the one depicted in Figure 4.2. However, a single vertex insertion can only create $\Theta(n)$ tetrahedra: observe that the boundary of the cavity is a planar graph, so the cavity has fewer than $2n$ boundary triangles.

It follows that during a sequence of *n* vertex insertion operations, at most $\Theta(n^2)$ tetrahedra are created. A tetrahedron can only be deleted if it is first created, so at most $\Theta(n^2)$ tetrahedra are deleted, albeit possibly most of them in a single vertex insertion. For the worst points sets, randomizing the vertex insertion order does not improve these numbers.

A special case that occurs frequently in practice—by all accounts it seems to be the norm—is the circumstance where the Delaunay triangulation has complexity linear, rather than quadratic, in the number of vertices, and moreover the intermediate triangulations produced during incremental insertion have expected linear complexity. For point sets with this property, a random insertion order guarantees that each vertex insertion will create and delete an expected constant number of tetrahedra, just as it does in the plane, and we shall see that the randomized incremental insertion algorithm with a conflict graph runs in expected $O(n \log n)$ time. This running time is often observed in practice, even in higher dimensions. Be forewarned, however, that there are point sets for which the final triangulation has linear complexity but the intermediate triangulations have expected quadratic complexity, thereby slowing down the algorithm dramatically.

Even for worst-case point sets, randomization helps to support fast point location. Recall that, excluding the point location step, the Bowyer–Watson algorithm runs in time proportional to the number of tetrahedra it deletes and creates, so the running time of the three-dimensional incremental insertion algorithm, *excluding* point location, is $O(n^2)$. With a conflict graph and a random insertion order, point location is no more expensive than this, so the randomized incremental insertion algorithm achieves a worst-case optimal expected running time of $O(n^2)$.

5.3 Biased randomized insertion orders

The advantage of inserting vertices in random order is that it guarantees that the expected running time of point location is optimal, and that pathologically slow circumstances like those illustrated in Figure 3.8 are unlikely to happen. But there is a serious disadvantage: randomized vertex insertions tend to interact poorly with the memory hierarchy in modern computers, especially virtual memory. Ideally, data structures representing tetrahedra and vertices that are close together geometrically should be close together in memory—a property called *spatial locality*—for better cache and virtual memory performance.

Fortunately, the permutation of vertices does not need to be uniformly random for the running time to be asymptotically optimal. A *biased randomized insertion order* (BRIO) is a permutation of the vertices that has strong spatial locality but retains enough randomness to obtain an expected running time of $O(n^2)$. Experiments show that a BRIO greatly improves the efficiency of the memory hierarchy—especially virtual memory.

Experiments also show that incremental insertion achieves superior running times in practice when it uses a BRIO but replaces the conflict graph with a point location method that simply walks from the previously inserted vertex toward the next inserted vertex; see Section 5.5. Although walking point location does not offer a strong theoretical guarantee on running time like a conflict graph does, this incremental insertion algorithm is perhaps the most attractive in practice, as it combines excellent observed speed with a simple implementation.

Let n be the number of vertices to triangulate. A BRIO orders the vertices in a sequence of *rounds* numbered zero through $\lceil \log_2 n \rceil$. Each vertex is assigned to the final round, round $\lceil \log_2 n \rceil$, with probability $1/2$. The remaining vertices are assigned to the second-last round with probability $1/2$, and so on. Each vertex is assign to round zero with probability $(1/2)^{\lceil \log_2 n \rceil} \leq 1/n$. The incremental insertion algorithm begins by inserting the vertices in round zero, then round one, and so on to round $\lceil \log_2 n \rceil$.

Within any single round, the vertices can be arranged in any order without threatening the worst-case expected running time of the algorithm, as Section 5.4 proves. Hence, we order the vertices within each round to create as much spatial locality as possible. One way to do this is to insert the vertices in the order they are encountered on a space-filling curve such as a Hilbert curve or a z-order curve. Another way is to store the vertices in an octree or k-d tree, refined so each leaf node contains only a few vertices; then order the vertices by a traversal of the tree. (Octree traversal is one way to sort vertices along a Hilbert or z-order curve.)

The tendency of vertices that are geometrically close together to be close together in the ordering does not necessarily guarantee that the data structures associated with them will be close together in memory. Nevertheless, experiments show that several popular Delaunay triangulation programs run faster with a BRIO than with a vertex permutation chosen uniformly at random, especially when the programs run out of main memory and have to resort to virtual memory.

Whether one uses the traditional randomized incremental insertion algorithm or a BRIO, one faces the problem of bootstrapping the algorithm, as discussed in Section 3.7. The most practical approach is to choose four affinely independent vertices, construct their Delaunay triangulation (a single tetrahedron), create four adjoining ghost tetrahedra, construct a conflict graph, and insert the remaining vertices in a random order (a uniformly chosen permutation or a BRIO). Even if the four bootstrap vertices are not chosen randomly, it is possible to prove that the expected asymptotic running time of the algorithm is not compromised.

5.4 Optimal point location by a conflict graph in $\mathbb{R}^3$

Section 3.6 describes how to use a conflict graph to perform point location. Conflict graphs generalize to higher dimensions, yielding a randomized incremental insertion algorithm that constructs three-dimensional Delaunay triangulations in expected $O(n^2)$ time, which is optimal in the worst case. Moreover, the algorithm runs in expected $O(n \log n)$ time in the special case where the Delaunay triangulation of a random subset of the input points has expected linear complexity.

Conflict graphs and the vertex redistribution algorithm extend to three or more dimensions straightforwardly, with no new ideas needed. A conflict is a vertex-tetrahedron pair consisting of an uninserted vertex and a tetrahedron whose open circumball contains it. For each uninserted vertex, the conflict graph records a tetrahedron that contains the vertex. If an uninserted vertex lies outside the growing triangulation, its conflict is an unbounded, convex ghost “tetrahedron,” each having one solid triangular facet on the triangulation boundary and three unbounded ghost facets. The ghost edges and ghost facets diverge from a point in the interior of the triangulation (recall Figure 3.10). For each tetrahedron, the conflict graph records a list of the uninserted vertices that choose that tetrahedron as their conflict. When a vertex is inserted into the triangulation, the conflict graph is updated exactly as described in Section 3.6.

Let us analyze the running time. No backward analysis is known for a BRIO, so the analysis given here differs substantially from that of Section 3.6. This analysis requires us to assume that the point set is generic, although the algorithm is just as fast for point sets that are not.

Let S be a generic set of n points in $\mathbb{R}^3$, not all coplanar. Let σ be a tetrahedron whose vertices are in S. We call σ a *j-tetrahedron* if its open circumball contains j vertices in S.

Because S is generic, $\text{Del}\, S$ is composed of all the 0-tetrahedra of S. However, the incremental insertion algorithm transiently constructs many other tetrahedra that do not survive to the end; these are 1-tetrahedra, 2-tetrahedra, and so forth. We wish to determine, for each j, how many j-tetrahedra exist and what is the probability that any one of them is constructed. From this we can determine the expected number of j-tetrahedra that appear.

The *triggers* of a j-tetrahedron are its four vertices, and the *stoppers* of a j-tetrahedron are the j vertices its open circumball contains. A tetrahedron is constructed if and only if all of its triggers precede all of its stoppers in the insertion order. The probability of that decreases rapidly as j increases.

Let f_j be the total number of j-tetrahedra, which depends on S, and let $F_j = \sum_{i=0}^{j} f_i$ be the total number of i-tetrahedra for all $i \leq j$. $\text{Del}\, S$ contains at most $O(n^2)$ tetrahedra, so $F_0 = f_0 = O(n^2)$. Unfortunately, it is a difficult open problem to find a tight bound on the number f_j; we must settle for a tight bound on the number F_j. The following proposition is an interesting use of the probabilistic method. It exploits the fact that if we compute the Delaunay triangulation of a random subset of S, we know the probability that any given j-tetrahedron will appear in the triangulation.

Proposition 5.1. *For a generic set S of n points, $F_j = O(j^2 n^2)$. If the Delaunay triangulation of a random r-point subset of S has expected $O(r)$ complexity for every $r < n$, then $F_j = O(j^3 n)$.*

P . Let R be a random subset of S, where each point in S is chosen with probability $1/j$. (This is a magical choice, best understood by noting that for any j-tetrahedron, the expected number of its stoppers in R is one.) Let $r = |R|$. Observe that r is a random variable with a binomial distribution. Therefore, the expected complexity of $\text{Del}\, R$ is $O(E[r^2]) = O(E[r]^2) = O(n^2/j^2)$.

Let σ be a k-tetrahedron for an arbitrary k. Because S is generic, $\text{Del}\, R$ contains σ if and only if R contains all four of σ’s triggers but none of its k stoppers. The probability of that is

$$p_k = \left(\frac{1}{j}\right)^4 \left(1 - \frac{1}{j}\right)^k.$$

If $k \leq j$, then

$$p_k \geq \left(\frac{1}{j}\right)^4 \left(1 - \frac{1}{j}\right)^j \geq \left(\frac{1}{j}\right)^4 \frac{1}{e},$$

where e is the base of the natural logarithm. This inequality follows from the identity $\lim_{j\to\infty}(1 + x/j)^j = e^x$.

The expected number of tetrahedra in Del R is

$$E[\text{size}(\text{Del}\, R)] = \sum_{k=0}^{n} p_k f_k \geq \sum_{k=0}^{j} p_k f_k \geq \sum_{k=0}^{j} \left(\frac{1}{j}\right)^4 \frac{f_k}{e} = \frac{F_j}{e j^4}.$$

Recall that this quantity is $O(n^2/j^2)$. Therefore,

$$F_j \leq e j^4 E[\text{size}(\text{Del}\, R)] = O(j^2 n^2).$$

If the expected complexity of the Delaunay triangulation of a random subset of S is linear, the expected complexity of Del R is $O(E[r]) = O(n/j)$, so $F_j = O(j^3 n)$. □

Proposition 5.1 places an upper bound on the number of j-tetrahedra with j small. Next, we wish to know the probability that any particular j-tetrahedron will be constructed during a run of the randomized incremental insertion algorithm.

Proposition 5.2. *If the permutation of vertices is chosen uniformly at random, then the probability that a specified j-tetrahedron is constructed during the randomized incremental insertion algorithm is less than $4!/j^4$.*

P . A j-tetrahedron appears only if its four triggers (vertices) appear in the permutation before its j stoppers. This is the probability that if four vertices are chosen randomly from the $j + 4$ triggers and stoppers, they will all be triggers; namely

$$\frac{1}{\binom{j+4}{4}} = \frac{j!4!}{(j+4)!} < \frac{4!}{j^4}.$$

□

Proposition 5.2 helps to bound the expected running time of the standard randomized incremental insertion algorithm, but we need a stronger proposition to bound the running time with a biased randomized insertion order.

Proposition 5.3. *If the permutation of vertices is a BRIO, as described in Section 5.3, then the probability q_j that a specified j-tetrahedron will be constructed during the incremental insertion algorithm is less than*

$$\frac{16 \cdot 4! + 1}{j^4}.$$

P . Let σ be a j-tetrahedron. Let i be the round in which its first stopper is inserted. The incremental insertion algorithm can create σ only if its last trigger is inserted in round i or earlier.

Consider a trigger that is inserted in round i or earlier for any $i \neq 0$. The probability of that trigger being inserted before round i is $1/2$. Therefore, the probability that all four triggers are inserted in round $i \neq 0$ or earlier is exactly 2^4 times greater than the probability that all four triggers are inserted before round i. Therefore, the probability q_j that the algorithm creates σ is bounded as follows.

$$\begin{aligned} q_j &\leq \text{Prob[last trigger round} \leq \text{first stopper round]} \\ &\leq 2^4 \text{ Prob[last trigger round} < \text{first stopper round]} + \text{Prob[last trigger round} = 0] \\ &\leq 16 \text{ Prob[all triggers appear before all stoppers in} \\ &\qquad\qquad \text{a uniform random permutation]} + (1/2)^{4\lceil \log_2 n \rceil} \\ &< 16 \frac{4!}{j^4} + \frac{1}{n^4} \\ &\leq \frac{16 \cdot 4! + 1}{j^4}. \end{aligned}$$

The fourth line of this inequality follows from Proposition 5.2. □

Theorem 5.4. *The expected running time of the randomized incremental insertion algorithm on a generic point set S in three dimensions, whether with a uniformly random permutation or a BRIO, is $O(n^2)$. If the Delaunay triangulation of a random r-point subset of S has expected $O(r)$ complexity for every $r \leq n$, then the expected running time is $O(n \log n)$.*

P . Section 3.6 shows that the total running time of the algorithm is proportional to the number of tetrahedra created plus the number of conflicts created. The expected number of tetrahedra created is $\sum_{j=0}^{n} q_j f_j$, with q_j defined as in Proposition 5.3. Because a j-tetrahedron participates in j conflicts, the expected number of conflicts created is $\sum_{j=0}^{n} j q_j f_j$. Hence, the expected running time is

$$O\left(\sum_{j=0}^{n} q_j f_j + \sum_{j=0}^{n} j q_j f_j\right) \leq O\left(n^2 + \sum_{j=1}^{n} j q_j f_j\right).$$

The $O(n^2)$ term accounts for the first term of the first summation, for which $j = 0$ and $q_j = 1$, because the algorithm constructs every 0-tetrahedron. The second summation is bounded as follows.

$$\begin{aligned}
\sum_{j=1}^{n} j q_j f_j &< \sum_{j=1}^{n} \frac{16 \cdot 4! + 1}{j^3} (F_j - F_{j-1}) \\
&= (16 \cdot 4! + 1) \left(\sum_{j=1}^{n} \frac{F_j}{j^3} - \sum_{k=1}^{n} \frac{F_{k-1}}{k^3} \right) \\
&= (16 \cdot 4! + 1) \left(\sum_{j=1}^{n-1} \frac{F_j}{j^3} + \frac{F_n}{n^3} - \sum_{k=2}^{n} \frac{F_{k-1}}{k^3} - F_0 \right) \\
&= (16 \cdot 4! + 1) \left(\sum_{j=1}^{n-1} \left(\frac{F_j}{j^3} - \frac{F_j}{(j+1)^3} \right) + \frac{F_n}{n^3} - F_0 \right) \\
&< (16 \cdot 4! + 1) \left(\sum_{j=1}^{n-1} \frac{F_j}{j^3 (j+1)^3} \left((j+1)^3 - j^3 \right) + \frac{F_n}{n^3} \right) \\
&= O\left(\sum_{j=1}^{n-1} \frac{j^2 n^2}{j^6} j^2 + n \right) \\
&= O\left(n^2 \sum_{j=1}^{n-1} \frac{1}{j^2} + n \right) \\
&= O(n^2).
\end{aligned}$$

The last line follows from Euler's identity $\sum_{j=1}^{\infty} 1/j^2 = \pi^2/6$. If the expected complexity of the Delaunay triangulation of a random subset of S is linear, $F_j = O(j^3 n)$, so the expected running time is $O(n \sum_{j=1}^{n-1}(1/j)) = O(n \log n)$. □

5.5 Point location by walking

In conjunction with a BRIO, a simple point location method called *walking* appears to outperform conflict graphs in practice, although there is no guarantee of a fast running time. A walking point location algorithm simply traces a straight line through the triangulation, visiting tetrahedra that intersect the line as illustrated in Figure 5.3, until it arrives at a tetrahedron that contains the new vertex. In conjunction with a vertex permutation chosen uniformly at random (rather than a BRIO), walking point location visits many tetrahedra and is very slow. But walking is fast in practice if it follows two guidelines: the vertices should be inserted in an order that has much spatial locality, such as a BRIO, and each walk should begin at the most recently created solid tetrahedron. Then the typical walk visits a small constant number of tetrahedra.

To avoid a long walk between rounds of a BRIO, the vertex order (e.g. the tree traversal or the direction of the space-filling curve) should be reversed on even-numbered rounds, so each round begins near where the previous round ends.

Researchers have observed that the three-dimensional incremental insertion algorithm with a BRIO and walking point location appears to run in linear time, not counting the initial $O(n \log n)$-

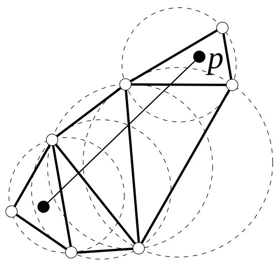

Figure 5.3: Walking to the triangle that contains p.

time computation of a BRIO. This observation holds whether they use a BRIO or a spatial ordering generated by traversing an octree with no randomness at all. Randomness is often unnecessary in practice—frequently, simply sorting the vertices along a space-filling curve will yield excellent speed—but because points sets like that illustrated in Figure 3.8 are common in practice, we recommend choosing a BRIO to prevent the possibility of a pathologically slow running time.

5.6 The gift-wrapping algorithm in $\mathbb{R}^3$

The simplest algorithm for retriangulating the cavity evacuated when a vertex is deleted from a three-dimensional Delaunay triangulation or CDT, or when a polygon is inserted or deleted in a CDT, is gift-wrapping. (See the bibliographic notes for more sophisticated vertex deletion algorithms, also based on gift-wrapping, that are asymptotically faster in theory.) The gift-wrapping algorithm described in Section 3.11 requires few new ideas to work in three (or more) dimensions. The algorithm constructs tetrahedra one at a time, and maintains a dictionary of unfinished triangular facets. The pseudocode for F and G W CDT can be adapted, with triangles replaced by tetrahedra, oriented edges replaced by oriented facets, and circumdisks replaced by circumballs.

The biggest change is that triangles, not segments, seed the algorithm. But the polygons in a PLC are not always triangles. Recall from Proposition 4.11 that a CDT of a PLC $\mathcal{P}$ induces a two-dimensional CDT of each polygon in $\mathcal{P}$. To seed the three-dimensional gift-wrapping algorithm, one can compute the two-dimensional CDT of a polygon (or every polygon), then enter each CDT triangle (twice, with both orientations) in the dictionary.

To gift-wrap a Delaunay triangulation, seed the algorithm with one strongly Delaunay triangle. One way to find one is to choose an arbitrary input point and its nearest neighbor. For the third vertex of the triangle, choose the input point that minimizes the radius of the circle through the three vertices. If the set of input points is generic, the triangle having these three vertices is strongly Delaunay.

If the input (point set or PLC) is not generic, gift-wrapping is in even greater danger in three dimensions than in the plane. Whereas the planar gift-wrapping algorithm can handle subsets of four or more cocircular points by identifying them and giving them special treatment, no such approach works reliably in three dimensions. Imagine a point set that includes six points lying on a common empty sphere. Suppose that gift-wrapping inadvertently tetrahedralizes the space around these points so they are the vertices of a hollow cavity shaped like Schönhardt's polyhedron (from Section 4.5). The algorithm will be unable to fill the cavity. By far the most practical solution is to symbolically perturb the points so that they are generic, as discussed in Section 2.9. The same perturbation should also be used to compute the two-dimensional CDTs of the PLC's polygons.

Another difficulty is that the input PLC might not have a CDT, in which case gift-wrapping will fail in one of two ways. One possibility is that the algorithm will fail to finish an unfinished facet, even though there is a vertex in front of that facet, because no vertex in front of that facet is visible from the facet's interior. This failure is easy to detect. The second possibility is that the algorithm will finish a facet by constructing a tetrahedron that is not constrained Delaunay, either because the tetrahedron's open circumball contains a visible vertex, or because the tetrahedron intersects the preexisting simplices wrongly (not in a complex). An attempt to gift-wrap Schönhardt's polyhedron brings about the last fate. The algorithm becomes substantially slower if it tries to detect these failures. Perhaps a better solution is to run the algorithm only on PLCs that are edge-protected or otherwise known to have CDTs.

A strange property of the CDT is that it is NP-hard to determine whether a three-dimensional PLC has a CDT, if the PLC is not generic. However, a polynomial-time algorithm is available for generic PLCs: run the gift-wrapping algorithm, and check whether it succeeded.

Gift-wrapping takes $O(nh)$ time for a Delaunay triangulation, or $O(nmh)$ time for a CDT, where n is the number of input points, m is the total complexity of the input polygons, and h is the number of tetrahedra in the CDT; h is usually linear in n, but could be quadratic in the worst case.

5.7 Inserting a vertex into a CDT in $\mathbb{R}^3$

Section 3.9 describes how to adapt the Bowyer–Watson vertex insertion algorithm to CDTs in the plane. The same adaptions work for three-dimensional CDTs, but there is a catch: even if a PLC $\mathcal{P}$ has a CDT, an augmented PLC $\mathcal{P} \cup \{v\}$ might not have one. This circumstance can be diagnosed after the depth-first search step of the Bowyer–Watson algorithm in one of two ways: by the fact that the cavity is not star-shaped, thus one of the newly created tetrahedra has nonpositive orientation, or by the fact that a segment or polygon runs through the interior of the cavity. An implementation can check explicitly for these circumstances, and signal that the vertex v cannot be inserted.

5.8 Inserting a polygon into a CDT

To "insert a polygon into a CDT" is to take as input the CDT $\mathcal{T}$ of some PLC $\mathcal{P}$ and a new polygon f to insert, and produce the CDT of $\mathcal{P}^f = \mathcal{P} \cup \{f\}$. It is only meaningful if $\mathcal{P}^f$ is a valid PLC—which implies that f's boundary is a union of segments in $\mathcal{P}$, among other things. It is only possible if $\mathcal{P}^f$ has a CDT. If $\mathcal{P}$ is edge-protected, then $\mathcal{P}^f$ is edge-protected (polygons play no role in the definition of "edge-protected"), and both have CDTs. But if $\mathcal{P}$ is not edge-protected, it is possible that $\mathcal{P}$ has a CDT and $\mathcal{P}^f$ does not; see Exercise 5.

The obvious algorithm echoes the segment insertion algorithm in Section 3.10: delete the tetrahedra whose interiors intersect f. All the simplices not deleted are still constrained Delaunay. Then retriangulate the polyhedral cavities on either side of f with constrained Delaunay simplices, perhaps by gift-wrapping. (Recall Figure 2.15.) Note that if $\mathcal{P}^f$ has no CDT, the retriangulation step will fail.

This section describes an alternative algorithm that uses bistellar flips to achieve the same result. One can construct a three-dimensional CDT of an n-vertex, ridge-protected PLC $\mathcal{P}$ in $O(n^2 \log n)$ time by first constructing a Delaunay triangulation of the vertices in $\mathcal{P}$ in expected $O(n^2)$ time with the randomized incremental insertion algorithm, then inserting the polygons one by one with the flip-based algorithm. For most PLCs that arise in practice, this CDT construction algorithm is likely to run in $O(n \log n)$ time and much faster than gift-wrapping.

The algorithm exploits the fact that when the vertices of the triangulation are lifted by the parabolic lifting map, every locally Delaunay facet lifts to a triangle where the lifted triangulation is locally convex. Say that a facet g, shared by two tetrahedra σ and τ, is *locally weighted Delaunay* if the lifted tetrahedra σ^+ and τ^+ adjoin each other at a dihedral angle, measured from above, of less than 180°. In other words, the interior of τ^+ lies above the affine hull of σ^+, and vice versa.

The algorithm's main idea is to move some of the lifted vertices vertically, continuously, and linearly so they rise above the paraboloid, and use bistellar flips to dynamically maintain local convexity as they rise. Recall from Figure 4.7 that a tetrahedral bistellar flip is a transition between the upper and lower faces of a 4-simplex. If the vertices of that simplex are moving vertically at different (but constant) speeds, they may pass through an instantaneous state in which the five vertices of the 4-simplex are cohyperplanar, whereupon the lower and upper faces are exchanged. In the facet insertion algorithm, this circumstance occurs when two lifted tetrahedra σ^+ and τ^+ that share a triangular facet g^+ become cohyperplanar, whereupon the algorithm uses the procedure F from Section 4.4 to perform a bistellar flip that deletes g.

The algorithm for inserting a polygon f into a triangulation $\mathcal{T}$ begins by identifying a region R: the union of the tetrahedra in $\mathcal{T}$ whose interiors intersect f, and thus must be deleted. The polygon insertion algorithm only performs flips in the region R. Let h be the plane aff f. Call the vertices in $\mathcal{P}$ on one (arbitrary) side of h *left vertices*, and the vertices on the other side *right vertices*. Vertices on h are neither. The flip algorithm linearly increases the heights of the vertices according to their distance from h, and uses flips to maintain locally weighted Delaunay facets in R as the heights change. Figure 5.4 is a sequence of snapshots of the algorithm at work.

Assign each vertex $v \in \mathcal{P}$ a time-varying *height* of $v_z(\kappa) = \|v\|^2 + \kappa d(v, h)$, where κ is the time and $d(v, h)$ is the Euclidean distance of v from h. (This choice of $d(\cdot, \cdot)$ is pedagogically useful but numerically poor; a better choice for implementation is to let $d(v, h)$ be the distance of v from h along one coordinate axis, preferably the axis most nearly perpendicular to h. This distance is directly proportional to the Euclidean distance, but can be computed without radicals.)

When a set of vertices is transformed affinely, its convex hull undergoes no combinatorial change. Likewise, an affine transformation of the vertex heights in a lifted triangulation does not change which facets are locally weighted Delaunay. In the facet insertion algorithm, however, each half of space undergoes a different affine transformation, so the simplices that cross the plane h change as the time κ increases. Observe that an algorithm in which only the heights of the right vertices change (at twice the speed) is equivalent. For numerical reasons, it is better to raise only half the vertices.

Let $\mathcal{P}(\kappa)$ be a time-varying weighted PLC, which is identical to $\mathcal{P}$ except that each right vertex v is lifted to a height of $v_z(\kappa)$. As κ increases, the algorithm F I P maintains a triangulation of R that is *weighted constrained Delaunay* with respect to $\mathcal{P}(\kappa)$, meaning that every triangular facet is locally weighted Delaunay except those included in a polygon.

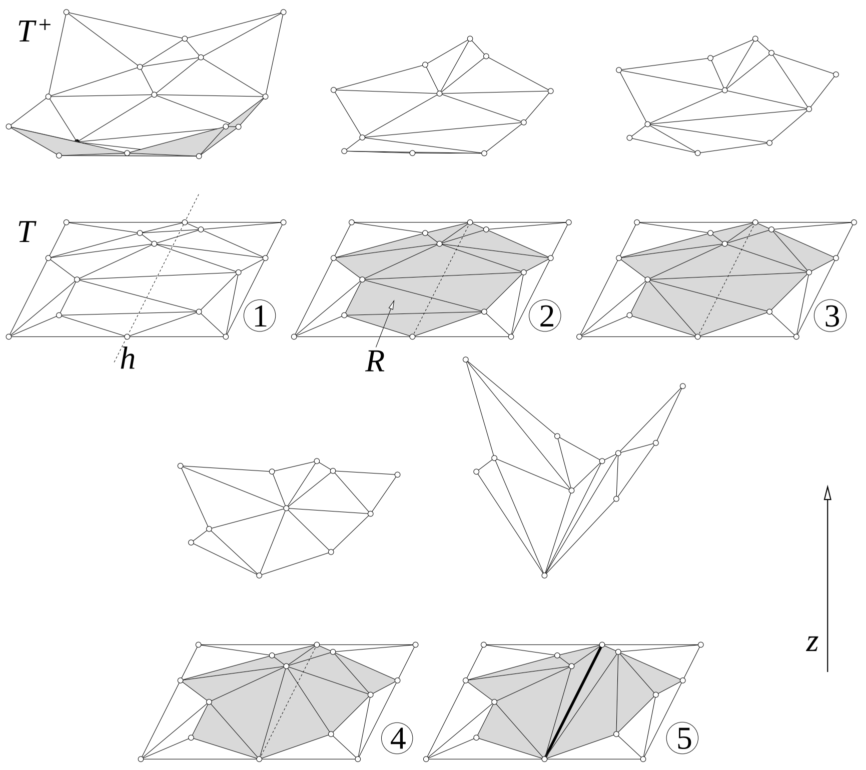

Figure 5.4: A two-dimensional example of inserting a segment into a CDT. The algorithm extends to any dimension.

Every simplex in the evolving triangulation that has no left vertex, or no right vertex, remains constrained Delaunay with respect to $\mathcal{P}(\kappa)$ as κ increases. The algorithm F I P deletes only simplices that have both a left and a right vertex *and* pass through f. All simplices outside the region R, or strictly on the boundary of R, remain intact.

When κ is sufficiently large, the flip algorithm reaches a state where no simplex in the region R has both a left vertex and a right vertex, hence f is a union of faces of the triangulation. At this time, the triangulation is the CDT of $\mathcal{P}^f$, and the job is done.

Pseudocode for F I P appears below. The loop (step 4) dynamically maintains the triangulation $\mathcal{T}$ as κ increases from 0 to ∞ and the lifted companions of the right vertices move up. For certain values of κ, the following event occurs: some facet g in the region R is no longer locally weighted Delaunay after time κ, because the two lifted tetrahedra that include g^+ are cohyperplanar at time κ. Upon this event, an update operation replaces these and other simplices that will not be locally weighted Delaunay after time κ with simplices that will be.

To ensure that it performs each bistellar flip at the right time, the algorithm maintains a priority queue (e.g. a binary heap) that stores any flip that might occur. For each facet g that could be flipped at some time in the future, the procedure C determines when g might be flipped and enqueues a flip event. The main loop of F I P repeatedly removes the flip with the least time from the priority queue, and performs a flip if the facet still exists, i.e. was not eliminated by other flips. When the queue is empty, $\mathcal{T}$ has transformed into the CDT of $\mathcal{P}^f$.

```
F I P ($\mathcal{T}, f$)

    1. Find one tetrahedron in $\mathcal{T}$ that intersects the interior of $f$ by a rotary search
       around an edge of $f$.
    2. Find all tetrahedra in $\mathcal{T}$ that intersect the interior of $f$ by a depth-first search.
    3. Initialize $Q$ to be an empty priority queue. For each facet $g$ in $\mathcal{T}$ that intersects
       the interior of $f$ and has a vertex on each side of $f$, call C ($g, Q$).
    4. While $Q$ is not empty
        (a) Remove $(g', \kappa)$ with minimum $\kappa$ from $Q$.
        (b) If $g'$ is still a facet in $\mathcal{T}$:
            (i) Call F ($\mathcal{T}, g'$) to eliminate $g'$.
            (i) For each facet $g$ that lies on the boundary of the cavity retriangulated
                by the flip and has a vertex on each side of $f$, call C ($g, Q$).

C ($g, Q$)

    1. Let $\sigma$ and $\tau$ be the tetrahedra that share the facet $g$.
    2. If the interior of $\sigma^+$ will be below aff $\tau^+$ at time $\infty$:
        (a) Compute the time $\kappa$ at which $\sigma^+$ and $\tau^+$ are cohyperplanar.
        (b) Insert $(g, \kappa)$ into the priority queue $Q$.
```

Just as gift-wrapping can fail when $\mathcal{P}$ is not generic, F F P can fail when $\mathcal{P}$ is not generic or simultaneous events occur for other reasons. For an implementation to succeed in practice, step 2 of C should perturb the vertex weights as described in Section 2.9 and in Exercise 2 of Chapter 3. It is possible for several events that take place at the same time (or nearly the same time, if roundoff error occurs) to come off the priority queue in an unexecutable order—recall that there are several circumstances annotated in the F pseudocode in which it might be impossible to perform a flip that eliminates a specified facet. In these circumstances, step 4(a) of F I P should dequeue an event with an admissible flip and hold back events with inadmissible flips until they become admissible.

The correctness proof for F I P is omitted, but it relies on the Constrained Delaunay Lemma. All the tetrahedra outside the region R remain constrained Delaunay, so their facets are constrained Delaunay, except those included in a polygon. When the algorithm is done, the original vertex heights are restored (returning them to the paraboloid). This is an affine transformation of the lifted right vertices, so the facets created by bistellar flips remain locally Delaunay, except the facets included in f. Therefore, every facet not included in a polygon is locally Delaunay. By the Constrained Delaunay Lemma, the final triangulation is a CDT of $\mathcal{P}^f$.

For an analysis of the running time, let m be the number of vertices in the region R, and let n be the number of vertices in $\mathcal{T}$.

Proposition 5.5. *Steps 3 and 4 of* F I P *run in $O(m^2 \log m)$ time.*

P . Every bistellar flip either deletes or creates an edge in the region R. Because the vertex weights vary linearly with time, an edge that loses the weighted constrained Delaunay property will never regain it; so once an edge is deleted, it is never created again. Therefore, F I -P deletes fewer than $m(m-1)/2$ edges and creates fewer than $m(m-1)/2$ edges over its lifetime, and thus performs fewer than m^2 flips. Each flip enqueues at most a constant number of events. Each event costs $O(\log m)$ time to enqueue and dequeue, yielding a total cost of $O(m^2 \log m)$ time. □

Miraculously, the worst-case running time for any sequence of F I P operations is only a constant factor larger than the worst-case time for one operation.

Proposition 5.6. *The running time of any sequence of valid* F I P *operations applied consecutively to a triangulation is* $O(n^2 \log n)$*, if step 1 of* F I P *is implemented efficiently enough.*

P . A sequence of calls to F I P , like a single call, has the property that every simplex deleted is never created again. (Every deleted simplex crosses a polygon, and cannot return after the polygon is inserted.) Therefore, a sequence of calls deletes fewer than $n(n-1)/2$ edges, creates fewer than $n(n-1)/2$ edges, and performs fewer than n^2 flips. Each flip enqueues a constant number of events. Each event costs $O(\log n)$ time to enqueue and dequeue, summing to $O(n^2 \log n)$ time for all events.

The cost of step 2 of F I P (the depth-first search) is proportional to the number of tetrahedra that intersect the relative interior of the polygon f. These tetrahedra either are deleted when f is inserted, or they intersect f's relative interior without crossing f. A tetrahedron can intersect the relative interiors of at most ten PLC polygons that it does not cross, so each tetrahedron is visited in at most eleven depth-first searches. At most $O(n^2)$ tetrahedra are deleted and created during a sequence of calls to F I P . Therefore, the total cost of step 2 over all calls to F I P is $O(n^2)$.

The cost of step 1 of F I P (identifying a tetrahedron that intersects the relative interior of f) is $O(n)$. This is a pessimistic running time; the worst case is achieved only if the edge used for the rotary search is an edge of $\Theta(n)$ facets. If $\Omega(n \log n)$ polygons are inserted into a single triangulation (which is possible but unlikely in practice), we must reduce the cost of step 1 below $O(n)$. This can be done by giving each segment a balanced search tree listing the adjoining facets in the triangulation, in rotary order around the segment. Then, step 1 executes in $O(\log n)$ time. The balanced trees are updated in $O(\log n)$ time per facet created or deleted. □

Recall from Section 4.4 that a bistellar flip replaces the top faces of a 4-simplex with its bottom faces. Because no face reappears after it is deleted, the sequence of flips performed during incremental polygon insertion is structurally similar to a four-dimensional triangulation. It has often been observed that most practical point sets have linear-size Delaunay triangulations in three or higher dimensions, so it seems like a reasonable inference that for most practical PLCs, the sequence of flips should have linear length. For those PLCs, incremental CDT construction with F I P runs in $\Theta(n \log n)$ time.

5.9 Notes and exercises

An incremental insertion algorithm that works in any dimension was discovered independently by Bowyer [33], Hermeline [111, 112], and Watson [222]. Bowyer and Watson submitted their articles to *Computer Journal* and found them published side by side in 1981.

Although there is rarely a reason to choose them over the Bowyer–Watson algorithm, there is a literature on vertex insertion algorithms that use bistellar flips. Joe [118] generalizes Lawson's flip-based vertex insertion algorithm [131] to three dimensions, Rajan [173] generalizes it to higher dimensions, Joe [119] improves the speed of Rajan's generalization, and Edelsbrunner and Shah [91] generalize Joe's latter algorithm to weighted Delaunay triangulations.

Clarkson and Shor [64] introduce conflict graphs and show that randomized incremental insertion with a conflict graph runs in expected $O(n^{\lceil d/2 \rceil})$ time for Delaunay triangulations in $d \geq 3$ dimensions. Amenta, Choi, and Rote [7] propose the idea of a biased randomized insertion order and give the analysis of the randomized incremental insertion algorithm in Section 5.4. The bound on the number of j-tetrahedra in Proposition 5.1 was itself a major breakthrough of Clarkson and Shor [64]. The simpler proof given here is due to Mulmuley [154]. See Seidel [194] for a backward analysis of the algorithm with a vertex permutation chosen uniformly at random, in the style of Section 3.6.

Guibas and Stolfi [106] give an algorithm for walking point location in a planar Delaunay triangulation, and Devillers, Pion, and Teillaud [72] compare walking point location algorithms in two and three dimensions. The discussion in Section 5.5 of combining a BRIO with walking point location relies on experiments reported by Amenta, Choi, and Rote [7].

The first gift-wrapping algorithm for constructing Delaunay triangulations in three or more dimensions appeared in 1979 when Brown [34] published his lifting map and observed that the 1970 gift-wrapping algorithm of Chand and Kapur [41] for computing general-dimensional convex hulls can construct Voronoi diagrams. The first gift-wrapping algorithm for constructing three-dimensional CDTs appears in a 1982 paper by Nguyen [159], but like the two-dimensional CDT paper of Frederick, Wong, and Edge [97], the author does not appear to have been aware of Delaunay triangulations at all. There is a variant of the gift-wrapping algorithm for CDTs that, by constructing the tetrahedra in a disciplined order and using other tricks to avoid visibility computations [200], runs in $O(nt)$ worst-case time, where n is the number of vertices and t is the number of tetrahedra produced.

Devillers [71] and Shewchuk [200] give gift-wrapping algorithms for deleting a degree-k vertex from a Delaunay or constrained Delaunay triangulation in $O(t \log k)$ time, where t is the number of tetrahedra created by the vertex deletion operation and can be as small as $\Theta(k)$ or as large as $\Theta(k^2)$. In practice, most vertices have a small degree, and naive gift-wrapping is usually fast enough.

The polygon insertion algorithm of Section 5.8 is due to Shewchuk [203]. Grislain and Shewchuk [105] show that it is NP-hard to determine whether a non-generic PLC has a CDT.

Exercises

1. You are using the incremental Delaunay triangulation algorithm to triangulate a cubical $\sqrt[3]{n} \times \sqrt[3]{n} \times \sqrt[3]{n}$ grid of n vertices. The vertices are not inserted in random order; instead, an

adversary chooses the order to make the algorithm as slow as possible. As an asymptotic function of n, what is the largest *total* number of changes that might be made to the mesh (i.e. tetrahedron creations and deletions, summed over all vertex insertions), and what insertion order produces that asymptotic worst case? Ignore the time spent doing point location.

2. Let S and T be two sets of points in $\mathbb{R}^3$, having s points and t points respectively. Suppose $S \cup T$ is generic. Suppose we are given $\text{Del}\, S$, and our task is to incrementally insert the points in T and thus construct $\text{Del}\,(S \cup T)$.

   We use the following algorithm. First, for each point p in T, find a tetrahedron or ghost tetrahedron in $\text{Del}\, S$ that contains p by brute force (checking each point against each tetrahedron), and thereby build a conflict graph in $O(ts^2)$ time. Second, incrementally insert the points in T in random order, with each permutation being equally likely.

   If we were constructing $\text{Del}\,(S \cup T)$ from scratch, with the insertion order wholly randomized, it would take at worst expected $O((s+t)^2) = O(s^2 + t^2)$ time. However, because S is *not* a random subset of $S \cup T$, the expected running time for this algorithm can be worse. An adversary could choose S so that inserting the points in T is slow.

   Prove that the expected time to incrementally insert the points in T is $O(t^2 + ts^2)$.

3. Extend the analysis of the randomized incremental insertion algorithm to point sets that are not generic. To accomplish that, we piggyback on the proof for generic point sets. Let S be a finite set of points in $\mathbb{R}^3$, not necessarily generic, and let $S[\omega]$ be the same points with their weights perturbed as described in Section 2.9. By Theorem 5.4, the randomized incremental insertion algorithm constructs $\text{Del}\, S[\omega]$ in expected $O(n^2)$ time, and in expected $O(n \log n)$ time in the optimistic case where the expected complexity of the Delaunay triangulation of a random subset of the points is linear.

   When the randomized incremental insertion algorithm constructs $\text{Del}\, S$, the tetrahedra and conflicts created and deleted are not necessarily the same as when it constructs $\text{Del}\, S[\omega]$, but we can argue that the numbers must be comparable.

   (a) Show that immediately after a vertex v is inserted during the construction of $\text{Del}\, S$, all the edges that adjoin v are strongly Delaunay. Show that immediately after v is inserted during the construction of $\text{Del}\, S[\omega]$, the same edges adjoin v, and perhaps others do too.

   (b) The boundary of the retriangulated cavity is planar. Use this fact to show that the number of tetrahedra created when v is inserted during the construction of $\text{Del}\, S$ cannot exceed the number of tetrahedra created when v is inserted during the construction of $\text{Del}\, S[\omega]$.

   (c) Recall that each polyhedral cell of the Delaunay subdivision of S has its vertices on a common sphere, and that the cell is subdivided into tetrahedra in $\text{Del}\, S[\omega]$. Show that the number of conflicts of a cell (vertices in its open circumball) in the Delaunay subdivision of S cannot exceed the number of conflicts of any of the corresponding tetrahedra in $\text{Del}\, S[\omega]$. Show that the number of conflicts created when v is inserted during the construction of $\text{Del}\, S$ cannot exceed the number of conflicts created when v is inserted during the construction of $\text{Del}\, S[\omega]$.

4. Recall from Figure 3.14 that gift-wrapping can fail to construct the Delaunay triangulation of a point set that is not generic. One way to make the algorithm robust is to use symbolic weight perturbations. A different way is to identify groups of points that are cospherical during the gift-wrapping step and triangulate them all at once.

   Design a tetrahedral gift-wrapping algorithm that implements the second suggestion, without help from any kind of perturbations. Recall from Figure 4.4 that polyhedra of the Delaunay subdivision cannot be subdivided into tetrahedra independently of each other. How does your solution ensure that these subdivisions will be consistent with each other?

5. Give an example of a PLC $\mathcal{P}$ that has a CDT and a polygon f such that $\mathcal{P} \cup \{f\}$ is a valid PLC but does not have a CDT.