Fast Segment Insertion and Incremental Construction of Constrained Delaunay Triangulations

Jonathan Richard Shewchuk, Brielin C. Brown

Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley, California 94720, USA

Abstract

The most commonly implemented method of constructing a constrained Delaunay triangulation (CDT) in the plane is to first construct a Delaunay triangulation, then incrementally insert the input segments one by one. For typical implementations of segment insertion, this method has a $\Theta(kn^2)$ worst-case running time, where n is the number of input vertices and k is the number of input segments.

We give a randomized algorithm for inserting a segment into a CDT in expected time linear in the number of edges the segment crosses. We demonstrate with a performance comparison that for segments that cross many edges, our algorithm is faster than gift-wrapping. We also show that a simple algorithm for *segment location*, which precedes segment insertion, is fast enough never to be a bottleneck in CDT construction. A result of Agarwal, Arge, and Yi implies that randomized incremental construction of CDTs by our segment insertion algorithm takes expected $O(n \log n + n \log^2 k)$ time. We show that this bound is tight by deriving a matching lower bound. Although there are CDT construction algorithms guaranteed to run in $O(n \log n)$ time, incremental CDT construction is easier to program and competitive in practice.

Lastly, we partly extend the analysis (albeit not the linear-time insertion algorithm) to randomized incremental CDT construction in three dimensions.

Keywords: constrained Delaunay triangulation, ϵ-net, randomized incremental construction, computational geometry

1. Introduction

The constrained Delaunay triangulation (CDT) in the plane, proposed by Lee and Lin [26], is a variant of the well-known Delaunay triangulation in which specified edges, sometimes called *segments*, are constrained to appear. The CDT is as close to being Delaunay as possible subject to those constraints. In particular, every edge of the CDT either is an input segment or is locally Delaunay. An edge in a triangulation is *locally Delaunay* if it is an edge of only one triangle, or it is an edge of two triangles and the Delaunay triangulation of those triangles' four vertices includes the edge, as illustrated at left in Figure 1.

The constraints imposed by CDTs have many uses, including representing boundaries of nonconvex objects, supporting better interpolation of discontinuous functions, and aiding the enforcement of boundary conditions in finite element meshes.

☆Supported in part by the National Science Foundation under Awards CCF-0635381, IIS-0915462, and CCF-1423560, in part by the University of California Lab Fees Research Program under Grant 09-LR-01-118889-OBRJ, and in part by an Alfred P. Sloan Research Fellowship.

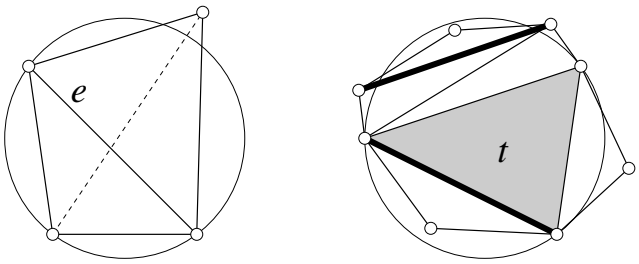

Figure 1: Left: the solid edge e is locally Delaunay. The dashed edge that crosses it is not. Right: The triangle t is constrained Delaunay, despite having two vertices inside its circumcircle. Bold lines represent segments, which block visibility.

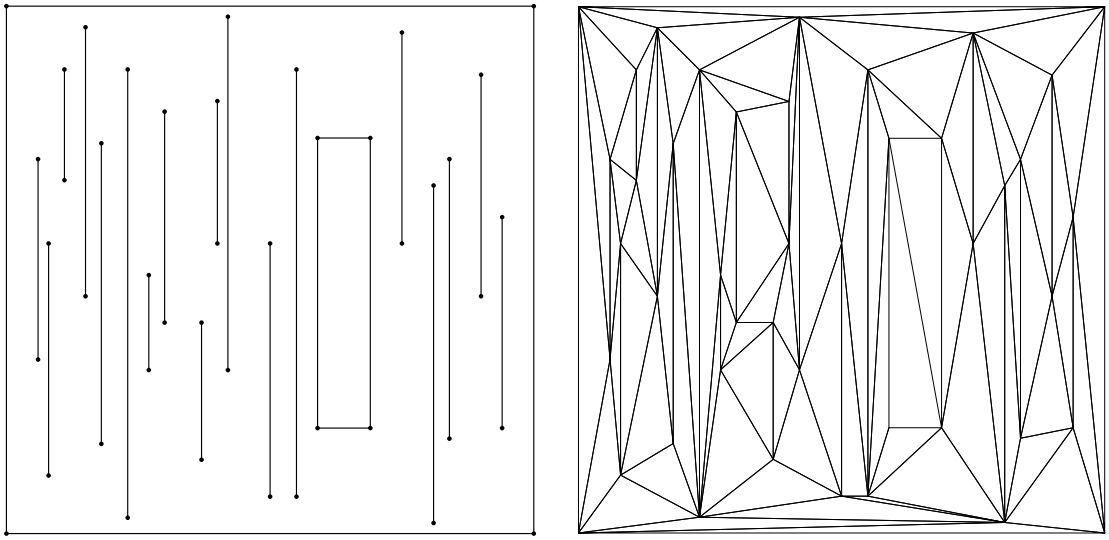

Figure 2: A planar straight line graph and its constrained Delaunay triangulation.

The input to a CDT construction algorithm is a *planar straight line graph* (PSLG), depicted in Figure 2. A PSLG $\mathcal{X}$ is a set of vertices and segments (constraining edges) that satisfies two restrictions: both endpoints of every segment in $\mathcal{X}$ are vertices in $\mathcal{X}$, and a segment in $\mathcal{X}$ may intersect other segments and vertices in $\mathcal{X}$ only at its endpoints. We seek a triangulation of the vertices in $\mathcal{X}$ that includes every segment in $\mathcal{X}$.

Throughout this article, every triangulation is understood to be a simplicial complex; thus it is a set containing vertices, edges, and triangles. A *triangulation of a PSLG* $\mathcal{X}$ is a simplicial complex $\mathcal{T} \supset \mathcal{X}$ such that $\mathcal{T}$ contains the same vertices as $\mathcal{X}$ (additional vertices are not permitted) and the union of simplices $\bigcup_{s\in\mathcal{T}} s$ is the convex hull of the vertices in $\mathcal{X}$. Note that every segment in $\mathcal{X}$ is an edge in $\mathcal{T}$.

We assume that the reader is familiar with Delaunay triangulations [14, 9]. CDTs use *visibility* to relax the Delaunay "empty circle condition." Two points p and q are *visible* to each other if the open line segment pq does not intersect a segment in $\mathcal{X}$. Consider a triangle t in some triangulation of $\mathcal{X}$; thus t's vertices are in $\mathcal{X}$ and t's interior intersects no segment in $\mathcal{X}$. A triangle t is *constrained Delaunay* if it satisfies these two conditions and t's circumcircle (circumscribing circle) encloses no vertex in $\mathcal{X}$ that is visible from a point in the interior of t, as illustrated at right in Figure 1. A *constrained Delaunay triangulation (CDT) of* $\mathcal{X}$ is a triangulation of $\mathcal{X}$ in which every triangle is constrained Delaunay, as illustrated at right in Figure 2. By the Delaunay Lemma [14, 26], a triangulation $\mathcal{T}$ of $\mathcal{X}$ is a CDT (has all its triangles constrained Delaunay) if and only if every edge in $\mathcal{T}$ that is *not* a segment in $\mathcal{X}$ is locally Delaunay.

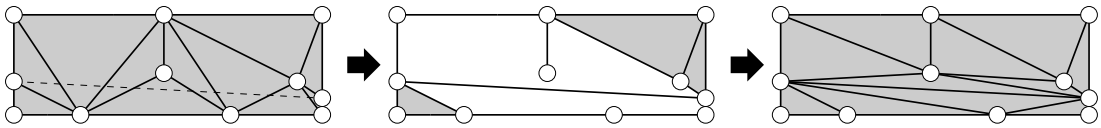

Figure 3: Inserting a segment into a constrained Delaunay triangulation.

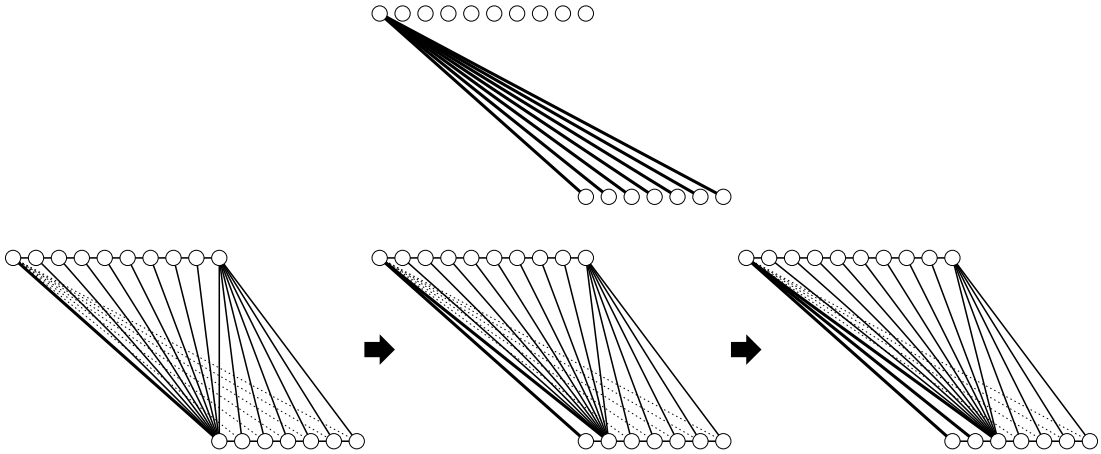

Figure 4: Top, an n-vertex PSLG for which inserting the k segments in order from left to right causes $\Theta(kn)$ structural changes. Bottom, an illustration of how each segment insertion causes the deletion and creation of $\Theta(n)$ edges and $\Theta(n)$ triangles in the CDT.

Two algorithms are known that construct the CDT of a PSLG with n vertices in $O(n \log n)$ time, which is optimal in the decision tree model of computation. One is a divide-and-conquer algorithm by Chew [10]. Its lineage stretches back to the first Delaunay triangulation algorithm to run in $O(n \log n)$ time, the 1975 divide-and-conquer algorithm of Shamos and Hoey [35], which was subsequently simplified and elaborated by Lee and Schachter [27] and Guibas and Stolfi [17]. The other is a sweepline algorithm by Seidel [33], which generalizes a Delaunay triangulation algorithm of Fortune [16] to CDTs.

To the best of our knowledge, Seidel's algorithm has not been implemented, and Chew's algorithm has been implemented only once (we do not recall by whom), perhaps because they are complicated. The only CDT construction algorithm widely used in practice begins by constructing an ordinary Delaunay triangulation first, then it inserts the segments into the triangulation one by one. To *insert a segment* is to delete all the edges and triangles that intersect its relative interior, create the new segment, and retriangulate the two polygonal cavities thus created (one on each side of the segment) with constrained Delaunay triangles, as illustrated in Figure 3. Note that the cavities might not be simple polygons, because they might have edges dangling in their interiors, as shown.

Although CDT construction by incremental segment insertion does not run in $O(n \log n)$ time, it is popular for good reasons: it takes advantage of the best existing implementations of (unconstrained) Delaunay triangulation algorithms; it is easier to implement than other CDT construction algorithms; its speed is often excellent in practice because many real-world inputs have few or no segments that cross many edges; and the ability to dynamically update a CDT by inserting a new segment is itself useful—for instance, in applications that support interactive geometric modeling [21]. Moreover, Agarwal, Arge, and Yi [1] show that if the k segments are inserted in random order, the expected number of edges and triangles deleted and created, summed over all segment insertions, is in $O(n \log^2 k)$. Compare this with the deterministic worst case of $\Theta(kn)$ for the PSLG illustrated in Figure 4.

In many implementations, each segment is inserted by a naive algorithm that takes $O(m^2)$ time, where m is the number of triangles whose interiors intersect the segment, yielding a CDT construction algorithm that takes $\Theta(kn^2)$ time for some PSLGs with n vertices and k segments. See Anglada [4] for a typical segment insertion algorithm that usually takes $\Theta(m^2)$ time, though it can achieve $\Theta(m \log m)$ best-case time when it has good luck with evenly subdividing the cavities. (Anglada's algorithm is a variant of well-known gift-wrapping algorithms; it gift-wraps triangles from the new segment out.)

Several algorithms have been proposed that compute the CDT of a simple polygon in linear time. A randomized algorithm of Klein and Lingas [24] computes the CDT in expected $O(m)$ time, and a later algorithm of Chin and Wang [12] runs in deterministic $O(m)$ time. Both algorithms rely on Chazelle's algorithm for triangulating a simple polygon in linear time [8], which is celebrated as a theoretical breakthrough but is considered too complicated for practical use. Lee and Lin [26] give a simpler $O(m \log m)$-time algorithm that is also simpler than Chew's or Seidel's algorithms, as it is specialized for simple polygons. Kao and Mount [23] give an $O(m \log m)$-time algorithm that, because it is specialized for segment insertion, is even simpler. Anglada's quadratic-time segment insertion algorithm remains the easiest to implement.

This article presents a randomized algorithm for inserting a segment into a CDT in expected $O(m)$ time. The algorithm is much easier to implement than the algorithms by Chew, Seidel, Klein and Lingas, and Chin and Wang. We provide pseudocode, which we turned into working C code in five hours.

We also show a matching $\Omega(n \log^2 k)$ lower bound on the number of structural changes during randomized incremental CDT construction, which resolves the long-standing question of the expected complexity of uniformly randomized incremental segment insertion for worst-case PSLGs. This lower bound is a surprise; we and others had not believed the upper bound was tight.

Our third contribution is to analyze a simple algorithm for *segment location*—finding one triangle deleted when a segment is inserted—and to show that it is fast enough never to be a bottleneck in CDT construction. Specifically, its running time is at worst proportional to the number of structural changes the CDT construction algorithm performs while building the CDT, even if the insertion order is not randomized.

With fast segment location and linear-time segment insertion, the randomized incremental segment insertion algorithm constructs the CDT of an n-vertex, k-segment PSLG in expected $O(n \log n + n \log^2 k)$ time. Although this running time falls short of optimality, experience with incremental CDT construction software shows that segment insertion rarely takes longer than constructing the initial Delaunay triangulation unless segments that intersect many edges are inserted by quadratic-time algorithms like Anglada's. Incremental segment insertion is likely to remain the most used CDT construction algorithm long into the future, so we think it is important to provide an understanding of its performance and how to make it run fast.

Our fourth contribution is to extend some of these results to three dimensions. Some tetrahedral CDTs can be constructed by first constructing a Delaunay triangulation, then incrementally inserting polygons into the CDT. We show that *polygon location*, like segment location in the plane, is never the bottleneck in incremental construction of a three-dimensional CDT. Although an n-vertex CDT can have $\Theta(n^2)$ tetrahedra, we show that for a class of nicely-behaved inputs that never have more than $O(n)$ tetrahedra during the construction process, the expected total number of structural changes during randomized incremental CDT construction is in $O(n \log^2 k)$, as in the plane. Unfortunately, it remains an open question whether polygons can be inserted in time proportional to the number of structural changes.

Although this article devotes most of its space to analyzing algorithms, an equally important motivation for us is to demonstrate to working programmers who write triangulation codes (but might not be researchers) that incremental segment insertion is a strong competitor among the known algorithms, and to show them how to avoid the hazard of very slow performance on the most difficult input PSLGs.

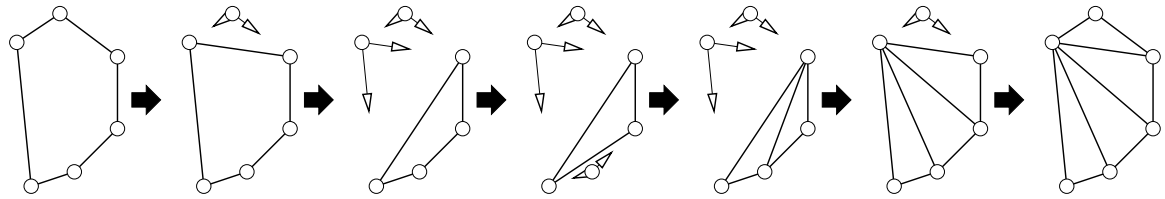

Figure 5: Chew's algorithm for computing the Delaunay triangulation of a convex polygon deletes vertices from the polygon in a random order to precompute the information needed for point location, then inserts the vertices in the opposite order.

2. Chew's Delaunay Vertex Deletion Algorithm

Our segment insertion algorithm is closely related to an algorithm of Paul Chew [11] for deleting a vertex from a Delaunay triangulation in expected $O(m)$ time, where m is the degree of the deleted vertex. The latter algorithm is a good preparation for understanding the former, more complicated algorithm. Our algorithm also uses Chew's as a subroutine.

Vertex deletion is an operation that updates a Delaunay triangulation so it has one less vertex and is still Delaunay. Chew's algorithm can delete vertices from CDTs as well.

For simplicity, consider the (seemingly easier) problem of constructing the Delaunay triangulation of a convex polygon. Chew's algorithm is a randomized incremental insertion algorithm that inserts one vertex at a time into the Delaunay triangulation. Let V be a sequence listing the m vertices of a convex polygon in counterclockwise order. The algorithm begins by generating a random permutation of V that dictates the order in which the vertices will be inserted. It constructs a triangle from the first three vertices of the permutation, then inserts the remaining vertices one by one.

Just before a vertex u is inserted, it lies outside the growing triangulation, but only one triangulation edge vw separates u from the triangulation's interior. *Point location* is the task of identifying the edge vw. Next, the algorithm inserts u by first identifying and deleting all the triangles whose circumcircles enclose u. These can be found quickly by a depth-first search from the triangle adjoining vw. Then, by extending new edges from u, the algorithm retriangulates the cavity formed by taking the union of the deleted triangles and $\triangle uvw$, as illustrated in the right half of Figure 5. This is essentially the *Bowyer–Watson algorithm* [5, 19, 20, 45] for inserting a vertex into a Delaunay triangulation.

The cleverest aspect of Chew's algorithm is how it performs point location. It does all point location in advance, before constructing any triangles, by imagining the incremental insertion algorithm running backward in time. Specifically, imagine taking the input polygon and removing vertices one by one, reversing the random permutation of V, yielding a shrinking sequence of convex polygons as illustrated in the left half of Figure 5. Removing a vertex u has the effect of joining its neighbors v and w with an edge vw, which is the edge that later will be sought for point location.

The algorithm maintains a circularly-, doubly-linked list of vertices representing the polygon. It walks through a random permutation of V in backward order, removing vertices from the circularly-linked list until only three remain. The algorithm constructs a triangle from the three surviving vertices, then inserts the other vertices in the permutation of V in forward order.

The same algorithm, with no changes, can also retriangulate the cavity evacuated when a vertex is deleted from a Delaunay triangulation, even though the cavity might not be convex. We will not justify that claim here, except to point out that Chew's algorithm is a disguised algorithm for deleting a vertex from a three-dimensional convex hull [25], which is related by the lifting map [6, 32, 15] to deleting a vertex from a two-dimensional Delaunay triangulation. On the lifting map, the Delaunay triangles adjoining a vertex lie on the boundary of a convex polyhedral cone; Chew's algorithm exploits this convexity.

Theorem 1. *Given an m-vertex polygon, Chew's algorithm runs in expected $O(m)$ time.*

Proof. The point location stage runs in deterministic $O(m)$ time. Chew derives the expected running time of the vertex insertion stage by *backward analysis*, an analysis technique that Seidel [34] summarizes thus: "*Analyze an algorithm as if it was running backwards in time, from output to input.*"

To simplify the analysis, we assume that every vertex set has a unique Delaunay triangulation, independent of the order in which the vertices are inserted. This assumption might not hold if there are four cocircular vertices, but there is an easy way to fix the analysis [9, Section 3.5], which we omit.

Every triangulation of an m-vertex polygon has $2m-3$ edges, each with two endpoints. Imagine deleting a vertex chosen uniformly at random; in expectation it adjoins $4 - 6/m$ edges.

With the algorithm running forward in time, the cost of inserting the last vertex is proportional to the number of edges that adjoin it after it is inserted. The expected number of those edges is less than four. The same reasoning holds for the other vertices. Summing this cost over all the vertices yields an expected linear running time. □

3. Inserting a Segment into a CDT

To "insert a segment into a CDT" is to take as input a CDT of a PSLG $\mathcal{X}$ and a new segment s to insert, and produce a CDT of $\mathcal{X} \cup \{s\}$. It is meaningful only if $\mathcal{X} \cup \{s\}$ is a valid PSLG—that is, $\mathcal{X}$ already contains the endpoints of s (otherwise, they must be inserted first), and the relative interior of s intersects no segment or vertex in $\mathcal{X}$. This section presents a segment insertion algorithm similar to Chew's algorithm. Its expected running time is linear in the number of edges the segment crosses.

Let $\mathcal{T}$ be a CDT of $\mathcal{X}$. If $s \in \mathcal{T}$, then $\mathcal{T}$ is also a CDT of $\mathcal{X} \cup \{s\}$. Otherwise, the algorithm begins by performing *segment location*: identifying a triangle in $\mathcal{T}$ whose interior intersects s. This can be done with a simple rotary traversal of the triangles adjoining the endpoint of s with lesser degree. In Section 8, we show that this method never increases the asymptotic running time of CDT construction.

Once one triangle whose interior intersects s is found, the others can be identified by a simple walk in time linear in their number. The algorithm deletes these triangles from $\mathcal{T}$. All the other triangles in $\mathcal{T}$ remain constrained Delaunay after s is inserted. Next, the algorithm adds s to the triangulation and retriangulates the two polygonal cavities on each side of s with constrained Delaunay triangles, as illustrated in Figure 3.

Let P and $\hat{P}$ be the two polygonal cavities; their edges include s. The randomized incremental insertion algorithm CavityCDT in Figures 6 and 7 retriangulates P, and a second call to CavityCDT retriangulates $\hat{P}$. Be forewarned that CavityCDT *cannot* compute the CDT of an arbitrary polygon; it depends upon the special nature of the cavities evacuated by segment insertion for its correctness.

CavityCDT differs from Chew's algorithm in several ways to account for the fact that P is not always convex. First, the vertices of the segment s are inserted first. Second, P might have edges dangling in its interior, like the segment connecting vertices 5 and 6 in Figure 8. In this case, imagine an ant walking a counterclockwise circuit of P's interior without crossing any edges; it will visit one or more vertices of P twice. Split each such vertex into two copies and pretend they are two separate vertices, like vertices 5 and 7 in the figure. (In rare circumstances, there may be three or more copies of a vertex.)

Third, CavityCDT maintains the invariant that after each vertex insertion, the computed triangulation is the CDT of the polygon whose boundary is specified by the subsequence of vertices inserted so far; we call this polygon a *subpolygon*. Because CavityCDT maintains a CDT and not merely a Delaunay triangulation, a newly inserted vertex sometimes causes a triangle to be deleted not because the new vertex lies inside the triangle's circumcircle, but because the two polygon edges adjoining the new vertex cut through the

```
CavityCDT(V)
{ V = ⟨v_0, v_1, ..., v_{m-1}⟩ is a sequence of vertices in counterclockwise order around }
{ a cavity evacuated when the segment v_0 v_{m-1} is inserted. Some vertices in the sequence }
{ are duplicated if there are dangling edges in the cavity. }
1     for i ← 1 to m − 2
2         next[i] ← i + 1      { The arrays next, prev: [1, m − 2] → [0, m − 1] represent }
3         prev[i] ← i − 1      {       a doubly-linked list of vertex indices. }
          { distance[i] is proportional to the distance from v_i to the line ←→ v_0 v_{m-1}. }
4         distance[i] ← Orient(v_0, v_i, v_{m-1})
          { π[1 ... m − 2] will always be a permutation of 1 ... m − 2. }
5         π[i] ← i
6     distance[0] ← 0; distance[m − 1] ← 0
      { Delete the vertices from the polygon in a random order. }
7     for i ← m − 2 downto 2
          { Select a vertex to delete that is not closer to ←→ v_0 v_{m-1} than both its neighbors. }
8         repeat
9             j ← a random integer in [1, i]
10        while distance[π[j]] < distance[prev[π[j]]] and distance[π[j]] < distance[next[π[j]]]
          { Point location: take the vertex v_{π[j]} out of the doubly-linked list. }
11        next[prev[π[j]]] ← next[π[j]]
12        prev[next[π[j]]] ← prev[π[j]]
          { Move the deleted vertex index π[j] to follow the live vertices. }
13        Swap π[i] with π[j]
14    CreateTriangle(v_0, v_{π[1]}, v_{m-1})      { Create the first triangle. }
15    for i ← 2 to m − 2
16        InsertVertex(v_{π[i]}, v_{next[π[i]]}, v_{prev[π[i]]})
17        if v_{π[i]} has been marked
18            Call Chew's algorithm to retriangulate the fan of triangles that have all 3 vertices marked
19            Unmark all the marked vertices
```

Figure 6: Expected linear-time algorithm for retriangulating a cavity evacuated by inserting a segment into a constrained Delaunay triangulation. The subroutines InsertVertex and Orient appear in Figure 7.

triangle. For example, the insertion of vertex 8 in Figure 8 deletes a triangle whose circumcircle does not enclose vertex 8. Line 21 of InsertVertex (Figure 7) accounts for this possibility with an orientation test.

Fourth, unlike in Chew's algorithm, the insertion of a vertex u can create new triangles that do not adjoin u, as illustrated in Figure 9. The three shaded triangles in the top triangulation must be deleted when u is inserted, but u is not inside their circumcircles. We call triangles with this property *crossed triangles*. After u's insertion, the corresponding new triangles (shaded in the bottom triangulation) may include some that do not adjoin u.

CavityCDT inserts u in a manner that initially connects u to all the vertices of all the deleted triangles. If there are crossed triangles, CavityCDT subsequently uses Chew's original algorithm to correctly retriangulate the shaded region (in expected linear time). We observe few crossed triangles in practice, so the overhead of occasionally invoking Chew's algorithm is unlikely to have much influence on the running time. (An alternative, easier to implement, is for CavityCDT to call itself recursively with uw serving as the

InsertVertex(u, v, w)
{ u is a new vertex we are inserting. Is the triangle $\triangle uvw$ constrained Delaunay? }
20 $x \leftarrow$ Adjacent(w, v) { Find $\triangle wvx$ on the other side of the edge vw from u. }
{ The edge vw survives if $\triangle wvx$ does not exist **or**
(u is not inside the circumcircle of $\triangle wvx$ **and** u is on the correct side of the edge vw). }
21 **if** $x = \emptyset$ **or** (InCircle(w, v, x, u) ≤ 0 **and** Orient(u, v, w) > 0)
22 CreateTriangle(u, v, w) { $\triangle uvw$ is constrained Delaunay. }
23 **else** { $\triangle uvw$ and $\triangle wvx$ are not constrained Delaunay. }
24 DeleteTriangle(w, v, x) { Flip edge $vw \rightarrow$ edge ux. }
25 InsertVertex(u, v, x)
26 InsertVertex(u, x, w)
27 **if** InCircle(w, v, x, u) ≤ 0 { For the sake of speed, reuse the computation from Line 21. }
{ $\triangle wvx$ is a crossed triangle. }
28 Mark vertices u, v, w, and x to be retriangulated later (in Line 18)

Orient(u, v, w)
{ Returns a positive value if u, v, and w occur in counterclockwise order. }

$$\textbf{return } \det \begin{bmatrix} u_1 - w_1 & u_2 - w_2 \\ v_1 - w_1 & v_2 - w_2 \end{bmatrix}.$$

InCircle(u, v, w, x)
{ Positive if x is strictly inside the circle passing through the positively oriented vertices u, v, and w. }

$$\textbf{return } \det \begin{bmatrix} u_1 - x_1 & u_2 - x_2 & |u - x|^2 \\ v_1 - x_1 & v_2 - x_2 & |v - x|^2 \\ w_1 - x_1 & w_2 - x_2 & |w - x|^2 \end{bmatrix}.$$

Figure 7: Expected linear-time algorithm for retriangulating a cavity evacuated by inserting a segment into a constrained Delaunay triangulation (continued from Figure 6). A triangulation data structure permits Adjacent(u, v) to look up the third vertex w of a triangle $\triangle uvw$ in $O(1)$ expected time, with the convention that u, v, and w are positively oriented (occur in counterclockwise order); it returns $w = \emptyset$ if there is no such triangle. The operations CreateTriangle and DeleteTriangle add and remove positively oriented triangles in expected $O(1)$ time. These running times can be achieved with a hash table that maps pairs of vertices to triangles; our implementation achieves them with a tree data structure.

segment, but we do not know whether this option preserves the expected linear running time.)

Fifth, a subpolygon might be *self-intersecting*. Observe in Figure 8 that deleting vertex 2 from the cavity P creates a subpolygon G in which the edge connecting vertices 1 and 3 crosses the edge connecting vertices 4 and 5, and the subpolygon's interior angle at vertex 3 exceeds 360°. By some definitions, G is not actually a polygon, although it is a polygon in the conventional sense of a looped chain of edges; and its triangulation in Figure 8 (bottom center) is not a simplicial complex, because it has triangles that overlap each other. Fortunately, it is like a CDT in two respects: it has all the combinatorial properties of a triangulation of a polygon—for example, its dual graph is a tree—and every edge is locally Delaunay.

The incremental vertex insertion algorithm works correctly even when these self-intersecting subpolygons arise, subject to one caveat: it will not correctly insert a vertex at which the polygon's internal angle is 360° or greater. For example, it cannot triangulate P in Figure 8 by inserting vertex 6 last, nor triangulate G by inserting vertex 3 last. These vertex insertions are anticipated and averted during the algorithm's point

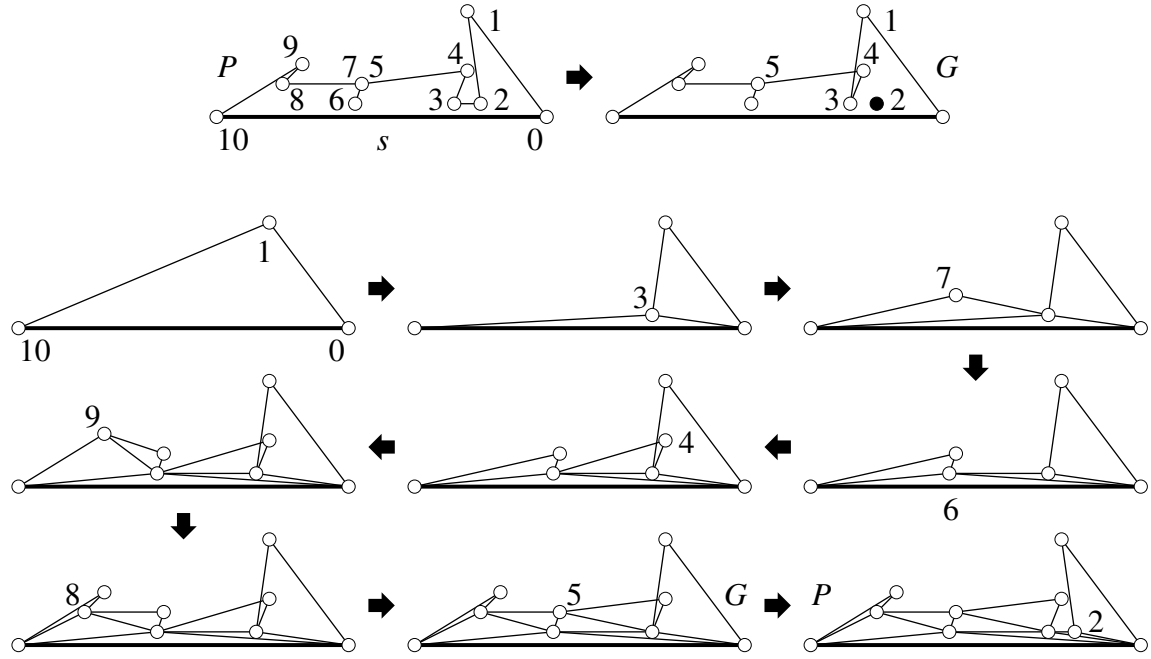

Figure 8: Computing the constrained Delaunay triangulation of a cavity obtained by inserting a segment s. The cavity has a repeated vertex, numbered 5 and 7, because of the dangling edge adjoining it. The deletion of vertex 2 creates a self-intersection, but the algorithm works correctly anyway.

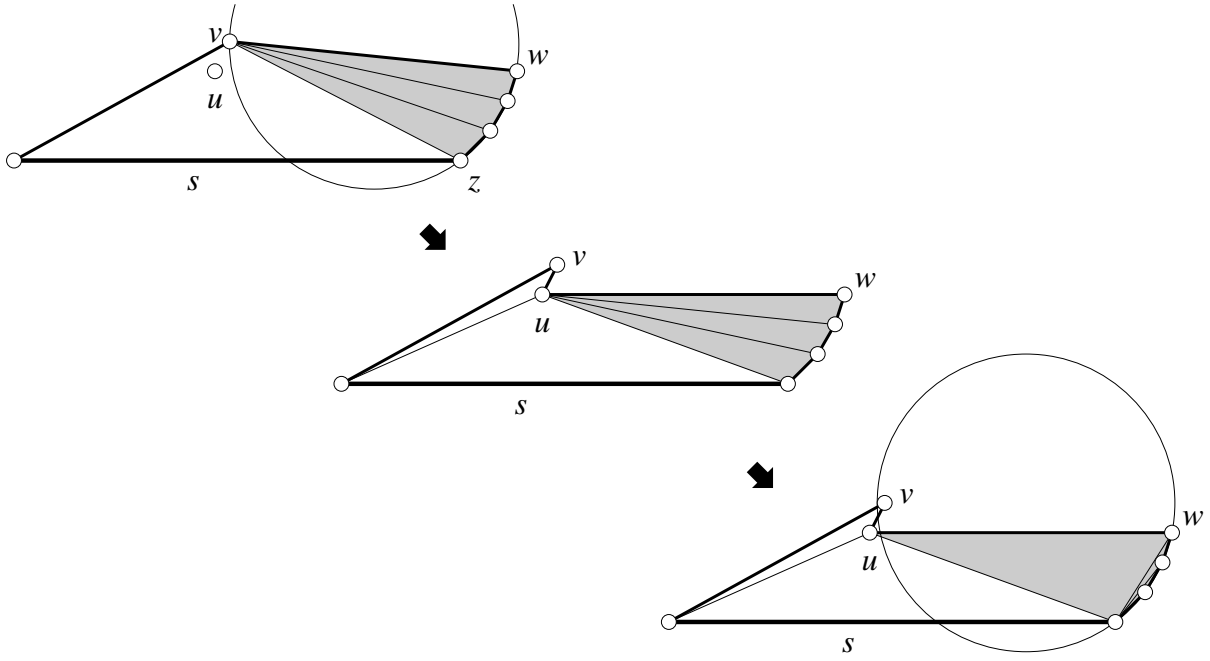

Figure 9: The shaded triangles in the top triangulation (called *crossed triangles*) are deleted by u's insertion despite u not being inside their circumcircles because the edge uw crosses them. InsertVertex produces the triangles in the middle triangulation, but the shaded triangles are not necessarily constrained Delaunay. We use Chew's algorithm to replace them with the shaded triangles in the bottom triangulation, which include two that do not adjoin u.

location step, when random vertices are deleted from P one by one until the subpolygon is reduced to a triangle. Hence, the random permutation by which the vertices are inserted is not chosen uniformly from all permutations of the vertices.

For the sake of speed, CavityCDT does not compute internal angles. Instead, let $\bar{s}$ be the line that includes the segment s. It is a property of the cavities created by segment insertion that a subpolygon vertex can have an internal angle of 360° or greater only if that vertex is closer to $\bar{s}$ than both its neighbors on the subpolygon chain. (We will justify this claim shortly.) CavityCDT declines to delete from P any vertex with the latter property (see Line 10). Later we will consider a second reason for this restriction (see Lemma 4).

Line 4 of CavityCDT computes the distance of each vertex of P from $\bar{s}$. The point location step (Lines 7–13) deletes vertices from P one by one, choosing uniformly at random from all the vertices that are not endpoints of s and are not closer to $\bar{s}$ than both their neighbors.

Be forewarned that CavityCDT cannot use the same triangulation data structure as the triangulation in which the segment s is being inserted, because CavityCDT sometimes temporarily creates triangles that conflict with those outside the cavity. For example, in Figure 8 the triangulation outside the cavity probably includes an edge connecting vertices 7 and 9, which is an edge of two triangles. CavityCDT temporarily creates a third triangle with this edge when it first inserts vertex 9. To avoid corrupting the data structure, CavityCDT requires the use of a separate, initially empty triangulation data structure. The final triangles must subsequently be copied to the main triangulation.

4. The Speed and Correctness of CavityCDT

To analyze CavityCDT, we must answer several questions: how do we understand a self-intersecting polygon? What does it mean for it to have a triangulation or a CDT? Does its CDT obey the same rules as a traditional CDT, such as the Delaunay Lemma? We will answer these questions by defining subpolygons to be topological spaces we call *multisheet polygons*, which are constructed by gluing ordinary polygon-shaped topological spaces together along their edges.

Before discussing those foundations, we take a brief detour to analyze the running time of CavityCDT. The following proof is not yet complete, because it relies on a fact about CDTs of subpolygons (the forthcoming Lemma 3) whose proof we must delay until we develop an understanding of multisheet polygons. The proof also relies implicitly on the fact that CavityCDT and InsertVertex work correctly. But the core argument is simple, so we put it up front.

Theorem 2. *Given an m-vertex cavity,* CavityCDT *runs in expected $O(m)$ time.*

Proof. The expected cost of InsertVertex is proportional to the number of edges adjoining the newly inserted vertex u plus the number of newly created triangles that do not adjoin u. We bound these numbers separately, both by backward analysis.

Every triangulation of an m-vertex polygon has $m-2$ triangles and $m-3$ interior edges. At least $(m-1)/2$ vertices are eligible to be the first vertex deleted during point location and the last vertex inserted. One of those vertices is chosen uniformly at random; in expectation it adjoins at most $2(m-3)/((m-1)/2) < 4$ interior edges, thus fewer than 6 edges total.

When the insertion of a new vertex u causes the creation of a triangle t that does not adjoin u, as illustrated in Figure 9, it happens because a new subpolygon edge adjoining u entirely crosses t's circumcircle and hides one or more vertices inside t's circumcircle. Consider CavityCDT running backward in time; what is the probability that a triangle t will be deleted during the deletion of a randomly chosen vertex u that is not a vertex of t? The forthcoming Lemma 3 states that there at most two vertices, besides the vertices of t, whose deletion could expose t to a vertex, thereby deleting t. The probability that u is one of those two

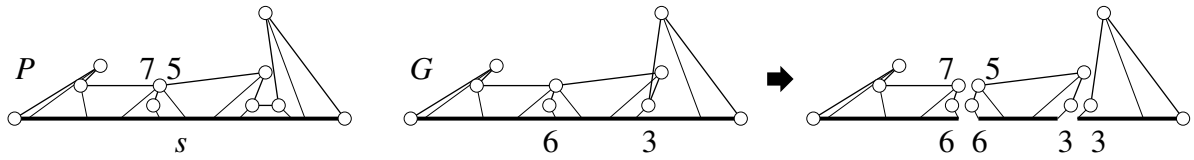

Figure 10: At left, each vertex has a sightline. Note that vertices 5 and 7 have different sightlines, although they are really a single repeated vertex. A self-intersecting polygon (center) can be interpreted topologically as several polygons glued together along their sightlines (right).

vertices is at most $4/(m-1)$. Summing this probability over the $m-2$ triangles, we find that the expected number of deleted triangles not adjoining u is less than 4.

Thus, forward in time, the expected cost of inserting each vertex is constant. Summing this cost over all the vertices yields an expected linear running time. □

Before proving the correctness of the algorithm CavityCDT, we must make sense of self-intersecting polygons and their triangulations. The original cavity P has a special property that enables CavityCDT to work despite the possibility of self-intersecting subpolygons: P is included in a union of triangles that cross the segment s. Therefore, each vertex v_i has a *sightline* b_i: a line segment connecting v_i to s strictly through P's interior, depicted in Figure 10. We choose each sightline to be a subset of an edge that was deleted to make way for s; these sightlines do not intersect each other.

A self-intersecting subpolygon G can be understood as a topological space defined by gluing ordinary quadrilaterals and triangles together along the sightlines, as illustrated. We call these constructions *multisheet polygons* because they can be assembled from multiple sheets of paper taped together. In such a multisheet polygon, two points with the same Euclidean coordinates are not necessarily the same point; they may be in different, overlapping quadrilaterals or triangles. We triangulate a multisheet polygon by subdividing it into triangles that form the topological space G when they are glued together along shared edges. We call this triangulation a CDT if all its edges are locally Delaunay.

One consequence of G's sightlines is that for any vertex v of G at which the internal angle is 360° or greater (e.g., vertex 3 or 6 in Figure 10), both v's neighbors on G's boundary chain are further from $\bar{s}$ (the affine hull of s) than v is—each because the other neighbor has a valid sightline. This observation justifies the algorithm's use of the latter property to screen out vertices with the former property in Lines 8–10.

Our correctness proof for CavityCDT relies on a constrained version of the famous *Delaunay Lemma*. In 1934, Boris Delaunay [14] showed that a triangulation of a point set is Delaunay (has every triangle Delaunay) if and only if every edge is locally Delaunay. In 1986, Lee and Lin [26] showed that a triangulation of a PSLG is constrained Delaunay (has every triangle constrained Delaunay) if and only if every edge is locally Delaunay except perhaps the PSLG segments.

This Constrained Delaunay Lemma extends to a multisheet polygon G as follows. Two points $p, q \in G$ are said to be *visible* to each other if there is a path from p to q in G that is a straight line segment. Observe that two distinct points in G can lie at the same position in Euclidean space yet not be visible to each other, because the shortest path connecting them in G goes around a corner. With this definition of "visible," the definition of "constrained Delaunay triangle" in the Introduction applies to triangulations of multisheet polygons. It is easy to extend the proof of Lee and Lin to show that a triangulation of a multisheet polygon has every triangle constrained Delaunay if and only if every edge is locally Delaunay.

We are ready to complete the proof of Theorem 2.

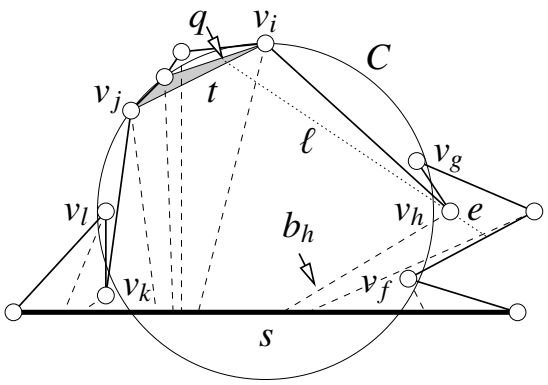

Figure 11: A subpolygon G' in which the deletion of a vertex (not shown) has exposed a vertex v_f inside the circumcircle C of a triangle t, so t is no longer constrained Delaunay. The dashed lines are sightlines. The configuration depicted here is impossible because v_g does not have a sightline.

Lemma 3. *Consider a subpolygon G whose CDT contains a triangle t. Imagine* CavityCDT *running backward in time; suppose a vertex of G is deleted. Besides t's vertices, there are at most two other vertices of G whose deletion would delete t.*

Proof. Let C be t's circumcircle. As t is constrained Delaunay (with respect to G), no vertex of G is both strictly inside C and visible from the interior of t. Suppose that the deletion of some vertex u of G causes t to be deleted, but u is not a vertex of t, as in Figure 9. Then G has a vertex (such as v in Figure 9) that is strictly inside C and becomes visible from t when u is deleted.

Let v_i and v_j be the vertices of t with the least index i and the greatest index j. The edge v_iv_j of t cuts G into two multisheet polygons, one having vertices numbered between i and j and one having the remaining vertices and the segment s, as illustrated in Figure 11. Each vertex's sightline can cross v_iv_j at most once: each vertex of G in the range $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ is visible from the interior of t because its sightline passes through t, but no other vertex's sightline intersects t's interior. As t is constrained Delaunay, no vertex of G in the range $v_i, v_{i+1}, \ldots, v_j$ is strictly inside C.

Let v_g be the vertex of G with the greatest index $g < i$ such that v_g is strictly inside C, if such a vertex exists, as illustrated in Figure 11. Let q be an arbitrary point in the interior of t. As t is constrained Delaunay, v_g is not visible from q. As G is simply connected, there is a unique shortest path from q to v_g through G; as G is a multisheet polygon that does not include a straight path from q to v_g, the shortest path is composed of two or more line segments. Let v_h be the vertex where this path first turns (hence v_h is visible from q). Symmetrically, let v_l be the vertex of G with the least index $l > j$ such that v_l is strictly inside C, if such a vertex exists, and let v_k the the first vertex on the shortest path from q to v_l, as illustrated. Observe that $g < h \leq i < j \leq k < l$.

As v_h is visible from q, v_h is not strictly inside C. Let ℓ be the line through q and v_h, and consider a line segment $e \subset \ell$ that extends from v_h directly away from q (extending the first segment of the shortest path) until it cuts G into two simply connected multisheet polygons, as illustrated in Figure 11. The polygon on one side of e has the vertex v_g, and the polygon on the other side includes the triangle t. Observe that no part of e is inside C (because q is inside C and v_h is not).

Consider any vertex v_f of G that is strictly inside C. We claim that v_f can become visible from the interior of t only if v_h or v_k is deleted; this claim establishes the lemma. Suppose for the sake of contradiction that there exists a subpolygon G' of G that retains the vertices v_f, v_h, and v_k and the vertices of t, but within G', v_f is visible from some point q' in t's interior—that is, the multisheet polygon G' includes the line

segment $v_f q'$. By our definitions of v_g and v_l, either $f \leq g$ or $f \geq l$; suppose without loss of generality that $f \leq g$. (The symmetric case, in which $f \geq l$, can be treated with the same reasoning in mirror image.)

As $f < h \leq i$, v_h's sightline b_h separates v_f from t in G'. Thus the line segment $v_f q'$ crosses b_h. We claim that $v_f q$ also crosses b_h. To see this, observe that b_h must cross the entire triangle $\triangle qq'v_f$, because $\triangle qq'v_f$ is strictly inside C, v_h is not inside C, and the other endpoint of b_h lies on the segment s and so cannot be in the interior of $\triangle qq'v_f$. But b_h cannot intersect qq', because b_h does not intersect t's interior. Therefore, b_h crosses the other two edges of $\triangle qq'v_f$. It follows that $v_f q$ crosses b_h as claimed, so v_f is on the same side of ℓ as b_h is. This implies that the line segment e separates the portion of G containing v_g from the portion containing v_f (and thus $f \neq g$).

These relationships impose irreconcilable constraints on v_g's sightline. The sightline for v_g terminates on the segment s between the sightlines for v_f and v_h, and the sightlines do not intersect each other. Hence v_g's sightline starts at v_g, which is inside C; then it crosses e, which is outside C; and then it crosses $v_f q$, which is inside C. But a sightline is straight; it cannot exit and reenter C. The lemma follows by contradiction. □

Next, we justify the use of Chew's algorithm to fix part of the triangulation (Line 18 of CavityCDT) when the insertion of a vertex u causes the deletion of crossed triangles whose circumcircles do not enclose u. We will show (see Lemma 4 below) that these crossed triangles always form a fan adjoining a single vertex v, shaded at left in Figure 9. The polygon that we retriangulate with Chew's algorithm (shaded at right in Figure 9) is not necessarily convex, so we must justify why Chew's algorithm will always succeed. Recall that Chew's algorithm is known to work correctly for retriangulating a cavity created when a vertex is deleted from a Delaunay triangulation (and more generally, for deleting a vertex from a three-dimensional convex hull [25]). The triangles in the fan adjoining v are separated by locally Delaunay edges. With reference to the figure, if we add the triangle $\triangle vuz$ to the fan, the edge vz is locally Delaunay as well, because the fan triangles are crossing triangles whose circumcircles do not enclose u. The polygon we are retriangulating is the polygon obtained by deleting v. (On the lifting map, v is lifted to the apex of a convex cone in three dimensions. The fact that the lifted vertices of the fan are in convex position from v's point of view is the precondition for Chew's algorithm to work correctly.) Therefore, Chew's algorithm correctly retriangulates the fan.

To see that the crossed triangles always form this fan configuration, suppose without loss of generality that s is horizontal, as in Figure 12, with the cavity above s and the vertex indices increasing from 0 at the right endpoint of s to $m - 1$ at the left endpoint. Let ℓ be the horizontal line (parallel to s) through u. By design, when u is inserted, its two neighbors (v and w) on the subpolygon G cannot both be above ℓ. Suppose without loss of generality that u's neighbor w, whose index is less than u's index, is on or below ℓ. Because there is a crossed triangle, u is a concave vertex of G, so u's other neighbor v (with greater index) must be strictly above ℓ.

Lemma 4. *Given the suppositions stated above, the crossed triangles form a fan of triangles sharing vertex v, and their other vertices have indices less than u's index.*

Proof. Let t be a crossed triangle. It has at least one vertex x whose index is greater than u's and one vertex y whose index is less. (If every vertex of t had a greater index, t would be protected from u by v's sightline; if every vertex of t had a lesser index, t would be protected from u by w's sightline.) Suppose for the sake of contradiction that $x \neq v$, as Figure 12 shows. Either $y = w$ or the triangle edge xy crosses the subpolygon edge uw, so y lies on or below ℓ. As xy also crosses the subpolygon edge uv, x lies strictly above ℓ.

Before s was inserted, v was connected by an edge of the CDT to a vertex v' on the other side of s, so there existed a circle C through v and v' that enclosed no vertex visible from the relative interior of the edge

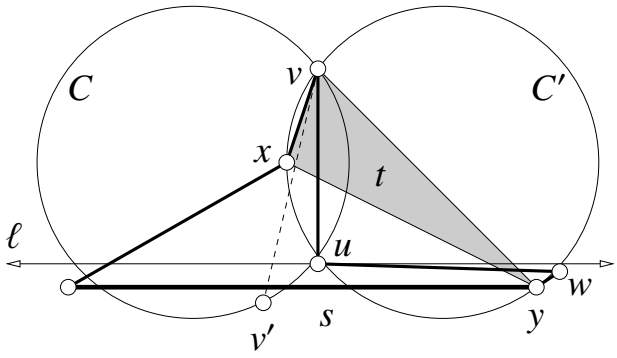

Figure 12: An impossible configuration: upon the insertion of u, a crossed triangle t has a vertex x above ℓ that is not u's neighbor on the subpolygon chain.

vv'. The portion of vv' terminating at s is a sightline for v in the original cavity P and in every subpolygon of P. Both u and x are visible (within P) from the point $vv' \cap s$, so neither u nor x is inside C. There is a point in the interior of t from which v is visible, because v's sightline intersects t's interior. As t is constrained Delaunay (with respect to the polygon just before u is inserted), v is not inside t's circumcircle C'. As t is a crossed triangle, u is not inside C' either.

The vertices v and x are above ℓ, and v' and y are below or on ℓ, so both circles C and C' (circumscribing vv' and xy) intersect ℓ, as illustrated. Neither circle encloses u; C intersects ℓ to the left of u, and C' intersects ℓ to the right of u (both possibly touching u). The edges vv' and xy cross each other above ℓ, so v' is outside C' and y is outside C. It is impossible to simultaneously satisfy the constraints that vv' and xy cross each other, a circle C through v and v' encloses neither x nor y, and a circle $C' \neq C$ through x and y encloses neither v nor v'. The result follows by contradiction. □

Our correctness proof uses the constrained Delaunay property of the triangles before each vertex insertion as a precondition to guarantee the locally Delaunay property of the edges after each vertex insertion. By the Constrained Delaunay Lemma, the latter property implies that the former holds at the beginning of the next vertex insertion.

Theorem 5. CavityCDT *correctly constructs the CDT of a cavity evacuated by the insertion of a segment into a CDT.*

Proof. Let $\mathcal{T}'$ and $\mathcal{T}$ be the triangulation before and after a top-level call to the recursive procedure InsertVertex inserts a new vertex u and Line 18 of CavityCDT fixes the marked triangles (if necessary). Let G' and G be the corresponding subpolygons without and with u. We will show that if $\mathcal{T}'$ is a CDT of G', then $\mathcal{T}$ is a CDT of G. The result follows by induction on the sequence of vertex insertions.

It is straightforward to check that a top-level call to InsertVertex is equivalent to gluing a new triangle $\triangle uvw$ onto an edge vw of $\mathcal{T}'$, then repeatedly checking each edge that is opposite the new vertex u in some triangle (initially the edge vw, later other edges that are exposed by flips) and flipping any such edge that is not locally Delaunay or forms a *fold* in the triangulation (Line 21 of InsertVertex). Every edge created by a flip has the vertex u, and every vertex of every triangle deleted by a flip gets connected to u.

Each InsertVertex call maintains the invariant that the set of stored triangles forms a combinatorial triangulation of a polygon—specifically, the dual graph of the triangulation is a tree whose leaves correspond to the edges of the polygon in circular order. Gluing on a new triangle is equivalent to replacing a leaf of the dual tree with a new degree-three node and two new leaves. Edge flips correspond to tree rotations.

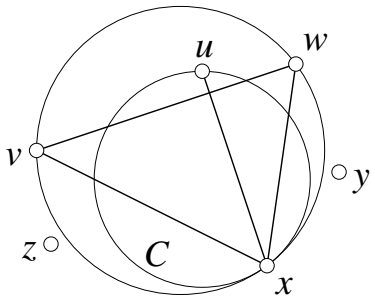

Figure 13: As $\triangle wvx$ was constrained Delaunay before u was inserted, ux is locally Delaunay.

If gluing $\triangle uvw$ onto $\mathcal{T}'$ does not create a fold at vw, then the modified triangulation is immediately a triangulation of G. Otherwise, pretend that the algorithm flips only fold edges until none survive, then flips edges that are not locally Delaunay. (CavityCDT actually flips the edges in a different order, but this does not change the final product.) Observe that u is safely confined between the sightlines of v and w; no other vertex lies between these sightlines, so no vertex can lie inside the inverted triangle $\triangle uvw$. It is straightforward to see that there is only one fold edge at any given time and the flips of the fold edges collectively remove $\triangle uvw$ from G', yielding G. Subsequent flips do not change the underlying topological space of the triangulation, so it remains a triangulation of G.

To show that the updated triangulation $\mathcal{T}$ is constrained Delaunay, we show that all its edges are locally Delaunay. Consider an edge ux created because the InCircle test in Line 21 found that the new vertex u is inside the circumcircle of a triangle $\triangle wvx \in \mathcal{T}'$; we wish to show that ux is locally Delaunay in $\mathcal{T}$. Let $\triangle uxy$ and $\triangle xuz$ be the triangles in $\mathcal{T}$ of which ux is an edge. The vertices y and z are visible (within G') from the interior of $\triangle wvx$, because they are visible (within G) from every point on ux, which passes through the interior of $\triangle wvx$ as Figure 13 shows. As $\mathcal{T}'$ is a CDT, $\triangle wvx$ is constrained Delaunay (prior to the insertion of u), so neither y nor z is inside the circumcircle of $\triangle wvx$. Let C be the circle that passes through u and x and is tangent to the circumcircle of $\triangle wvx$ at x, as illustrated. As the circumcircle encloses C, neither y nor z is inside C, so ux is locally Delaunay in $\mathcal{T}$ as claimed.

Let us categorize the edges of $\mathcal{T}$ and show they are all locally Delaunay. By the foregoing argument, every edge adjoining u because of the Incircle test in Line 21 is locally Delaunay. Every edge that adjoins the same two triangles in $\mathcal{T}$ it adjoined in $\mathcal{T}'$ is locally Delaunay because it was locally Delaunay in $\mathcal{T}'$. Every edge that adjoins one old triangle surviving from $\mathcal{T}'$ and one triangle new to $\mathcal{T}$ is locally Delaunay either because the Incircle test in Line 21 chose not to flip it or because the new triangle replaces a crossing triangle after a vertex stopped being visible. Every edge on the boundary of G is locally Delaunay because it is an edge of only one triangle. The only edges that fit none of these categories are the new edges created by Line 18 of CavityCDT, which are locally Delaunay by the correctness of Chew's algorithm. Therefore, every edge in $\mathcal{T}$ is locally Delaunay, and by the Constrained Delaunay Lemma, $\mathcal{T}$ is a CDT. □

5. A Comparison of Two Cavity Retriangulation Implementations

We have implemented CavityCDT and Anglada's gift-wrapping algorithm [4] so that we could compare their speeds on cavities of different sizes. Both implementations use Shewchuk's floating-point Orient and InCircle predicates [36], which are fast despite being robust. We timed two kinds of cavity, illustrated in Figure 14: cavities in which all the vertices are collinear except the segment endpoints, and perturbed versions of those cavities in which the vertices are jittered by random vertical movements proportional

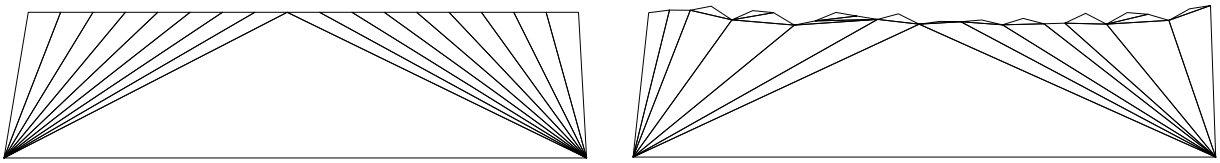

Figure 14: Constrained Delaunay triangulations of our input cavities, collinear or jittered.

		Anglada		CavityCDT		
	vertices	time (μs)	InCircle tests	time (μs)	InCircle tests	Orient tests (Line 21 only)
Cavities in	10	0.41	16	1.29	13.02	7.78
which most	30	4.20	196	4.20	67.07	43.01
vertices are	100	47.27	2,401	14.14	271.00	178.14
collinear	1,000	4,748.62	249,001	152.65	2,959.48	1,968.93
	10	0.57	14.41	1.53	11.03	7.17
Jittered	85	16.26	620.04	16.26	161.39	106.04
cavities	100	20.68	804.34	19.10	191.48	125.84
	1,000	719.67	32,184.86	180.63	1,920.00	1,268.00

Table 1: Timings for CavityCDT and Anglada's gift-wrapping algorithm compiled by "gcc -O3" on a MacBook Pro with a 3.06 GHz Intel dual core, 8 GB memory (1.07 GHz DDR3 SDRAM), and a 6 MB level-two cache. Each number is an average over 10,000 runs with different jittering for each run.

to the distances between the vertices. Although these examples do not represent the variety of cavities that can come up in practice, they *are* representative of the most common ways that the cavity geometry influences the running times. Collinear points, which occur frequently in practice, bring out the worst of gift-wrapping's quadratic running time. Jittered vertices hide the quadratic growth until the number of vertices is greater, because the gift-wrapping algorithm has more luck finding balanced subdivisions of small recursive subproblems.

Table 1 tabulates average running times and numbers of predicate calls for the two algorithms. Calls to Orient in Line 4 of CavityCDT are not counted because we optimized Line 4 by taking advantage of subexpressions that are reused every time it iterates and by not using exact arithmetic. We optimized Line 21 to reduce the number of Orient tests by taking advantage of the fact that if a call to InsertVertex finds that its input triangle $\triangle uvw$ has positive orientation, all its recursive calls to InsertVertex will also have positively oriented input triangles that do not require testing.

Given the cavities with collinear vertices as input, CavityCDT becomes faster than gift-wrapping for 30 vertices or more, and the advantage grows rapidly. Given the jittered cavities, CavityCDT is faster for 85 vertices or more. We observe that CavityCDT performs fewer InCircle calls on average for 8 or more vertices. In a higher-precision implementation with expensive InCircle calls, the balance would shift further in favor of CavityCDT.

Because both algorithms are easy to implement, a CDT construction program can choose between them according to the cavity size. We grant that cavities of 85 vertices or more are a small minority of those that arise in real-world CDT construction, but it would take just one segment that crosses 100,000 edges to make one regret not having the linear-time algorithm as an option.

6. The Cost of Randomized Incremental Segment Insertion

Agarwal, Arge, and Yi [1] show that when k segments are inserted into an n-vertex CDT in random order, the total expected number of structural changes (triangles and edges created and deleted) is in $O(n \log^2 k)$. For completeness, we reprise their analysis, filling in many missing details. In Section 7, we exhibit a PSLG for which this bound is tight. Thus, if an $O(n \log k)$-time incremental segment insertion algorithm exists, it will require a smarter insertion order, not just a better analysis.

Let $\mathcal{X}$ be a PSLG with n vertices and k segments. Let S be the set of segments in $\mathcal{X}$. We use the theory of ϵ-nets to bound the expected maximum number of segments in $\mathcal{X}$ that a line segment in the plane can cross without crossing any segment in $\mathcal{X}$ that has already been inserted. Different line segments in the plane can intersect different subsets of S, but unless k is small, not all 2^k subsets are possible. The plane imposes a structure such that the number of possible subsets is polynomial in k.

Lemma 6. *Let S be a set of k non-crossing segments in the plane. Consider the sequence of segments in S whose relative interiors intersect a fixed line, written in the order of the intersection points. Let Q be the set of all such sequences, for all lines in the plane, with the proviso that a sequence and its reverse are considered equivalent and are not counted as distinct members of Q. The cardinality of Q is at most $8k^2 + 1$.*

Proof. We use the standard planar geometric duality by which a point $p = (p_x, p_y)$ dualizes to a line p^* with the formula $y = p_x x - p_y$, and vice versa. Planar duality preserves incidences between points and lines, so if a primal point p lies on a primal line ℓ, then the dual point ℓ^* lies on the dual line p^*. Let U be the set of vertices of the segments in S; then $|U| \leq 2k$. Vertical lines do not have duals in this formulation, so assume without loss of generality that the coordinate system is rotated so that no two vertices in U have the same x-coordinate. Hence, every vertical line can be tilted slightly so that it is not perfectly vertical but it intersects the same segment interiors as before.

Let $\mathcal{A}$ be the arrangement (expressed as a polygonal complex) formed by the lines dual to the vertices in U. The total number of faces of all dimensions in $\mathcal{A}$—vertices, edges, and 2-faces—is at most $2(2k)^2 + 1 = 8k^2 + 1$ [44]. If two primal lines ℓ_1 and ℓ_2 dualize to points in the same face of $\mathcal{A}$, it is possible to translate and rotate ℓ_1 to ℓ_2 without causing it to pass over a vertex in U or change its incidences with the vertices in U; it follows that ℓ_1 and ℓ_2 intersect the same segments in S in the same order. Therefore, for every sequence of segments $Q \in \mathcal{Q}$, there is a face of $\mathcal{A}$ whose interior points all dualize to lines that generate the sequence Q. It follows that the number of sequences in $\mathcal{Q}$ is less than or equal to the number of faces of $\mathcal{A}$. □

Lemma 7. *Let S be a set of k non-crossing segments in the plane. For every possible line segment e in the plane, consider the subset of segments in S whose relative interiors intersect e. Let M be the set of all such subsets, for all line segments in the plane. The cardinality of M is at most $f(k)$, where $f(k) = k(k + 1)(8k^2 + 1)/2 + 1$.*

Proof. The sequence of segments in S whose relative interiors intersect a line segment is an interval taken from the sequence of segments whose relative interiors intersect a line. By Lemma 6, there are at most $8k^2 + 1$ such sequences, each having length at most k. A sequence of length k has $k(k + 1)/2$ nonempty intervals. We add one for the empty set. □

For our purposes, the exact bound given by Lemma 7 is not important; it suffices that the bound is polynomial, rather than exponential, in k.

Next, we reprise a famous result of Haussler and Welzl [18] on ϵ-nets. Let S_1 and S_2 be two random sequences obtained by taking i independent samples from S with replacement (hence segments may be

repeated). For some $\epsilon \in (0, 1)$ and $k = |S|$, let E be the event that there exists a line segment e that does not intersect any segment in S_1, but e intersects at least ϵk segments in S. If E does not occur, S_1 is said to be an *ϵ-net* for S. We want to show that E is very unlikely when i is sufficiently large, thus S_1 is very likely to be an ϵ-net. Let E_+ be the event that there exists a line segment e that does not intersect any segment in S_1, but intersects at least ϵk segments in S and at least $\epsilon i - 1$ segments in S_2. (Repeated segments in S_2 may be counted multiple times.) The following lemma shows that event E usually entails event E_+.

Lemma 8 (Adapted from Haussler and Welzl [18], Lemma 3.4)**.** $\Pr[E] < 2\Pr[E_+]$.

Proof. Suppose that event E occurs; then we will show that E_+ also occurs with probability at least $1/2$. Event E implies that there exists a line segment e that intersects at least ϵk of the k segments in S. Let γ be the number of segments in S_2 that e intersects; if $\gamma \geq \epsilon i - 1$, then event E_+ also occurs. Because S_2 is chosen by randomly choosing i segments from S with replacement, γ is a binomial random variable with expectation ϵi. Let η be the median value of γ—the lesser median if there are two medians. Kaas and Buhrman [22] show that for any binomial distribution, $\lfloor \epsilon i \rfloor \leq \eta \leq \lceil \epsilon i \rceil$. Therefore, the probability that event E_+ also occurs when E occurs is

$$\Pr[\gamma \geq \epsilon i - 1] \geq \Pr[\gamma \geq \lfloor \epsilon i \rfloor] \geq \Pr[\gamma \geq \eta] > 1/2.$$

Thus $\Pr[E_+] = \Pr[E] \cdot \Pr[E_+|E] > \Pr[E]/2$. □

Let E_* be the event that there exists a line segment e that intersects no segment in S_1, but intersects at least $\epsilon i - 1$ segments in S_2.

Lemma 9 (Adapted from Haussler and Welzl [18], Lemma 3.5)**.** $\Pr[E_+] \leq \Pr[E_*] \leq f(2i)\, 2^{1-\epsilon i}$*, where* $f(j) = j(j+1)(8j^2+1)/2 + 1$.

Proof. The first inequality follows because event E_+ implies event E_*. For the second inequality, imagine sampling $2i$ segments from S with replacement to form a sequence S_{12}, then randomly choosing i of those segments to form S_1; the remainder form S_2.

Let e be a fixed line segment in the plane, and let c be the number of segments that intersect e among the $2i$ segments in S_{12}. The line segment e can trigger the event E_* only if none of those c segments are chosen for S_1, which happens with probability

$$\binom{2i-c}{i} \Big/ \binom{2i}{i} = \frac{i}{2i} \cdot \frac{i-1}{2i-1} \cdots \frac{i-c+1}{2i-c+1} \leq 2^{-c}.$$

Moreover, e can trigger E_* only if $c \geq \epsilon i - 1$, so the probability that e triggers E_* is at most $2^{1-\epsilon i}$. (The magic of this upper bound is that it is independent of how many segments in S intersect e.)

Now, consider the probability that *any* line segment in the plane triggers E_*. By Lemma 7, there are at most $f(2i)$ subsets of S_{12} that a line segment e can pick out. If two line segments intersect exactly the same segments in S_{12}, then either both of them trigger E_*, or neither do. Therefore, the probability of event E_* is at most $f(2i)\, 2^{1-\epsilon i}$. □

Let $\Pi = \langle s_1, s_2, \ldots, s_k \rangle$ be a permutation of the k segments in $\mathcal{X}$, chosen uniformly at random from the set of all such permutations. Let $\mathcal{T}_0$ be the Delaunay triangulation of the n vertices in $\mathcal{X}$, ignoring the segments. For $i \in [0, k]$, let $\mathcal{T}_i$ be the CDT constructed by inserting the first i segments in Π.

A *conflict* is a segment-edge pair (s, e) consisting of an edge $e \in \mathcal{T}_i$ and a segment $s \in \mathcal{X}$ that crosses e. When the segment s_{i+1} is inserted into the triangulation $\mathcal{T}_i$, it deletes every edge in $\mathcal{T}_i$ it conflicts with. An edge e is said to have c conflicts if it crosses c segments in $\mathcal{X}$.

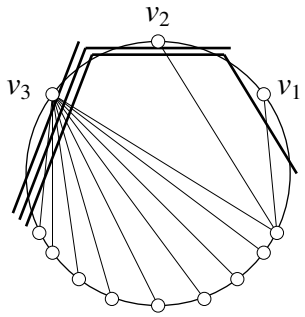

Figure 15: The pulling vertices v_1, v_2, and v_3 conflict with one, two, and three segments (thick lines), respectively. The triangulation (thin lines) connects the $\Theta(n)$ pushing vertices to v_3 until a segment conceals it.

Theorem 10 (Agarwal, Arge, and Yi [1])**.** *The expected number of edges deleted over the duration of the randomized incremental segment insertion algorithm is in* $O(n \log^2 k)$.

Proof. By Lemmas 8 and 9, the probability $\Pr[E]$ that there exists a line segment that intersects at least ϵk segments in $\mathcal{X}$ but intersects no segment among i segments sampled randomly from $\mathcal{X}$ with replacement satisfies $\Pr[E] < 4 f(2i) 2^{-\epsilon i}$. This probability is not increased by sampling without replacement, as the incremental algorithm does. Setting $\epsilon = (5 \log_2 k)/i$ yields $\Pr[E] < 4 f(2i)/k^5 \in O(1/k)$. Therefore, the first i randomly chosen segments are likely to be a $(5 \log_2 k)/i$-net for the segments in $\mathcal{X}$.

Let e be an edge with c conflicts in the triangulation $\mathcal{T}_i$. When a randomly chosen segment s_{i+1} is inserted, the probability that e is deleted is $c/(k-i)$. The probability that there exists an edge with more than $(5k \log_2 k)/i$ conflicts is at most $\Pr[E]$. As $\mathcal{T}_i$ has fewer than $3n$ edges, the expected number of edges deleted over the duration of the algorithm is less than

$$3n + 3n \sum_{i=1}^{k-1} \left(\frac{5k \log_2 k}{i(k-i)} + \Pr[E] \right) \in O(n \log^2 k). \qquad \square$$

Therefore, the total expected cost of all calls to CavityCDT is in $O(n \log^2 k)$. With the cost of constructing the initial Delaunay triangulation $\mathcal{T}_0$ and performing segment location for each segment prior to inserting it (see Section 8), the incremental CDT construction algorithm runs in expected $O(n \log n + n \log^2 k)$ time.

7. A Lower Bound for Randomized Incremental Segment Insertion

There is a matching $\Omega(n \log^2 k)$ lower bound on the expected number of structural changes. To see this, consider the PSLG in Figure 15, which is similar to an example Agarwal, Arge, and Yi [1] use to establish an $\Omega(n \log k)$ lower bound. The example uses two sequences of nearly cocircular vertices. On the bottom half of the circle is a sequence of $\Theta(n)$ *pushing vertices* that lie precisely on the circle. On the top half is a smaller sequence of m *pulling vertices* $v_1, v_2, \ldots, v_m$ that are perturbed so they lie just inside the circle. Each pulling vertex v_j is concealed by j segments, all of whose endpoints lie outside the circle. The total number of segments is $k = m(m+1)/2$. No segment conceals more than one pulling vertex. The pulling vertices are placed so that edges of the CDT connect every pushing vertex to the pulling vertex with greatest index that is not concealed, which we call the *dominant* pulling vertex. Every edge connecting a pushing vertex to a pulling vertex v_j has j conflicts. Say that v_j is *alive* if no segment that conceals it has been inserted yet, and

call the j segments that conceal v_j its *conflicts*. When a newly inserted segment conflicts with the dominant pulling vertex, $\Theta(n)$ edges are deleted and $\Theta(n)$ new edges are created adjoining the new dominant pulling vertex.

We analyze the longevity of a pulling vertex v_c with c conflicts with a method developed by Clarkson [13]. After i segments have been inserted, the likelihood that v_c is still alive is

$$\begin{aligned}\Pr[v_c \text{ alive}] &= \binom{k-c}{i} \Big/ \binom{k}{i} = \frac{k-i}{k} \cdot \frac{k-i-1}{k-1} \cdots \frac{k-i-c+1}{k-c+1} \\ &> \left(\frac{k-i-c}{k-c}\right)^c.\end{aligned}$$

Consider the $(i+1)$th segment insertion for any

$$i \in \left[\frac{\alpha k \ln k}{m}, \min\left\{\frac{\alpha}{2}k^{1-2\alpha}\ln k, \frac{k}{2} - m\right\}\right]$$

where $\alpha > 0$ is a constant we will choose shortly. For every pulling vertex v_c that has $c \le (\alpha k \ln k)/i$ conflicts, the range of i implies that $c \le m$ and $i \le k/2 - c$, so

$$\begin{aligned}\Pr[v_c \text{ alive}] &> \left(\frac{k-i-c}{k-c}\right)^{(\alpha k \ln k)/i} = \left(1 + \frac{i}{k-i-c}\right)^{-(\alpha k \ln k)/i} \\ &\ge e^{-(\alpha k \ln k)/(k-i-c)} = k^{-\alpha k/(k-i-c)} \\ &\ge k^{-2\alpha}.\end{aligned}$$

The middle inequality follows because $1 + x \le e^x$ for all x. Consider the $(\alpha k \ln k)/(2i)$ vertices in the range from $v_{(\alpha k \ln k)/(2i)}$ to $v_{(\alpha k \ln k)/i}$. The probability that at least one of them is alive is

$$\begin{aligned}\Pr[\text{one of } v_{(\alpha k \ln k)/(2i)}, \ldots, v_{(\alpha k \ln k)/i} \text{ alive}] &> 1 - \left(1 - k^{-2\alpha}\right)^{(\alpha k \ln k)/(2i)} \\ &\ge 1 - e^{-(\alpha k^{1-2\alpha} \ln k)/(2i)}.\end{aligned}$$

We assume that $i \le \frac{\alpha}{2}k^{1-2\alpha}\ln k$, so with probability at least $1 - e^{-1} > 0.63$ there is a live pulling vertex with at least $(\alpha k \ln k)/(2i)$ conflicts. Edges connect the live pulling vertex with the most conflicts to the $\Theta(n)$ pushing vertices, and the $(i+1)$th segment insertion deletes these edges with probability at least $(\alpha k \ln k)/(2i(k-i))$. We choose α to be a positive constant less than $1/4$. Therefore, the expected number of edges deleted during the duration of the algorithm is at least

$$\sum_{i=(\alpha k \ln k)/m \in \Theta(\sqrt{k}\ln k)}^{\min\left\{\frac{\alpha}{2}k^{1-2\alpha}\ln k,\, k/2-m\right\}} 0.63\frac{\alpha k \ln k}{2i(k-i)}\Theta(n) = \Theta(n \log^2 k).$$

The cogent observation is that there is a range of values for i spanning a factor of $k^{0.5-2\alpha}$ in which the dominant live vertex has many conflicts. Although this range is narrow (and hidden—it took us a long time to realize it existed), it suffices to allow the summation $\sum 1/i$ to contribute a $\Theta(\log k)$ factor. In this range, each doubling of i contributes $\Theta(n \log k)$ structural changes, and i is doubled $\Theta(\log k)$ times. Figure 16 depicts this idea visually.

Our lower bound example is related to the coupon collector's problem. Imagine that the pulling vertices represent a set of m types of coupons you wish to collect, and that when you buy a coupon (choose a random

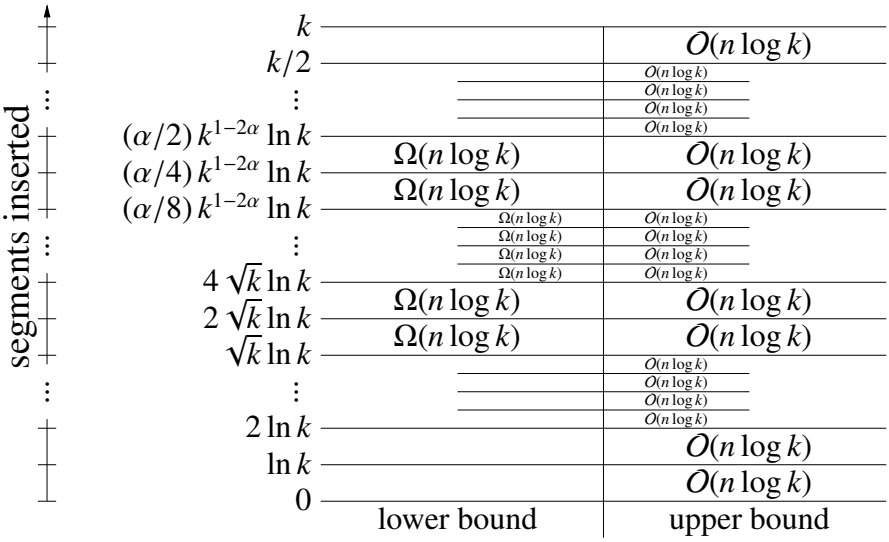

Figure 16: For each doubling of the number i of inserted segments from $i = \ln k$ to $i = k$, at most expected $O(n \log k)$ structural changes occur, establishing the upper bound of $O(n \log^2 k)$. For each doubling from $i \in \Theta(\sqrt{k} \ln k)$ to $i \in \Theta(k^{1-2\alpha} \ln k)$, at least expected $\Omega(n \log k)$ structural changes occur for the PSLG in Figure 15, establishing the lower bound of $\Omega(n \log^2 k)$.

segment), it is of type v_i with some probability p_i. Coupon collection can be modeled as a Poisson process; the probability of collecting every coupon $v_{i+1}, v_{i+2}, \ldots, v_m$ before collecting v_i is [7]

$$\Pr[v_i \text{ after } v_{i+1}, \ldots, v_m] = \int_0^\infty p_i e^{-p_i t} \prod_{j>i} \left(1 - e^{-p_j t}\right) \mathrm{d}t.$$

Our analysis is equivalent to asking, if $p_i = i/k$ where $k = \sum_{i=1}^{m} i$, how often do you collect a coupon for the first time such that every coupon with greater index has already been collected? We have not been able to find a published asymptotic bound in terms of m for this problem, but we have just shown that the expected answer is in $\Theta(\log^2 k) = \Theta(\log^2 m)$. For comparison, if every coupon arrives with equal probability, the expected answer is in $\Theta(\log m)$.

It is well known that the problem of sorting a set of numbers can be reduced to computing a triangulation (Delaunay or not) of a point set, so every algorithm for computing a triangulation of n points takes $\Omega(n \log n)$ worst-case time in the decision tree model of computation. Therefore, CDT construction by uniformly randomized incremental segment insertion takes expected $\Omega(n \log n + n \log^2 k)$ time for worst-case PSLGs.

8. The Cost of Segment Location

Recall that for an uninserted segment s, segment location is the act of identifying a triangle in the current CDT whose interior intersects s. We perform segment location with a simple rotary traversal of the triangles adjoining the endpoint of s with lesser degree, taking time proportional to that endpoint's degree. One way to identify the endpoint of lesser degree is to record the vertex degrees and update them with every change to the triangulation. A simpler alternative is to perform simultaneous rotary searches around both endpoints of s, interleaving steps around one endpoint with steps around the other; a suitable triangle will be found in time proportional to the lesser degree.

In practice, most segment location operations take $O(1)$ time. But what if both endpoints of many segments have large degrees? Here, we show that the total cost of this segment location method is at worst

proportional to the number of vertices plus the number of structural changes that occur during segment insertion. Therefore, segment location never worsens the asymptotic running time of incremental CDT construction. This result holds regardless of the order in which the segments are inserted, and does not depend upon randomization.

Consider incrementally constructing the CDT of a PSLG $\mathcal{X}$ with n vertices and k segments. Let l be the total number of edges that are created during all the segment insertion operations, including edges that are subsequently deleted. We have seen in Section 6 that in expectation, $l \in O(n \log^2 k)$ when the segment insertion order is randomized; but often l is much smaller.

Theorem 11. *During incremental construction of the CDT of $\mathcal{X}$ (whether randomized or not), the total time for segment location as described above is in $O(n + l)$.*

Proof. As segments are inserted into the CDT, the degrees of the vertices change. For $i \in [0, k]$, let $\mathcal{T}_i$ be the CDT after the first i segments have been inserted. For $j \in [1, n]$, let d_j be the maximum degree of the vertex with index j over all the triangulations $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$. It follows from Euler's formula that the sum of vertex degrees in $\mathcal{T}_0$ is at most $6n - 12$, so $\sum_{j=1}^{n} d_j \leq 6n + 2l - 12$. The following argument shows that all the segment location operations together take $O(\sum_{j=1}^{n} d_j)$ time, and therefore take $O(n + l)$ time.

Suppose that we index the vertices in nonincreasing order of maximum degree, so that $d_i \geq d_j$ whenever $i < j$. (This indexing is not related to the insertion order!) The time to locate a segment $s \in \mathcal{X}$ whose vertices are numbered i and j with $i < j$ is at worst proportional to $\min\{d_i, d_j\} = d_j$; call that number the *cost* of s. For all $i \in [3, n]$, let $\mathcal{X}_i \subseteq \mathcal{X}$ be the PSLG induced by taking only the first i vertices in $\mathcal{X}$ and the segments whose endpoints are both among the first i vertices. Let the *cost* of $\mathcal{X}_i$ be the total cost of all its segments.

Observe that the total time for segment location is proportional to the cost of $\mathcal{X} = \mathcal{X}_n$. We show that this total is less than $3 \sum_{j=1}^{n} d_j$. As $\mathcal{X}$ is planar, $\mathcal{X}_i$ has at most $3i - 6$ edges as a consequence of Euler's formula. $\mathcal{X}_3$ has at most three edges (a triangle) with total cost at most $d_2 + 2d_3$. $\mathcal{X}_4$ has at most six edges, but at most three of those are in $\mathcal{X}_3$; every edge in $\mathcal{X}_4$ but not in $\mathcal{X}_3$ has cost d_4. Hence $\mathcal{X}_4$ has cost at most $d_2 + 2d_3 + 3d_4$. Similarly, $\mathcal{X}_5$ has at most nine edges. If $\mathcal{X}_5$ has edges not in $\mathcal{X}_4$, they cost d_5 each, so $\mathcal{X}_5$ has cost at most $d_2 + 2d_3 + 3d_4 + 3d_5$. By inductive application of this reasoning, the cost of $\mathcal{X}_i$ is at most $d_2 + 2d_3 + 3 \sum_{j=4}^{i} d_j$, and the cost of $\mathcal{X}$ is less than $3 \sum_{j=1}^{n} d_j$. □

9. Extensions to Three Dimensions

Incremental construction of constrained Delaunay triangulations generalizes to three or more dimensions, albeit under limited circumstances. Not every polyhedron has a CDT—or even a tetrahedralization [31, 30]—and not every CDT can be constructed incrementally. However, with the insertion of extra vertices on its edges, every polyhedron can be converted into a structure that does have a CDT that can be constructed incrementally [37, 42]. This fact is used by some algorithms for generating high-quality tetrahedral meshes [40, 41].

The natural analog of a PSLG in three dimensions is called a *piecewise linear complex* (PLC), introduced by Miller et al. [28]. For our purposes, a three-dimensional PLC $\mathcal{X}$ is a set of vertices, segments, and polygons (not necessarily convex) that obey certain rules characteristic of complexes; for example, the boundary of every polygon in $\mathcal{X}$ must be a union of segments in $\mathcal{X}$, the endpoints of every segment in $\mathcal{X}$ must be vertices in $\mathcal{X}$, and the intersection of any two elements of $\mathcal{X}$ must be a union of elements of $\mathcal{X}$. See elsewhere [9, 39] for a complete definition.

The segments and polygons in $\mathcal{X}$ constrain how $\mathcal{X}$ can be triangulated. For example, to support the application of boundary conditions, a PLC $\mathcal{X}$ may have vertices and segments floating in the relative interior of a polygon, or vertices, segments, and polygons floating inside the convex hull of $\mathcal{X}$; a triangulation of $\mathcal{X}$

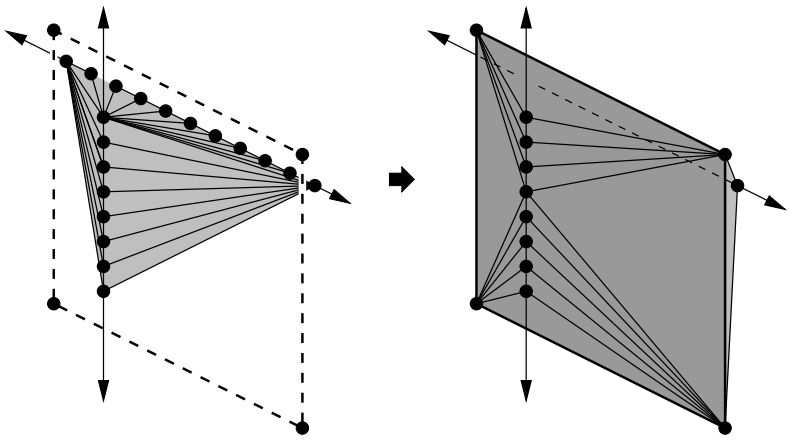

Figure 17: The Delaunay triangulation on the left has n vertices and $\Theta(n^2)$ tetrahedra. The insertion of a single square polygon causes most of the tetrahedra to be deleted.

must respect these constraints. A *triangulation of a PLC* $\mathcal{X}$ is a simplicial complex $\mathcal{T}$ such that $\mathcal{T}$ contains the same vertices as $\mathcal{X}$, every segment in $\mathcal{X}$ is an edge in $\mathcal{T}$, every polygon in $\mathcal{X}$ is a union of triangles in $\mathcal{T}$, and the union of simplices $\bigcup_{s\in\mathcal{T}} s$ is the convex hull of the vertices in $\mathcal{X}$. A *constrained Delaunay triangulation (CDT) of* $\mathcal{X}$ is a triangulation of $\mathcal{X}$ in which every triangular face is either locally Delaunay or included in a polygon in $\mathcal{X}$. A CDT differs from an ordinary Delaunay triangulation in that a triangular face that lies in a polygon is not required to be locally Delaunay.

A PLC $\mathcal{X}$ is said to be *edge-protected* if for each segment $s \in \mathcal{X}$, there exists a closed ball whose boundary passes through the endpoints of s, but the ball contains no other vertex in $\mathcal{X}$. If $\mathcal{X}$ is edge-protected, every Delaunay triangulation of the vertices in $\mathcal{X}$ contains every segment in $\mathcal{X}$, but it does not necessarily respect the polygons in $\mathcal{X}$ (and therefore it is not necessarily a triangulation of $\mathcal{X}$). The *CDT Theorem* [39] states that every edge-protected PLC has a CDT, which does respect the polygons.

An incremental polygon insertion algorithm of Shewchuk [38] constructs the CDT of an n-vertex, edge-protected PLC by constructing a Delaunay triangulation of the vertices, then inserting the polygons one by one in a manner that performs at most $O(n^2)$ structural changes. (The running time of the algorithm is in $O(n^2 \log n)$; the extra $\log n$ factor accounts for a priority queue that controls the order in which the structural changes occur.) This number of structural changes is asymptotically worst-case optimal: Figure 17 depicts a PLC for which the insertion of just one polygon causes $\Theta(n^2)$ structural changes. Thus the question of the number of structural changes during polygon insertion in worst-case PLCs is resolved, although the question of time complexity is not; we hope that an $O(n^2)$-time CDT construction algorithm will be discovered.

However, there are reasons to believe that CDT construction is faster for most PLCs that arise in practice. It is well known that the construction of a Delaunay triangulation takes $\Omega(n^2)$ time in the worst case, because some Delaunay triangulations have $\Theta(n^2)$ tetrahedra; but for many point sets, the randomized incremental algorithm for constructing Delaunay triangulations runs in expected $O(n \log n)$ time and performs expected $O(n)$ structural changes. This circumstance occurs when the Delaunay triangulation has size $O(n)$ and the expected size of the Delaunay triangulation of a random subset of the input vertices is linear in the size of the random subset [3]. Many real-world inputs have this property, which explains why three-dimensional incremental Delaunay triangulators are observed to run quickly in practice on most inputs. Can a similar statement be made about three-dimensional incremental CDT construction?

Here we show that, under assumptions that apply to many common real-world inputs, the expected number of structural changes required for randomized incremental construction of the CDT of an edge-protected

PLC with n vertices, k segments, and m polygons is in $O(n \log k \log m)$, matching the bound for CDT construction in the plane. (Note that $m \in O(k)$; see Lemma 12 below.) However, we do not know an algorithm that inserts a polygon in time linear in the number of structural changes. Under somewhat stronger assumptions, Shewchuk's CDT construction algorithm [38], randomized, runs in expected $O(n \log n \log k \log m)$ time. Note that fast incremental polygon insertion also requires a fast polygon location algorithm, which we describe in Section 10.

We begin with a useful lemma. A polygon can have arbitrarily many segments, and a segment can be an edge of arbitrarily many polygons, but the number of polygons is limited by the number of segments.

Lemma 12. *Let $\mathcal{X}$ be a PLC in $\mathbb{R}^3$ with $k > 0$ segments and m polygons. Then $m < 2k$.*

Proof. Imagine placing a sphere of infinitesimal radius around a vertex $v \in \mathcal{X}$. The segments and polygons adjoining v induce a planar graph on the sphere: each segment adjoining v pierces the sphere at a single point, and each polygon adjoining v pierces the sphere in a circular arc adjoining two or more such points. Define a graph by considering only the arcs subtending angles strictly less than 180° on the sphere; thus there cannot be more than one arc adjoining any pair of points. If a polygon $f \in \mathcal{X}$ induces such an arc for v, call v a *convex corner* of f. Each polygon has at least three convex corners, because every vertex of its convex hull is a convex corner.

The number of arcs in a nonempty planar graph (with no pair of points connected by more than one arc and no point connected to itself) is less than three times its number of points. By summing these numbers over every vertex $v \in \mathcal{X}$, we find that the number of convex corners in $\mathcal{X}$ is less than three times the number of segment-vertex incidences in $\mathcal{X}$. Therefore, the number of polygons in $\mathcal{X}$ is less than the number of segment-vertex incidences, which is twice the number of segments. □

Next, we ask how many different subsets of the PLC polygons can be stabbed by a line or a line segment.

Lemma 13. *Let $\mathcal{X}$ be a PLC in $\mathbb{R}^3$ with k segments and m polygons. Consider the sequence of polygons in $\mathcal{X}$ whose relative interiors[1] intersect a fixed line, written in the order of the intersection points, leaving out polygons coplanar with the line. Let $\mathcal{Q}$ be the set of all such sequences, for all lines in $\mathbb{R}^3$. The cardinality of $\mathcal{Q}$ is in $O(k^4)$.*

Proof. Imagine extending each segment in $\mathcal{X}$ to a line in $\mathbb{R}^3$, and let L be the set of these lines. Observe that $|L| \leq k$. To any pair of lines (ℓ, ℓ') in $\mathbb{R}^3$, we assign a *mutual orientation* designated zero, positive, or negative. If ℓ and ℓ' are coplanar, their mutual orientation is zero. Otherwise, imagine each line as a vector directed from lexicographically lesser to lexicographically greater coordinates. We say that the mutual orientation of (ℓ, ℓ') is positive if the cross product $\ell \times \ell'$ points from ℓ to ℓ', and negative otherwise.

We map each line $\ell \subset \mathbb{R}^3$ to a point in $\mathbb{R}^4$ by taking the y- and z-coordinates where ℓ intersects the plane $x = 0$ and concatenating the y- and z-coordinates where ℓ intersects the plane $x = 1$. (This map does not take lines perpendicular to the x-axis, but we can account for those lines by repeating the argument with the coordinate system rotated.) Consider a line ℓ that maps to a point (y, z, v, w) and a line ℓ' that maps to a point (y', z', v', w'). The mutual orientation of the two lines is the sign of $(y - y')(w - w') + (z - z')(v - v')$; for example, ℓ and ℓ' are coplanar if and only if this expression is zero.

[1]Rigorously speaking, "relative interior" is the wrong term here, because PLCs allow vertices and segments to float in the relative interior of a polygon, and intersections of the fixed line with these vertices and segments should not count as intersections with the polygon. Read "relative interior" as shorthand for "the polygon after removing all points in the PLC vertices and segments." With this adjustment, the polygons in a PLC truly have disjoint "relative interiors," because the definition of PLC requires that if two polygons intersect, their intersection is a union of PLC segments and vertices.

If we treat $(y, z, v, w) \in \mathbb{R}^4$ as a vector of independent variables and (y', z', v', w') as a constant, the expression is a quadratic function from $\mathbb{R}^4$ to $\mathbb{R}$ whose sign is the orientation of (ℓ, ℓ') for a fixed ℓ' and a varying ℓ. The *zero-surface* of this function is the set of points in $\mathbb{R}^4$ satisfying $(y-y')(w-w')+(z-z')(v-v') = 0$. Consider the arrangement $\mathcal{A}$ of surfaces in $\mathbb{R}^4$ defined by overlaying these zero-surfaces for every $\ell' \in L$. If two lines ℓ_1 and ℓ_2 map to points in the same face of $\mathcal{A}$, then it is possible to translate and rotate ℓ_1 to ℓ_2 without changing its orientation with respect to any line in L. It follows that ℓ_1 and ℓ_2 intersect the relative interiors of the same polygons in $\mathcal{X}$ in the same order, possibly excepting polygons coplanar with both ℓ_1 and ℓ_2. The maximum number of faces in such an arrangement of zero-surfaces is in $O(k^4)$ [29, 2], and thus the number of sequences in Q is also in $O(k^4)$. □

Lemma 14. *Let $\mathcal{X}$ be a PLC in $\mathbb{R}^3$. Let P be a subset of the polygons in $\mathcal{X}$. Let m be the number of polygons in P, and let k be the number of segments of those polygons (i.e., segments in $\mathcal{X}$ included in at least one polygon in P). Consider the subset of polygons in P whose relative interiors intersect a fixed line segment, leaving out polygons coplanar with the line segment. Let M be the set of all such subsets, for all line segments in $\mathbb{R}^3$. The cardinality of M is in $O(k^4m^2)$.*

Proof. Follows from Lemma 13 by the same reasoning used to prove Lemma 7. □

Let $\mathcal{X}$ be an edge-protected PLC. Let $\Pi = \langle f_1, f_2, \ldots, f_m \rangle$ be a permutation of the m polygons in $\mathcal{X}$, chosen uniformly at random from the set of all such permutations. Let $\mathcal{T}_0$ be the Delaunay triangulation of the n vertices in $\mathcal{X}$, ignoring the polygons. Because $\mathcal{X}$ is edge-protected, $\mathcal{T}_0$ contains every segment in $\mathcal{X}$. For $i \in [0, m]$, let $\mathcal{T}_i$ be the CDT constructed by inserting the first i polygons in Π.

A *conflict* is a polygon-edge pair (f, e) consisting of an edge $e \in \mathcal{T}_i$ and a polygon $f \in \mathcal{X}$ whose relative interior intersects the relative interior of e. When the polygon f_{i+1} is inserted into the triangulation $\mathcal{T}_i$, it deletes every edge in $\mathcal{T}_i$ it conflicts with. An edge e is said to have c conflicts if it crosses c polygons in $\mathcal{X}$.

Theorem 15. *In the randomized incremental polygon insertion algorithm, suppose that the expected number of tetrahedra in each intermediate triangulation $\mathcal{T}_0, \ldots, \mathcal{T}_m$ is in $O(g(n))$ for some function $g(n)$. The expected number of structural changes made over the duration of the algorithm is in $O(g(n) \log k \log m)$.*

Proof. Lemmas 8 and 9 extend to polygons in three dimensions, with the function $f(j)$ in Lemma 9 replaced by another function satisfying $f(j) \in O(k^4 j^2)$ per Lemma 14. Thus, the probability $\Pr[E]$ that there exists a line segment that intersects at least ϵm polygons in $\mathcal{X}$ but intersects no polygon among $i \leq m$ polygons sampled randomly from $\mathcal{X}$ with replacement satisfies $\Pr[E] \in O(4k^4(2i)^2\, 2^{-\epsilon i})$. This probability is not increased by sampling without replacement, as the incremental algorithm does. As $i \leq m < 2k$ by Lemma 12, setting $\epsilon = (7 \log_2 k)/i$ yields $\Pr[E] \in O(1/k)$. Therefore, the first i randomly chosen polygons are likely to be a $(7 \log_2 k)/i$-net for the polygons in $\mathcal{X}$.

Let e be an edge with c conflicts in the triangulation $\mathcal{T}_i$. When a randomly chosen polygon f_{i+1} is inserted, the probability that e is deleted is $c/(m - i)$. The probability that there exists an edge with more than $(7m \log_2 k)/i$ conflicts is at most $\Pr[E]$. A tetrahedron is deleted only if one of its six edges is deleted. By assumption, $\mathcal{T}_i$ has expected $O(g(n))$ tetrahedra, so the expected number of tetrahedra deleted over the duration of the algorithm is in

$$O\left(6\, g(n) \sum_{i=1}^{m-1} \left(\frac{7m \log_2 k}{i(m-i)} + \Pr[E] \right)\right) = O(g(n) \log k \log m).$$

The final triangulation $\mathcal{T}_m$ has expected $O(g(n))$ tetrahedra, so the expected number of tetrahedra created is also in $O(g(n) \log k \log m)$. □

Moreover, Shewchuk [38] shows that the total number of structural changes is in $O(n^2)$, regardless of the polygon insertion order, so with a randomized insertion order the expected number of structural changes is in

$$O(\min\{g(n) \log k \log m, n^2\}).$$

Unfortunately, there is no known algorithm that can retriangulate the cavities evacuated by the insertion of a polygon in time linear in the number of structural changes. Shewchuk's algorithm for polygon insertion, which is based on bistellar flips, incurs two additional costs: an $O(\log n)$-time cost per flip to perform priority queue (binary heap) operations that ensure that flips occur in the correct order; and the fact that the number of bistellar flips could far exceed the number of necessary structural changes because some tetrahedra are created only to be immediately deleted again during a single polygon insertion operation, though the total number of flips never exceeds $O(n^2)$. Experiments [43] suggest that it is uncommon for the number of flips to exceed the number of deleted tetrahedra by more than a small constant, and our intuition is that such circumstances are analogous to the circumstances in which a Delaunay triangulation has a superlinear size—possible for inputs exhibiting a certain regular structure, but not the norm. For many, probably most, PLCs that arise in practice, $g(n) \in O(n)$, so we anticipate that randomized incremental polygon insertion implemented with bistellar flips will often run in $O(n \log n \log k \log m)$ time in practice.

10. Polygon Location in Three-Dimensional CDTs

Just as segment location is the first step to inserting a segment into a planar CDT, the first step to inserting a polygon f into a three-dimensional CDT is *polygon location*: the act of identifying a tetrahedron in the current CDT whose interior intersects f.

We assume that every segment of f is an edge of the current triangulation (a safe assumption if the underlying PLC is edge-protected). In analogy to vertex degrees, we define the *degree* of an edge in a triangulation to be the number of tetrahedra that include the edge. Our polygon location method is to perform a simple rotary traversal of the tetrahedra around the segment of f with least degree, taking time proportional to that segment's degree in the current triangulation.

To analyze segment insertion in the plane, in Section 8 we used the fact that after one triangle is found whose interior intersects the new segment, the other conflicting triangles can be found by a depth-first search in time linear in their number, and the cost of the search can be charged to the time spent deleting those triangles. Unfortunately, this charging scheme does not suffice to analyze polygon insertion in three dimensions, because a polygon might already be partially present in the CDT: the CDT may already contain triangles that are subsets of the polygon, as well as edges coinciding with the polygon's relative interior. We can still use a careful depth-first search to find all the tetrahedra whose interiors intersect the new polygon, but the cost of that search is proportional to the number of tetrahedra deleted *plus* the degrees of the triangulation edges that are included in the new polygon but are not PLC segments (because it may be necessary to search all the tetrahedra adjoining these edges). Fortunately, a triangulation edge that is not a PLC segment can be included in only one PLC polygon.

Consider incrementally constructing the CDT of a PSLG X with k segments and m polygons. Let l be the total number of distinct tetrahedra that exist in one or more of the triangulations $\mathcal{T}_0, \ldots, \mathcal{T}_m$ (i.e., the tetrahedra in the initial Delaunay triangulation and the tetrahedra created during polygon insertion operations). The following theorem shows that the total cost of polygon location is dominated by the cost of simply creating all those tetrahedra.

Theorem 16. *During incremental construction of the CDT of X (whether randomized or not), the total time for polygon location as described above is in $O(l)$.*

Proof. The time to locate a polygon $f \in \mathcal{X}$ is at worst proportional to the number of segments of f (as we must check the degree of each segment), which we call the *first cost* of f, plus the degree of its lowest-degree segment, which we call the *second cost* of f, plus the number of tetrahedra deleted when f is inserted, which we call the *third cost* of f, plus the sum of the degrees of any triangulation edges included in f that are not segments, which we call the *fourth cost* of f.

The total first cost of all the polygons is equal to the number of polygon-segment incidences in $\mathcal{X}$. Each such incidence induces a distinct triangle-edge incidence in the final triangulation $\mathcal{T}_m$. $\mathcal{T}_m$ has at most 12 triangle-edge incidences per tetrahedron, and there are at most l tetrahedra in $\mathcal{T}_m$. Therefore, the total first cost of all the polygons at most $12l$.

The total third cost of all the polygons is at most l, because a tetrahedron must be created before it is deleted, and a deleted tetrahedron is never created again.

Let E be the set containing every edge that exists in one or more of the triangulations $\mathcal{T}_0, \ldots, \mathcal{T}_m$. For $e \in E$, let d_e be the maximum degree of e over all the triangulations $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m$. Each tetrahedron has six edges, so $\sum_{e \in E} d_e \leq 6l$. Therefore, the total fourth cost of all the polygons at most $6l$.

Suppose that the segments in $\mathcal{X}$ are all in E (a necessary condition for the algorithm to succeed). Order the segments in $\mathcal{X}$ by their maximum degrees, from greatest to least. For all $i \in [1, k]$, let $\mathcal{X}_i \subseteq \mathcal{X}$ be the PLC induced by taking all the vertices in $\mathcal{X}$, the first i segments (with greatest maximum degrees), and the polygons in $\mathcal{X}$ whose segments are all among those i segments. By Lemma 12, $\mathcal{X}_i$ has fewer than $2i$ polygons. By a repetition of the same inductive reasoning employed in the proof of Theorem 11, the total second cost of all the polygons is less than $2 \sum_{e \in E} d_e \leq 12l$. □

11. Conclusions and Open Problems

Although this article emphasizes the time complexity of randomized incremental segment insertion, we draw some conclusions useful to programmers implementing CDT construction codes. First, when some segments cross very large numbers of edges, a faster segment insertion algorithm can make enough of a difference to justify its implementation. Second, there is a very simple segment location method that never compromises incremental insertion's asymptotic running time, with or without randomization. Third, it might be worthwhile to randomize the order in which the segments are inserted; compare the expected $O(n \log^2 k)$ upper bound on the number of structural changes with the $\Theta(kn)$ structural changes that can occur with a deterministic ordering. Fourth, the $O(n \log^2 k)$ upper bound is almost always too pessimistic in practice; circumstances in which this much work is required are difficult to devise and unlikely to occur in the real world. Fifth, and most importantly, if incremental segment insertion is implemented intelligently, it is fast enough; implementing a more complicated $O(n \log n)$-time CDT construction algorithm is unlikely to repay the effort.

Considering the $\Omega(n \log^2 k)$ lower bound on the expected number of structural changes that the randomized incremental segment insertion algorithm performs on some PSLGs, we wonder whether the algorithm could be faster if the segments were inserted in a biased order. For example, it is possible to choose two segments at random, determine which one crosses the fewest edges, and insert it in time proportional to the number of edges it crosses—without taking the time to count all the edges the other segment crosses. Does this procedure reduce the expected asymptotic number of structural changes?

Interestingly, the $O(n \log^2 k)$ upper bound on the number of structural changes does not rely on the constrained Delaunay property; the analysis applies however the cavities are retriangulated. Are there other algorithms for computing optimal triangulations that can be sped up by randomized incremental segment insertion?

Finally, how quickly can an algorithm insert a polygon into a three-dimensional CDT? There is no obvious reason to doubt that an algorithm exists that runs in expected time linear in the number of structural changes, but neither is there an obvious reason to be confident that one exists.

Acknowledgments

We thank Kevin Yi for introducing us to his work on CDT structural changes [1], Pankaj Agarwal and Ken Clarkson for discussions of the probabilistic methods behind it [13, 18], Peter Shor for pointing out the connection to the coupon collector's problem, and James Martin for introducing us to the Poissonization of that problem. We also thank James O'Brien for the inspiration of writing "This is hopeless. You will never figure it out" on the whiteboard while we worked out the lower bound.

References

[1] Pankaj K. Agarwal, Lars Arge, and Ke Yi. *I/O-Efficient Construction of Constrained Delaunay Triangulations*. Unpublished manuscript; http://www.cse.ust.hk/~yike/cdt/paper.pdf, 2005. Most of this paper appears in the Proceedings of the Thirteenth European Symposium on Algorithms, pages 355–366, October 2005, but the published version omits the analysis of the number of structural changes performed by randomized incremental segment insertion.

[2] Pankaj K. Agarwal and Micha Sharir. *Arrangements and their Applications*. Handbook of Computational Geometry (Amsterdam, The Netherlands), pages 49–119. Elsevier Science, 2000.

[3] Nina Amenta, Sunghee Choi, and Günter Rote. *Incremental Constructions con BRIO*. Proceedings of the Nineteenth Annual Symposium on Computational Geometry (San Diego, California), pages 211–219. Association for Computing Machinery, June 2003.

[4] Marc Vigo Anglada. *An Improved Incremental Algorithm for Constructing Restricted Delaunay Triangulations*. Computers and Graphics **21**(2):215–223, March 1997.

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

[6] Kevin Q. Brown. *Voronoi Diagrams from Convex Hulls*. Information Processing Letters **9**(5):223–228, December 1979.

[7] Mark Brown, Erol A. Peköz, and Sheldon M. Ross. *Coupon Collecting*. Probability in the Engineering and Informational Sciences **22**(2):221–229, March 2008.

[8] Bernard Chazelle. *Triangulating a Simple Polygon in Linear Time*. Discrete & Computational Geometry **6**:485–524, December 1991.

[9] Siu-Wing Cheng, Tamal Krishna Dey, and Jonathan Richard Shewchuk. *Delaunay Mesh Generation*. CRC Press, Boca Raton, Florida, December 2012.

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

[11] ———. *Building Voronoi Diagrams for Convex Polygons in Linear Expected Time*. Technical Report PCS-TR90-147, Department of Mathematics and Computer Science, Dartmouth College, 1990.

[12] Francis Chin and Cao An Wang. *Finding the Constrained Delaunay Triangulation and Constrained Voronoi Diagram of a Simple Polygon in Linear Time*. SIAM Journal on Computing **28**:471–486, 1998.

[13] Kenneth L. Clarkson. *New Applications of Random Sampling in Computational Geometry*. Discrete & Computational Geometry **2**(1):195–222, December 1987.

[14] Boris Nikolaevich Delaunay. *Sur la Sphère Vide*. Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk **7**:793–800, 1934.

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

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

[17] Leonidas J. Guibas and Jorge Stolfi. *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*. ACM Transactions on Graphics **4**(2):74–123, April 1985.

[18] David Haussler and Emo Welzl. *ϵ-Nets and Simplex Range Queries*. Discrete & Computational Geometry **2**(1):127–151, December 1987.

[19] François Hermeline. *Une Methode Automatique de Maillage en Dimension n*. Ph.D. thesis, Université Pierre et Marie Curie, Paris, France, 1980.

[20] ———. *Triangulation Automatique d'un Polyèdre en Dimension N*. RAIRO Analyse Numérique **16**(3):211–242, 1982.

[21] Mark Howison. *CAD Tools for Creating Space-Filling 3D Escher Tiles*. Master's thesis, University of California at Berkeley, Berkeley, California, May 2009. Available as Technical Report UCB/EECS-2009-56.

[22] R. Kaas and J. M. Buhrman. *Mean, Median and Mode in Binomial Distributions*. Statistica Neerlandica **34**(1):13–18, March 1980.

[23] Thomas C. Kao and David M. Mount. *Incremental Construction and Dynamic Maintenance of Constrained Delaunay Triangulations*. Proceedings of the Fourth Canadian Conference on Computational Geometry (St. John's, Newfoundland), pages 170–175, August 1992.

[24] Rolf Klein and Andrzej Lingas. *A Linear-Time Randomized Algorithm for the Bounded Voronoi Diagram of a Simple Polygon*. Proceedings of the Ninth Annual Symposium on Computational Geometry (San Diego, California), pages 124–132. Association for Computing Machinery, May 1993.

[25] ———. *A Note on Generalizations of Chew's Algorithm for the Voronoi Diagram of a Convex Polygon*. Proceedings of the Fifth Canadian Conference on Computational Geometry (Waterloo, Ontario, Canada), pages 370–374, August 1993.

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

[27] Der-Tsai Lee and Bruce J. Schachter. *Two Algorithms for Constructing a Delaunay Triangulation*. International Journal of Computer and Information Sciences **9**(3):219–242, 1980.

[28] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel J. Walkington, and Han Wang. *Control Volume Meshes Using Sphere Packing: Generation, Refinement and Coarsening*. Proceedings of the 5th International Meshing Roundtable (Pittsburgh, Pennsylvania), pages 47–61, October 1996.

[29] Richard Pollack and Marie-Françoise Roy. *On the Number of Cells Defined by a Set of Polynomials*. Comptes Rendus de l'Académie des Sciences, Série I, Mathématique **316**(6):573–577, 1993.

[30] Jim Ruppert and Raimund Seidel. *On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra*. Discrete & Computational Geometry **7**(3):227–253, 1992.

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

[32] Raimund Seidel. *Voronoi Diagrams in Higher Dimensions*. Diplomarbeit, Institut für Informationsverarbeitung, Technische Universität Graz, Graz, Austria, 1982.

[33] ———. *Constrained Delaunay Triangulations and Voronoi Diagrams with Obstacles*. 1978–1988 Ten Years IIG (H. S. Poingratz and W. Schinnerl, editors), pages 178–191. Institut für Informationsverarbeitung, Technische Universität Graz, Graz, Austria, 1988.

[34] ———. *Backwards Analysis of Randomized Geometric Algorithms*. New Trends in Discrete and Computational Geometry (János Pach, editor), Algorithms and Combinatorics, volume 10, pages 37–67. Springer-Verlag, Berlin, 1993.

[35] Michael Ian Shamos and Dan Hoey. *Closest-Point Problems*. 16th Annual Symposium on Foundations of Computer Science (Berkeley, California), pages 151–162, October 1975.

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

[37] ———. *Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery*. Proceedings of the 11th International Meshing Roundtable (Ithaca, New York), pages 193–204. Sandia National Laboratories, September 2002.

[38] ———. *Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips*. Proceedings of the Nineteenth Annual Symposium on Computational Geometry (San Diego, California), pages 181–190. Association for Computing Machinery, June 2003.

[39] ———. *General-Dimensional Constrained Delaunay Triangulations and Constrained Regular Triangulations I: Combinatorial Properties*. Discrete & Computational Geometry **39**(1–3):580–637, March 2008.

[40] Jonathan Richard Shewchuk and Hang Si. *Higher-Quality Tetrahedral Mesh Generation for Domains with Small Angles by Constrained Delaunay Refinement*. Proceedings of the Thirtieth Annual Symposium on Computational Geometry (Kyoto, Japan), pages 290–299. Association for Computing Machinery, June 2014.

[41] Hang Si. *Constrained Delaunay Tetrahedral Mesh Generation and Refinement*. Finite Elements in Analysis and Design **46**:33–46, January–February 2010.

[42] Hang Si and Klaus Gärtner. *Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations*. Proceedings of the Fourteenth International Meshing Roundtable (San Diego, California) (Byron W. Hanks, editor), pages 147–163, September 2005.

[43] Hang Si and Jonathan Richard Shewchuk. *Incrementally Constructing and Updating Constrained Delaunay Tetrahedralizations with Finite Precision Coordinates*. Proceedings of the 21st International Meshing Roundtable (San Jose, California), pages 173–190, October 2012.

[44] Jakob Steiner. *Einige Gesetze über die Theilung der Ebene und des Raumes*. Journal für die Reine und Angewandte Mathematik **1**:349–364, 1826.

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