A Randomized Algorithm for Triangulating a Simple Polygon in Linear Time

Edgar A. Ramos *

December 14, 2000

Abstract

We describe a randomized algorithm by Amato *et al.* (2000) for computing the trapezoidal decomposition of a simple polygon. Its expected running time is linear in the size of the polygon. By a well-known and simple linear time reduction, this implies a linear time algorithm for triangulating a simple polygon. The algorithm is considerably simpler than Chazelle's (1991) optimal deterministic algorithm, and it can be viewed as a of this algorithm and a non-optimal randomized algorithms due to Clarkson *et al.* (1991) and to Seidel (1991). An important new idea is the use of random sampling on subchains of the initial polygonal chain. As in Chazelle's algorithm, it is indispensable to include a "bottom-up" preprocessing phase, in addition to the actual "top-down" construction.

*Max-Planck-Institut für Informatik, Saarbrücken, Germany. E-mail: ramos@mpi-sb.mpg.de

Contents

0 Overview

Polygon triangulation is a classic problem in computational geometry and one of the first problems studied in the field. There are several other problems dealing with polygons that have efficient solutions that begin with polygon triangulation as a preprocessing step. Thus, there has been considerable interest in finding efficient algorithms for this problem. After initial algorithms with a running time $\Theta(n \log n)$, where n is the number of edges in the polygon, a sequence of works improved this bound: Tarjan and Van Wyk [18] and Kirkpatrick *et al.* [13] obtained deterministic algorithms with running time $\Theta(n \log \log n)$; using randomization, Clarkson *et al.* [9], Clarkson *et al.* [6, 7], and Seidel [17] obtained randomized algorithms with expected running time $\Theta(n \log^* n)$. Finally, Chazelle [3] showed that it is possible to triangulate a polygon in linear time. Unfortunately, Chazelle's linear time algorithm is quite complex, and so remained open the question of whether randomization could be used to obtain a simpler algorithm. We describe here a recent randomized algorithm by Amato *et al.* [2] that runs in expected linear time. The algorithm is considerably simpler than Chazelle's.

The new algorithm, as all the others faster than $\Theta(n \log n)$, actually compute the *visibility map* or *trapezoidation*. See Sec.1. A triangulation can be derived from the trapezoidation with a simple linear time algorithm. The new trapezoidation algorithm follows an approach similar to that of the non-optimal randomized algorithms by Clarkson *et al.* [6, 7] and Seidel [17]. Their approach is to compute the trapezoidation of a successively finer sample from P using an algorithm for arbitrary edges (thus with superlinear running time), in $O(\log^* n)$ rounds. See Secs. 2 and 3. The fact that the edges come from a simple polygonal chain is used to efficiently perform the computation of the *conflict lists* of the trapezoidation of the sample. This is done in each round, by *walking* along the original polygonal chain in the trapezoidation. Unfortunately, an approach that maintains the lists of *edge conflicts* for the trapezoidation of the sample is doomed to spend at least linear time per round. To avoid this, the new algorithm first decomposes the input polygonal chain into smaller subchains, and then takes a sample from the resulting set of subchains. See Sec. 6. Then, it constructs a decomposition with a number of regions that is proportional to the number of subchains involved (rather than edges), and such that the regions have *bounded complexity* (here, one counts the number of subchains that determine the region, rather than the edges). See Sec. 4. The latter requirement originates in the need to be able to derive appropriate bounds for the sizes of the conflict lists, and in the need to have a decomposition that can be traversed efficiently as one follows the polygonal chain. This concept, termed *conformality*, also appears in Chazelle's algorithm. However, here the problem is simpler as one deal with as small sample compared to the input polygonal chain. The algorithm maintains *chain-conflict lists*, rather than edge conflict lists, and to compute them, it traverses the decomposition along the polygonal chain. See Sec. 5. In order to perform this traversal efficiently one needs, in addition to the conformality, a data structure for each subchain that answers intersection queries between a vertical edge, called a *portal*, and the subchain. See Sec. 7. As in Chazelle's algorithm, this leads to a preprocessing phase that constructs these data structures prior to the actual construction phase. These phases proceed bottom-up and top-down respectively. See Secs. 8 and 9. Randomization also plays an important role in the preprocessing phase. A final difficulty is the proof of appropriate *sampling bounds* for the sizes of the *chain-conflict* lists of the conformal decomposition: such bounds are known under *locality* or *monotonicity* properties that the conformal decomposition computed by the algorithm does not satisfy [1, 8, 10, 14, 16]. Nevertheless, some weaker bounds can be proved using the fact that the regions in the decomposition are chosen from a relatively small (not much larger than the decomposition itself) "pool" of candidates that satisfy a locality property. See Sec. 10. The bounds obtained suffice to verify that the expected running time of the algorithm is linear. See Sec. 11.

Part I
Preliminaries

1 Trapezoidation and Triangulation

Let S be a set of n *edges* (line segments) in the plane with nonintersecting interiors (common vertices are allowed). We assume that no two vertices have the same horizontal coordinate; this can be simulated symbolically through lexicographic ordering. For $R \subseteq S$, the *trapezoidation* of R is obtained as follows: from each vertex, rays are drawn vertically up and down until each hits another edge in S (because of the generic position assumption, they do not hit a vertex). The edges in S together with the vertical *visibility* rays decompose the plane into connected regions that are called *trapezoids* (though they degenerate into triangles and can also be unbounded). Each trapezoid is bounded by at most two edges in S (one above and one below) and by at most four vertical rays (at most two on the left and at most two on the right). Let $\mathcal{T}(R)$ denote the set of trapezoids in the trapezoidation of R. See Fig. 1.

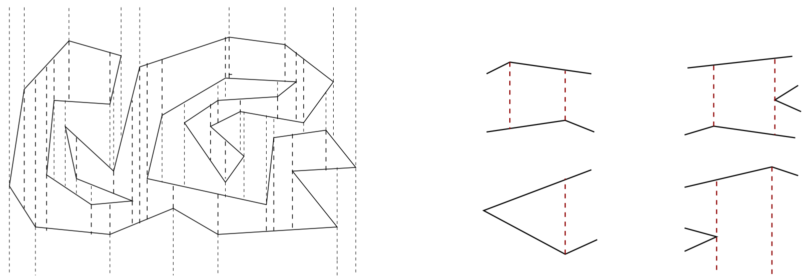


Figure 1: Trapezoidation of a simple polygon and some possible trapezoidal shapes.

We are interested here in the case in which S is the set of edges of a *simple polygon,* that is, a closed non-selfintersecting polygonal (*i.e.*, piecewise linear) curve. It is well-known that for a simple polygon, a simple procedure can in linear time derive a triangulation from its trapezoidation. This is illustrated in Fig. 2. (Actually, one can extract more triangulation edges from the trapezoidation.)

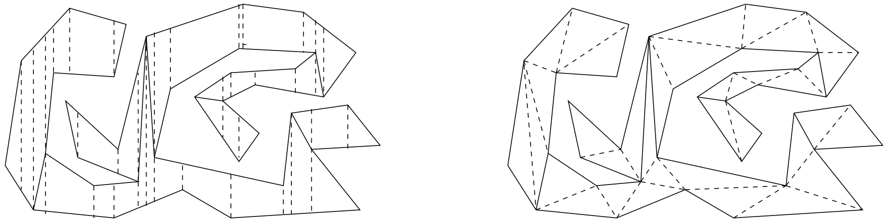


Figure 2: Trapezoidation and triangulation of a simple polygon. The visibility rays in the interior of the polygon (longer dashes) are used to derive a triangulation. A diagonal from each *reflex* vertex to the opposite vertex in the corresponding trapezoid (solid diagonals) decomposes the polygon into *monotone* polygons. Each monotone polygon can then be easily triangulated in linear time (dashed diagonals).

The algorithms we describe construct a representation of the trapezoidation that allows an efficient traversal. More specifically, the algorithms maintain its *adjacency graph* $\mathcal{G}(R) = \mathcal{G}(\mathcal{T}(R))$: its *nodes* are the trapezoids, and two of them are connected with an *arc* if they are adjacent thorugh a vertical ray. Thus, each *node-trapezoid* has pointers to its neighbors on its left and its right. Furthermore, a trapezoid also has pointers to the edges above and below (if any). From now on, we assume that the algorithms that compute the trapezoidation $\mathcal{T}(R)$, also construct $\mathcal{G}(R)$.

2 Basic Incremental Algorithm

First, we review a simple randomized incremental algorithm for computing the trapezoidation of a set S of nonintersecting edges (not necessarily belonging to a simple polygon). This algorithm is the basis for the polygon triangulation algorithm in [17] and for the one here as well. It is a simplification of previous algorithms by Clarkson and Shor [8] and Mulmuley [15] for the case in which the edges are nonintersecting. Consider an ordering $s_1, s_2, \ldots, s_n$ of the edges in S, and let $S_j = \{s_1, \ldots, s_j\}$. The algorithm constructs $\mathcal{T}(S)$ incrementally, that is, it successively "inserts" s_j into $\mathcal{T}(S_{j-1})$ until $\mathcal{T}(S)$ is obtained. For $\mathcal{T}(S_{j-1})$, in addition to $\mathcal{G}(S_{j-1})$, it maintains a data structure $\mathcal{Q}(S_{j-1})$ for point location in $\mathcal{T}(S_{j-1})$. $\mathcal{Q}(S_{j-1})$ is a directed acyclic graph with a source node corresponding to the plane and a leaf corresponding to each trapezoid in $\mathcal{T}(S_{j-1})$. Each nonleaf node has degree two and is associated either with an edge in which case its two children correspond to above and below the edge, or with an endpoint of an edge in which case the two children correspond to left and right of the vertical line through that vertex. See Fig. 3.

The edge s_j is added to $\mathcal{T}(S_{j-1})$ to obtain $\mathcal{T}(S_j)$ in two steps (see Fig. 4):

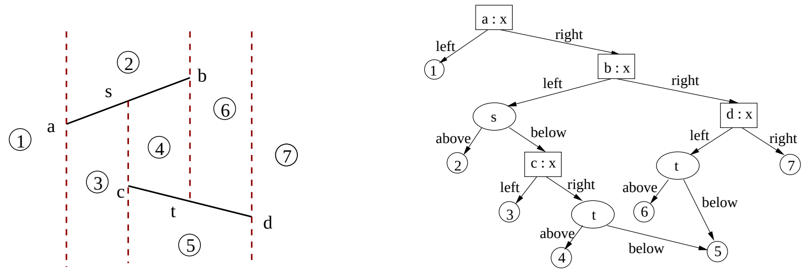


Figure 3: The trapezoidation of $\{s, t\}$ and a corresponding point location data structure.

(i) **Insert endpoints:** Locate each endpoint of s_j in $\mathcal{Q}(S_{j-1})$, update $\mathcal{Q}(S_{j-1})$ by making the two resulting leaves into internal nodes with queries corresponding to the left and right trapezoids, and update $\mathcal{G}(S_{j-1})$ to include the new trapezoids; and

(ii) **Trace edge:** From one of the endpoints, the edge s_j is traced using $\mathcal{G}(S_{j-1})$, thus determining the trapezoids that *conflict* (intersected by) s_j; these are removed from, and the new resulting trapezoids added to $\mathcal{G}(S_{j-1})$, and $\mathcal{Q}(S_{j-1})$ is updated accordingly, to finally obtain $\mathcal{G}(S_i)$ and $\mathcal{Q}(S_j)$.

If the ordering $s_1, \ldots, s_n$ is a random permutation of the edges, then the expected time for completing the construction is $O(n \log n)$. The argument is as follows: In $\mathcal{T}(S_j)$, the average number of rays of rays abutting upon an edge is at most $4j$, so the expected number of trapezoids traced in inserting s_i is at most 4. Let Δ_j be the trapezoid in $\mathcal{T}(S_j)$ containing a query point q, then the probaility that Δ_{j-1} is not the same as Δ_j is at most $5/j$ ($4/j$ correspond to the 4 edges determining Δ_j, the extra $1/j$ corresponds to a trickier case, see [17]). Thus, the expected time needed to insert s_j is $O(\log n)$ and the total expected timed needed to construct $\mathcal{G}(S)$ and $\mathcal{Q}(S)$ is $O(n \log n)$. This is optimal for arbitrary nonintersecting edges (not part of a simple polygon), and even for a polygon, examples can be constructed where $\Omega(n \log n)$ expected time is needed. We will refer to this algorithm as `BasicInc`.

3 Seidel's Algorithm

It is instructive first to outline the non-optimal randomized algorithm of Seidel [17]. This is related to previous ones in [6] and [7]. Let ℓ_0 be a simple polygonal chain and let S be the set of edges in ℓ_0. For $R \subseteq S$ and $\Delta \in \mathcal{T}(R)$, let $S_{|\Delta}$ denote the *conflict list* of Δ, that is, the set of those edges in S that intersect (the interior of) Δ, and let $n_\Delta = |S_{|\Delta}|$. Let $r_i = \lceil n / \log^{(i)} n \rceil$.[1] As before, let $S_j = \{s_1, \ldots, s_j\}$, and then $R_i = S_{r_i}$. The algorithm constructs the trapezoidation in $k = O(\log^* n)$ rounds: In the ith round, starting with $\mathcal{T}(R_{i-1})$, the

[1]Recall that the ith iterated logarithm is defined by $\log^{(0)} n = n$ and for $i > 0$, $\log^{(i)} n = \log(\log^{(i-1)} n)$, and that $\log^* n$ denotes the largest integer l so that $\log^{(l)} n \geq 1$.

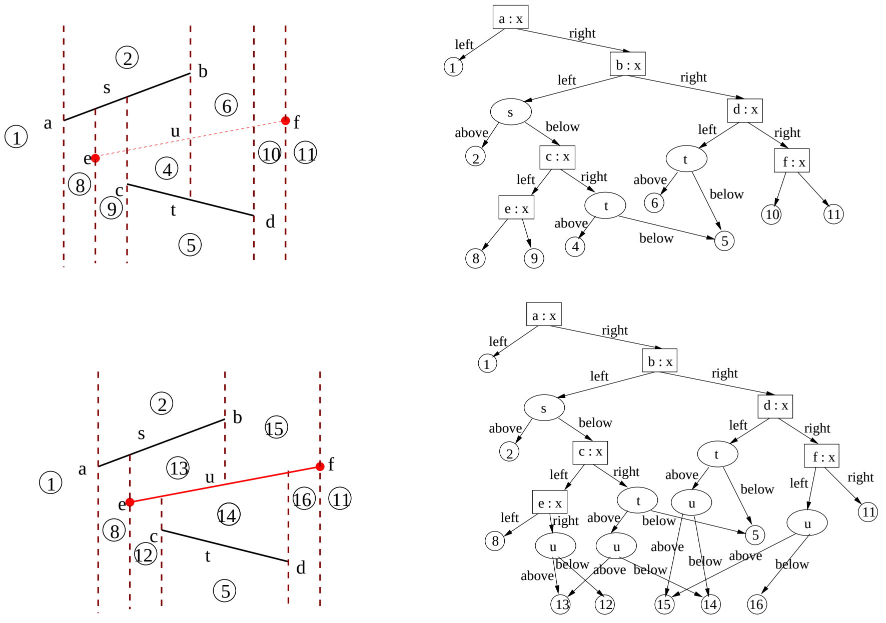


Figure 4: Insertion of edge u, after Step (i) on top and after Step (ii) on bottom.

edges in $R_i - R_{i-1}$ are inserted one by one, as described in the previous subsection, to obtain $\mathcal{T}(R_i)$. To make the algorithm more efficient than the basic incremental algorithm, the algorithm also computes for each $\Delta \in \mathcal{T}(R_i)$, the conflict list $S_{|\Delta}$ by starting at an endpoint of ℓ_0 and walking along it in $\mathcal{T}(R_i)$. This takes time proportional to n plus the total number of conflicts, that is, $\sum_{\Delta \in \mathcal{T}(R_i)} |S_{|\Delta}|$. The procedure is summarized in Fig. 5.

Sampling Bound and Analysis

The algorithm can be analyzed with the use of the following *sampling bound*. Let R be an *r-sample* from S, that is, R is taken with equal probability among all the subsets of S of size r. We also say that R is a p-sample where $p = r/n$.[2]

$$\mathbf{E}\left[\sum_{\Delta \in \mathcal{T}(R)} n_\Delta\right] = O(r). \tag{1}$$

See [8, 16] or Section 10. Noting that R_i is an r_i-sample from S, Eqn. (1) implies that the expected total size of the conflict lists is $O(n)$. This implies a bound of $O(n)$ for the expected

[2]Note that p is the probability of selecting each element in R, but not independently. In [2], a different sampling model is used in which a p-sample $R \subseteq S$ is formed by taking each $s \in S$ into R independently with probability p.

`Seidel's-Trapezoidation` (i-th round)

Input: $\mathcal{T}(R_{i-1})$, $\mathcal{Q}(R_{i-1})$ and conflicts with S
Output: $\mathcal{T}(R_i)$, $\mathcal{Q}(R_i)$ and conflicts with S

1. Starting with $\mathcal{T}(R_{i-1})$ and $\mathcal{Q}(R_{i-1})$, insert the edges in $R_i - R_{i-1}$, in a random order, to obtain $\mathcal{T}(R_i)$ and $\mathcal{Q}(R_i)$, following `BasicInc`.
2. "Walk" along ℓ_0 in $\mathcal{T}(R_i)$ to obtain $S_{|\Delta}$ for all $\Delta \in \mathcal{T}(R_i)$

Figure 5: Seidel's algorithm.

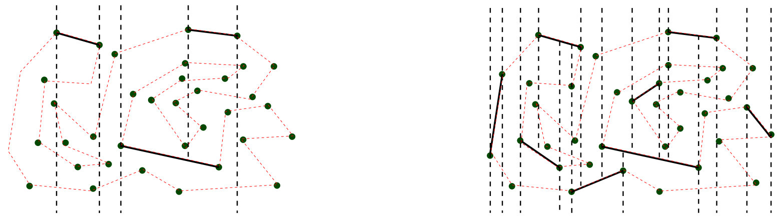


Figure 6: Two rounds of Seidel's algorithm. At the end of a round, each trapezoid has a list of the polygon vertices that it contains.

time required by all steps in a round, except the point location in Step 1. The argument for the basic algorithm above indicates that the expected time for a point location in the i-th round is at most proportional to

$$\frac{1}{r_{i-1}} + \frac{1}{r_{i-1}+2} + \cdots + \frac{1}{r_i} = O\left(\log\left(\frac{r_i}{r_{i-1}}\right)\right) = O\left(\log\left(\frac{\log^{(i-1)} n}{\log^{(i)} n}\right)\right) = O(\log^{(i)} n).$$

The number of point locations is $r_i - r_{i-1} \leq r_i = n/\log^{(i)} n$. So, the cost of the point locations in each round $O(n)$. Hence the total expected running time is $O(n \log^* n)$.

Part II
The Linear-Time Algorithm

The algorithm can be viewed as a refinement of Seidel's algorithm, in which sampling is applied to subchains of the original chain ℓ_0 rather than to edges. More precisely, the chain ℓ_0 is divided into a set L of subchains of length λ, and then an r-sample $K \subseteq L$. Then, we need to develop an appropriate decomposition of the plane induced by these subchains that is *conformal* and *granular*. This decomposition, $\widetilde{\mathcal{T}}(K)$, is called *chain-trapezoidation* and each of its regions $\widetilde{\Delta}$ is called a *chain-trapezoid*. For each chain-trapezoid $\widetilde{\Delta}$ in $\widetilde{\mathcal{T}}(K)$, let $L_{|\widetilde{\Delta}}$ denote the set of subchains in L that intersect (the interior of) $\widetilde{\Delta}$. Let $\widetilde{n} = |L|$, $\widetilde{n}_{\widetilde{\Delta}} = |L_{|\widetilde{\Delta}}|$. In analogy with Eqn. (1), one would conjecture a bound

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} \widetilde{n}_{\widetilde{\Delta}}\right] = O(\widetilde{n}). \tag{2}$$

In particular, the expected total chain-conflict size would be $O(\widetilde{n}) = O(n/\lambda)$ and it could be made appropriately sublinear by choosing λ sufficiently large. Though we cannot prove such a bound, in Section 10 we prove a weaker but sufficient one. This is a first step towards obtaining a linear time algorithm.

One then needs a way to compute these chain-conflict lists in a way that avoids "walking" over every edge of ℓ_0 (as in the non-optimal algorithm). This is done by performing a breadth-first search of the adjacency graph of $\widetilde{\mathcal{T}}(K)$, that uses a *portal-chain intersection* data structure to determine whether the chain crosses a portal and so reaches into the adjacent chain-trapezoid. We call this *hopping*.

The algorithm works on a sequence of partitions of ℓ_0 into successively smaller subchains $L_0, L_1, L_2, \ldots$, each one sampled with a succesively increasing probability $p_1, p_2, p_3, \ldots$. In this way, the sequence of samples $K_0, K_1, K_2, \ldots$ starts with a coarse refinement sampled lightly and ends with a fine refinement sampled heavily (the whole chain ℓ_0).

The construction of a portal-query intersection data structure forces a two-phase computation pattern. The *up-phase* considers the *refinement tree* induced by the samples K_i on the individual subchains $\ell \in L_{i-1}$, that is, $K_i^\ell = K_i \cap L_i^\ell$ where L_i^ℓ is the refinement of ℓ in the $(i-1)$-st level. The *up-phase* starts at the leaves of the refinement tree and proceeds in a bottom-up manner constructing, for each $\ell \in L_{i-1}$, $\widetilde{\mathcal{T}}(K_i^\ell)$ and its chain-conflict lists with respect to L_i^ℓ. The *down-phase* starts at the top level of the gradation and proceeds in a top-down manner constructing $\widetilde{\mathcal{T}}(K_i)$ and its chain-conflict lists with respect to L_i (it actually considers a slightly bigger set K_i^+). The resulting construction follows a pattern similar to the one followed by Chazelle's: in the up-phase the algorithm obtains a successively more global picture but with less detail, and then in the down-phase the algorithm successively refines the global picture until the desired trapezoidation is obtained.

4 Chain-Trapezoidation

The algorithm applies sampling to subchains of the original polygonal chain, rather than individual edges. To effectively deal with such samples, we need a method for defining subproblems of constant descriptive complexity. Consider a set L of $\widetilde{n}$ chains with a corresponding set S of n edges. Let $K \subseteq L$ be a subset of chains of L and let $R \subseteq S$ be the corresponding set of edges. For convenience, we write $\mathcal{T}(K)$ and $\mathcal{G}(K)$ to denote $\mathcal{T}(R)$ and $\mathcal{G}(R)$ respectively. This planar subdivision has $O(|R|)$ faces and each face has at most 2 edges and 4 vertical rays on its boundary. While $\mathcal{T}(K)$ has size $\Theta(|R|)$, for the algorithm to run in linear time, a planar subdivision with $O(|K|)$ regions, each of which is *conformal*: each region is bounded by portions of $O(1)$ subchains in K and at most $O(1)$ vertical rays

determined by their vertices. We obtain this subdivision by selecting certain rays of $\mathcal{T}(K)$. The *candidate rays* for this selection are the *ray-pairs* (one ray upward and one ray downward) incident to a *reflex* vertex of a chain (a vertex without incident polygonal edges either on its left or on its right side).

`conformal`$(K, \mathcal{T}(K))$

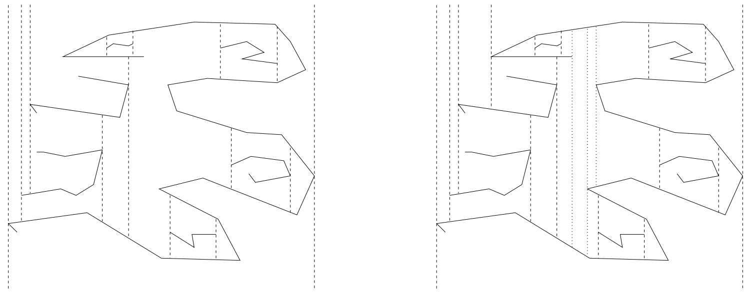
Input: A set of chains K and the trapezoidation $\mathcal{T}(K)$ of its edges.
Output: Conformal decomposition $\widetilde{\mathcal{T}}(K)$, and its adjacency graph.

1. Select the extreme ray-pairs of each chain. Let $\overline{T}$ be the planar subdivision (which is simply connected) induced by the chains in K and these selected ray-pairs. (See Fig. 8(a).)
2. For each face f of $\overline{T}$ and a candidate ray-pair ρ in f, select ρ if each of the three faces in which ρ splits f has a ray selected in Step 1 on its boundary.
3. Construct $\widetilde{\mathcal{T}}(K)$, the decomposition induced by all the selected ray-pairs, and its adjacency graph. (See Fig. 8(b).)

Figure 7: Conformal decomposition procedure.

The procedure is summarized in Fig. 7. It selects rays in two steps. In Step 1, the procedure selects all the extreme ray-pairs, that is, those originating from the leftmost and rightmost vertices of each chain. We claim that the faces of the resulting subdivision $\overline{T}$ are simply connected (see Fig. 8(a)). This is because if one such face f has a hole, then the ray-pairs from the leftmost and rightmost extreme vertices of chains inside the hole would contradict the existence of the hole. In Step 2, the procedure selects other candidate ray-pairs in a "non-local" manner, so as to obtain a subdivision whose faces are conformal (see Fig. 8(b)). Let $\overline{T}$ be the planar subdivision (which is simply connected) induced by the chains in K and the ray-pairs selected in Step 1. For each face f of $\overline{T}$, in Step 2, the procedure selects a candidate ray-pair ρ in f if each of the three faces in which ρ splits f has a ray selected in Step 1 on its boundary. The selection is performed through simple traversals of $\mathcal{T}(K)$ that use time linear in its size.

For completeness, we elaborate further on this. For a face f, let τ_f be the dual tree of the portion of $\mathcal{T}(K)$ in f. Root τ_f at an arbitrary node-trapezoid that has at most two non-selected rays on its boundary (this is so that each node-trapezoid in the rooted tree, even the root, has at most two children). The rooting determines for each node-trapezoid a corresponding "region below" it that includes the trapezoid itself, and a "region above" it (that excludes the trapezoid). Perform a depth first tree traversal that *marks* with blue a node-trapezoid if its region below has a ray selected in Step 1 on its boundary: After visiting the children of the trapezoid, this is marked with blue if either of its children was marked with blue or if it has a selected ray on its boundary; a leaf is marked with blue. Another tree traversal marks with red a node-trapezoid (without erasing the blue marks) if the region above has a selected ray on its boundary: Mark with red a node-trapezoid if either its parent



(a) (b)

Figure 8: Conformal decomposition example: (a) Subdivision after Step 1, and (b) subdivision after Step 2 (new edges are dotted).

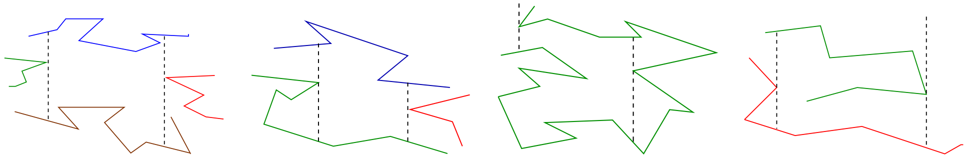


Figure 9: Some example chain-trapezoids. A chain-trapezoid is determined by up to four distinct chains, but can be determined by a single one as in the third example. Also, because ray-pairs from endpoints are not necessarily selected, the situation in the fourth example can occur.

is marked with red or its sibling is marked with blue; the root is marked with red if it has a selected ray on its boundary. With these blue and red marks, for each candidate ray-pair ρ, the unique trapezoid Δ_ρ that has both rays of ρ on its boundary can determine whether ρ must be selected: ρ is selected if Δ_ρ has a red mark and both of its children have a blue mark.

This algorithm clearly runs in $O(|R|)$ time: All steps can be performed by simple traversals of $\mathcal{T}(K)$ using time linear in its size. Moreover, the total number of selected ray-pairs is $O(|K|)$: It is clear that $\overline{\mathcal{T}}$ has $O(|K|)$ selected ray-pairs; furthermore, since we select only ray-pairs for which each of the corresponding three regions has a ray selected in Step 1 on its boundary, at most $O(|K|)$ additional rays are selected in Step 2. Note that in the resulting subdivision, each face is bounded by at most two chains and at most two ray-pairs or single rays (the portion of a chain bounding one of these faces does not need to be monotone; one of the bounding chains itself can also determine one or both bounding ray-pairs). See Fig. 9. We refer to the selected (single) rays as *portals*, to the conformal faces as *chain-trapezoids* and to the resulting decomposition, denoted $\widetilde{\mathcal{T}}(K)$, as the *chain-trapezoidation*. Again, abusing notation, $\widetilde{\mathcal{T}}(K)$ also denotes the collection of conformal faces. Therefore, we have the following:

Lemma 4.1 *Let K be a set of chains and let $R \subseteq S$ be the corresponding set of edges, and suppose that we are given a planar subdivision representation of the trapezoidation $\mathcal{T}(R)$ of the edges in R. Then procedure* `conformal` *constructs in $O(|R|)$ time a decomposition, the chain-trapezoidation $\widetilde{\mathcal{T}}(K)$, that has size $O(|K|)$ and is conformal.*

5 Hopping Along a Chain

The basic operation is determining for a subchain ℓ the chain-trapezoids in a chain-trapezoidation $\widetilde{\mathcal{T}}$ with which it conflicts. The computation makes use of a *portal-chain intersection* data structure $\mathcal{D}(\ell)$ that for a query portal ρ decides whether ρ intersects ℓ. Suppose that we know a chain trapezoid $\widetilde{\Delta}_0 \in \widetilde{\mathcal{T}}$ that contains the an endpoint e of ℓ. Note that the chain-trapezoids in $\widetilde{\mathcal{T}}$ that conflict with ℓ are connected. Thus, we perform a breadth-first-search traversal of the adjacency graph of $\widetilde{\mathcal{T}}$, starting with $\widetilde{\Delta}_0$. When a chain-trapezoid $\widetilde{\Delta} \in \widetilde{\mathcal{T}}$ is visited, it is labeled as a *conflict* and each of the portals that separates $\widetilde{\Delta}$ from an unvisited chain-trapezoid $\widetilde{\Delta}' \in \widetilde{\mathcal{T}}$ is tested for conflict with ℓ using $\mathcal{D}(\ell)$. If ℓ conflicts with the portal, the traversal visits $\widetilde{\Delta}'$. Note that ℓ can zig-zag arbitrarily within the set of chain-trapezoids that it intersects. See Fig. 10.

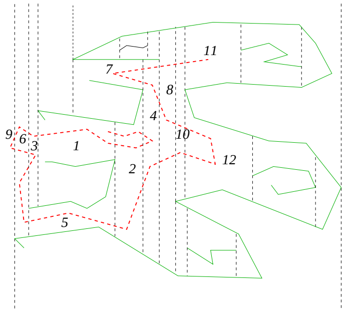


Figure 10: Computing the chain-conflicts for a chain. Note that only one conflict per portal is found, and that the order the conflicts are discovered (indicated by numbers) does not necessarily reflect their occurrences on the chain. The portals not queried are shown lighter.

This procedure performs $O(1)$ portal-chain conflict queries per conflict actually found. The location of the first endpoint e is given by the location of the second endpoint of the preceding chain ℓ' (known already if $\ell' \in K_i^+$). The location of ℓ's second endpoint e' can be determined by performing a point location query for each chain-trapezoid found to be in conflict with ℓ. This is necessary since the conflicts were computed not by a linear scan of ℓ, but rather by "hopping" between portals.

6 Gradation of Subchains

The sampling in the algorithm is performed on sets of subchains of the original polygonal chain obtained by refinement as follows. Let ℓ_0 be the initial simple polygonal chain and n be its length. We fix a function g that will determine the branching factor in the *refinement tree*. We decompose ℓ_0 into collections L_i of subchains of length λ_i, $i = 0, \ldots, k$, starting with $L_0 = \{\ell_0\}$ and $\lambda_0 = n$ and with L_i, $i > 1$, obtained by decomposing each chain $\ell \in L_{i-1}$ of into a set L_i^ℓ of subchains each with size in the range $[\lambda_i, 2\lambda_i)$, where $\lambda_i = g(\lambda_{i-1})$. The refinement proceeds until the subchains have length smaller than certain fixed constant. The depth of the tree depends on the choice of g, and can range between $\Theta(\log^* n)$ and $\Theta(\log n / \log\log n)$. Thus, the set of subchains in the i-th *refinement level* is $L_i = \bigcup_{\ell \in L_{i-1}} L_i^\ell$. We denote the total number of subchains in L_i by $\widetilde{n}_i = |L_i| = n/\lambda_i$.

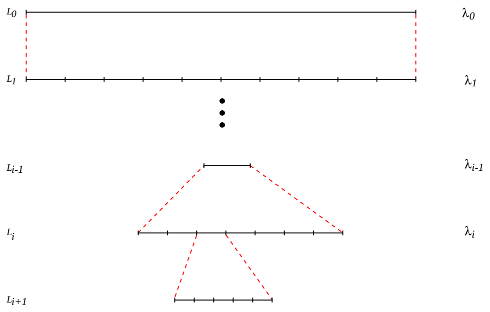


Figure 11: Gradation of subchains.

The algorithm simplifies the problem by taking a random sample K_i of subchains from L_i of a size such that one can afford to compute the trapezoidation of K_i using a sub-optimal algorithm. We fix a function h that will determine how the *sampling probability* depends on the subchain length. Specifically, for each $i \geq 1$, we choose a global probability $p_i = h(\lambda_{i-1})$, and let K_i be a p_i-sample from L_i. We call this sequence $K_0, K_1, K_2, \ldots$, a *gradation* of ℓ_0. Though a wide range of choices for g is possible, we fix one that gives $k = O(\log^* n)$:

$$\lambda_i = \log^2 \lambda_{i-1}, \qquad p_i = 1/\log^3 \lambda_{i-1}.$$

In the i-th round, it is more convenient to deal with the set of subchains K_i^+ that consists of the subchains in K_i and the subchains in L_i contained in all the previous samples K_j, $j < i$. That is, $K_0^+ = K_0$ and for $i \geq 1$,

$$K_i^+ = K_i \cup \bigcup_{\ell \in K_{i-1}^+} L_i^\ell.$$

Note that the number of the latter subchains is

$$\sum_{j<i} \widetilde{n}_j \cdot p_j \cdot \frac{\lambda_j}{\lambda_i} = \widetilde{n}_i \cdot \sum_{j<i} p_j = \widetilde{n}_i \cdot o(p_i).$$

That is, the size of K_i^+ is dominated by the size of K_i. As a result, from the analysis in Section 10, it will follow that adding the subchains of previous samples does not affect substantially the randomness of the sample K_i.

In the up-phase, the algorithm considers for each $i-1$ and $\ell \in L_{i-1}$, the subset

$$K_i^\ell = K_i \cap L_i^\ell$$

from L_i^ℓ (the restriction of K_i to L_i^ℓ). Note that K_i^ℓ is not really a p_i-sample. However, in the analysis, we will see that the sampling bounds apply for this sort of sample, and so we also refer to it as a p_i-sample.

7 Portal-Chain Intersection Queries

For each ℓ in L_{i-1}, $i = 1, \ldots, k$, the portal-chain intersection query data structure $\mathcal{D}(\ell)$ consists of:

(i) $\widetilde{\mathcal{T}}(K_i^\ell)$ and a corresponding point location structure with query time $O(|\ell|) = \log \lambda_{i-1}$;

(ii) for each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^\ell)$, the chain-conflict list $L^\ell_{i|\widetilde{\Delta}}$;

(iii) if ℓ is not a leaf in the refinement tree, for each $\ell' \in L_i^\ell$, the data structure $\mathcal{D}(\ell')$.

A portal-chain conflict query for an arbitrary portal ρ and chain $\ell \in L_{i-1}$ first uses $\mathcal{D}(\ell)$'s point location data structure $\widetilde{\mathcal{T}}(K_i^\ell)$ to locate the endpoints of ρ. If ρ's endpoints are contained in different chain-trapezoids in $\widetilde{\mathcal{T}}(K_i^\ell)$, then ρ must intersect ℓ, and a conflict is reported. Otherwise, ρ is entirely contained in some $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^\ell)$, and the query continues recursively in the data structures $\mathcal{D}(\ell')$, for each subchain ℓ' that bounds $\widetilde{\Delta}$ or is in $\widetilde{\Delta}$'s conflict list $L^\ell_{i|\widetilde{\Delta}}$. This query procedure is summarized in Fig. 12.

`conflict?`$(\rho, \ell, i-1)$

Input: A portal ρ and a chain ℓ in L_{i-1}.
Output: `Yes` if ρ intersects ℓ, else `No`

1. Determine $\widetilde{\Delta}, \widetilde{\Delta}' \in \widetilde{T}_i^\ell$ which contain the endpoints of ρ
2. If $\widetilde{\Delta}$ and $\widetilde{\Delta}'$ are different then return `Yes`
3. For each ℓ' that bounds $\widetilde{\Delta}$ or in $L^\ell_{i|\widetilde{\Delta}}$ do
   if `conflict?`(ρ, ℓ', i) = `Yes` then return `Yes`
4. Return `No`

Figure 12: Portal-chain query procedure.

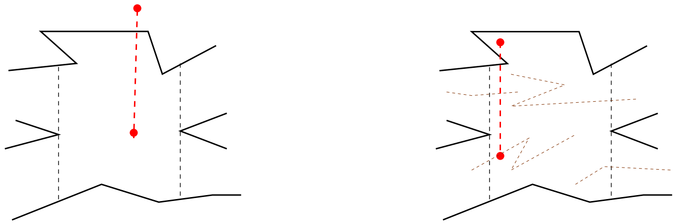


Figure 13: Two cases in the portal-chain conflict query: The portal endpoints are in different or in the same chain-trapezoid.

8 Up-Phase

In the preprocessing phase, the algorithm constructs data structures for portal-chain conflict queries to be used to hop along the chains in the top-down phase. Recall the gradation of subchains L_i, $i = 0, \ldots, k$, defined above and that for $\ell \in L_{i-1}$, L_i^ℓ is the set of subchains of ℓ in L_i. Let $K_i^\ell = L_i^\ell \cap K_i$ (as already pointed out, though not a real p_i-sample, we can think of it as such).

Note that computation of conflict lists for $\widetilde{\mathcal{T}}(K_i^\ell)$ in Step 2 also uses "hopping" and, hence, the procedure `conflict?`. This is what makes the bottom-up processing necessary. The i-th round of the up-phase is summarized in Fig. 14.

> `Up-Phase` (i-th round)
>
> *Input:* $\widetilde{\mathcal{T}}(K_j^\ell)$ and conflicts with L_j^ℓ for each $\ell \in L_{j-1}$, $j > i$
> *Output:* $\widetilde{\mathcal{T}}(K_i^\ell)$ and conflicts with L_i^ℓ for each $\ell \in L_{i-1}$
>
> For each $\ell \in L_{i-1}$ do
>
> 1. Compute $\mathcal{T}(K_i^\ell)$ and a corresponding point location structure
> 2. Compute $\widetilde{\mathcal{T}}(K_i^\ell)$ using `conformal`$(K_i^\ell, \mathcal{T}(K_i^\ell))$
> 3. Compute $L_{i|\widetilde{\Delta}}$ for all $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^\ell)$ by "hopping" along L_i^ℓ in $\widetilde{\mathcal{T}}(K_i^\ell)$

Figure 14: Up-phase procedure.

Running Time. In Sec. 11, using the sampling bounds obtained in the next section, we verify that the construction of the data structures $\mathcal{D}(\ell)$ is completed in expected $O(n)$ time. Moreover, we verify that the expected query time for a chain $\ell \in L_i$ is $O(\log^{3+\epsilon} \lambda_i)$, where $\epsilon > 0$ is an arbitrary small fraction. In summary:

Theorem 8.1 *The randomized two-phase algorithm constructs the trapezoidation of a simple polygon of size n in expected $O(n)$ time.*

9 Down-Phase

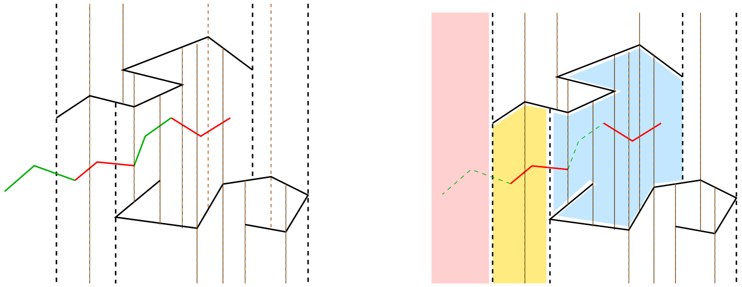
The down-phase consists of a round per each level in the gradation of the polygonal chain. In the i-th round, given the decomposition $\widetilde{\mathcal{T}}(K_{i-1}^+)$ and its conflict lists with respect to L_{i-1}, the algorithm adds the subchains in K_i^+ to $\widetilde{\mathcal{T}}(K_{i-1}^+)$ as summarized in Fig. 15.

> `Down-Phase` (i-th round)
>
> *Input:* $\widetilde{\mathcal{T}}(K_{i-1}^+)$ and conflicts with L_{i-1}
> *Output:* $\widetilde{\mathcal{T}}(K_i^+)$ and conflicts with L_i
>
> 1. For each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$ construct a plds $\mathcal{Q}(\widetilde{\Delta})$ for $\mathcal{T}(K_{i-1}^+)$ inside $\widetilde{\Delta}$.
> 2. For each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$ determine $K_{i|\widetilde{\Delta}}$:
>   Check for each $\ell' \in K_i^\ell$, $\ell \in L_{i-1|\widetilde{\Delta}}$, whether ℓ' conflicts with $\widetilde{\Delta}$ (using $\mathcal{D}(\ell')$ and $\mathcal{Q}(\widetilde{\Delta})$).
> 3. Starting with $\mathcal{T}(K_{i-1}^+)$ insert each $s \in R_i$ until $\mathcal{T}(K_i^+)$ is obtained:
>   An endpoint of $s \in \ell \in K_i$ is located by trying each $\mathcal{Q}(\widetilde{\Delta})$ with $\ell \in K_{i|\widetilde{\Delta}}$, then s is traced in the current trapezoidation, and both the trapezoidation and the point location data structures are updated.
> 4. Obtain $\widetilde{\mathcal{T}}(K_i^+)$ using `conformal`$(K_i^+, \mathcal{T}(K_i^+))$
> 5. Compute $L_{i|\widetilde{\Delta}}$ for all $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^+)$, by "hopping" along L_i in $\widetilde{\mathcal{T}}(K_i^+)$

Figure 15: Down-phase procedure.

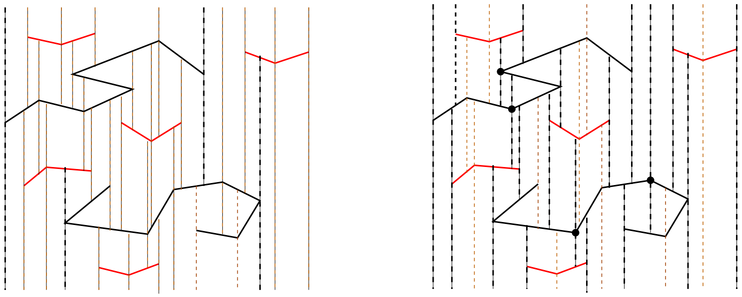
In Step 1, $\mathcal{Q}(\widetilde{\Delta})$ is constructed by running `BasicInc` using the chains that bound $\widetilde{\Delta}$ as input. Step 2 determines the conflict list $K_{i|\widetilde{\Delta}}$ by checking for each $\ell \in L_{i-1|\widetilde{\Delta}}$ and $\ell' \in L_i^\ell \cap K_i$, whether ℓ' intersects $\widetilde{\Delta}$: if so either ℓ' intersects one of the portals of $\widetilde{\Delta}$, or its endpoints are inside $\widetilde{\Delta}$. These queries are solved with $\mathcal{D}(\ell')$ and $\mathcal{Q}(\widetilde{\Delta})$ respectively. Step 3 inserts all the edges in K_i incrementally in a random ordering, following procedure `BasicInc`. An endpoint of $s \in \ell \in K_i$ is located by trying each $\mathcal{Q}(\widetilde{\Delta})$ with $\ell \in K_{i|\widetilde{\Delta}}$. The procedure `conformal` in Step 4 was described in Section 4; it takes time linear in the size of $\mathcal{T}(K_i^+)$ and returns the (conformal) chain-trapezoidation $\widetilde{\mathcal{T}}(K_i^+)$ (including its adjacency graph). In Step 5, the conflict lists $L_{i|\widetilde{\Delta}}$ for $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^+)$ are found chain by chain, using the adjacency graph of $\widetilde{\mathcal{T}}(K_i^+)$ and the data structures $\mathcal{D}(\ell)$ to "hop" along L_i in $\widetilde{\mathcal{T}}(K_i^+)$, as described in Section 5.

Running Time. In Sec. 11, using the sampling bounds obtained in the next section, we verify that given the data structures $\mathcal{D}(\ell)$ with query time $O(\log^{3+\epsilon} \lambda_i)$, for $\ell \in L_i$, the top-down construction phase is completed in expected time $O(n)$.



(a) (b)

Figure 16: Location of endpoints: (a) shows the chain-trapezoidation of two chains, and another one, ℓ, some of whose subchains are to be inserted (those solid in (b)). The chain-trapezoids in conflict with ℓ appear shaded in (b). Consider the subchain of ℓ to the left, say ℓ'. ℓ' determines for each of the three shaded chain-trapezids whether it conflicts with it. Since it conflicts with the first two, then each vertex in ℓ' checks both of them to determine the one that contains it. This also gives the precise trapezoid that contains the vertex.



(a) (b)

Figure 17: Insertion and conformalization: The two subchains of the previous figure, and three others are inserted. (a) shows the new trapezoidation. (b) shows the new chain-trapezoidation.

Part III
Analysis

10 Sampling Bounds

To analyze the running time of the algorithm, we need bounds on the sizes of the subproblems resulting by taking a random sample from a set of chains and then constructing its chain-trapezoidation. Let K be a p-sample from L, and recall that for a chain-trapezoid $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K)$, $L_{|\widetilde{\Delta}}$ denotes the list of conflicts of $\widetilde{\Delta}$ in L, and that $\widetilde{n}_{\widetilde{\Delta}} = |L_{|\widetilde{\Delta}}|$. We cannot prove the bound in Eqn. (2). Such a bound could be proved in the framework of *configuration spaces*, when certain *locality* [8, 16] or *monotonicity* [10, 1] properties hold for the decomposition induced by the sample (see [14] for a survey), but neither of these properties hold for the chain-trapezoidation. Fortunately, it is possible to prove a weaker bound that is only a factor $O(\log\lambda)$ larger, and that suffices to verify that the algorithm has expected linear running time. The proof of the bound uses a standard trick [8, 4]: one obtains a nontrivial bound for a p-sample in terms of a trivial bound for a $(p/2)$-sample. First, we need a fact about the chain-trapezoidation that limits the amount of non-locality in the definition of the chain-trapezoidation.

Limited Non-Locality

Recall that a chain-trapezoid is bounded by at most two chains and at most two ray-pairs, each one originating from another chain (but possibly a bounding chain). We say that these at most four chains *determine* the chain-trapezoid. Let $\widetilde{\mathcal{T}}^*(L)$ be the set of all chain-trapezoids determined by L, that is, those chain-trapezoids determined by a subset of at most four chains in L (but note that some other chains in L can conflict with such trapezoids). Let $\widetilde{\mathcal{T}}^c(K)$ be the set of all *candidate* chain-trapezoids determined by K and with empty conflict list with respect to K. Note that $\widetilde{\mathcal{T}}^c(K)$ is bigger than $\widetilde{\mathcal{T}}(K)$ as there are candidate chain-trapezoids determined by K that were not chosen in the construction of $\widetilde{\mathcal{T}}(K)$. For $\widetilde{\Delta} \in \widetilde{\mathcal{T}}^*(L)$, let $\delta(\widetilde{\Delta}) \subseteq L$ denote the set of those up to four chains that determine $\widetilde{\Delta}$. For $\widetilde{\Delta} \in \widetilde{\mathcal{T}}^*(L)$, we have the *locality* property:

$$\widetilde{\Delta} \in \widetilde{\mathcal{T}}^c(K) \quad \text{iff} \quad \delta(\widetilde{\Delta}) \subseteq K \text{ and } L_{|\widetilde{\Delta}} \cap K = \emptyset. \tag{3}$$

The following lemma states that the chain-trapezoidation is choosen out of a relatively small "pool" of candidates that satisfy a locality property.

Lemma 10.1 *Let K be a set of chains each of length at most λ. Then, $|\widetilde{\mathcal{T}}^c(K)| = O(|K|\lambda^2)$.*

Proof. In $\mathcal{T}(K)$, let a *region* be the union of the trapezoids corresponding to a connected subgraph of $\mathcal{G}(K)$. For each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K)$ consider the maximum region $\Gamma_{\widetilde{\Delta}}$ of $\mathcal{T}(K)$ that contains $\widetilde{\Delta}$ and is bounded by the same one or two chains that bound $\widetilde{\Delta}$. Note that $\Gamma_{\widetilde{\Delta}}$ may not be conformal. Any candidate chain-trapezoid is a subregion of $R_{\widetilde{\Delta}}$ for some $\widetilde{\Delta}$. Since clearly the number of subregions of any $R_{\widetilde{\Delta}}$ is $O(\lambda^2)$, and the size of $\widetilde{\mathcal{T}}(K)$ is $O(|K|)$, then it follows that the size of $\widetilde{\mathcal{T}}^c(K)$ is $O(|K|\lambda^2)$. ◻

Bounds

We use this result now to prove bounds for the chain-conflict list sizes in the chain-trapezoidal decomposition of a random sample of chains. Recall that a p-sample from L is a subset of S chosen with equal probability out of all subsets of L of size $p|L|$ (we ignore that this may not be an integer).

Lemma 10.2 *Let L be a set of $\widetilde{n}$ chains of length λ, and let $n = \lambda\widetilde{n}$ be the total number of edges. Let $K \subseteq L$ be a p-sample, $\widetilde{r} = p\widetilde{n}$ its (expected) size, and let $\widetilde{\mathcal{T}}(K)$ be its chain-trapezoidal decomposition. For $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K)$, we write $\widetilde{n}_{\widetilde{\Delta}} = |L_{|\widetilde{\Delta}}|$. Let f be a positive nondecreasing function such that $f(O(x)) = O(f(x))$. Then*

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} f(p\widetilde{n}_{\widetilde{\Delta}})\right] = O(\widetilde{r}\cdot f(\log\lambda)) \tag{4}$$

Proof. Let $K' \subseteq L$ be a $(p/2)$-sample. Recall that $\widetilde{\mathcal{T}}^c(K)$ is the set of candidate chain-trapezoids for K, and that for these chain-trapezoids the locality property in Eqn. (3) holds. Thus (assuming $\widetilde{r}$ is even),

$$\begin{aligned}
\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} &= \frac{\binom{\widetilde{n}-\delta(\widetilde{\Delta})-\widetilde{n}_{\widetilde{\Delta}}}{\widetilde{r}-\delta(\widetilde{\Delta})}}{\binom{\widetilde{n}}{\widetilde{r}}} \\
&= \left(\frac{\binom{\widetilde{n}-\delta(\widetilde{\Delta})-\widetilde{n}_{\widetilde{\Delta}}}{\widetilde{r}-\delta(\widetilde{\Delta})}}{\binom{\widetilde{n}}{\widetilde{r}}}\cdot\frac{\binom{\widetilde{n}}{\widetilde{r}/2}}{\binom{\widetilde{n}-\delta(\widetilde{\Delta})-\widetilde{n}_{\widetilde{\Delta}}}{\widetilde{r}/2-\delta(\widetilde{\Delta})}}\right)\cdot\frac{\binom{\widetilde{n}-\delta(\widetilde{\Delta})-\widetilde{n}_{\widetilde{\Delta}}}{\widetilde{r}/2-\delta(\widetilde{\Delta})}}{\binom{\widetilde{n}}{\widetilde{r}/2}} \\
&\le C_\delta\cdot e^{-p\widetilde{n}_{\widetilde{\Delta}}/2}\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\}
\end{aligned} \tag{5}$$

where $C_\delta = e^{d+1}$ with d an upper bound for all $\delta(\widetilde{\Delta})$, after some "gymnastics" with the binomial coefficients that we omit here. Using this upper bound, we obtain

$$\begin{aligned}
\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)} f(p\widetilde{n}_{\widetilde{\Delta}})\right] &= \sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} \\
&= O\left(\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)}\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\}\right) \\
&= O(\mathbf{E}[|\widetilde{\mathcal{T}}^c(K')|]) \\
&= O(\widetilde{r}\lambda^2),
\end{aligned}$$

using Lemma 10.1. Let $\tau > 0$ be a parameter. Then, using again the upper bound above,

$$\begin{aligned}
\mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\right] &= \sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)\\ \widetilde{n}_{\widetilde{\Delta}}>2(\log\tau)/p}} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} \\
&= O(1)\cdot\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot e^{-p\widetilde{n}_{\widetilde{\Delta}}/2}\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\} \\
&= \frac{O(1)}{\tau}\cdot\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\} \\
&= \frac{O(1)}{\tau}\cdot\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')} f(p\widetilde{n}_{\widetilde{\Delta}})\right] = \frac{O(1)}{\tau}\cdot\mathbf{E}[|\widetilde{\mathcal{T}}^c(K'')|] = O\left(\widetilde{r}\cdot\frac{\lambda^2}{\tau}\right),
\end{aligned}$$

where K'' is a $(p/4)$-sample. Finally, using $\tau = \lambda^2$,

$$\begin{aligned}
\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} f(p\widetilde{n}_{\widetilde{\Delta}})\right] &= \mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\right] + \mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)\\ \widetilde{n}_{\widetilde{\Delta}}\le\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\right] \\
&\le \mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\right] + f(2\log\tau)\cdot\mathbf{E}[|\widetilde{\mathcal{T}}(K)|] \\
&= O\left(\widetilde{r}\cdot\frac{\lambda^2}{\tau}\right) + O(f(2\log\tau)\cdot\widetilde{r}) = O(f(\log\lambda)\cdot\widetilde{r}).
\end{aligned}$$

◻

For the analysis of the query time of the ray-shooting data structure, we also need a bound for the expectation of the conflict list size of the chain-trapezoid that contains a *fixed* point x.

Lemma 10.3 *Let $K \subseteq L$ be a p-sample, and for a fixed point x, let $\widetilde{\Delta}_x$ be the chain-trapezoid in $\widetilde{\mathcal{T}}(K)$ that contains x. Then*

$$\mathbf{E}\left[|L_{|\widetilde{\Delta}_x}|\right] = O\left(\frac{1}{p}\cdot\log\lambda\right) = O\left(\frac{\widetilde{n}}{\widetilde{r}}\cdot\log\lambda\right).$$

Proof. The proof is analogous to the one for the previous lemma and we only present an outline. First, note that the number of candidate chain-trapezoids in $\widetilde{\mathcal{T}}^c(K)$ that contain x

is $O(\lambda^2)$, and so

$$\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)\\ x\in\widetilde{\Delta}}} \mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} = \mathbf{E}\left[\left|\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K) : x\in\widetilde{\Delta}\}\right|\right] = O(\lambda^2).$$

Then, using Eqn. (5) and following derivations similar to those in the previous lemma, we obtain the sequence of bounds:

$$\mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\\ x\in\widetilde{\Delta}}} p\widetilde{n}_{\widetilde{\Delta}}\right] = O(\lambda^2), \quad \mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}\\ x\in\widetilde{\Delta}}} p\widetilde{n}_{\widetilde{\Delta}}\right] = O\left(\frac{\lambda^2}{\tau}\right), \quad \mathbf{E}[p\widetilde{n}_{\widetilde{\Delta}_x}] = O(\log\lambda).$$

$\square$

11 Running Time Analysis

First, we note that the sampling bound of Lemma 10.2 holds for K_i^+ even though it contains, in addition to the true random p_i-sample K_i, all the subchains of previous samples K_j, $j<i$. This is because, as noted in Section II, the size is dominated by K_i and so, from the proof of the lemma, the right hand side of Eqn. (4) is not affected.

In the analysis below, two specific sums appear that can be bounded using Eqn. (4):

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} \widetilde{n}_{\widetilde{\Delta}}\right] = O(\widetilde{n}\cdot\log\lambda), \tag{6}$$

and

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} \widetilde{n}_{\widetilde{\Delta}}\log(\widetilde{n}_{\widetilde{\Delta}})\right] = O\left(\widetilde{n}\cdot\log\lambda\cdot\log\left(\frac{\log\lambda}{p}\right)\right). \tag{7}$$

Up-Phase

First, we verify the query time. The expected query time $Q(\lambda_i)$ is the sum of the time needed for a point location query, plus the expected time needed for all the recursive queries. Thus, we have

$$\begin{aligned} Q(\lambda_{i-1}) &= O(\log\lambda_{i-1}) + O\left(\left(\frac{1}{p_i}\right)\log\lambda_i\right)\cdot Q(\lambda_i) \\ &= O(\log\lambda_{i-1}) + O((\log\lambda_{i-1})^3\log\lambda_i)\cdot Q(\lambda_i) \\ &= O(\log^{3+\epsilon}\lambda_{i-1}), \end{aligned}$$

where ϵ is any positive fraction. We have used Lemma 10.3 to bound the expected number of chain-conflicts as $O((1/p_i)\log\lambda_i)$. Note that it is valid to use the bound for a *fixed* point, as the random choices in the i-th and later rounds are independent of the random choices in earlier rounds. Next, we estimate the expected construction time for $\ell\in L_i$. From the previous description, using Eqn. (6), the expected time is

$$\begin{aligned} &O\left((p_i\lambda_{i-1})\log(p_i\lambda_{i-1}) + \left(\frac{\lambda_{i-1}}{\lambda_i}\log\lambda_i\right)\cdot Q(\lambda_i)\right) = \\ &= O\left(\frac{\lambda_{i-1}}{\log^3\lambda_{i-1}}\log\left(\frac{\lambda_{i-1}}{\log^3\lambda_{i-1}}\right) + \frac{\lambda_{i-1}}{\log^2\lambda_{i-1}}\cdot\log(\log^2\lambda_{i-1})\cdot\log^{3+\epsilon}(\log^2\lambda_{i-1})\right) \\ &= O\left(\frac{\lambda_{i-1}}{\log\lambda_{i-1}}\right), \end{aligned}$$

where the first term accounts for the construction of $\widetilde{\mathcal{T}}(K_i^\ell)$, and the second term accounts for $O(1)$ portal-chain intersection queries per chain-conflict. Adding over all $\ell\in L_{i-1}$, the construction time in the $(i-1)$-st level is $O(n/\log\lambda_{i-1})$, and adding over all i, the total construction time is $O(n)$.

Down-Phase

Consider the i-th round, and let $\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_i^+)$. For the purpose of analysis, let us write $\widetilde{n}_{i,\widetilde{\Delta}} = |L_{i|\widetilde{\Delta}}|$ and $\widetilde{r}_{i,\widetilde{\Delta}} = |K_{i|\widetilde{\Delta}}|$. Recall that $\widetilde{n}_i = |L_i| = n/\lambda_i$. Note that the expected value of $\widetilde{r}_{i,\widetilde{\Delta}}$ is bounded by $p_i\cdot\widetilde{n}_{i-1,\widetilde{\Delta}}\cdot\frac{\lambda_{i-1}}{\lambda_i}$. In Step 1.a, for each $\ell\in L_{i-1|\widetilde{\Delta}}$, checking whether $\ell'\in L_i^\ell\cap K_i$ intersects $\widetilde{\Delta}$ takes expected time is $O(\log^{3+\epsilon}\lambda_{i-1})$, using the data structure $\mathcal{D}$ for the chains bounding $\widetilde{\Delta}$. Thus, using Eqn. (6), the total expected time for this step is big-O of

$$\begin{aligned} \mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_{i-1}^+)} p_i\cdot\widetilde{n}_{i-1,\widetilde{\Delta}}\cdot\frac{\lambda_{i-1}}{\lambda_i}\cdot\log^{3+\epsilon}\lambda_{i-1}\right] &= O\left(p_i\cdot(\widetilde{n}_{i-1}\log\lambda_{i-1})\cdot\frac{\lambda_{i-1}}{\lambda_i}\cdot\log^{3+\epsilon}\lambda_{i-1}\right) \\ &= O\left(\frac{n}{\log^{1-\epsilon}\lambda_{i-1}}\right). \end{aligned}$$

(Here, the term $\log^{3+\epsilon}\lambda_{i-1}$ could be reduced to $\log\lambda_{i-1}+\log^{3+\epsilon}\lambda_i$, if we construct a point location data structure for each $\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_{i-1}^+)$.) For $\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_{i-1}^+)$, let $\widetilde{r}_{i,\widetilde{\Delta}} = |K_{i|\widetilde{\Delta}}|$ and let $r_{i,\widetilde{\Delta}} = \widetilde{r}_{i,\widetilde{\Delta}}\lambda_i$ be the corresponding number of edges. The expected value of $\widetilde{r}_{i,\widetilde{\Delta}}$ is $p_i(\lambda_{i-1}/\lambda_i)\widetilde{n}_{i-1,\widetilde{\Delta}}$ and the expected value of $r_{i,\widetilde{\Delta}}$ is $p_i\lambda_{i-1}\widetilde{n}_{i-1,\widetilde{\Delta}}$. In Step 3, $\mathcal{T}(K_i^+)$ is computed using procedure `BasicInc`. For each $\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_{i-1}^+)$, this takes time $O(r_{i,\widetilde{\Delta}}\log r_{i,\widetilde{\Delta}})$. Using Eqn. (7), the total expected time is big-O of

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K_{i-1}^+)} r_{i,\widetilde{\Delta}}\log r_{i,\widetilde{\Delta}}\right] = O\left(\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{T}_{i-1}} (p_i\lambda_{i-1}\widetilde{n}_{i-1,\widetilde{\Delta}})\log(p_i\lambda_{i-1}\widetilde{n}_{i-1,\widetilde{\Delta}})\right]\right)$$

$$= O\left(p_i \cdot \lambda_{i-1} \cdot \widetilde{n}_{i-1} \cdot \log \lambda_{i-1} \cdot \log\left(\frac{p_i \lambda_{i-1}}{p_{i-1}} \log \lambda_{i-1}\right)\right)$$
$$= O\left(n \cdot p_i \cdot \log^2 \lambda_{i-1}\right) = O\left(\frac{n}{\log \lambda_{i-1}}\right).$$

The conformal decomposition in Step 4 is constructed as described in Section 4 and requires expected time big-O of

$$\mathbf{E}[|\mathcal{T}(K_i^+)|] = O(p_i \cdot \widetilde{n}_i \cdot \lambda_i) = O\left(\frac{n}{\log^3 \lambda_{i-1}}\right).$$

In Step 5, the conflict lists for regions $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^+)$ with subchains in L_i are found using the data structures $\mathcal{D}$, $O(1)$ queries per conflict determined, so the expected time is

$$O(\widetilde{n}_i \cdot \log \lambda_i \cdot Q(\lambda_i)) = O\left(\frac{n}{\lambda_i} \cdot \log \lambda_i \cdot \log^{3+\epsilon} \lambda_i\right) = O\left(n \cdot \frac{\log^{4+\epsilon} \lambda_i}{\lambda_i}\right) = O\left(\frac{n}{\log \lambda_{i-1}}\right).$$

The sum of all these contributions over all the rounds is $O(n)$.

12 Concluding Remarks

Though the new randomized algorithm for triangulating a simple plygon is considerably simpler than Chazelle's deterministic algorithm, at the same time, it is comparatively more complicated than Seidels randomized algorithm. Furthermore, since for any practical value of n, $\log^* n$ is a small constant, the new algorithm is not likely to be of practical value.

We mention two points brought up by Raimund Seidel. The first one (in [17]) is that for this work to make any sense, one needs to assume a computation model in which a random number between 1 and n can be obtained in constant time (for a logarithmic cost would already imply an $\Omega(n \log n)$ bound for the running time). As he points out, this seems a reasonable assumption, since we are assuming a model in which arithmetic operations take constant time. The second one (private communication) is the possibility of obtaining a "simple" algorithm given the knowledge of the existence of Chazelle's linear time algorithm. The algorithm he suggests is randomized and in fact uses his own simple non-optimal algorithm [17] executed a number of levels. Actually, a more general argument is possible: Consider a problem for which a complicated linear time algorithm is known for solving the problem and a simple linear time algorithm is known for verifying the correctness of a candidate output; a "simple" algorithm for the problem is obtained by considering a canonical list of all possible algorithms and running each for a number of steps (dovetailing over the set of pairs index-integer where the index corresponds to an algorithm and the integer to the constant in the linear time bound), until the verification succeeds. Though such an algorithm may be objectively regarded as simple, when measured by its size, it seems hardly acceptable as simple.

Some open questions: Is the conjectured tighter sampling bound for the chain-trapezoidation true? Is it possible to combine the new polygon trapezoidation algorithm with a segment intersection algorithm to obtain an algorithm that can report the k intersections of a chain of n segments in time $O(n + k)$? (this is a conjecture of Chazelle) Can it be parallelized? Can the approach of sampling on subchains lead to efficient algorithms for other problems on simple polygons? Does a linear time randomized algorithm considerably simpler than the one described in these notes exist ? (The authors share with Bernard Chazelle the belief that this is not the case.) Finally, does a deterministic algorithm simpler than Chazelle's exist?

References

[1] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27:654–667, 1998.

[2] N. M. Amato, M. Goodrich and E. A. Ramos. Linear-time simple polygon triangulation made easier via randomization. In *ACM Sympos. Comput. Geometry,* 2000.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[4] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[5] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Trans. Graph.*, 3(2):135–152, 1984.

[6] K. L. Clarkson, R. Cole, and R. E. Tarjan. Erratum: Randomized parallel algorithms for trapezoidal diagrams. *Internat. J. Comput. Geom. Appl.*, 2(3):341–343, 1992.

[7] K. L. Clarkson, R. Cole, and R. E. Tarjan. Randomized parallel algorithms for trapezoidal diagrams. *Internat. J. Comput. Geom. Appl.*, 2(2):117–133, 1992.

[8] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[9] K. L. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[10] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. *Discrete Comput. Geom.*, 14:261–286, 1995.

[11] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3(2):153–174, 1984.

[12] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7(4):175–179, 1978.

[13] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log \log n)$ time with simple data structures. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 34–43, 1990.

[14] J. Matoušek. Derandomization in computational geometry. *J. Algorithms*, 20:545–580, 1996.

[15] K. Mulmuley. A fast planar partition algorithm, I. *J. Symbolic Comput.*, 10(3-4):253–280, 1990.

[16] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[17] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991.

[18] R. E. Tarjan and C. J. Van Wyk. An $O(n \log \log n)$-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988. Erratum in 17 (1988), 106.