A Randomized Algorithm for Triangulating a Simple Polygon in Linear Time

Nancy M. Amato [*] Michael T. Goodrich [†] Edgar A. Ramos [‡]

December 7, 2000

Abstract

We describe a randomized algorithm 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. Our algorithm is considerably simpler than Chazelle's (1991) celebrated optimal deterministic algorithm. The new algorithm can be viewed as a combination of Chazelle's algorithm and of simple non-optimal randomized algorithms due to Clarkson *et al.* (1991) and to Seidel (1991). As in Chazelle's algorithm, it is indispensable to include a bottom-up preprocessing phase, in addition to the actual top-down construction. An essential new idea is the use of random sampling on subchains of the initial polygonal chain, rather than on individual edges as is normally done.

1 Introduction

Polygon triangulation is a classic problem in computational geometry, and one of the first problems studied in the field [12]. Furthermore, there are several other problems in computational geometry dealing with polygons that have efficient solutions that begin with polygon triangulation as a preprocessing step (e.g., see [14, 15]). Thus, there has been considerable interest in finding efficient algorithms for this problem.

[*]Texas A&M University, College Station, TX. E-mail: `amato@cs.tamu.edu`. Research supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-9805823, and EIA-9810937, DOE ASCI ASAP Grant B347886, and by Texas Higher Education Coordinating Board Grant ARP-036327-017.

[†]The Johns Hopkins University, Baltimore, MD. E-mail: `goodrich@jhu.edu` This work was supported by ARO MURI Grant DAAH04-96-1-0013 and NSF Grant CCR-9732300.

[‡]Max-Planck-Institut für Informatik, Saarbrücken, Germany. E-mail: `ramos@mpi-sb.mpg.de`

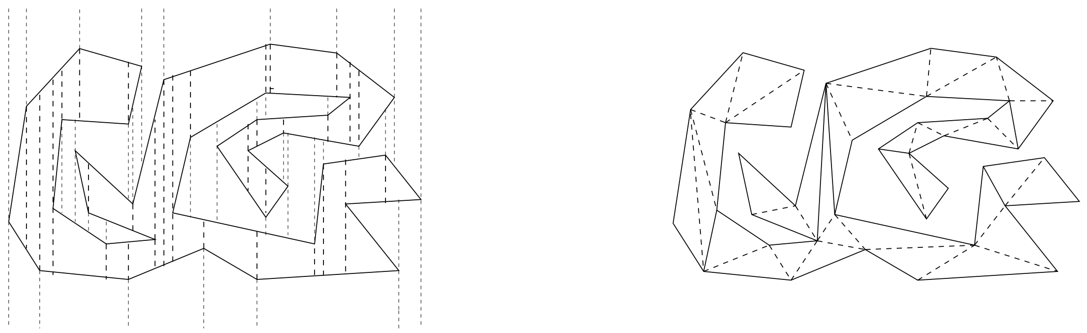

Figure 1: Trapezoidation and triangulation of a simple polygon. The visibility rays in the interior of the polygon (longer dashes) are used to derive a triangulation. A diagonal from each *reflex* vertex to the opposite vertex in the corresponding trapezoid (solid diagonals) decomposes the polygon into *monotone* polygons. Each monotone polygon can then be easily triangulated in linear time (dashed diagonals).

1.1 Related Work

Garey *et al.* [12] were the first to provide a non-trivial algorithm for the polygon triangulation problem. Their algorithm runs in $\Theta(n \log n)$ time and is based on an elegant plane-sweeping paradigm. Asano *et al.* [2] show that this bound is in fact optimal for polygons that may contain holes. For simple polygons without holes, the lower bound of Asano *et al.* does not hold, however. This fact, and the importance of the polygon triangulation problem, in turn prompted several researchers to work on methods for beating the $\Theta(n \log n)$ time bound for this problem.

Fournier and Montuno [11] and Chazelle and Incerpi [5] showed, even prior to the Asano *et al.* lower bound result, that to triangulate a simple polygon in linear time it is sufficient to produce a trapezoidal decomposition (*trapezoidation*) of a simple polygon. A trapezoidation is formed by shooting a vertical (visibility) ray through each vertex of the polygon, stopping each ray as soon as it hits another segment on the polygon. See Fig. 1. Since this early work showing the importance of trapezoidation for triangulation, every published triangulation algorithm has concentrated on improving the running time of producing a trapezoidation of a simple polygon. For example, Tarjan and Van Wyk [21] and Kirkpatrick *et al.* [16] showed that the trapezoidation step can be performed in $\Theta(n \log \log n)$ time, resulting in a similar running time for the polygon triangulation problem. Using randomization, Clarkson *et al.* [9], Clarkson *et al.* [6, 7], and Seidel [20] gave simple randomized algorithms that run in $\Theta(n \log^* n)$ expected time. Finally, in a much celebrated and anticipated result, Chazelle [3] showed that one could, in fact, triangulate a polygon in linear time. Unfortunately, the trapezoidation method utilized by this optimal deterministic algorithm is quite complex. Indeed, this conceptual complexity has led many researchers, including Chazelle [3] himself, to ask whether there is a simple randomized algorithm for triangulating a polygon in linear time. To our knowledge, no linear-time randomized algorithm has been presented previously.

1.2 Our Results

We describe a randomized algorithm for computing the trapezoidation of a simple polygon. The expected running time of our algorithm is linear in the size of the polygon. As already mentioned, from the trapezoidation, a triangulation of the polygon can be obtained in linear time using well-known methods [5, 11]. Thus, our algorithm provides a randomized algorithm for polygon triangulation that runs in linear expected time. In addition, our algorithm is considerably simpler than Chazelle's optimal deterministic algorithm; hence, it effectively responds to the open problem posed by Chazelle as to the existence of a simple randomized triangulation algorithm that runs in linear expected time.

The general approach of our algorithm for computing a trapezoidation of a simple polygon P follows that of the non-optimal randomized algorithms of Clarkson *et al.* [6, 7] and Seidel [20]. That is, we 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. 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, we decompose the original polygonal chain into subchains, sample from the resulting set of subchains and, taking advantage of the *coherence* between edges in the polygonal chain, maintain lists of *subchain conflicts* for the resulting subproblems, rather than edge conflicts.

A technical difficulty in this approach is the definition of the subproblems defined by a set of subchains. For the approach to work, one needs a decomposition with a size that is proportional to the number of subchains involved, and with *faces* (subproblems) of *bounded complexity.* 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 *walks* along the polygonal chain. This concept also appears in Chazelle's algorithm; following him, we call this bounded-complexity property *conformality.* Fortunately, our problem is simpler; we describe a simple procedure that computes a conformal decomposition in time linear in the number of edges in the set of subchains. This is actually sublinear in the size of the input chain because it is performed for a small sample. In order to traverse the decomposition efficiently, we need a data structure for each subchain that answers intersection queries between a vertical edge, called a *portal*, and the subchain. Thus, as in Chazelle's algorithm, we need a preprocessing phase that constructs these data structures prior to the actual construction phase. These phases proceed bottom-up and top-down, respectively. Randomization also plays an important role in the preprocessing phase. Chazelle has "argued" that such a combination of bottom-up and top-down approaches is indispensable [3].

A final technicality is the proof of appropriate *sampling bounds* for the sizes of the *chain-conflict* lists of our conformal decomposition: such bounds are known under *locality* or *monotonicity* properties that our decomposition does not satisfy [1, 8, 10, 17, 19]. Fortunately, we can prove appropriate bounds using the fact that, although the faces in the decomposition do not satisfy a locality property, they are chosen from a relatively small "pool" of candidates that satisfy a locality property.

This paper is organized as follows. First, for comparison purposes, we present a detailed outline of a non-optimal randomized algorithm (Sec. 2). We then describe our procedure to compute a conformal decomposition for a set of chains (Sec. 3) and our linear time algorithm (Sec. 4). Finally, we obtain appropriate sampling bounds (Sec. 5) for bounding the running time of our algorithm (Sec. 6). We conclude with some remarks and state some open problems (Sec. 7).

2 A Non-optimal Algorithm

For the purpose of comparison with our algorithm, and as a gentle introduction, we outline a non-optimal randomized algorithm which is an adaptation from those in [6, 7] and [20]. Let ℓ_0 be a simple polygonal chain, S be the corresponding set of polygon edges, and let $n = |S|$. We make the non-degeneracy assumption that no two vertices have the same horizontal coordinate; this can be simulated symbolically through lexicographic ordering. For $R \subseteq S$, let $\mathcal{T}(R)$ denote the usual (vertical) *trapezoidal decomposition* or *trapezoidation* of the plane induced by R, obtained by introducing *vertical visibility rays* from the endpoints of edges in R. This *planar subdivision* has $O(|R|)$ *trapezoids* (faces) and each trapezoid has at most 2 edges (on top and bottom) and 4 vertical rays (on left and right) on its boundary. For $\Delta \in \mathcal{T}(R)$, 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 $n_\Delta = |S_{|\Delta}|$. We adopt the following sampling model: For p with $0 \le p \le 1$, a *p-sample* R from S is obtained by taking each $s \in S$ into R with probability p independently.

2.1 Algorithm Outline

The algorithm constructs the trapezoidation of a successively finer random sample in $O(\log^* n)$ rounds. Specifically, we define a global probability $p_i = 1/\log^{(i)} n$ for round i in the computation, and let R_i be a p_i-sample from S chosen in this round (so each $s \in S$ is taken with probability p_i independently). Furthermore, let $R_i^+ = \bigcup_{j \le i} R_j$. Note that R_i^+ is a p_i^+-sample from S where $p_i^+ \le \sum_{j \le i} p_j = \Theta(p_i)$. In the i-th round, given $\mathcal{T}(R_{i-1}^+)$ and its conflicts with respect to S (that is, $S_{|\Delta}$ for $\Delta \in \mathcal{T}(R_{i-1}^+)$) the algorithm constructs $\mathcal{T}(R_i^+)$ and its conflicts with respect to S as summarized in Fig. 2.

Step 1.a, for a $\Delta \in \mathcal{T}(R_{i-1}^+)$, involves a simple scan of the conflict list $S_{|\Delta}$ and hence takes time $O(n_\Delta)$.[1] Step 1.b computes T_Δ, which is $\mathcal{T}(R_{i|\Delta} \cup \{e_1, e_2\})$ restricted to Δ, where e_1, e_2 are the (non-vertical) edges bounding Δ. This takes time $O(r_\Delta \log r_\Delta)$, where $r_\Delta = |R_{i|\Delta}|$, using an algorithm for computing a trapezoidation with this complexity. In particular, we choose to use a simple randomized incremental algorithm described in [20], which we will refer to as `Basic-Trapezoidation` (this is a simplification of the algorithms of Clarkson and Shor [8] and Mulmuley [18] to the case of non-intersecting segments). This algorithm produces a representation of the resulting trapezoidation which allows each trapezoid to

[1] Alternatively, one can maintain for each $s \in S$ the list of trapezoids it intersects, and then the scan is not necessary.

```
Non-Optimal-Trapezoidation ($i$-th round)

Input: $\mathcal{T}(R^+_{i-1})$ and its conflicts with $S$
Output: $\mathcal{T}(R^+_i)$ and its conflicts with $S$

1. For each $\Delta \in \mathcal{T}(R^+_{i-1})$
      a. Determine $R_{i|\Delta}$
      b. $T_\Delta \leftarrow \begin{cases} \mathcal{T}(R_{i|\Delta} \cup \{e_1, e_2\}) \text{ restricted to } \Delta, \\ \text{where } e_1, e_2 \text{ are the edges that bound } \Delta \end{cases}$
2. Merge all the $T_\Delta$, $\Delta \in \mathcal{T}(R^+_{i-1})$, into $\mathcal{T}(R^+_i)$
3. Compute $S_{|\Delta}$ for all $\Delta \in \mathcal{T}(R^+_i)$, by "walking" along $\ell_0$ in $\mathcal{T}(R^+_i)$
```

Figure 2: Non-optimal trapezoidation procedure.

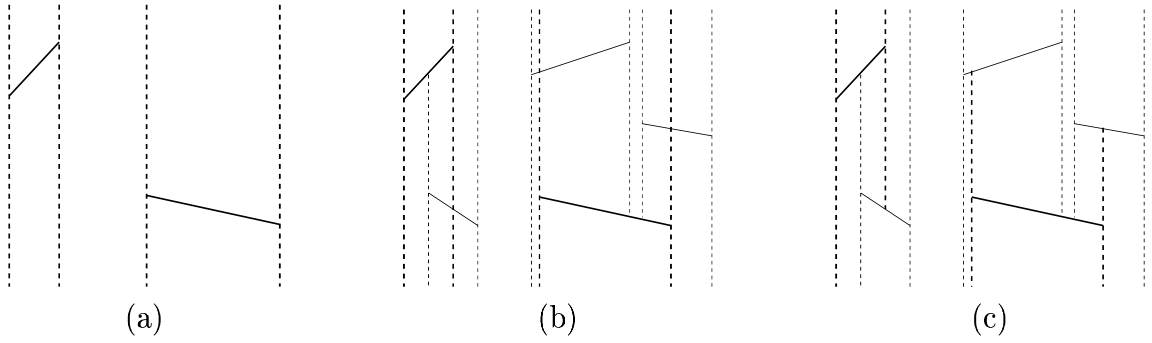

Figure 3: Non-optimal trapezoidation example. The three thin edges are added to the trapezoidation determined by the two thick edges: (a) before new edges inserted, (b) local trapezoidations after Step 1, and (c) trapezoidation after merging in Step 2.

determine in constant time the adjacent polygon edges on top and bottom (if any) and the adjacent trapezoids to the left and right (up to four). Another byproduct of this algorithm is a *point location data structure* for the resulting trapezoidation which allows one to locate the trapezoid that contains a query point in time that is logarithmic in the size of the decomposition. Step 2 involves "stitching" together pieces of trapezoids in $\mathcal{T}(R^+_i)$ that are "chopped" by vertical-rays in $\mathcal{T}(R^+_{i-1})$. In other words, vertical-rays that are no longer necessary are removed. See Fig. 3. It takes total time linear in the sizes of the T_Δ's, and hence, the time required is dominated by that of Step 1. Since each trapezoid in $\mathcal{T}(R^+_i)$ has at most four neighbors through vertical rays, then, assuming that an appropriate data structure is used, Step 3 can be performed in time proportional to the size of ℓ_0, which is n, plus the total number of segment-trapezoid conflicts found.

2.2 Sampling Bound and Analysis

The algorithm can be analyzed with the use of the following *sampling bound.* Let R be a *p-sample* from S. Then, for any function f such that $f(x) = O(e^{x/2})$,

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

where $r = pn$ is the expected size of R. See [8, 19] or Section 5. Using $f(x) = x$ in Eqn. (1), the expected total size of the conflict lists is $O(n)$. This implies a bound of $O(n)$ for the expected time required by all steps in a round, except Step 1.b. Denoting the expectation with respect to the first i samples by $\mathbf{E}_{\leq i}$, the total expected time required by Step 1.b is

$$\begin{aligned}
\mathbf{E}_{\leq i}\left[\sum_{\Delta \in \mathcal{T}(S_{\rho(i)})} O(r_\Delta \log r_\Delta)\right] &= \mathbf{E}_{\leq i-1}\left[\sum_{\Delta \in \mathcal{T}(S_{\rho(i)})} O\left((p_i n_\Delta) \log(p_i n_\Delta)\right)\right] \\
&= \left(\frac{p_i}{p_{i-1}}\right) \log\left(\frac{p_i}{p_{i-1}}\right) \cdot O(p_{i-1} n) \\
&= \frac{1}{\log^{(i)} n} \log\left(\frac{\log^{(i-1)} n}{\log^{(i)} n}\right) \cdot O(n) = O(n),
\end{aligned}$$

where we have used both $f(x) = x \log x$ and $f(x) = x$ in Eqn. (1). Note that in the first line, the expectation on the left includes the random choice of R_i; this has disappeared on the right after replacing $r_\Delta \log r_\Delta$ with its expectation over the random choice of R_i given the outcome R^+_{i-1} of the first $i - 1$ random choices. Finally, since the number of rounds is $O(\log^* n)$, the total expected time required by the algorithm is $O(n \log^* n)$.

3 Conformal Decomposition

Our algorithm applies sampling to subchains of the original polygonal chain, rather than individual edges. In order 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 $\mathcal{T}(K)$ to denote the trapezoidation $\mathcal{T}(R)$. We also use here the algorithm `Basic-Trapezoidation` to compute $\mathcal{T}(K)$ and so, as already discussed, we have available an efficient planar subdivision representation of $\mathcal{T}(K)$ (which allows the efficient traversal of $\mathcal{T}(K)$ needed below). For our application, we need a planar subdivision with $O(|K|)$ faces, each of which is *conformal* [3], that is, bounded by portions of at most $O(1)$ chains in K and at most $O(1)$ vertical rays determined by their vertices. We obtain this subdivision *retraction* by selecting certain rays of $\mathcal{T}(K)$. The *candidate rays* for this selection are those *ray-pairs* (one ray upward and one ray downward) incident to a *reflex* vertex, that is, 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. 5(a).)

  2. For each face $f$ of $\overline{T}$ and a candidate ray-pair $\rho$ in $f$, select $\rho$ if each of the
     three faces in which $\rho$ 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. 5(b).)
```

Figure 4: Conformal decomposition procedure.

The procedure is summarized in Fig. 4. 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 $\overline{T}$ (induced by chains in K and these selected ray-pairs) are simply connected (see Fig. 5(a)). This is because if one such face f had 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. 5(b)). For each face f of $\overline{T}$, Step 2 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 next.

For a face f, let τ_f be the dual (adjacency) tree of the portion of $\mathcal{T}(K)$ in f. Consider τ_f rooted at an arbitrary *node-trapezoid*. The rooting naturally determines for each node-trapezoid a corresponding "region below" it that includes the trapezoid itself, and a "region above" it (that excludes the trapezoid). In other words, the vertical ray between a trapezoid and its parent separates the two regions. Perform a depth first tree traversal that *marks* a node-trapezoid blue if its region below has a ray selected in Step 1 on its boundary: After visiting the children of the trapezoid, the trapezoid is marked blue if either of its children was marked blue or if it has a selected ray on its boundary. Another tree traversal marks a node-trapezoid red (without erasing the blue marks) if the region above has a selected ray on its boundary: Mark a node-trapezoid red if either its parent is marked red or one of its siblings is marked blue; the root is marked 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 its marks and those of its children indicate that the three faces in which the ray-pair splits f has a ray selected in Step 1 on its boundary (we omit the listing of the cases).

Clearly, the complete procedure runs in $O(|R|)$ time. Moreover, the total number of

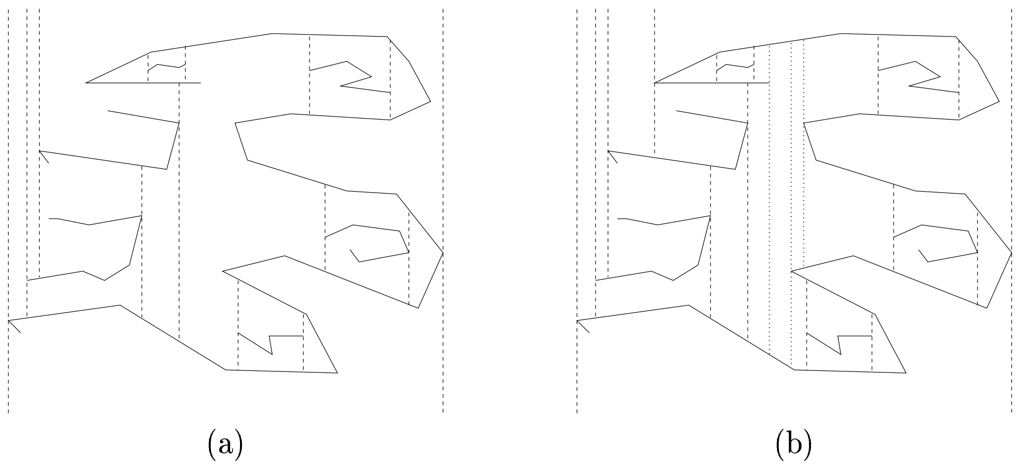
Figure 5: Conformal decomposition example: (a) Subdivision after Step 1, and (b) subdivision after Step 2 (new edges are dotted).

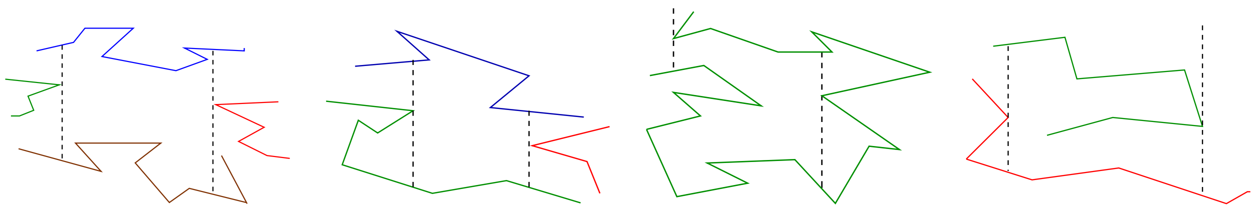
Figure 6: Some example chain-trapezoids. A chain-trapezoid is determined by up to four distinct chains, but can be determined by a single one as in the third example. Also, because ray-pairs from endpoints are not necessarily selected, the situation in the fourth example can occur.

selected ray-pairs is $O(|K|)$: It is clear that $\overline{T}$ 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. 6.

Let $\widetilde{\mathcal{T}}(K)$ be the collection of all the conformal faces. Therefore, we have the following:

Lemma 3.1 *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 the procedure* `conformal` *constructs in $O(|R|)$ time a conformal subdivision $\widetilde{\mathcal{T}}(K)$ containing $O(|K|)$ faces.*

We refer to the selected (single) rays as *portals*, to the conformal faces as *chain-trapezoids* (as they are defined by chains rather than by edges), and to the conformal decomposition $\widetilde{\mathcal{T}}(K)$ as the *chain-trapezoidal* decomposition or *chain-trapezoidation.*

4 The Linear-Time Algorithm

Our trapezoidation algorithm can be viewed as a refinement of the non-optimal algorithm of Section 2, 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 a p-sample $K \subseteq L$ is obtained by taking each $\ell \in L$ into K with probability p independently. For each chain-trapezoid $\widetilde{\Delta}$ in the chain-trapezoidation $\widetilde{\mathcal{T}}(K)$, let $L_{|\widetilde{\Delta}}$ denote the set of subchains in L that intersect (the interior of) $\widetilde{\Delta}$. Let $n = |L|$, $\widetilde{n}_{\widetilde{\Delta}} = |L_{|\widetilde{\Delta}}|$. In analogy with Eqn. (1), one would conjecture a bound

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

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 5 we prove a weaker one that is sufficient for our purposes (there is an extra $\log\lambda$ factor on the left hand side). This is a first step towards obtaining a linear time algorithm.

At the same time, our algorithm can also be viewed as a simplification of Chazelle's algorithm. Like Chazelle's algorithm, our construction follows a pattern in which first a bottom-up phase develops a successively more global but coarse approximation to the trapezoidation, and then a top-down phase successively refines this coarse approximation into the final trapezoidation. Both consider a subdivision of the input chain into successively finer subchains. This sequence of chain refinements is referred to as a *refinement tree*. However, while for Chazelle's algorithm the approximation is the analog of our chain-trapezoidation (a *granular* and *conformal submap*, in his terms, and similar to our chain-trapezoidations) for *all* the subchains in each level starting with the coarser level, for our algorithm the approximation is the chain-trapezoidation for a *small* random sample of the subchains at each level. The resulting hierarchy of samples is called a *gradation tree*, or just gradation. Using only a random sample implies less work to do, and it can then be performed with simpler methods. In this section, we first define precisely the sequence of chain refinements and the corresponding gradation tree of samples, then we give an outline of the two phases of the algorithm, and finally describe the top-down construction phase and the bottom-up preprocessing phase.

4.1 Chain Refinements and Gradation of Samples

The sampling in our algorithm is performed on a sequence of chain refinements, or *refinement tree*, with $O(\log^* n)$ levels defined as follows (Chazelle's algorithm uses $O(\log n)$ levels). Let ℓ_0 be the initial simple polygonal chain of size n. 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 $\ell \in L_{i-1}$ into a set L_i^ℓ of subchains each of size

$\lambda_i = \log^2 \lambda_{i-1}$, and ending with $k = O(\log^* n)$ so that $\lambda_k = O(1)$.[2] Thus, the subchains in the i-th level of the refinement tree are $L_i = \bigcup_{\ell \in L_{i-1}} L_i^\ell$. We denote the total number of subchains in L_i by $\widetilde{n}_i = |L_i| = n/\lambda_i$.

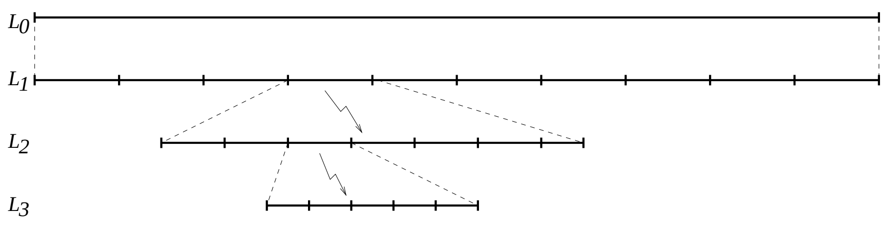

Figure 7: Gradation of subchains.

Instead of attempting to compute $\widetilde{\mathcal{T}}(L_i)$ directly (the analog of what Chazelle's algorithm does), our algorithm further 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 [8, 12, 18]. Specifically, for each $i \geq 1$, we choose a global probability $p_i = 1/\log^3 \lambda_{i-1}$, and let K_i be a p_i-sample from L_i. 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,

$$K_i^+ = K_i \cup \{\ell | \ell \in L_i \text{ and } \ell \subset \ell' \text{ where } \ell' \in K_j, j < i\}.$$

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 5, it will follow that adding the subchains of previous samples does not affect substantially the randomness of the sample K_i. The resulting hierarchy of samples is called a *gradation tree*, or just gradation.

4.2 Overview of the Algorithm

As mentioned previously, our polygon trapezoidation algorithm consists of two phases. The main phase proceeds top-down constructing the decompositions $\widetilde{\mathcal{T}}(K_i^+)$ iteratively. For each chain trapezoid $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^+)$, the algorithm maintains its *chain-conflict* list $L_{i|\widetilde{\Delta}} \subseteq L_i$, that is, the set of subchains in L_i that intersect (the interior of) $\widetilde{\Delta}$. Maintaining chain-conflict lists, rather than edge-conflict lists, is essential to the efficiency of our algorithm. At the beginning of the i-th round, we have $\widetilde{\mathcal{T}}(K_{i-1}^+)$ and the chain-conflict lists $L_{i-1|\widetilde{\Delta}}$ for each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$, then the algorithm adds K_i to $\widetilde{\mathcal{T}}(K_{i-1}^+)$ to obtain $\widetilde{\mathcal{T}}(K_i^+)$, and computes the new chain-conflict lists by following the chain ℓ_0 without actually scanning every edge. In

[2]Note that if, in analogy with $\log^* n$, we define $\log^{2(i)} n$ by iterating the $\log^2$ operation, and then $\log^{2*} n$, one obtains that $\log^{2*} n = O(\log^* n)$.

```
Top-Down ($i$-th round)
 Input: $\widetilde{\mathcal{T}}(K_{i-1}^+)$ and its conflicts with $L_{i-1}$
 Output: $\widetilde{\mathcal{T}}(K_i^+)$ and its conflicts with $L_i$

  1. For each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$
      a. Determine $K_{i|\widetilde{\Delta}}$
      b. $T_{\widetilde{\Delta}} \leftarrow \begin{cases} \mathcal{T}(K_{i|\widetilde{\Delta}} \cup \{\ell_1, \ell_2\}) \text{ restricted to } \widetilde{\Delta}, \\ \text{where } \ell_1, \ell_2 \text{ are the chains that bound } \widetilde{\Delta} \end{cases}$
  2. Merge all the $T_{\widetilde{\Delta}}$, $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$, into $\mathcal{T}(K_i^+)$
  3. Obtain $\widetilde{\mathcal{T}}(K_i^+)$ using conformal($K_i^+, \mathcal{T}(K_i^+)$)
  4. 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 8: Top-down phase procedure.

a preprocessing phase, the algorithm constructs for each chain $\ell \in L_i$, $i = 1, \ldots, k$, a data structure $\mathcal{D}(\ell)$ that supports portal-chain intersection queries: given a portal ρ, determine whether the chain ℓ intersects ρ. These queries are needed for the efficient computation of chain-conflict lists during the construction phase. These data structures also support ray-shooting queries (given a point x, determine the lowest point in ℓ hit by a vertical ray upward from x) which can then be used for testing whether a query point is contained in a chain-trapezoid (perform ray-shooting queries on the two bounding chains and determine if the result corresponds to hitting them from inside the chain-trapezoid).

The construction (and also the query processing) of $\mathcal{D}(\ell)$ uses recursively the data structures $\mathcal{D}(\ell')$ for all children ℓ' of ℓ in the refinement tree. We will see that as a result the construction reduces to computing, for each i from k down to 0 and each chain $\ell \in L_{i-1}$, the chain-trapezoidation of the random sample K_i^ℓ of its subchains, together with its conflict lists with L_i^ℓ. So the computation of the $\mathcal{D}(\ell)$'s corresponds to the bottom-up phase discussed above. In contrast, the actual construction proceeds in a top-down manner: for $i = 0$ up to $i = k$, it computes the chain-trapezoidation of the random sample K_i, together with its conflict lists with respect to L_i.

4.3 Top-Down Construction Phase

We now precisely describe how our algorithm performs the top-down construction phase. In the i-th round, given the decomposition $\widetilde{\mathcal{T}}(K_i^+)$ 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. 8.

Step 1.a 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 $\widetilde{\Delta}$. Both queries are solved by the same data structures $\mathcal{D}$ constructed during the preprocessing phase. However, note that the point location query is on chains in L_{i-1} and, consequently, it is more expensive (we could afford to construct a faster standard point location data structure for $\widetilde{\Delta}$, but it is not necessary). Let $\widetilde{r}_{i,\widetilde{\Delta}} = |K_{i|\widetilde{\Delta}}|$. Step 1.b computes the trapezoidation $\mathcal{T}(K_{i|\widetilde{\Delta}} \cup \{\ell_1, \ell_2\})$ restricted to $\widetilde{\Delta}$, where ℓ_1 and ℓ_2 are the two chains bounding $\widetilde{\Delta}$. This uses a simple algorithm with running time $O(r_{i,\widetilde{\Delta}} \log r_{i,\widetilde{\Delta}})$ [8, 12, 18], where $r_{i,\widetilde{\Delta}} = O((\widetilde{r}_{i,\widetilde{\Delta}} + 1)\lambda_i)$ is the number of edges involved in all these chains (the plus 1 accounts for ℓ_1 and ℓ_2). Step 2, with a simple traversal of all the $T_{\widetilde{\Delta}}$'s, "stitches" together trapezoids of $\mathcal{T}(K_i^+)$ "chopped" by the portals of $\widetilde{\mathcal{T}}(K_{i-1}^+)$; this takes time linear in the total size of the $T_{\widetilde{\Delta}}$. The procedure `conformal` in Step 3 was described in Section 3; 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 4, 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 next.

Hopping. If a chain is already in K_i^+, then it is part of the boundary for some chain-trapezoids and it can automatically be recorded as part of their conflict lists. So, consider some $\ell \in L_i \setminus K_i^+$ and suppose that we know a chain trapezoid $\widetilde{\Delta}_0 \in \widetilde{\mathcal{T}}(K_i^+)$ that contains the first endpoint e of ℓ. Note that the chain-trapezoids in $\widetilde{T}_i$ that conflict with ℓ are connected. Thus, we perform a breadth-first-search traversal of the adjacency graph of $\widetilde{\mathcal{T}}(K_i^+)$, starting with $\widetilde{\Delta}_0$. When a chain-trapezoid $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_i^+)$ is visited, it is labeled as a *conflict* and each of the portals that separates $\widetilde{\Delta}$ from an unvisited chain-trapezoid $\widetilde{\Delta}' \in \widetilde{\mathcal{T}}(K_i^+)$ 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. 9.

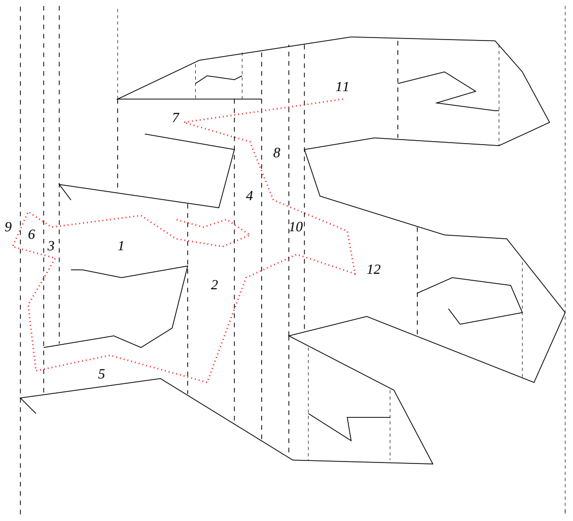

Figure 9: Computing the chain-conflicts for a chain. Note that only one conflict per portal is found, and that the order the conflicts are discovered (indicated by numbers) does not necessarily reflect their occurrences on the chain. The portals not queried are shown lighter.

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.

Running Time. In Sec. 6, using the sampling bounds obtained in the next section, we show 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 $O(n)$.

4.4 Bottom-Up Preprocessing 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 sequence of chain refinements 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 $K_i^\ell = L_i^\ell \cap K_i$. Note that since K_i is a p_i-sample from L_i, then K_i^ℓ is a p_i-sample from L_i^ℓ.[3]

For each ℓ in L_{i-1}, $i = 1, \ldots, k$, we construct a data structure $\mathcal{D}(\ell)$ that consists of:

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

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

For the construction of $\widetilde{\mathcal{T}}(K_i^\ell)$, we use `Basic-Trapezoidation`, which also produces the required point location data structure.

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 and every subchain ℓ' that bounds $\widetilde{\Delta}$ or is in $\widetilde{\Delta}$'s conflict list $L^\ell_{i|\widetilde{\Delta}}$, which includes the subchains that bound $\widetilde{\Delta}$ (see Fig. 11). This query procedure is summarized in Fig. 10. The query procedure for ray-shooting, which determines the lowest intersection point, is similar and we omit it.

The computation of conflict lists for $\widetilde{\mathcal{T}}(K_i^\ell)$ during the construction of $\mathcal{D}(\ell)$ also uses hopping and, hence, `conflict?` recursively. The bottom-up preprocessing phase is summarized in Fig. 12.

Running Time. In Sec. 6, using the sampling bounds obtained in the next section, we show that the construction of the data structures $\mathcal{D}(\ell)$ is completed in expected $O(n)$ time. Moreover, we show that, even though we recurse on each subchain in a conflict list, 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. Thus, we can summarize our results in the following theorem.

[3]This phase could use a sampling independent of that in the construction phase, but this is not necessary.

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

Input: A portal $\rho$ and a chain $\ell$ in $L_{i-1}$.
Output: Yes if $\rho$ intersects $\ell$, else No

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

Figure 10: Portal-chain query procedure.

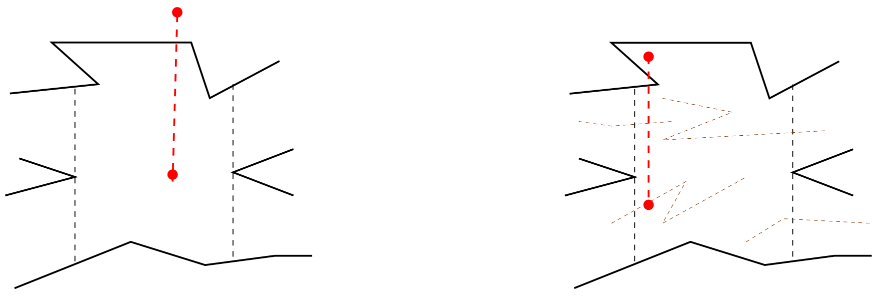

Figure 11: Two cases in the portal-chain conflict query: The portal endpoints are in different or in the same chain-trapezoid.

Theorem 4.1 *Our randomized two-phase algorithm constructs the trapezoidation of a simple polygon of size n in expected $O(n)$ time.*

5 Sampling Bounds

To analyze the running time of our 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 $\widetilde{n}_{\widetilde{\Delta}} = |L_{|\widetilde{\Delta}}|$. Unfortunately, we cannot prove the bound in Eqn. (2). Such a bound can be proved in the framework of *configuration spaces*, when certain *locality* [8, 19] or *monotonicity* [10, 1] properties hold for the decomposition induced by the sample (see [17] for a survey), but neither of these properties hold for our chain-trapezoidation. Fortunately, we can prove a weaker bound that is only a factor $O(f(\log \lambda))$ larger, and that suffices to verify that our algorithm has expected linear running time. The proof of the bound uses a standard trick [8, 4]: one

```
Bottom-Up ($i$-th round)
  Input: $\widetilde{\mathcal{T}}(K_j^\ell)$ and its conflicts with $L_j^\ell$ for each $\ell \in L_{j-1}$, $j > i$
  Output: $\widetilde{\mathcal{T}}(K_i^\ell)$ and its conflicts with $L_i^\ell$ 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 $L_i^\ell$ in $\widetilde{\mathcal{T}}(K_i^\ell)$
```

Figure 12: Bottom-up phase procedure.

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.

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 chain-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 our 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 following *locality* property:

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

Though our chain-trapezoidation lacks locality, or even monotonicity, the following lemma states that we choose it out of a relatively small "pool" of candidates that satisfy the locality property.

Lemma 5.1 *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 the adjacency graph of $\mathcal{T}(K)$. For each $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K)$ consider the maximum region $R_{\widetilde{\Delta}}$ of $\mathcal{T}(K)$ that contains $\widetilde{\Delta}$ and is bounded by the same one or two chains that bound $\widetilde{\Delta}$. Note that $R_{\widetilde{\Delta}}$ 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 $O(|K|\lambda^2)$. ◻

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.

Lemma 5.2 *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*

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

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,

$$\begin{aligned}
\mathrm{Prob}\{\widetilde{\Delta} \in \widetilde{\mathcal{T}}^c(K)\} &= p^{\delta(\widetilde{\Delta})}(1-p)^{\widetilde{n}_{\widetilde{\Delta}}} \\
&= 2^{\delta(\Delta)}\cdot\left(\frac{1-p}{1-p/2}\right)^{\widetilde{n}_{\widetilde{\Delta}}}\cdot(p/2)^{\delta(\widetilde{\Delta})}(1-p/2)^{\widetilde{n}_{\widetilde{\Delta}}} \\
&\le 16\cdot e^{-p\widetilde{n}_{\widetilde{\Delta}}/2}\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\},
\end{aligned} \tag{5}$$

using $\delta(\widetilde{\Delta}) \le 4$ and $(1-p)/(1-p/2) \le 1-p/2 \le e^{-p/2}$. Using Eqn. (5) and Lemma 5.1, we obtain

$$\begin{aligned}
\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)} f(p\widetilde{n}_{\widetilde{\Delta}})\right] &= \sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} \\
&= O\left(\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)}\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\}\right) \\
&= O(\mathbf{E}[|\widetilde{\mathcal{T}}^c(K')|]) \\
&= O(\widetilde{r}\lambda^2),
\end{aligned} \tag{6}$$

using Lemma 5.1. Let $\tau > 0$ be a parameter. Then, using again Eqn. (5),

$$\begin{aligned}
\mathbf{E}\left[\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\right] &= \sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)\\ \widetilde{n}_{\widetilde{\Delta}}>2(\log\tau)/p}} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K)\} \\
&\le 16\cdot\sum_{\substack{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)\\ \widetilde{n}_{\widetilde{\Delta}}>\frac{2(\log\tau)}{p}}} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot e^{-p\widetilde{n}_{\widetilde{\Delta}}/2}\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\} \\
&\le \frac{16}{\tau}\cdot\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^*(L)} f(p\widetilde{n}_{\widetilde{\Delta}})\cdot\mathrm{Prob}\{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')\} \\
&= \frac{16}{\tau}\cdot\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}^c(K')} f(p\widetilde{n}_{\widetilde{\Delta}})\right]
\end{aligned}$$

$$= \frac{O(1)}{\tau} \cdot \mathbf{E}[|\widetilde{\mathcal{T}}^c(K'')|] = O\left(\widetilde{r} \cdot \frac{\lambda^2}{\tau}\right),$$

where K'' is a $(p/4)$-sample and we have used the bound in Eqn. (6). Finally, using $\tau = \lambda^2$,

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

$\square$

We have not tried to determine the weakest (less stringent) conditions for the function f under which the lemma holds.

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. This is established in the following lemma.

Lemma 5.3 *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

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

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

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

$\square$

6 Running Time Analysis

First, we note that the sampling bound of Lemma 5.2 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 4, 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):

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

and for $\alpha, \beta > 0$,

$$\mathbf{E}\left[\sum_{\widetilde{\Delta}\in\widetilde{\mathcal{T}}(K)} (\alpha\widetilde{n}_{\widetilde{\Delta}}+\beta)\log((\alpha\widetilde{n}_{\widetilde{\Delta}})+\beta)\right] = O\left(p\widetilde{n}\cdot\left(\frac{\alpha\log\lambda}{p}+\beta\right)\cdot\log\left(\frac{\alpha\log\lambda}{p}+\beta\right)\right). \tag{8}$$

6.1 Preprocessing 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

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

where ϵ is any positive fraction. We have used Lemma 5.3 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. (7), 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 $O(n)$.

6.2 Construction 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. (7), the total expected time for this step is at most proportional to

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

(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 $\widetilde{\Delta} \in \widetilde{\mathcal{T}}(K_{i-1}^+)$.) In Step 1.b, $T_{\widetilde{\Delta}}$ is computed using an algorithm with running time $O(r_{i,\widetilde{\Delta}} \log r_{i,\widetilde{\Delta}})$ where $r_{i,\widetilde{\Delta}} = O((\widetilde{r}_{i,\widetilde{\Delta}} + 1)\lambda_i)$. Using Eqn. (8), the total expected time is at most proportional to

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

The conformal decomposition in Step 3 is constructed as described in Section 3 and requires expected time at most proportional to

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

In Step 4, 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 $O(n)$.

7 Concluding Remarks

We have presented a randomized algorithm for computing the trapezoidation of a simple polygon, and hence a triangulation, that runs in expected time that is linear in the number of edges. The algorithm is considerably simpler than Chazelle's algorithm. On the other hand, it is comparatively more complicated than the non-optimal randomized algorithms and, since for any practical value of n, $\log^* n$ is a small constant, our algorithm is not likely to be of practical value. An incremental version of our algorithm (like Seidel's) is possible. In such a version, the edges are added one by one to the previous trapezoidation according to a random permutation which is somewhat modified according to the chain sampling. In this way some gain in simplicity is achieved. Also, we point out that, unlike Seidel's algorithm, the number of rounds does not have to be $O(\log^* n)$.

We conclude by mentioning some questions that remain open. Is the conjectured tighter sampling bound for our conformal decomposition true? Is it possible, as conjectured by Bernard Chazelle, to combine our 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 $O(n + k)$? Can our linear time algorithm be parallelized? (Goodrich [13] has given a fast and work optimal implementation of Chazelle's algorithm in a parallel computation model.) Can the approach of sampling on subchains lead to efficient algorithms for other problems on simple polygons? Finally, do simpler deterministic or randomized algorithms for the polygon triangulation problem exist?

Acknowledgements

We would like to thank Bernard Chazelle for useful comments and discussions.

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] T. Asano, T. Asano, and R. Y. Pinter. Polygon triangulation: Efficiency and minimality. *J. Algorithms*, 7:221–231, 1986.

[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] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995.

[14] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[15] J. Hershberger. Optimal parallel algorithms for triangulated simple polygons. *Internat. J. Comput. Geom. Appl.*, 5:145–170, 1995.

[16] 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.

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

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

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

[20] 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.

[21] 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.