A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons*

Raimund Seidel†
Computer Science Division
University of California Berkeley
Berkeley CA 94720
USA

seidel@cs.berkeley.edu

Abstract

This paper presents a very simple incremental randomized algorithm for computing the trapezoidal decomposition induced by a set S of n line segments in the plane. If S is given as a simple polygonal chain the expected running time of the algorithm is $O(n \log^* n)$. This leads to a simple algorithm of the same complexity for triangulating polygons. More generally, if S is presented as a plane graph with k connected components, then the expected running time of the algorithm is $O(n \log^* n + k \log n)$. As a by-product our algorithm creates a search structure of expected linear size that allows point location queries in the resulting trapezoidation in logarithmic expected time. The analysis of the expected performance is elementary and straightforward. All expectations are with respect to "coinflips" generated by the algorithm and are not based on assumptions about the geometric distribution of the input.

*Large Portions of the research reported here were conducted while the author visited the Free University Berlin.
†Supported by NSF grant CCR-8809040 and Presidential Young Investigator Award CCR-9058440.

1 Introduction

Polygon triangulation has been a problem of great appeal to computational geometers: It can be easily stated as "given the coordinates of the n vertices of a simple polygon P in order around P, find $n-3$ diagonals that partition P into $n-2$ triangles;" it is of importance in various application areas; and finally, the actual computational complexity of the problem remained unresolved for more than a decade until very recently.

A brief history: Garey et al. [GJPT] were the first to publish an $O(n \log n)$ algorithm based on sweeping in 1978. Four years later another algorithm with the same complexity was published by Chazelle [C82]. The $O(n \log n)$ bound was then improved to bounds of the form $O(n \log C_P)$, where C_P is a "shape" parameter no bigger than n that depends on the polygon P to be triangulated (for instance the number of reflex vertices [HM],[FM], or the "sinuosity" of P [CI]). On a different front an ever-increasing class of polygons were shown to be triangulatable in linear time [TA82], [T88]. After a false start, Tarjan and Van Wyk [TV] made a major breakthrough with an $O(n \log \log n)$ algorithm in 1986. This time bound was matched by a different but simpler algorithm by Kirkpatrick et al. [KKT] three years later. In the mean time Clarkson et al. had published a randomized algorithm with $O(n \log^* n)$ expected running time [CTV]. Finally in 1990 Chazelle discovered a linear time deterministic algorithm [C90], which settles the question about the intrinsic computational complexity of triangulating once and for all.

This paper presents another randomized algorithm with $O(n \log^* n)$ expected running time. Its virtues lie in its simplicity. It uses no divide-and-conquer or recursion, and no "Jordan-sorting" [CTV],[HMRT]. Its expected performance admits a very straightforward and self-contained analysis. Finally, it is practical and relatively simple to implement, a property that very few, if any, of the algorithms mentioned can claim.

Like most of its predecessors our algorithm does not triangulate a polygon P directly, but rather it computes what is known as the "horizontal visibility map" or "trapezoidal decomposition" of P. By an observation of Fournier and Montuno [FM] it is easy to derive from such a decomposition of P in linear time a triangulation of P.

Section 2 of this paper gives some basic definitions and facts about trapezoidal decompositions and how they relate to triangulations. Section 3 describes the new algorithm.

2 Trapezoidations

Our setting is the Euclidean plane with the usual Cartesian x-y coordinate system. We will feel free to use intuitive notions such as "vertical" (meaning parallel to the y-axis), "horizontal," "left of," etc.

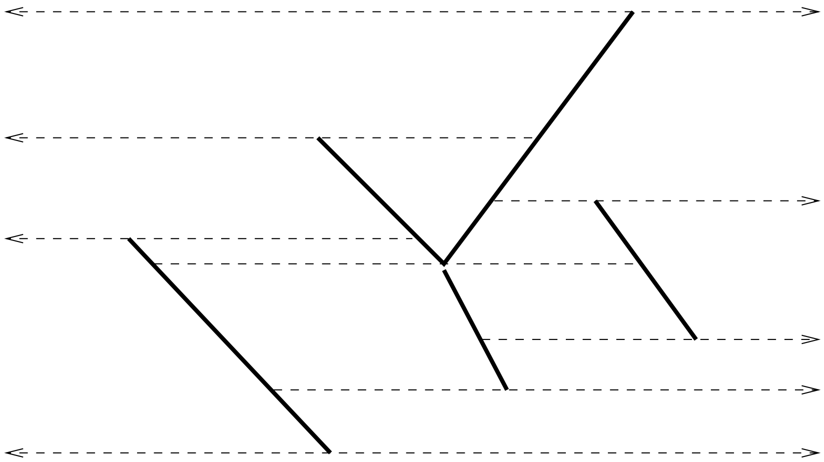

Figure 1: A trapezoidation of 5 segments

We call two straight line segments in the plane *non-crossing* iff their intersection is either empty or a common endpoint. Consider a set S of n non-horizontal, non-crossing closed line segments. Starting at each endpoint of each segment in S draw two horizontal rays, one towards the left and one towards the right, each extending until it hits a segment of S. For a segment endpoint p we call the union of these two possibly truncated rays emanating from p the *horizontal extension* through p. The segments of S together with the horizontal extensions through the endpoints form a plane graph, which we call the *trapezoidation* of S, or $\mathcal{T}(S)$ for short (see Figure 1). As each face of $\mathcal{T}(S)$ has two horizontal sides (one of which might have length 0) we are justified in calling the faces of $\mathcal{T}(S)$ *trapezoids*.

What is the complexity of $\mathcal{T}(S)$? If S consists of n segments, then there are at most $2n$ segment endpoints. Through each endpoint there is a horizontal extension whose endpoints in turn can contribute at most two vertices to $\mathcal{T}(S)$. Thus $\mathcal{T}(S)$ has at most $6n$ vertices, and since it is a planar graph it follows immediately that it has $O(n)$ edges and faces. In particular, the number of faces (trapezoids) is at most $3n + 1$, which can be easily established using a sweep argument.

For our purposes it would be advantageous if each trapezoid of $\mathcal{T}(S)$ had at most two "neighboring" trapezoids above it and at most two neighboring trapezoids below it. By this we mean that if a trapezoid has a non-degenerate upper (lower) side, then this side should have non-empty intersection with the lower (upper) sides of at most two other trapezoids. This condition holds automatically if no two distinct endpoints of segments in S have the same y-coordinate. From now on we will assume that S satisfies this "non-degeneracy" assumption. As noted already in [CTV], no generality is lost with this assumption, since it can always be achieved by rotating the coordinate system by an sufficiently small amount. Better yet, this perturbative rotation can simply be simulated using a lexicographic technique: if two distinct points have the same y-coordinate, then the one with smaller x-coordinate is considered the "lower" one (see Figure 2).

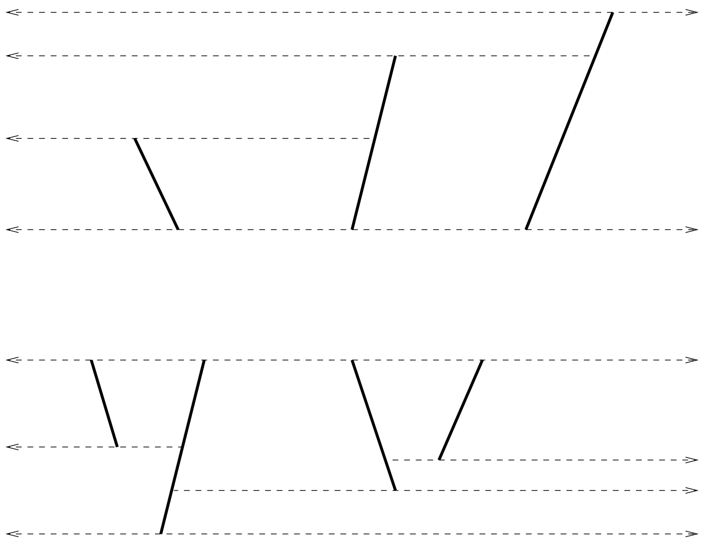

Figure 2: (a) A “degenerate” trapezoidation

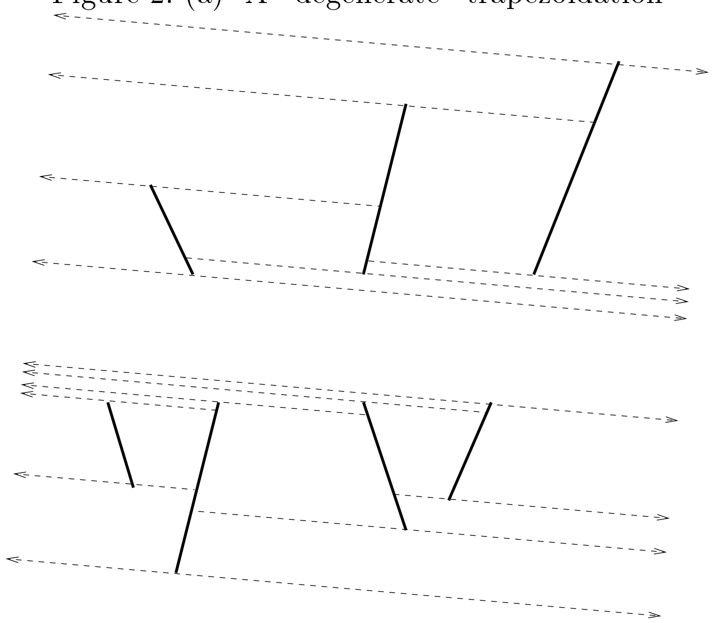

Figure 2: (b) Degeneracies removed by rotation of coordinate system (or lexicographic method)

We end this section by giving the connection between computing trapezoidations and triangulating simple polygons. This connection was originally established in [FM] and [CI]. We include it for the sake of completeness.

Let $S = \{s_0, s_1, \ldots s_{n-1}\}$ be a set of n segments that form a simple and closed polygonal curve; i.e. s_i and s_{i+1} share a common endpoint, and $s_i \cap s_j = \emptyset$ iff $|i - j| > 1$ (all index arithmetic here is modulo n). This polygonal curve bounds a simple polygon P with n vertices. *Triangulating* P amounts to finding a set of $n-3$ non-crossing diagonals of P that partition P into $n-2$ triangles.

Lemma 1 *Let $S = \{s_0, s_1, \ldots s_{n-1}\}$ be the set of edges around a simple polygon P. If $\mathcal{T}(S)$ is available, then a triangulation of P can be computed in $O(n)$ time.*

Proof: (Sketch) Assume that no two vertices of P have the same y-coordinate. As mentioned above, this assumption entails no loss of generality.

We compute a triangulation of P from a trapezoidation $\mathcal{T}(S)$ in three stages:

First remove from consideration all trapezoids of $\mathcal{T}(S)$ that do not lie in the interior of P.

Second, for each of the remaining trapezoids check whether it has two vertices of P on its boundary that do not lie on the same side. If such a pair of vertices exists draw a diagonal between them.

The diagonals introduced in the second phase partition P into a number of subpolygons, each of which has a very special form: its boundary consists of two y-monotone chains, one of which is a single edge. A polygon of such a form can easily be triangulated in linear time by repeatedly "cutting off" convex corners of the y-monotone chain (see Figure 3). For more details see [FM],[CI]. ■

3 The Algorithm

For the purposes of this section we assume a representation of the trapezoidation $\mathcal{T}(S)$ that allows for each trapezoid $\tau \in \mathcal{T}(S)$ to determine in constant time the segments of S that bound τ to its left and to its right (if any), the (up to two) adjacent trapezoids of $\mathcal{T}(S)$ above τ, and the (up to two) adjacent trapezoids below τ. Here we assume non-degeneracy in the sense outlined in the previous section, namely that no two non-identical segment endpoints have the same y-coordinate. Again, this condition can easily be simulated using lexicographic techniques. Our representation allows efficient navigation through $\mathcal{T}(S)$, i.e. tracing a curve C through $\mathcal{T}(S)$ at cost proportional to the complexity of C plus the number of trapezoid traversals, provided C crosses no segment of S. Note that the size of such a representation is linear in the number of segments of S. Let $\mathcal{Q}(S)$ be a point location query structure for $\mathcal{T}(S)$, i.e. $\mathcal{Q}(S)$ is a directed acyclic graph with one source and with exactly one sink for each trapezoid of $\mathcal{T}(S)$. Each non-sink node has outdegree 2 and is

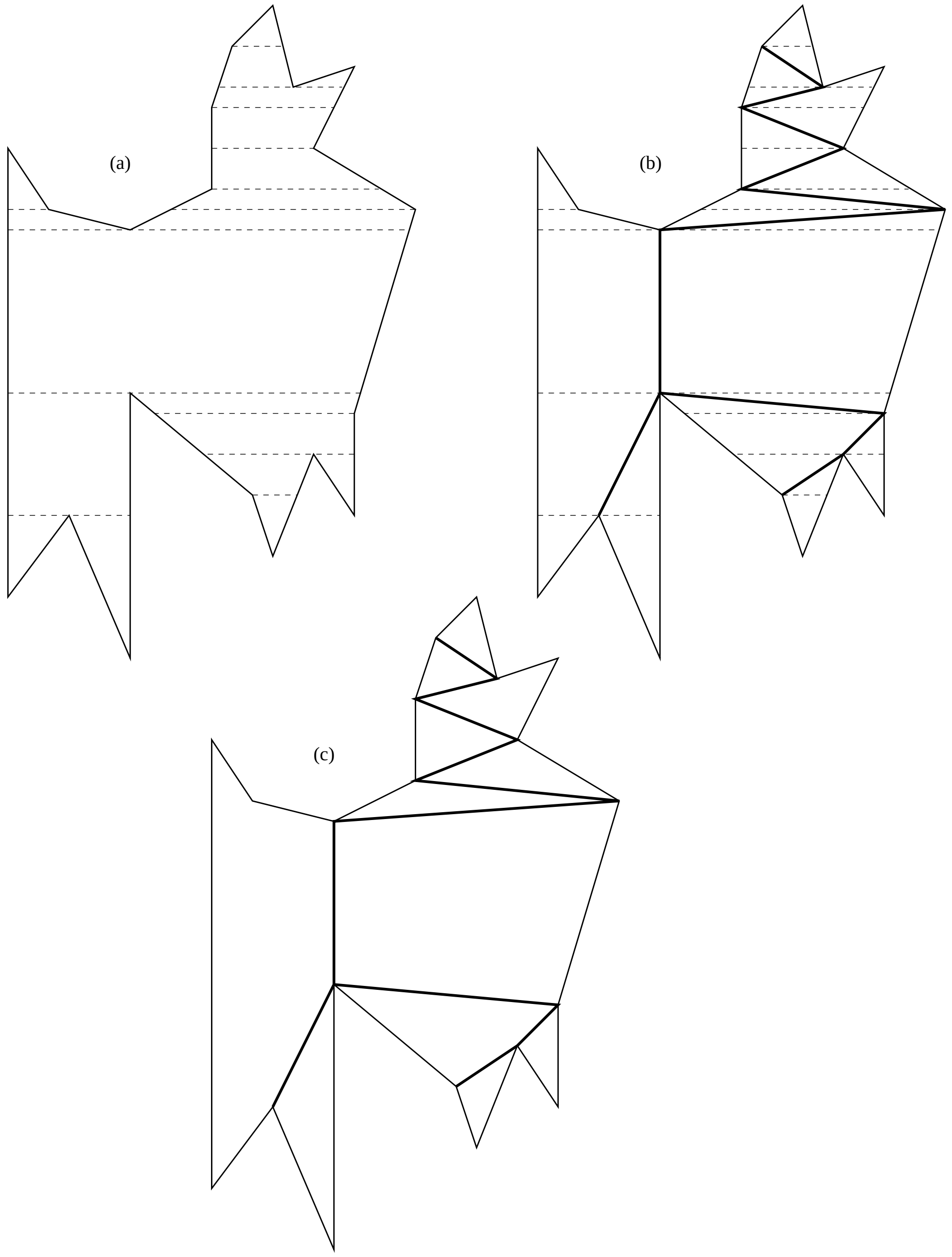

Figure 3: (a) Trapezoids inside the polygon; (b) Introducing diagonals; (c) Decomposition of polygon into pieces that are easy to triangulate.

labeled either X, in which case it has a segment of S associated with it as a key, or the node is labeled Y, in which case its associated key is a real number, namely the y-coordinate of an endpoint of some segment in S (or in other words, the horizontal extension through the endpoint). A query with query point q is supposed to proceed as follows: It starts at the source of $\mathcal{Q}(S)$ and proceeds along a directed path to some sink whose corresponding trapezoid of $\mathcal{T}(S)$ is to contain q. At each non-sink node along the way the decision which of the two outedges to follow is dictated by the outcome of the comparison of q with the key (At an X-node: Is q left or right of the key segment? At a Y-node: Is q's y-coordinate less or greater than the key? or: Does q lie below or above the horizontal extension?). We assume that the trapezoids of $\mathcal{T}(S)$ and the sinks of $\mathcal{Q}(S)$ are properly correlated, i.e. for a trapezoid of $\mathcal{T}(S)$ one can tell in constant time its corresponding sink of $\mathcal{Q}(S)$, and vice versa (see Figure 4).

Let s be a non-horizontal segment with upper endpoint a and lower endpoint b that crosses no segment of S, and let S' be $S \cup \{s\}$. We consider the problem of computing $\mathcal{T}(S')$ and $\mathcal{Q}(S')$, given $\mathcal{T}(S)$ and $\mathcal{Q}(S)$.

If the upper endpoint a is not an endpoint of some segment in S already, we first use $\mathcal{Q}(S)$ to locate the trapezoid of τ_a of $\mathcal{T}(S)$ that contains a. We split τ_a with a horizontal line through a to obtain a new trapezoidal decomposition $\mathcal{T}'$. The sink of $\mathcal{Q}(S)$ that corresponds to τ_a becomes a Y-node whose key is the y-coordinate of a and whose two successors are two new nodes, which are sinks and correspond to the two new trapezoids of $\mathcal{T}'$. Thus we obtained besides $\mathcal{T}'$ also a new query structure $\mathcal{Q}'$.
If a was an endpoint of some segment of S already, then let $\mathcal{T}'$ be $\mathcal{T}$ and let $\mathcal{Q}'$ be $\mathcal{Q}$.

Next we proceed likewise with the lower endpoint b to obtain from $\mathcal{T}'$ and $\mathcal{Q}'$ a trapezoidation $\mathcal{T}''$ and a corresponding query structure $\mathcal{Q}''$.

Now we "thread" segment s through $\mathcal{T}''$, i.e. we determine all trapezoids of $\mathcal{T}''$ that are intersected by s, cut them in two, and on each side of s we merge contiguous sequences of trapezoids that agree in their bounding segments. Thus we have obtained $\mathcal{T}(S')$.

We obtain $\mathcal{Q}(S')$ as follows: For each "new" trapezoid of $\mathcal{T}(S')$ along s we create a new corresponding sink-node. Each sink of $\mathcal{Q}''$ that corresponds to a trapezoid of $\mathcal{T}''$ that is cut by s becomes an X-node whose key is s and whose two successors are the appropriate new sinks.

The time necessary to obtain $\mathcal{T}(S')$ and $\mathcal{Q}(S')$ from $\mathcal{T}(S)$ and $\mathcal{Q}(S)$ consists of the query time for locating the two endpoints of s and the "threading time" for s. Note that the latter is proportional to the number of horizontal lines of $\mathcal{T}(S)$ that are cut by s, or equivalently, it is proportional to the number of horizontal lines of $\mathcal{T}(S')$ that abut upon s.

What is the total time necessary for constructing $\mathcal{T}(S)$ and $\mathcal{Q}(S)$ by successive insertion of the segments of S, starting with empty structures? Clearly this depends

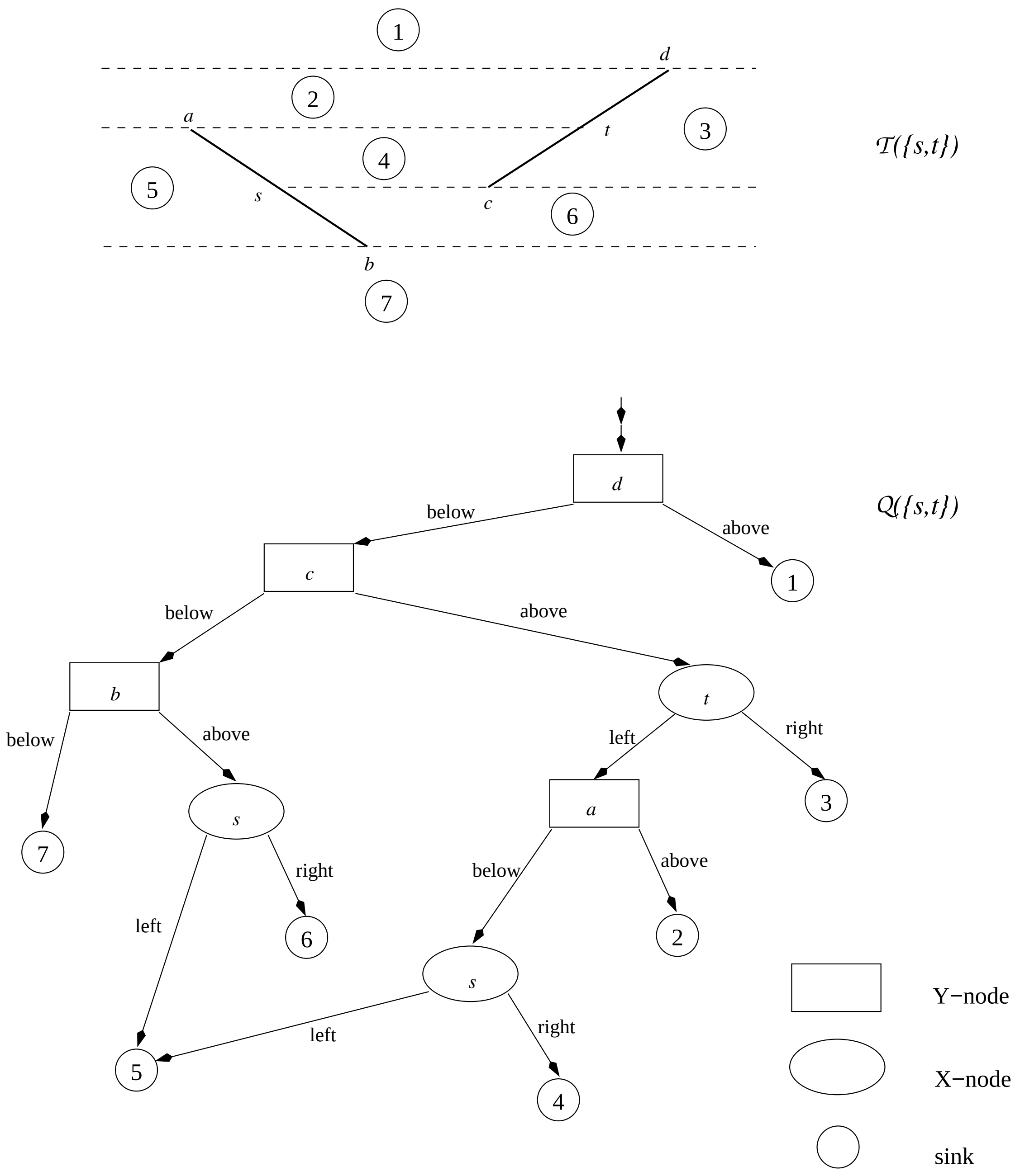

Figure 4: A trapezoidation and an associated query structure

very much on the order of insertion. Note that $\mathcal{T}(S)$ is independent of this order, but $\mathcal{Q}(S)$ and the query time it admits very much depend on this order. It is not hard to concoct bad examples: Consider, for instance, a set S of n vertical segments that all intersect the x-axis. If these segments are inserted by increasing x-coordinate, then the resulting query structure will have very long source–sink paths and locating a query point on the x-axis to the right of all segments will take $O(n)$ time.

In the following we argue that if the segments of S are inserted in *random* order, with each ordering occuring equally likely, then $\mathcal{Q}(S)$ will behave nicely in expectation, and the overall expected construction time will be reasonable. For this purpose we will need to show something about the expected point location query time and the expected "threading" time.

Lemma 2 *Let $s_1, \ldots, s_n$ be a random ordering of the segments of S, and let $S_i = \{s_1, \ldots, s_i\}$ for $0 \leq i \leq n$.*

For $1 < i \leq n$ the expected number of horizontal lines of $\mathcal{T}(S_{i-1})$ that are intersected by the relative interior of s_i is at most 4.

Proof: For a segment $s \in S_i$ let $deg(s, \mathcal{T}(S_i))$ denote the number of horizontal extensions of $\mathcal{T}(S_i)$ that abut upon s_i, i.e. they end in the relative interior of s.

As the number of horizontal extensions of $\mathcal{T}(S_{i-1})$ that are intersected by s_i is the same as the number of horizontal extensions of $\mathcal{T}(S_i)$ that abut upon s_i, we are interested in the expected value $Exp(deg(s_i, \mathcal{T}(S_i)))$.

As there are at most $2i$ horizontal extensions in $\mathcal{T}(S_i)$, each abutting upon at most two segments, we have $\sum_{s \in S_i} deg(s, \mathcal{T}(S_i)) \leq 4i$. Because of the random ordering, s_i is any one of the i segments in S_i with equal probability. Thus $Exp(deg(s_i, \mathcal{T}(S_i))) \leq 4$. ■

Let $H_n = 1 + 1/2 + 1/3 + \cdots + 1/n$, and recall that $H_n = \Theta(\log n)$, in particular for $n > 1$ we have $\log_e n < H_n < 1 + \log_e n$.

Lemma 3 *Let $s_1, \ldots, s_n$ be a random ordering of the segments of S, and let $S_i = \{s_1, \ldots, s_i\}$ for $0 \leq i \leq n$.*

For $1 \leq i \leq n$ let $\mathcal{T}(S_i)$ and $\mathcal{Q}(S_i)$ be the trapezoidation and query structure for S_i obtained from $\mathcal{T}(S_{i-1})$ and $\mathcal{Q}(S_{i-1})$ by inserting segment s_i.

If q is any query point, then, taken over all orderings of S, the expected number of key comparisons necessary to locate q in $\mathcal{T}(S_n)$ using $\mathcal{Q}(S_n)$ is at most $5H_n$, i.e. $O(\log n)$.

Proof: Let τ_j be the trapezoid in $\mathcal{T}(S_j)$ that contains q. Assuming τ_{i-1} is known, what is E_i, the expected number of comparisons necessary to identify τ_i? In other words, assuming we know the trapezoid of $\mathcal{T}(S_{i-1})$ that contains q, what is the expected number of comparisons necessary to determine the trapezoid of $\mathcal{T}(S_i)$ that contains q?

Clearly no comparisons are necessary if τ_{i-1} and τ_i are the same. If they are different, then at least one of the sides of τ_i has to be part of the new segments s_i or part of the horizontal extension through one of the endpoints of s_i.

If the right side of τ_i is part of s_i, then exactly one comparison is necessary to identify τ_i, namely an X-comparison between q and s_i. Because of the random ordering every one of the i segments of S_i has the same chance $1/i$ of being s_i. Thus the probability that s_i happens to be the segment bounding τ_i to its right is at most $1/i$ (τ_i might be unbounded to the right). Thus the expected number of comparisons between q and the right side of τ_i is at most $1/i$. By symmetry the same holds for the left side of τ_i.

Similarly, if the upper side of τ_i is part of a horizontal extension through an endpoint of s_i, then one comparison is necessary to identify τ_i, namely a Y-comparison between q and that extension. Again, because of the random ordering this happens with probability $1/i$ at most, and thus the expected number of comparisons between q and the upper side of τ_i is at most $1/i$. The same holds for the lower side of τ_i.

Thus the expected number of comparisons between q and the sides of τ_i is at most $4/i$. (Note that here we are exploiting the fact that the expectation of a sum of random variables is simply the sum of the individual expectations, even if the random variables are dependent.) However, Mike Hohmeyer at U.C. Berkeley has pointed out that one additional kind of comparison can occur: Assume the upper side of τ_i is part of the horizontal extension of an endpoint of s_i (which happens with probability at most $1/i$), in particular assume the upper side of τ_i is the horizontal extension of the lower endpoint of s_i. In addition assume that no horizontal extension abuts upon the relative interior of s_i and that no other segment of S_i shares an endpoint with s_i. This means that the segment s_i is contained in the interior of the trapezoid τ_{i-1}. By the way we obtain the query structure $\mathcal{Q}(S_i)$ from $\mathcal{Q}(S_{i-1})$ it is clear that in order to locate q in τ_i at first also a comparison between q and the horizontal extension of the *upper* endpoint of s_i has to be made. Thus if this particular configuration occurs, one additional comparison has to be performed with probability at most $1/i$.

It follows that E_i, the expected number of comparisons necessary to ascertain that q lies in τ_i knowing that it lies in τ_{i-1} is at most $5/i$.

To prove the lemma it now suffices to observe that the expected overall query time is clearly $\sum_{1\leq i\leq n} E_i$. ∎

Lemmas 2 and 3 together with the preceding discussion immediately imply the following:

Theorem 1 *Let S be a set of n non-crossing, non-horizontal line segments in the plane. Let the trapezoidation $\mathcal{T}(S)$ and the query structure $\mathcal{Q}(S)$ be built incremen-*

tally by inserting the segments of S in random order.

1. *The expected time necessary to build $\mathcal{T}(S)$ and $\mathcal{Q}(S)$ is $O(n \log n)$.*
2. *The expected size of $\mathcal{Q}(S)$ is $O(n)$.*
3. *For any query point q the expected time for locating q in $\mathcal{T}(S)$ via $\mathcal{Q}(S)$ is $O(\log n)$.*

(All expectations are with respect to the random ordering of S, where each permutation of S is assumed to occur equally likely.)

In the following we show that if S is the set of segments of a simple polygonal chain, then $\mathcal{T}(S)$ and $\mathcal{Q}(S)$ can be built even faster.

The expensive part of inserting a segment seems to be the location queries for its endpoints. How could one do without them? If S derives from a polygonal chain C, then an obvious fix seems to be to insert the segments in order along C. This way one endpoint of s_i coincides with an endpoint of the just inserted segment s_{i-1} whose location is known. The location of the other endpoint could be determined while "threading" s_i through the current trapezoidation. Unfortunately this strategy forgoes the use of randomization, and it is not true any more that the expected cost of "threading" s_i through the current trapezoidation is constant. Indeed it is not hard to concoct examples where this "threading" cost is $O(i)$ for $n/2 \leq i \leq n$, which leads to an $O(n^2)$ algorithm.

We will pursue the following strategy: We will still insert the segments in random order. But every once in a while we will stop and locate all endpoints (which are vertices of C) in the current trapezoidation by tracing out C. To document the efficiency of this approach we will need two lemmas: one tells how much this intermediate location information helps with the searches later on; the other one tells how expensive it is to trace C through the current trapezoidation.

Lemma 4 *Let $s_1, \ldots, s_n$ be a random ordering of the segments of S, and let $S_i = \{s_1, \ldots, s_i\}$ for $0 \leq i \leq n$.*

For $1 \leq i \leq n$ let $\mathcal{T}(S_i)$ and $\mathcal{Q}(S_i)$ be the trapezoidation and query structure for S_i obtained from $\mathcal{T}(S_{i-1})$ and $\mathcal{Q}(S_{i-1})$ by inserting segment s_i.

Let $1 \leq j \leq k \leq n$. If q is a query point whose location in $\mathcal{T}(S_j)$ is known (along with the corresponding sink node of $\mathcal{Q}(S_j)$), then q can be located via $\mathcal{Q}(S_k)$ in $\mathcal{T}(S_k)$ in expected time at most $5(H_k - H_j)$, which is $O(\log(k/j))$.

(The expectation is with respect to all possible orderings of S.)

Proof: Proceed as in the proof of Lemma 3. However, the expected query time in this case is $\sum_{j<i\leq k} E_i$. ■

Lemma 5 *Let S be a set of n non-crossing, non-horizontal segments in the plane and let R be a random subset of S of size r. Let Z be the number of intersections between horizontal trapezoid sides of $\mathcal{T}(R)$ and segments in $S \setminus R$.*

The expected value of Z is at most $4(n-r)$, where the expectation is over all subsets R of S of size r.

Proof: For $T \subset S$ and $s \in T$ let $deg(s, \mathcal{T}(T))$ denote the number of horizontal extensions of endpoints of segments in T that abut upon the relative interior of s. As every horizontal extension abuts upon at most two segments we have $\sum_{s \in T} deg(s, \mathcal{T}(T)) \leq 4|T|$.

For $R \subset S$ and $s \notin R$ observe that the number of horizontal trapezoid sides of $\mathcal{T}(R)$ that are intersected by s is exactly $deg(s, \mathcal{T}(R \cup \{s\}))$. We are interested in

$$\frac{1}{\binom{n}{r}} \sum_{\substack{R \subset S \\ |R|=r}} \sum_{s \in S \setminus R} deg(s, \mathcal{T}(R \cup \{s\})).$$

Expressing this double sum slightly differently, this is the same as

$$\frac{1}{\binom{n}{r}} \sum_{\substack{R' \subset S \\ |R'|=r+1}} \sum_{s \in R'} deg(s, \mathcal{T}(R')) \leq \frac{1}{\binom{n}{r}} \sum_{\substack{R' \subset S \\ |R'|=r+1}} 4|R'| = 4(r+1)\frac{\binom{n}{r+1}}{\binom{n}{r}} = 4(n-r).$$

■

Before presenting the final algorithm and its analysis a bit of notation: Let $\log^{(i)} n$ denote the i-th iterated logarithm, i.e. $\log^{(0)} n = n$ and for $i > 0$ we have $\log^{(i)} n = \log(\log^{(i-1)} n)$. For $n > 0$ let $\log^* n$ denote the largest integer ℓ so that $\log^{(\ell)} n \geq 1$, and for $n > 0$ and $0 \leq h \leq \log^* n$ let $N(h)$ be shorthand for $\lceil n / \log^{(h)} n \rceil$.

The input to the algorithm below is a simple polygonal chain C of n segments in consecutive order along C.

1. Generate $s_1, s_2, \ldots, s_n$, a random ordering of the segments of C.
2. Generate $\mathcal{T}_1$, the trapezoidation for the set $\{s_1\}$ along with the corresponding search structure $\mathcal{Q}_1$.
3. For $h = 1$ to $\log^* n$ do

   3.1 For $N(h-1) < i \leq N(h)$ do

      3.1.1 Obtain trapezoidation $\mathcal{T}_i$ and search structure $\mathcal{Q}_i$ from $\mathcal{T}_{i-1}$ and $\mathcal{Q}_{i-1}$ by inserting segment s_i.

   3.2 Trace C through $\mathcal{T}_{N(h)}$ to determine for each endpoint of all non-inserted segments the containing trapezoid of $\mathcal{T}_{N(h)}$.

4. For $N(\log^* n) < i \leq n$ do

   4.1 Obtain trapezoidation $\mathcal{T}_i$ and search structure $\mathcal{Q}_i$ from $\mathcal{T}_{i-1}$ and $\mathcal{Q}_{i-1}$ by inserting segment s_i.

What is the expected running time of this algorithm? We assume that step 1 can be executed in linear time (see Section 4 for a discussion of this assumption). Step 2 takes constant time.

For step 3 let us consider some h, with $1 \leq h \leq \log^* n$. By Lemma 5 Step 3.2 can be performed in $O(n)$ expected time. What about step 3.1? The expected cost of step 3.1.1, inserting segment s_i, is the sum of the expected time necessary to locate s_i's endpoints in $\mathcal{T}_{i-1}$ and the expected time for "threading" s_i through $\mathcal{T}_{i-1}$. By Lemma 2 the latter is constant. Since the location of s_i's endpoints in $\mathcal{T}_{N(h-1)}$ is known already, by Lemma 4 the expected location time is only $O(\log(i/N(h-1)))$. Since $i \leq n$ and $N(h-1) = \lceil n/\log^{(h-1)} n \rceil$ this is $O(\log^{(h)} n)$. For fixed h step 3.1.1 is executed at most $N(h) = \lceil n/\log^{(h)} n \rceil$ times. Thus for fixed h the total cost of step 3 is $O(n)$, and hence the entire expected cost of step 3 over all h is $O(n \log^* n)$.

The analysis for step 4 is similar to the one of step 3.1. Note that $N(\log^* n) \geq n/e$, and therefore the expected point location cost is constant. It follows that step 4 takes $O(n)$ expected time. Thus we have proved the following:

Theorem 2 *Let S be a set of n non-crossing segments in the plane that are presented as a simple polygonal chain C.*

The above algorithm computes the trapezoidation $\mathcal{T}(S)$ along with a point location structure $\mathcal{Q}(S)$ for this trapezoidation in expected time $O(n \log^* n)$.

The point location structure uses expected $O(n)$ space and admits an expected $O(\log n)$ query time.

This theorem admits easy generalizations. Note that the algorithm exploits only one property of the chain C, namely its connectedness. If S was presented as a connected plane graph G with the order of the incident edges around each vertex given explicitly, then the result would continue to hold. Only the tracing of step 3.2 of the algorithm would have to be performed in order of some graph traversal of G. Allowing the graph G to have several connected components leads to a unification of Theorems 1 and 2.

Theorem 3 *Let S be a set of n non-crossing segments in the plane that are presented as a plane graph with k connected components. Let $\mathcal{T}(S)$ be the trapezoidation of S.*

> *Trapezoidation $\mathcal{T}(S)$ along with a query structure $\mathcal{Q}(S)$ can be built in time $O(n \log^* n + k \log n)$.*
>
> *The expected size of $\mathcal{Q}(S)$ is $O(n)$.*
>
> *For any query point q the expected time for locating q in $\mathcal{T}(S)$ via $\mathcal{Q}(S)$ is $O(\log n)$.*

Proof: Apply the algorithm outlined above, but modify step 3.2 so as to trace the segments of each connected component of the graph through the current trapezoidation in the order of some graph traversal, say, depth-first-search. In order to start the tracing of one connected component, one of its vertices has to be located in the current trapezoidation. Because of Lemma 3 the cumulative expected cost for this over the entire algorithm is $O(\log n)$ per component. ∎

4 Remarks

An algorithm somewhat similar to the one described in this paper has also been discovered by Clarkson, Cole, and Tarjan [CCT]. However, their approach is based on divide-and-conquer and the main thrust of their approach is towards a fast parallel trapezoidation algorithm.

Our algorithm can be viewed as a holistic version of the divide-and-conquer algorithm in [CTT] or its predecessor in [CTV]. The $\log^* n$ stops in our segment insertion routine at which the curve is traced through the current trapezoidation correspond to the $\log^* n$ levels of recursion in the divide-and-conquer algorithms. However, by always dealing with the global trapezoidation our algorithm avoids the myopia brought about by the very high divisiveness of the divide-and-conquer algorithms.

The algorithm presented in this paper seems to require that every one of the $n!$ permutations of n segments can be generated with equal likelihood. This might seem unsatisfactory from the theoretical point of view, since this requires $\Theta(n \log n)$

random bits. As a matter of fact this makes the main result of this paper questionable at best, if a model of computation is used that requires unit cost for single random bits. For our result to be interesting we need to be able to obtain random integers distributed over a polynomially sized range at *unit cost*. However, such a model of computation is not at all unreasonable: From the practical point of view today's pseudo-random number generators do provide such "random" integers at unit cost. From the theoretical point of view it seems unfair to work with a uniform cost model for the arithmetic but with a bit model for the randomness. Note that if a bit model is assumed throughout, then the number of random bits required is proportional to the input size (measured in bits, of course).

The question of estimating the probability that the running time of our algorithm significantly exceeds its expectation is currently being investigated.

5 Acknowledgements

I would like to thank Mike Hohmeyer for pointing out an error in the original version of the proof of Lemma 3. I would also like to thank Emo Welzl for inviting me to the Freie Universität Berlin. I doubt this research would have happened without the stimulating visit there.

6 References

[C82] Chazelle, B. *A theorem on polygon cutting with applications*, Proc. 23rd Annual IEEE Symp. on Found. of Comput. Sci. (1982), 339–349.

[C90] Chazelle, B. *Triangulating a simple polygon in linear time*, Princeton Univ. Computer Science Tech. Rep. CS-TR-264-90; to appear in Proc. 31st Annual IEEE Symp. on Found. of Comput. Sci. (1990).

[CI] Chazelle, B., Incerpi, J. *Triangulation and shape-complexity*, ACM Trans. on Graphics 3 (1984), 135–152.

[CCT] Clarkson, K., Cole, R., and Tarjan R.E. *Randomized Parallel Algorithms for Trapezoidal Diagrams*, to appear in Proc. 7th ACM Symposium on Computational Geometry (1991).

[CTV] Clarkson, K., Tarjan, R.E., Van Wyk, C.J. *A fast Las Vegas algorithm for triangulating a simple polygon*, Disc. and Comput. Geom. 4(5) (1989), 432–432.

[FM] Fournier, A., Montuno, D.Y. *Triangulating simple polygons and equivalent problems*, ACM Trans. on Graphics 3. (1984), 153–174.

[GJPT] Garey, M.R., Johnson, D.S., Preparata, F.P., Tarjan, R.E. *Triangulating a simple polygon*, Inform. Process. Lett. 7(1978), 175–180.

[HM] Hertel, S., Mehlhorn, K. *Fast triangulation of a simple polygon*, Proc. Conf. Found. Comput. Theory, New York, Lecture Notes on Computer Science 158 (1983), 207–218.

[HMRT] Hoffman, K., Mehlhorn, K., Rosenstiehl, P., Tarjan, R.E. *Sorting Jordan sequences in linear time using level-linked search trees*, Inform. and Control 68 (1986), 170–184.

[KKT] Kirkpatrick, D.G., Klawe, M.M., Tarjan, R.E. $O(n \log\log n)$ *polygon triangulation with simple data structures*, Proc. 6th Annual ACM Symp. Comput. Geom. (1990), 34–43.

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

[TA82] Toussaint, G. and Avis, D. *On a convex hull algorithm for polygons and its applications to triangulation problems*, Pattern Recognition 15,1 (1982), 23–29.

[T88] Toussaint, G. *An output-complexity-sensitive polygon triangulation algorithm*, Report SICS-86.3, McGill University, Montreal, 1988.