1 Intersecting Line Segments

Note: These notes cover an algorithm that is slightly different from the one given in class, which is actually the one referred to as possible exercise in the remarks on the top of page 5.

Let S be a set of n straight line segments in the plane. We are interested in finding all intersecting pairs of segments in S.

The first non-trivial algorithm for solving this problem was given by Bentley and Ottmann in 1979 [1]. It was based on the sweep paradigm and achieved a worst case running time of $O((K+n)\log n)$, where K is the output size, namely the number of intersecting pairs of segments in S. Since it is possible that all segments of S intersect each other, this algorithm can have an $O(n^2 \log n)$ running time, which is inferior to the $O(n^2)$ time of the trivial method of checking every pair in S. Thus the question arose, whether a bound of the form $O(K+n\log n)$ was possible. Since it is easy to show by reduction from element uniqueness that $\Omega(n \log n)$ is a lower bound to the segment intersection problem, and since $\Omega(K)$ is also a lower bound as at least this much time has to be spent on output, one could not hope for anything better than $O(K+n\log n)$.

In 1983 Chazelle [2] came close to this goal with a rather complicated algorithm whose worst case running time was $O(K+n\log^2 n/\log\log n)$. Finally, five years later he and Edelsbrunner [3] designed an even more complicated deterministic algorithm that did achieve the $O(K+n\log n)$ worst case running time. Around the same time independently Mulmuley [8] as well as Clarkson and Shor [5] developed rather simple randomized algorithms with $O(K+n\log n)$ expected running time. Clarkson and Shor based the analysis of the running time of their algorithm on the general theory of random sampling. They even managed to come up with a version of the algorithm that required only $O(n)$ space. Mulmuley analyzed the performance of his algorithm via probabilistic games that he developed for this purpose. His analysis is reasonably complex, however it yields rather tight constants.

In this section we present Mulmuley's algorithm and give a very simple analysis of its expected performance that is based on our backwards view.

For the sake of ease of presentation let us assume we are dealing with a set S of n segments that is non-degenerate in the sense that no two segments of S have the same endpoint, no three segments intersect in a common point, no two segment endpoints have the same x-coordinate, and that no segment endpoint lies in the relative interior of some other segment. As usual such non-degeneracy could be simulated by standard perturbation methods [6, pp. 185], or also the algorithm could easily be modified so that none of these assumptions are necessary.

Mulmuley's algorithm does more than just determine which pairs of segments in S intersect. It constructs what we call the *trapezoidal decomposition* induced by S. This decomposition $\mathcal{T}(S)$ can be intuitively defined as follows: First draw a sufficiently large axis-parallel rectangle frame F that contains in its interior all segments of S. Next draw all segments of S in the rectangle F. Finally, from each intersection point and from each segment endpoint draw its *vertical extensions*, i.e. start drawing two vertical rays, one going up, the other going down, that extend until they hit a segment of S or the boundary of F (see Figure 2). Thus F is decomposed into trapezoids that each have two vertical sides (one of which can have length 0). Using a sweep argument it is not hard to prove that $\mathcal{T}(S)$ contains exactly $3(n+K)+1$ trapezoids; thus, when viewed as a planar graph, $\mathcal{T}(S)$ has $O(K+n)$ faces, edges, and vertices.

Let us define for any subset $R \subset S$ the trapezoidal decomposition $\mathcal{T}_S(R)$ in a similar way as follows: Draw all segments of R in the rectangle frame F, and for each intersection point of segments in R as well as for each endpoint of a segment in S (note: *in* S) draw its vertical extensions. But now the rays that form the vertical extensions extend only until they hit a segment of R or the boundary of the rectangle F (see Figure 3). The decomposition $\mathcal{T}_S(R)$ partitions F into $2n + r + 3K_R + 1$ trapezoids, where $r = |R|$ and K_R is the number of pairs of intersecting segments in R, and thus $\mathcal{T}_S(R)$ as a planar graph has $O(n + K_R)$ faces, edges, and vertices.

Mulmuley's algorithm for computing $\mathcal{T}(S) = \mathcal{T}_S(S)$ is very simple: First compute $\mathcal{T}_S(\emptyset)$ and then insert the segments of S into the trapezoidation in random order to compute $\mathcal{T}_S(R)$ for an every increasing $R \subset S$.

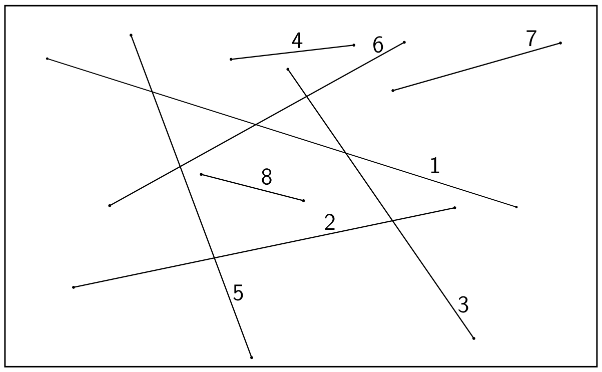

Figure 2a: Set S of 8 segments in a frame

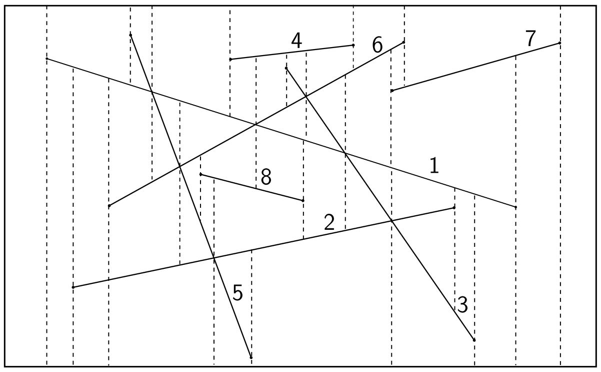

Figure 2b: Trapezoidation $\mathcal{T}(S)$

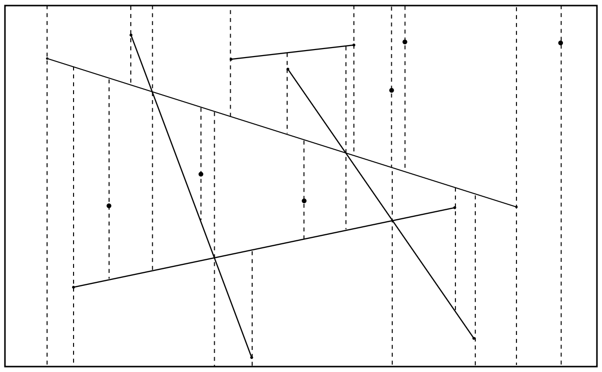
Figure 3: $\mathcal{T}_S(R)$ for $R = \{1, 2, 3, 4, 5\}$

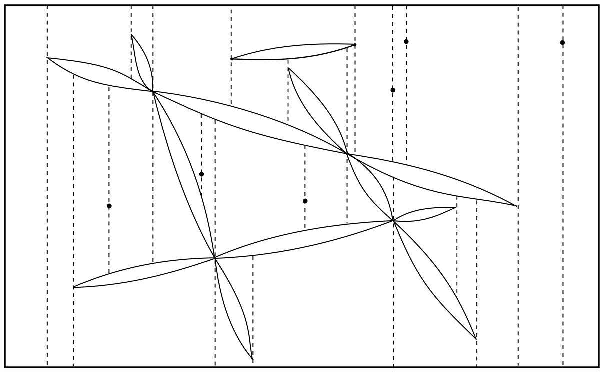
Figure 4: Introduction of zero-width faces for pieces yields $G_S(R)$

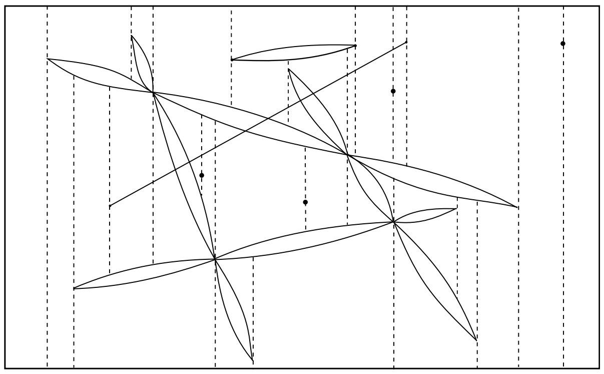
Figure 5a: Inserting segment 6

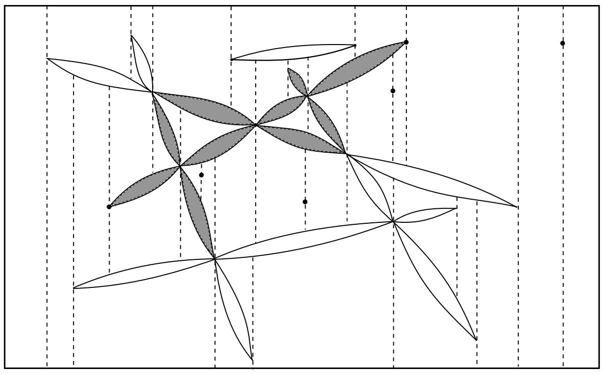
Figure 5b: $\mathcal{P}_R(6)$ shaded

We will express his algorithm for computing $\mathcal{T}_S(R)$ recursively:

> If $R = \emptyset$ then compute $\mathcal{T}_S(R)$ directly by sorting the endpoints of S by their x-coordinates.
>
> Otherwise randomly pick a segment $s \in R$, recursively compute $\mathcal{T}_S(R \setminus \{s\})$, and introduce s into this trapezoidation to obtain $\mathcal{T}_S(R)$.

We still have to specify how a segment s is introduced into the current trapezoidation. In order to do this we need to give more detail about the representation of trapezoidations. Mulmuley chooses a somewhat idiosyncratic representation that essentially works as follows: Consider $\mathcal{T}_S(R)$, and consider some segment $s \in R$. Assume s is intersected by i other segments in R. Thus s is partitioned into $i+1$ *pieces*. Conceptually Mulmuley makes each piece of each segment in R into a narrow zero-width face, and obtains this way from $\mathcal{T}_S(R)$ a plane graph that we denote by $G_S(R)$. Figure 4 should make the idea clear. Note that $G_S(R)$ has the same number of vertices as $\mathcal{T}_S(R)$ and thus it also has complexity $O(n + K_R)$. Also note that in the graph $G_S(R)$ faces that correspond to trapezoids of $\mathcal{T}_S(R)$ have at most six edges around them, only the zero-width faces can have more than a constant number of edges around them.

How does one now introduce segment s into $G_S(R')$ to obtain $G_S(R)$, where $R' = R \setminus \{s\}$? It works in two phases (that of course could be combined into one). In the first phase one determines which faces and edges of $G_S(R')$ are intersected by s. In the second phase the new faces of $G_S(R)$ are created. This involves splitting the faces of $G_S(R')$ that are intersected by s, creating the zero-width faces for the pieces of s, introducing the vertical extensions from the intersection points of s with the other segments of R, and merging faces that are separated by vertical edges that are not part of vertical extensions any more because of the introduction of s. We leave the details of the second phase to the reader.

The first phase can simply be done as follows: Since the two endpoints of s are vertices of $G_S(R')$ already, we can determine in constant time, say, the leftmost face of $G_S(R')$ that is intersected by s. Now we "thread" s through $G_S(R')$ in the usual way, similar to the incremental line arrangement construction algorithm [7, 4]: We walk along the segment s;

assume we just entered a face f through some edge e; we determine through which edge the segment s leaves f and which new face the segment enters by simply testing all edges of f (say, in clockwise order starting after e). Now we have reached a new face, and we repeat. Obviously, this procedure can also tell when we have reached the right endpoint of s.

What is the time necessary for introducing segment s? It is not hard to see that the cost of phase one dominates the cost of phase two. Thus it suffices to consider just the cost of phase one. This cost is clearly the sum of the degrees of all the faces of $G_S(R')$ that are intersected by s (here the degree of a face f of $G_S(R')$, for short $deg(f, G_S(R'))$, is the number of edges of $G_S(R')$ incident to f). However, what does this evaluate to for a random segment $s \in R$?

Now let us apply backwards analysis. The first important step is to express the cost of phase one of introducing a segment s into $G_S(R')$ in terms of the result graph $G_S(R)$ and not in terms of $G_S(R')$. It is not hard to see that this cost is given by $\sum_{f \in \mathcal{P}_R(s)} deg(f, G_S(R))$, where $\mathcal{P}_R(s)$ is the set of all zero-width faces of $G_S(R)$ that either derive from pieces of s in $G_S(R)$ or from those pieces of other segments in R that are incident to intersection points with s (see Figure 5).

It follows that if s is randomly chosen from the r segments in R, then the expected cost of adding s to $G_S(R \setminus \{s\})$ is proportional to

$$\frac{1}{r} \sum_{s \in R} \sum_{f \in \mathcal{P}_R(s)} deg(f, G_S(R)) \ .$$

But in this double sum every piece of a segment in R contributes at most three times. Thus if $\mathcal{P}(R)$ denotes the set of all faces that derive from pieces of segments in R, then this double sum is at most

$$\frac{3}{r} \sum_{f \in \mathcal{P}(R)} deg(f, G_S(R)) \ .$$

Since $G_S(R)$ is a planar graph, this sum is clearly proportional to the complexity of the graph, and thus it is $O(n + K_R)$. It follows that the expected cost of introducing the last segment of R is $O(\frac{n}{r} + \frac{K_R}{r})$.

But what is the expected value of K_R? By the way the algorithm proceeds it is clear that R is a random subset of S of size r. Now if $\{s, t\}$ is one of the K pairs of intersecting segments in S, what is the probability that they are both in R? Clearly $\frac{r(r-1)}{n(n-1)}$. It follows that the expected number of pairs of intersecting segments in R, i.e. the expectation K_r of K_R is $\frac{r(r-1)}{n(n-1)}K$.

Thus the expected cost of introducing the last segment into $G_S(R)$ is $O(\frac{n}{r} + \frac{r-1}{n(n-1)})$. To obtain the expected cost for all recursive calls of the entire algorithm one clearly only needs to sum this expression for $1 \leq r \leq n$, which yields

$$O(nH_n + K),$$

where $H_n = 1 + 1/2 + \ldots + 1/n \approx \log n$. Since computing $G_S(\emptyset)$ just amounts to sorting $2n$ numbers it follows that the expected running time of the entire algorithm is $O(K + n \log n)$.

Remarks: As an exercise the reader may want to try this type of analysis on a version of this algorithm that does not use zero-width faces and uses instead of $G_S(R)$ simply a planar graph representation of $\mathcal{T}_S(R)$. Thus the trapezoids can have arbitrarily many edges around them.

In the presentation of Mulmuley's algorithm we assumed non-degeneracy. This is not too much of an issue, except in cases where many segments intersect in one point. In such a situation it would be desirable to obtain an expected running time of $O(I + n \log n)$, where is I is the number of intersection points between segments. (Note that if all segments intersect in one point, then $I = 1$ but $K = \binom{n}{2}$.) By fairly obvious modifications of the algorithm outlined above it is possible to achieve this $O(I + n \log n)$ bound. The only complications arise in the analysis, since I_r, the analogue of K_r, seems to be difficult to express in a nice closed form. However, the following can be shown and saves the analysis: If X is the set of intersection points between segments of S and if for $p \in X$ the number of segments of S that intersect in p is denoted by $d(p)$, then[1]

$$\sum_{1 \le r \le n} \frac{I_r}{r} = \sum_{p \in X} (H_{d(p)} - 1) \ .$$

But since $\sum_{p \in X} d(p) = O(I + n)$, clearly $\sum_{p \in X}(H_{d(p)} - 1)$ and therefore also $\sum_{1 \le r \le n} I_r/r$ is $O(I + n)$.

Finally we should point out that the algorithm described here does not exploit at all the straightness of the segments in S or the fact that any two segments intersect at most once. With very straightforward modifications the algorithm can be adapted to construct the trapezoidal decomposition induced by a set S of "segments," where each member $s \in S$ is a bounded x-monotone curve (i.e. every vertical line intersects s at most once), where every pair $s, s' \in S$ intersect in a finite number of points, and where for any vertical line ℓ one can determine in $O(1)$ time the "first" intersection point between s and s' to the right of ℓ. No changes in the analysis are necessary at all. It is still $O(I + n \log n)$, where I now stands for the number of intersection points and can be arbitrarily large.

[1]The derivation of this formula — at least as done by the author — is not completely straightforward and requires some massaging of sums involving quotients of binomial coefficients. In particular one needs to show that

$$\sum_{1 \le r \le n} \frac{1}{r} \left[1 - \frac{\binom{n-d}{r}}{\binom{n}{r}} - d \frac{\binom{n-d}{r-1}}{\binom{n}{r}} \right] = H_d - 1 \, ,$$

where the quantity in the square brackets is the probability that an intersection point among d segments of S exists in a random sample of r segments.

References

[1] J.L. Bentley and T.A. Ottmann. ‘‘*Algorithms for Reporting and Counting Geometric Intersections.”* IEEE Transactions on Computers 28 (1979) pp 643–647.

[2] B. Chazelle. *“Reporting and Counting Segment Intersections.”* J. Computer System Science 32 (1986) pp 156–182.

[3] B. Chazelle and H. Edelsbrunner. *“An Optimal Algorithm for Intersecting Line Segments in the Plane.”* Proc. 29th IEEE Symp. on Foundations of Computer Science (1988) pp 590–600.

[4] B. Chazelle, L.J. Guibas, and D.T. Lee. *“The Power of Geometric Duality.”* BIT 25 (1985) pp 76–90.

[5] K.L. Clarkson and P.W. Shor. *Applications of Random Sampling in Computational Geometry, II.”* Discrete & Computational Geometry 4 (1989), pp 387–421.

[6] H. Edelsbrunner. **Algorithms in Combinatorial Geometry.** Springer Verlag (1987).

[7] H. Edelsbrunner, J. O’Rourke, and R. Seidel. *“Constructing Arrangements of Hyperplanes and Applications.”* SIAM J. on Computing 15 (1986), pp 341–363.

[8] K. Mulmuley. *“A Fast Planar Partition Algorithm: Part I.”* Proc. 29th IEEE Symp. on Foundations of Computer Science (1988), pp 580–589.