A Variant of the Line Sweep Algorithm with User-defined Uncertainty Quantification

Zhixuan Li

October 21, 2021

1 Introduction

Computing the intersections of a set of line segments is one fundamental routine in computational geometry. The line sweep algorithm [**?**] is an efficient scheme for solving this problem. In this article, we introduce a variant of the line sweep algorithm. While preserving the efficient nature, our algorithm is amenable to float-point errors, and is adaptive to problems of all scales by using a user-defined uncertainty quantification. In addition, it is capable of computing the incident segments for each intersection, assuming no precondition on the input.

2 Algorithm

Our algorithm accepts a multiset of line segments as input. The plain version of the line sweep algorithm fails to deal with cases such as improper intersections and overlapping segments. In order to deal with overlapping segments, we introduce the concept of a *bundle*. A bundle is an unordered set of segments that overlaps at the level of the sweep line. In our algorithm, the status structure $\mathcal{T}$ is designed to be an ordered set whose elements are bundles. Accordingly, adding segments into $\mathcal{T}$ now adds to the corresponding bundle, and intersecting adjacent segments now becomes intersecting adjacent bundles, etc.

3 Analysis

3.1 Correctness

For any input S, we can choose the slope of the sweep line to be some tiny positive δ so that no segment in S is parallel to the sweep line. Algorithmically, this is equivalent to : (i) using a horizontal sweep line; (ii) sorting the event points using their x coordinates as minor keyword (y coordinates as major keyword). Hence it is safe to assume the input does not contain any horizontal segment.

Before moving on to prove the corrrectness of Algorithm 1, we would like to make some comments on the algorithm. In fact, finding the intersections is done in Line 2, Algorithm 2. The call to Algorithm 2 happens in three places : Line 10-11 in Algorithm 1; Line 17-24 in Algorithm 1; and Line 3 in Algorithm 3. Intuitively speaking,

- Line 17-24 tests for intersection among each pair of bundles that become newly adjacent.

Algorithm 1 findIntersections(S)

Input : $S = \{s_1, \cdots, s_n\}$ is a multiset of line segments that may contain overlapping segments.
Precondition : None
Output : Ω is the point set of the intersections. inc(p) is the segments incident to p, $\forall p \in \Omega$.
Postcondition : Each intersection point with its incident segments is reported.
For overlapping segments, endpoints of the overlie are reported.

1: Empty the priority queue $\mathcal{Q}$. Let all the endpoints of the segments in S enqueue.
2: For each endpoint in $\mathcal{Q}$, initialize $L(p), U(p)$ accordingly, where $L(p)$ is the set of the segments whose upper endpoint is p, and $U(p)$ likewise. Initialize $C(p)$ to empty.
3: Empty the status structure $\mathcal{T}$.
4:
5: **while** $\mathcal{Q}$ is not empty **do**
6: Pop the top event in $\mathcal{Q}$, denoted as p
7: **if** $L(p) \cup C(p) = \emptyset$ **then**
8: Pick an arbitrary segment from $U(p)$, say s'.
9: $B' =$ **addToStatus**$(\mathcal{Q}, \mathcal{T}, s')$.
10: **findNewEvent**$(\mathcal{Q}, \{s'\}$,pred(B'),$p)$.
11: **findNewEvent**$(\mathcal{Q}, \{s'\}$,succ(B'),$p)$.
12: /* pred(B'), succ(B') stand for the predecessor and the succesor of bundle B' in $\mathcal{T}$. */
13: Remove s' from $\mathcal{T}$.
14: **end if**
15: Remove $L(p) \cup C(p)$ from $\mathcal{T}$.
16: For each segment $s \in C(p) \cup U(p)$ do **addToStatus**$(\mathcal{Q}, \mathcal{T}, s)$.
17: **if** $C(p) \cup U(p) = \emptyset$ **then**
18: Let B' be the successor of the rightmost bundle among $L(p)$ before deletetion.
19: **findNewEvent**$(\mathcal{Q}, B'$,pred(B'),$p)$.
20: **else**
21: Let B_l be the leftmost bundle among $U(p) \cup C(p)$, and B_r be the rightmost one.
22: **findNewEvent**$(\mathcal{Q}, B_l$,pred(B_l),$p)$.
23: **findNewEvent**$(\mathcal{Q}, B_r$,succ(B_r),$p)$.
24: **end if**
25:
26: **if** $|L(p) \cup C(p) \cup U(p)| >= 2$ **then**
27: $\Omega \leftarrow \Omega \cup \{p\}$.
28: inc$(p) \leftarrow L(p) \cup C(p) \cup U(p)$.
29: **end if**
30: **end while**

Algorithm 2 findNewEvent($\mathcal{Q}, B_l$,B_r,p)

Side effect : $\mathcal{Q}$ is modified.

1: **for** each pair of segments s_1, s_2 from B_l, B_r respectively **do**
2: **for** each intersection q of s_1 and s_2 **do**
3: **if** p equals q or p is prior to q **then**
4: Let q enqueue.
5: Add s_1 to $C(q)$ if $s_1 \notin L(q) \cup U(q)$ and do the same for s_2.
6: **end if**
7: **end for**
8: **end for**

Algorithm 3 addToStatus($\mathcal{Q}, \mathcal{T}, s$)

Side effect : $\mathcal{Q}, \mathcal{T}$ is modified.

1: **if** s belongs to some existing bundle B in $\mathcal{T}$ **then**
2: $B = B \cup \{s\}$.
3: **findNewEvent**($\mathcal{Q}, \{s\}, B \setminus \{s\}, p$).
4: **else**
5: Create a new bundle $B' = \{s\}$. Let $\mathcal{T} = \mathcal{T} \cup \{B'\}$.
6: **end if**

- Line 3 in Algorithm 3 calculates the endpoints of the overlie (which is implied by the condition in Line 1 being satisfied).
- Line 7-13 in Algorithm 1 deals with a special case which will be explained later.

In the plain version of the line sweeping algorithm, only the first measure is taken. The second and the third measure is needed here because we need to deal with degenerate cases. In the proof of the following Proposition, we argue that these three measures combined are all we need in order to correctly find out the segments incident to an event point.

Proposition 1. *Upon the finish of Line 14 in Algorithm 1, all segments incident to the current event point p are found.*

Proof. Segments that has p as endpoint can certainly be found due to the work of Line 3. So we focus on showing $C(p)$ contains all the segments that have p in their interior.

1. The conclusion natually holds under the trivial case of $C(p) = \emptyset$.
2. Since the sweeping line is moving up-down, at the moment right before the sweep line reaches the current event point, the sweep line intersects with all the bundles in $L(p) \cup C(p)$. If $L(p) \cup C(p)$ contributes to two or more bundles, then each pair of neighbouring bundles has already been tested in the previous events due to the work of Line 17-24. In this case, $C(p)$ can be correctly found.
3. Now suppose the contrary, i.e., $L(p) \cup C(p)$ consists of only one bundle. Recall that Algorithm 3 guarantees that inside this bundle, each pair of segments is tested against when they are inserted into $\mathcal{T}$. So if $L(p)$ is non-empty, say there is some $s' \in L(p)$, all segments in $C(p)$ has been tested against s' in the previous events (either in the upper event of s' or in the upper events of the segments in $C(p)$).
4. The last possibility is that $L(p) \cup C(p)$ consists of only one bundle and $L(p) = \emptyset$. However this implies $U(p) \neq \emptyset$ or otherwise, p cannot be an event point. Note that the segments in $C(p)$ never have the chance of being tested against the segments in $U(p)$, so upon the finish of Line 6 $C(p)$ is still empty. Line 7 serves as the entering condition for this case, and Line 8-13 recovers the true $C(p)$.

□

Proposition 2. *All the intersections are reported by Algorithm 1.*

Proof. By Proposition 1 and Line 26-29, it suffices to show any intersection has ever been presented in $\mathcal{Q}$.

If p is a proper intersection, then slightly above p there exists two (or more) adjacent bundles emitted from p. Line 17-24 guarantees that any pair of bundles that become newly adjacent is tested, thus p is put enqueued at some moment. If p is an improper intersection, p must be an endpoint, and it is in $\mathcal{Q}$ from the beginning.

□

3.2 Complexity

Table 1 is a list of notations we shall use in the complexity analysis.

S	The multiset of segments.
Ω	The set of intersections.
n	$\lvert S\rvert$, the number of segments.
I	$\lvert\Omega\rvert$, the number of intersections.
e	The total number of event points.
$m(p)$	$\lvert L(p) \cup C(p) \cup U(p)\rvert$.
m	$\sum_p m(p)$ as p ranges over all event points.
k	$\sum_{p\in\Omega} \lvert\mathrm{inc}(p)\rvert$, the number of output.

Table 1: Notations.

Definition 1. *The* bundle size $\mu(S)$ *of a multiset of segments S is the maximum number of overlapping segments. In symbol,*

$$\mu(S) = \max\{|U| : U \subset S, \textit{the intersection of members of } U \textit{ is a segment}\}. \tag{1}$$

Proposition 3. *If $\mu(S) = \mathcal{O}(1)$, Algorithm 1 takes time $\mathcal{O}(m \log n)$.*

Proof. The initialization in Line 1-2 involves $2n$ times of maintenance of balanced-tree (MOB), thus it takes $\mathcal{O}(n \log n)$. Line 3 takes $\mathcal{O}(1)$. In each loop, Line 6-14, 17-24 involes constant many times of MOB and intersection findings because $\mu(S) = \mathcal{O}(1)$. Hence these lines of code cost $\mathcal{O}(e \log n)$ throughout the algorithm. In Line 26-29, output to Ω involves at most $2e$ insertions to a linear container since event points are sorted, while output to $\mathrm{inc}(p)$ takes $\mathcal{O}(k)$. Again from $\mu(S) = \mathcal{O}(1)$, Line 15-16 involes $\mathcal{O}(m)$ times of MOB, so it takes $\mathcal{O}(m \log n)$.

Up to now we have the estimation of $\mathcal{O}((n+e+m)\log n)$, eliminating the insignificant terms. But it's easy to see that $n \le m, e \le m$ and the conclusion follows.

□

Proposition 4. *If $\mu(S) = \mathcal{O}(1)$, Algorithm 1 takes time $\mathcal{O}((n + I)\log n)$.*

Proof. We follow the approach of Lemma 2.3 in [**?**] to show $m = \mathcal{O}(n + I)$.

From now on we interpret S as a planar graph embedded in the plane. Let n_e denote the number of edges, and n_v the number of vertices. Since $m(p)$ is bounded by $\mu(S)$ times the degree of p, it follows that m is bounded by $\mu(S)$ times the sum of degrees. But every edge adds 2 to the sum of degrees, hence m is bounded by $2\mu(S)n_e$. By *Euler's formula*, $n_e = \mathcal{O}(n_v)$. By the definition of I, we have $n_v \le 2n + I$. Combined with Proposition 3, the time complexity is indeed $\mathcal{O}((n + I)\log n)$. □

Proposition 5. *Algorithm 1 consumes $\mathcal{O}(n + I)$ space.*

Proof. The size of the status structure $\mathcal{T}$ never exceeds n. By definition the event queue $\mathcal{Q}$ along with $L(p), C(p), U(p)$ takes $\mathcal{O}(m)$ space. From the proof of Proposition 3 and 4 we know $\mathcal{O}(n + m) = \mathcal{O}(n + I)$. □

4 Test