# CSCE-629 Analysis of Algorithms

**Fall 2019**

**Instructor:** Dr. Jianer Chen
**Office:** HRBB 338C
**Phone:** 845-4259
**Email:** chen@cse.tamu.edu
**Office Hours:** T,Th 10:50 am–12:30 pm

**Teaching Assistant:** Qin Huang
**Office:** HRBB 309D
**Phone:** (979) 402-6216
**Email:** huangqin@email.tamu.edu
**Office Hours:** MWF 3:30 pm–4:30 pm

## Course Notes 3. Segment Intersection

In this notes, we illustrate a technique called *geometric sweeping* and use it to solve the segment intersection problem. Geometric sweeping is a generalization of a technique called *plane sweeping*, that is primarily used for 2-dimensional problems. In most cases, we will illustrate the technique for 2-dimensional cases. The generalization to higher dimensions is straightforward. This technique is also known as the *scan-line method* in computer graphics, and is used for a variety of applications, such as shading, polygon filling, among others.

The technique is intuitively simple. Suppose that we have a line in the plane. To collect the geometric information we are interested in, we slide the line in some way so that the whole plane will be "scanned" by the line. While the line is sweeping the plane, we stop at some points and update our recording. We continue this process until all interesting objects are collected.

There are two basic structures associated with this technique. One is for the *sweeping line status*, which is an appropriate description of the relevant information of the geometric objects at the sweeping line, and the other is for the *event points*, which are the places we should stop and update our recording. Note that the structures may be implemented in different data structures under various situations. In general, the data structures should support efficient operations that are necessary for updating the structures while the line is sweeping the plane.

We explain the idea by a more concrete example, called Segment Intersection problem:

Segment Intersection. Given $n$ line segments in the plane, find all intersections.

Suppose that we have a vertical line $L$. We sweep the plane from left to right. At every moment, the *sweeping line status* contains all segments intersecting the line $L$, sorted by the $y$-coordinates of their intersecting points with $L$. The sweeping line status is modified whenever one of the following three cases occurs:

1. The line $L$ hits the left-end of a segment $S$. In this case, the segment $S$ was not seen before and it may have intersections with other segments on the right side of the line $L$, so the segment $S$ should be added to the sweeping line status;

2. The line $L$ hits the right-end of a segment $S$. In this case, the segment $S$ cannot have any intersections with other segments on the right side of the line $L$, so the segment $S$ can be deleted from the sweeping line status;

3. The line $L$ hits an intersection of two segments $S_1$ and $S_2$. In this case, the relative positions of the segments $S_1$ and $S_2$ in the sweeping line status should be swapped, since

the segments in the sweeping line status are sorted by the $y$-coordinates of their intersection points with the line $L$.

It is easy to see that the sweeping line status of the line $L$ will not be changed when it moves from left to right unless it hits either an endpoint of a segment or an intersection of two segments. Therefore, the set of *event points* consists of the endpoints of the given segments and the intersection points of the segments. We sort the event points by their $x$-coordinates.

We use two data structures *EVENT* and *STATUS* to store the event points and the sweeping line status, respectively, such that the set operations MINIMUM, INSERT, and DELETE can be performed efficiently (for example, they can be 2-3 trees). At very beginning, we suppose that the line $L$ is far enough to the left so that no segments intersect $L$. At this moment, the sweeping line status is an empty set. We sort all endpoints of the segments by their $x$-coordinates and store them in *EVENT*. These are the event points at which the line $L$ should stop and update the sweeping line status. However, the list is not complete since an intersection point of two segments should also be an event point. Unfortunately, these points are unknown to we at beginning. For this, we update the structure *EVENT* in the following way. Whenever we find an intersection point of two segments while the line $L$ is sweeping the plane, we add the intersection point to *EVENT*. But how do we find these intersection points? Note that if the next event point to be hit by the sweeping line $L$ is an intersection point of two segments $S_1$ and $S_2$, then the segments $S_1$ and $S_2$ should be adjacent in the sweeping line status. Therefore, whenever we change the adjacency relation in *STATUS*, we check for intersection points for new adjacent segments. When the line $L$ reaches the left-most endpoint of the segments, all possible intersection points are collected.

These ideas are summarized in the algorithm given in Figure 1.

Let us analyze the complexity of the algorithm. Step 1, sorting the $2n$ endpoints of the segments, can be done in time $O(n \log n)$, if we employ an efficient sorting algorithm, for example, MergeSort. Step 2 takes constant time $O(1)$. To count the time spent by the **while** loop of step 3, suppose there are $m$ intersection points for these $n$ segments. In the **while** loop, each segment is inserted then deleted from the structure ST exactly once, and each event point is inserted then deleted from the structure EV exactly once. There are $2n + m$ event points. If we suppose that the operations MINIMUM, INSERT, and DELETE can all be done in time $O(\log N)$ on a set of $N$ elements, then processing each segment takes at most $O(\log n)$ time, and processing each event point takes at most $O(\log(n + m))$ time. Therefore, the algorithm runs in time

$$\begin{aligned} & O(n \log n) + O(1) + n \times O(\log n) + (n + m) \times O(\log(n + m)) \\ = \; & O((n + m) \log(n + m)) \end{aligned}$$

Observe that $m$ is at most $n^2$, so $\log(n + m) = O(\log n)$. Thus we conclude that the algorithm SEGMENT-INTERSECTION runs in time $O((n + m) \log n)$.

We remark that the time complexity of the above algorithm depends on the number $m$ of intersection points of the segments and the algorithm is not always efficient. For example, when the number $m$ is of order $\Omega(n^2)$, then the algorithm runs in time $O(n^2 \log n)$, which is even worse than the straightforward method that picks every pair of segments and computes their intersection point. On the other hand, if the number $m$ is of order $\Omega(n)$, then the algorithm runs efficiently in time $O(n \log n)$.

**Algorithm** SEGMENT-INTERSECTION
GIVEN: $n$ line segments $S_1$, $S_2$, ..., $S_n$
OUTPUT: all intersections of these segments

{Implicitly, we use a vertical line $L$ to sweep the plane. At any moment, the segments intersecting $L$ are stored in ST(ATUS), sorted by the $y$-coordinates of their intersection points with $L$. The event points stored in EV(ENT) are sorted by their $x$-coordinates }

1. Sort the endpoints of the segments and put them in EV;
2. ST $= \emptyset$;
3. **while** EV $\neq \emptyset$ **do**
3.1 $p$ = MINIMUM(EV); DELETE(EV, $p$);
3.2 **if** $p$ is a right-end of some segment $S$ **then**
let $S_i$ and $S_j$ be the two segments adjacent to $S$ in ST;
**if** $p$ is an intersection point of $S$ with $S_i$ or $S_j$ **then** REPORT($p$);
DELETE(ST, $S$);
**if** $S_i$ and $S_j$ intersect at $p_1$ and $x(p_1) \geq x(p)$ **then** INSERT(EV, $p_1$);
3.3 **else if** $p$ is a left-end of some segment $S$ **then**
INSERT(ST, $S$);
let $S_i$ and $S_j$ be the adjacent segments of $S$ in ST;
**if** $p$ is an intersection point of $S$ with $S_i$ or $S_j$ **then** REPORT($p$);
**if** $S$ intersects $S_i$ at $p_1$ with $x(p_1) > x(p)$, INSERT(EV, $p_1$);
**if** $S$ intersects $S_j$ at $p_2$ with $x(p_2) > x(p)$, INSERT(EV, $p_2$);
3.4 **else if** $p$ is an intersection of segments $S_i$ and $S_j$ with $S_i$ on the left of $S_j$ in ST **then**
REPORT($p$);
swap the positions of $S_i$ and $S_j$ in ST;
Let $S_k$ be the segment left to $S_j$ and let $S_h$ be the segment right to $S_i$ in ST;
**if** $S_k$ and $S_j$ intersect at $p_1$ with $x(p_1) > x(p)$ **then** INSERT(EV, $p_1$);
**if** $S_h$ and $S_i$ intersect at $p_2$ with $x(p_2) > x(p)$ **then** INSERT(EV, $p_2$).

Figure 1: Algorithm for SEGMENT INTERSECTION