2. Line segment intersection via sweep algorithm

Introduction. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of line segments in the plane such that the intersection of any two segments is either empty or a single point. We want to find an effective algorithm which looks for all intersections of these segments. Moreover, we ask the algorithm to assign to every intersection the list of all segments on which it lies.

How to compute the intersection of two segments ab a cd? Every point of the segment ab is of the form

$$p = \lambda a + (1-\lambda)b\,, \quad \text{where} \quad \lambda \in [0,1]\,.$$

Similarly a point on cd is

$$q = \mu c + (1-\mu)d\,, \quad \text{where} \quad \mu \in [0,1]\,.$$

The intersection is obtained by solving the equation

$$\lambda a + (1-\lambda)b = \mu c + (1-\mu)d\,.$$

Comparing x and a y-coordinates we get the following system of two linear equations with two unknowns λ a μ

$$\lambda a_x + (1-\lambda)b_x = \mu c_x + (1-\mu)d_x$$
$$\lambda a_y + (1-\lambda)b_y = \mu c_y + (1-\mu)d_y$$

If there is a solution of this system such that $\lambda \in [0,1]$ and $\mu \in [0,1]$, the segments have an intersection. In opposite case the segments do not have an intersection.

A trivial algorithm takes all pairs of segments (s_i, s_j), $1 \le i < j \le n$, and computes their intersections. Since the number of all pairs is $\binom{n}{2}$, the time complexity of such an algorithm is $O(n^2)$. In many cases it happens that the number of intersections is much less than n^2 which makes this algorithm (in these cases) ineffective. Hence we deal with an algorithm which is much more suitable for such cases and works in time

$$O\big((n+k)\log n\big),$$

where k is the number of intersections found. This algorithm uses the *method of sweep line*.

Sweep line algorithm. The method of sweep line is based on the following geometric concept: A horizontal line - called a sweep line - moves in the plane, its path begins above all the segments of S and ends below them. The algorithm slowly moves the line from the top to bottom and along the way performs some actions. These actions occur only when the sweep line passes through prominent points, so-called *events*. Only two type of points are considered as events. Each endpoint of all the given segments is an event and all the intersections already calculated by the algorithm are events. The algorithm detects whether the segments from the nearest left and right neighborhoods of the event have an intersection below the sweep line. If they do, the algorithm classifies the intersection among events into so called *queue*.

FIGURE 2.1

Structures connected with sweep line algorithm. Two structures are usually associated with the sweep line method - an event queue and a binary balanced tree. The queue of events Q is a ordered sequence of points of interest (events) for the algorithm. The layout is "from top to bottom" and "from left to right", which is formally lexicographic ordering: a point p is stored before a point q if

$$p_y > q_y \quad \text{or} \quad (p_y = q_y \text{ and } p_x < q_x)\,.$$

Basically, the event queue determines the order in which the sweep line passes our events. When the sweep line passes an event, this event is removed from the queue, while some other events may be queued. In our algorithm, all the endpoints of the segments from the set S are initially stored in the queue.

FIGURE 2.2 The event queue Q is the sequence $(p_2,\ p_3,\ q_1,\ q_2,\ q_3)$.

FIGURE 2.3 After the sweep line l passes the event p_2, the event queue Q changes into $(r_1,\ p_3,\ q_1,\ q_2,\ q_3)$.

A further structure associated with the sweep line algorithm is a binary balanced tree $\mathcal{T}$. It describes the order of segments which intersect the sweep line l. This order is taken from the left to the right and it is captured in the leaves of the tree $\mathcal{T}$. The nodes of the tree are named after leaves which are rightmost in the left subtree of this node.

FIGURE 2.4

What happens when the sweep line passes an event p**.** Denote $L(p)$ the set of the segments from the set S which have p as its lower endpoint, denote $C(p)$ the set of the segments from S which contain p as its inner point, and finally, let $U(p)$ be the set of the segments which have p as its upper endpoint.

If the position of the sweep line is tightly above the event p, the tree $\mathcal{T}$ contains the segments from $L(p) \cup C(p)$ in its leaves. It can contain additional segments but surely not from the set $U(p)$. When the sweep line passes the point p, the segments $L(p)$ will disappear from the tree $\mathcal{T}$, while the segments from $U(p)$ will be added and the segments from $C(p)$ will change their order. We formally do this in the following way: at first, we remove the leaves in $L(p) \cup C(p)$ from the tree $\mathcal{T}$ and rebalance the tree after each removal. Then we add the segments in $C(p) \cup U(p)$ between the leaves of the tree in the right order and we rebalance the tree again. (In fact, we rebalance the tree after adding any single segment.) If the union $L(p) \cup C(p) \cup U(p)$ contains at least two segments, we mark p as an intersection. In any case we discard the point p from the event queue Q.

FIGURE 2.5

When the sweep line passes through the point p other actions of the algorithm consist of computing the intersections of the segments passing through the point p and adjacent segments (which are the nearest left and right segments to p). We use the order of segments tightly under the event p, eg. a moment after the sweep line passes p.

(1) If $C(p) \cup U(p) = \emptyset$, we find out whether the nearest segment to the left of p (denote it as s_l) and the nearest segment to the right of p (denote it as s_p) have an intersection under the sweep line. If they do, we put the intersection into the queue.

FIGURE 2.6

(2) If $C(p) \cup U(p) \neq \emptyset$, denote s' and s'' the leftmost segment and the rightmost segment from $C(p) \cup U(p)$, respectively. Let s_l be the nearest left neighbour of the segment s' and let s_p be the nearest right neighbour of the segment s''. We look for the intersections $s' \cap s_l$ and $s'' \cap s_p$ under the sweep line l. If some of them exist we put them into the event queue Q.

FIGURE 2.7

ALGORITHMS:

FIND INTERSECTION pseudo.pdf page3. Correct the text in the line Output in the following way:
Output. The set of intersection points among the segments in S such that each intersection point is associated with the segments which contain it.

HANDLE EVENT POINT pseudo.pdf page4

FIND NEW EVENT pseudo.pdf page 5

Animation of the algorithm.

(1) The sweep line l will move from top to bottom. The points in the event queue Q are marked by the black colour.
(2) The segments $s' = s'' = s_1$. The segments s_l and s_p do not exist. We do not compute any intersections.
(3) The segments $s' = s'' = s_2$, $s_l = s_1$ and s_p does not exist. The intersection of s_l and s' exists and lies under the point p_2. We put it into the event queue.
(4) $s' = s'' = s_3$, $s_p = s_1$ and s_l does not exist. The segments s'' and s_p have no intersection.
(5) After the sweep line passes the event p_4, this point is put into the list of intersections. Now, $s' = s_2$, $s'' = s_1$, $s_l = s_3$ and s_p does not exist. The intersection of s' and s_l exists and lies under the point p_4. We put it into the event queue.
(6) $s' = s'' = s_4$, $s_l = s_1$ and s_p does not exist. The segments s' and s_l have no intersection.
(7) s' and s'' do not exist, $s_l = s_2$ and $s_p = s_4$. The intersection of s_l and s_p is identical with the intersections of the segments s_2 and s_3 which has already been included into the event queue.
(8) $s' = s'' = s_5$, $s_l = s_4$ and s_p does not exist. The segments s' and s_l have no intersection.
(9) $s' = s'' = s_6$, $s_p = s_3$ and s_l does not exist. The segments s'' and s_p have no intersection.

(10) After the sweep line passes the event p_9, this point is put into the list of intersections. Now $s' = s_4$, $s'' = s_2$, $s_l = s_6$ and $s_p = s_5$. The intersection of s' and s_l exists and lies under the point p_9. We put it into the event queue. The segments s'' and s_p have no intersection.
(11) The point p_{10} is put into the list of intersections. $s' = s_4$, $s'' = s_6$, s_l does not exist and $s_p = s_2$. The segments s'' and s_p have no intersection.
(12) s',s'' and s_p do not exist, $s_l = s_2$. We do not look for any intersections.
(13) s', s'' and s_l do not exist, $s_p = s_6$. We do not look for any intersections.
(14) s', s'' and s_p do not exist, $s_l = s_6$. We do not look for any intersections.
(15) After passing the point p_{14} the event queue is empty. The algorithm is finished.

Lemma 2.1. *The algorithm finds all intersections.*

Proof. We have to show that every intersection is computed when the sweep line passes an event. Let p be an intersection of two or more segments. Suppose that all the intersections above the point p have been already computed and included into the event queue. FIGURE 2.8

In the FIGURE the events in which p is detected as the intersection are denoted by the red colour. □

Lemma 2.2. *The running time of the algorithm is $O\big((n+k)\log n\big)$, where n is the number of segments and k is the number of intersections found.*

We need the Euler Formula from graph theory to prove it.

Theorem 2.3 (Euler Formula)**.** *Let G be a planar graph with n_v vertices, n_e edges and n_f faces. Then $n_v - n_e + n_f \geq 2$. If G is connected graph, we get the equality*

$$n_v - n_e + n_f = 2\,.$$

Corollary 2.4. *For any planar graph*

$$n_e \leq 3(n_v - 1).$$

Proof. Every bounded face is bordered at least by three edges, every edge is adjacent to at most two faces. That is why the number of all faces (included the unbounded one) is

$$n_f \leq \frac{2n_e}{3} + 1.$$

Substituting this inequality into the Euler inequality, we get

$$n_v - n_e + \frac{2n_e}{3} + 1 \geq 2.$$

From here we get the required inequality. □

Proof of Lemma 2.2. To order $2n$ endpoints of given segments lexicographicly into a queue it takes time $O(n \log n)$. To rebalance a binary tree with at most n leaves after discarding or adding a leaf needs the time $O(\log n)$.

For an event p denote $m(p)$ the number of elements of the set $L(p) \cup C(p) \cup U(p)$. The running time of the algorithm HANDLE EVENT POINT in the event p is

$$O\big(m(p)\log n\big)\,.$$

Hence the running time of the whole algorithm is

$$O\big(n \log n\big) + O\big(\sum_p m(p) \log n\big)\,.$$

To compute this sum we apply the Euler Formula to the planar graph determined by the set S. Verteces of this graph are all endpoints and intersections of segments from the set S. Denote $s(p)$ the degree of the vertex p, i. e. the number of edges coming from p. Obviously, $m(p) \le s(p)$. FIGURE 2.9 $m(p) = 4$, $s(p) = 6$

Now it is sufficient to realize that $\sum_p s(p) = 2n_e$ and $n_v \le 2n + k$. Applying Corollary 2.4 we get

$$\begin{aligned}\sum m(p) \le \sum s(p) = 2n_e \le 6(n_v - 1) \le 6(2n + k - 1)\\ = 12n + 6k - 6 \le 12(n + k)\,.\end{aligned}$$

So the running time of the algorithm is $O\big((n + k) \log n\big)$. □