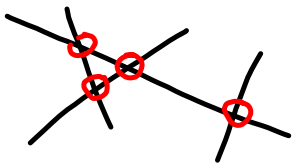
Line segment intersection algorithm

Input : $\{s_1, \ldots, s_n\}$ finite set of line segments



Output: set of intersection points

How do we find intersection point of $\vec{ab}$ & $\vec{cd}$?

- Points on $\vec{ab}$ are $p = \lambda a + (1-\lambda)b$ for $\lambda \in [0,1]$.
  .... $\vec{cd}$ .. $q = \mu c + (1-\mu)d$. $\mu \in [0,1]$.
- Solve $\begin{cases} \lambda a_x + (1-\lambda)b_x = \mu c_x + (1-\mu)d_x \\ \lambda a_y + (1-\lambda)b_y = \mu c_y + (1-\mu)d_y \end{cases}$
  a system of 2 equations, 2 unknowns λ & μ.
- Solution, if it exists, is intersection point.

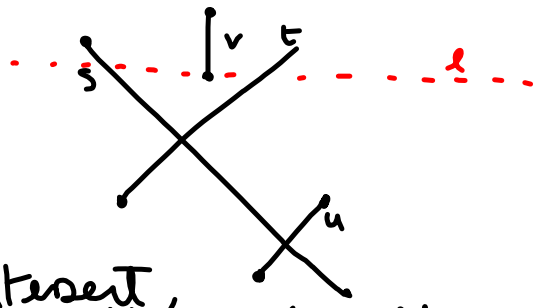
Simple algorithm: given n line segments, test each pair for intersection.
No. of pairs is $\binom{n}{2} = \frac{n \cdot n-1}{2}$.
Time complexity: $O\left(\frac{n}{2}\right) = O(n^2)$.

- Inefficient in practice.
Idea: often fewer than $\binom{n}{2}$ intersections.
Aim: test for fewer intersections.
Will describe "output sensitive algorithm"
with complexity $O((n+k)\log n)$

n: no. of line segments
k: no of intersections.

Called a sweep-line algorithm.

- Intuitive picture :



Imagine line l running from top to bottom down page.

"sweep-line"

- If 2 segments intersect, they must become adjacent / neighbours at some "event point" ↴ endpoint or previously computed intersection point.

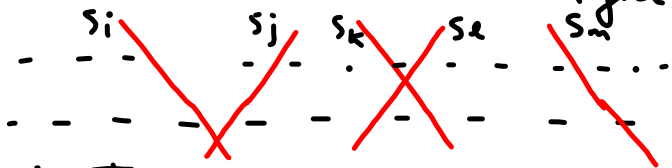
Idea: test line segments for intersection only when they become neighbours.

Structures associated to algorithm

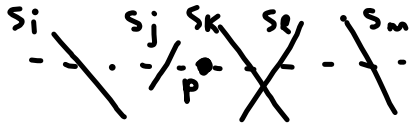
① - "Event queue" Q ~ a balanced binary tree
- leaves of Q store endpoints of line segments & computed intersections "event points"
- Updated as algorithms.
- Order on event points: (top to bottom, left to right)
  i.e. $p < q \Leftrightarrow p_y > q_y$ or $(p_y = q_y \ \& \ p_x < q_x)$.

Example (E-Learning)
- E-Learning says Q only a queue, but need a binary balanced tree.
  - Inserting a new point into Q takes time $O(\log n)$
  - Finding next point takes $O(\log n)$.

② "Status structure" T ~ also balanced binary tree
- T stores order of segments intersecting the sweep-line (left to right)



- order in T at l_1 is $s_i < s_j < s_k < s_\ell < s_m$
- - - - - l_2 is $s_i < s_j < s_\ell < s_k < s_m$
- Segments leaves of b.b.tree - see Fig 2.4 of E-Learning.
- Inserting, deleting segments from T takes time $O(\log n)$
- Finding left, right neighbours of a point p on sweepline takes $O(\log n)$



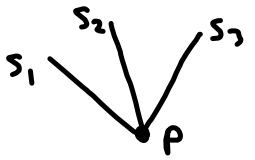
left neighbour of p is s_j
right neighbour of p is s_k.

③ Also, for each event point p, store the sets

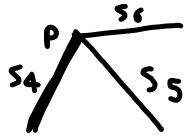
$L(p)$ = { segments with p as lower endpoint }

$U(p)$ = { . . . . . p as upper endpoint }

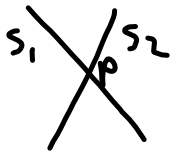
$C(p)$ = { . . . . p as interior point }



$L(p) = \{s_1, s_2, s_3\}$



$U(p) = \{s_4, s_5\}$



$C(p) = \{s_1, s_2\}$

Algorithm:

Input: $\{s_1, \ldots, s_n\}$

Output: intersection points + sets $L(p)$, $U(p)$ & $C(p)$ of segments on which they lie.

1) Initilise an empty "event queue" Q - add endpoints of our segments to Q. Store $L(p)$ & $U(p)$ for each endpoint.

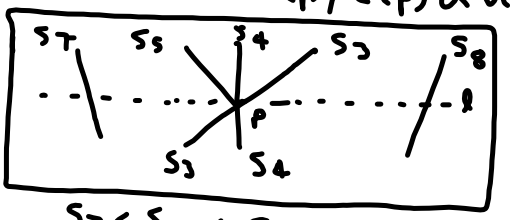
2) Initialise empty tree T.

3) At next event point $p \in Q$

At event point p,

- if $|L(p) \cup C(p) \cup U(p)| \geq 2$, report p as an intersection point together with $L(p)$, $C(p)$ & $U(p)$.
- Delete p from Q.
- Update tree T:
  remove segments from $L(p)$, reverse order of those in $C(p)$, add those in $U(p)$.

(Formally, removing segments from $L(p) \cup C(p)$, rebalancing after each removal, then add segments from $U(p) \cup C(p)$, rebalancing after each insertion.)



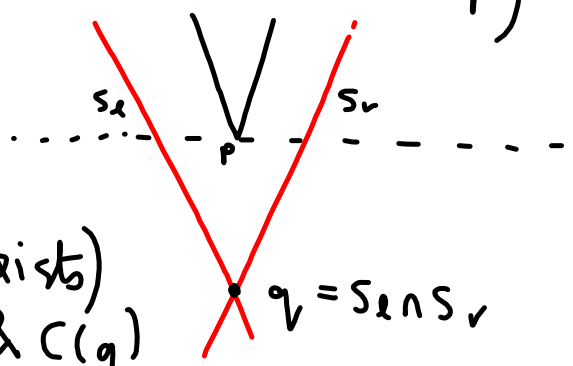
$s_7 < s_5 < s_4 < s_3 < s_8$

$\mapsto s_7 < s_4 < s_3 < s_8$

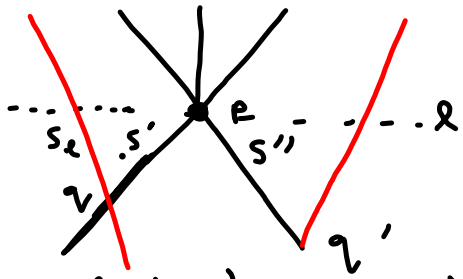
Compute intersections

- IF $U(p) \cup C(p) = \emptyset$ (nothing coming out below p)
- Find left & right neighbours, s_ℓ & s_r, using the tree T, if they exist.
- Calculate $s_\ell \cap s_r$ (if it exists)
- Update sets $L(q)$, $U(q)$ & $C(q)$

& if q is a new intersection point add it to Q.



- Else, $U(p) \cup C(p) \neq \emptyset$ (segments coming out below p)
- let s', s'' be leftmost & rightmost segments in $U(p) \cup C(p) \subseteq T$



- Calculate left neighbour s_l of s' & calculate $q = s_l \cap s'$, & add to Q if new intersection point & update $L(q), U(q), C(q)$.
- Sim. calculate right neighbour s_r of s'' & the int. pt. $q' = s'' \cap s_r \ldots$ (& new event point)

④ When Q is empty, the algorithm terminates.

E-Learning has animation.

Running time

1) At start, order $2n$ endpoints into bin. bal. tree - $O(n \log n)$
2) Let $m(p) = |L(p) \cup C(p) \cup U(p)|$

Actions at event point p:
- add or remove segments to / from T - $O(\log n)$ } total $O(m(p) \log n)$
- Find S', S'', S_ℓ, S_r $O(\log n)$ each } total $O(4 \log n)$
- Computing intersection $O(1)$
- Inserting intersection point in Q - $O(\log n)$

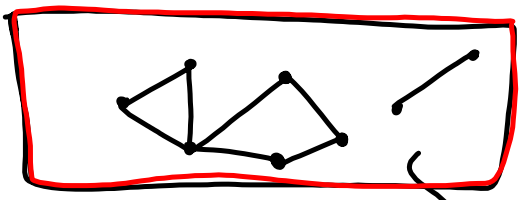
Total: $O(n \log n) + \sum_{p \text{ event}} m(p) O(\log n)$

Will simplify this formula using graph theory.

- To simplify, use Euler's Formula for planar graphs

$V - E + F \geq 2$

$8 - 8 + 3 \geq 2$



- Each edge is adjacent to at most 2 faces, each bounded face adjacent to at least 3 edges.

So $3BF \leq 2E$ so

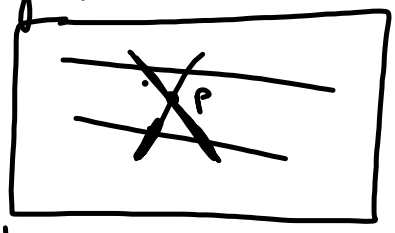
bounded faces

$F - 1 = BF \leq 2E/3$ so $F \leq 2E/3 + 1$

$V - E + F \geq 2 \Rightarrow$

$V - E + 2E/3 + 1 \geq 2 \Rightarrow V - 1 \geq E/3 \Rightarrow E \leq 3(V-1)$.

Segments, endpoints & intersections form a planar graph



events are vertices, endpoints & intersections.

Degree of vertex p
$s(p)$ = no. of edges coming out of p.

- In above example, $s(p) = 4$.
- . . . . . , $m(p) = 2$.
- In general, $m(p) \leq s(p)$.
- Then $\sum_{p \text{ an ew. pt}} m(p) \leq \sum_{p \text{ vertex}} s(p) = 2E$ (each edge in planar graph has exactly 2 endpoints)

$$\leq 6(V-1)$$
$$\leq 6(2n + k - 1)$$

(n: endpoints, k: intersection pts)

$$\leq 12(n+k).$$

Complexity:

$$O(n \log n) + \sum m(p) O(\log n)$$
$$\leq O(n \log n) + 12^{p}(n+k) O(\log n)$$
$$= O((n+k) \log n).$$