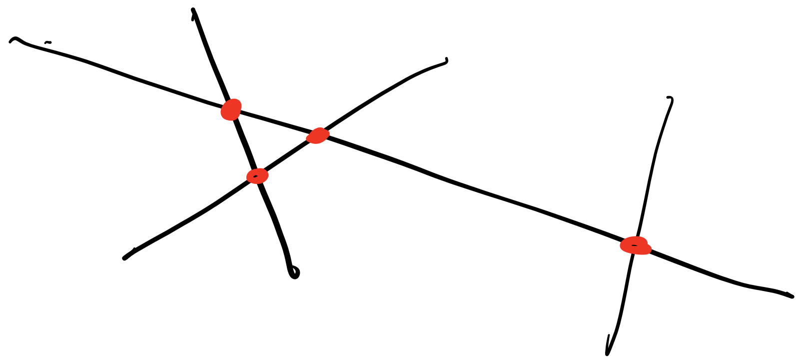
Line segment intersection algorithm

Input $\{s_1, \ldots, s_n\}$ Finite set of line segments.



Output: set of intersection points

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

Points on $\overrightarrow{ab}$ are $p = \lambda a + (1-\lambda)b$ for $\lambda \in [0,1]$.
$\overrightarrow{cd}$ ... $q = \mu c + (1-\mu)d$ ... $\mu \in [0,1]$.

Solve
$$\begin{cases} \lambda a_x + (1-\lambda)b_x = \mu c_x + (1-\mu)d_x \\ \lambda a_y + (1-\lambda)b_y = \mu c_y + (1-\mu)d_y \end{cases}$$

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 is $O\binom{n}{2} = O(n^2)$.

Inefficient! We will describe a more efficient algorithm.

Idea: often fewer than $\binom{n}{2}$ intersections.

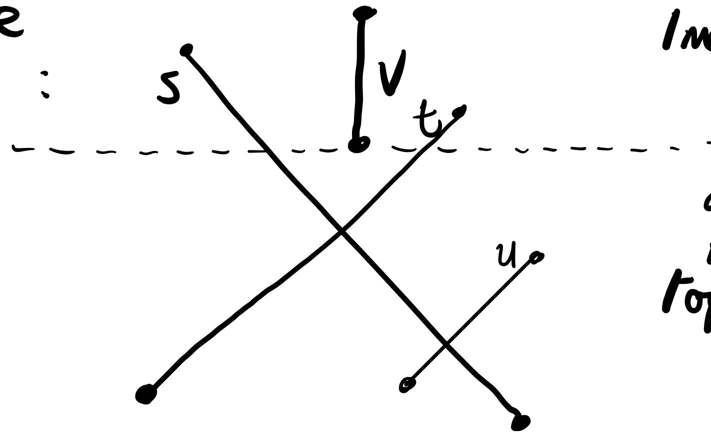
Aim: test for fewer intersections.

Will describe "output sensitive" algorithm with complexity

$$O(\underbrace{(n}_{\text{no of line segments}} + \underbrace{k}_{k \text{ number of intersections found}}) \log n)$$

Called a "sweep line algorithm".

Intuitive picture:



Imagine a line l running down the page from top to bottom.

- If 2 segments intersect, they must become adjacent / neighbours at some "event point"

  $\underbrace{\text{event point}}_{}$ endpoint or an earlier 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.

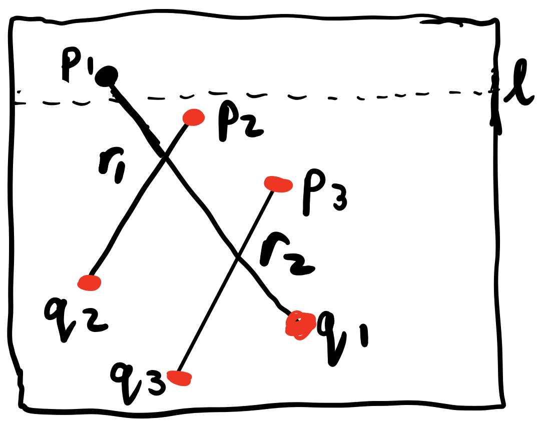
- Leaves of Q store the endpoints & computed intersections.

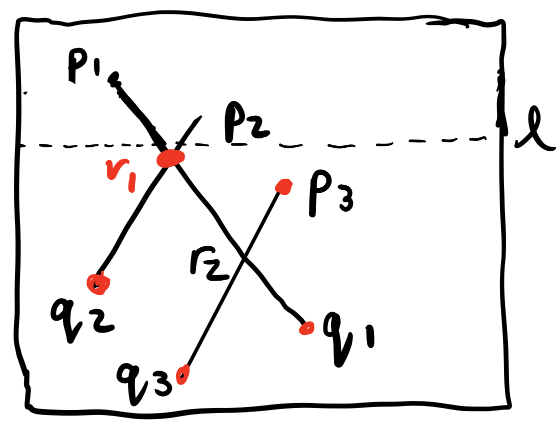
event points

- Updated as algorithm runs.
- Order on events points in Q is lexicographic: (top to bottom, left to right)

ie. $p < q \iff p_y > q_y$ or $(p_y = q_y \;\&\; p_x < q_x)$

Example (E-Learning 2.2)





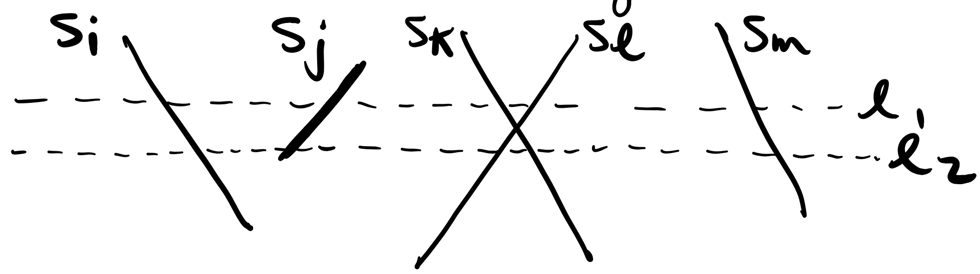
$p_2 < p_3 < q_2 < q_1 < q_3$ $\qquad$ $r_1 < p_3 < q_2 < q_1 < q_3$

(E-Learning: it says we only Q to be a queue, but need a bal. bin. tree.)

- Inserting a new point to Q takes time $O(\log n)$.
- Finding next point in Q takes $O(\log n)$

(2) "Status structure" T ~ also a balanced binary tree

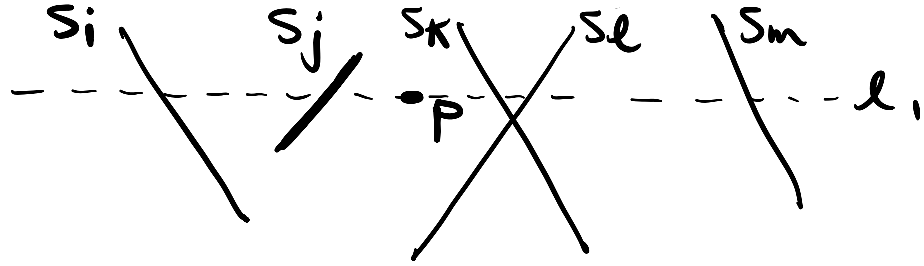
- T stores the order of segments intersecting the sweep-line (left to right)



- Order in T at ℓ_1 is $s_i < s_j < s_k < s_\ell < s_m$

  — — — — — at ℓ_2

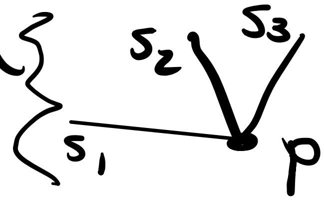
  $s_i < s_j < s_\ell < s_k < s_m$
- Ordered segments - leaves of bal. bin. tree ~ see Fig 24 in E-Learning
- Inserting, deleting segments from T takes time $O(\log n)$
- Finding left, right neighbours of a point p on sweep-line takes time $O(\log n)$



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

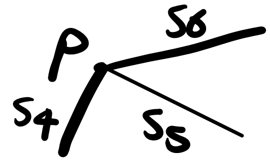
(3) Also, we store for each event point p, the sets

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



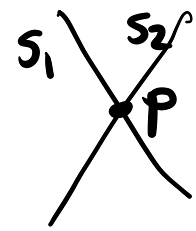
$L(p) = \{s_1, s_2, s_3\}$

$U(p) = \{$ - - - - - - p as upper endpoint $\}$



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

$C(p) = \{$ - - - - - - p an interior point $\}$



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

Algorithm

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

Output : intersection points p plus sets $L(p), U(p)$ & $C(p)$ of segments.

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

2) Initialise empty tree T.

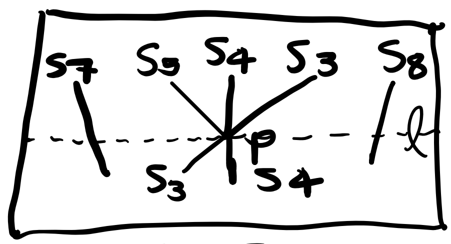
3) At next event point $p \in Q$

a) if p intersection point, report it with $L(p), C(p)$ & $U(p)$.

b) Delete p from Q.

c) Update tree T:

remove segments from $L(p)$, reversing order of those in $C(p)$, add those of $U(p)$



$s_7 < s_5 < s_4 < s_3 < s_8$

$\mapsto s_7 < s_4 < s_3 < s_8$

d) Compute intersections & add to Q.

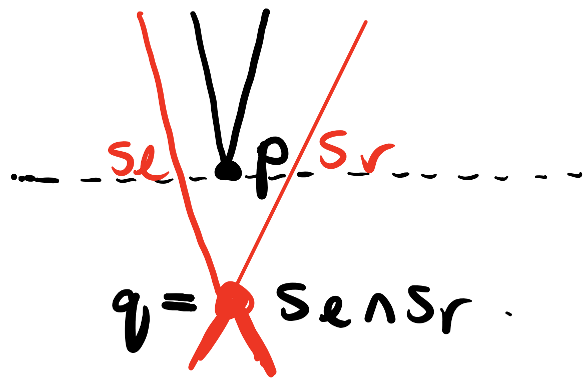
4) When Q empty, stop.

Details on d) - compute intersections

I

if $U(p) \cup C(p) = \emptyset$ (nothing coming out below p)

Find left & right neighbours of p, s_ℓ & s_r, using the tree T, if they exist.

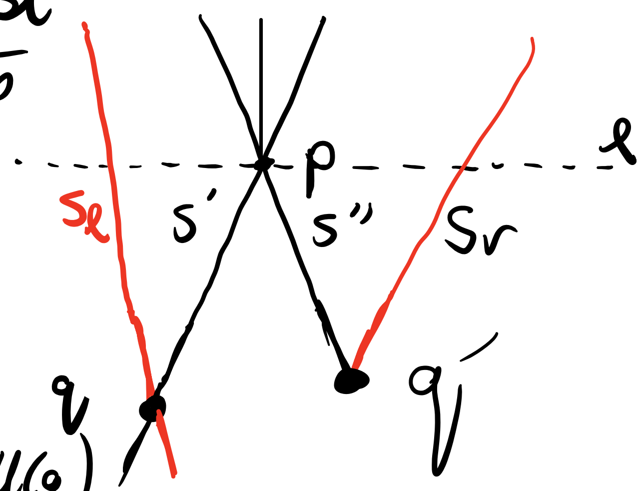


- Calculate $s_\ell \cap s_r$
- Update sets $L(q)$, $U(q)$ & $C(q)$ & if q is a new intersection point we add it to Q.

2

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 $q = s_\ell \cap s'$ – update $L(q), C(q), U(q)$ & add q to Q if it is a new intersection point.



- Similarly calculate right neighbour s_r of s'' & the intersection $q' = s'' \cap s_r$ ....

Running time

1) At beginning of alg., order $2n$ endpoints into bal. binary tree Q $- O(n\log n)$

2) Let $m(p) = L(p) \cup C(p) \cup U(p)$

Actions at event point p:

- add or remove a segment to/from T - $O(\log n)$ $\Big\}$ Total $O(m(p)\log n)$
- Find s', s'', s_ℓ, s_r $O(\log n)$ each (so $O(4\log n)$ total)
- Computing intersection $O(1)$
- Inserting int. point in Q - $O(\log n)$

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

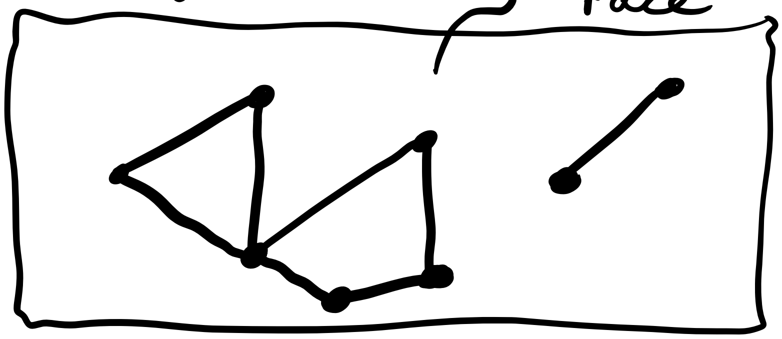
Will simplify this expression using a little graph theory.

To simplify, we use <u>Euler's formula</u> for planar graphs

count unbounded face

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

$$8 - 8 + 3 \geq 2$$



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

so $3\,BF \leq 2E$ so

bounded faces

$$F - 1 = BF \leq 2E/3 \text{ so}$$

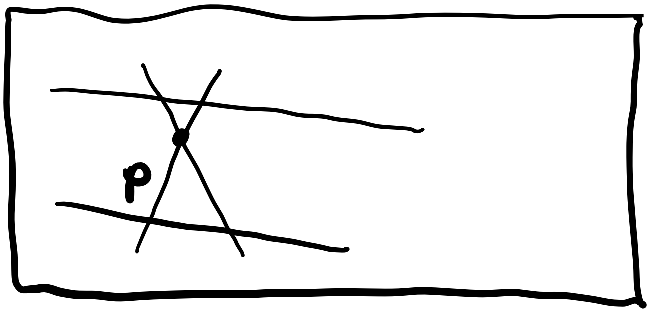
$$F \leq 2E/3 + 1$$

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

$$V - E + 2E/3 + 1 \geq 2 \Rightarrow$$

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

Segments, endpoints & intersections form a planar graph



events are the vertices - endpoints & intersections.

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

- In the above planar graph, $s(p) = 4$
- In this example, $m(p) = 2$.

In general, $m(p) \leq s(p)$.

Then

$$\sum_{p \text{ an event pt.}} m(p) \leq \sum_{p} s(p)$$

$$= 2E \quad \left(\text{each edge in a planar graph has exactly two endpoints}\right)$$

$$\leq 6(V-1)$$

$$\leq 6(\underbrace{2n}_{\text{endpoints}} + \underbrace{k}_{\text{int. points}} - 1)$$

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

Complexity: $O(n\log n) + \sum_{p} m(p)\, O(\log n)$

$\leq O(n\log n) + 12(n+k)O(\log n) = O((n+k)\log n)$.

This is the output sensitive complexity that we claimed at the beginning of the lecture.

Sweep-line algorithm will also be used next week for "map overlay" and in later weeks.