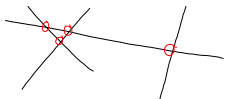
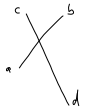
Line segment intersection algorithm

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



Output: set of intersection points

zář 27-15:30



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

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

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 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.

zář 27-16:05

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

Aim: test for fewer pairs of intersections.

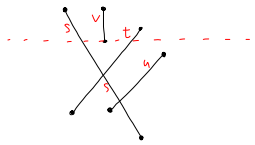
Will describe "output sensitive" algorithm with running time $O((n+k)\log n)$

k number of intersections found.

- Called a "sweep line algorithm"

zář 27-16:13

Intuitive picture



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

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

Idea: test lines for intersection just when they become neighbours.

zář 27-16:18

Structures associated with the algorithm

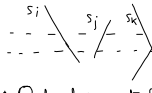
① - A queue Q consisting of endpoints of segments & computed intersection.

- Q will be updated as algorithm runs.
- Points in Q ordered lexicographically (top to bottom, left to right) ie. $p < q$ (p appears before q in the queue Q) if $p_y > q_y$ or ($p_y = q_y$ & $p_x < q_x$).

zář 27-16:25

② Balanced binary tree T ~ status structure:

stores order of segments intersecting the sweep line.



Order at l_1 is $s_i < s_j < s_k < s_e < s_m$.

Order at l_2 is $s_i < s_j < s_e < s_k < s_m$.

- Ordered segments form leaves of a balanced binary tree - see Figure 2.4 in E-learning.
- Inserting, removing points from tree takes time $O(\log n)$.
- Using tree we can find left & right neighbours of point p in time $\log n$ – test if p lies to left & descend.

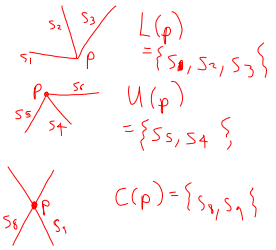
zář 27-16:29

Also, we will store for each event point p, sets:

$L(p)$: segments with p as lower endpoint

$U(p)$: . . . . . p as upper endpoint.

$C(p)$: . . . . . p an interior endpoint.



zář 27-16:40

Algorithm:

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

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

1) Initialise empty queue Q – add endpoints of segments to Q. Store $L(p)$ & $U(p)$ for endpoints.

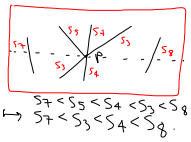
2) Initialise empty tree T.

3) At next event point $p \in Q$ →

zář 27-16:45

At event point p,

- if $|L(p) \cup C(p) \cup U(p)| \geq 2$, report p as an intersection point, with $L(p)$, $C(p)$, $U(p)$.
- Delete p from Q.
- Update tree T:
  - remove segments from $L(p)$, reverse order of those in $C(p)$.
  - Formally remove segments from $L(p) \cup C(p)$ – rebalance tree after each removal.
  - Add segments from $U(p) \cup C(p)$ – rebalancing after each insertion.



zář 27-16:49

Compute intersections

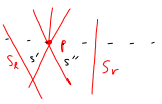
If $U(p) \cup C(p) = \emptyset$

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

- Calculate $q_1 = s_\ell \cap s_r$.
- Update sets $L(q_1), U(q_1), C(q_1)$ & if q_1 is a new intersection point add q_1 to queue.

Else $U(p) \cup C(p) \neq \emptyset$:

- let s' & s'' be leftmost & rightmost segments in $U(p) \cup C(p) \subseteq T$.
- Calculate left neighbour s_ℓ of s' & right neighbour s_r of s''.
- Calculate $q_1 = s_\ell \cap s'$ – update $L(q_1), C(q_1), U(q_1)$ & add q to Q if new int. point.
- Calculate $q_2 = s'' \cap s_r$ . . .



zář 27-16:58

Running time

1) At beginning of algorithm, order $2n$ endpoints into queue — $O(n \log n)$

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

Actions at p:

- adding or removing a segment to/from T — $O(\log n)$
- finding s', s'', s_ℓ, s_r . . . $O(\log n)$ each
- computing intersection $O(1)$
- inserting intersection pt in Q – $O(\log n)$

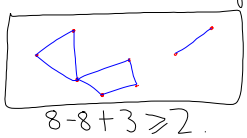
Total: $O(m(p) \log n)$

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

zář 27-17:18

To simplify, use Euler's formula for planar graphs:

$V - E + F \geq 2$



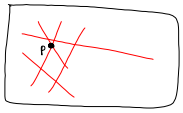
- Each edge adjacent to at most 2 faces, each bounded face adjacent to at least 3 edges

$\Rightarrow F \leq \frac{2E}{3} + 1$.

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

zář 27-17:26

Segments, endpoints & intersections form planar graph



events - vertices : endpoints & intersections.

- Degree of vertex p - call $s(p)$ - no. of edges coming out p.
- Eg. $s(p) = 4$.
  - $m(p) = 2$. Generally $m(p) \leq s(p)$.

$$\sum_p m(p) \leq \sum_p s(p) = 2E \leq 6(V-1) \leq 6(2n+k-1)$$

each edge in a graph has exactly 2 endpoints

endpoints: intersection points $\leq 12(n+k)$.

Complexity: $O(n \log n) + \sum_p m(p) O(\log n) \leq O(n \log n) + 12(n+k) O(\log n)$
$= O(n+k) \log n$.

zář 27-17:34