

Definition

- given n line segments in the plane, find all intersections

Intersection Check

- other segment's endpoints must be on different sides
- for both segments
- check 4 triple-orientations

$$\chi(a, b, c) = \text{sign} \left| \begin{pmatrix} 1 & 1 & 1 \\ a_x & b_x & c_x \\ a_y & b_y & c_y \end{pmatrix} \right| = \begin{cases} +1 & \text{ccw} \\ 0 & \text{coll.} \\ -1 & \text{cw} \end{cases}$$

- counterclockwise = left

Observations

- intersection check takes constant time
- up to $\Theta(n^2)$ intersections
 - worst case takes $\Omega(n^2)$
 - output-sensitive algorithm needed

Plane Sweep Idea

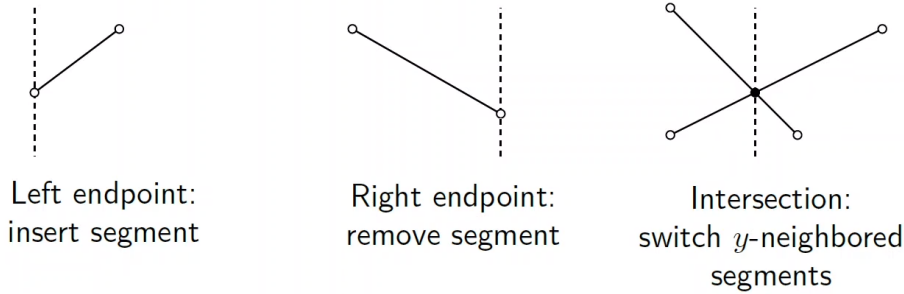
- if two segments intersect $\rightarrow x$ -intervals overlap
 - inverse not always true
- scan from left to right through all x -values with vertical L
 - at every point
 - * consider segments hit by L
 - * check for intersection
 - intersections must be neighbored on L
 - * at some point

Algorithm:

- Maintain y -order on L .
- Check y -neighbored segments for intersections

The plane-sweep is *event-based*, where an event is a change in the y -order.

- events



–

Implementation

- used data structures
 - X
 - * contains x -coordinates of known, future events
 - ◆ incoming start and endpoints
 - * operations
 - ◆ insert
 - ◆ remove x -minimum
 - [[Queue]], [[Heap]]
 - search tree Y
 - * contains y -ordered set of segments intersecting L
 - * operations
 - ◆ insert (startpoints)
 - ◆ remove (endpoints)
 - ◆ switch neighbours (intersection)
 - ◆ dictionary, [[(2-4)-Bäume]]
- pseudocode

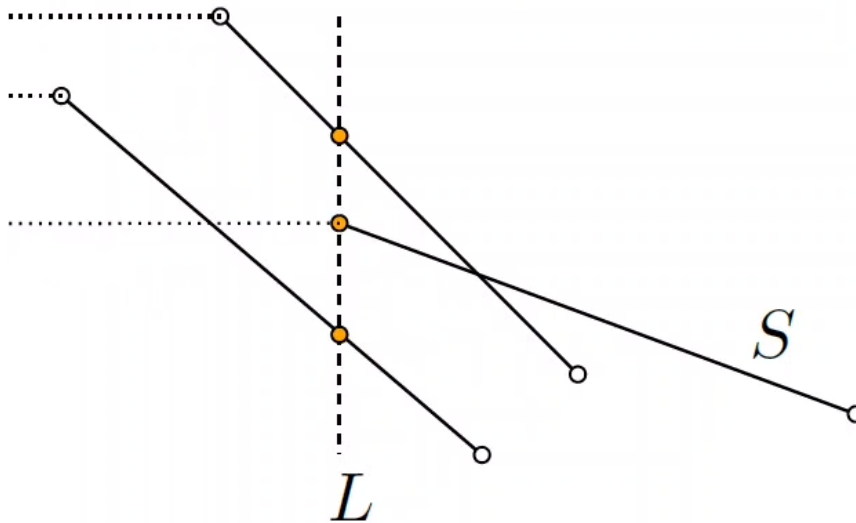
$X = \emptyset, Y = \emptyset$

Insert x -coordinates of the start- and endpoints of all segments into X .

while $X \neq \emptyset$:

 1. Get minimum m of X and remove it from X .
 2. IF m left endpoint THEN insert its segment into Y
 ELSE IF m right endpoint THEN remove its segment from Y
 ELSE (m intersection) switch the order of the intersecting segments in Y
 3. FOR all new neighboring pairs in Y (at most two):
 IF neighboring pair intersects in p AND p is to the right of L
 THEN report p and insert x -coordinate of p into X

–
- y -order depends on actual y -coordinates at this x -coordinate



- at most two new neighbours
 - above and below new minimum x
 - easier to find if linked
 - * maybe link leaves of [(2-4)-Bäume] with pointers

Analysis

n segments, k intersections, $0 \leq k \leq \binom{n}{2} = \Theta(n^2)$

- **In X:** Per segment we insert two events, per intersection one. We later remove all of these events.
 $\Rightarrow O(n + k)$ space, and
 $O((n + k) \log(n + k)) = O((n + k) \log n)$ time
- **In Y:** We insert and remove every segment exactly once. For every intersection we switch a pair of segments. $O(1)$ per switch, if we link intersections to their segments (linked leaves in the 2-4-tree).
 $\Rightarrow O(n)$ space and $O(n \log n + k)$ time

• **In total:** $O((n + k) \log n)$ **time** and $O(n + k)$ **space**.

- detecting same intersection twice possible
 - must be prevented with check
 - does not affect time complexity

If we insert for every segment only the first not yet reached intersection into X , the algorithm uses only $O(n)$ space with the same running time.

- can be further reduced to

- $O(n \log n + k)$
- stop at first intersection possible
 - The algorithm works as intersection-**detector** in time $O(n \log n)$ and optimal space $O(n)$ (set $k = 0$ or $k = 1$).