Plane Sweep Algorithms
Carola Wenk
Line Segment Intersection

• Input: A set $S=\{s_1, \ldots, s_n\}$ of (closed) line segments in $\mathbb{R}^2$

• Output: All intersection points between segments in $S$
General position

Assume that “nasty” special cases don’t happen:
- No line segment is vertical
- Two segments intersect in at most one point
- No three segments intersect in a common point
Line Segment Intersection

- $n$ line segments can intersect as few as 0 and as many as $\binom{n}{2} = O(n^2)$ times.
- Simple algorithm: Try out all pairs of line segments
  → Takes $O(n^2)$ time
  → Is optimal in worst case
- Challenge: Develop an output-sensitive algorithm
  - Runtime depends on size $k$ of the output
  - Here: $0 \leq k \leq (n^2+n)/2$
  - Our algorithm will have runtime: $O((n+k) \log n)$
  - Best possible runtime: $O(n \log n + k)$
    → $O(n^2)$ in worst case, but better in general
Plane Sweep: An Algorithm Design Technique

- Simulate sweeping a vertical line from left to right across the plane.
- Maintain **cleanliness property**: At any point in time, to the left of sweep line everything is clean, i.e., properly processed.
- **Sweep line status**: Store information along sweep line
- **Events**: Discrete points in time when sweep line status needs to be updated

**Algorithm** Generic_Plane_Sweep:

Initialize sweep line status $S$ at time $x = -\infty$

Store initial events in event queue $Q$, a priority queue ordered by $x$-coordinate

while $Q \neq \emptyset$

  // extract next event $e$:
  $e = Q$.extractMin();

  // handle event:
  Update sweep line status
  Discover new upcoming events and insert them into $Q$
Plane sweep algorithm

- **Cleanliness property:**
  - All intersections to the left of sweep line $l$ have been reported

- **Sweep line status:**
  - Store segments that intersect the sweep line $l$, ordered along the intersection with $l$.

- **Events:**
  - Points in time when sweep line status changes combinatorially (i.e., the order of segments intersecting $l$ changes)
  - Endpoints of segments (insert in beginning)
  - Intersection points (compute on the fly during plane sweep)

Algorithm Generic_Plane_Sweep:

- **Initialize sweep line status $S$ at time $x = -\infty$**
- **Store initial events in event queue $Q$, a priority queue ordered by $x$-coordinate**
- **while** $Q \neq \emptyset$
  - // extract next event $e$:
  - $e = Q$.extractMin();
  - // handle event:
  - Update sweep line status
  - Discover new upcoming events and insert them into $Q$
Event Handling

1. Left segment endpoint
   - Add segment to sweep line status
   - Test adjacent segments on sweep line \( l \) for intersection with new segment (see Lemma)
   - Add new intersection points to event queue
Event Handling

2. Intersection point
   - Report new intersection point
   - Two segments change order along $l$
     → Test new adjacent segments for new intersection points (to insert into event queue)

Note: “new” intersection might have been already detected earlier.
Event Handling

3. Right segment endpoint
   - Delete segment from sweep line status
   - Two segments become adjacent. Check for intersection points (to insert in event queue)
Sweep Line Status

- Store segments that intersect the sweep line $l$, ordered along the intersection with $l$.
- Need to insert, delete, and find adjacent neighbor in $O(\log n)$ time.
- Use **balanced binary search** tree, storing the order in which segments intersect $l$ in leaves.

![Diagram of Sweep Line Status](image-url)
Balanced Binary Search Tree
-- a bit different

$key[x]$ is the maximum key of any leaf in the left subtree of $x$. 
Balanced Binary Search Tree -- a bit different

$key[x]$ is the maximum key of any leaf in the left subtree of $x$. 
Event Queue

• Need to keep events sorted:
  – Lexicographic order (first by $x$-coordinate, and if two events have same $x$-coordinate then by $y$-coordinate)
• Need to be able to remove next point, and insert new points in $O(\log n)$ time
• Need to make sure not to process same event twice
  ⇒ Use a priority queue (heap), and possibly extract multiples
  ⇒ Or, use balanced binary search tree
Runtime

• Sweep line status updates: $O(\log n)$
• Event queue operations: $O(\log n)$, as the total number of stored events is $\leq 2n + k$, and each operation takes time
  
  $O(\log(2n+k)) = O(\log n^2) = O(\log n)$

  $k = O(n^2)$

• There are $O(n+k)$ events. Hence the total runtime is $O((n+k) \log n)$
Plane Sweep: An Algorithm Design Technique

• Plane sweep algorithms (also called sweep line algorithms) are a special kind of incremental algorithms
• Their correctness follows inductively by maintaining the cleanliness property
• Common runtimes in the plane are $O(n \log n)$:
  – $n$ events are processed
  – Update of sweep line status takes $O(\log n)$
  – Update of event queue: $O(\log n)$ per event