Plane Sweep Algorithms and Segment Intersection

Carola Wenk
Closest Pair

- **Problem:** Given $P \subseteq \mathbb{R}^2$, $|P| = n$, find the distance between the closest pair in $P$
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
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 for Closest Pair

- **Problem:** Given $P \subseteq \mathbb{R}^2$, $|P| = n$, find the distance of the closest pair in $P$

- **Sweep line status:**
  - Store current distance $\Delta$ of closest pair of points to the left of sweep line
  - Store points in $\Delta$-strip left of sweep line
  - Store pointer to leftmost point in strip

- **Events:** All points in $P$. No new events will be added during the sweep.
  $\rightarrow$ Presort $P$ by $x$-coordinate.

---

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 for Closest Pair, II

\[ O(n \log n) \]
- Presort \( P \) by \( x \)-coordinate
- How to store points in \( \Delta \)-strip?
  - Store points in \( \Delta \)-strip left of sweep line in a balanced binary search tree, ordered by \( y \)-coordinate
  - Add point, delete point, and search in \( O(\log n) \) time
- **Event handling:**
  - New event: Sweep line advances to point \( p \in P \)
  - Update sweep line status:
    - Delete points outside \( \Delta \)-strip from search tree by using previous leftmost point in strip and \( x \)-order on \( P \)
    - Compute candidate points that may have distance \( \leq \Delta \) from \( p \):
      - Perform a search in the search tree to find points in \( \Delta \)-strip whose \( y \)-coordinates are at most \( \Delta \) away from \( p \).
      - Because of the cleanliness property each pair of these points has distance \( \leq \Delta \).
    - A \( \Delta \times 2\Delta \) box can contain at most 6 such points.
  - Check distance of these points to \( p \), and possibly update \( \Delta \)
- No new events necessary to discover

**Total runtime:** \( O(n \log n) \)
Balanced Binary Search Tree
-- a bit different

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

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

\textbf{Range-Query}([7, 41])
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
Geometric Intersections

• Important and basic problem in Computational Geometry
• Solid modeling: Build shapes by applying set operations (intersection, union).
• Robotics: Collision detection and avoidance
• Geographic information systems: Overlay two subdivisions (e.g., road network and river network)
• Computer graphics: Ray shooting to render scenes
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 $S$
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: $0 \leq k \leq c n^2$, where $c$ is a constant
  - 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
Complexity

• Why is runtime $O(n \log n + k)$ optimal?
• The element uniqueness problem requires $\Omega(n \log n)$ time in algebraic decision tree model of computation (Ben-Or ’83)
• Element uniqueness: Given $n$ real numbers, are all of them distinct?
• Solve element uniqueness using line segment intersection:
  – Take $n$ numbers, convert into vertical line segments. There is an intersection iff there are duplicate numbers.
  – If we could solve line segment intersection in $o(n \log n)$ time, i.e., strictly faster than $\Theta(n \log n)$, then element uniqueness could be solved faster. Contradiction.
Intersection of two line segments

- Two line segments $ab$ and $cd$
- Write in terms of convex combinations:
  \[
  p(s) = (1-s) a + s b \quad \text{for } 0 \leq s \leq 1 \\
  q(t) = (1-t) c + t d \quad \text{for } 0 \leq t \leq 1 \\
  \]
  Intersection if $p(s)=q(t)$

  \Rightarrow \text{Equation system}

  \[
  (1-s) a_x + s b_x = (1-t) c_x + t d_x \\
  (1-s) a_y + s b_y = (1-t) c_y + t d_y
  \]

- Solve for $s$ and $t$. In division, if divisor $= 0$ then line segments are parallel (or collinear). Otherwise get rational numbers for $s$ and $t$. Either use floating point arithmetic or exact arithmetic.
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 $Q$
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
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
  $\Rightarrow$ Use a priority queue (heap), and possibly extract multiples
  $\Rightarrow$ Or, use balanced binary search tree
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.
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)
Intersection Lemma

- **Lemma**: Let $s$, $s'$ be two non-vertical segments whose interiors intersect in a single point $p$. Assume there is no third segment passing through $p$. Then there is an event point to the left of $p$ where $s$ and $s'$ become adjacent (and hence are tested for intersection).

- **Proof**: Consider placement of sweep line infinitesimally left of $p$. $s$ and $s'$ are adjacent along sweep line. Hence there must have been a previous event point where $s$ and $s'$ become adjacent.
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 $O((n+k) \log n)$