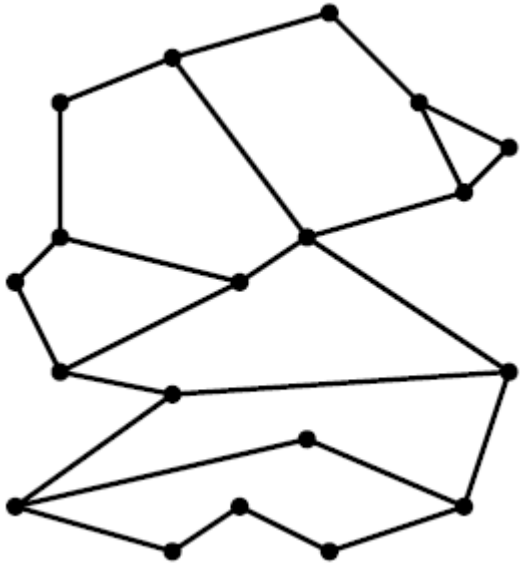
KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
KAIST
한국과학기술원
SINCE 1971

Point Location

Geometric
Computing

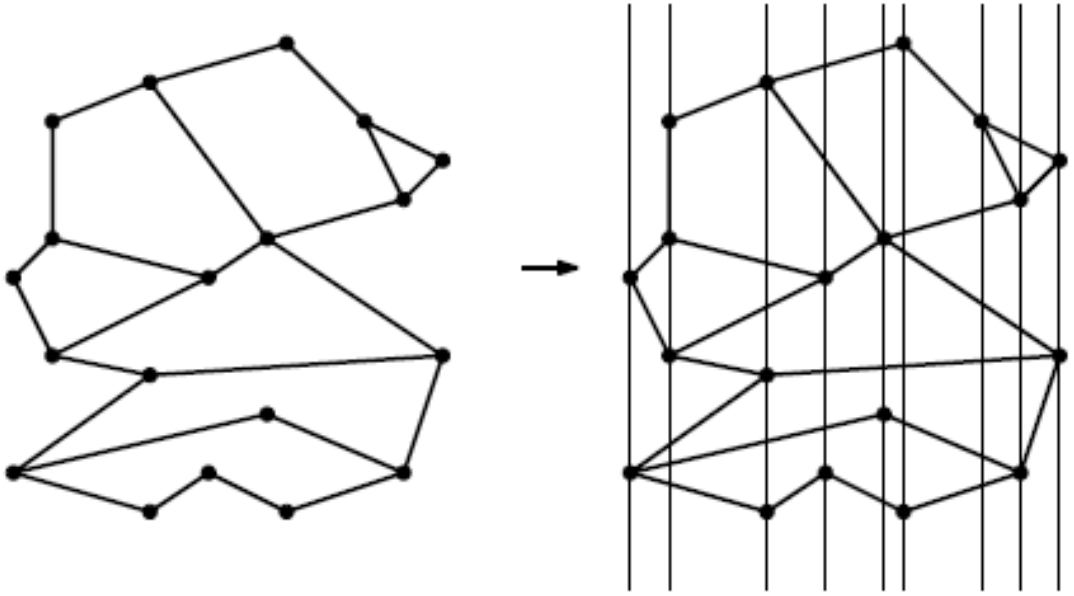
Point Location

Given a polygonal subdivision S of the plane with n edges, preprocess S so that given a query point q, we can efficiently determine which face *f* of S contains q.



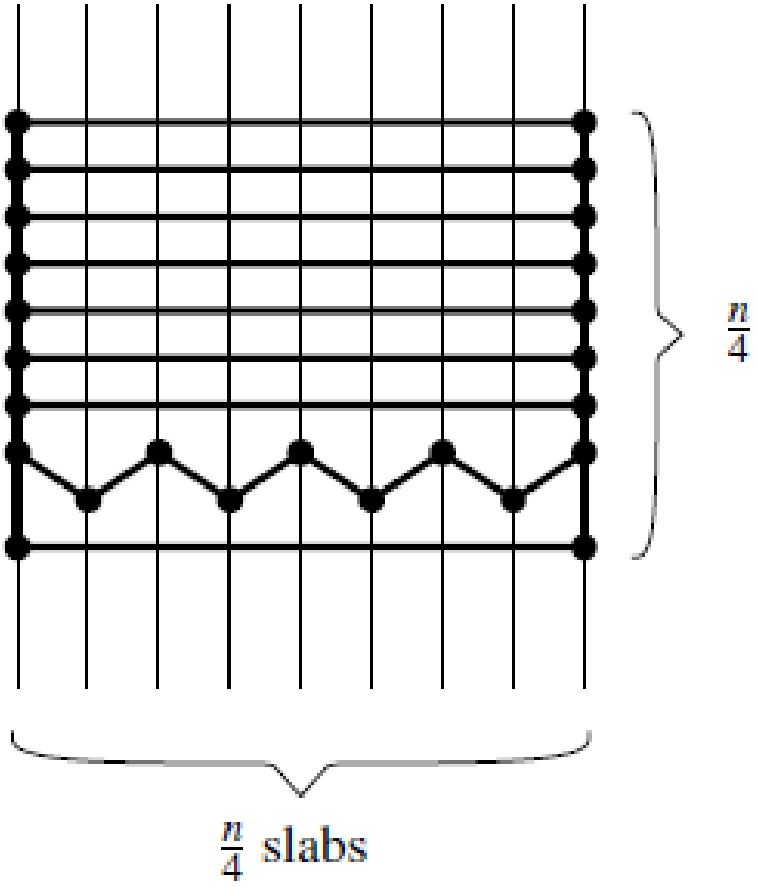
Partition into slab

- Store x-coord of vertices in sorted array
- We can determine in O(log n) time the slab containing a query point q.
- In a slab, sort edges from top to bottom.
- Query time O(log n)



Partition into slab

- Space $O(n^2)$

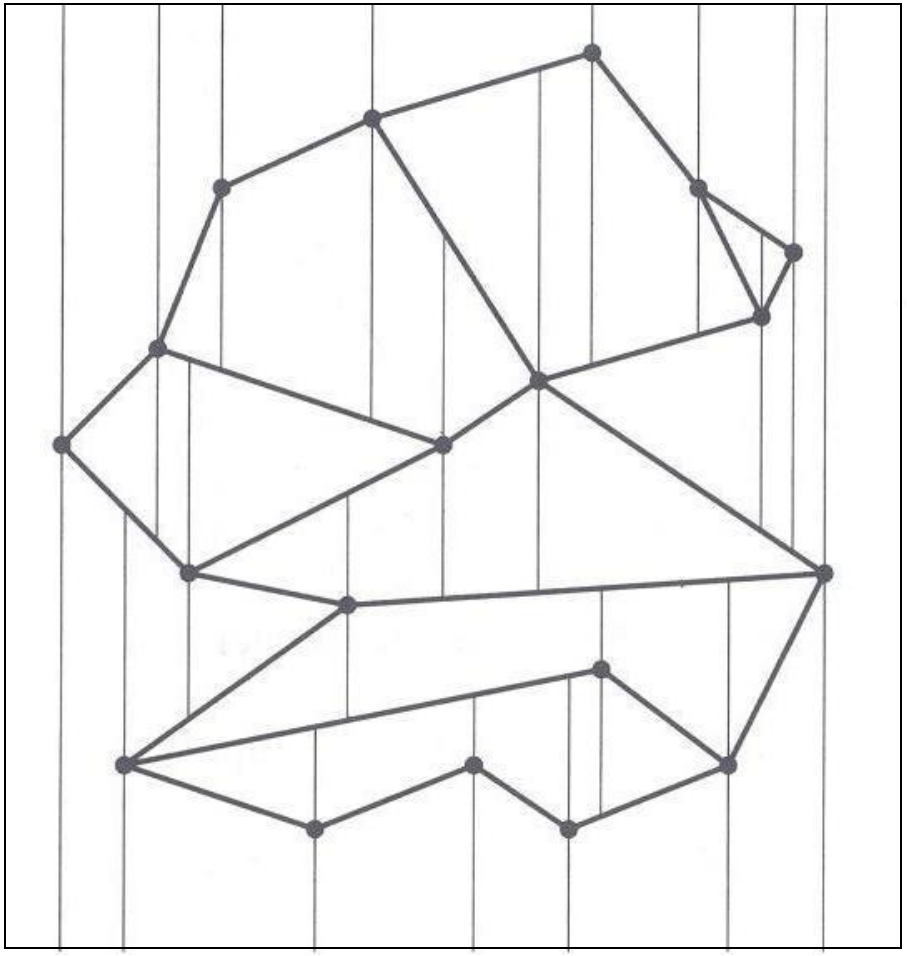


Assumption

- The input is a set S of *n* nonvertical line segments.
- They do not intersect except possibly at their endpoints.
- No two distinct endpoints have the same x-coordinates.
- The initial subdivision is contained within a large bounding rectangle.

Trapezoidal Map

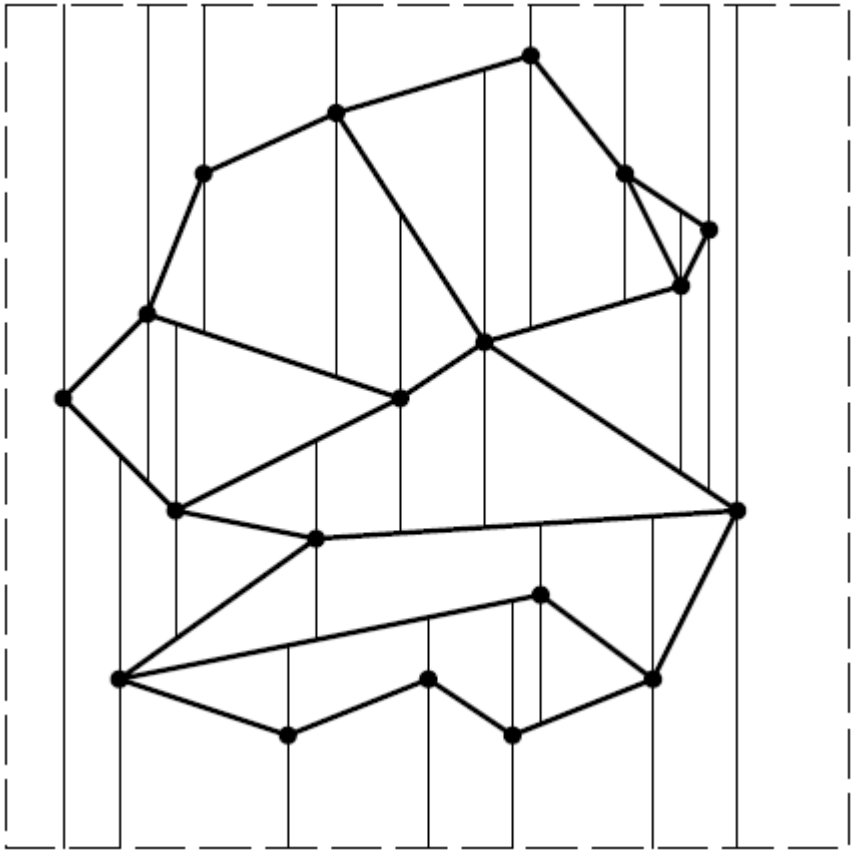
Trapezoidal Map T(S) of S is obtained by drawing vertical extensions from every endpoint of segments in S until they hit another segment or boundary of R.



R

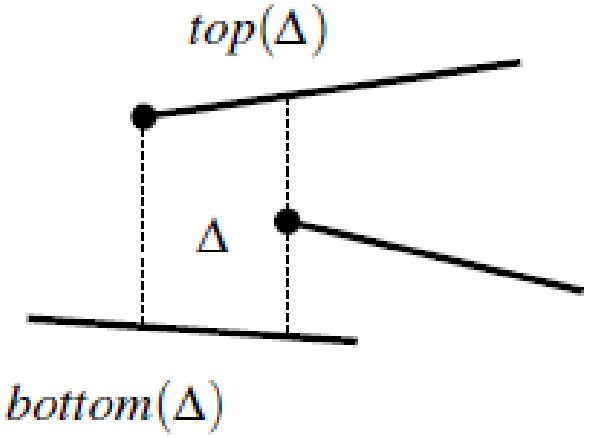
Observation

- Each face in the trapezoidal map of a set S of line segments has one or two vertical side and exactly two non vertical sides.
- Left or right side might degenerate to a line segment of length 0.



Observation

- Each trapezoid is defined by 4 entities from the original subdivision – a segment on top, a segment on bottom, a bounding vertex on the left, a bounding vertex on the right.



Complexity

- Lemma: The trapezoidal map T(S) of a set S of n line segment in general position contains at most 6n+4 vertices and at most 3n+1 trapezoids.

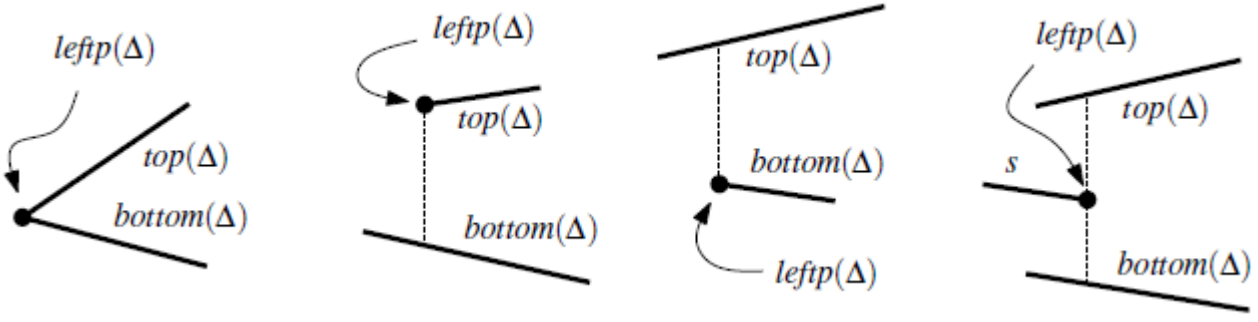
Pf) Maximum # of vertices :

vertices of R(4) + endpoints of segments(2n) + endpoints of two vertical extensions(4n)

Complexity

Proof) Maximum # of Trapezoids: 3n+1

- For each trapezoid, its left side is bounded by a vertex in the original subdivision.
- Right endpoint of each segment can be left bounding vertex for exactly one trapezoid. (= n)
- Left point of each segment can be left bounding vertex of at most two different trapezoids. (= 2n)
- Left side of R can bound 1 trapezoid. (= 1)

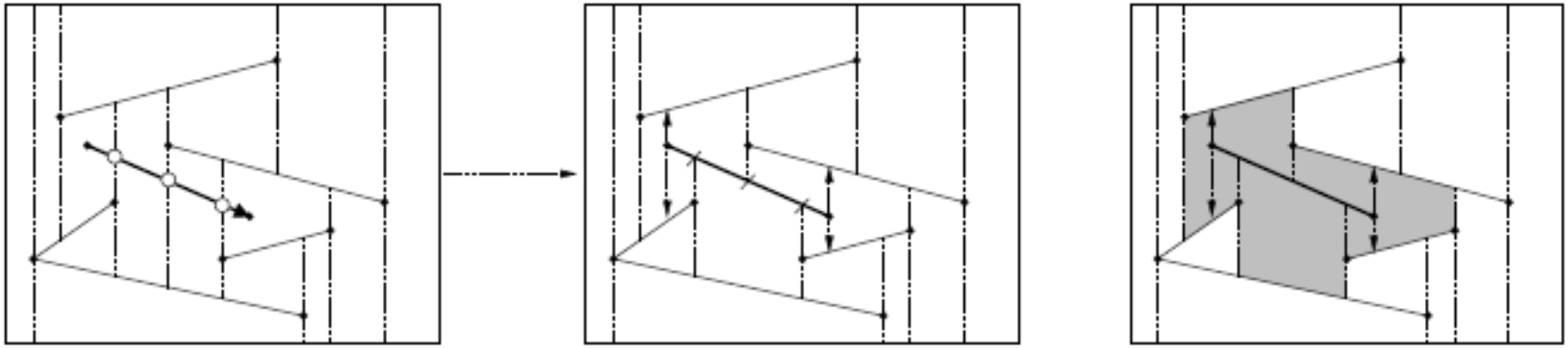


Trapezoidal Map

- An example of converting a complex shape into a disjoint collection of simpler objects.
- Since it is a refinement of original subdivision, once we know which face of the map a query point lies in, we will know which face of the original subdivision it lies in.
- We may construct the trapezoidal decomposition using “plane sweep”.
- But, we need “point location!” so use “randomized incremental!”

Randomized Incremental Algorithm

- Start with the initial bounding rectangle. Insert segments one by one in random order, updating the trapezoidal map.
- Locate the left endpoint of the segment and trace the line segment determining which trapezoids it intersects and fix these trapezoids.
- Shoot bullet rays from the endpoints of the segment.
- Trim back earlier bullet paths hit by the new segment



Locate left endpoint and determine intersections

Shoot new bullet paths and trim intersecting rays

Newly created trapezoids

Analysis

- Observe that the structure of the trapezoidal map does not depend on the order in which the segments are added.
- Claim : Ignoring the time spent to locate the left endpoint of a segment, the time to insert the ith segment and update the trapezoidal map is $O(k_i)$, where k_i is the number of newly created trapezoids.

Analysis

- Claim : Ignoring the time spent to locate the left endpoint of a segment, the time to insert the ith segment and update the trapezoidal map is $O(k_i)$, where k_i is the number of newly created trapezoids.
- Pf) let k be the number of bullet paths that this segment intersects. Need to shoot 4 bullets and trim each of k bullet paths, so total 4+k operations needed.
- If new segment crosses no bullet path, 4 new trapezoids.
- For each of the k bullet paths we cross, we add one more new trapezoid.
- Thus, k_i =k+4 update operations needed and each update takes O(1) time, so $O(k_i)$ time needed.

Analysis (1/4)

- Lemma : Consider the randomized incremental construction of a trapezoidal map and let k_i be the number of newly created trapezoids when ith segment is added. Then $E[k_i]=O(1)$, where the expectation is taken over all permutations of the segments.
- Pf) “Backwards analysis” : analyze expected value assuming the last insertion was random.
- Let T_i denote the trapezoidal map after insertion of ith segment.
- Among i segments in T_i , each one has an equal probability 1/i of being the last one added.
- For each segment s, we want to count the number of trapezoids that would have been created, had s been the last segment to be added.

Analysis (2/4)

- We say that a trapezoid Δ "depends" on s, if s would have created Δ, had s been added last.

- Let $\delta(\Delta,s) = 1$ if Δ depends on s, 0 otherwise.

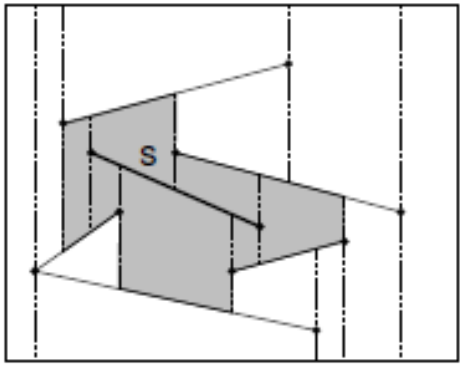
$$E(k_i) = \frac{1}{i}\sum_{s\in S_i}(\text{no. of trapezoids that depend on } s) = \frac{1}{i}\sum_{s\in S_i}\sum_{\Delta\in\mathcal{T}_i}\delta(\Delta,s).$$

Analysis (3/4)

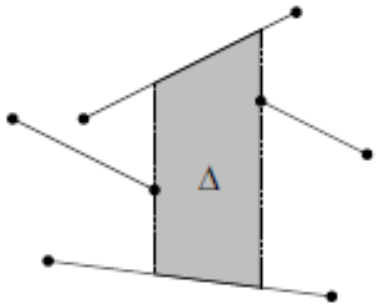
- Rather than counting the number of trapezoids that depend on each segment, count number of segments that each trapezoid depend on.

$$E(k_i) = \frac{1}{i} \sum_{s \in S_i} (\text{no. of trapezoids that depend on } s) = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s).$$

$$E(k_i) = \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} \sum_{s \in S_i} \delta(\Delta, s)$$



The trapezoids that depend on s



The segments that the trapezoi
Δ depends on.

Analysis (4/4)

- Each trapezoid is dependent on at most 4 segments.

$$E(k_i) \leq \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} 4 = \frac{1}{i} 4|\mathcal{T}_i| = \frac{1}{i} 4O(i) = O(1)$$

- Thus, total expected number of trapezoids created in the entire process is O(n).

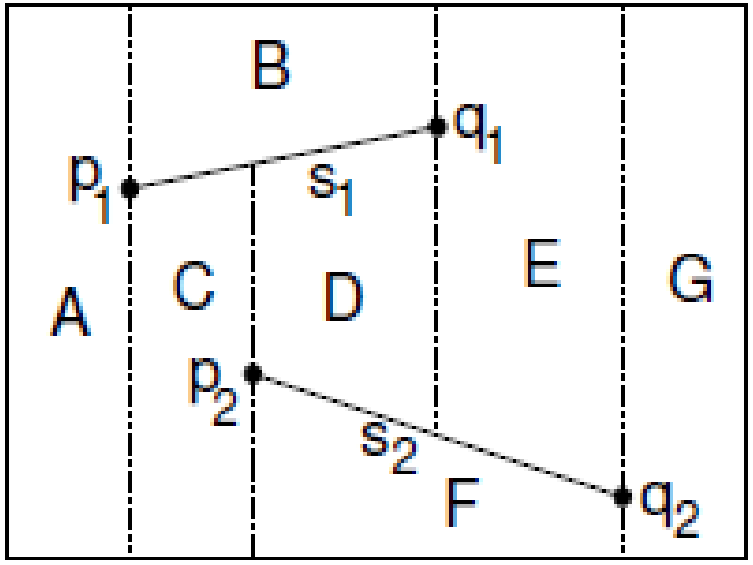
Summary

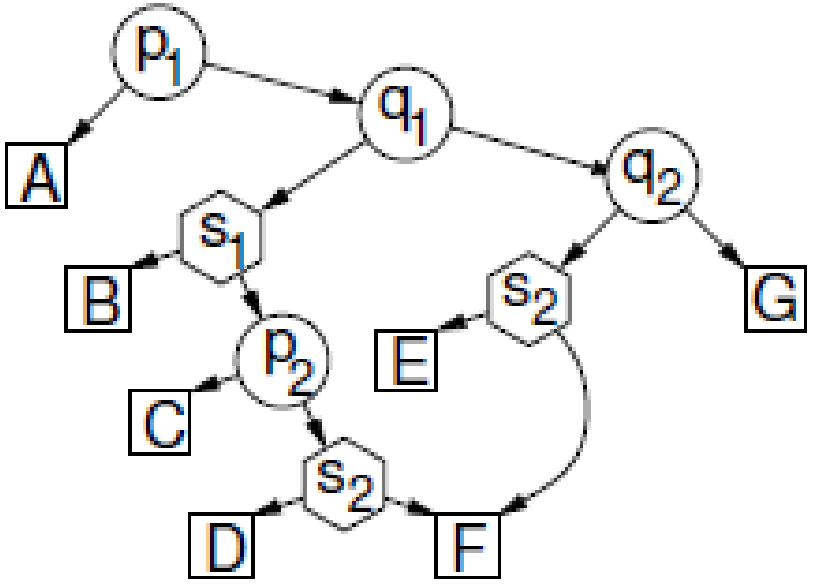
- For each insertion of segment, time to update the trapezoidal map = O(number of newly created trapezoids) = O(1) expected.
- How about point location? Need to locate the left endpoint of the new segment first : O(log n) expected time.
- Total running time : Σ O(1 + log n) = O(n log n) expected.
- How to locate the trapezoid containing the left endpoint of new segment?

Point location data structure

- A rooted directed acyclic graph
- Each node has either 2 or 0 outgoing edges.
- One leaf for every trapezoid.
- There are two types of internal nodes
  - x node : labeled with an endpoint of segment
    - Left child : points lying to the left of x node
    - Right child : points lying to the right of x node
  - y node : labeled by segment itself
    - Left child : points above the segment
    - Right child : points below the segment

Point location data structure





Point location data structure

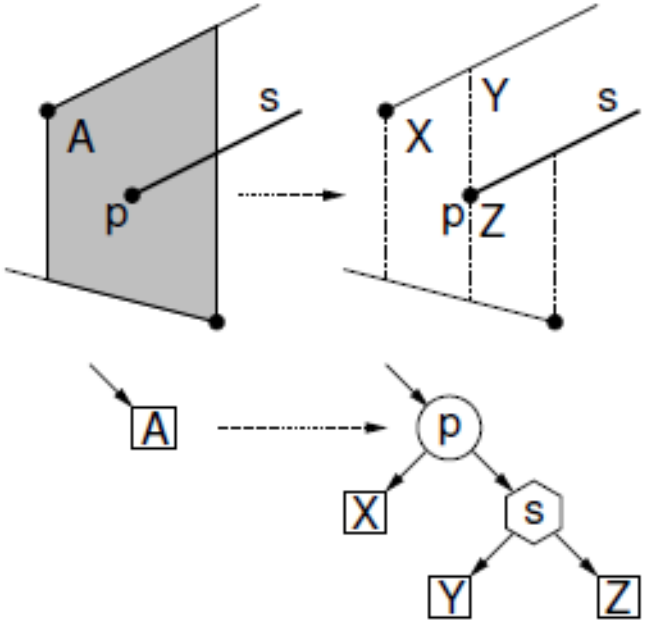
- The query point starts from the root and proceeds along the directed path to one of the leaves.
- This leaf corresponds to the trapezoid containing q.
- At x node, test is made if q lie right or left or that node.
- At y node, we check if q is above or below that segment.

Incremental construction

- The search structure depends on the order in which the segments are added.
- When we add a new segment s, we replace the leaves for deleted trapezoid with a small search structure, which locates the new trapezoid that contains the query point.
- 3 cases depending on how many endpoints of s lie within the current trapezoid.
  - Single (left or right) endpoint
  - No segment endpoint
  - 2 segment endpoints

Incremental construction

- Single (left or right) endpoint

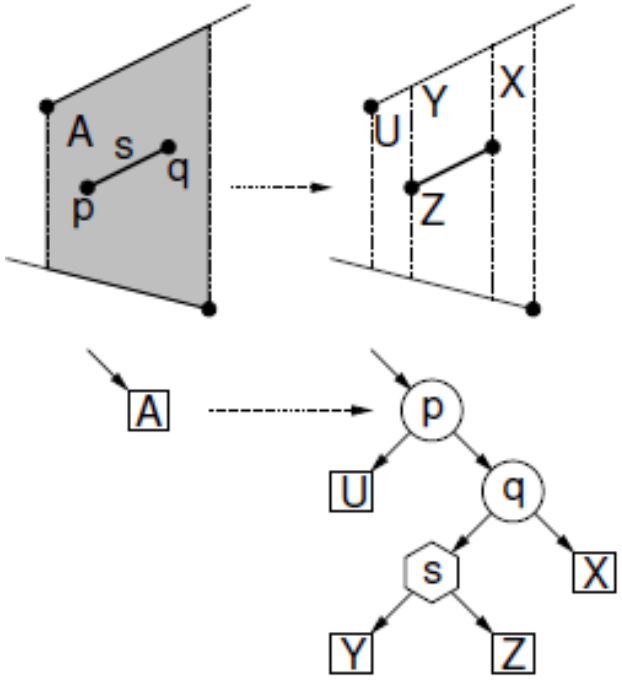


Incremental construction

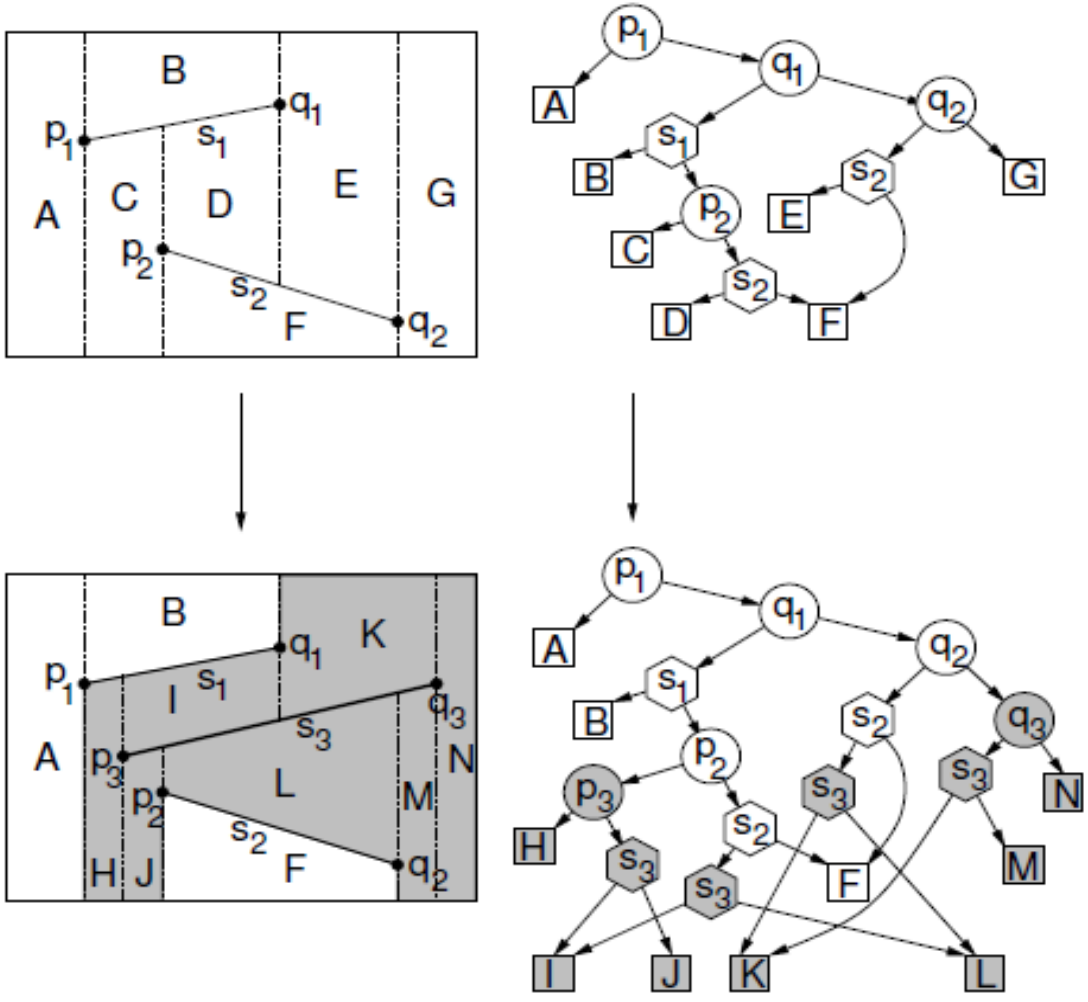
- No segment endpoint : when the segment cuts completely through a trapezoid. A single trapezoid is replaced by 2 trapezoids, one above and one below the segment.

Incremental construction

- Two segment endpoints



Incremental construction



Analysis (1/3)

- Claim : The expected size of the point location data structure is O(n) and the expected query time is O(log n).
- Pf) size : number of new nodes with each insertion is proportional to the number of newly created trapezoids, which is O(1) expected, implying total O(n) expected size.
- We'll show that for a fixed query point, the expected search path length is O(log n).
- Let q denote the query point. We'll consider how q moves incrementally through the structure with addition of each new line segment.

Analysis (2/3)

- Let Δ_i denote the trapezoid of the map that q lies in after the insertion of i segments.
- If $\Delta_{i-1} = \Delta_i$, the insertion of the ith segment did not affect the trapezoid that q was in, so q will stay where it is.
- Otherwise (If $\Delta_{i-1} \neq \Delta_i$), the insertion of the ith segment caused q's trapezoid to be deleted.
- So, q must locate itself w.r.t. the newly created trapezoids that overlap Δ_{i-1} .
- Since there are at most 4 such trapezoids, O(1) work needed to locate q.
- q may fall at most 3 levels in the search graph (the worst case occurs in the 2-endpoints case when q falls into one of the trapezoids lying above or below the segment).

Analysis (3/3)

- Since q may fall at most 3 levels with each change of its containing trapezoid, the expected search path length is at most 3 x sum of the probabilities that q changes its trapezoid at each insertion.
- Let p_i denote this probability (taken over random insertion orders, irrespective of the choice of q).
- Let D(q) denote the average depth of q in the final search graph.
- $D(q) \leq 3 \Sigma p_i \leq 3 \Sigma 4/i \leq 12 \ln n = O(\log n)$
- To show $p_i \leq 4/i$, use “backwards analysis”.
- Consider the trapezoid that contains q after ith insertion.
- The trapezoid depends on at most 4 segments.
- Thus, the probability that the last insertion caused q to belong to a new trapezoid is at most 4/i.

Guarantees on search time

- We showed that the expected search time is O(log n).
- No guarantee that the worst search time is O(log n).
- Sec. 6.4 proves that the probability that the maximum search path exceeds $3\lambda \ln(n+1)$ is at most $2/(n+1)^{\lambda \ln 1.25 - 3}$
- Using this fact, we can construct a search structure with O(log n) worst-case query time and O(n) worst-case space.
- Run the algorithm and keep track of the depth. As soon as the depth exceeds c log n for some suitably chosen c, then stop and start over again with new random order.
- The above lemma implies that such failure occurs rarely, so after constant number of trials, we'll succeed in constructing data structure of O(log n) depth.
- Similar argument applies to space.