CSE520: Computational Geometry
Lecture 11
Point Location

Antoine Vigneron

Ulsan National University of Science and Technology

June 15, 2020

Outline

- Today, we study a geometric data structure problem: Point location.
- We will solve it using trapezoidal maps and a randomized incremental construction (RIC).

Reference:

- Textbook Chapter 6.
- Dave Mount's lecture notes, lectures 14 and 15.

Problem Statement

Problem (Point location in a planar subdivision)

Preprocess a planar straight-line graph $\mathcal{G}$ so as to be able to answer the following queries efficiently:

- *Input: A query point q.*
- *Output: The face of $\mathcal{G}$ that contains q.*

Remarks:

One dimension	Two dimensions
Sorting	Trapezoidal map
Searching	Point location
Quicksort	RIC
Random BST	History graph

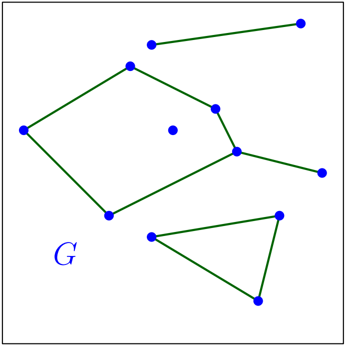
Results

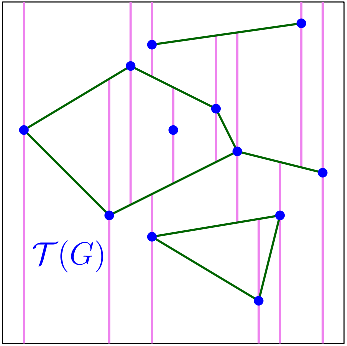
- Expected query time: $O(\log n)$
- Expected preprocessing time: $O(n \log n)$
- Expected space usage: $O(n)$

- Can we hope to do better?
  - There is a deterministic algorithm with same time and space bounds.
  - I may present it later this semester.

Trapezoidal Map

- Start with a PSLG $\mathcal{G}$.
  - General position assumption: No two vertices have same x-coordinate.
- The trapezoidal map $\mathcal{T}(\mathcal{G})$ is obtained by drawing vertical edges downward and upward from each vertex.



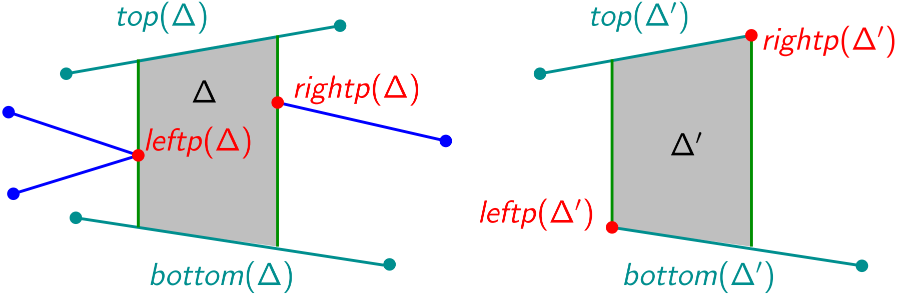


- We draw a bounding box around $\mathcal{G}$ so that there is no infinite face, hence each faces of $\mathcal{T}(\mathcal{G})$ is a trapezoid.

Trapezoids

Each trapezoid Δ of $\mathcal{T}(\mathcal{G})$ is determined by:

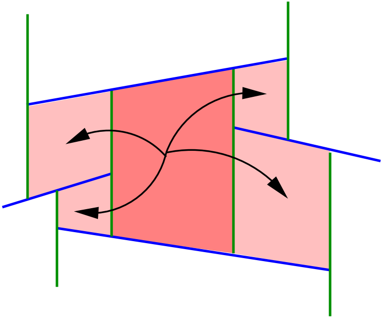
- A bottom segment $\mathit{bottom}(\Delta)$.
- A top segment $\mathit{top}(\Delta)$.
- A left vertex $\mathit{leftp}(\Delta)$.
- A right vertex $\mathit{rightp}(\Delta)$.



Neighbors

Definition (neighbors)

We say that two trapezoids are neighbors if they share a vertical edge.



- By our general position assumption, a trapezoid has at most 4 neighbors.

Data Structure

- Doubly Connected Edge List,
- or simpler: Just store adjacency relations from previous two slides.
  - The vertices of $\mathcal{G}$ are represented by their coordinates.
  - The edges of $\mathcal{G}$ point to their left and right endpoints.
  - Each trapezoid Δ of $\mathcal{T}(\mathcal{G})$ has pointers to:
    - *bottom*(Δ), *top*(Δ).
    - *leftp*(Δ), *rightp*(Δ).
    - Its (at most) 4 neighbors.

Trapezoidal Map

The trapezoidal map $\mathcal{T}(\mathcal{G})$ has:

- $O(n)$ vertices,
- $O(n)$ edges,
- $O(n)$ faces (=trapezoids).

Point location in a trapezoidal map:

- Construct the trapezoidal map by RIC.
- Use the history graph to perform point location.

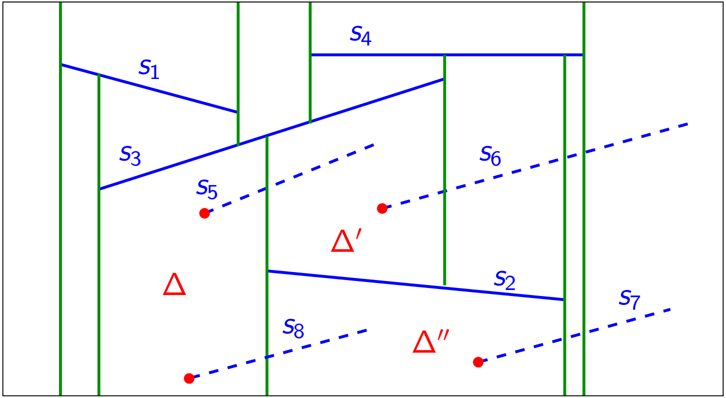
Point Location in a PSLG $\mathcal{G}$

- First compute $\mathcal{T}(\mathcal{G})$ and associated search structure by RIC.
- Then augment the search structure with pointers from each face of $\mathcal{T}(\mathcal{G})$ to the face in $\mathcal{G}$ that contains it.
- Perform point location in $\mathcal{T}(\mathcal{G})$.
- Find the corresponding face in $\mathcal{G}$.

Randomized Incremental Construction of $\mathcal{T}(G)$

- Let S denote the set of edges of $\mathcal{G}$.
- Compute a random permutation $(s_1, s_2, \dots s_n)$ of S.
- We denote $S_i = \{s_1, s_2 \dots s_i\}$ for any $i \leqslant n$.
- Initialize the data structure: DCEL or adjacency relations for the bounding box.
- Initialize a conflict list for the left endpoints of the segments of S.
  - Initially, only one conflict list (for the interior face of the bounding box) that gathers all the segments in S.
  - Keep a pointer from each $s \in S$ to its conflicting trapezoid, which is the interior face of the bounding box.

Conflict Lists: Example



- $\mathcal{L}(\Delta) = \{s_5, s_8\}$
- $\mathcal{L}(\Delta') = \{s_6\}$
- $\mathcal{L}(\Delta'') = \{s_7\}$

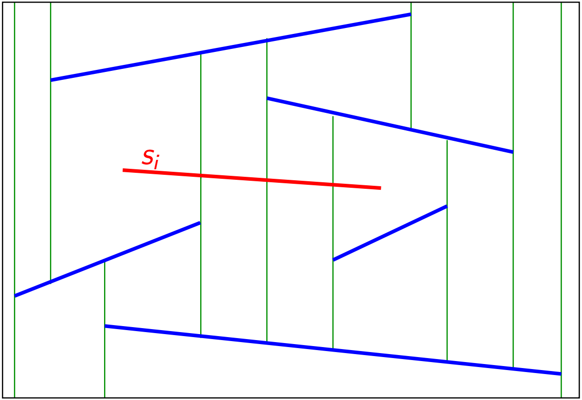
Idea

At step i we maintain:

- A representation of $\mathcal{T}(S_i)$.
- For each trapezoid Δ of $\mathcal{T}(S_i)$:
  - a conflict list $\mathcal{L}(\Delta)$ of pointers to all the segments in $S \setminus S_i$ whose left endpoint is in Δ.
- For each $s \in S \setminus S_i$:
  - a pointer to the trapezoid Δ of $\mathcal{T}(S_i)$ that contains its left endpoint.

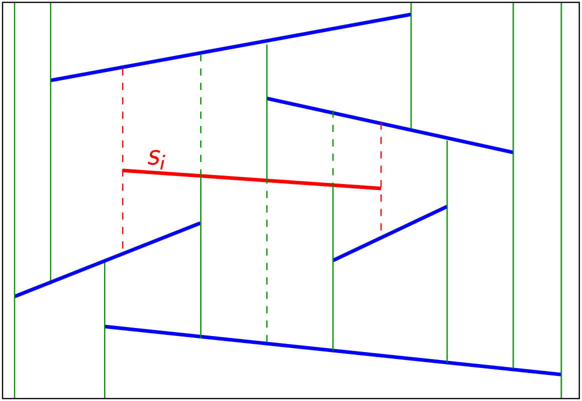
Then we insert s_{i+1} and update this data structure.

Inserting s_i



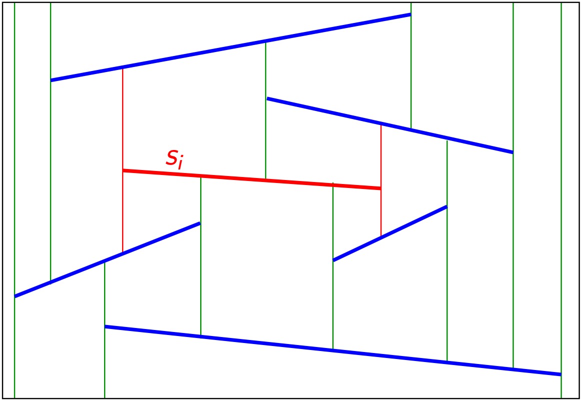
s_i may cross several trapezoids of $\mathcal{T}(S_{i-1})$.

Inserting s_i



Each trapezoid is split into at most 4 trapezoids.

Inserting s_i

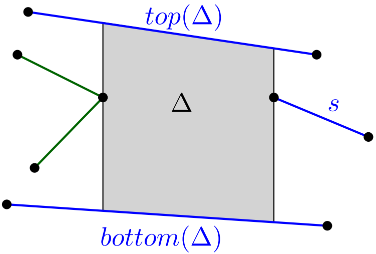


Some trapezoids are merged.

Terminology

Definition

We say that a trapezoid Δ of $\mathcal{T}(S_i)$ is *defined by* a segment $s \in S_i$ if Δ does not appear in $\mathcal{T}(S_i \setminus \{s\})$.



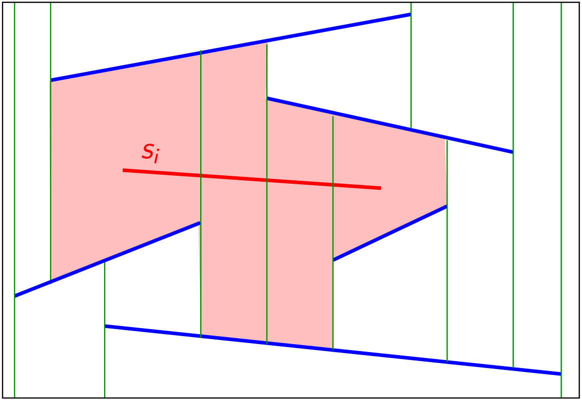
Here, the trapezoid Δ is defined only by the three segments $top(\Delta)$, $bottom(\Delta)$ and s.

Observation

A trapezoid is defined by at most 4 segments.

Zone of s_i

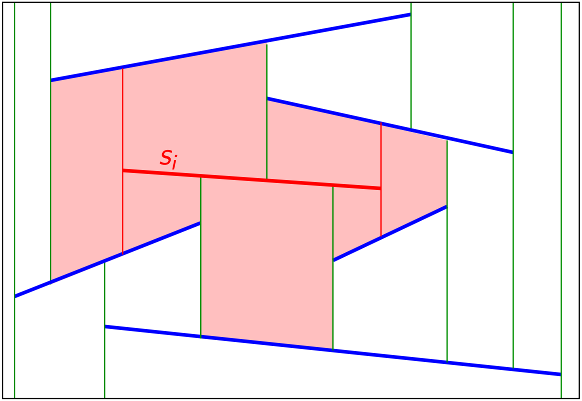
The *zone* of s_i in $\mathcal{T}(S_{i-1})$ is the union of all the cells that intersect s_i.



It is the union of all the trapezoids of $\mathcal{T}(S_{i-1})$ that are destroyed when we insert s_i.

Zone of s_i

It is also the union of all the trapezoids in $\mathcal{T}(S_i)$ that are defined by s_i.



It is the union of all the trapezoids created when we insert s_i.

Updating the Trapezoidal Map

- From our data structure, we know which trapezoid in $\mathcal{T}(S_{i-1})$ contains the left endpoint of s_i.
- We sweep a vertical line from left to right and update the trapezoidal map.
- Everything is done within the zone of s_i.
- Only two trapezoids intersect the sweep line at any time.

- Let k_i be the number of trapezoids in $\mathcal{T}(S_i)$ that are defined by s_i.
- There are at most k_i events.
- So the update can be done in $O(k_i)$ time.

Updating the Conflict Information

- We also need to update the conflict lists.
- Non-inserted left endpoints move from destroyed trapezoids to newly created trapezoids.
- Each destroyed trapezoid is contained in the union of at most 4 new trapezoids.
- So updates can be done in time $O(X_i)$ where X_i is the number of left endpoints of non-inserted segments in the zone of s_i.
- X_i is also the number of left endpoints in the trapezoids of $\mathcal{T}(S_i)$ that are defined by s_i, plus 1.

Analysis: Bound on $E[k_i]$

- Trapezoid $\Delta \in \mathcal{T}(S_i)$ is newly created iff it is defined by s_i.
- For each segment $s \in S_i$ and for each trapezoid $\Delta \in \mathcal{T}(S_i)$, let
  - $\delta(\Delta, s) = 1$ if s defines Δ.
  - $\delta(\Delta, s) = 0$ otherwise.
- The number of trapezoids defined by s is

$$\sum_{\Delta \in \mathcal{T}(S_i)} \delta(\Delta, s).$$

Analysis: Bound on $E[k_i]$

- We use backward analysis.
- We assume that S_i is fixed.
- Then s_i can be any segment in S_i with probability $1/i$.
- Then

$$E[k_i] = \frac{1}{i} \sum_{s \in S_i} \left(\sum_{\Delta \in \mathcal{T}(S_i)} \delta(\Delta, s) \right).$$

- We reverse the order of summation:

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

  - Note: This technique is called *double counting*.

Analysis: Bound on $E[k_i]$

- What is $\sum_{s \in S_i} \delta(\Delta, s)$?
- It is the number of segments that define Δ.
- So it is at most 4.
- Therefore

$$E[k_i] \leqslant \frac{1}{i} \sum_{\Delta \in \mathcal{T}(S_i)} 4.$$

- There are $O(i)$ trapezoids in $\mathcal{T}(S_i)$ so

$$E[k_i] = \frac{1}{i} O(i) = O(1).$$

Analysis: Bound on X_i

- We also need to bound the number X_i of non-inserted left endpoints in newly created trapezoids.
- Backward analysis: S_i is fixed.
- Let $s \in S \setminus S_i$.
- Let Δ be the trapezoid in $\mathcal{T}(S_i)$ that contains the left endpoint of s.
- What is the probability that Δ is newly created?
  - It is the probability that s_i is one of the (at most 4) segments that define Δ.
  - So it is at most $4/i$.
- So

$$E[X_i] \leqslant \frac{4(n-i)}{i}.$$

Analysis

- Let $T(n)$ be the construction time.
- Then by linearity of expectation

$$E[T(n)] = O\left(\sum_{i=1}^{n} E[k_i] + E[X_i]\right).$$

- So

$$E[T(n)] = O\left(\sum_{i=1}^{n} 1 + \frac{n}{i}\right) = O\left(n \sum_{i=1}^{n} \frac{1}{i}\right) = O(n \log n).$$

- It is an expected time on worst case input.

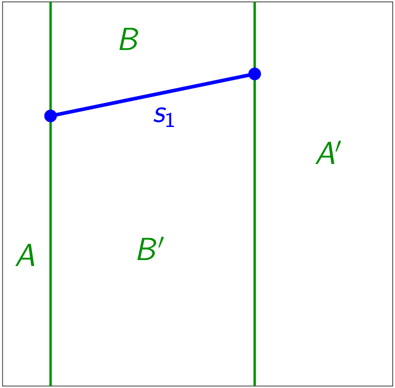
Point Location Data Structure

- Reference: D. Mount lecture 15.

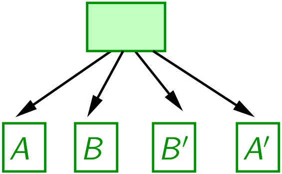
- The history graph records the history of the RIC.
- Lecture 10:
  - Quicksort $\Rightarrow$ history graph $\Rightarrow$ searching.
- Here:
  - RIC $\Rightarrow$ history graph $\Rightarrow$ point location.
- In Lecture 10, the history graph was a tree.
  - Here it is a *DAG*: Directed Acyclic Graph.

- Expected preprocessing time: $O(n \log n)$.
- Expected space usage: $O(n)$.
- Expected query time: $O(\log n)$.

Example (1)



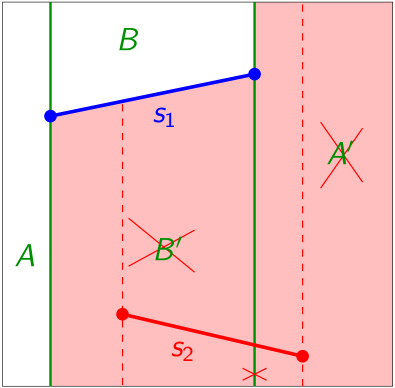
Trapezoidal map



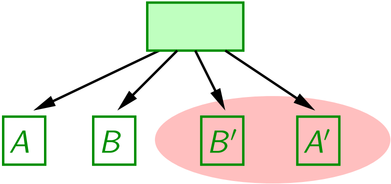
(The root corresponds to the bounding box)

History graph

Example (2)



Trapezoidal map



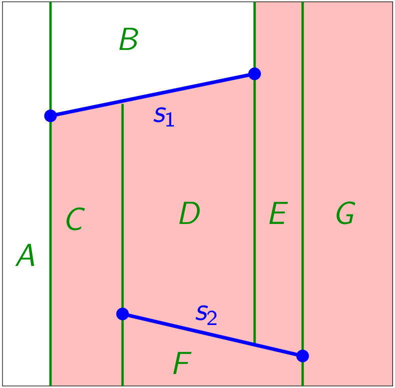
A' and B' are deleted from the trapezoidal map but remain in the history graph

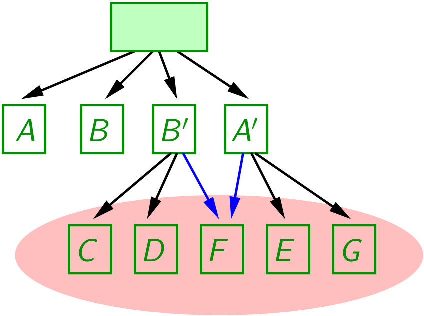
History graph

Update of the History Graph

- Connect each destroyed trapezoid to all the newly created trapezoids that overlap it.
- Overlap means the interiors intersect; not just touching along an edge.

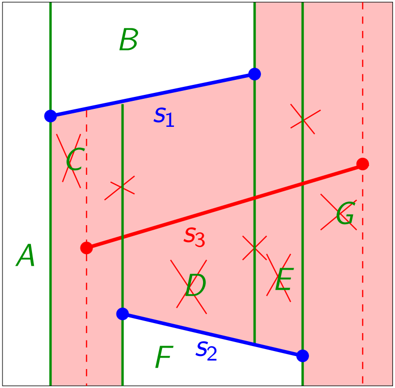
Example (3)

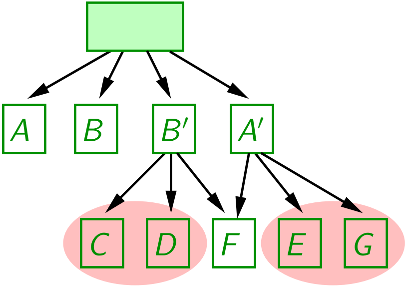




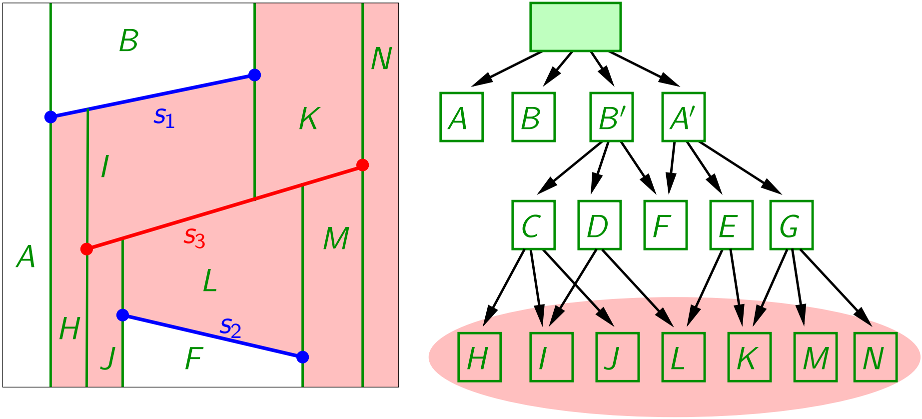
- The history graph is not a tree: Two edges point to F.
- It is a DAG.

Example (4)





Example (5)



Analysis

Update time of DAG:

- In time proportional to the number of new trapezoids.
  - That is, $O(k_i)$.
  - Proof: When we update the trapezoidal map by plane sweep, we create at most 3 DAG edges at each event. There are $O(k_i)$ events. (Slide 21.)
- We have seen that $E[k_i] = O(1)$.
- Hence, building the DAG takes $O(n)$ time overall.
- We also need $O(n \log n)$ time for the RIC of the trapezoidal map.

Answering Queries

- The query point q is given by its coordinates.
- If the current node is a leaf, then we are done.
- Otherwise, one of the descendants of the current trapezoid is a trapezoid that contains q.
- Since there are at most 4 descendants, we can find it in $O(1)$ time.
- Go down to this descendant and repeat the process.

Analysis

Let $Q(n)$ denote the query time, and let q denote the query point.

- At step i, let Δ_i be the trapezoid of $\mathcal{T}(S_i)$ that contains q.
- Each step where Δ_i changes, we go down in the DAG.
- So the length of the search path for q is proportional to the number of times Δ_i changes.
- $Q(n)$ is proportional to the length of the search path.

Analysis

- How many times does Δ_i change?
- We use backwards analysis: S_i is fixed.
- What is the probability that Δ_i is new?
- It is equal to the probability that s_i defines Δ_i.
- Δ_i is defined by $\leqslant 4$ segments.
- So this probability is $\leqslant 4/i$.
- Thus

$$E[Q(n)] = O\left(\sum_{i=1}^{n} \frac{4}{i}\right) = O(\log n).$$

Concluding Remarks

- Simple and efficient data structure for a difficult problem.
- Implementable and practical.
- But
  - Analysis is not easy.
  - Non-deterministic: Some insertion orders give
    - $\Theta(n^2)$ construction time.
    - $\Theta(n^2)$ space usage.
    - $\Theta(n)$ query time.
- The time bounds hold with high probability. (Not proved in CSE520.)
- In practice, like quicksort, outperforms known deterministic data structures.