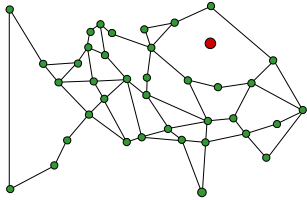


## Planar Point Location

### Planar Point Location



---

---

---

---

---

---

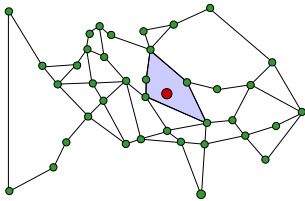
---

---

## Planar Point Location

**Input:** A planar subdivision  $S$  with  $n$  edges

**Aim:** Preprocess  $S$  such that point location queries can be answered efficiently.



Report the face of  $S$  that contains  $q$ .

---

---

---

---

---

---

---

---

## Planar Point Location

Any ideas?

---

---

---

---

---

---

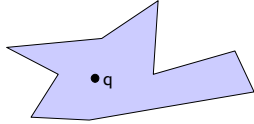
---

---

### 1<sup>st</sup> idea

**Idea:** For every face  $f$  within  $S$ , does  $q$  lie within  $f$ ?

**Time:** Each face can be checked in linear time  
 $\Rightarrow O(n)$  time and space in total



---

---

---

---

---

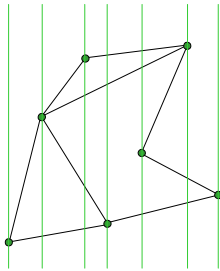
---

---

---

### 2<sup>nd</sup> idea

**Idea:** Draw a vertical line through every vertex (refinement).



This partitions the plane into vertical slabs

---

---

---

---

---

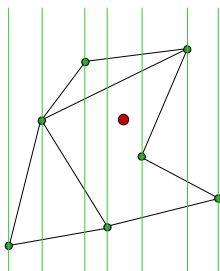
---

---

---

### 2<sup>nd</sup> idea

**Idea:** Draw a vertical line through every vertex (refinement).



Finding the correct vertical slab can be done by binary search  $O(\log n)$  time.

---

---

---

---

---

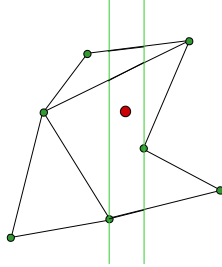
---

---

---

### 2<sup>nd</sup> idea

Idea: Draw a vertical line through every vertex (refinement).



Finding the correct vertical slab can be done by binary search  $O(\log n)$  time.

How do we answer a query within a slab?

Binary search!  $O(\log n)$  time.

(Since an edge in the slab completely crosses the slab we can use binary search.)

---

---

---

---

---

---

---

---

---

---

### 2<sup>nd</sup> idea

Query:

1. Search for the vertical slab that contains  $q$ . Binary search  $\Rightarrow O(\log n)$  time

2. Search for the trapezoid in the vertical slab.  $\Rightarrow O(\log n)$  time

Total time:  $O(\log n)$

---

---

---

---

---

---

---

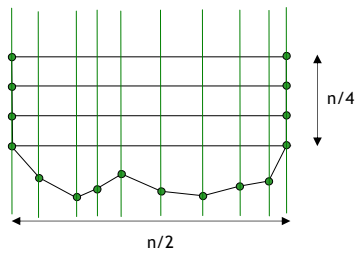
---

---

---

### 2<sup>nd</sup> idea

Storage:  $O(n)$  slabs,  $O(n)$  edges/slab  $\Rightarrow O(n^2)$  storage




---

---

---

---

---

---

---

---

---

---

### Summary

	Query time	Space
1 <sup>st</sup> idea	$O(n)$	$O(n)$
2 <sup>nd</sup> idea	$O(\log n)$	$O(n^2)$

---

---

---

---

---

---

---

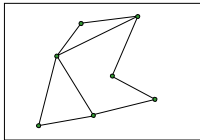
---

### 3<sup>rd</sup> idea

Can we make a good refinement of the subdivision that has small complexity?

Two simplifications:

1. Assume subdivision has a bounding box
2. No two vertices have the same x-coordinate.



---

---

---

---

---

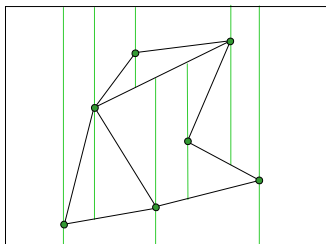
---

---

---

### Trapezoidal Decomposition

For every vertex of  $S$  draw two vertical extension, one extension going upward and one going downward. Result is called  $T(S)$ .



---

---

---

---

---

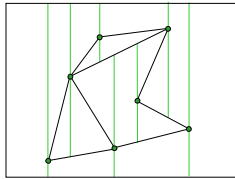
---

---

---

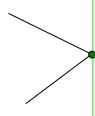
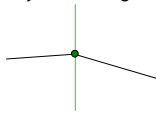
### Trapezoidal Decomposition

**Lemma:** Each face in the refinement  $T(S)$  has one or two vertical sides and exactly two non-vertical sides.



**Proof:**

- Every interior angle is at most  $180^\circ$ .



$\Rightarrow$  Every face is convex!

---

---

---

---

---

---

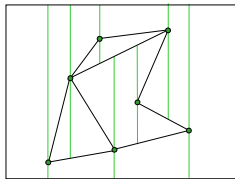
---

---

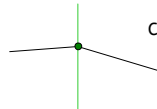
### Trapezoidal Decomposition

- Since every face is convex it can have at most two vertical sides.

- Proof for "exactly two non-vertical sides":



Assume the opposite - then there are two adjacent non-vertical sides between vertical extensions.



Contradiction!

---

---

---

---

---

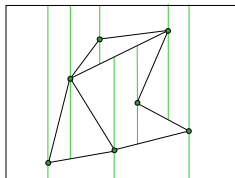
---

---

---

### Trapezoidal Decomposition

$\Rightarrow$  A face is either a trapezoid or a triangle.



**Complexity of the subdivision  $T(S)$**

- #vertices of  $T(S)$
- 1. vertex of bounding box 4
- 2. vertices of  $S$   $\leq 2n$
- 3. intersection between segments and extension  $\leq 2(2n)$

Total:  $\leq 6n+4$

---

---

---

---

---

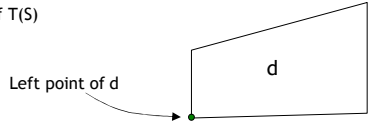
---

---

---

### Trapezoidal Decomposition

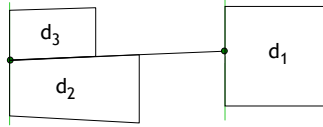
#faces of  $T(S)$



Consider a segment

$n$  segments  
"  $\leq 3$  faces/segment "

$\Rightarrow 3n+1$  faces




---

---

---

---

---

---

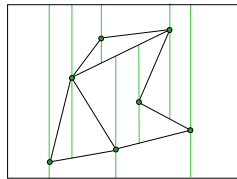
---

---

### Trapezoidal Decomposition

# faces  $\leq 3n+1$   
# vertices  $\leq 6n-4$   
# edges  $\leq ?$

Complexity of  $T(S)$  is  $O(n)$




---

---

---

---

---

---

---

---

### Compute a decomposition

**Aim:** Produce a trapezoidal decomposition  $T(S)$  and a data structure  $D$  that supports point location queries.

**Algorithm:** TrapezoidalMap( $S$ )

**Input:** set  $S$  of  $n$  non-crossing line segments

**Output:** trapezoidal map  $T$  and a search structure  $D$  for  $T$

1. Set  $T$  to be the bounding box of  $S$
2. Set  $D$  to be a leaf containing the bounding box
3. Compute a random order of the segments in  $S = \{s_1, \dots, s_n\}$
4. For  $i \leftarrow 1$  to  $n$  do
  - a. Find the set  $d_1, \dots, d_k$  of trapezoids in  $T$  intersected by  $s_i$
  - b. Remove  $d_1, \dots, d_k$  from  $T$  and replace by new trapezoids
  - c. Remove the leaves in  $D$  corresponding to  $d_1, \dots, d_k$  and create new leaves for the new trapezoids.

---

---

---

---

---

---

---

---

### Correctness proof

Prove that the following invariant is maintained:  
T is a trapezoidal map for  $S_i = \{s_1, \dots, s_i\}$  and  
D is a valid search structure for  $T(S_i)$ .

Base case:  $T(S_0)$   
D is a single leaf node



Induction hypothesis: Assume it is true for  $T(S_{i-1})$  and  $D(S_{i-1})$ .

Induction step: Invariant is true when edge  $s_i$  is about to be processed - prove that it is correct after  $s_i$  has been inserted.

---

---

---

---

---

---

---

---

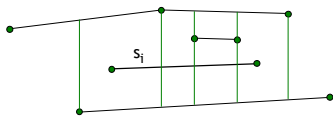
---

---

### Correctness proof

When  $s_i$  is inserted which regions are affected?

All trapezoids that are intersected by  $s_i$ .



Note that a trapezoid in  $T(S_{i-1})$  intersected by  $s_i$  is not a trapezoid in  $T(S_i)$ .

---

---

---

---

---

---

---

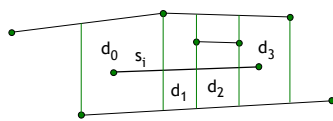
---

---

---

### Correctness proof

How do we find the trapezoids intersected by  $s_i$ ?



Use  $D(S_{i-1})!$

Let  $k$  be the number of intersected trapezoids.

---

---

---

---

---

---

---

---

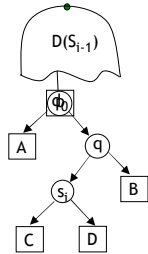
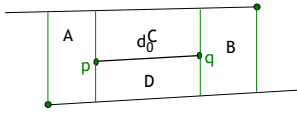
---

---

### Correctness proof

#### Update T and D

(a)  $k=1$



p and q are x-nodes in D and  $s_i$  is a y-node

---

---

---

---

---

---

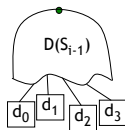
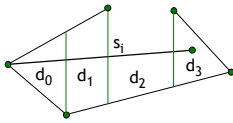
---

---

### Correctness proof

#### Update T and D

(b)  $k>1$




---

---

---

---

---

---

---

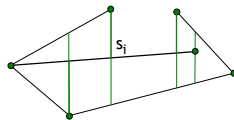
---

### Correctness proof

#### Update T

(b)  $k>1$

1. Add the extensions of the endpoints of  $s_i$ .
2. Shorten the vertical extension that intersect  $s_i$ .




---

---

---

---

---

---

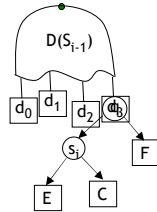
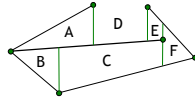
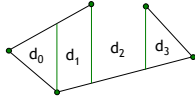
---

---

### Correctness proof

Update D

(b)  $k > 1$



1. If  $d_0$  (or  $d_{k-1}$ ) is intersected by an endpoint of  $s_i$  then replace  $d_0$  (or  $d_{k-1}$ ) with an x-node (endpoint) and a y-node (segment).

---

---

---

---

---

---

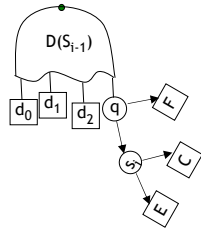
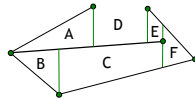
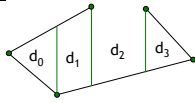
---

---

### Correctness proof

Update D

(b)  $k > 1$



2. The other leaves in  $d_0, \dots, d_k$  are replaced with y-nodes for  $s_i$ .

---

---

---

---

---

---

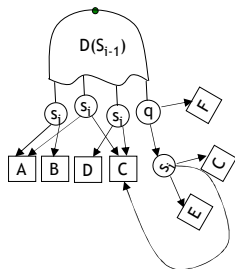
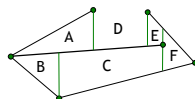
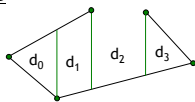
---

---

### Correctness proof

Update D

(b)  $k > 1$



2. The other leaves in  $d_0, \dots, d_k$  are replaced with y-nodes for  $s_i$ .

---

---

---

---

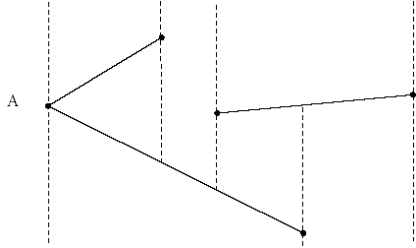
---

---

---

---

### Example



28

---

---

---

---

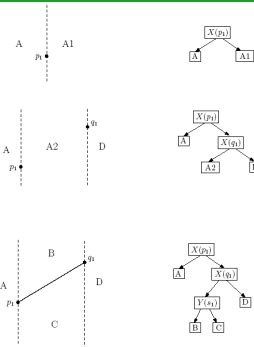
---

---

---

---

### Example



29

---

---

---

---

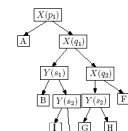
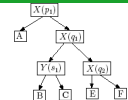
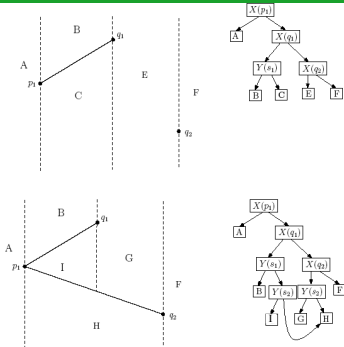
---

---

---

---

### Example



30

---

---

---

---

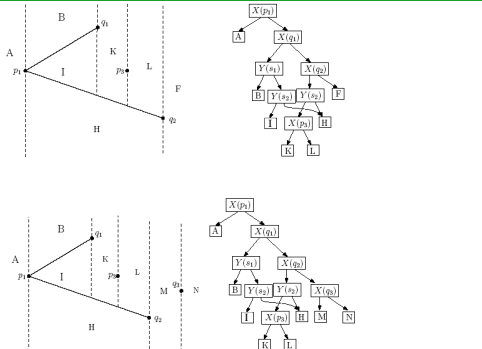
---

---

---

---

### Example




---

---

---

---

---

---

---

---

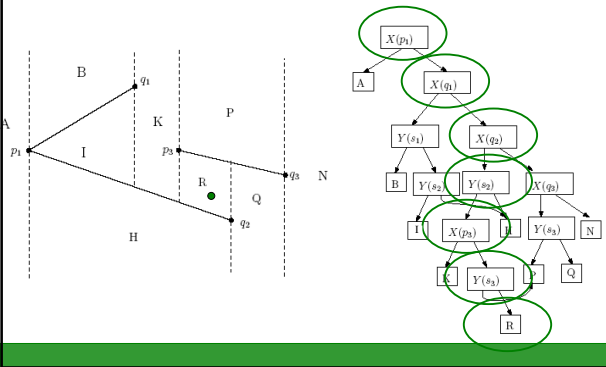
---

---

---

---

### Example




---

---

---

---

---

---

---

---

---

---

---

---

### Correctness proof

Correctness follows from the invariant!

- We proved that  $T$  was a trapezoidal decomposition for  $S_i$  when  $i=0$
- Assumed it was true for  $i=m-1$
- Constructed a valid trapezoidal decomposition  $T$  for  $S_m$ , using  $T$  for  $S_{m-1}$

The same holds for the query structure  $D$ .

---

---

---

---

---

---

---

---

---

---

---

---

### Complexity (worst case)

Height of D             $O(n)$   
 Query time            Height of D  
 Preprocessing time    $\sum_i (\text{height}(D_i) + k_i)$  [k is the number of new faces]  
                                $= O(n^2)$

---

---

---

---

---

---

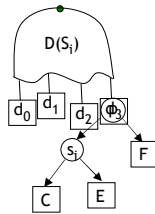
---

---

### Expected query complexity

What is the expected height?

Worst case: depth increases by 3 in each iteration.



Fix a query point q. What is the expected query time for q w.r.t. the n! possible insertions orders?

---

---

---

---

---

---

---

---

### Expected query complexity

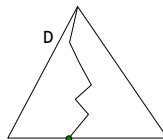
Consider the path traversed in D when querying with point q.

Every node on the path was created in some iteration of the algorithm.

$X_i$  - the number of nodes on that path created in iteration i.

Since S and q are fixed,  $X_i$  is a random variable (it only depends on the order of the segments).

$$\sum_i X_i$$




---

---

---

---

---

---

---

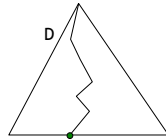
---

### Expected query complexity

We know:  $X_i \leq 3$

If  $P_i$  is the probability that a node has been added to the path in iteration  $i$  then

$$E[X_i] \leq 3 \cdot P_i \quad (*)$$




---

---

---

---

---

---

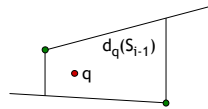
---

---

### Expected query complexity

How can we bound  $P_i$ ?

$d_q(S_{i-1})$  - the trapezoid containing the query point  $q$  in  $T(S_{i-1})$ .



**Observation:**

A node is added to the path iff  $d_q(S_i) \neq d_q(S_{i-1})$ .

in other words:  $P_i = \Pr[d_q(S_i) \neq d_q(S_{i-1})]$

---

---

---

---

---

---

---

---

### Expected query complexity

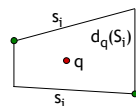
**Observation:** All trapezoids inserted in iteration  $i$  are adjacent to segment  $s_i$ .

$\Rightarrow s_i$  has to intersect  $d_q(S_{i-1})$ .

**Backward analysis!**

Consider  $T(S_i)$ . What is the probability that  $d_q(S_i)$  disappears when we remove  $s_i$ ?

$d_q(S_i)$  disappears if and only if  $d_q(S_i)$  has  $s_i$  as a top or bottom boundary, or one of  $s_i$ 's endpoints as left or right boundary.




---

---

---

---

---

---

---

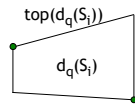
---

### Expected query complexity

What is the probability that  $\text{top}(d_q(S_i))$  disappears?

The segments have been inserted in random order so every segment is equally likely to be  $s_i$ .

$$\Rightarrow \Pr[s_i = \text{top}(d_q(S_i))] = 1/i$$



Same for bottom, left, and right side:

$$P_i = \Pr[d_q(S_i) \neq d_q(S_{i-1})] = \Pr[d_q(S_i) \notin T(S_{i-1})] \leq 4/i \quad (*)$$

---

---

---

---

---

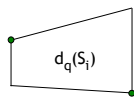
---

---

---

### Expected query complexity

Putting it together:



$$E[\sum_i x_i] = \sum_i E[x_i] \leq \sum_i 3 \cdot P_i \leq 12 \cdot \sum_i 1/i = 12 H_n$$

$$\text{where } H_n < \ln n + 1$$

Expected query time =  $O(\log n)$

---

---

---

---

---

---

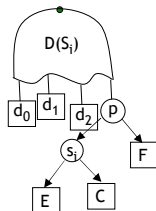
---

---

### Expected space complexity

What is the size of  $D$ ?

Bound the number of nodes. Note that every leaf corresponds to a trapezoid.



$$\#nodes = (3n+1) + \sum_i (\#internal nodes created in iteration i)$$

---

---

---

---

---

---

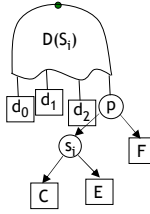
---

---

### Expected space complexity

$k_i$  = number of trapezoids created in step  $i$ .  
(#new leaves in  $D$ )

Expected size:  
 $O(n) + E[\sum_i (k_i - 1)] < O(n) + \sum_i E[k_i]$




---

---

---

---

---

---

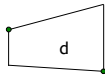
---

---

### Expected space complexity

Fix  $i$  where  $0 < i \leq n$ .

For a trapezoid  $d \in T(S_i)$  and a segment  $s \in S_i$



$$F(d,s) = \begin{cases} 1 & \text{if } d \text{ disappears from } T(S_i) \text{ when } s \text{ is removed} \\ 0 & \text{otherwise} \end{cases}$$

$k_i$  - the number of trapezoids that disappears when  $s_i$  is removed.

$$\sum_{s \in S_i} \sum_{d \in T(S_i)} F(d,s) = 4i$$

$$E[k_i] = 1/i \cdot \sum_{s \in S_i} \sum_{d \in T(S_i)} F(d,s) = O(i)/i = O(1)$$

---

---

---

---

---

---

---

---

### Expected complexity

Expected number of newly created trapezoids is  $O(1)$  in each iteration of the algorithm, thus  $O(n)$  space in total.

Expected construction time:

$$= \text{time to locate } d_0 + O(k_i)$$

$$= \sum_i ( E[\text{height}(T(S_i))] + O(E[k_i]) )$$

$$\leq \sum_i ( O(\log i + O(1)) ) = O(n \log n)$$

---

---

---

---

---

---

---

---

## Planar Point Location

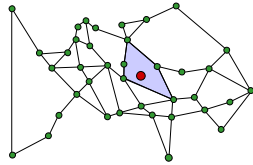
**Input:** A planar subdivision  $S$  with  $n$  edges

**Aim:** Preprocess  $S$  such that point location queries can be answered efficiently.

**Preprocessing:**  
 $O(n \log n)$  expected time

**Space:**  
 $O(n)$  expected size

**Query time:**  
 $O(\log n)$  expected time



---

---

---

---

---

---

---

---