Point Location

Read Chapter 6 of the textbook

1

Point Location

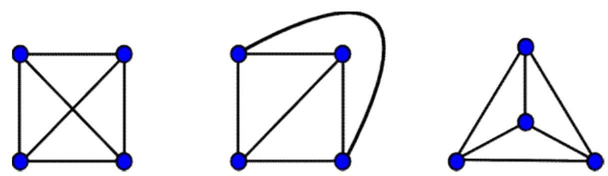
Given a straight-line planar subdivision S with n edges:

- Store S in a data structure $D(S)$
- Given a query point Q, use $D(S)$ to report the face(s) of S that contains Q

2

Planar Graphs

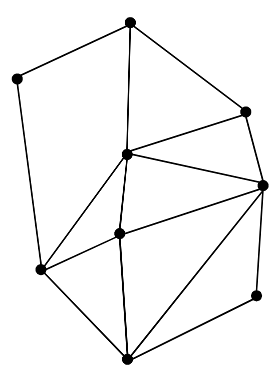
- A *planar graph* is a graph that can be drawn on the plane in such a way that its edges intersect only at their endpoints, i.e., it can be drawn in such a way that no edges cross each other



3

Planar Straight Line Graphs

- An embedding of a planar graph that uses only straight edges



4

Representation

- A planar straight line graph (*PSLG*) is a collection of vertices, edges, faces and a description of their incidences:
  - Each vertex is incident on all edges and faces that contain it
  - Each edge is incident on two faces and two vertices
  - Each face is incident on all edges and vertices that bound it
- Basic operation: list incidences of a vertex, of an edge, or of a face *in order*

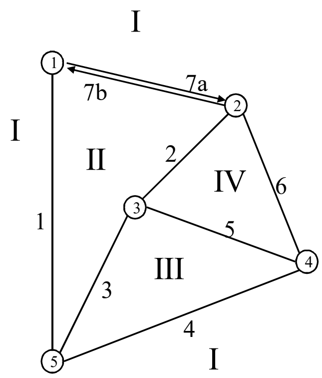
5

Doubly Connected Edge List

- Simple data structure for a (connected) PSLG
- Useful to view edges as directed:
  - each edge becomes two directed "half" edges
  - one incident face, one predecessor, one successor per edge
- Three collections of <u>fixed-size</u> records:
  - vertex (coordinates and *any* incident edge)
  - half edge (origin, predecessor, successor, twin)
  - face (*any* incident half-edge)
- Each record may store additional attributes
- Total size is $O(n)$, where n = # of vertices (prove!)

6

DCEL: An Example



ver	coord	edge
1	(x_1,y_1)	7a
2	(x_2,y_2)	6a
3	(x_3,y_3)	5a
4	(x_4,y_4)	5b
5	(x_5,y_5)	1b

face	edge
I	1b
II	1a
III	3a
IV	2a

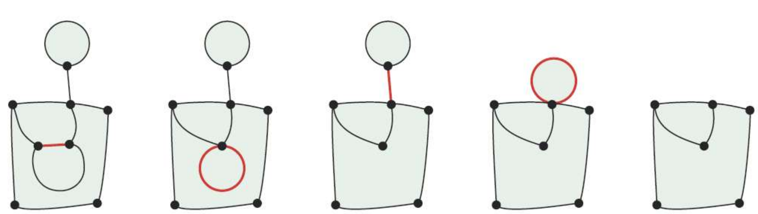
edge	origin	twin	face	next	prev
1a	1	1b	II	3b	7b
1b	5	1a	I	7a	4a
2a	2	2b	IV	5a	6b
2b	3	2a	II	7b	3b
3a	3	3b	III	4b	5b
3b	5	3a	II	2b	1a
4a	4	4b	I	1b	6a
4b	5	4a	III	5b	3a
5a	3	5b	IV	6b	2a
5b	4	5a	III	3a	4b
6a	2	6b	I	4a	7a
6b	4	6a	IV	2a	5a
7a	1	7b	I	6a	1b
7b	2	7a	II	1a	2b

7

Euler's Planar Graph Formula

Theorem (Euler's Formula). Let G be a connected planar (multi)graph with n vertices, e edges, and f faces (with unbounded outer face). Then $n - e + f = 2$.

Proof (by induction on $|E|$).

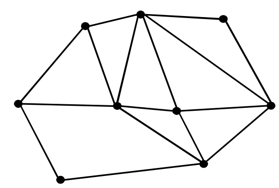


(8, 11, 5) (7, 10, 5) (7, 9, 4) (6, 8, 4) (6, 7, 3)

8

DCEL Space Analysis

- Let *D be the DCEL of a PSLG G* with *n* vertices, *e* edges and *f* faces



- Every face of *P* has at least 3 edges and each edge is incident on two faces $\Rightarrow$ $3f \le 2e$

$$n + f - 2 = e \geq 3f/2 \qquad e = n + f - 2$$

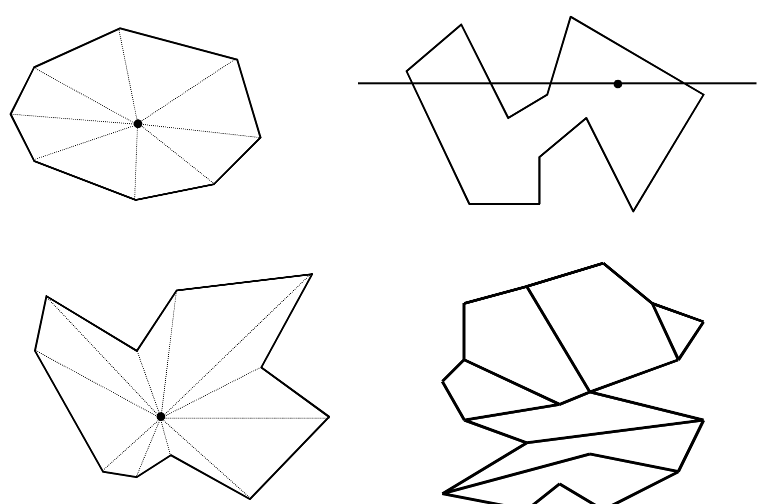
$$n \geq f/2 + 2 \qquad e \leq n + (2n-4) - 2$$

$$f \leq 2n - 4 \qquad e \leq 3n - 6$$

- Total size is $O(n)$, where n = # of vertices

9

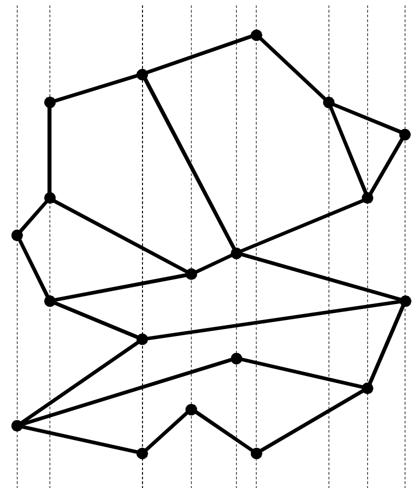
Types of Planar Subdivisions



10

Slab Method

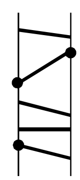
- Refine the subdivision S into a subdivision S' consisting of *slabs*. Each slab is an ordered sequence of trapezoids
- Perform two binary searches: one to locate the slab and one to locate the trapezoid containing Q



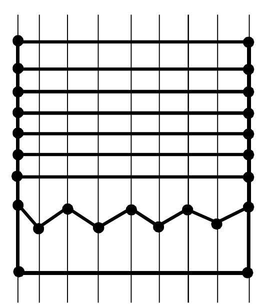
11

Complexity of Slab Method

- Time: $O(\log n)$



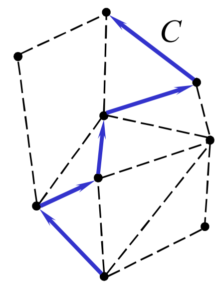
- Space: $O(n^2)$



12

Chain Method: Basics

- A *chain* C is a simple path in G that starts and ends at vertices of the unbounded region

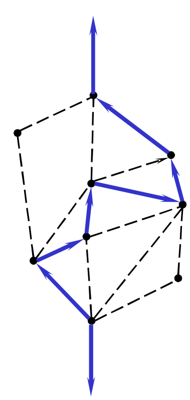


- Focus on chains that start (end) at the lowest (highest) vertex of G. Why?

13

Chain Basics...

- A chain partitions plane into left and right regions (with respect to *directed* edges



14

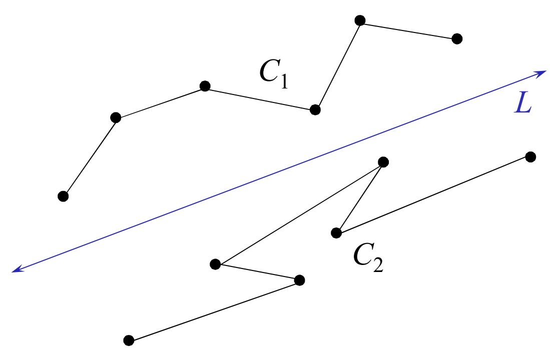
Chain Discrimination

- The *discrimination* of q against C determines the side of C that contains point q
- Would like to find a chain C such that:
  - C partitions G into sides of similar complexity
  - discrimination is easy
- Monotone chains will do!

15

Monotone Chains

- A chain C is *monotone* with respect to a line L if any line perpendicular to L intersects C at most once



16

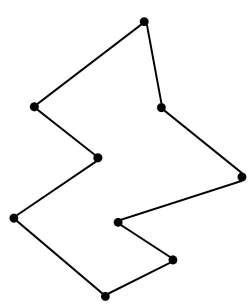
Monotone Chain Discrimination

- Let $C = \langle v_1, \ldots, v_k \rangle$ be monotone with respect to y-axis
  $\ell(p)$ denote the projection of p onto $L = y$-axis
- Note that $\ell(v_1) < \ell(v_2) < \ldots < \ell(v_k)$
- To discriminate q against C
  Binary search to find $\ell(v_i) < \ell(q) < \ell(v_{i+1})$
  Determine on which side of support($\overrightarrow{v_i v_{i+1}}$) q lies

17

Point location on Monotone Polygons

- In a monotone polygon the left (right) boundary is a monotone chain
- Can do point location in $O(\log n)$ time



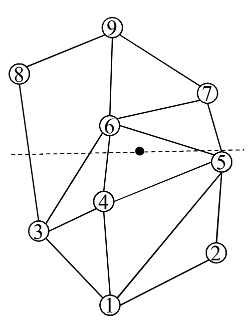
18

Complete Sets of Chains

- A set Δ of chains is *complete* with respect to G if:
  - All chains of Δ are monotone
  - Each edge of G belongs to at least one chain
  - For any two chains C_1 and C_2 of Δ all the vertices of C_1 that are not vertices of C_2 are on the same side of C_2.
- Chains of Δ are *ordered* !

19

Point location using Complete Sets



$$C_1 = \langle 1,3,8,9 \rangle$$
$$C_2 = \langle 1,3,6,9 \rangle$$
$$C_3 = \langle 1,3,4,6,9 \rangle$$
$$C_4 = \langle 1,4,5,6,9 \rangle$$
$$C_5 = \langle 1,5,6,7,9 \rangle$$
$$C_6 = \langle 1,2,5,7,9 \rangle$$

$$C_1 \prec C_2 \prec C_3 \prec C_4 \prec C_5 \prec C_6$$

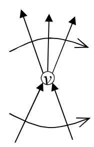
$$C_3 \prec q \prec C_4$$

- If Δ has r chains and longest chain has p vertices, can do point location in $O(\log r \log p)$ time

20

Chain Assignment

- Notation:



$\text{in}(v) =$ sorted list of incoming edges
$\text{out}(v) =$ sorted list of outgoing edges
$W(e) = \#$ of chains that contain e

$$W_{\text{in}}(v) = \sum_{e \in \text{in}(v)} W(e)$$

$$W_{\text{out}}(v) = \sum_{e \in \text{out}(v)} W(e)$$

- Find integer-valued $W(\cdot)$ such that:

$$W(e) > 0, \forall e$$
$$W_{\text{in}}(v_i) = W_{\text{out}}(v_i),\ 0 < i < n$$

21

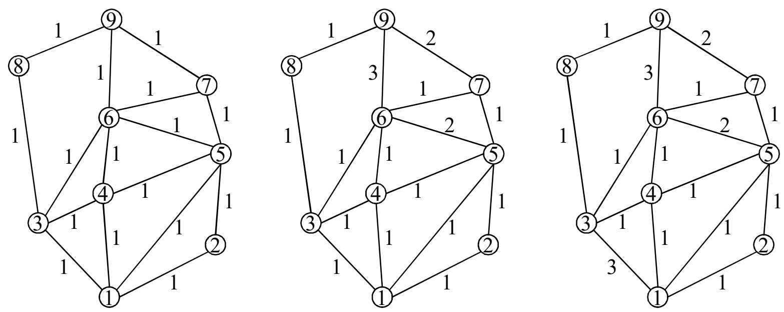
Computing # of Chains through an Edge

- Two pass algorithm:
  - sort the vertices by increasing y-coordinate
  1. make sure outgoing flow ≥ incoming flow
  2. make sure incoming flow ≥ outgoing flow

Set $W(e) \leftarrow 1, \forall e$
for $i \leftarrow 2$ to $n-1$ do
  if $W_{\text{in}}(v_i) > W_{\text{out}}(v_i)$ then
    $e \leftarrow$ first edge of $\text{out}(v_i)$
    $W(e) \leftarrow W_{\text{in}}(v_i) - W_{\text{out}}(v_i) + 1$
for $i \leftarrow n-1$ down to 2 do
  if $W_{\text{out}}(v_i) > W_{\text{in}}(v_i)$ then
    $e \leftarrow$ first edge of $\text{in}(v_i)$
    $W(e) \leftarrow W_{\text{out}}(v_i) - W_{\text{in}}(v_i) + W(e)$

22

Example



Before After Pass I After Pass II

$C_1 = \langle 1,3,8,9 \rangle \quad C_2 = \langle 1,3,6,9 \rangle$

$C_3 = \langle 1,3,4,6,9 \rangle \quad C_4 = \langle 1,4,5,6,9 \rangle$

$C_5 = \langle 1,5,6,7,9 \rangle \quad C_6 = \langle 1,2,5,7,9 \rangle$

23

Chain Assignment 1

Input. Weights $W(e), e \in E$

Output. A complete set of chains

$c \leftarrow 1$

for $i \leftarrow 1$ to $n-1$ do

  foreach e in out(v_i) do

    $\text{Min}(e) \leftarrow c$

    $\text{Max}(e) \leftarrow c + W(e) - 1$

    Add edge e to C_k, for each $\text{Min}(e) \le k \le \text{Max}(e)$

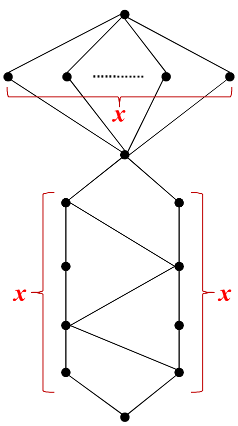
    $c \leftarrow \text{Max}(e) + 1$

  $d \leftarrow$ first edge of in(v_{i+1})

  $c \leftarrow \text{Min}(d)$

24

Worst Case Performance



$n = 3x + 3$ vertices

$x = (n-3)/3$ chains each of length $x+3$

Data structure requires $O(n^2)$ space

A query requires $O(\log^2 n)$ time

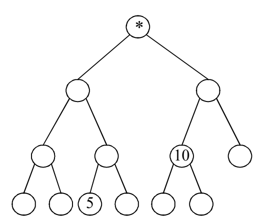
Problem: some edges belong to many chains

Can we find a way to store each edge a constant number of times?

25

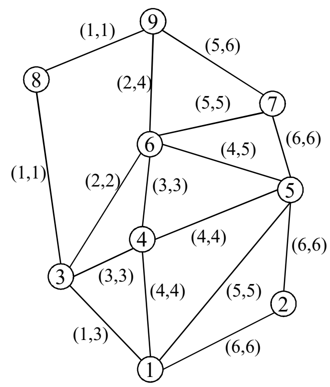
Chain Assignment

$c \leftarrow 1$
for $i \leftarrow 1$ to $n-1$ do
  foreach e in out(v_i) do
    $\text{Min}(e) \leftarrow c$
    $\text{Max}(e) \leftarrow c + W(e) - 1$
    $k \leftarrow \text{LCA}(\text{Min}(e), \text{Max}(e))$
    Assign edge e to C_k
    $c \leftarrow \text{Max}(e) + 1$
  $d \leftarrow$ first edge of in(v_{i+1})
  $c \leftarrow \text{Min}(d)$



26

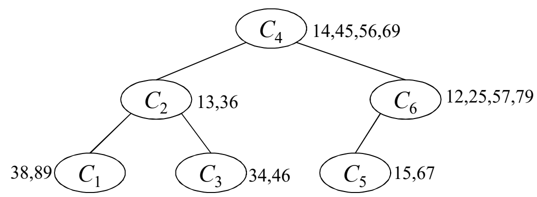
Example



$C_1 = \langle 1,3,8,9 \rangle \quad C_2 = \langle 1,3,6,9 \rangle$

$C_3 = \langle 1,3,4,6,9 \rangle \quad C_4 = \langle 1,4,5,6,9 \rangle$

$C_5 = \langle 1,5,6,7,9 \rangle \quad C_6 = \langle 1,2,5,7,9 \rangle$



27

Query Algorithm

Input: query point $q(x, y)$ and a complete chain set $C_1, \ldots, C_m$

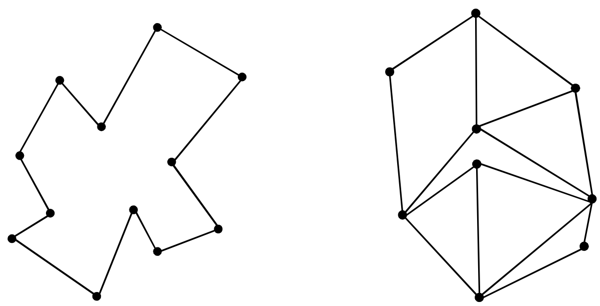
Output: region of G that contains q

Invariant: keep $l < r$ so at all times $C_l < q < C_r$

1) if $y > y_n$ or $y < y_1$ then return unbounded region
2) $l \leftarrow 0$; $r \leftarrow m+1$; $u \leftarrow$ root of T
3) $j \leftarrow \text{rank}(u)$
4) if $l \geq j$ then $\{u \leftarrow \text{right}(u)$; goto 3$\}$
5) if $r \leq j$ then $\{u \leftarrow \text{left}(u)$; goto 3$\}$
6) find $e = (v_i, v_{i+1})$ in C_j such that $y_i \leq y \leq y_{i+1}$
7) if q is to the right of e
   then $\{l \leftarrow \text{Max}(e)$; $f \leftarrow$ face to the right of $e\}$
   else $\{r \leftarrow \text{Min}(e)$; $f \leftarrow$ face to the left of $e\}$
8) if $r - l = 1$
   then return f
   else goto 4

28

- Does *every* planar straight line subdivision admit a complete set of chains?



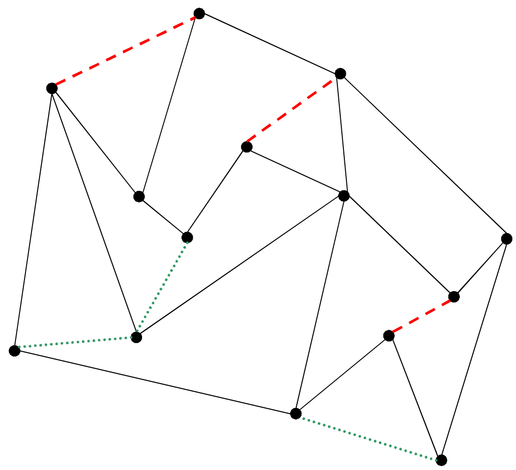
29

Regular Subdivisions

- Let $v_1,\ldots,v_n$ be the vertex list of G sorted in ascending order by y-coordinate
- Vertex v_j is *regular* if there are $i<j<k$ such that both v_iv_j and v_jv_k are edges of G
- G is regular if every vertex j, $1<j<n$, is regular
- G admits a complete set of chains iff G is regular
- A regular super-graph H of a non-regular graph G can be computed in $O(n \log n)$ time
  - the size of H is $O(n)$
  - the resulting PSLG is a refinement of G

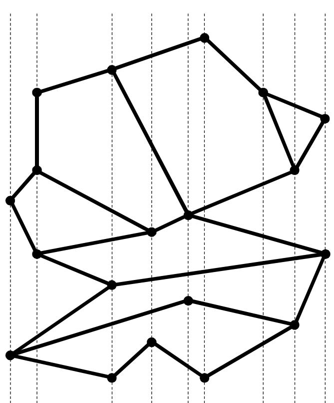
30

Making a PSLG Regular



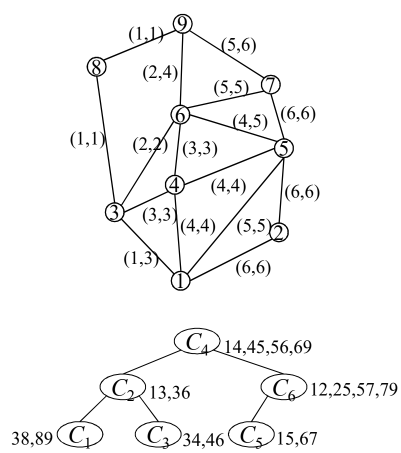
31

Slab Method



Performance: $(n^2, \log n)$

Chain Method

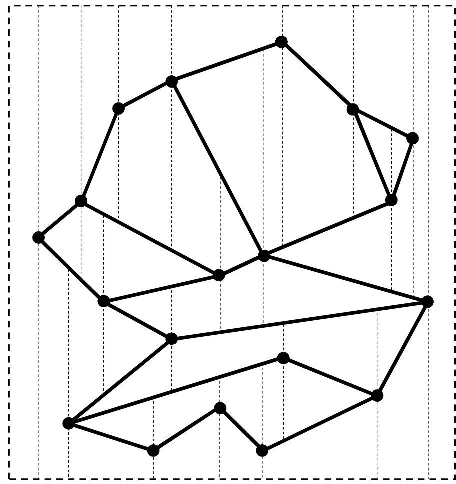


Performance: $(n, \log^2 n)$

32

Trapezoidal Map

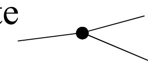
- Draw two *vertical extensions* or *walls* from every endpoint *p* and stop when they meet another segment of *S* or the boundary of an enclosing box *R*
- Every face of *T*(*S*) is a trapezoid with two vertical sides



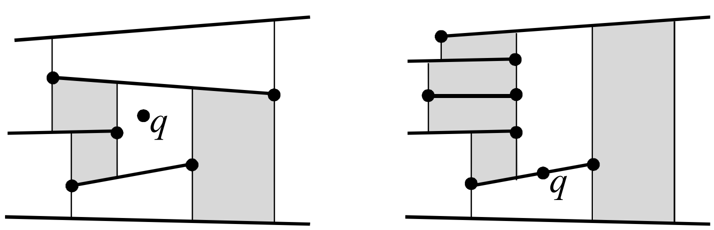
33

General Position

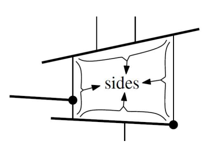
- No two endpoints have the same *x*-coordinate
  (but an endpoint may be incident on >1 segment)



- Query points fall strictly inside trapezoids



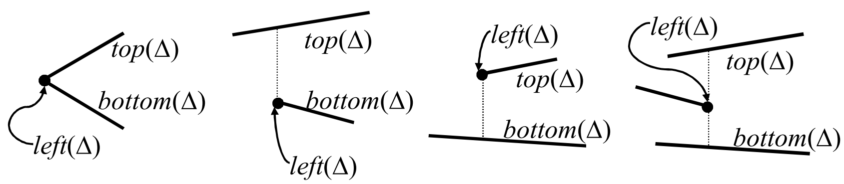
- Each face has one or two vertical sides and two non-vertical sides



34

Trapezoid Boundaries

- Each trapezoid Δ can be specified by two segments [*top*(Δ), *bottom*(Δ)], and two endpoints [*left*(Δ), *right*(Δ)]

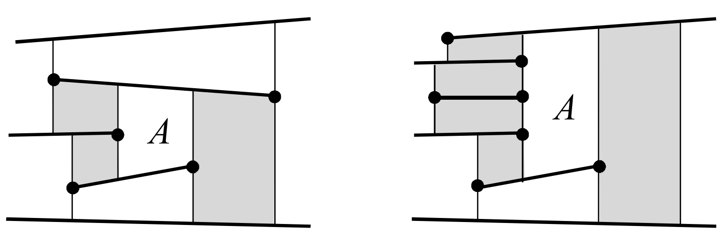


- Endpoints of extensions are never computed!

35

Adjacency

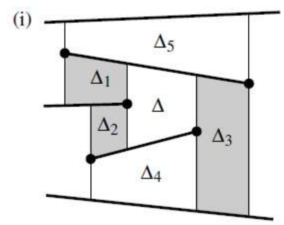
- Two trapezoids are *adjacent* if they share part of a vertical edge
- In general position, a trapezoid *A* has <u>at most</u> 4 adjacent trapezoids, called the *neighbors* of *A*



36

Trapezoid Neighbors

- Each trapezoid Δ has up to 4 neighbors: upper/lower left and upper/lower right



- If Δ and Δ′ are adjacent along the left edge of Δ then:
  Δ′ is the *upper left* neighbor of Δ if *top*(Δ) = *top*(Δ′),
  Δ′ is the *lower left* neighbor of Δ if *bottom*(Δ) = *bottom*(Δ′)

37

Trapezoidal Map Complexity

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

Theorem. The trapezoidal map $T(S)$ of a set S of n line segments in general position requires $O(n)$ storage.

38

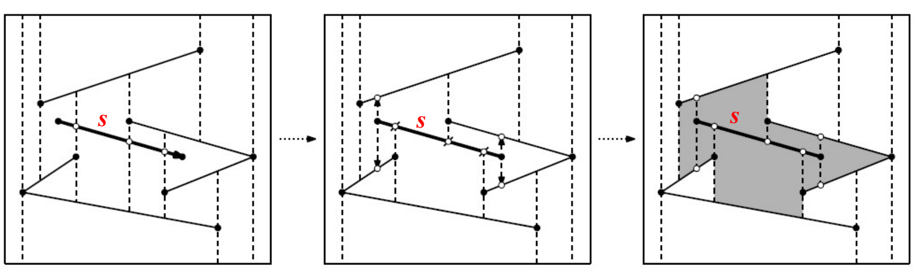
Trapezoidal Map Representation

- $T(S)$ is simpler than DCEL
- Consists of a collection of fixed-size records:
  - Input endpoint: 2 coordinates
  - Input segment: 2 pointers to endpoints
  - Trapezoid: 8 pointers (2 to segments top & bottom, 2 to endpoints left & right, 4 to neighboring trapezoids (some may be null))
- The geometry of a trapezoid is not stored explicitly!

39

Computing the Trapezoidal Map

- Will use a randomized incremental algorithm
  a) Locate left endpoint of new segment ***s*** and find intersections with vertical extensions
  b) Shoot vertical rays from ends of *s* and "trim" trapezoids
  c) Create new trapezoids

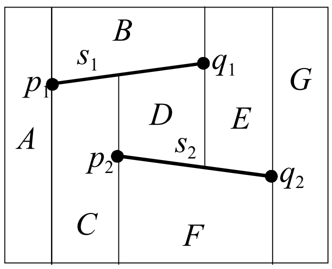


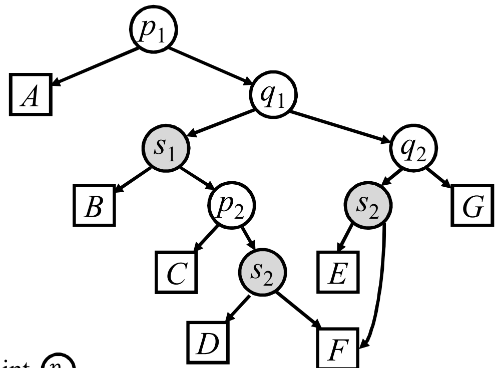
(a) (b) (c) 7 new trapezoids

40

A Data Structure for Point Location

- $D(S)$ consists of a trapezoidal map $T(S)$ cross-referenced with a *dag* search structure $G(S)$



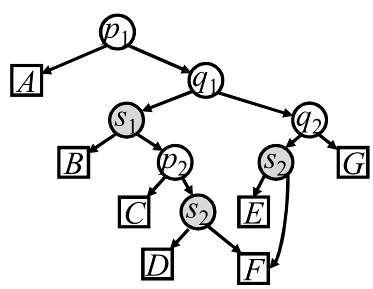


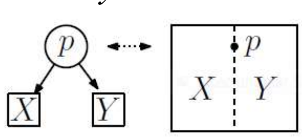
- Internal nodes of G:
  - x-nodes labeled with an endpoint p_i
  - y-nodes labeled with an segment s_k

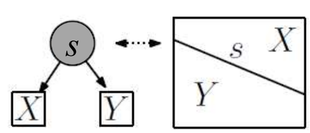
41

Search Structure $G(S)$

- Directed acyclic graph
- One source node
- One sink node for each trapezoid in $T(S)$
- Internal nodes have degree 2
  - Each x-node stores an endpoint p_i
  - Each y-node stores a segment s_k



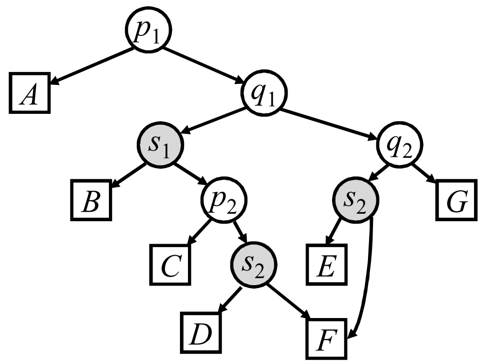




42

Search Structure $G(S)$…

- A query q proceeds from the root to a sink node that corresponds to the trapezoid that contains it



43

Constructing $G(S)$

- Incremental randomized algorithm
  - Start with $T(S_0) = G(S_0) = \{R\}$
  - Compute $T(S_i)$, $G(S_i)$ from $T(S_{i-1})$, $G(S_{i-1})$
  - Most orders result in good query time
- *Key insights*
  - A trapezoid Δ in $T(S_{i-1})$ is not in $T(S_i)$ iff s_i intersects Δ
  - If $\Delta_0,...,\Delta_k$ are the trapezoids intersected by s_i, from left to right, then Δ_j is a neighbor of Δ_{j-1}

44

Data Structure Construction

Input: A set S of non-crossing line segments

Output: Trapezoidal map $T(S)$ and search structure $G(S)$

1. Determine bounding box R and initialize map T and search structure G
2. Compute a random permutation $s_1,\ldots,s_n$ of S
3. **for** $i \leftarrow 1$ **to** n **do**

***Loop invariant*.** At i-th iteration, T is the map for S_{i-1} and G its search structure

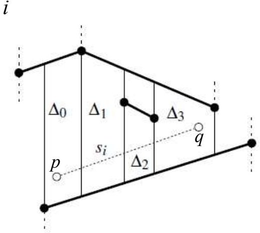
4. find the trapezoids $\Delta_0,\ldots,\Delta_k$ in T intersected by s_i
5. Replace $\Delta_0,\ldots,\Delta_k$ in T with new trapezoids due to s_i
6. Replace $\Delta_0,\ldots,\Delta_k$ in G with leaves for new trapezoids
   Add enough inner nodes to link to old inner nodes

45

Trapezoids Intersected by New Segment

Input: Trapezoidal map T and a new segment $s_i=[p,q]$

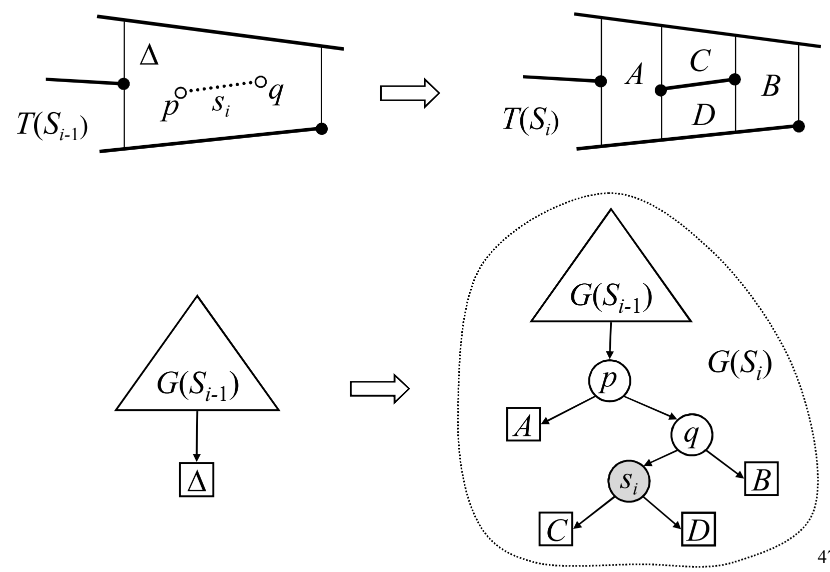
Output: Trapezoids $\Delta_0,\ldots,\Delta_k$ intersected by s_i



1. Search with p in G to find Δ_0
2. $j \leftarrow 0$
3. **while** q lies to the right of *right*(Δ_j) **do**
4.   **if** *right*(Δ_j) lies above s_i **then**
5. $\Delta_{j+1} \leftarrow$ lower right neighbor of Δ_j
6.   **else** $\Delta_{j+1} \leftarrow$ upper right neighbor of Δ_j
7. $j \leftarrow j + 1$
8. **return** $\Delta_0,\ldots,\Delta_j$

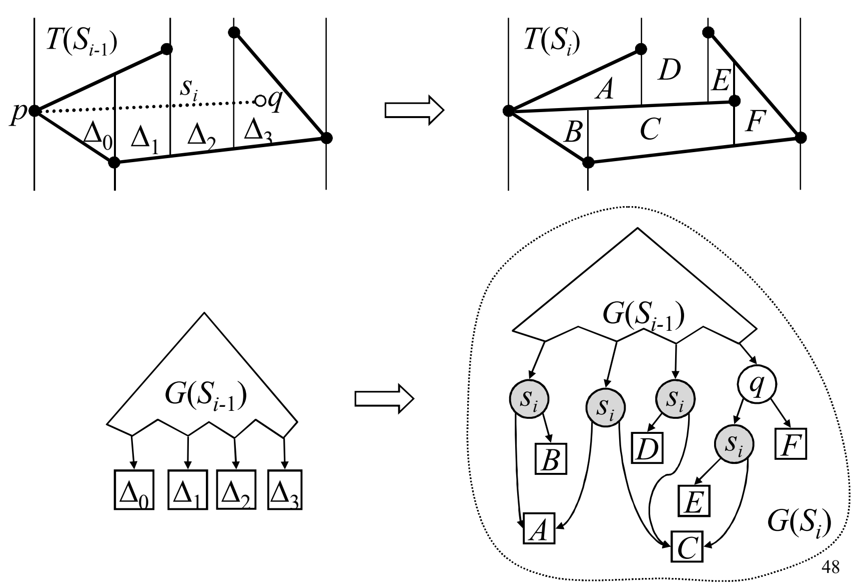
46

Updating T and G: k=0



47

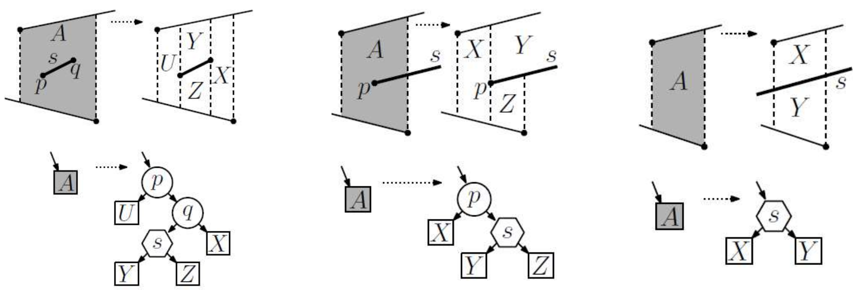
Updating T and G: k>0



48

Cases

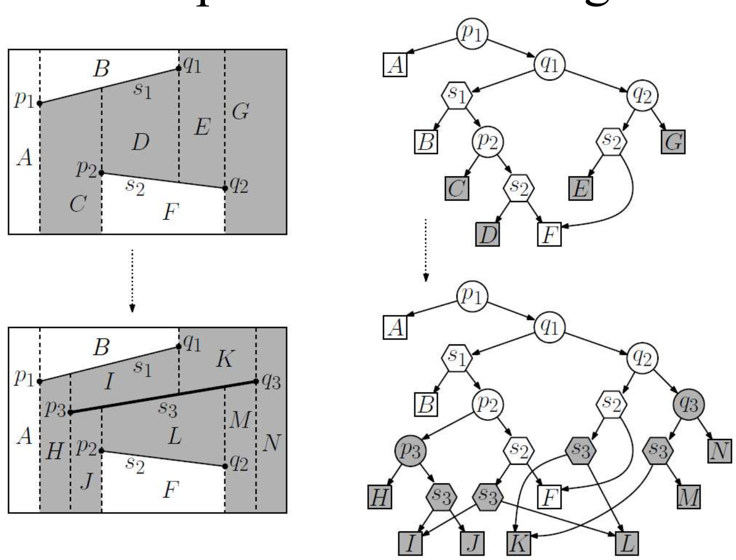
- There are 3 cases depending on the number h of endpoints of s_i inside a trapezoid Δ_j



$h = 2$ $h = 1$ $h = 0$

49

Example of Leaf Sharing

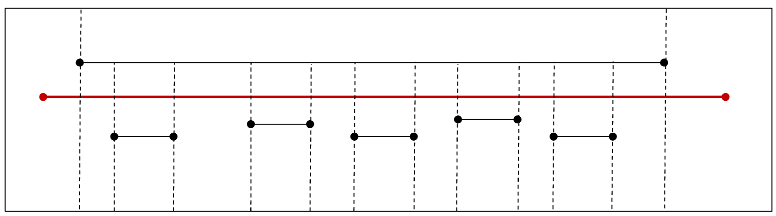


- Through sharing, each trapezoid appears once as leaf

50

Analysis

- Ignoring the time spent to locate the left endpoint of s_i, the time to insert s_i and update the trapezoidal map is $O(k_i)$, where k_i is the number of new trapezoids
- How many new trapezoids can be created from the insertion of a segment?
  - Clearly $E(k_i) = O(n)$
  - But, in fact, $E(k_i) = O(1)$



51

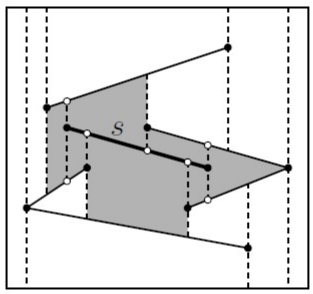
Analysis…

- Each segment of S_i has probability $1/i$ to have been inserted last
- We say that a trapezoid Δ of the current trapezoidal map *depends* on a segment *s*, if *s* would have caused Δ to be created, had *s* been inserted last
- We want to count the number of trapezoids that depend on each segment of S_i, and then compute the average over all segments.

52

Analysis…

- For trapezoid Δ and segment s of S_i we define $\delta(\Delta, s) = 1$ if Δ depends on s; and $\delta(\Delta, s) = 0$, otherwise.
- The number of trapezoids that depend on s is simply $\sum_{\Delta \in T_i} \delta(\Delta, s)$



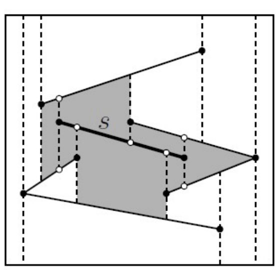
53

Analysis…

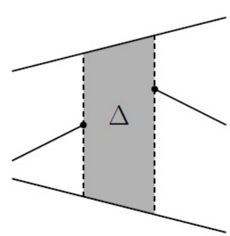
- The expected number of new trapezoids is

$$E(k_i) = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in T_i} \delta(\Delta, s) = \frac{1}{i} \sum_{\Delta \in T_i} \sum_{s \in S_i} \delta(\Delta, s) \leq \frac{1}{i} \sum_{\Delta \in T_i} 4$$

$$= \frac{4}{i} |T_i| = O(1)$$



Trapezoids that depend on *s*



Segments that Δ depend on

54

Space Analysis…

Size of $T(S)$ is $O(n)$, so enough to bound # nodes in G.

Size of G is $O(n)+\sum_{i=1}^{n}(2k_i-1)$.

$$E[k_i]=\frac{1}{i}\sum_{s\in S_i}\sum_{\Delta\in T(S_i)}\delta(\Delta,s)\le\frac{4\,|T(S_i)|}{i}=\frac{O(i)}{i}=O(1)$$

55

Query Time Analysis

$P_q(n)=$ path length in G for a fixed but arbitrary query q

$X_i=$ increase in path length for q when inserting s_i

$\Delta_q(S)=$ trapezoid containing q in $T(S)$

Iteration i contributes to $P_q(n)$ if $\Delta_q(S_i)\ne\Delta_q(S_{i-1})$

$P_i=\Pr(\Delta_q(S_i)\ne\Delta_q(S_{i-1}))$

Since $X_i\le 3$, $E[X_i]\le 3P_i$

We use backwards analysis to estimate P_i : consider $T(S_i)$,
find probability that $\Delta_q(S_i)$ "disappears" when removing s_i
A trapezoid disappears from S_i if one of left, right, top, or
bottom disappears from $S_i\Rightarrow P_i\le 4/i$

$$E[P_q(n)]=\sum_{i=1}^{n}E[X_i]\le 3\sum_{i=1}^{n}\frac{4}{i}=12\sum_{i=1}^{n}\frac{1}{i}=O(\log n)$$

56

Complexity

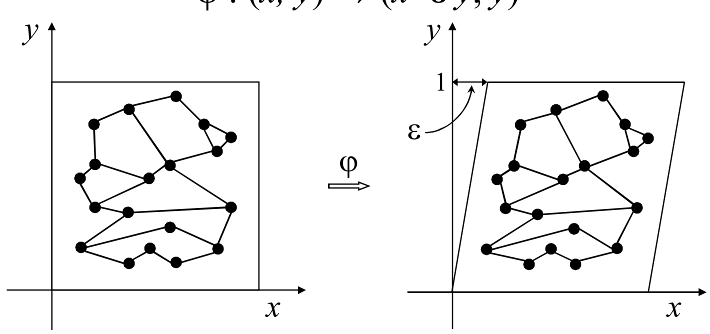
Theorem. The trapezoidal map and search structure can be computed in $O(n \log n)$ expected time. The expected size of the data structure is $O(n)$ and, for any query point q, the expected query time is $O(\log n)$.

57

Symbolic Perturbation

- We assumed endpoint x-coordinates are unique and queries reside in interior of trapezoids
- To fix this, shear input along x-axis by a small $\varepsilon > 0$:

$$\varphi : (x, y) \rightarrow (x+\varepsilon \cdot y, y)$$



- This imposes a strict horizontal order for a set of distinct points

58

Shear Properties

- $\varphi : (x, y) \rightarrow (x+\varepsilon \cdot y, y)$
- Eliminates duplicate x-coordinates (provided ε is small enough)
- Preserves order in x-direction of input points (provided ε is small enough)
- Trapezoidal map computed on φS, not on S
- Sheared query φq executed on sheared map $T(\varphi S)$
- Shear is not computed explicitly (φp stored as p)

59

Testing at x-Nodes

Given $\varphi q = (q_x + \varepsilon q_y, q_y)$ and $\varphi p = (p_x + \varepsilon p_y, p_y)$ compare φq with vertical line through φp

if $q_x < p_x$ **then return** left
else if $q_x > p_x$ **then return** right
else if $q_y < p_y$ **then return** left
else if $q_y > p_y$ **then return** right
else same point

Note : φq cannot lie on the vertical line through φp, unless $p = q$

60

Testing at y-Nodes

Given segment φs with endpoints $\varphi p_1 = (x_1 + \varepsilon y_1, y_1)$ and $\varphi p_2 = (x_2 + \varepsilon y_2, y_2)$ and $\varphi q = (x + \varepsilon \cdot y, y)$, compare φq with segment φs

Precondition : $x_1 + \varepsilon y_1 \le x + \varepsilon y \le x_2 + \varepsilon y_2$

if $x_1 = x_2$ **then** φq is on φs
else if q is below s **then** φq is below φs
else if q is above s **then** φq is above φs
else φq is on φs

61

A Tail Estimate

Lemma. Let S be a set of n non-crossing line segments and q a query point. For any $\lambda > 0$ the probability that the search path for q contain more than $3\lambda \ln(n+1)$ nodes is at most

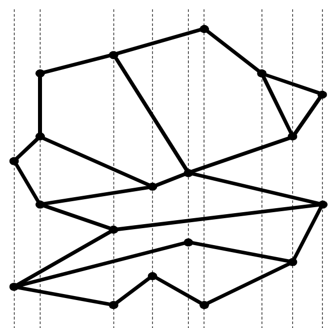
$$\frac{1}{(n+1)^{\lambda \ln(1.25)-1}}$$

Proof. Omitted (see textbook).

62

A Tail Estimate…

Lemma. Let S be a set of n non-crossing line segments. For any $\lambda > 0$ the probability that the depth of the search structure is more than $3\lambda \ln(n+1)$ is at most $\dfrac{2}{(n+1)^{\lambda \ln(1.25) - 3}}$



63

A Deterministic Result

Theorem. Let S be a planar subdivision with n edges. There exists a point location data structure $D(S)$ for S that uses $O(n)$ storage in the worst case and has $O(\log n)$ query time in the worst case

64