Computational Geometry Lecture
Point Location

INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

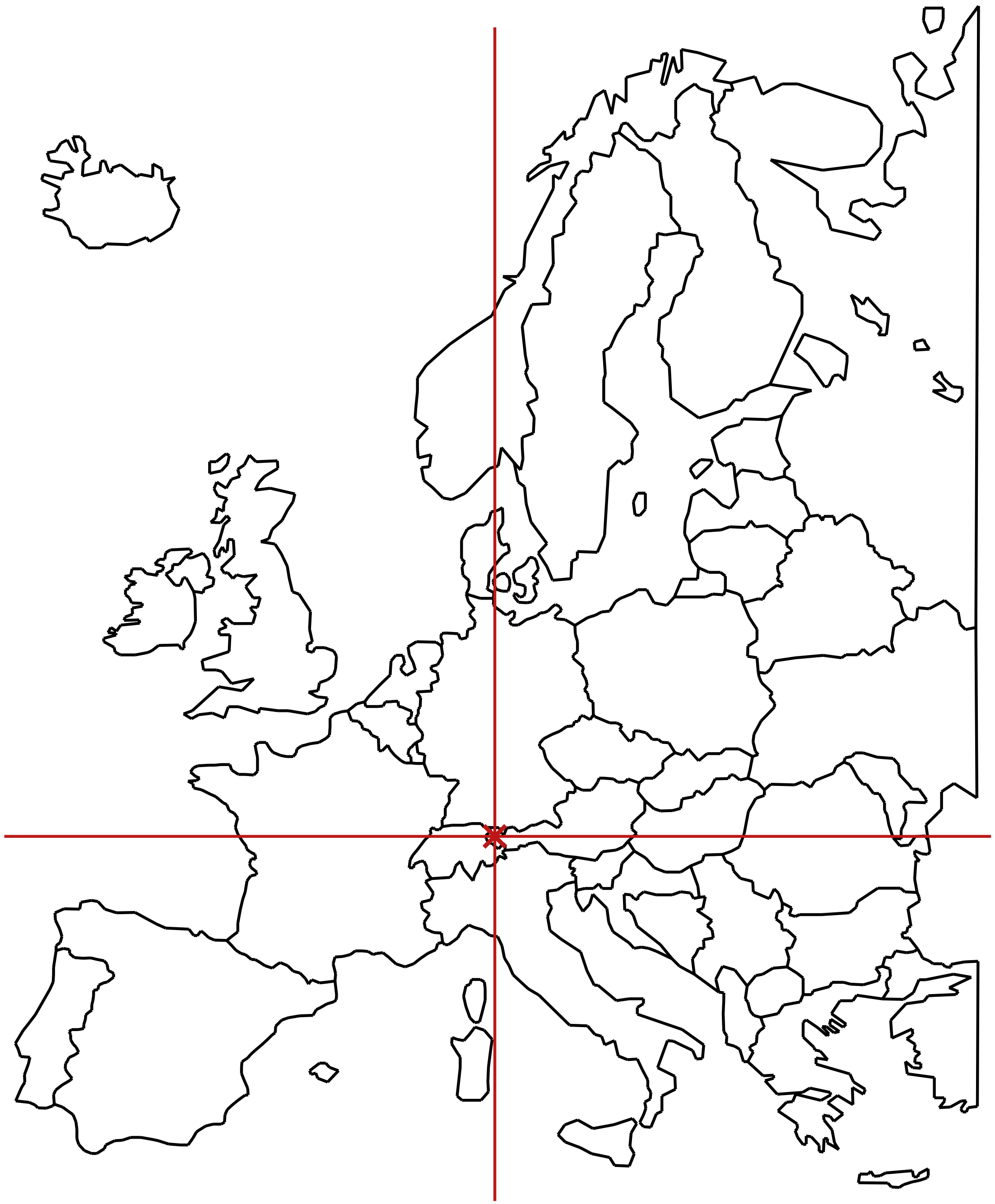
Chih-Hung Liu · Tamara Mchledidze

20.06.2018

Motivation

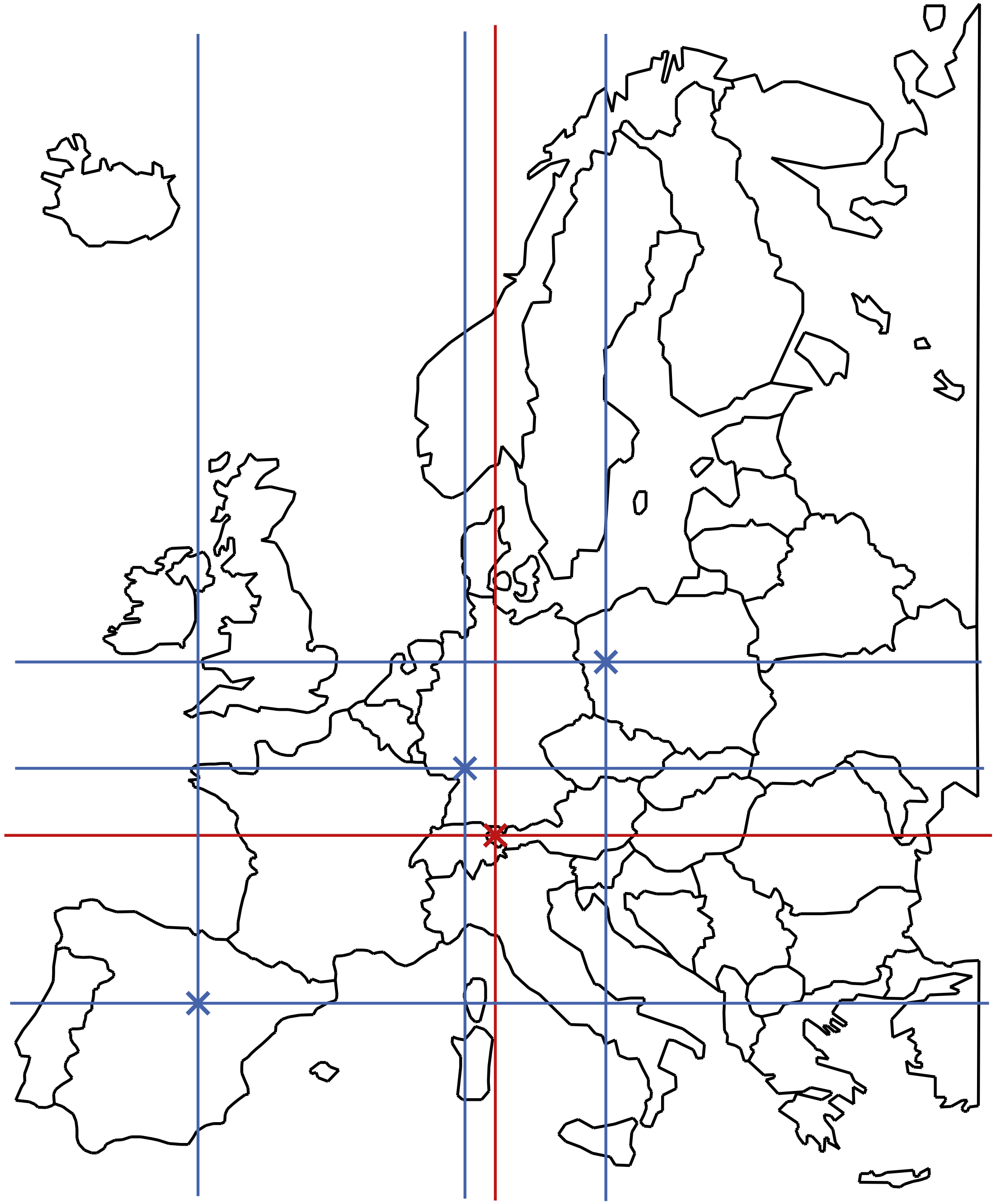
Given a position $p = (p_x, p_y)$ in a map, determine in which country p lies.

Motivation



Given a position $p = (p_x, p_y)$ in a map, determine in which country p lies.

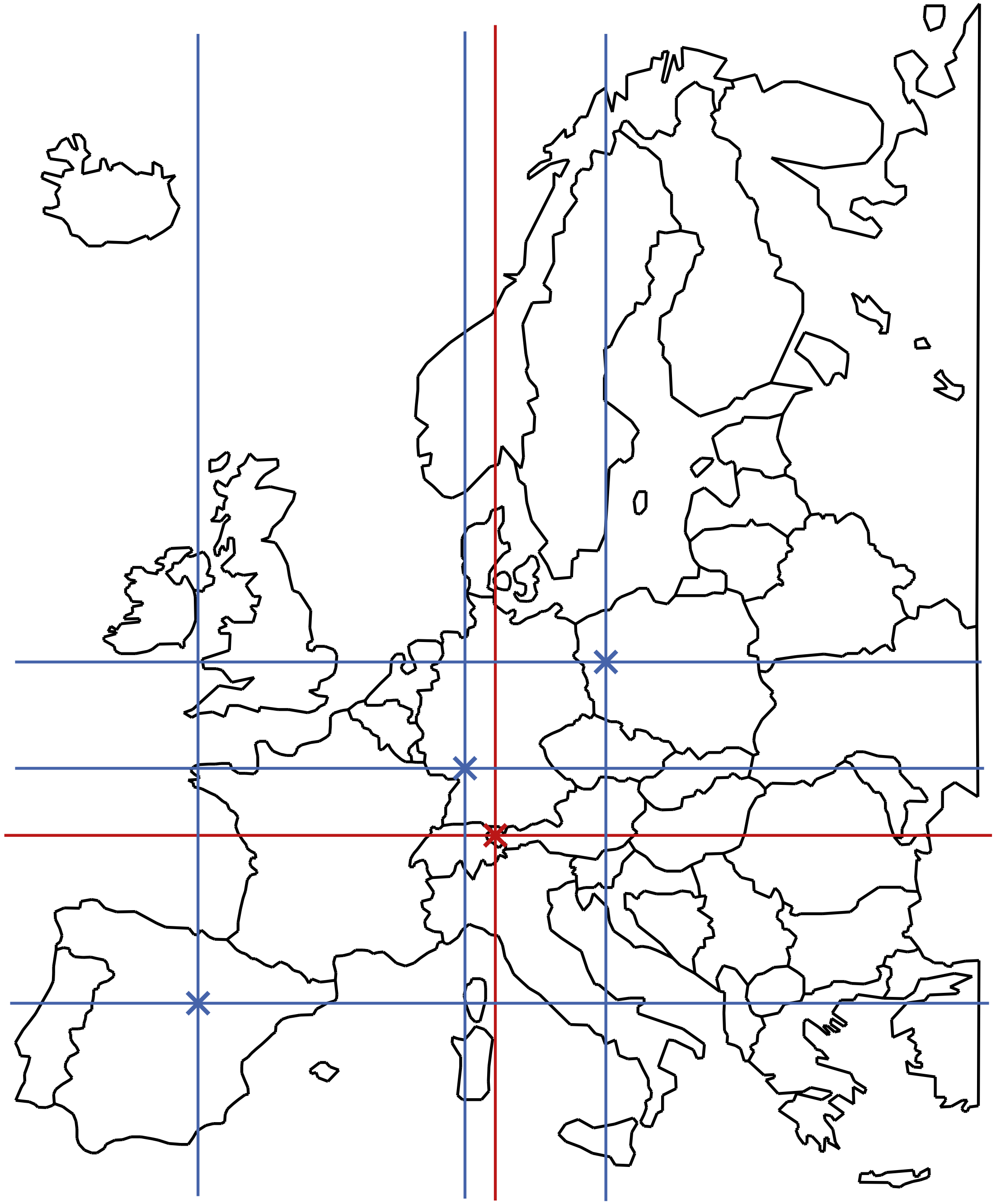
Motivation



Given a position $p = (p_x, p_y)$ in a map, determine in which country p lies.

more precisely:
Find a data structure for efficiently answering such point location queries.

Motivation

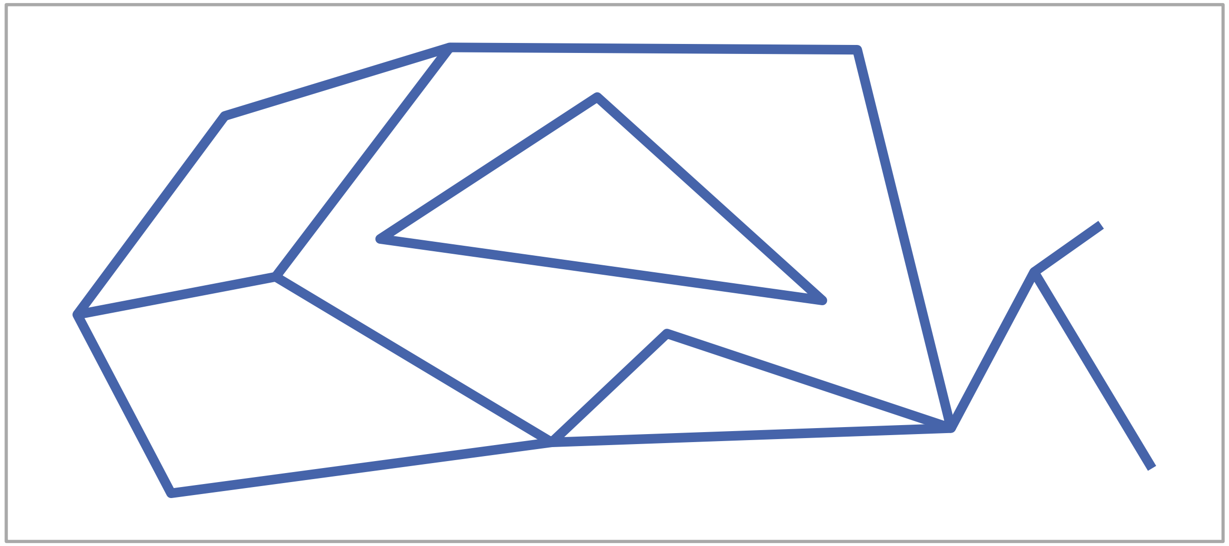


Given a position $p = (p_x, p_y)$ in a map, determine in which country p lies.

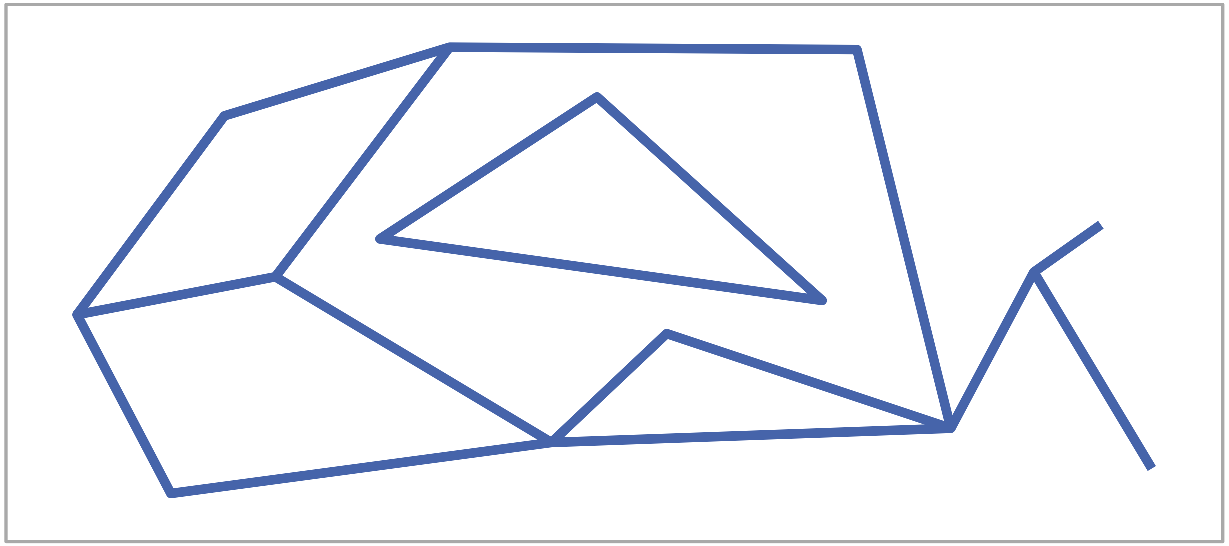
more precisely:
Find a data structure for efficiently answering such point location queries.

The map is modeled as a subdivision of the plane into disjoint polygons.

Problem Setting

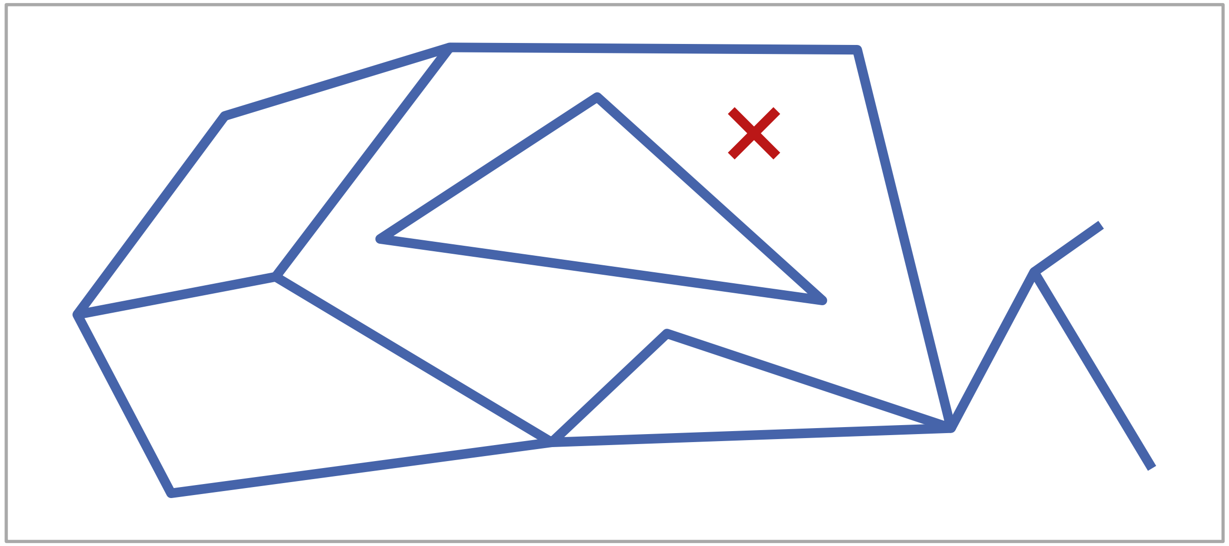


Problem Setting



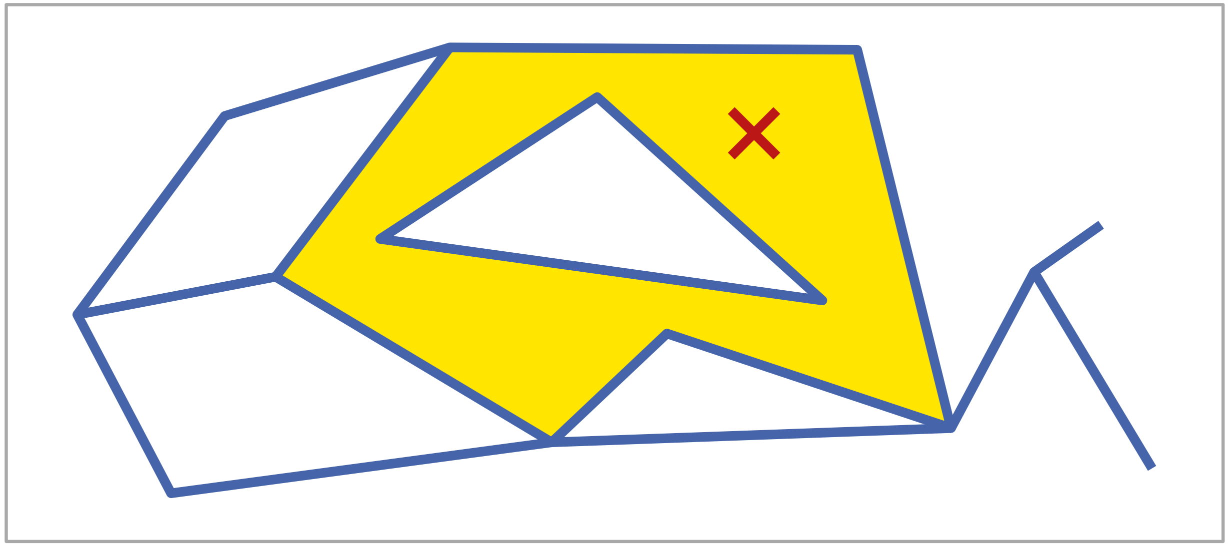
Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Problem Setting



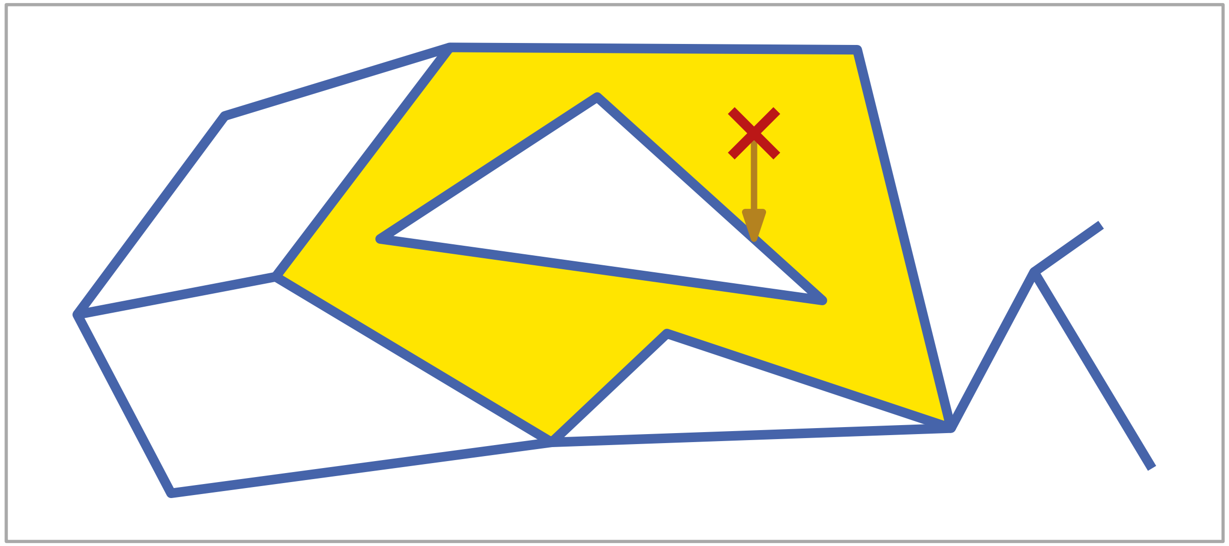
Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Problem Setting



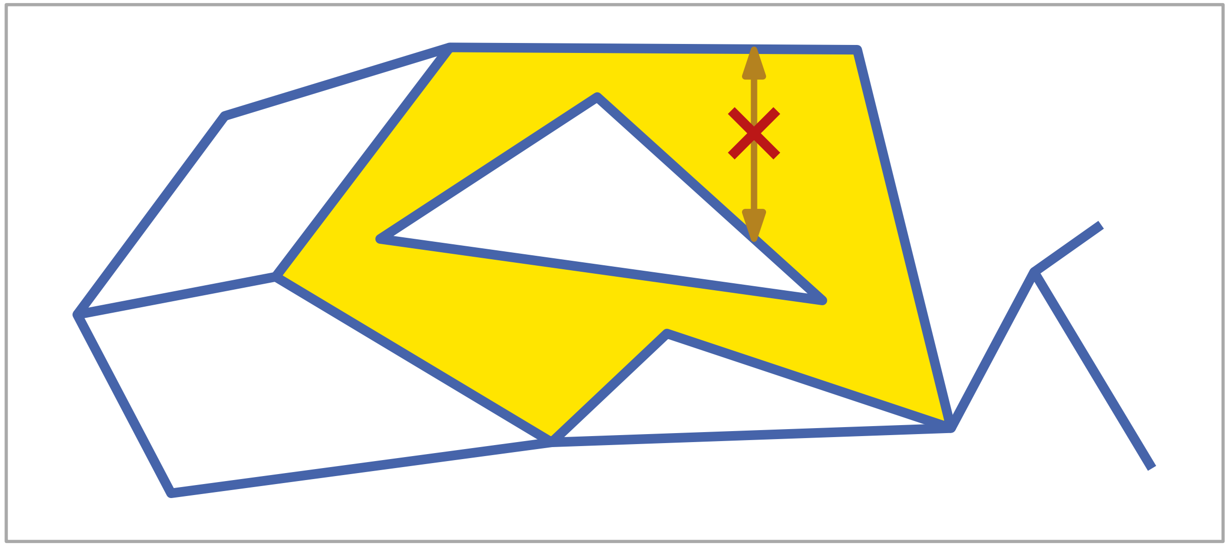
Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Problem Setting



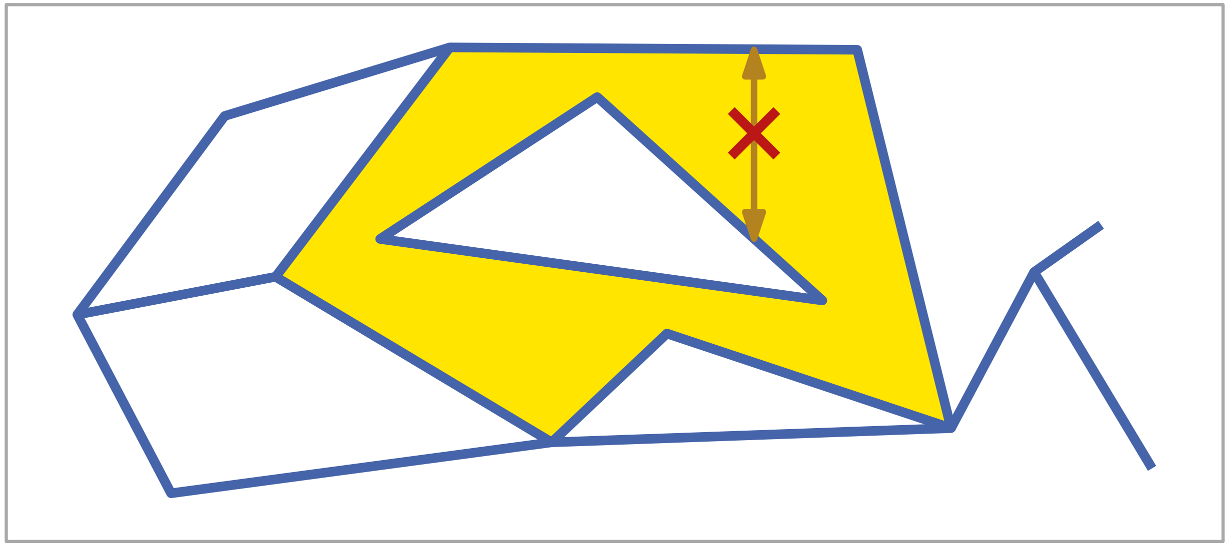
Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

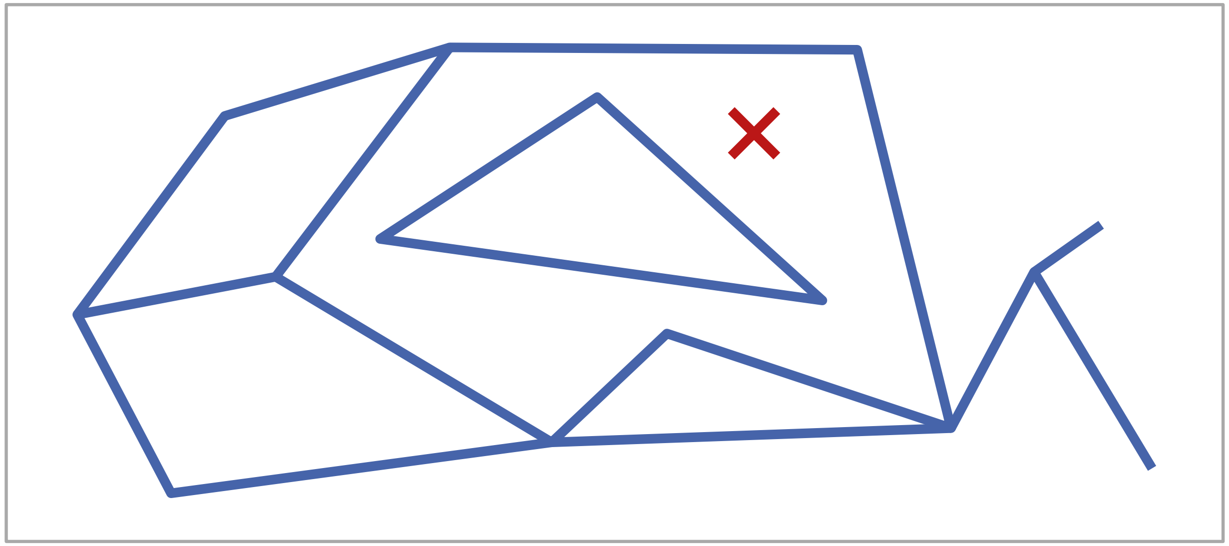
Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Think for 2 minutes!

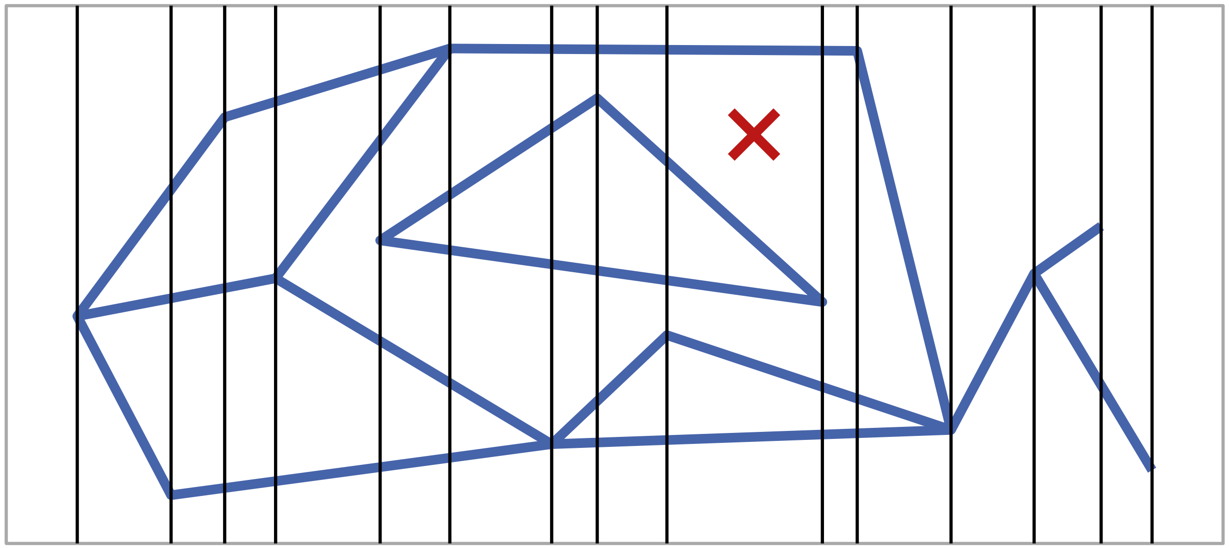
Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

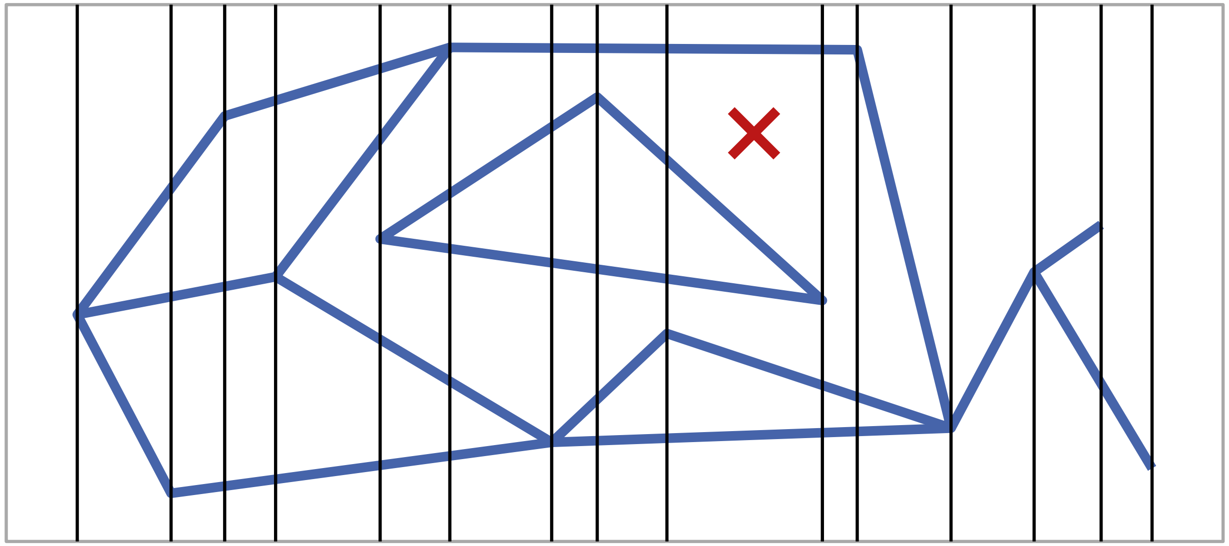
Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Problem Setting

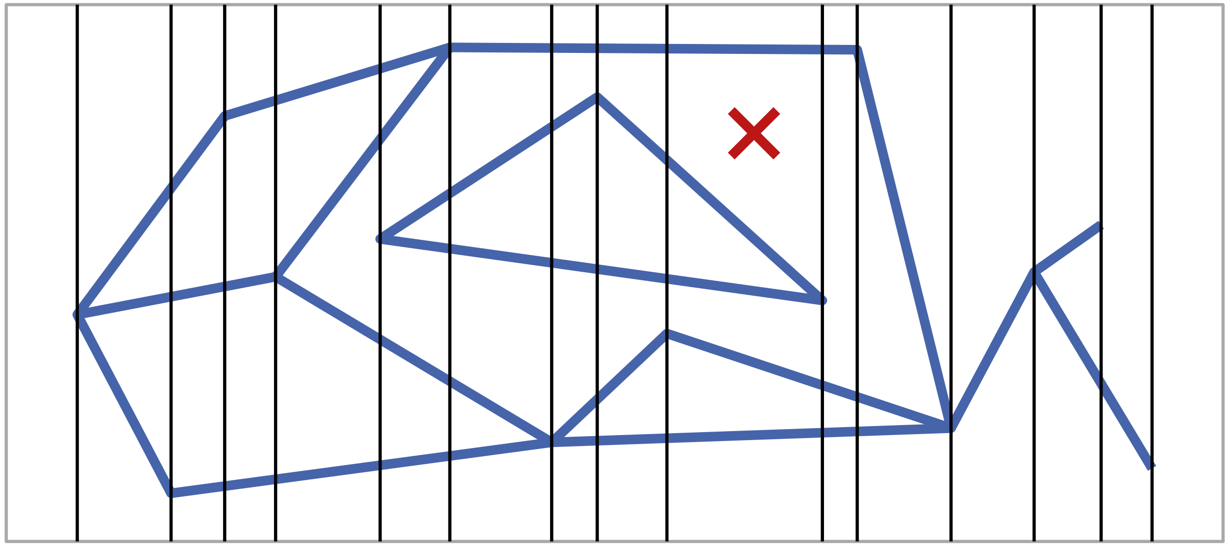


Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:

Problem Setting

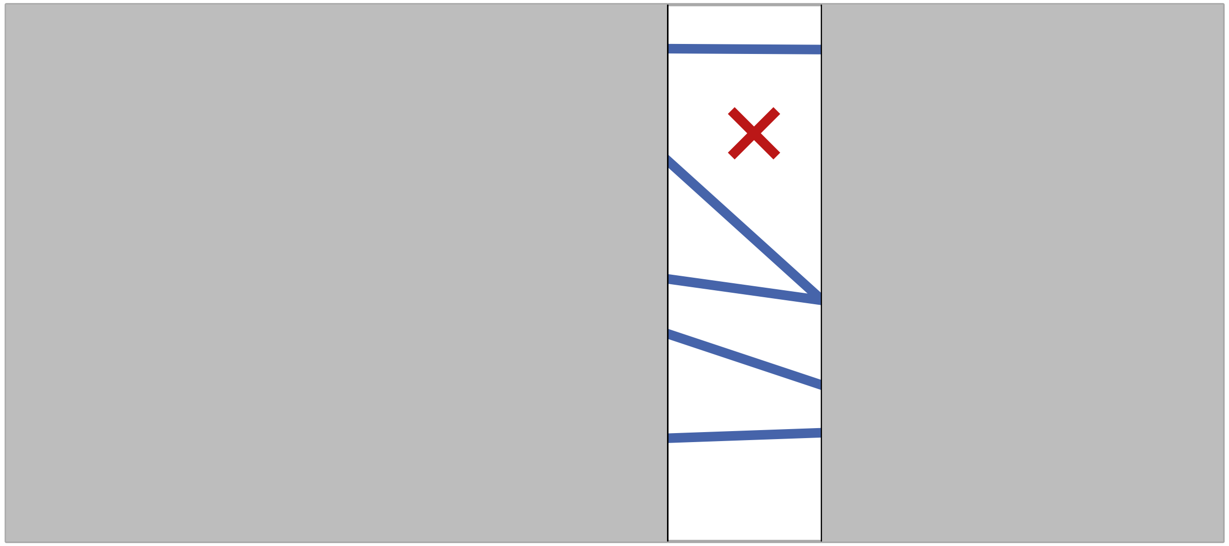


Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab

Problem Setting

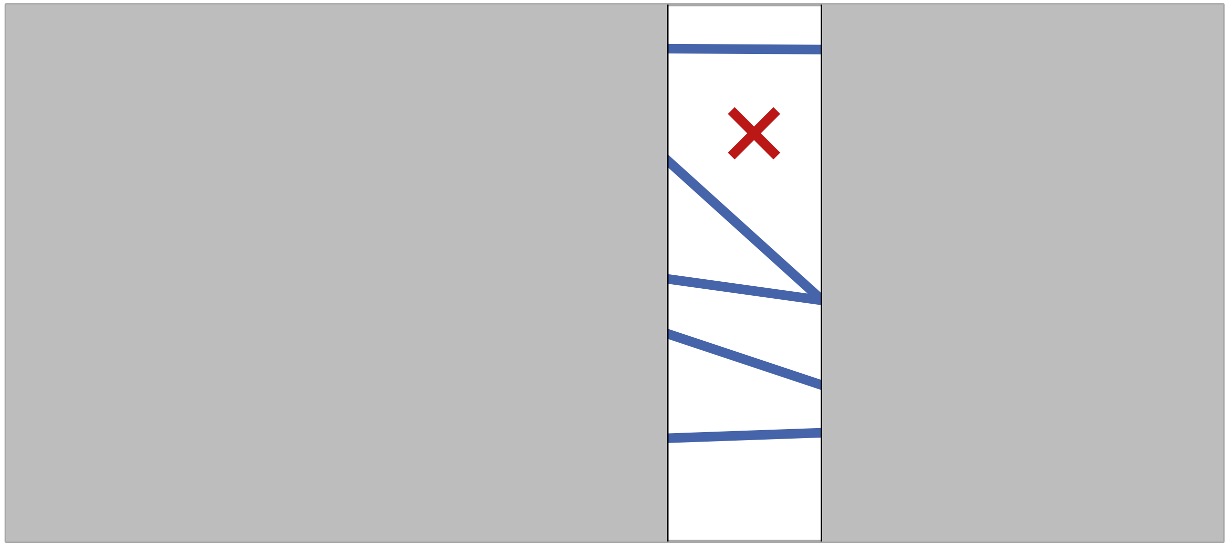


Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab

Problem Setting

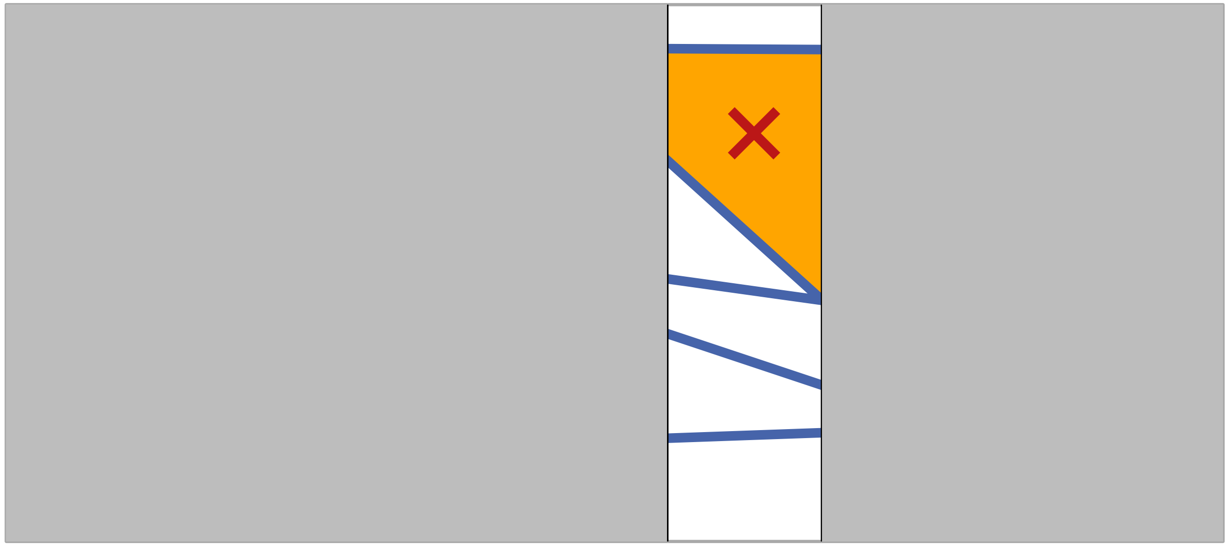


Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

Problem Setting

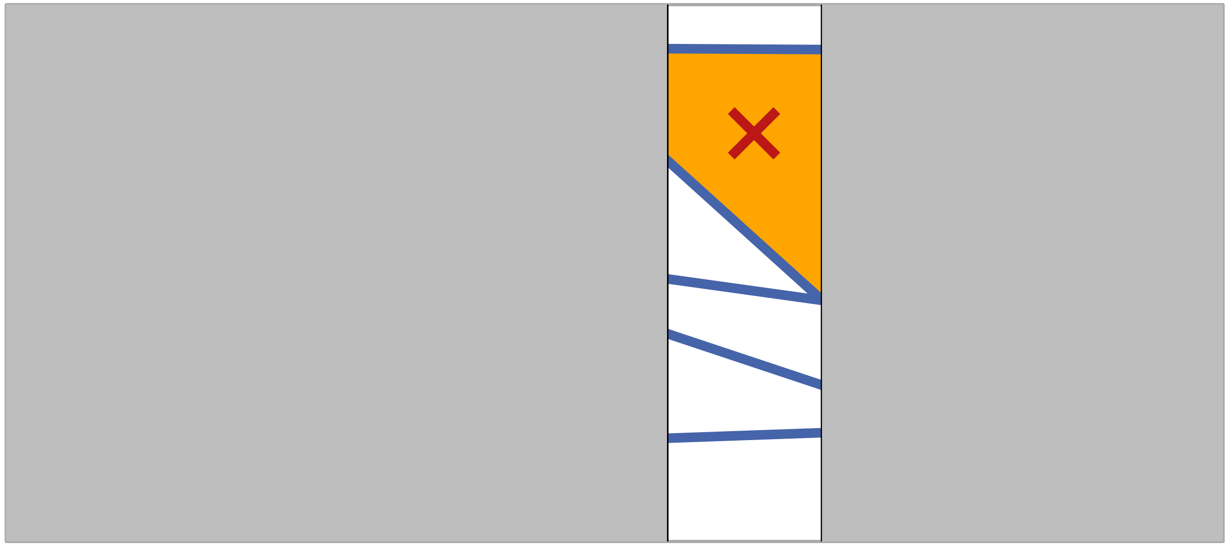


Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

Problem Setting



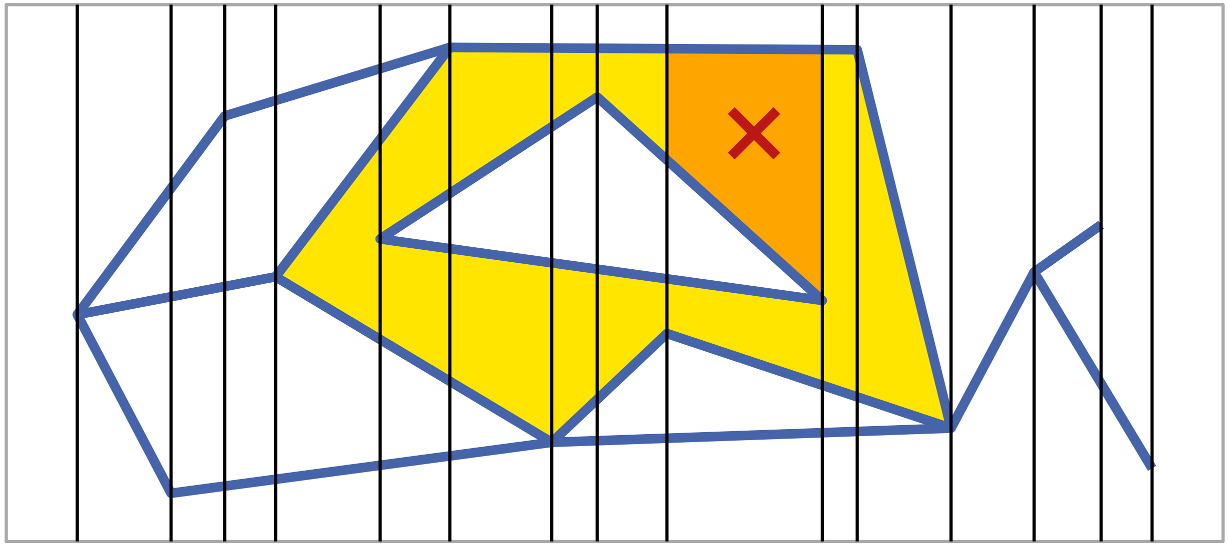
Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

} 2 binary searches

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

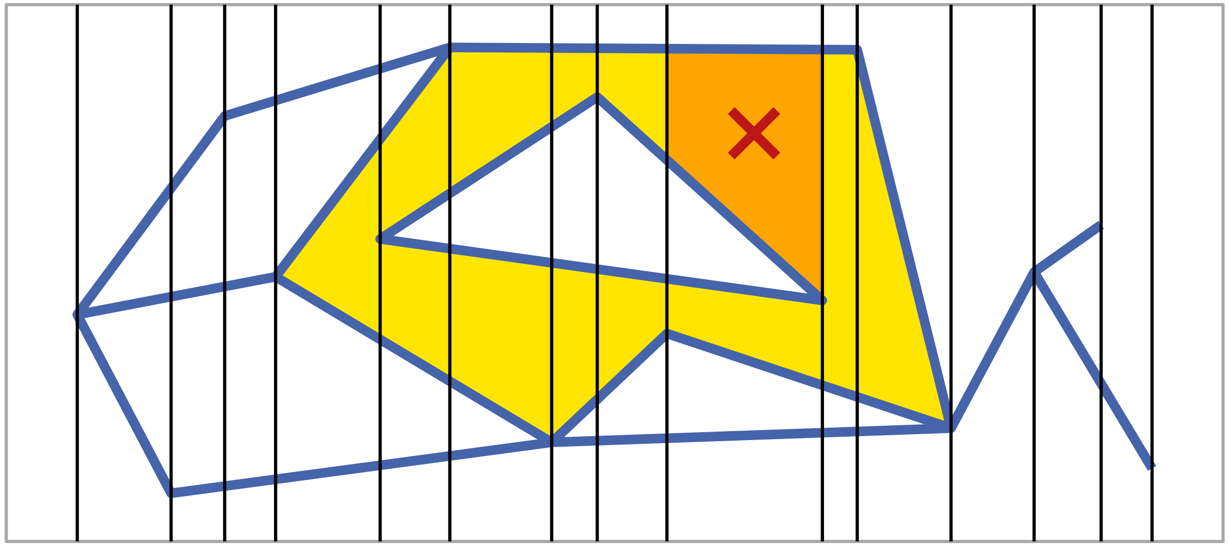
Solution: Partition $\mathcal{S}$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

$\Big\}$ 2 binary searches

$O(\log n)$ time

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

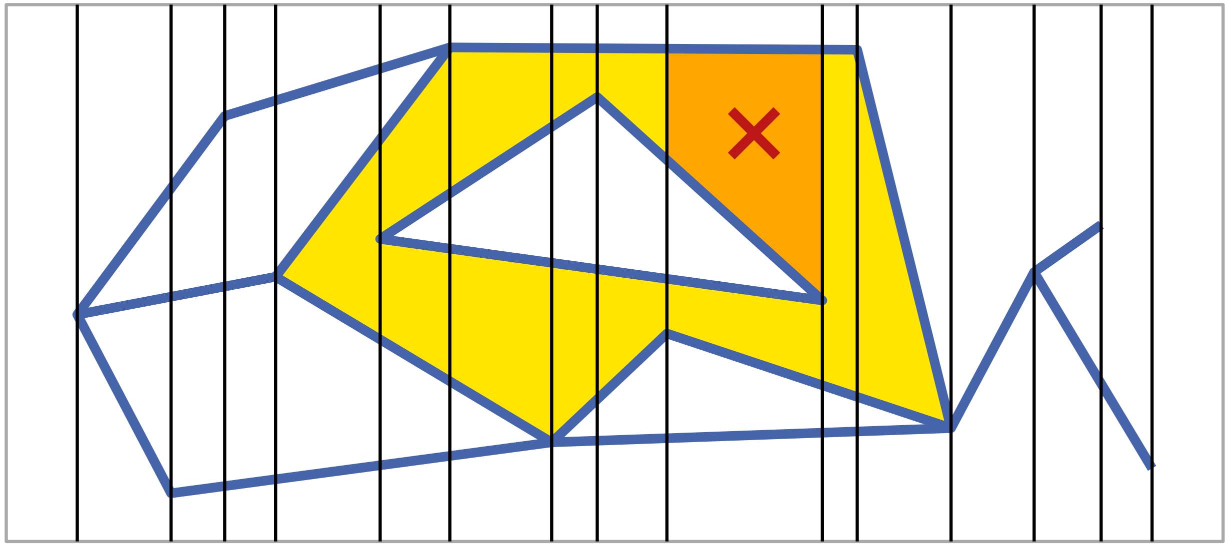
Query:
- find correct slab
- search this slab

$\Big\}$ 2 binary searches

$O(\log n)$ time

But:

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

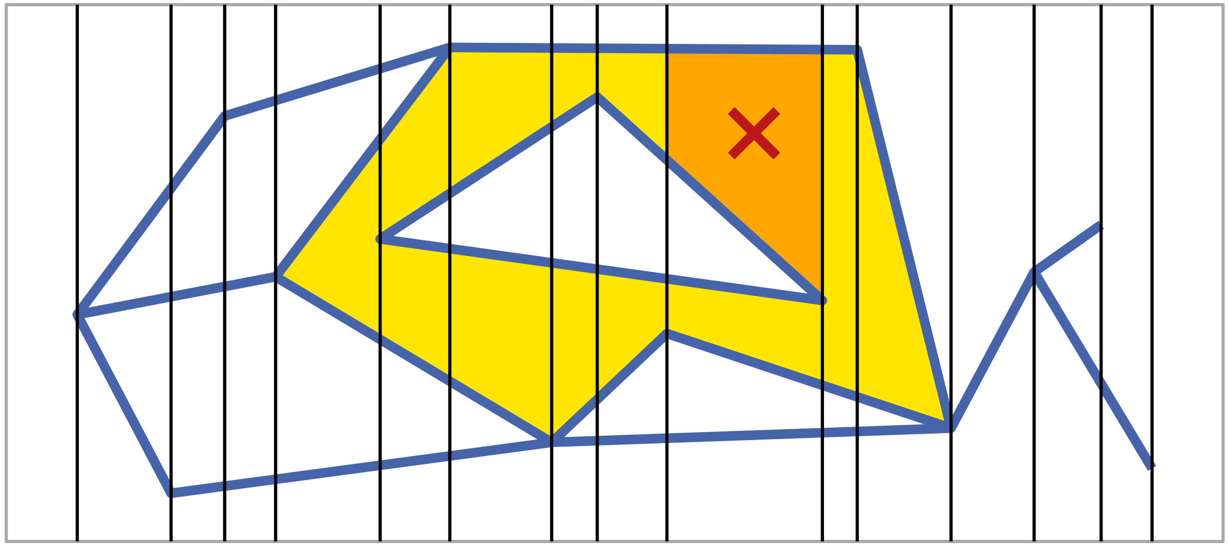
$O(\log n)$ time

Query:
- find correct slab
- search this slab

} 2 binary searches

But: Space?

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

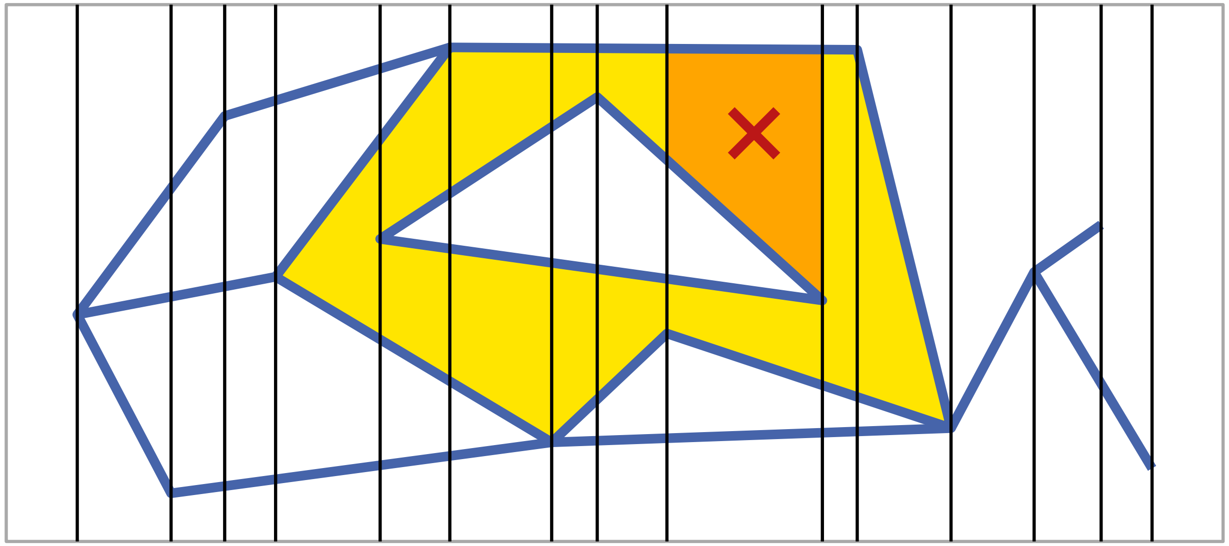
Query:
- find correct slab
- search this slab

} 2 binary searches

$O(\log n)$ time

But: Space? $\Theta(n^2)$

Problem Setting



Goal: Given subdivision $\mathcal{S}$ of the plane with n segments, construct data structure for fast point location queries.

Solution: Partition $\mathcal{S}$ at points into vertical slabs.

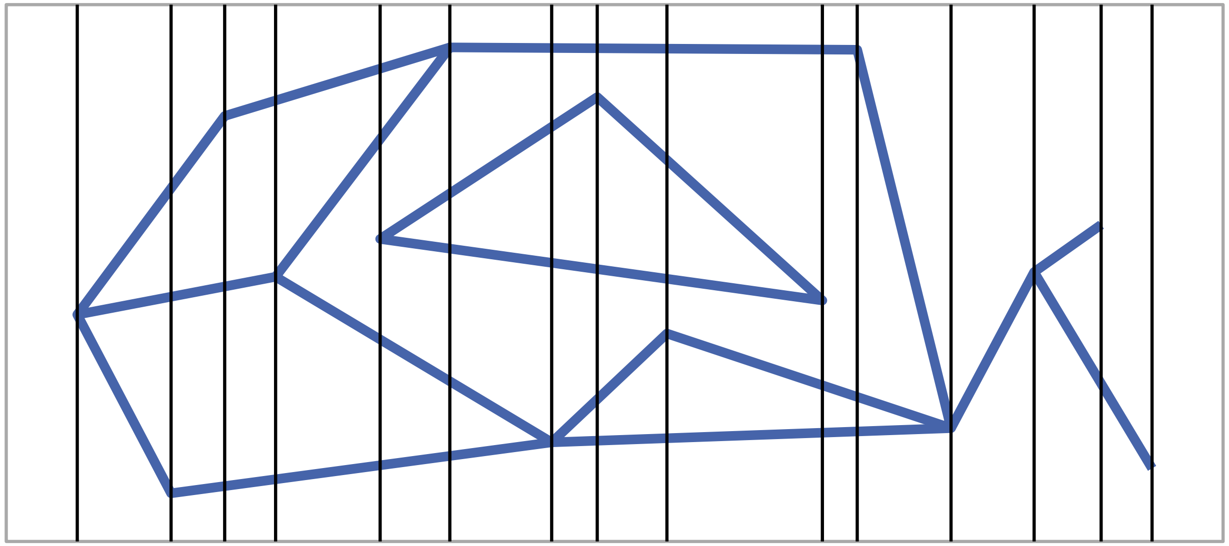
Query:
- find correct slab
- search this slab

$\left.\right\}$ 2 binary searches — $O(\log n)$ time

But: Space? $\Theta(n^2)$ **Question:** lower bound example?

Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.



Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

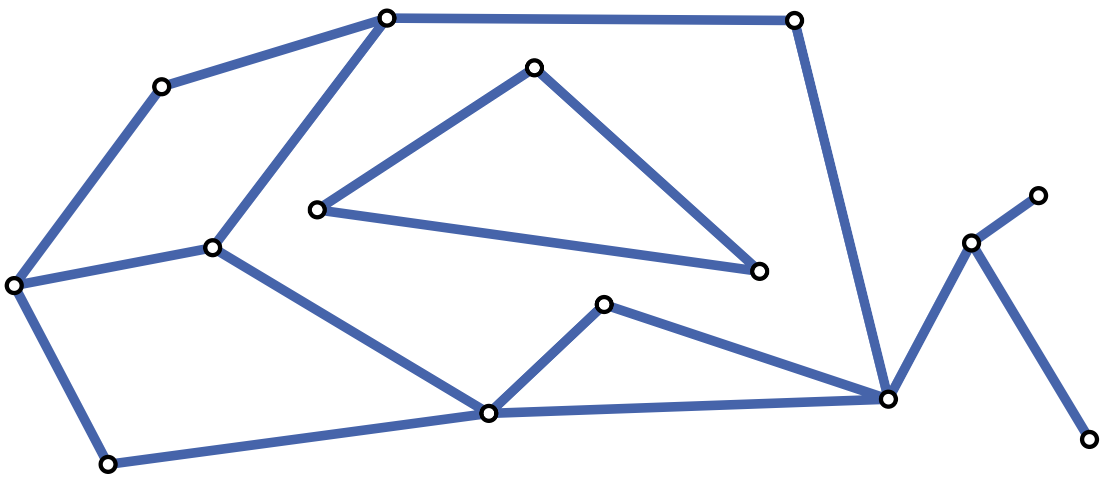
Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

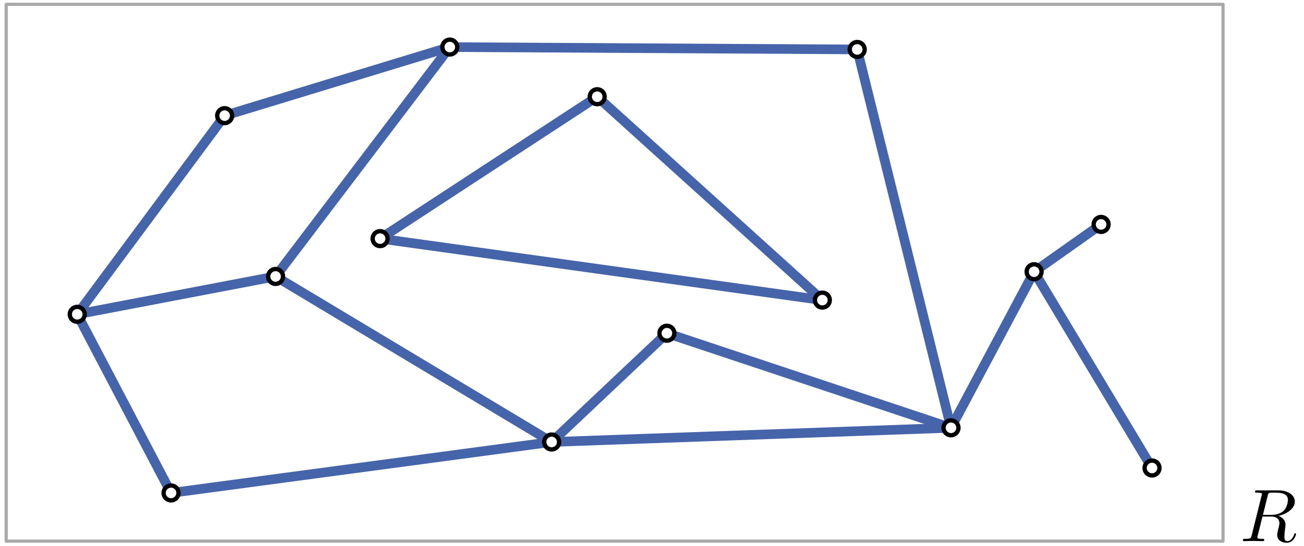


Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$



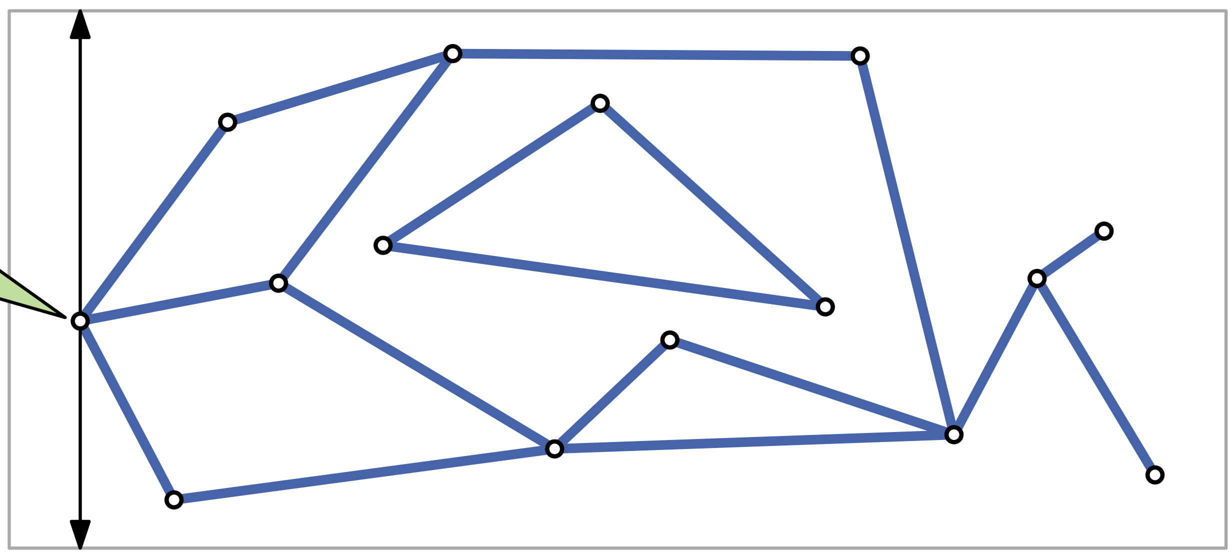
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

edge from each endpoint vertically up and down to neighboring segment



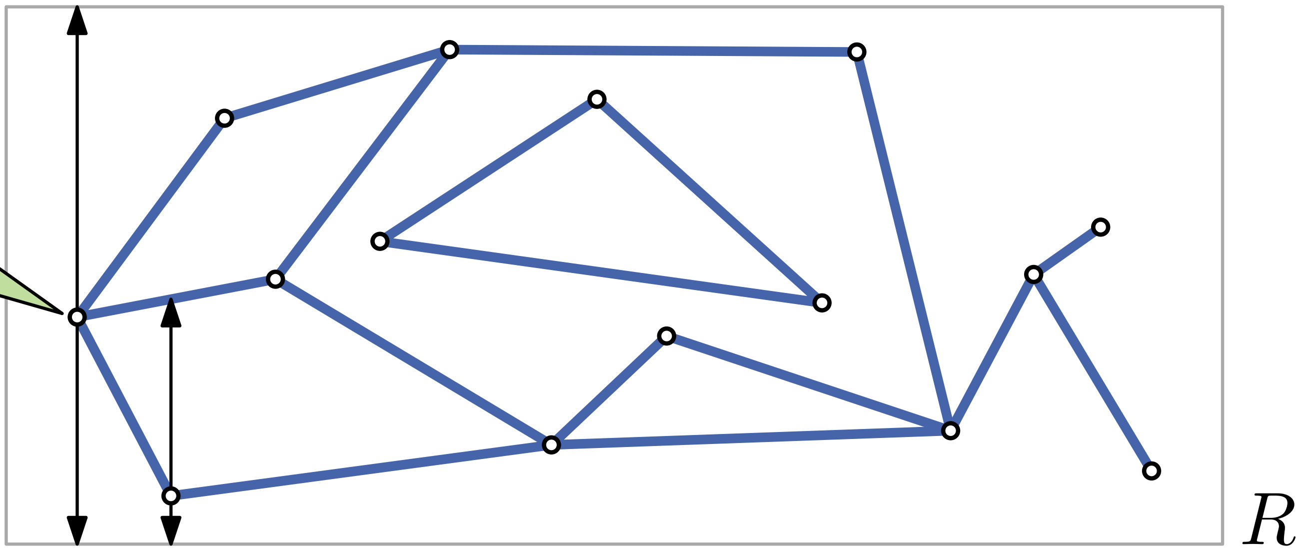
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

edge from each endpoint vertically up and down to neighboring segment

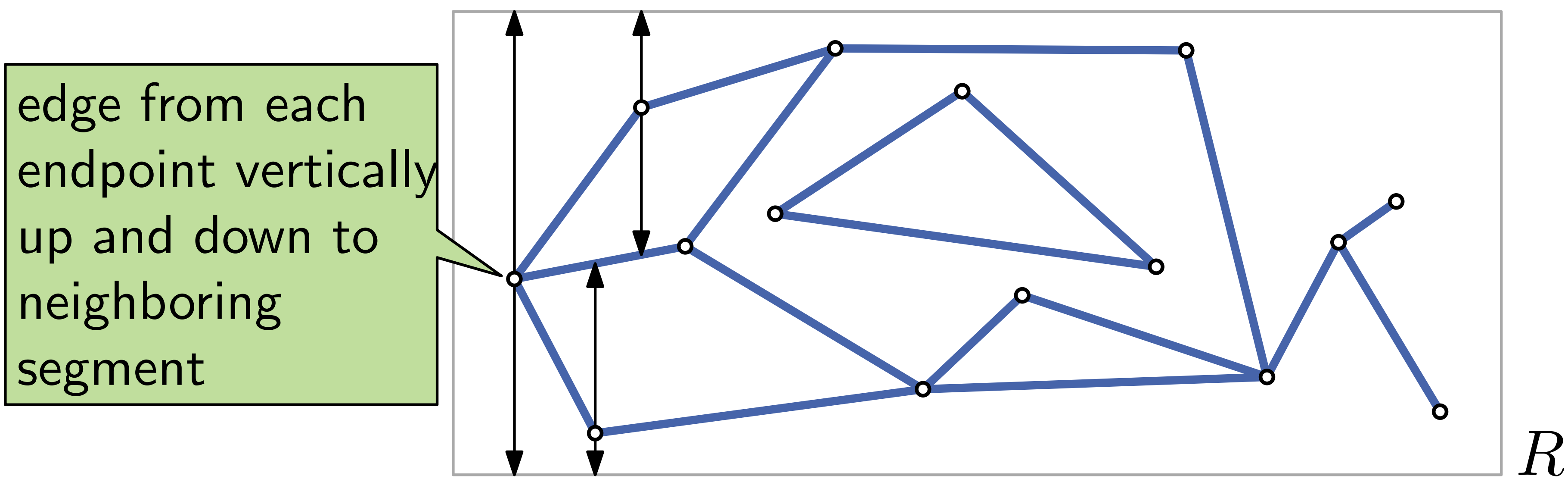


Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$



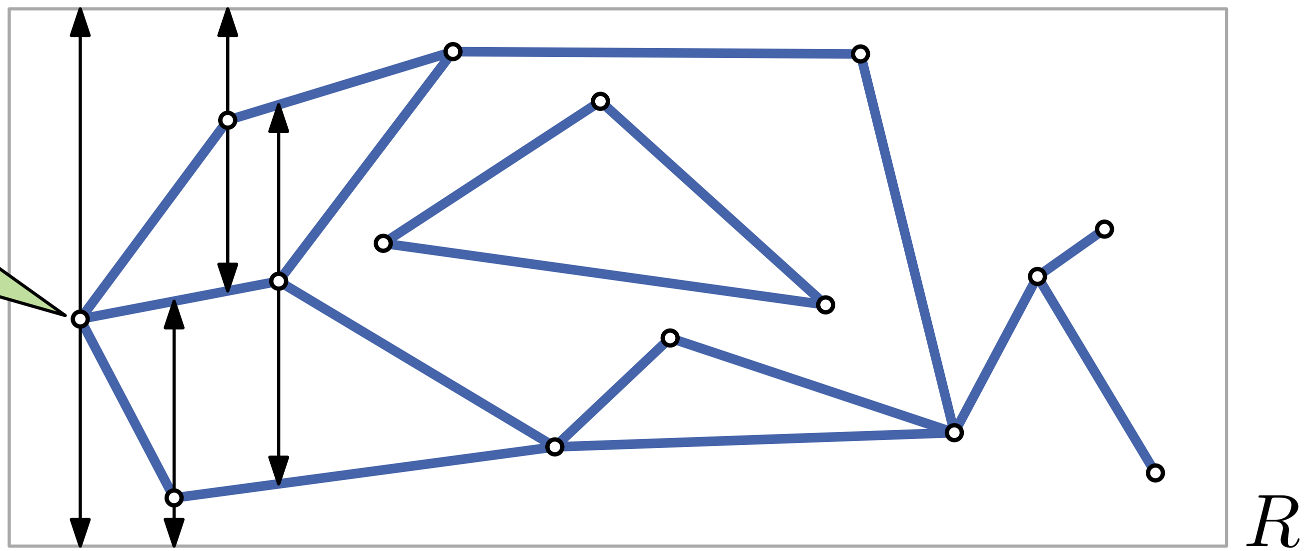
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

edge from each endpoint vertically up and down to neighboring segment



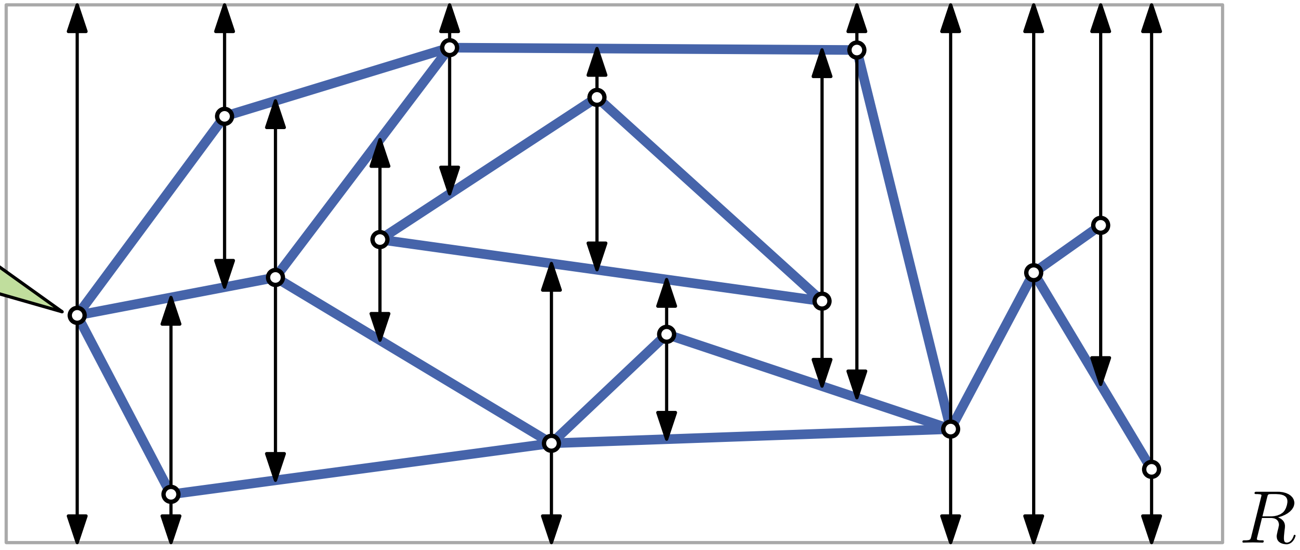
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

edge from each endpoint vertically up and down to neighboring segment



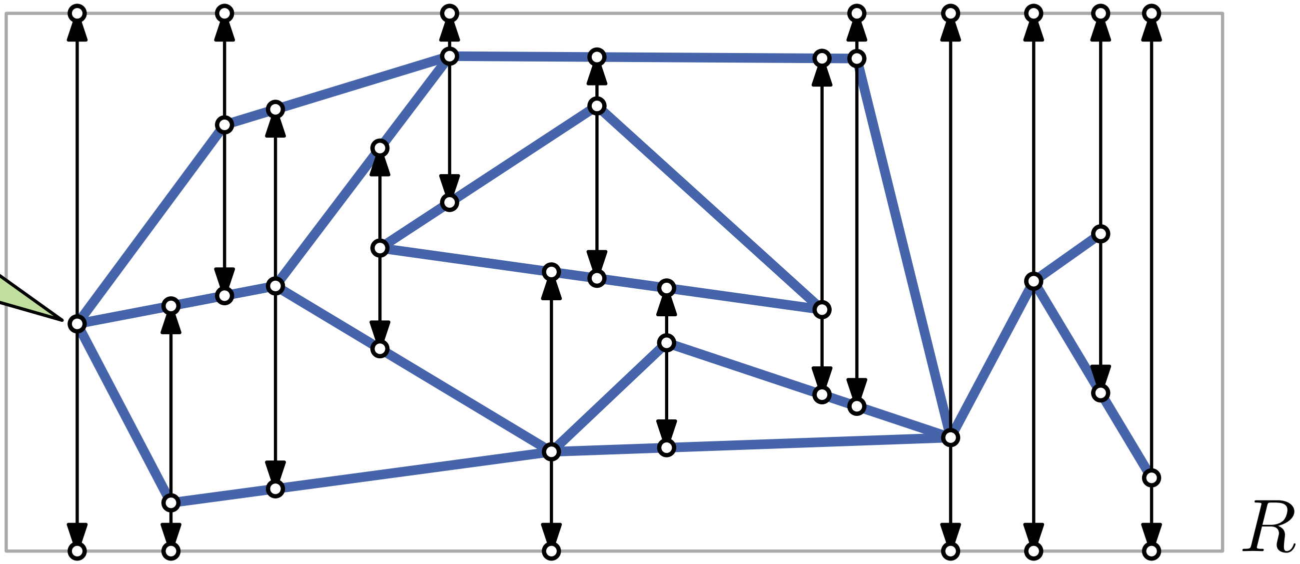
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

edge from each endpoint vertically up and down to neighboring segment

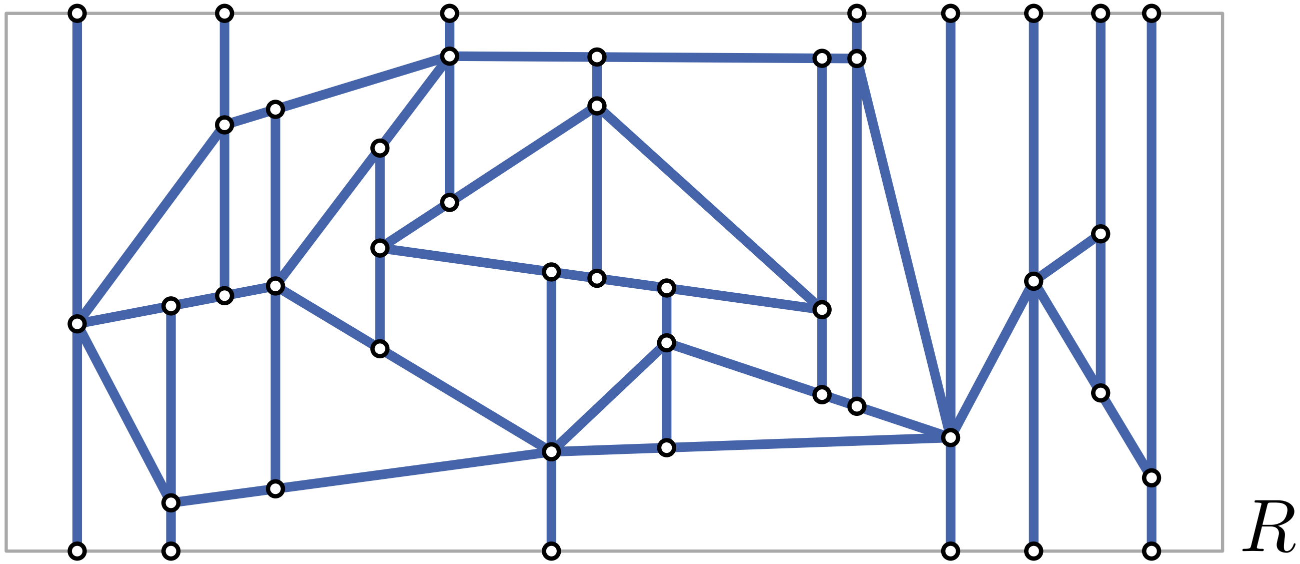


Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$



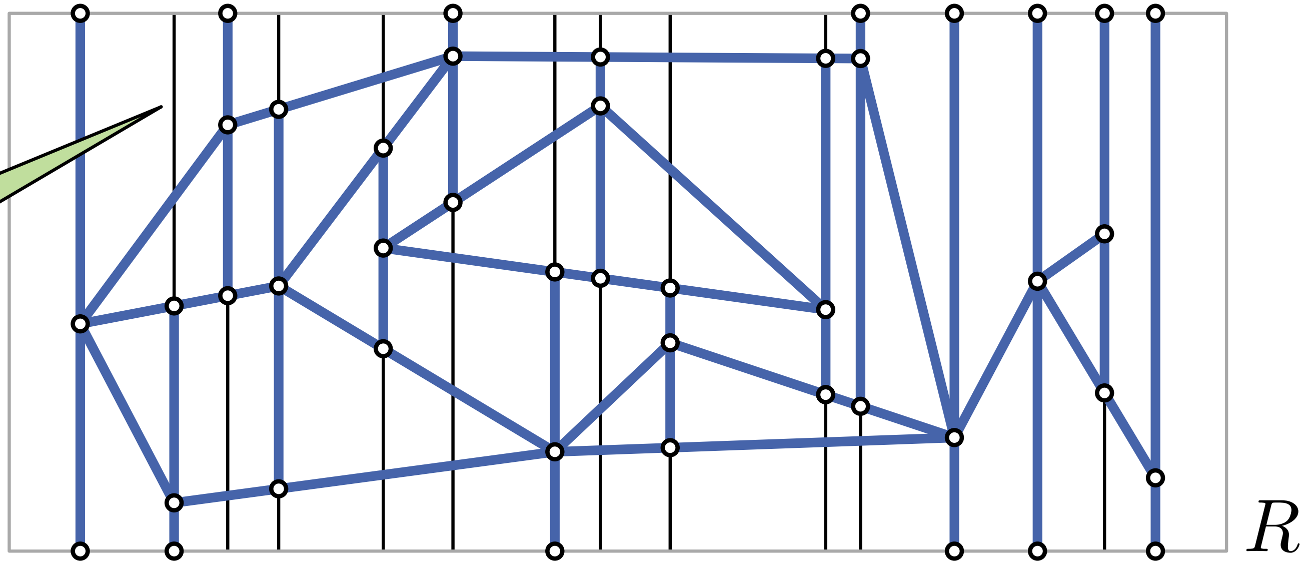
Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$

Compare to slab partition

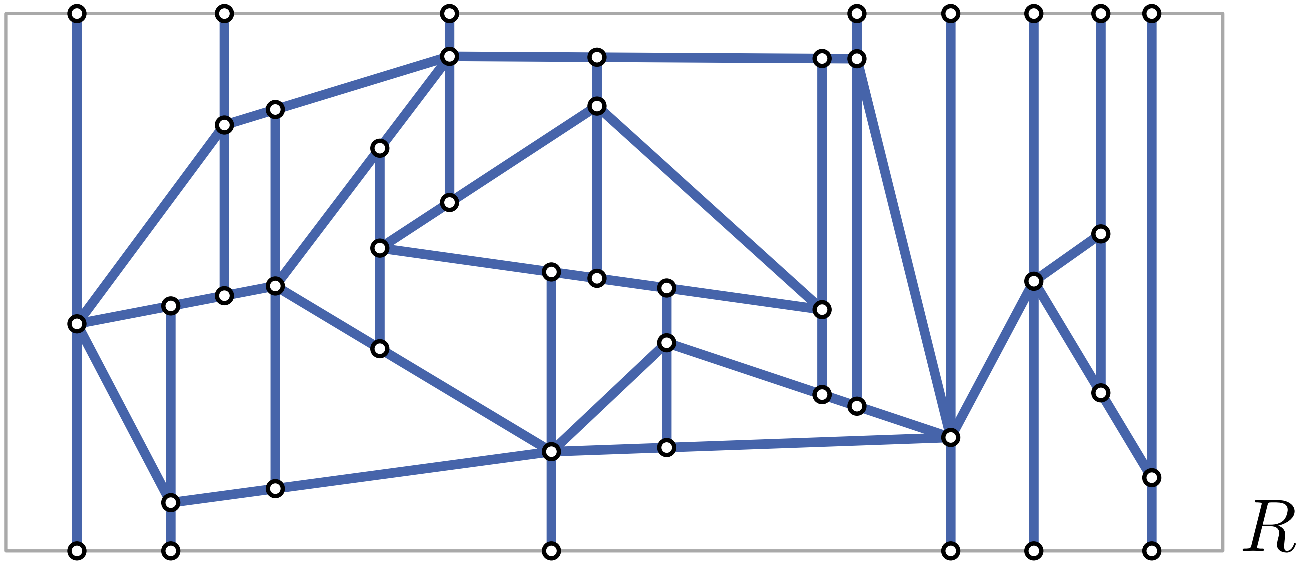


Reducing the Complexity

Observation: Slab partition is a refinement $\mathcal{S}'$ of $\mathcal{S}$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $\mathcal{S}$ with lower complexity!

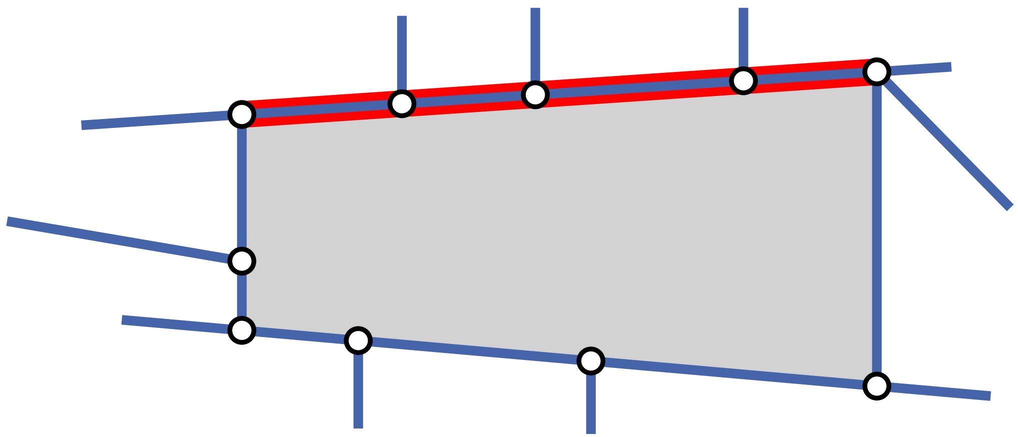
Solution: *Trapezoidal map* $\mathcal{T}(\mathcal{S})$



Assumption: $\mathcal{S}$ is in *general position*, i.e., no two segment endpoints have the same x-coordinate

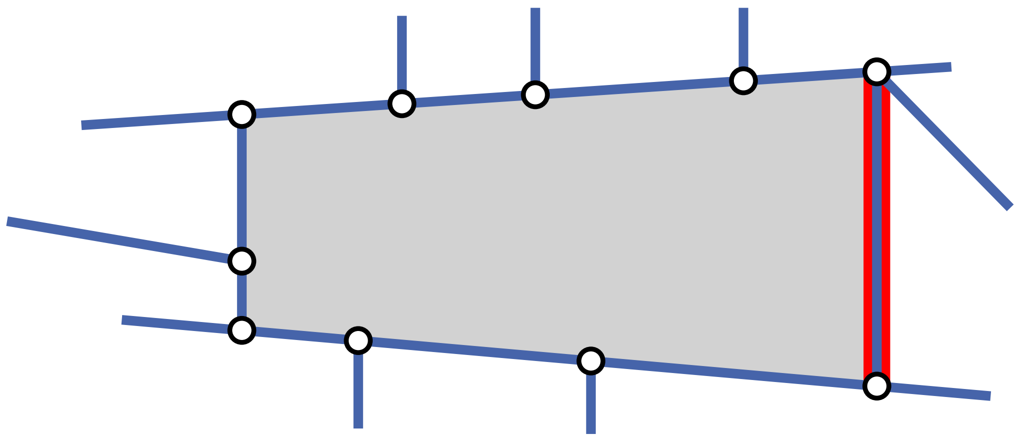
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



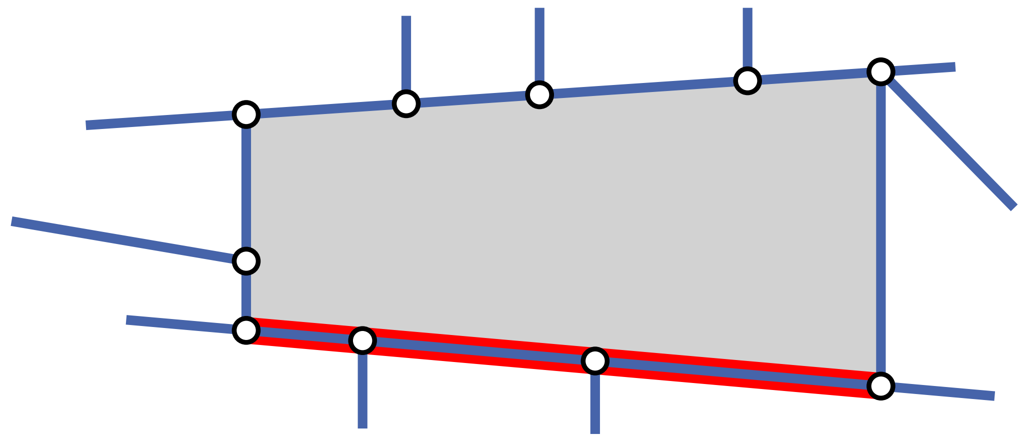
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



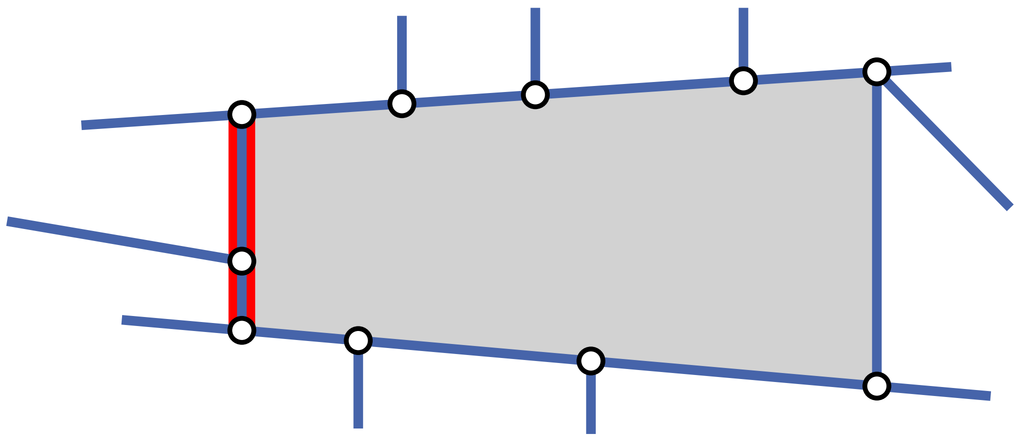
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



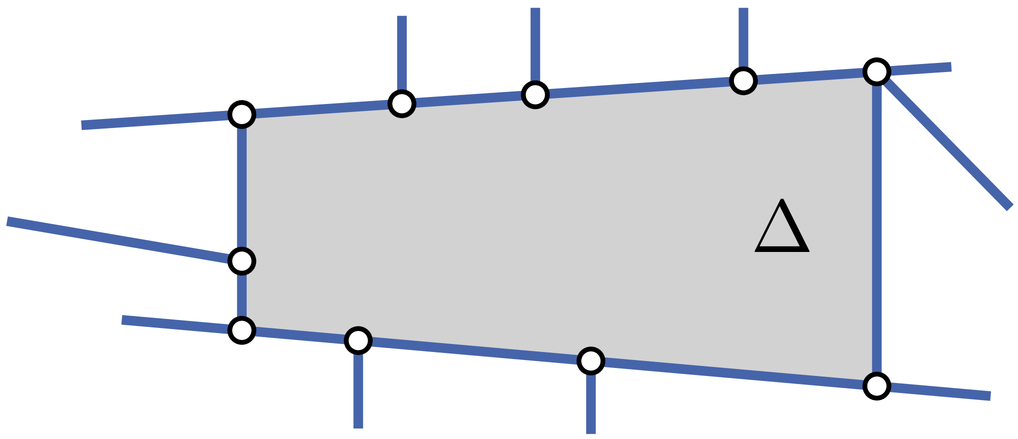
Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Notation

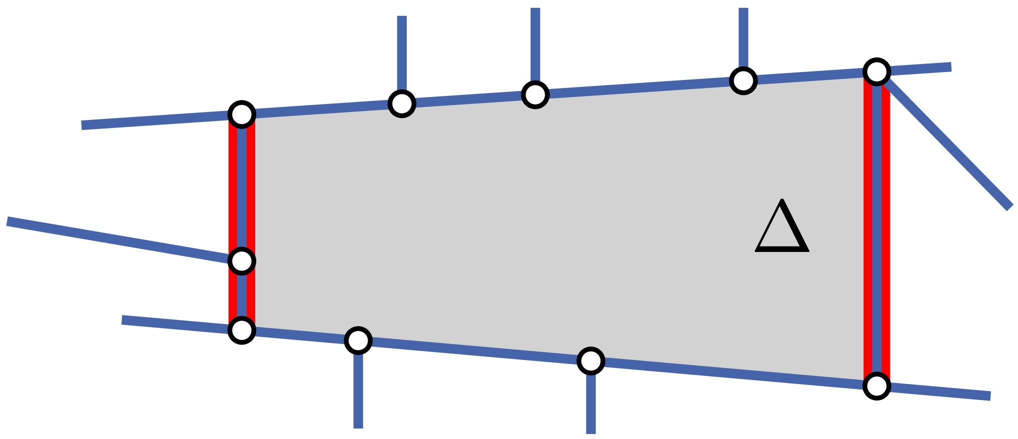
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.

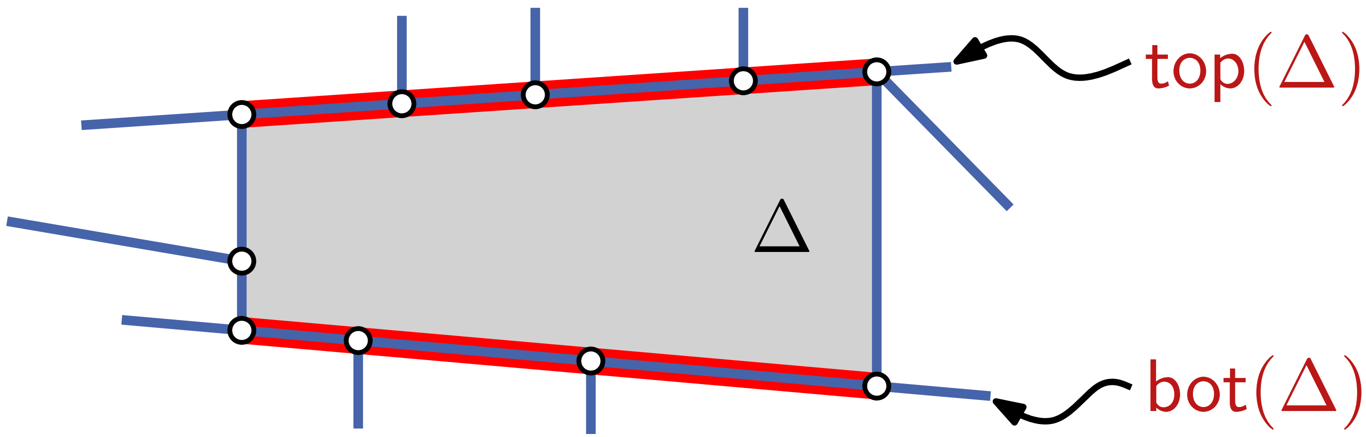


Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

- one or two vertical sides

Notation

Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.

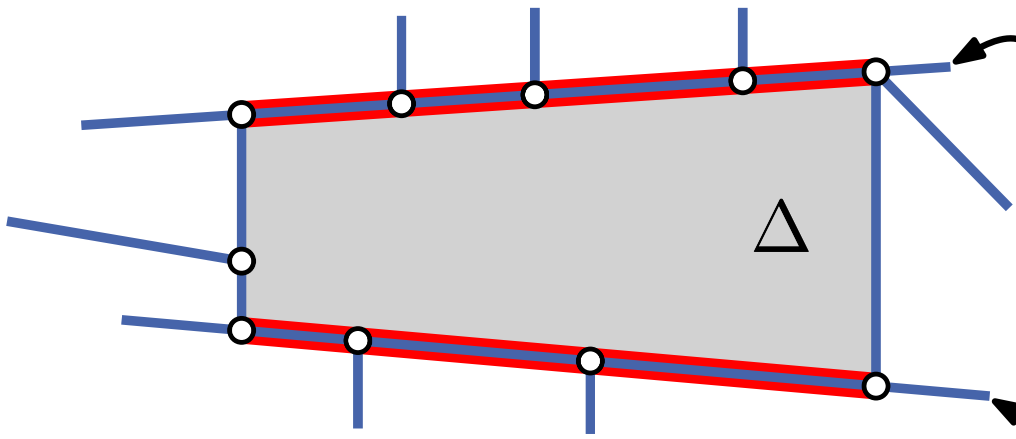


Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

- one or two vertical sides
- two non-vertical sides

Notation

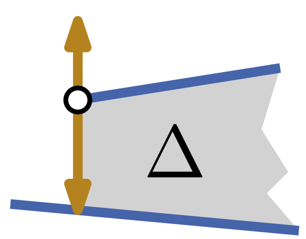
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

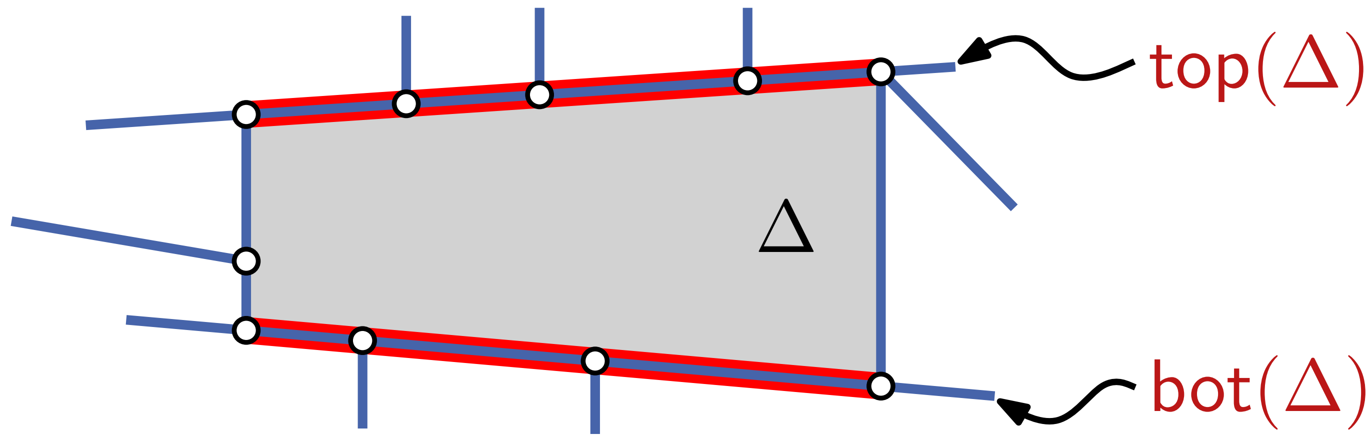
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

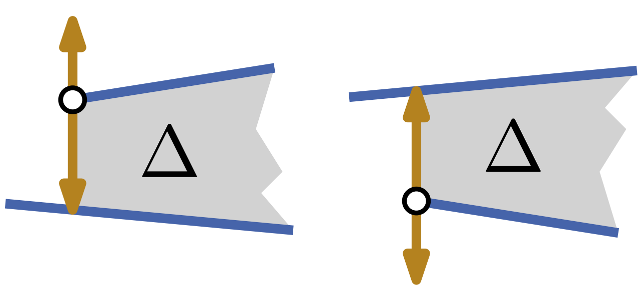
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

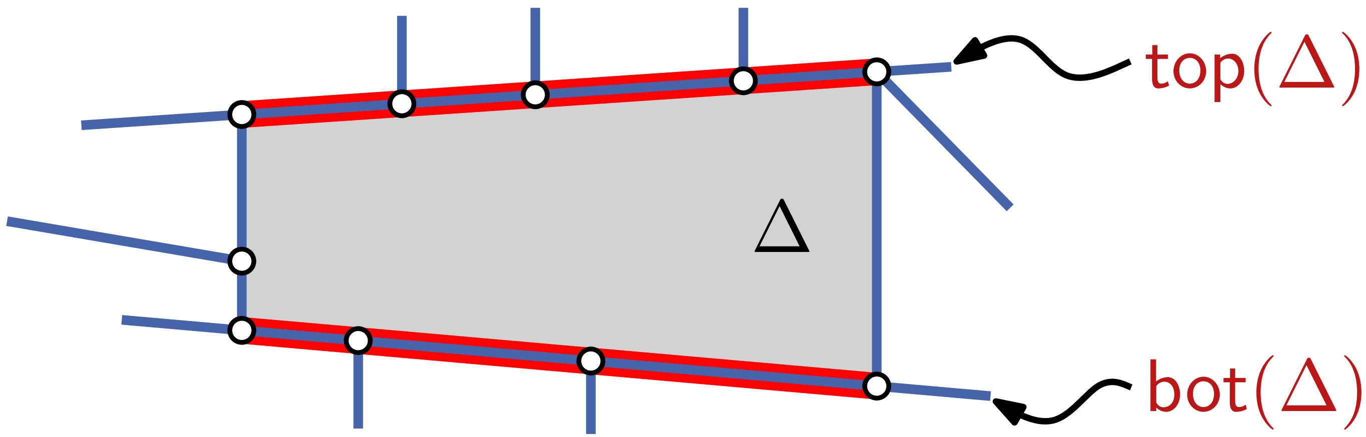
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

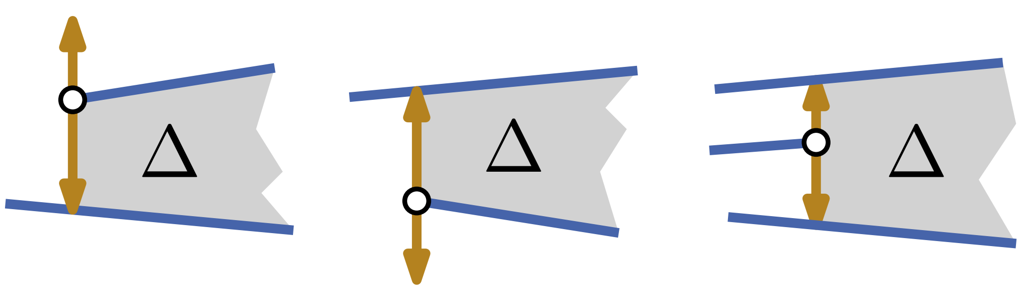
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

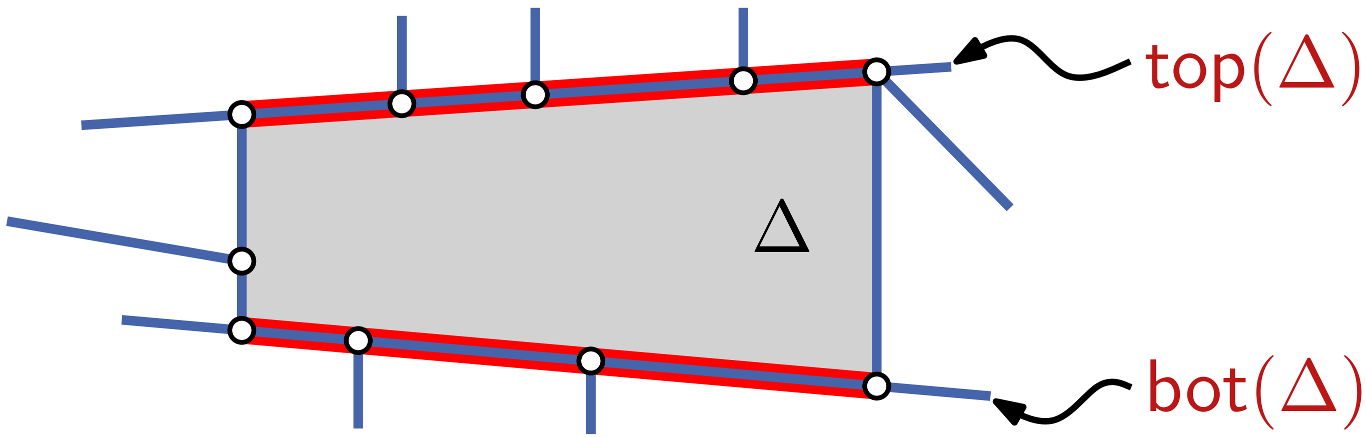
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

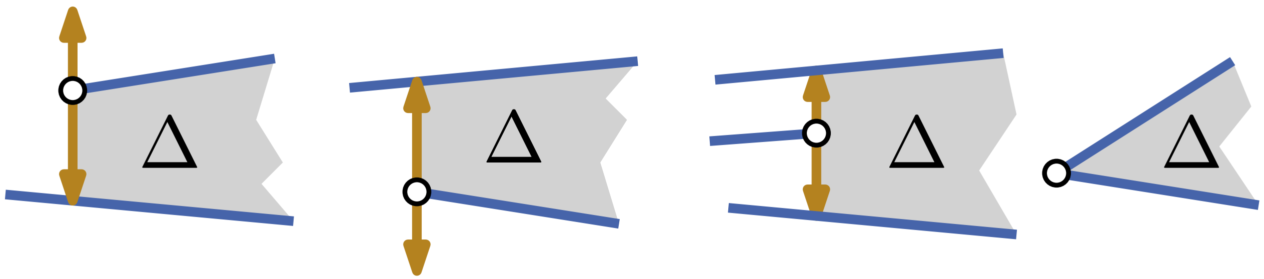
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

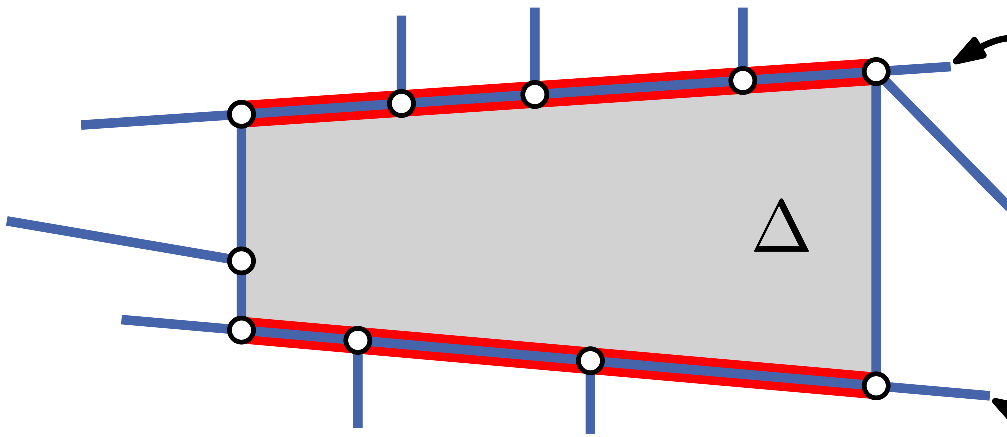
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

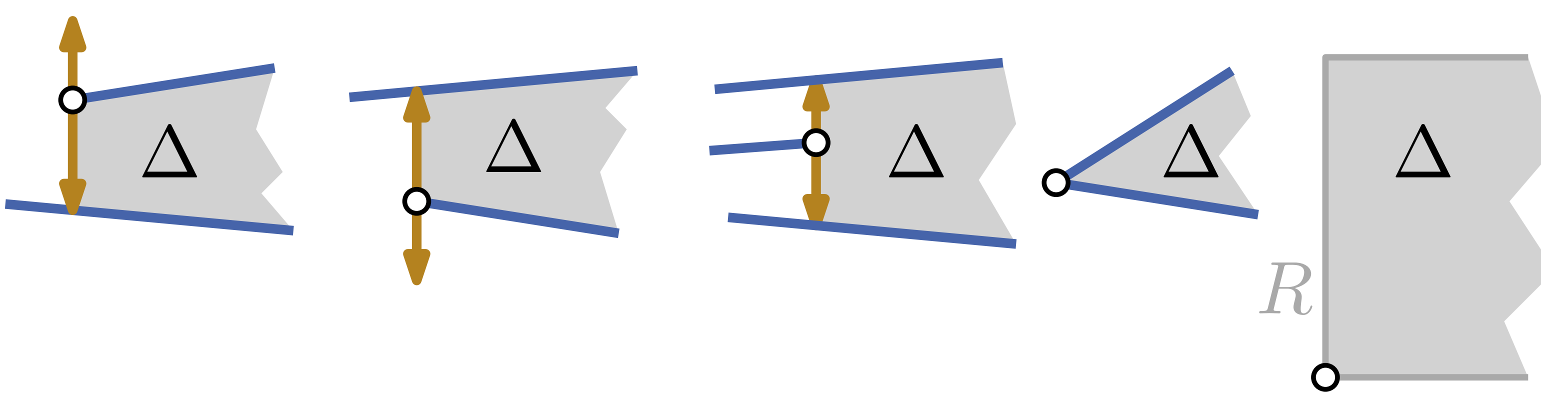
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

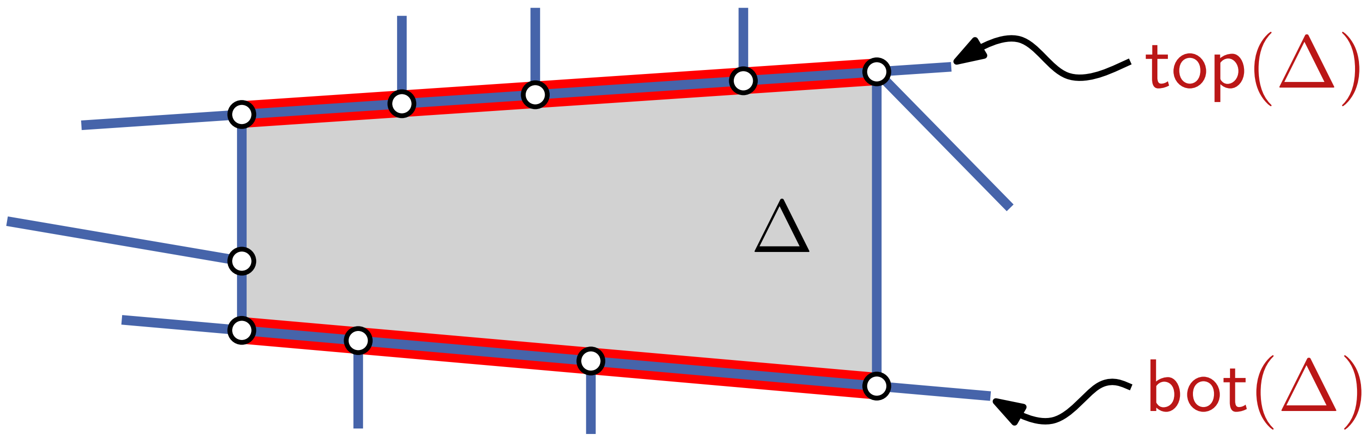
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

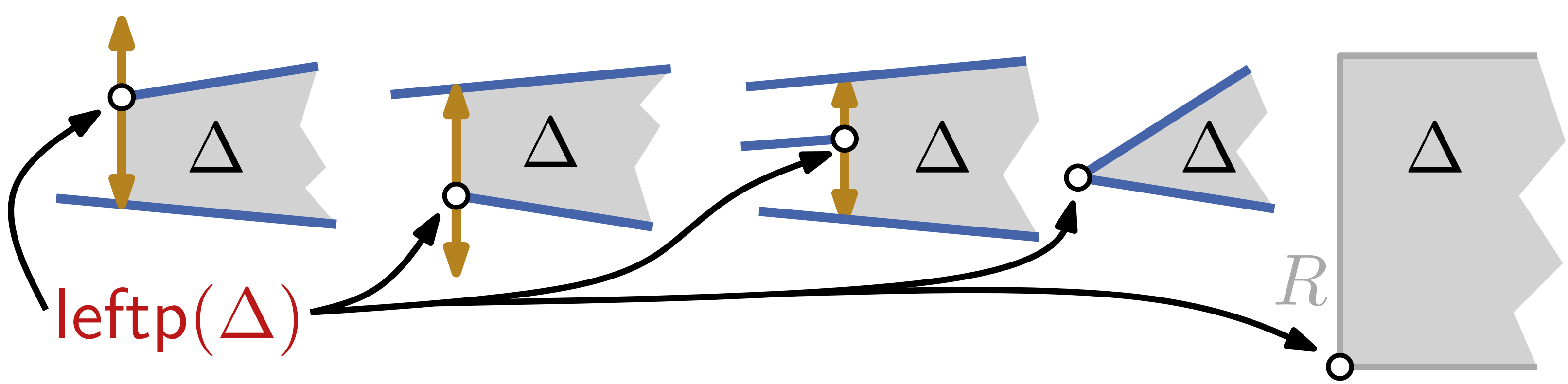
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

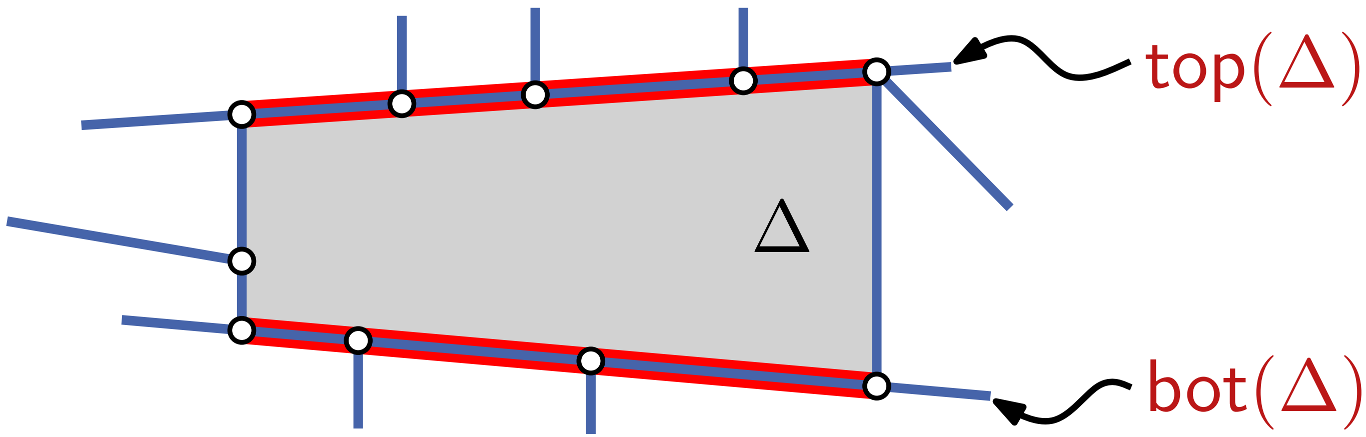
- one or two vertical sides
- two non-vertical sides

Left side:



Notation

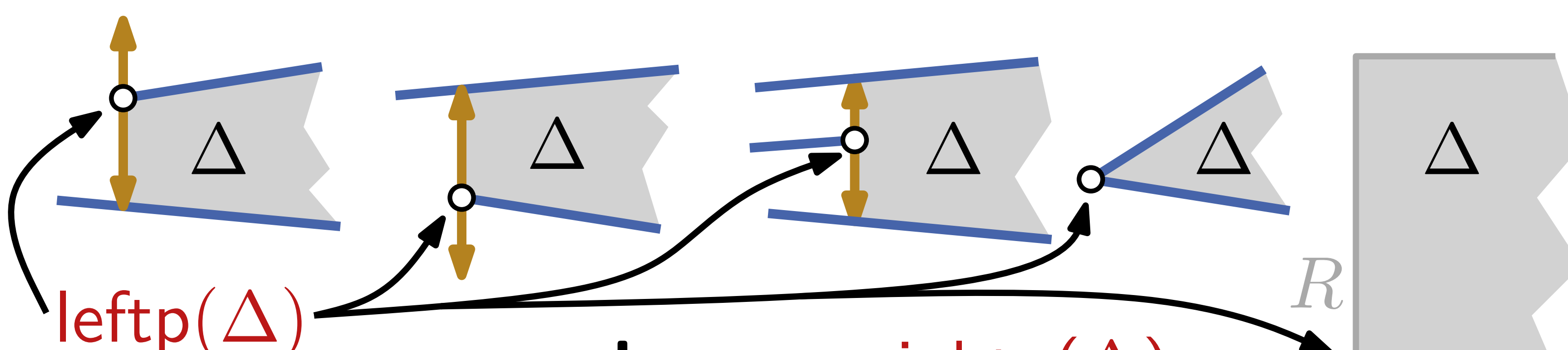
Definition: A *side* of a face of $\mathcal{T}(\mathcal{S})$ is a segment of maximal length contained in face boundary.



Observation: $\mathcal{S}$ in general position $\Rightarrow$ each face Δ of $\mathcal{T}(\mathcal{S})$ has:

- one or two vertical sides
- two non-vertical sides

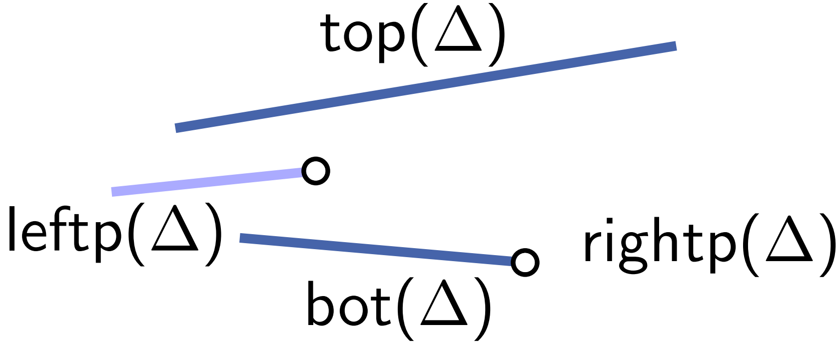
Left side:



analogous: rightp(Δ)

Complexity of the Trapezoidal Map

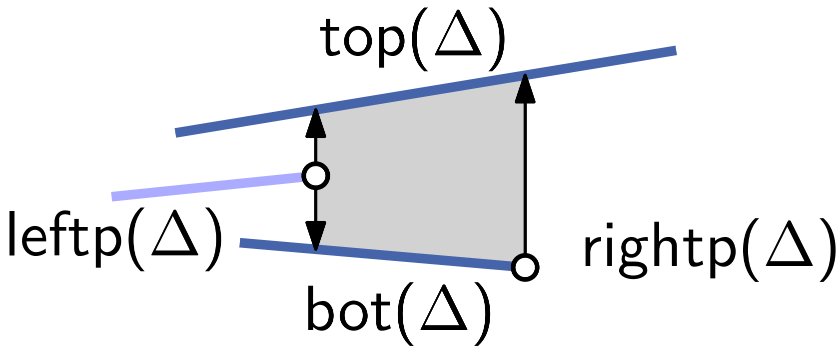
Obs.: A trapezoid Δ is uniquely defined by bot(Δ), top(Δ), leftp(Δ) and rightp(Δ).



Complexity of the Trapezoidal Map

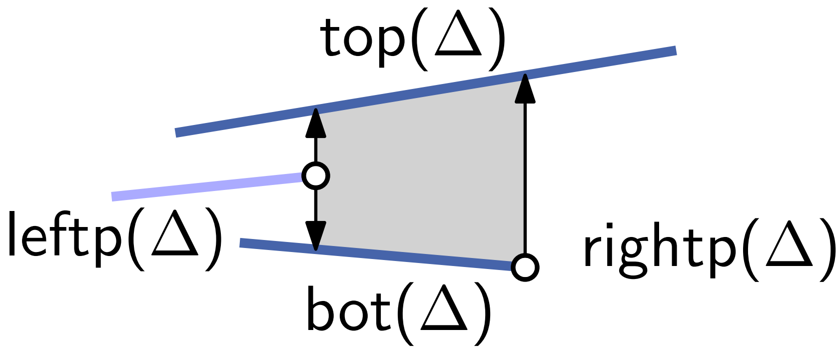
Karlsruhe Institute of Technology

Obs.: A trapezoid Δ is uniquely defined by bot(Δ), top(Δ), leftp(Δ) and rightp(Δ).



Complexity of the Trapezoidal Map

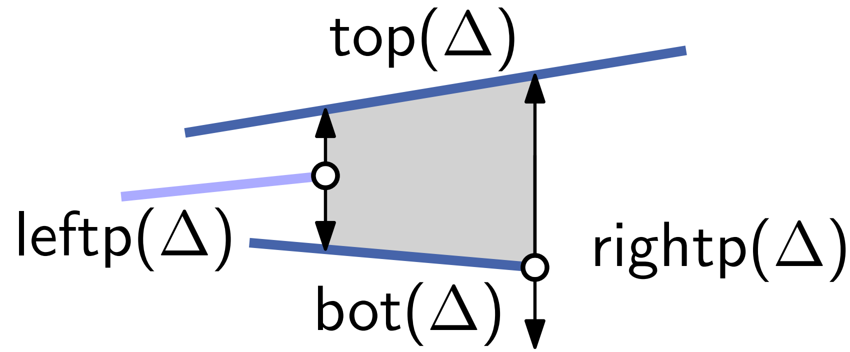
Obs.: A trapezoid Δ is uniquely defined by bot(Δ), top(Δ), leftp(Δ) and rightp(Δ).



Lemma 1: The trapezoidal map $\mathcal{T}(\mathcal{S})$ of a set $\mathcal{S}$ of n segments in general position contains at most vertices and at most trapezoids.

Complexity of the Trapezoidal Map

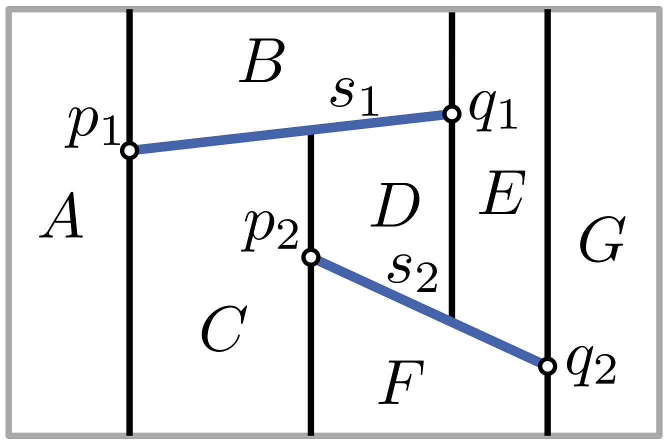
Obs.: A trapezoid Δ is uniquely defined by bot(Δ), top(Δ), leftp(Δ) and rightp(Δ).



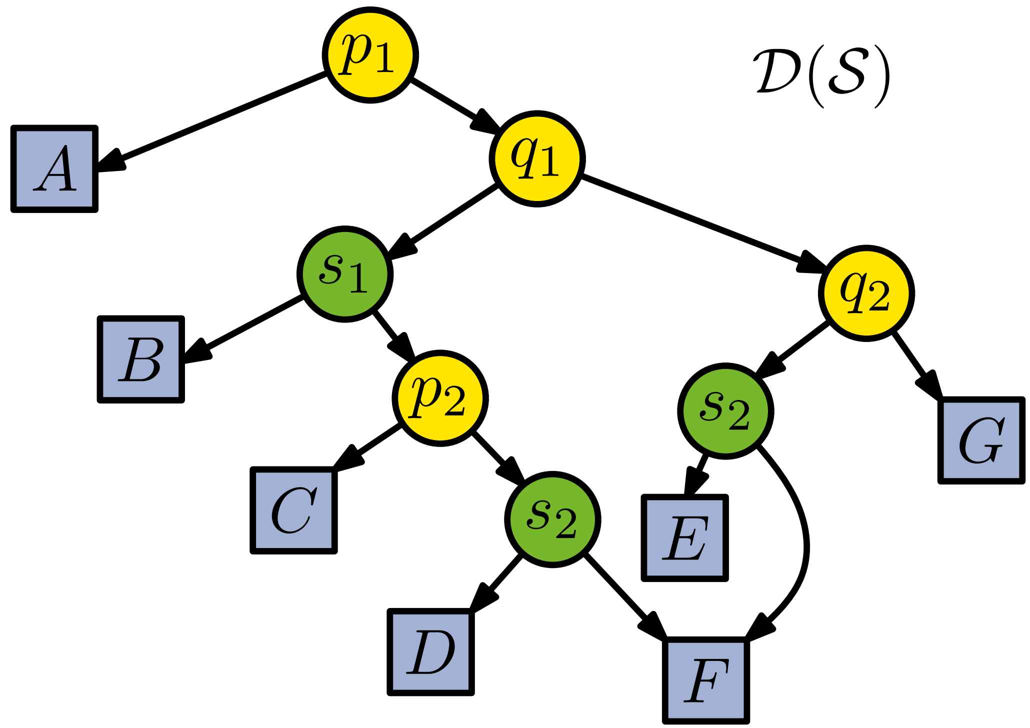
Lemma 1: The trapezoidal map $\mathcal{T}(\mathcal{S})$ of a set $\mathcal{S}$ of n segments in general position contains at most $6n + 4$ vertices and at most $3n + 1$ trapezoids.

Search Structure

Goal: Compute the trapzoidal map $\mathcal{T}(\mathcal{S})$ and simultaneously a data structure $\mathcal{D}(\mathcal{S})$ for point location in $\mathcal{T}(\mathcal{S})$.



$\mathcal{T}(\mathcal{S})$



$\mathcal{D}(\mathcal{S})$ is a DAG with:

- p — x-node for point p tests left/right of p
- s — y-node for segment s tests above/below s
- Δ — leaf node for trapezoid Δ

Incremental Algorithm

TrapezoidalMap($\mathcal{S}$)

Input: set $\mathcal{S} = \{s_1, \dots, s_n\}$ of crossing-free segments
Output: trapezoidal map $\mathcal{T}(\mathcal{S})$ and search structure $\mathcal{D}(\mathcal{S})$
initialize $\mathcal{T}$ and $\mathcal{D}$ for $R =$ BBox($\mathcal{S}$)

for $i \leftarrow 1$ **to** n **do**
  $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$
  $\mathcal{T} \leftarrow \mathcal{T} \setminus H$
  $\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of s_i
  $\mathcal{D} \leftarrow$ replace leaves for H by nodes and leaves for new trapezoids
return $(\mathcal{T}, \mathcal{D})$

Incremental Algorithm

TrapezoidalMap($\mathcal{S}$)

Input: set $\mathcal{S} = \{s_1, \dots, s_n\}$ of crossing-free segments
Output: trapezoidal map $\mathcal{T}(\mathcal{S})$ and search structure $\mathcal{D}(\mathcal{S})$
initialize $\mathcal{T}$ and $\mathcal{D}$ for $R = \text{BBox}(\mathcal{S})$

for $i \leftarrow 1$ **to** n **do**
    $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus H$
    $\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of s_i
    $\mathcal{D} \leftarrow$ replace leaves for H by nodes and leaves for new trapezoids
return $(\mathcal{T}, \mathcal{D})$

Problem: Size of $\mathcal{D}$ and query time depend on insertion order

Incremental Algorithm

TrapezoidalMap($\mathcal{S}$)

Input: set $\mathcal{S} = \{s_1, \dots, s_n\}$ of crossing-free segments
Output: trapezoidal map $\mathcal{T}(\mathcal{S})$ and search structure $\mathcal{D}(\mathcal{S})$
initialize $\mathcal{T}$ and $\mathcal{D}$ for $R = \mathrm{BBox}(\mathcal{S})$
$\mathcal{S} \leftarrow$ **RandomPermutation**($\mathcal{S}$)
for $i \leftarrow 1$ **to** n **do**
    $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus H$
    $\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of s_i
    $\mathcal{D} \leftarrow$ replace leaves for H by nodes and leaves for new trapezoids
return $(\mathcal{T}, \mathcal{D})$

Problem: Size of $\mathcal{D}$ and query time depend on insertion order

Solution: Randomization!

Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

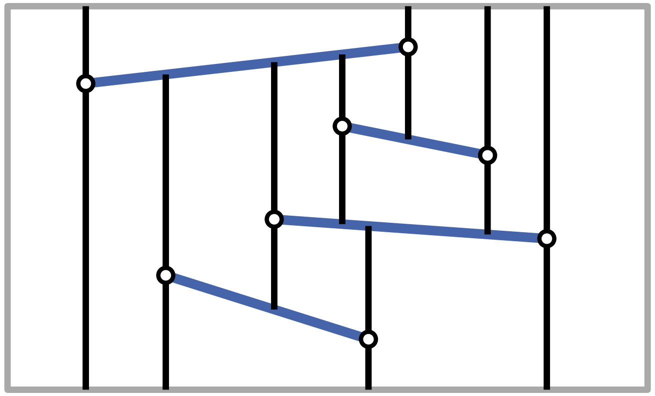
Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

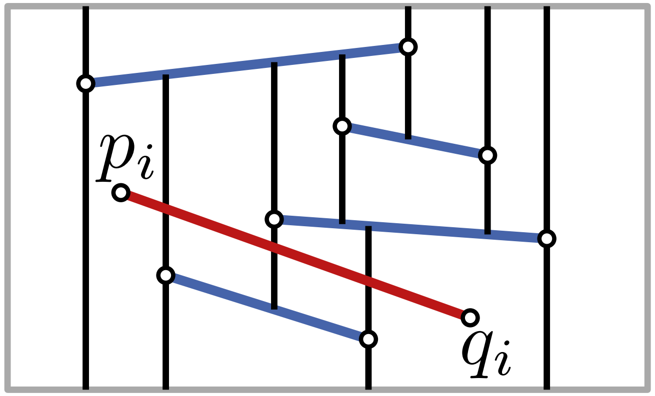


Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

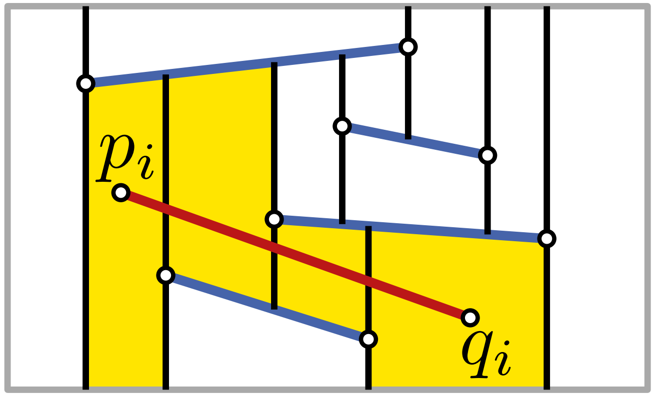


Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

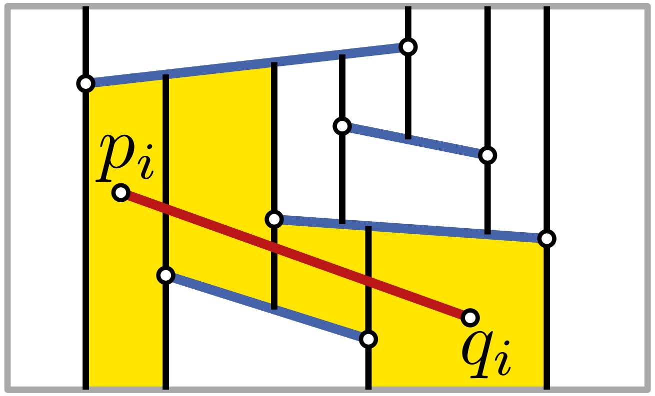


Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$



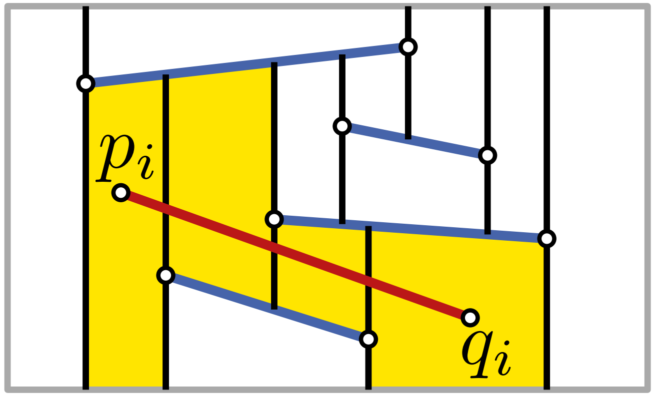
Task: How do you find the set H of trapezoids from left to right?

Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$



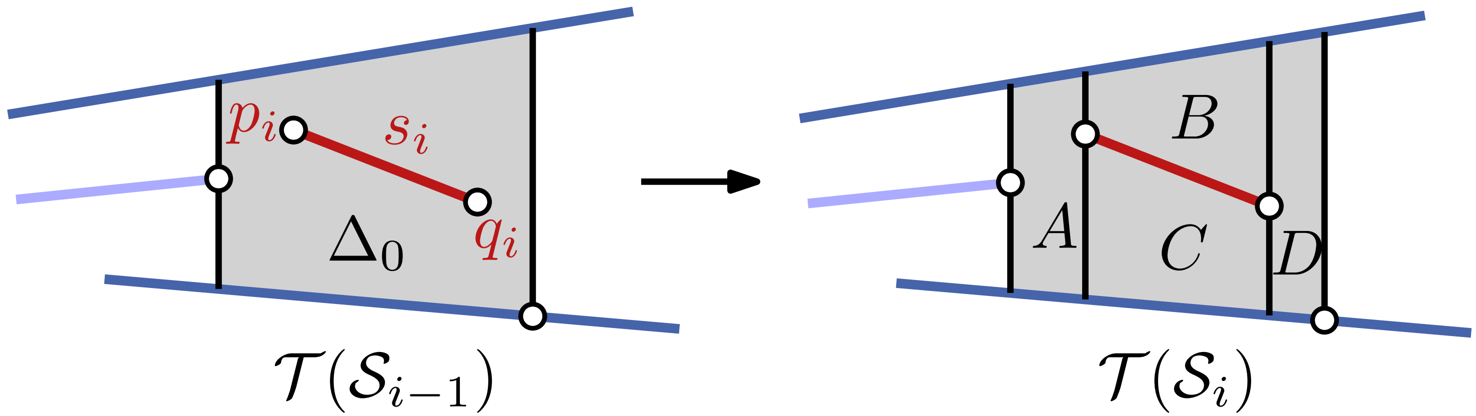
Task: How do you find the set H of trapezoids from left to right?

$\Delta_0 \leftarrow$ FindTrapezoid$(p_i, \mathcal{D})$; $j \leftarrow 0$
while right endpoint q_i right of rightp(Δ_j) **do**
    if rightp(Δ_j) above s_i **then**
        $\Delta_{j+1} \leftarrow$ lower right neighbor of Δ_j
    else
        $\Delta_{j+1} \leftarrow$ upper right neighbor of Δ_j
    $j \leftarrow j + 1$
return $\Delta_0, \ldots, \Delta_j$

Updating $\mathcal{T}(\mathcal{S})$ and $\mathcal{D}(\mathcal{S})$

Step 2: Update $\mathcal{T}$ and $\mathcal{D}$

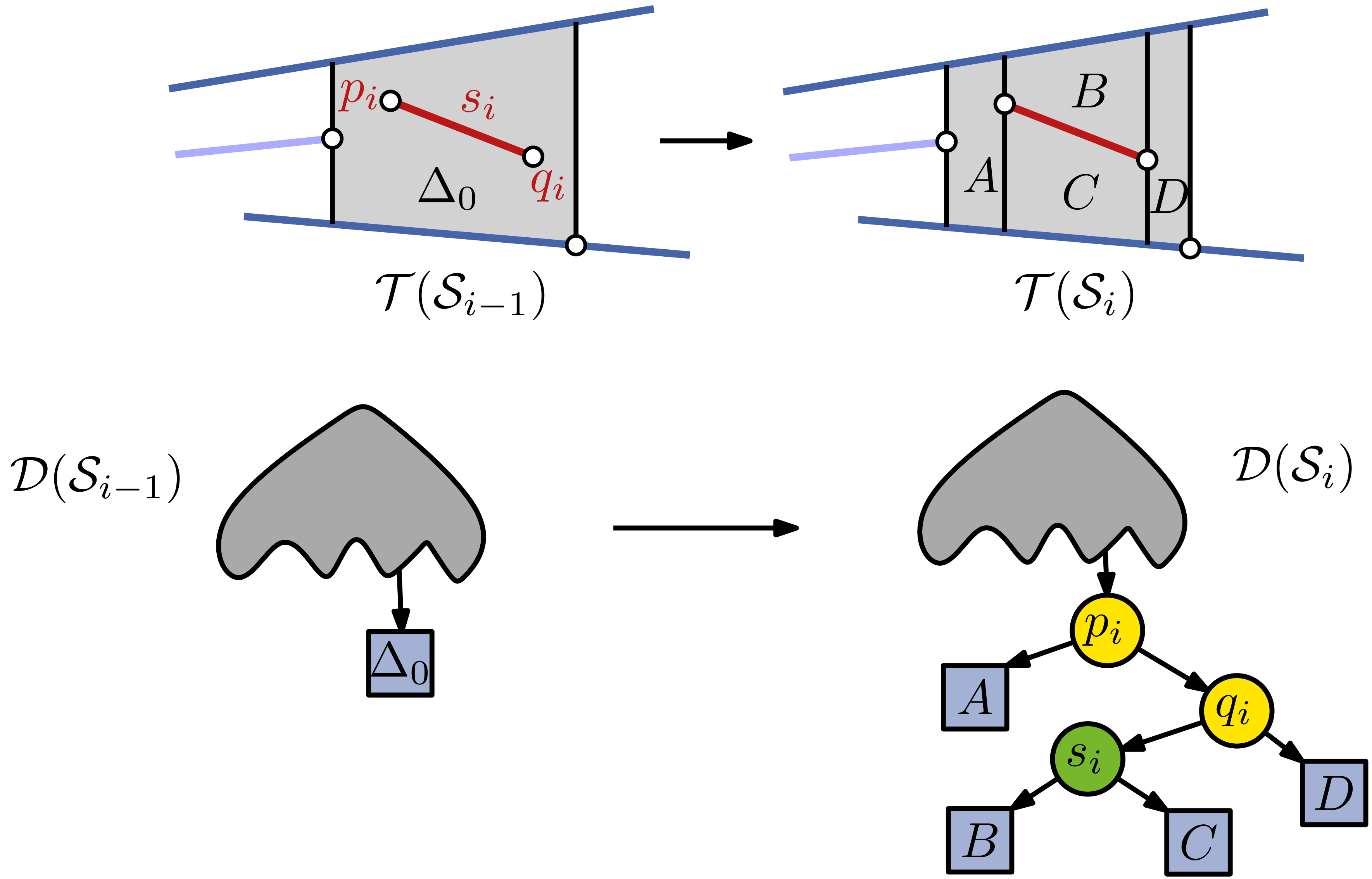
- Case 1: $s_i \subset \Delta_0$



Updating $\mathcal{T}(\mathcal{S})$ and $\mathcal{D}(\mathcal{S})$

Step 2: Update $\mathcal{T}$ and $\mathcal{D}$

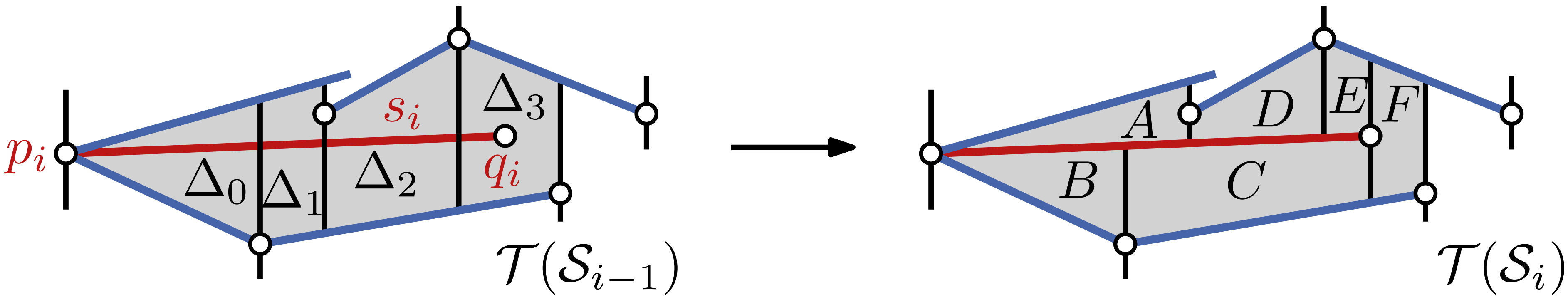
- Case 1: $s_i \subset \Delta_0$



Updating $\mathcal{T}(\mathcal{S})$ and $\mathcal{D}(\mathcal{S})$

Step 2: Update $\mathcal{T}$ and $\mathcal{D}$

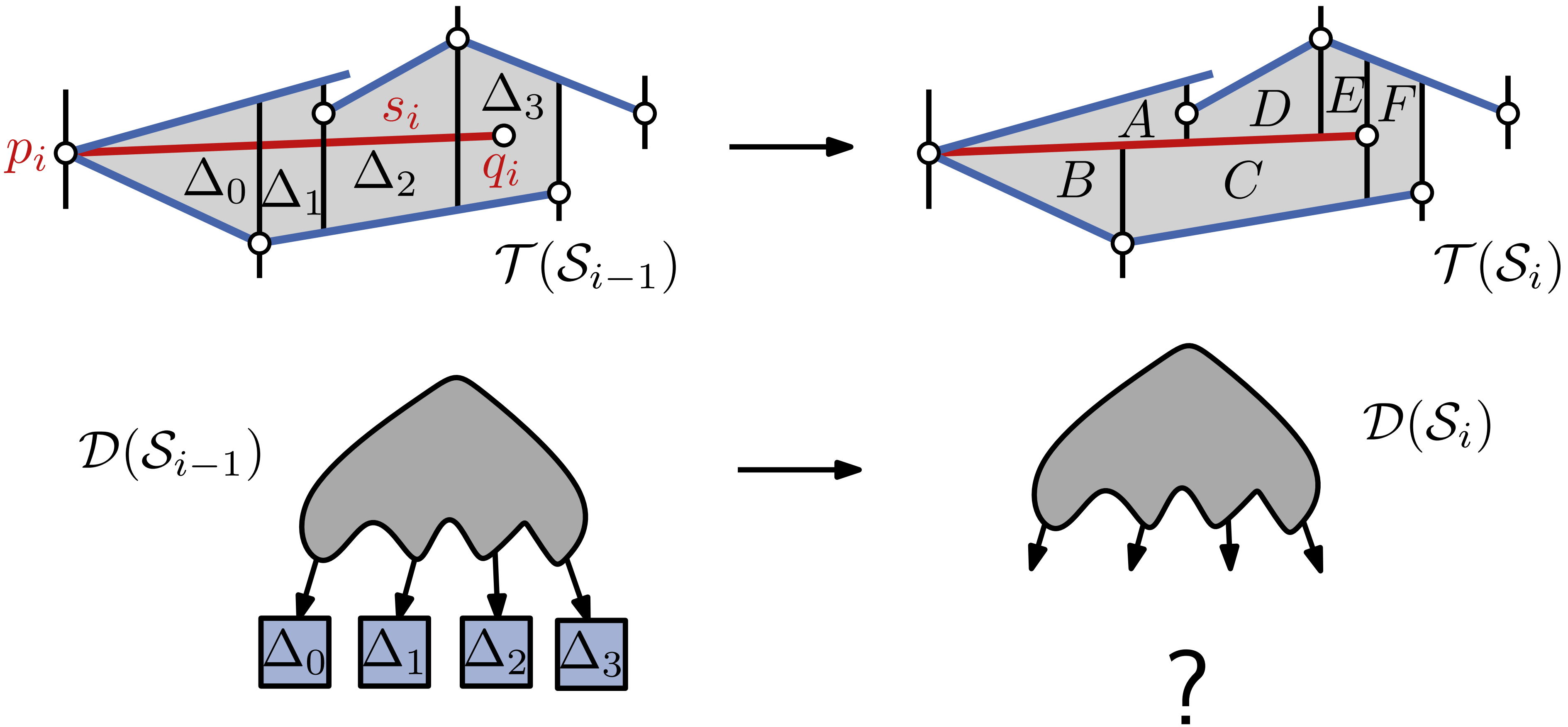
- Case 1: $s_i \subset \Delta_0$
- Case 2: $|\mathcal{T} \cap s_i| \geq 2$



Updating $\mathcal{T}(\mathcal{S})$ and $\mathcal{D}(\mathcal{S})$

Step 2: Update $\mathcal{T}$ and $\mathcal{D}$

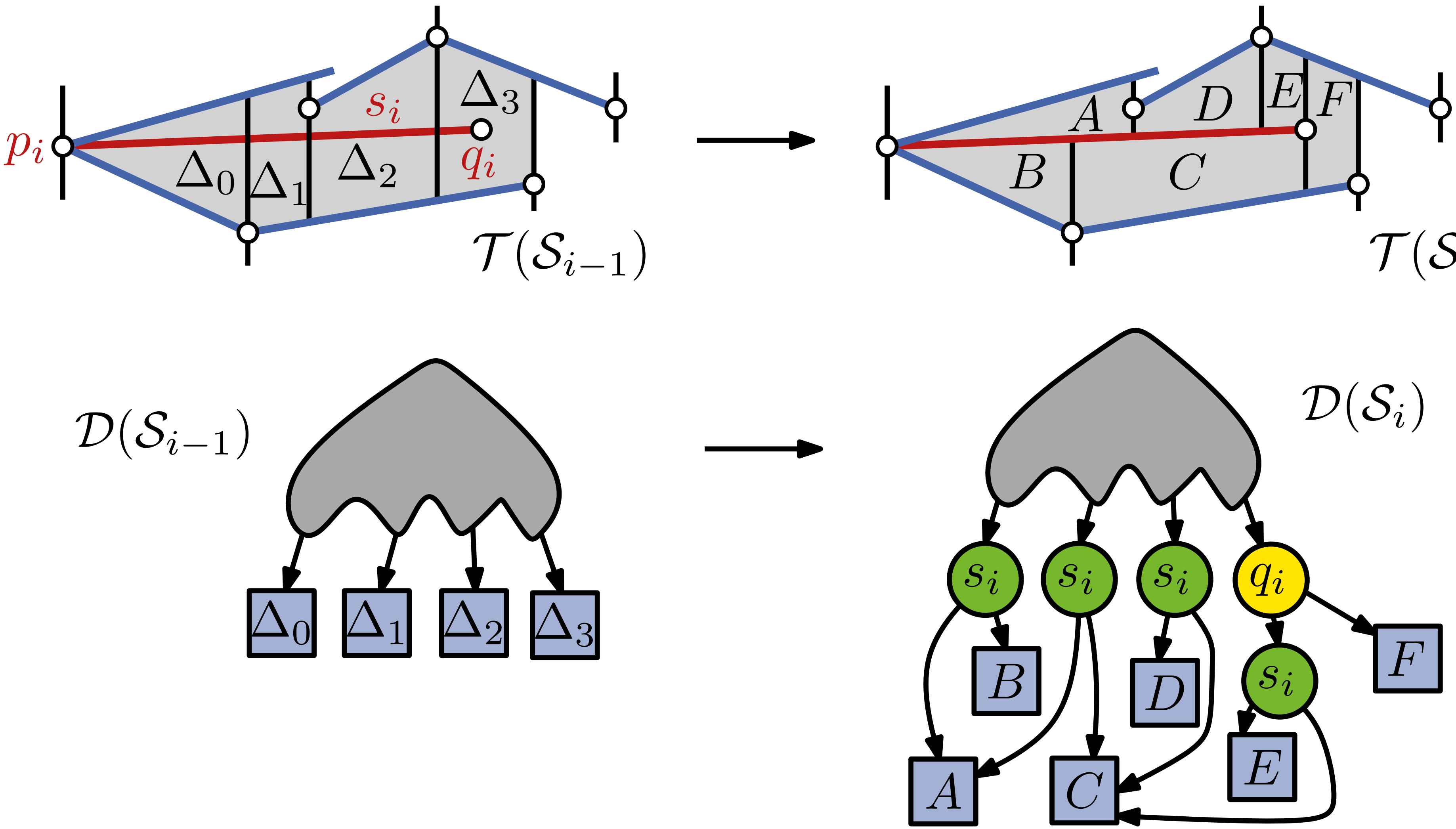
- Case 1: $s_i \subset \Delta_0$
- Case 2: $|\mathcal{T} \cap s_i| \geq 2$



Updating $\mathcal{T}(\mathcal{S})$ and $\mathcal{D}(\mathcal{S})$

Step 2: Update $\mathcal{T}$ and $\mathcal{D}$

- Case 1: $s_i \subset \Delta_0$
- Case 2: $|\mathcal{T} \cap s_i| \geq 2$



Analysis

Thm 1: The algorithm computes the trapezoidal map $\mathcal{T}(\mathcal{S})$ and the search structure $\mathcal{D}$ for a set $\mathcal{S}$ of n segments in *expected* $O(n \log n)$ time. The *expected* size of $\mathcal{D}$ is $O(n)$ and the *expected* query time is $O(\log n)$.

Analysis

Thm 1: The algorithm computes the trapezoidal map $\mathcal{T}(\mathcal{S})$ and the search structure $\mathcal{D}$ for a set $\mathcal{S}$ of n segments in *expected* $O(n \log n)$ time. The *expected* size of $\mathcal{D}$ is $O(n)$ and the *expected* query time is $O(\log n)$.

Observations:

- worst case: size of $\mathcal{D}$ is quadratic and query time is linear
- hope: that happens rarely!
- consider expected time and size over all $n!$ permutations of $\mathcal{S}$
- the theorem holds independently of the input set $\mathcal{S}$

Analysis

Thm 1: The algorithm computes the trapezoidal map $\mathcal{T}(\mathcal{S})$ and the search structure $\mathcal{D}$ for a set $\mathcal{S}$ of n segments in *expected* $O(n \log n)$ time. The *expected* size of $\mathcal{D}$ is $O(n)$ and the *expected* query time is $O(\log n)$.

Observations:

- worst case: size of $\mathcal{D}$ is quadratic and query time is linear
- hope: that happens rarely!
- consider expected time and size over all $n!$ permutations of $\mathcal{S}$
- the theorem holds independently of the input set $\mathcal{S}$

Proof:

- define random variables and consider their expected values
- perform *backward analysis*

$\rightarrow$ details on blackboard

Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Lemma 2: Let $\mathcal{S}$ be a set of n crossing-free segments, let q be a query point and let $\lambda > 0$. Then
Pr[search path for q longer than $3\lambda \ln(n+1)$]
$\leq 1/(n+1)^{\lambda \ln 1.25 - 1}$.

No proof. (or see Chapter 6.4)

Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Lemma 2: Let $\mathcal{S}$ be a set of n crossing-free segments, let q be a query point and let $\lambda > 0$. Then
Pr[search path for q longer than $3\lambda \ln(n+1)$]
$\leq 1/(n+1)^{\lambda \ln 1.25 - 1}$.

Lemma 3: Let $\mathcal{S}$ be a set of n crossing-free segments and $\lambda > 0$. Then
Pr[max. search path in $\mathcal{D}$ longer than $3\lambda \ln(n+1)$]
$\leq 2/(n+1)^{\lambda \ln 1.25 - 3}$.

Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Lemma 2: Let $\mathcal{S}$ be a set of n crossing-free segments, let q be a query point and let $\lambda > 0$. Then
Pr[search path for q longer than $3\lambda \ln(n+1)$]
$\leq 1/(n+1)^{\lambda \ln 1.25 - 1}$.

Lemma 3: Let $\mathcal{S}$ be a set of n crossing-free segments and $\lambda > 0$. Then
Pr[max. search path in $\mathcal{D}$ longer than $3\lambda \ln(n+1)$]
$\leq 2/(n+1)^{\lambda \ln 1.25 - 3}$.

Thm 2: Let $\mathcal{S}$ be a subdivision of the plane with n edges. There is a search structure for point location within $\mathcal{S}$ that has $O(n)$ space and $O(\log n)$ query time.

Degenerate Inputs

Two assumptions:

- no two segment endpoints have the same x-coordinates
- always unique answers (left/right) on the search path

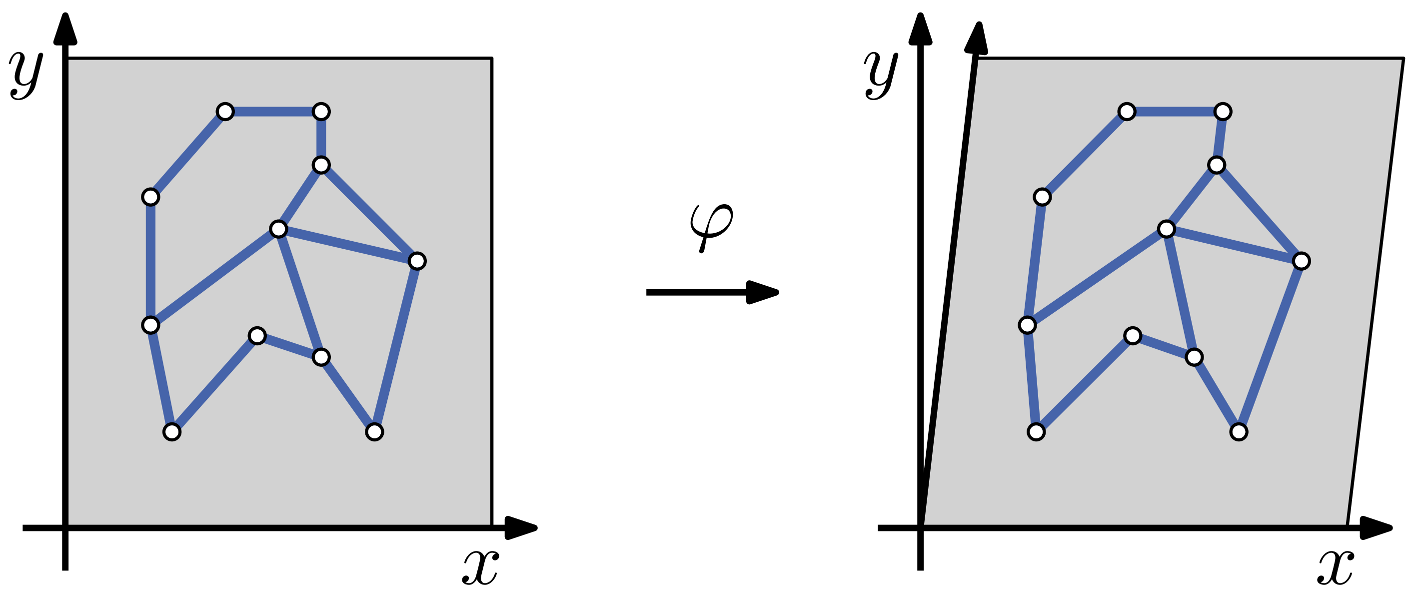
Degenerate Inputs

Two assumptions:

- no two segment endpoints have the same x-coordinates
- always unique answers (left/right) on the search path

solution: symbolic shear transformation

$$\varphi : (x, y) \mapsto (x + \varepsilon y, y)$$



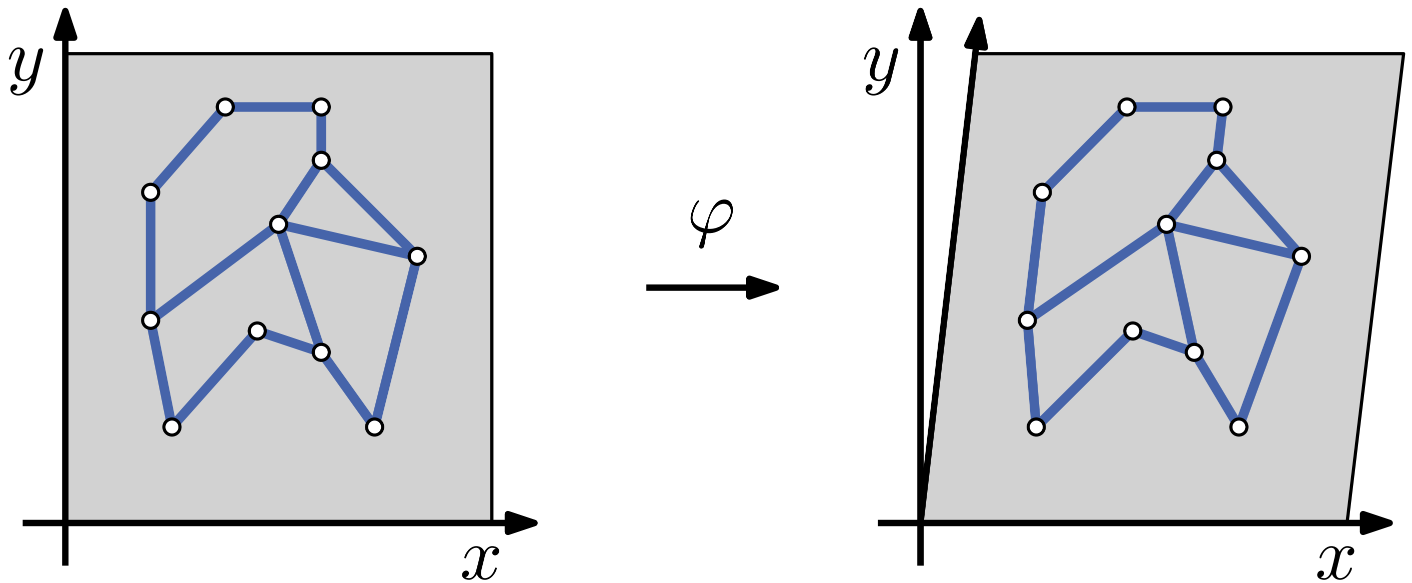
Degenerate Inputs

Two assumptions:

- no two segment endpoints have the same x-coordinates
- always unique answers (left/right) on the search path

solution: symbolic shear transformation

$$\varphi : (x, y) \mapsto (x + \varepsilon y, y)$$



Here $\varepsilon > 0$ is chosen such that the x-order $<$ of the points does not change.

Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map φ maintains point–line relations

Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map φ maintains point–line relations
- Run algorithm for $\varphi\mathcal{S} = \{\varphi s \mid s \in \mathcal{S}\}$ and φp.

Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map φ maintains point–line relations
- Run algorithm for $\varphi\mathcal{S} = \{\varphi s \mid s \in \mathcal{S}\}$ and φp.
- Two basic operations for constructing $\mathcal{T}$ and $\mathcal{D}$:
  1. is q left or right of the vertical line through p?
  2. is q above or below the segment s?

Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map φ maintains point–line relations
- Run algorithm for $\varphi\mathcal{S} = \{\varphi s \mid s \in \mathcal{S}\}$ and φp.
- Two basic operations for constructing $\mathcal{T}$ and $\mathcal{D}$:
  1. is q left or right of the vertical line through p?
  2. is q above or below the segment s?
- Locating a point q in $\mathcal{T}(\mathcal{S})$ works by locating φq in $\mathcal{T}(\varphi\mathcal{S})$.

$\rightarrow$ see Chapter 6.3 in [De Berg et al. 2008]

Discussion

Are there similar methods for higher dimensions?

Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink '04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.

Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink '04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.

Are there dynamic data structures that allow insertions and deletions?

Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink '04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.

Are there dynamic data structures that allow insertions and deletions?

Dynamic data structures for point location are well known, see the survey by [Chiang, Tamassia '92]. A more recent example by [Arge et al. '06] needs $O(n)$ space, $O(\log n)$ query time and $O(\log n)$ update time (insertions).