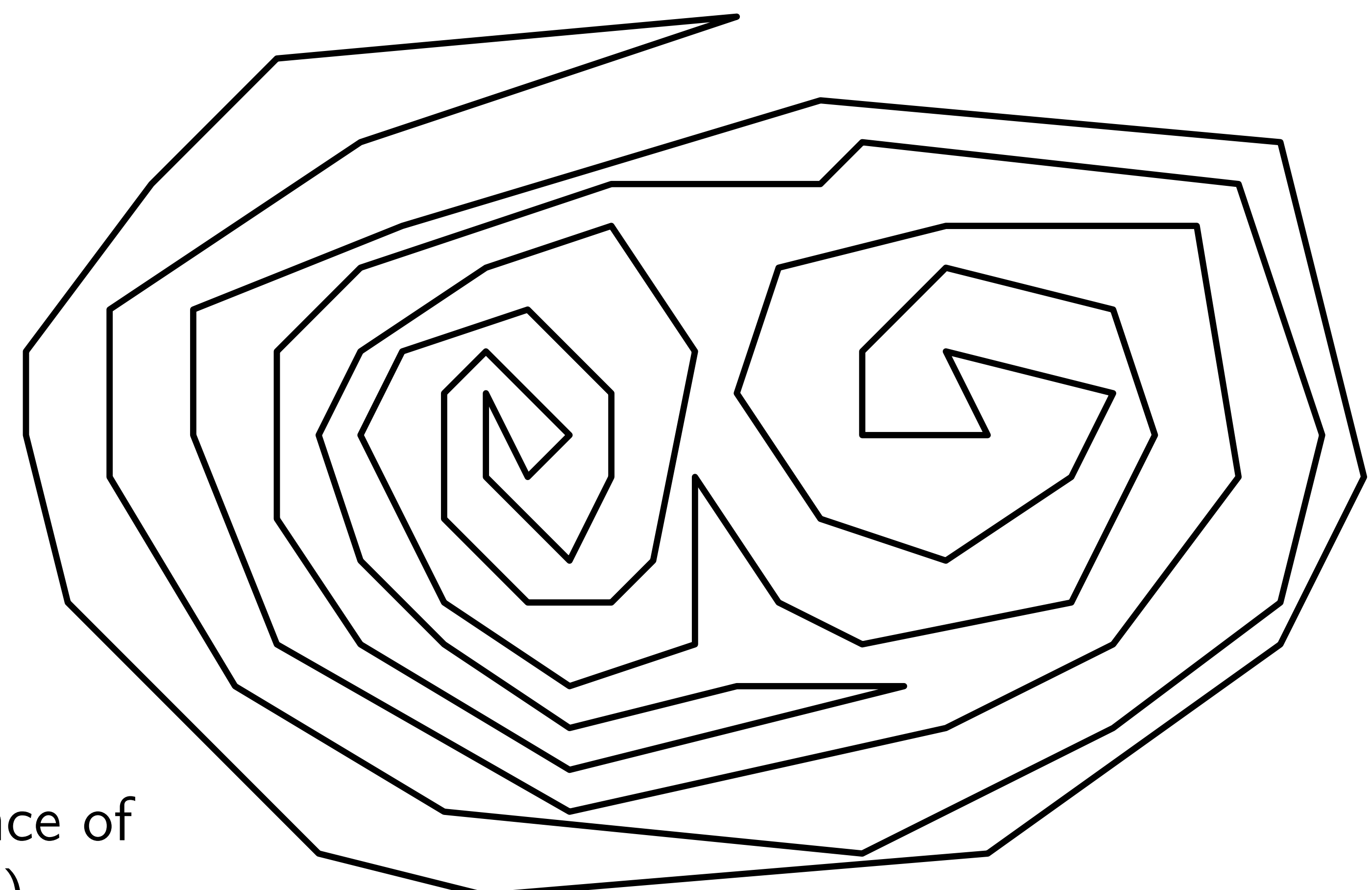
Planar Point Location

Preprocess a given polygon P so that for every query point q it can be determined quickly whether q is inside P or not.

Planar Point Location

Preprocess a given polygon P so that for every query point q it can be determined quickly whether q is inside P or not.



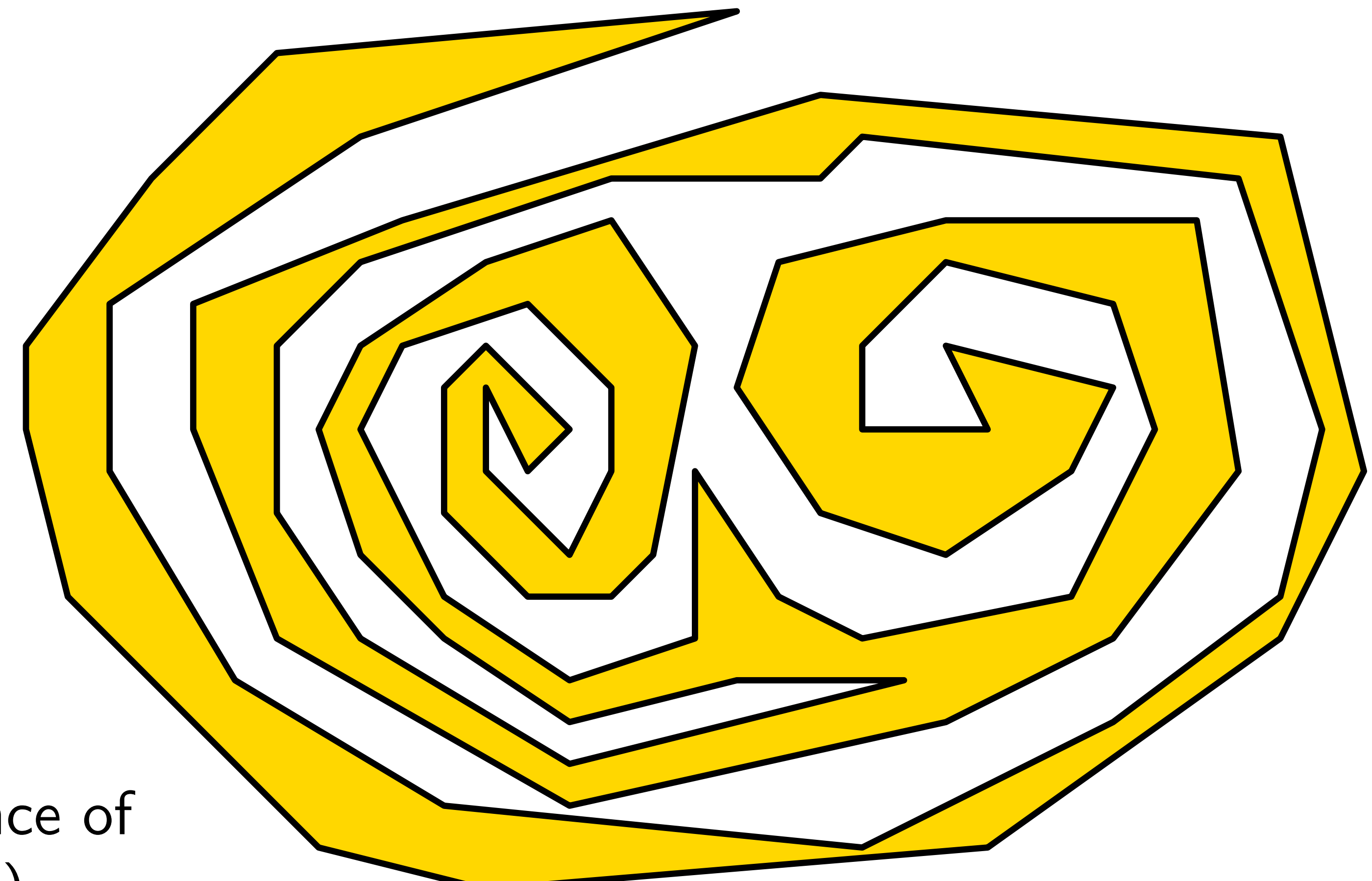
q given by its coordinates

P given by circular sequence of its corners (by coordinates)

SIC Saarland Informatics Campus

Planar Point Location

Preprocess a given polygon P so that for every query point q it can be determined quickly whether q is inside P or not.

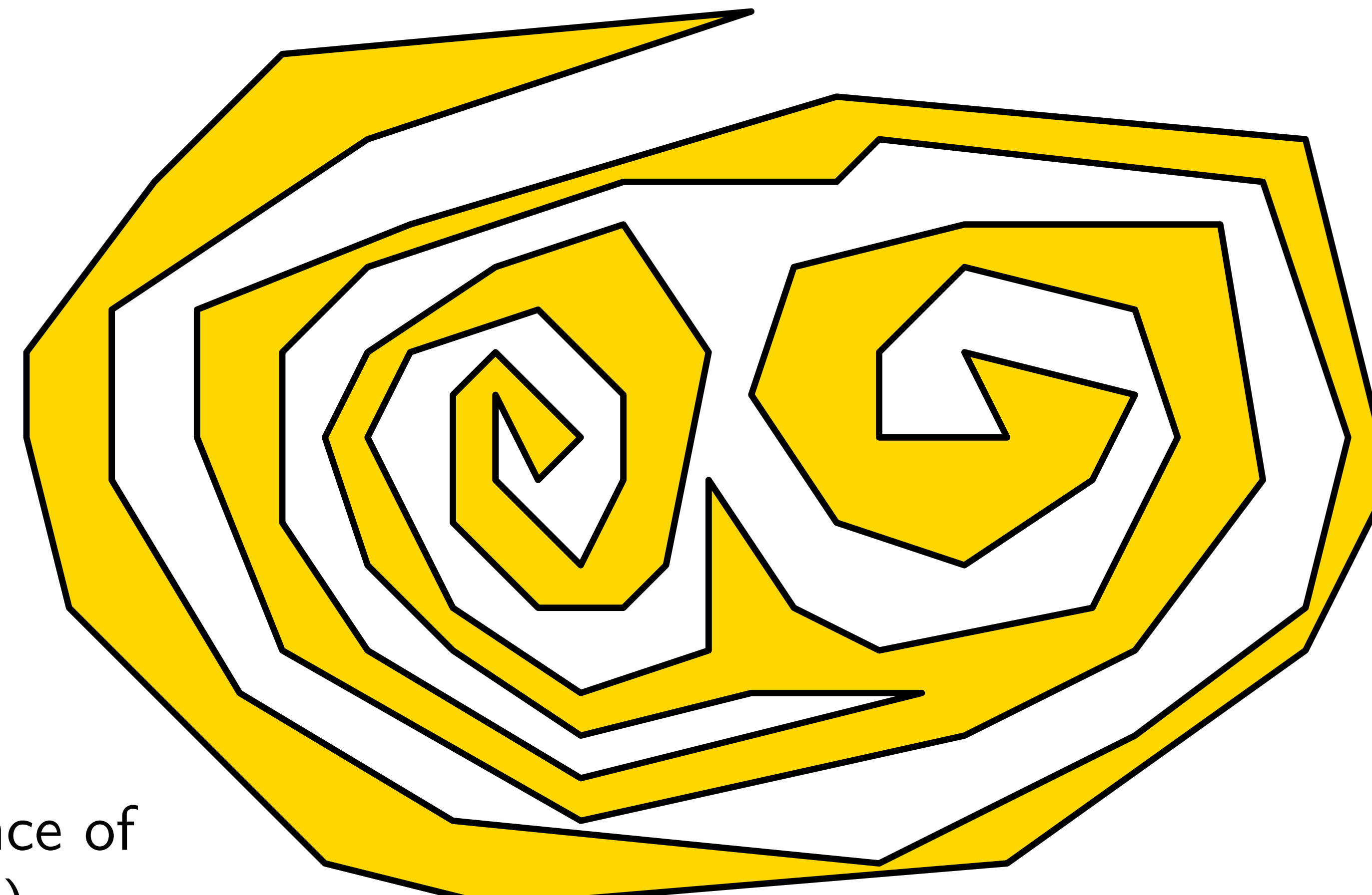


q given by its coordinates

P given by circular sequence of its corners (by coordinates)

Point in Polygon Test

Preprocess a given polygon P so that for every query point q it can be determined quickly whether q is inside P or not.

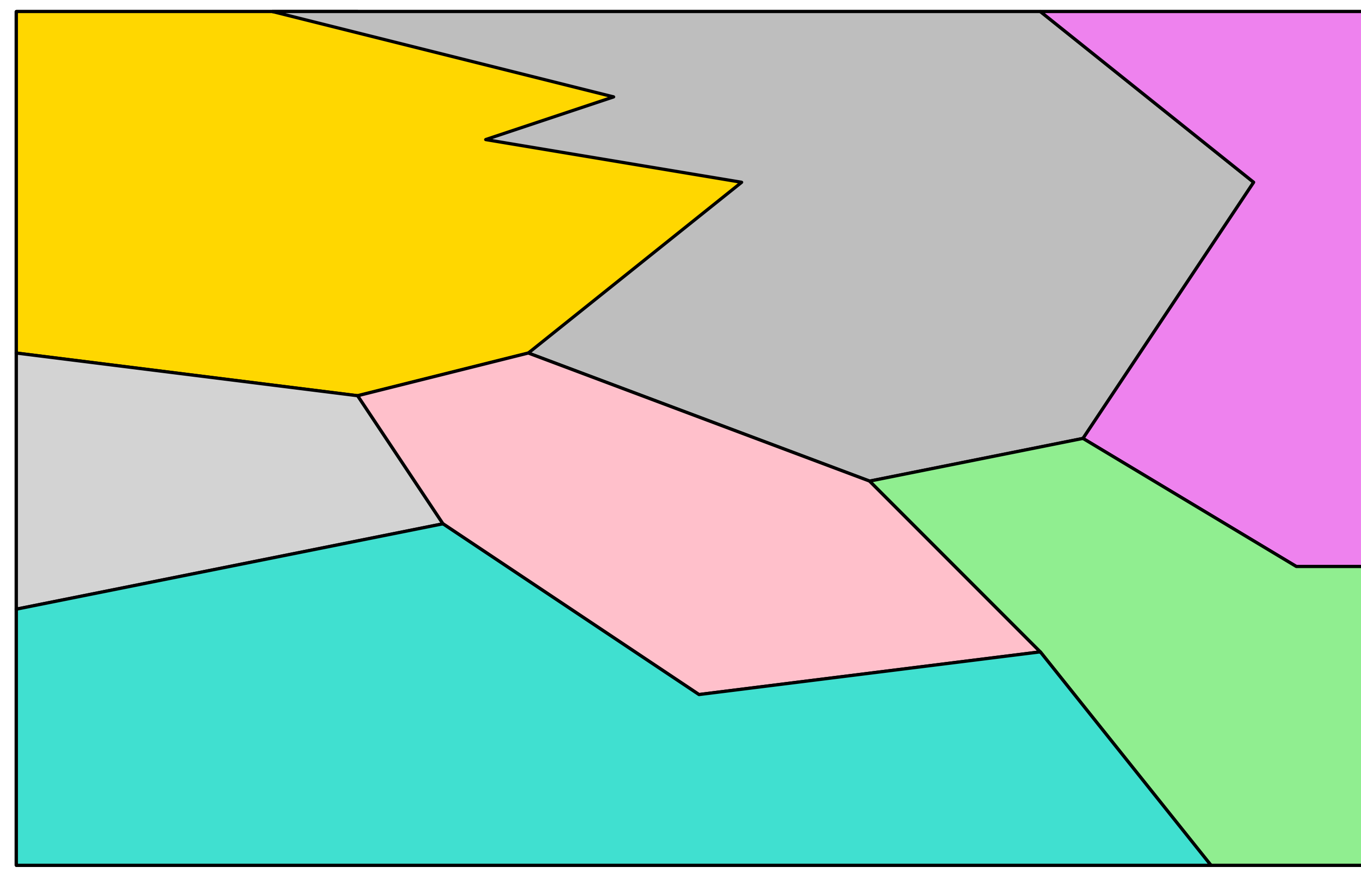


q given by its coordinates

P given by circular sequence of its corners (by coordinates)

Planar Point Location

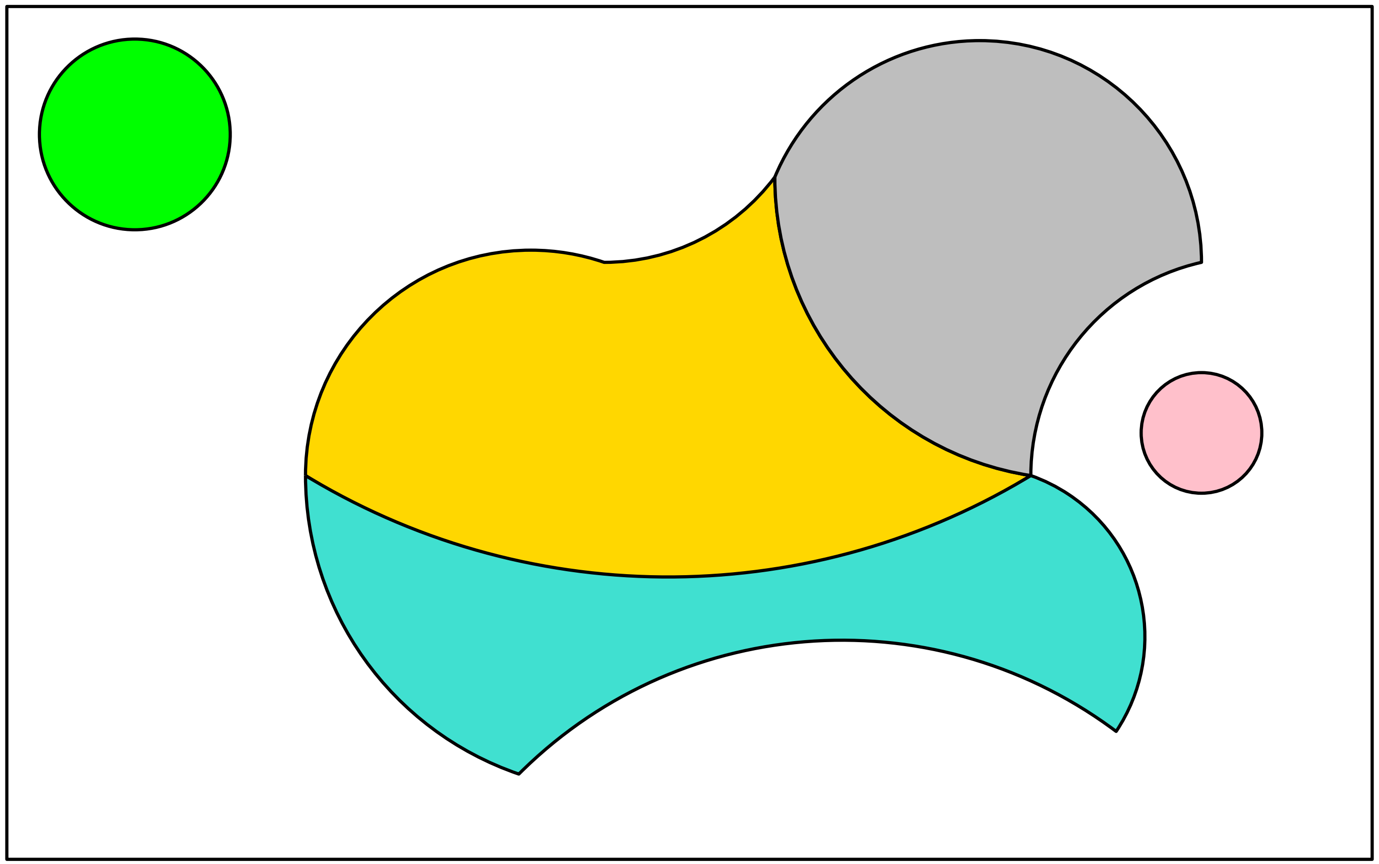
Preprocess a given partition of the plane (or a bounding box) so that for every query point q it can be determined quickly which region of the partition contains q.



Saarland Informatics Campus

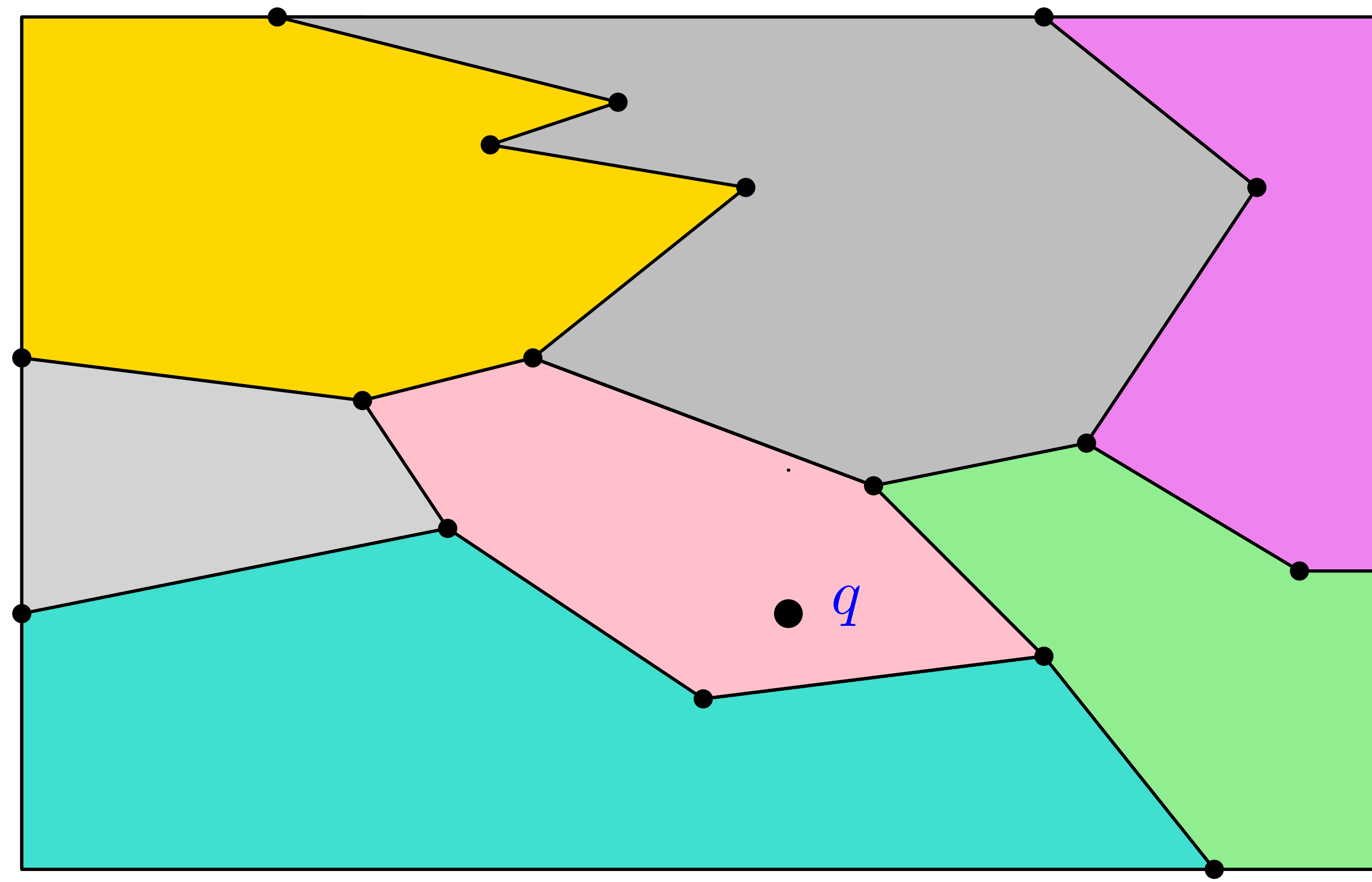
Planar Point Location

Preprocess a given partition of the plane (or a bounding box) so that for every query point q it can be determined quickly which region of the partition contains q.



Vertical Ray Shooting

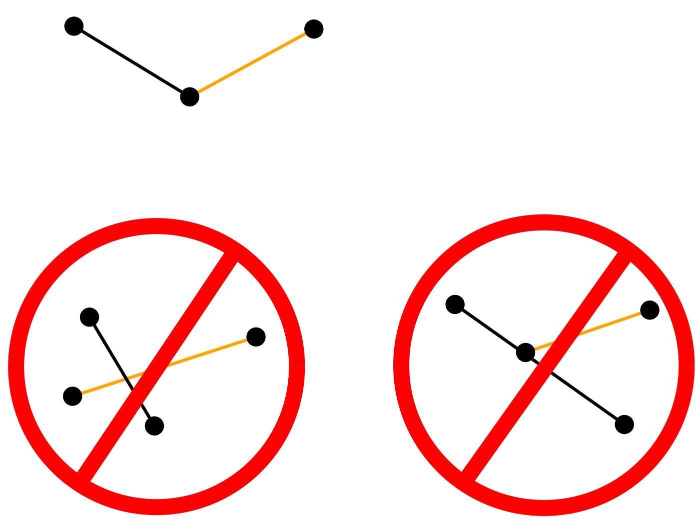
Preprocess a given set S of non-crossing segments in the plane (or a bounding box) so that for every query point q it can be determined quickly which segment of S lies immediately above(below) q.

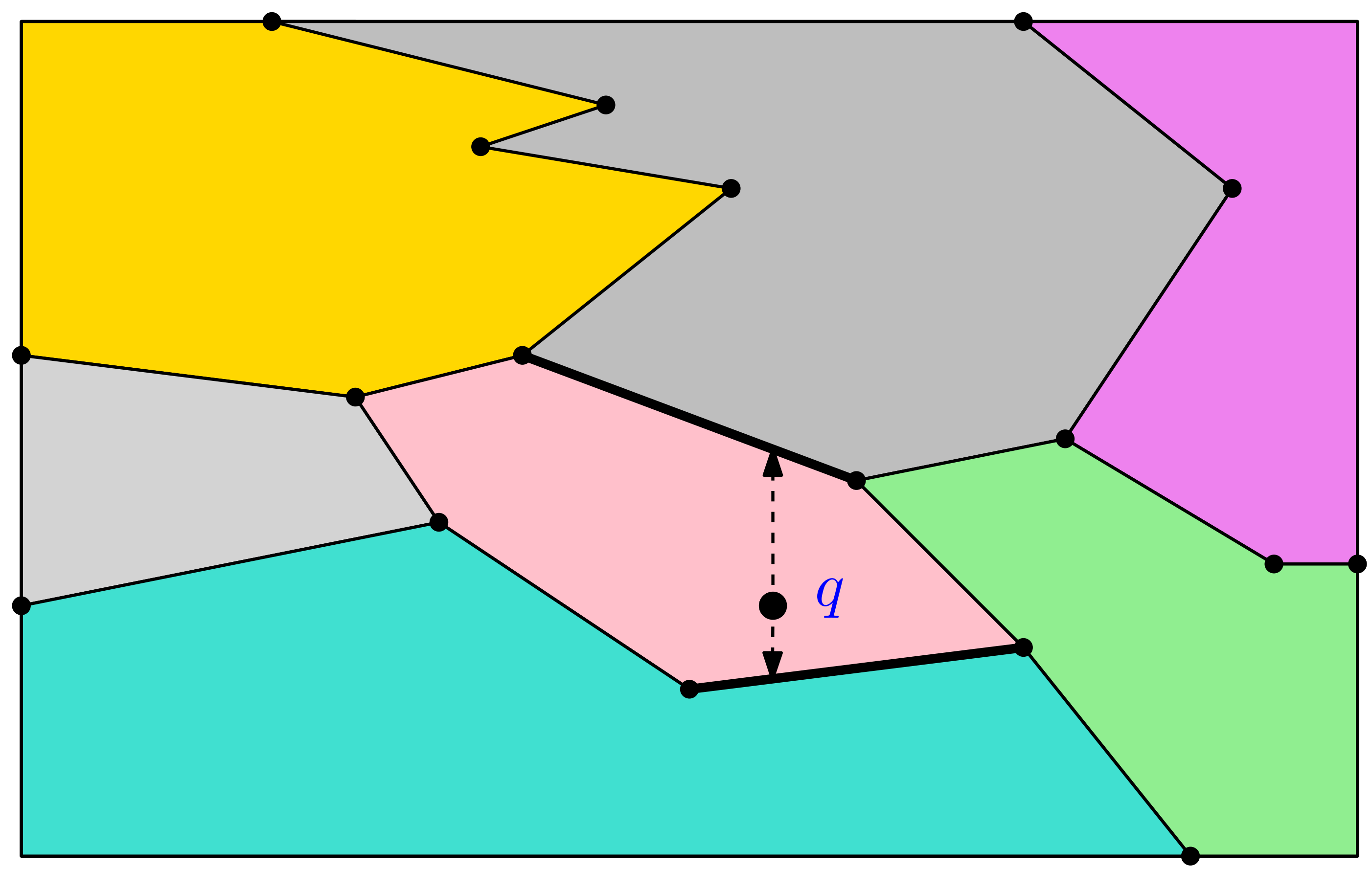


Vertical Ray Shooting

Preprocess a given set S of non-crossing segments in the plane (or a bounding box) so that for every query point q it can be determined quickly which segment of S lies immediately above(below) q.

if they intersect, then they intersect in a common endpoint

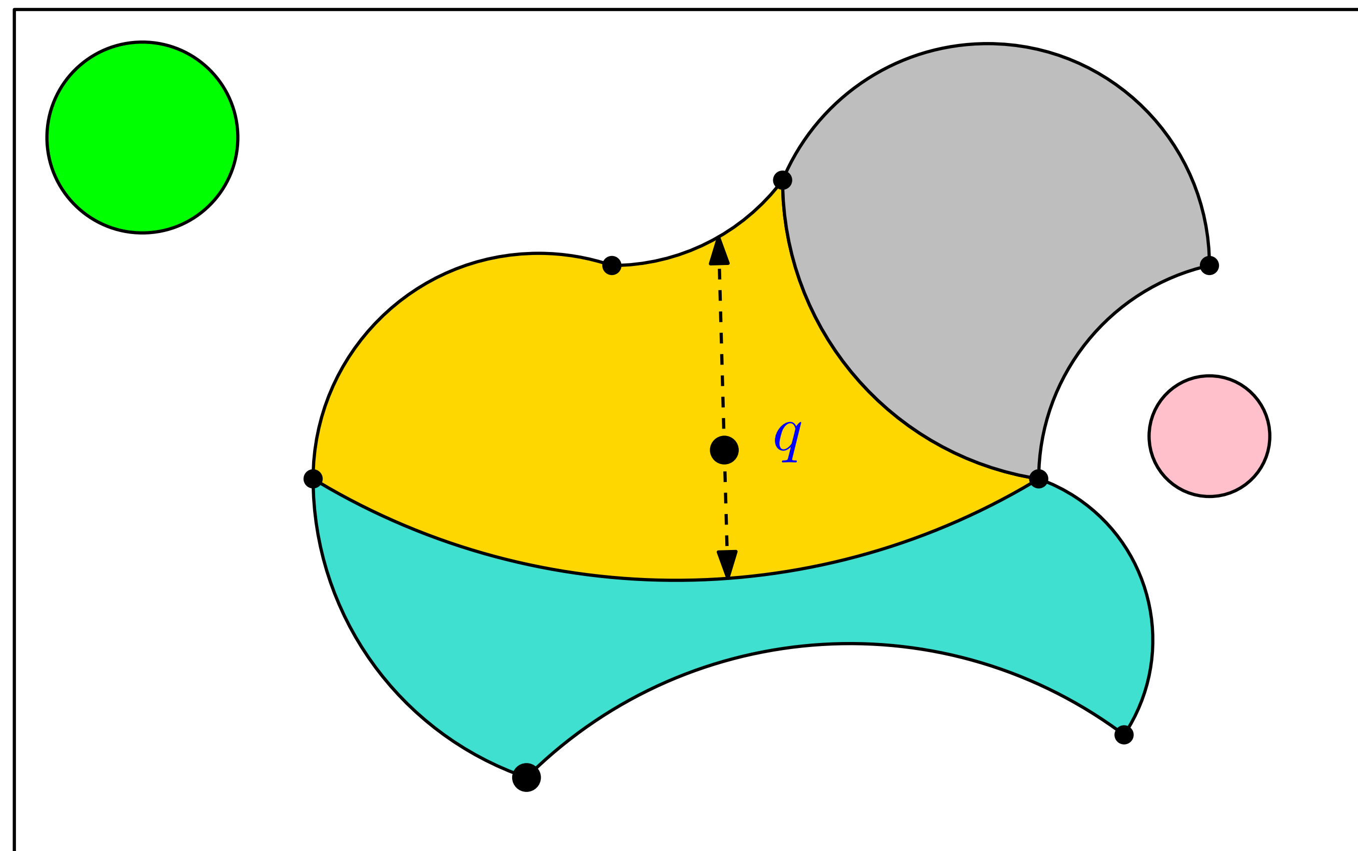




Saarland Informatics Campus

Vertical Ray Shooting

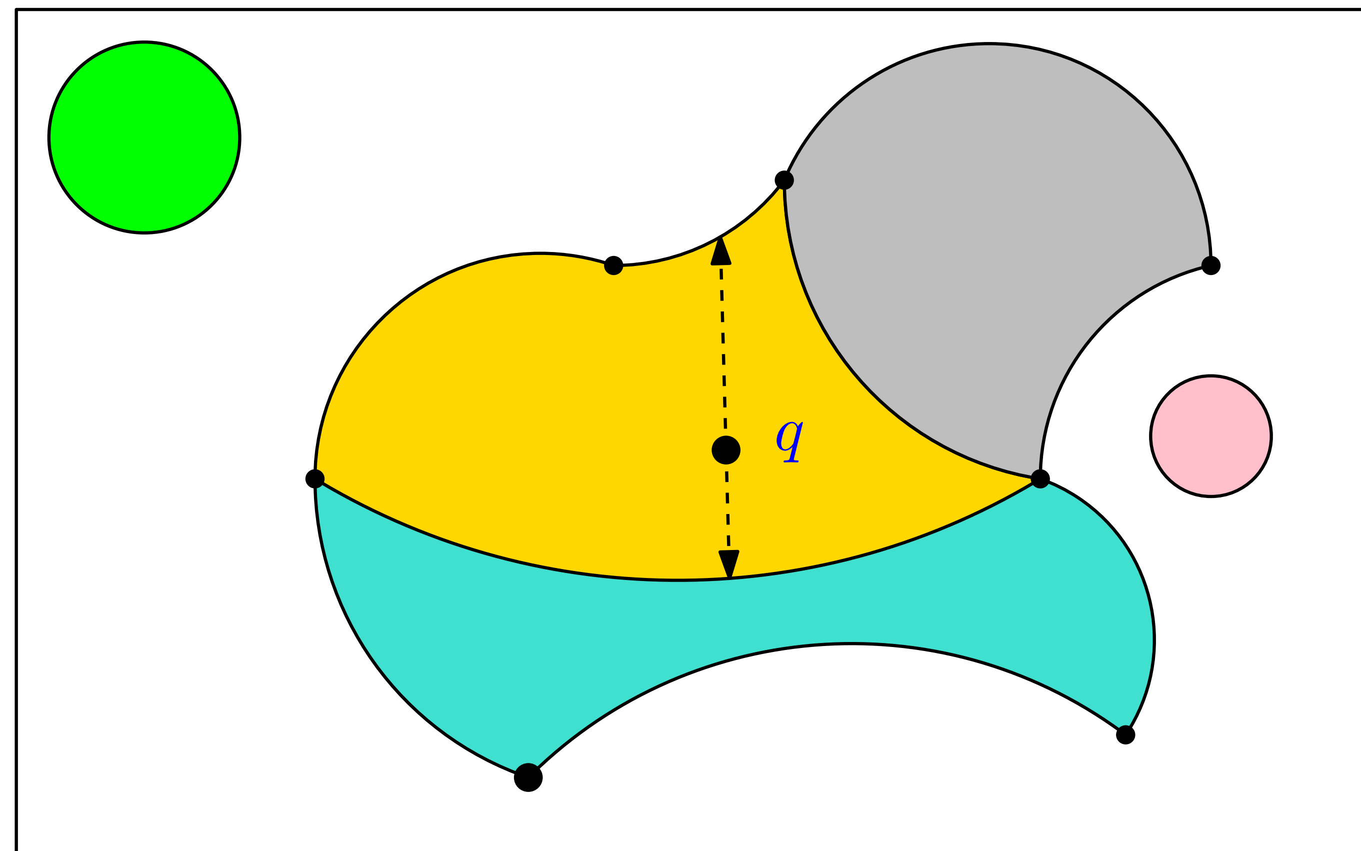
Preprocess a given set S of non-crossing curves in the plane (or a bounding box) so that for every query point q it can be determined quickly which curve of S lies immediately above(below) q.



SIC Saarland Informatics Campus

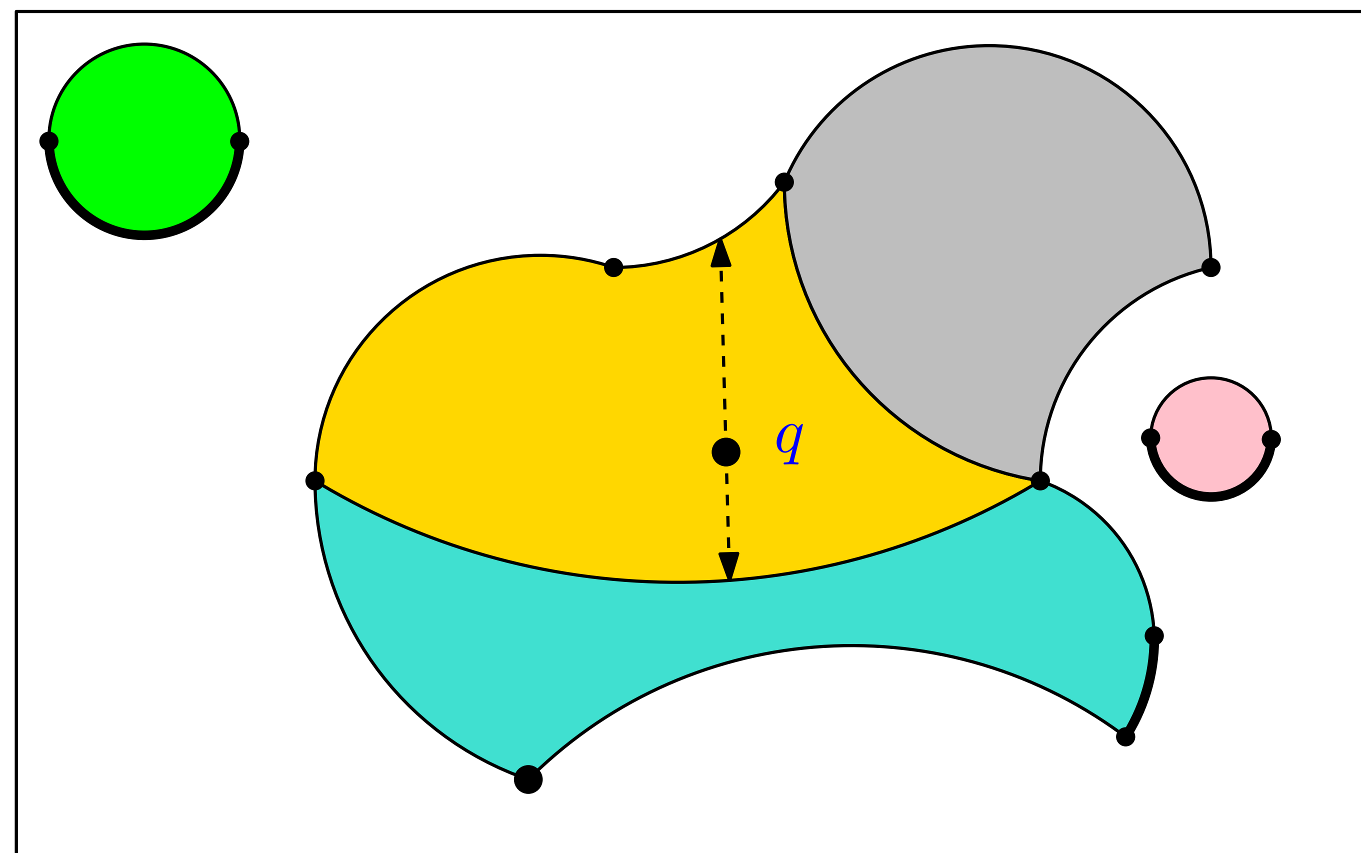
Vertical Ray Shooting

Preprocess a given set S of non-crossing x-monotone curves in the plane (or a bounding box) so that for every query point q it can be determined quickly which curve of S lies immediately above(below) q.



Vertical Ray Shooting

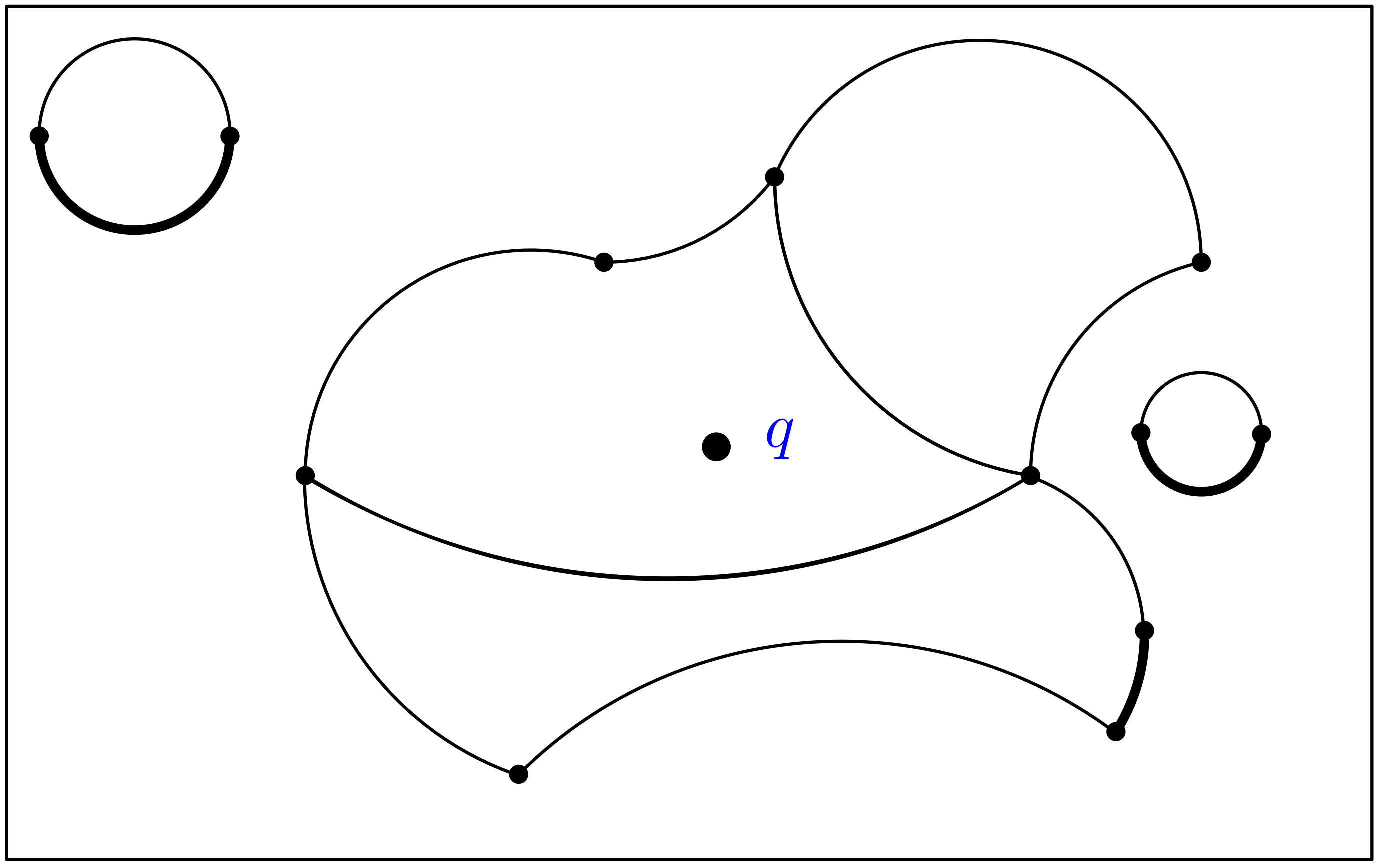
Preprocess a given set S of non-crossing x-monotone curves in the plane (or a bounding box) so that for every query point q it can be determined quickly which curve of S lies immediately above(below) q.



Vertical Ray Shooting

Preprocess a given set S of non-crossing x-monotone curves in the plane (or a bounding box) so that for every query point q it can be determined quickly which curve of S lies immediately above(below) q.

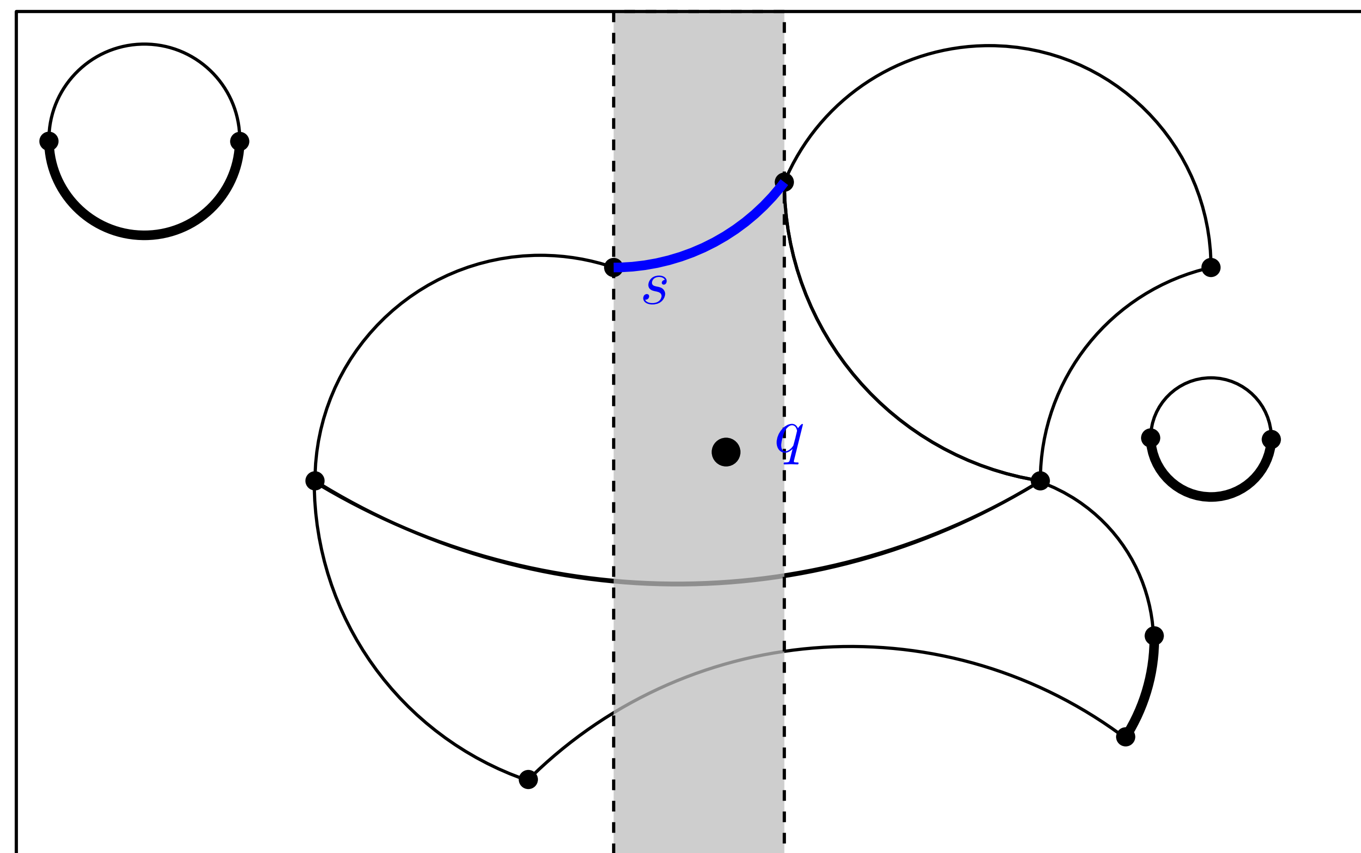
Computational assumption
If the vertical line through a point q intersects an x-monotone segment s then it can be determined in constant time whether q lies above, on, or below s.



Vertical Ray Shooting

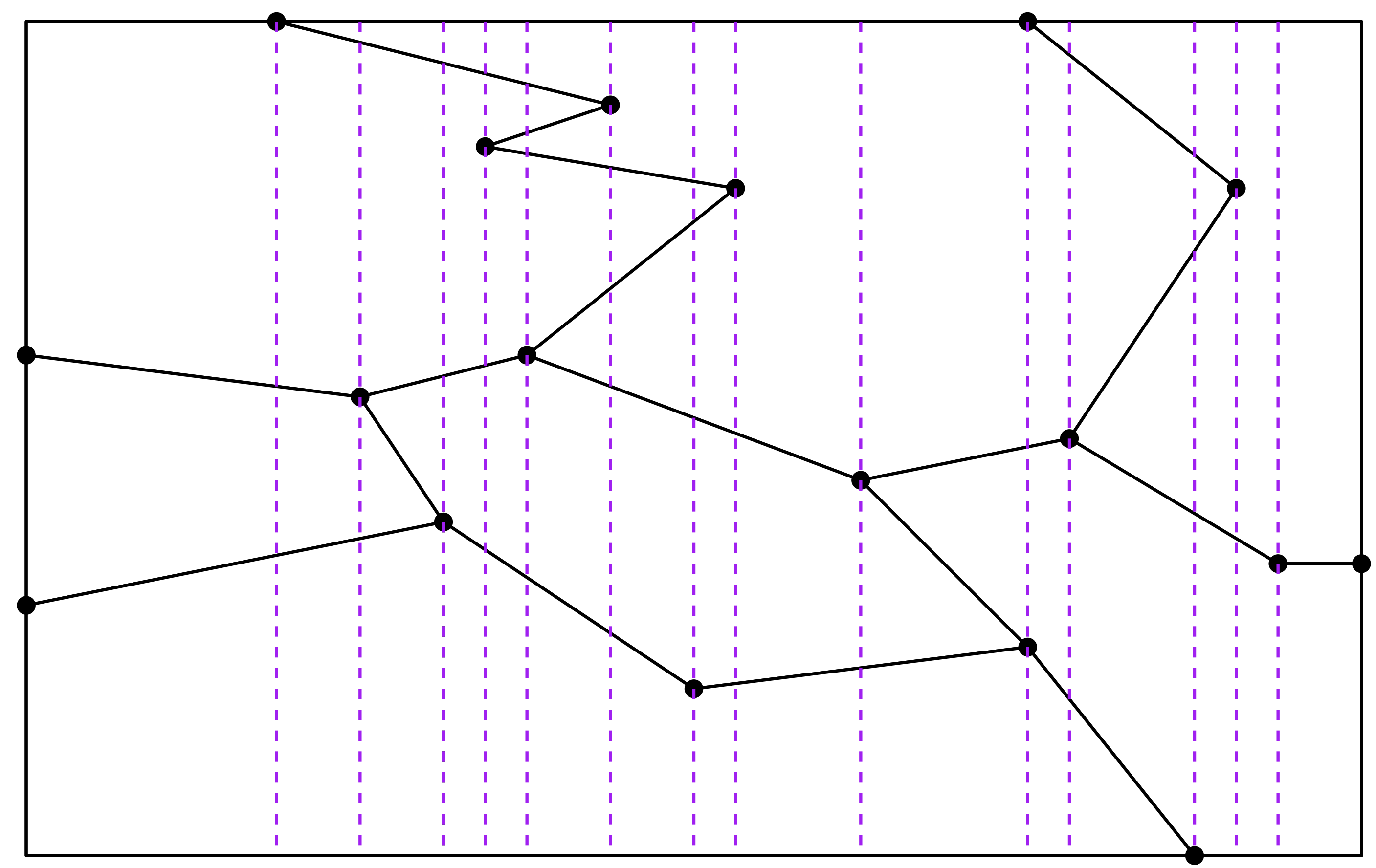
Preprocess a given set S of non-crossing x-monotone curves in the plane (or a bounding box) so that for every query point q it can be determined quickly which curve of S lies immediately above(below) q.

Computational assumption
If the vertical line through a point q intersects an x-monotone segment s then it can be determined in constant time whether q lies above, on, or below s.



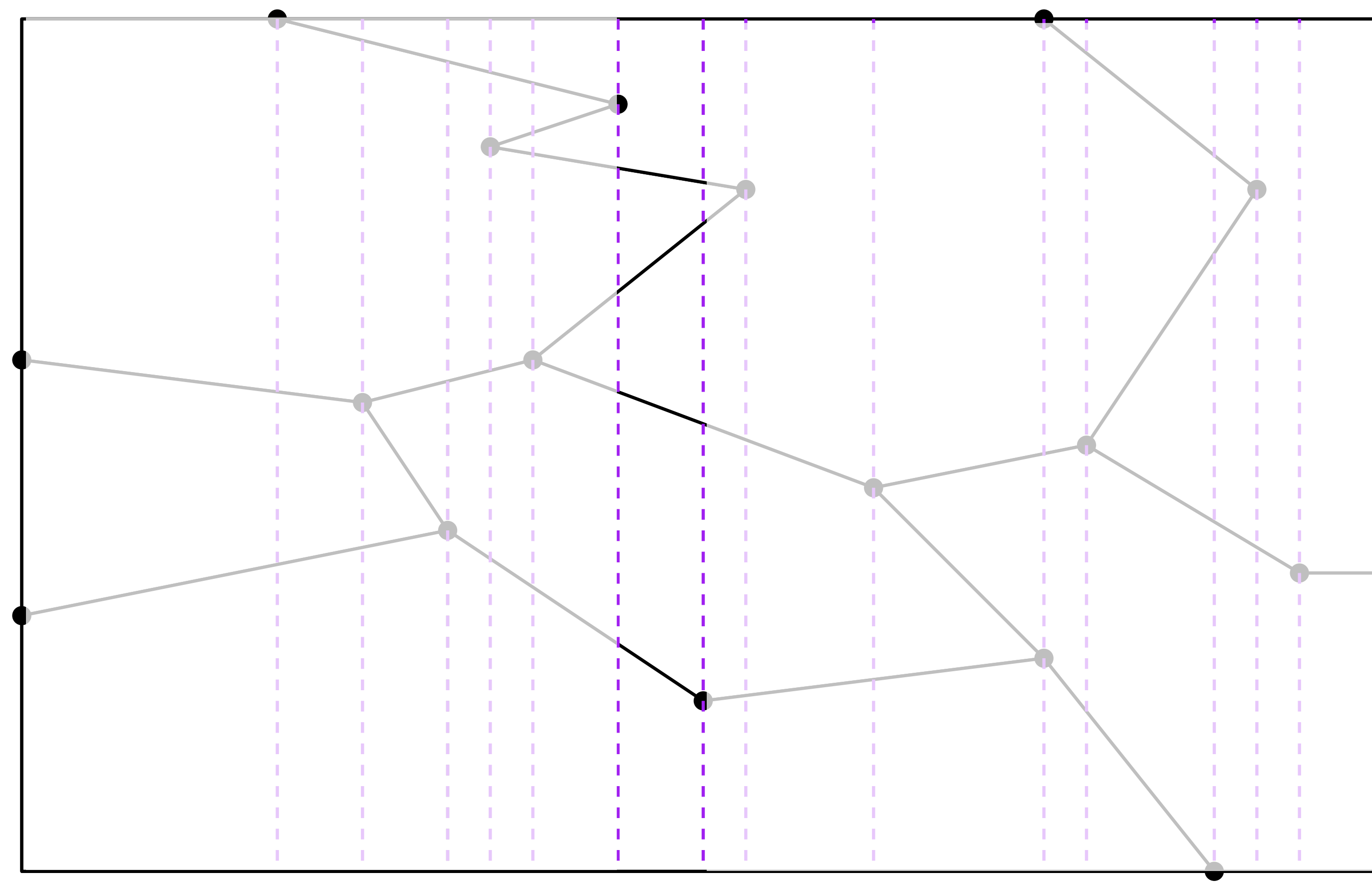
The Slab Method of Dobkin & Lipton

1. Draw a vertical line through each segment endpoint, which partitions the bounding box into slabs.
   Build a binary search structure (x-structure) that allows to determine the slab containing a query point in logarithmic time.



The Slab Method of Dobkin & Lipton

1. Draw a vertical line through each segment endpoint, which partitions the bounding box into slabs.
   Build a binary search structure (x-structure) that allows to determine the slab containing a query point in logarithmic time.

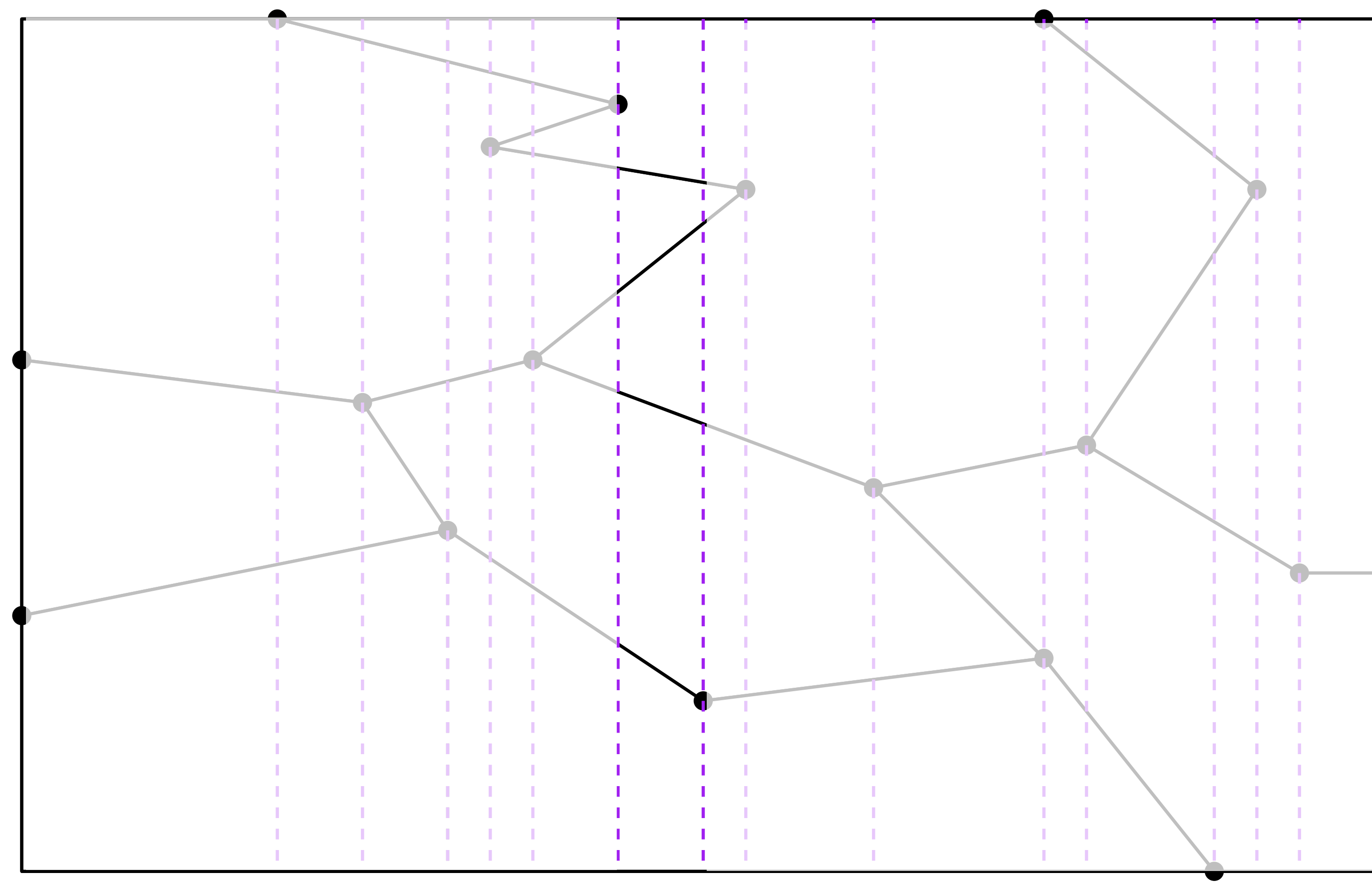


2. In each slab the segments crossing the slab are totally ordered vertically.

   For each slab build a binary search structure (y-structure) to determine the segments immediately above and below the query point.

The Slab Method of Dobkin & Lipton

1. Draw a vertical line through each segment endpoint, which partitions the bounding box into slabs.
   Build a binary search structure (x-structure) that allows to determine the slab containing a query point in logarithmic time.



2. In each slab the segments crossing the slab are totally ordered vertically.

   For each slab build a binary search structure (y-structure) to determine the segments immediately above and below the query point.

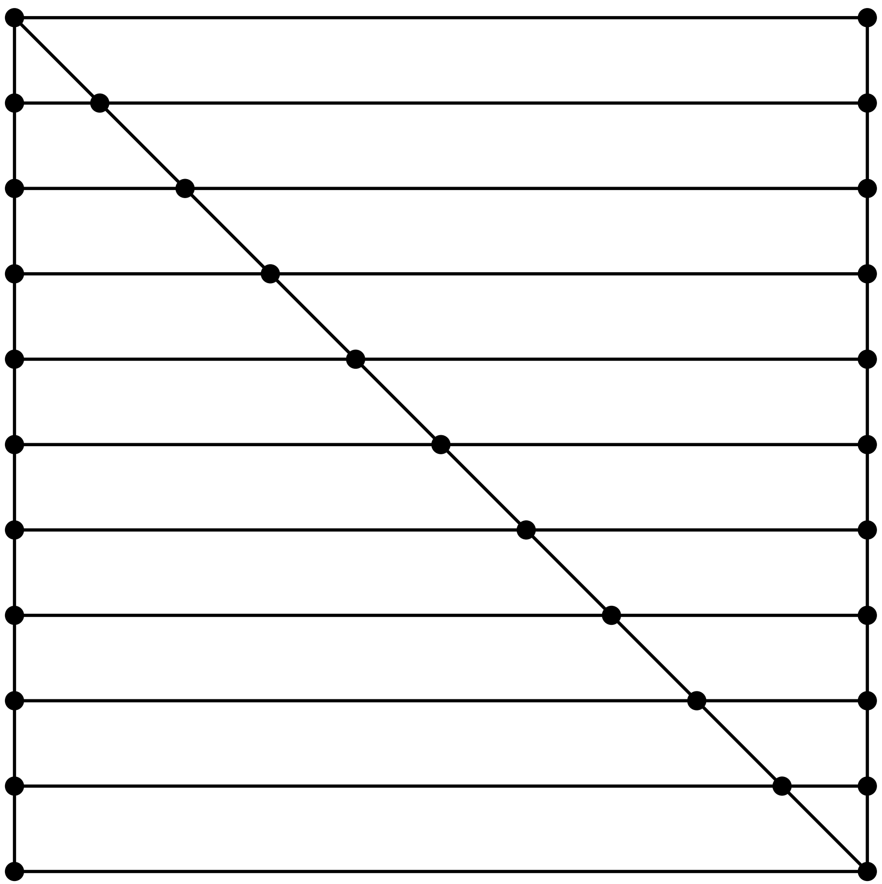
Query time is logarithmic:

$$Q(n) = 2\log_2 n + O(1)$$

The Slab Method of Dobkin & Lipton

Query time: $Q(n) = O(\log n)$

Space usage: $S(n) = O(n^2)$ in the worst case.



Inverse Range Searching Based Methods

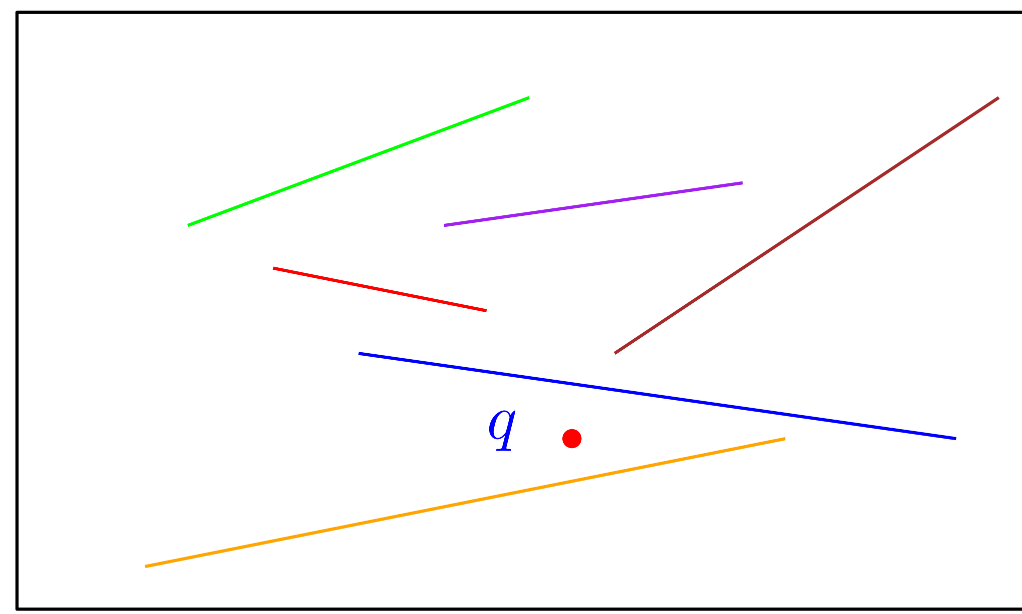
Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.

Inverse Range Searching Based Methods

Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.

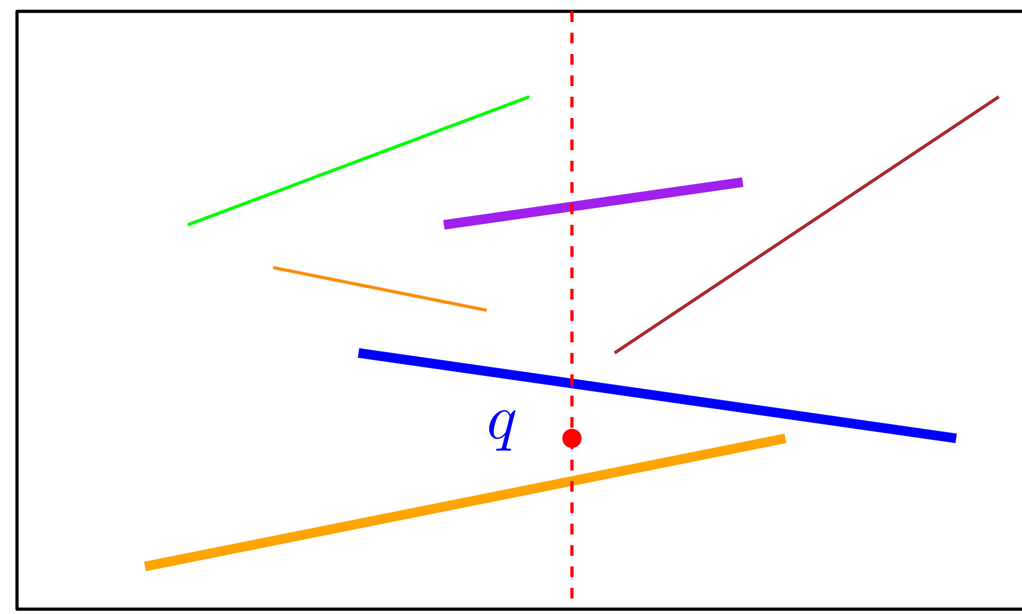


Saarland Informatics Campus

Inverse Range Searching Based Methods

Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.



Saarland Informatics Campus

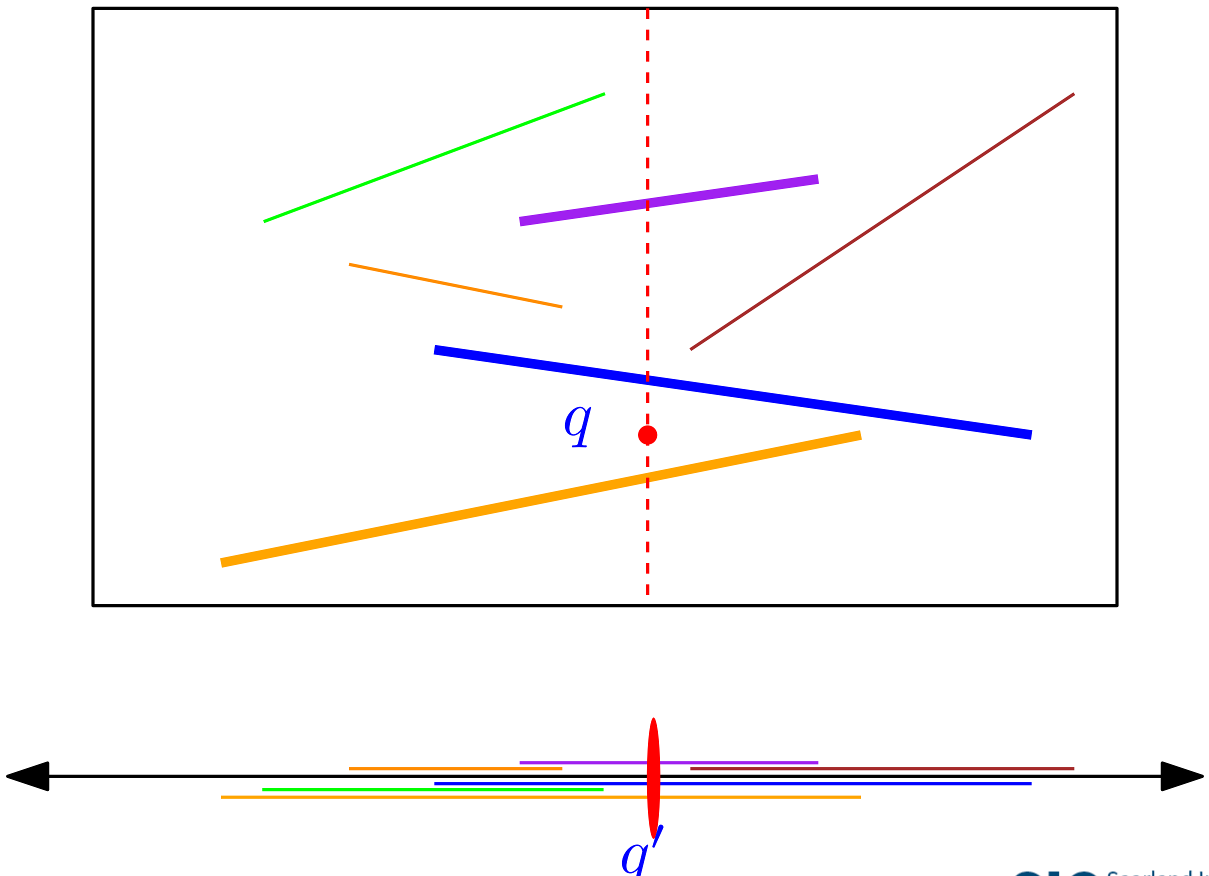
Inverse Range Searching Based Methods

Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.

q' projection of q onto horizontal axis;
s' projection of s onto horizontal axis;

Step 1 corresponds to 1-dimensional problem of finding the intervals s' that contain q' ("inverse range searching")



Inverse Range Searching Based Methods

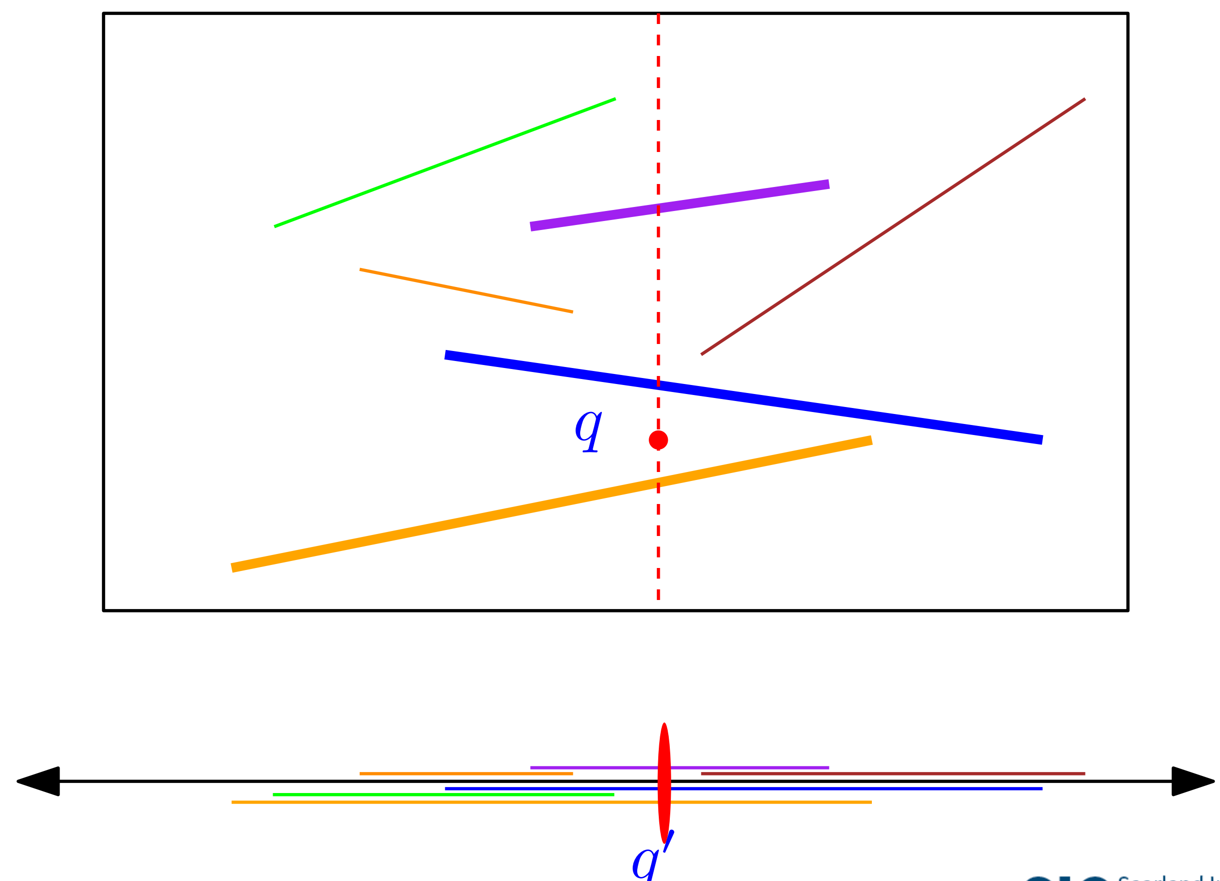
Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.

q' projection of q onto horizontal axis;
s' projection of s onto horizontal axis;

Step 1 corresponds to 1-dimensional problem of finding the intervals s' that contain q' ("inverse range searching")

Possible solutions via
segment tree or
interval tree



Inverse Range Searching Based Methods: Segment Tree

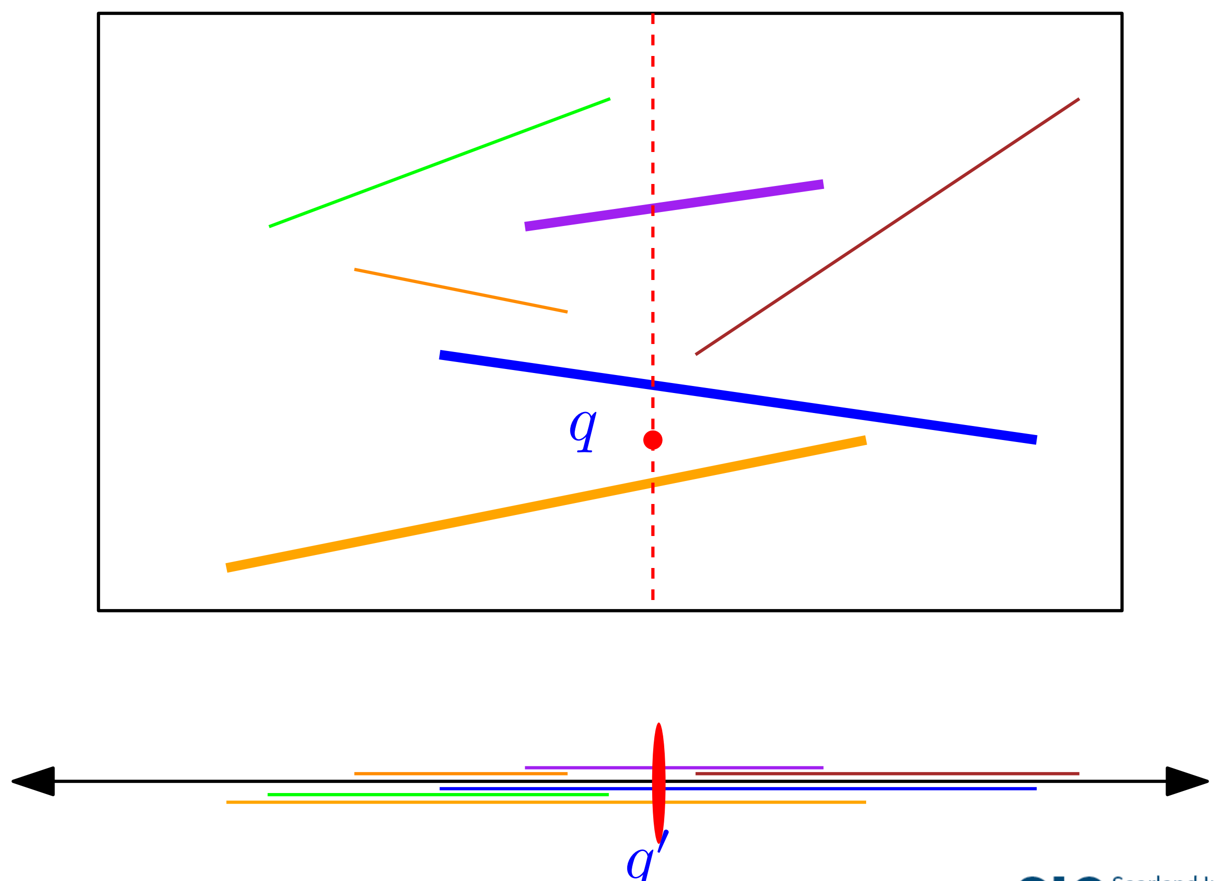
Idea for processing query point q:

1 Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2 Find the correct answer within $S(q)$.

Segment tree provides $S(q)$ as disjoint union of $O(\log n)$ canonical sets of segments (some S_v's from the segment tree)

Proprocess each canonical set S_v to allow vertical binary search for q

Search for q in each of the relevant canonical sets.



Query time $Q(n) = O(\log^2 n)$
Space usage $S(n) = O(n \log n)$

Inverse Range Searching Based Methods: Segment Tree

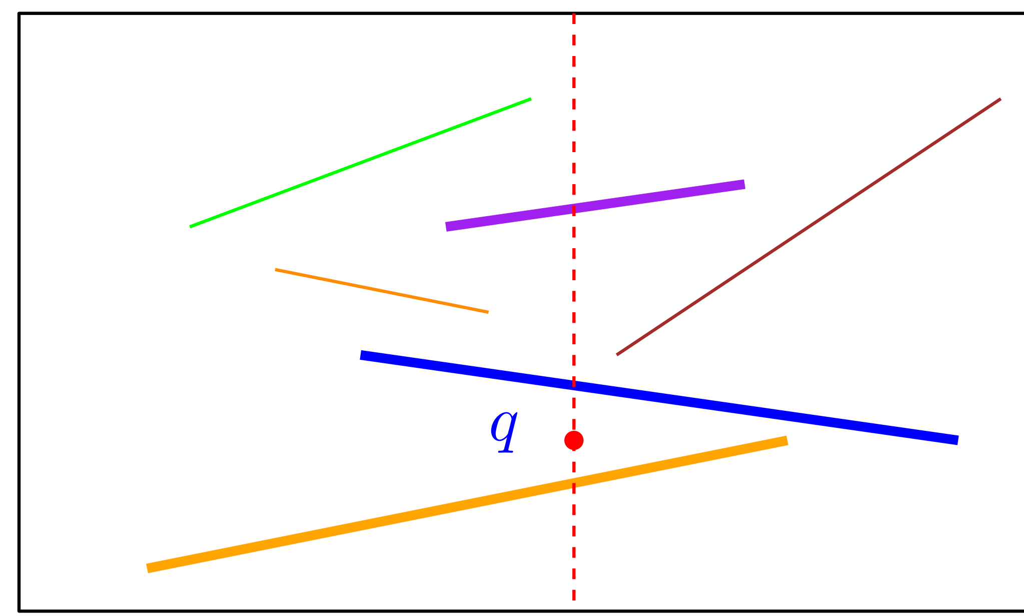
Idea for processing query point q:

1. Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.
2. Find the correct answer within $S(q)$.

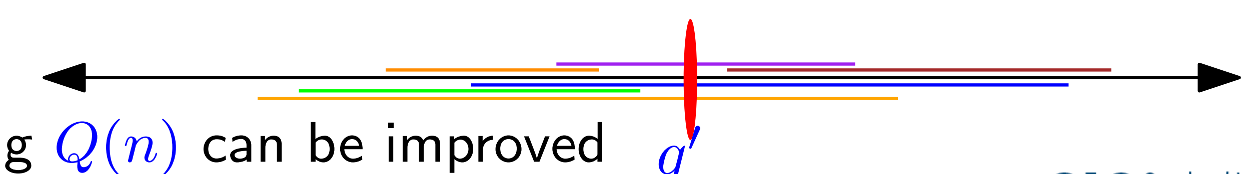
Segment tree provides $S(q)$ as disjoint union of $O(\log n)$ canonical sets of segments (some S_v's from the segment tree)

Proprocess each canonical set S_v to allow vertical binary search for q

Search for q in each of the relevant canonical sets.



Query time $Q(n) = O(\log^2 n)$
Space usage $S(n) = O(n \log n)$



With appropriate fractional cascading $Q(n)$ can be improved to $O(\log n)$. (homework)

Inverse Range Searching Based Methods: Interval Tree

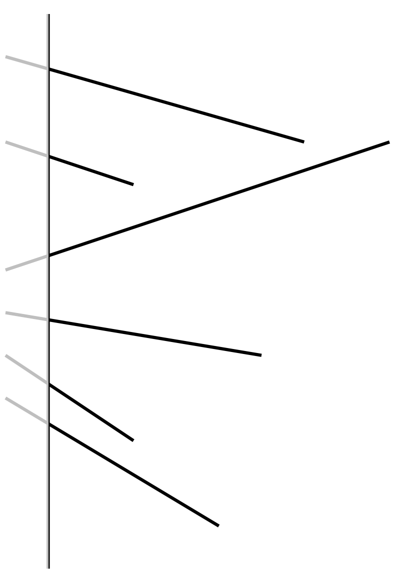
Idea for processing query point q:

1 Identify the set $S(q)$, the set of segments in S that intersect the vertical line through q.

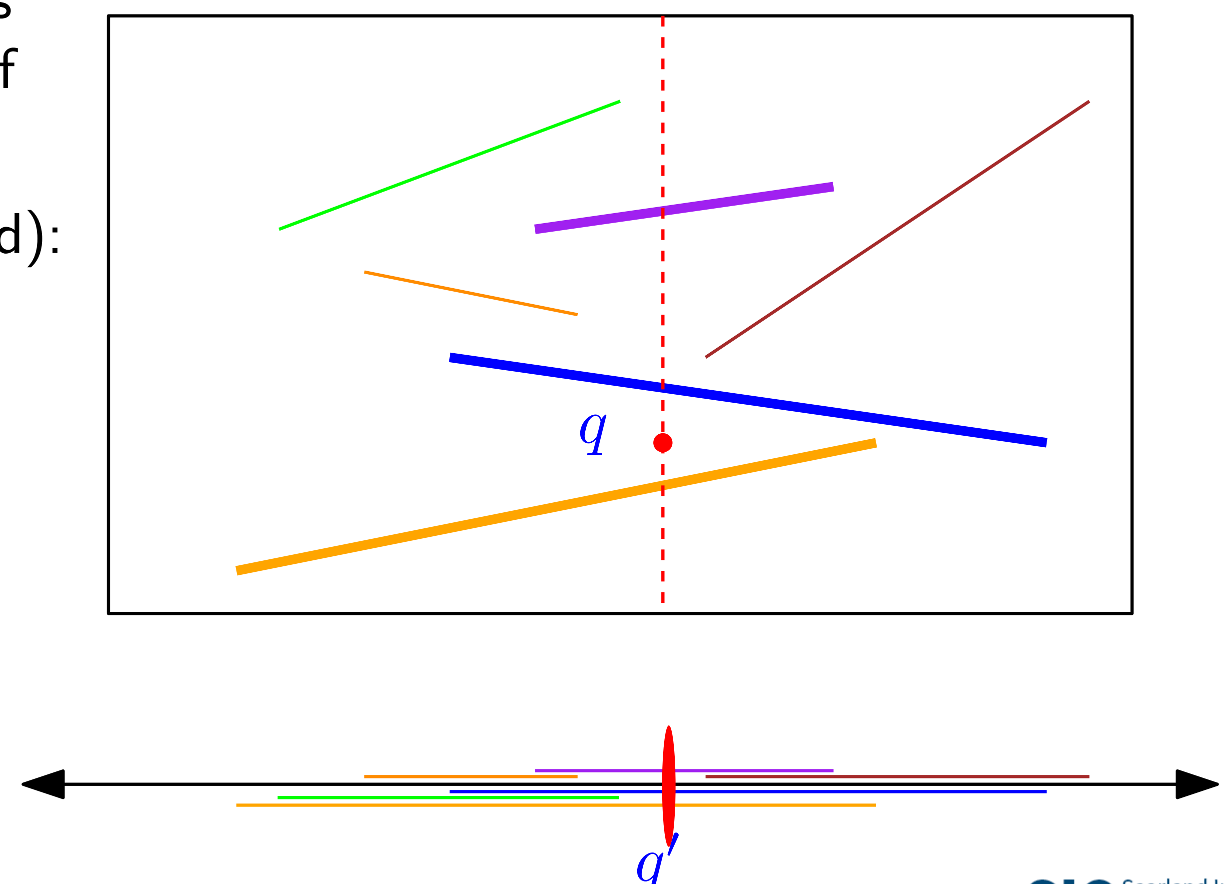
2 Find the correct answer within $S(q)$.

Interval tree provides a superset of $S(q)$ as disjoint union of $O(\log n)$ canonical sets of segments.

They have the following form (left attached):



or mirror image (right attached)



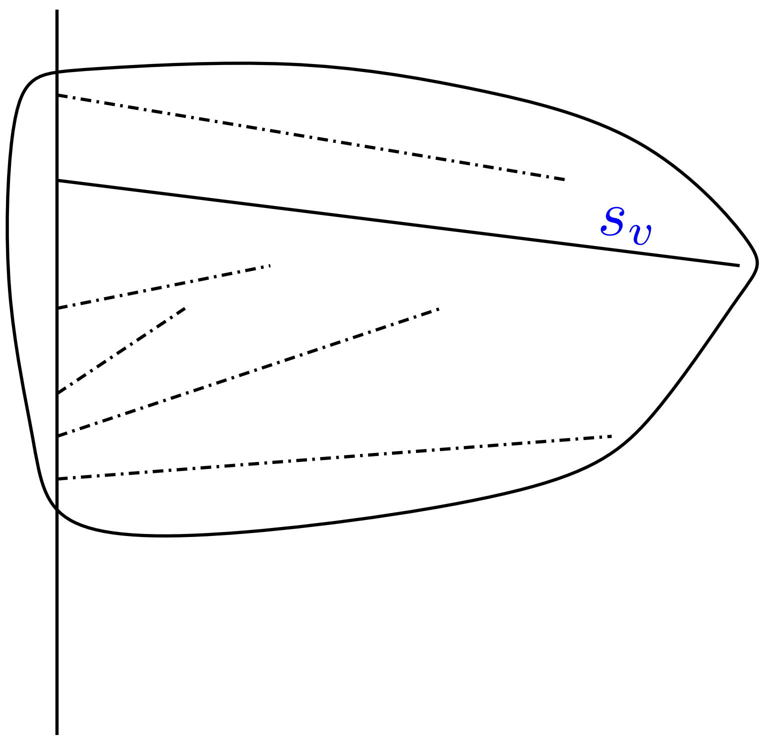
Inverse Range Searching Based Methods: Interval Tree

Want to do fast vertical ray shooting in left attached segments.

segments are vertically ordered according to their attachment point;

build binary tree T whose leaves are the segments in this vertical ordering;

for each node v in the tree store the segment s_v from T_v that extends furthest away from the attachment line;



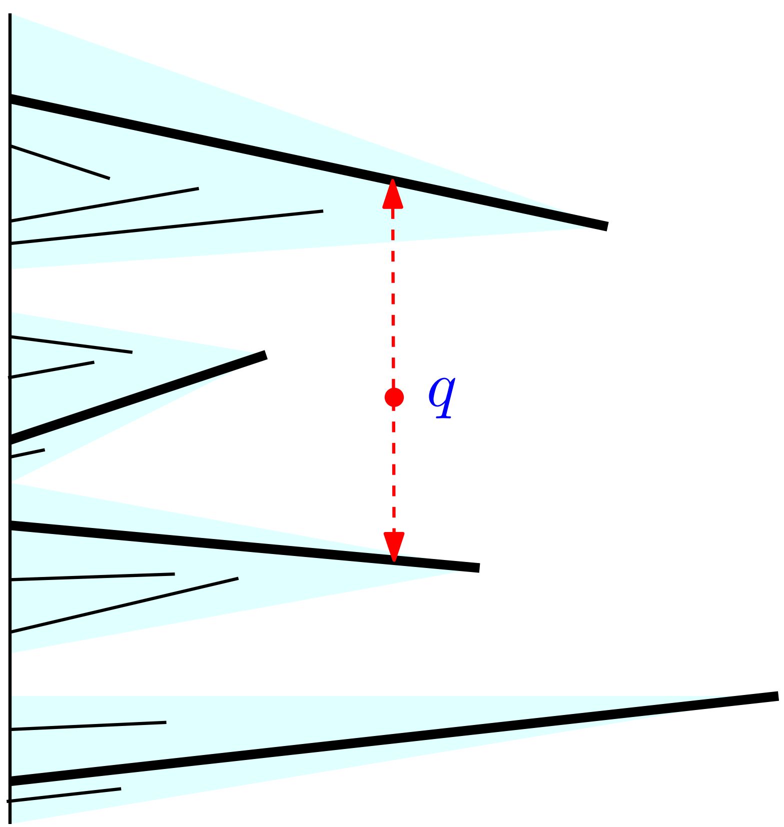
SIC Saarland Informatics Campus

Inverse Range Searching Based Methods: Interval Tree

Want to do fast vertical ray shooting in left attached segments.

Vertical ray shooting among the s_v's from 4 nodes of T on the same level allows to eliminate at least two subtrees from consideration.

Recurse in the remaining trees.

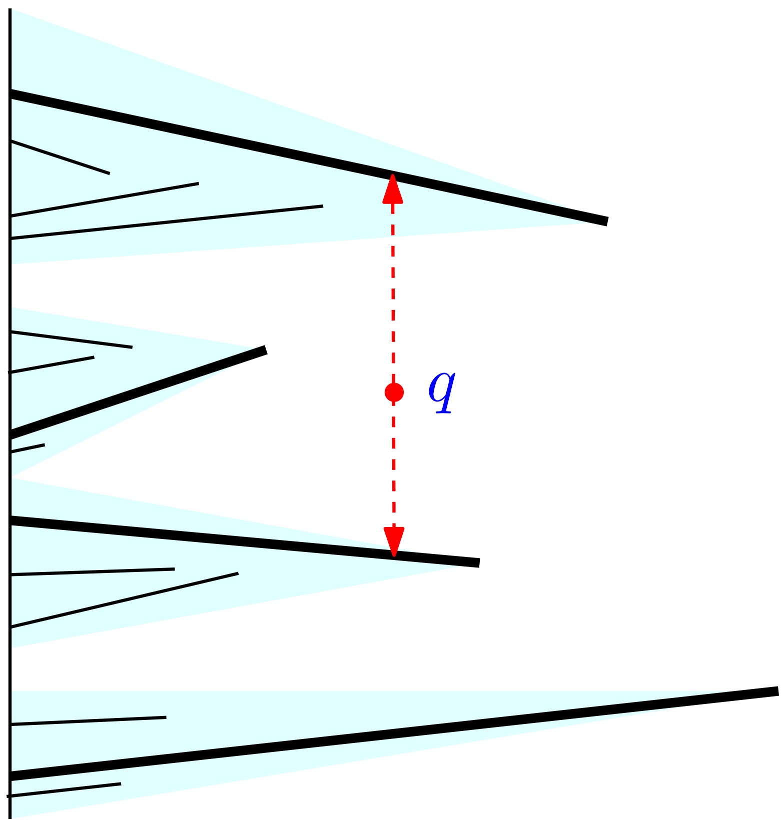


Inverse Range Searching Based Methods: Interval Tree

Want to do fast vertical ray shooting in left attached segments.

Vertical ray shooting among the s_v's from 4 nodes of T on the same level allows to eliminate at least two subtrees from consideration.

Recurse in the remaining trees.



constant number of comparisons necessary to descend down one level in the tree

Therefore logarithmis search time within one set of attached segments

$Q(n) = O(\log^2 n)$ since $O(\log n)$ attached sets need to be searched

$S(n) = O(n)$ since every segment occurs in only two attachment sets

Cheng and Janardan 1992

SIC Saarland Informatics Campus

Optimal Planar Point Location ?

Segment tree + fractional cascading: $Q(n) = O(\log n)$ $S(n) = O(n \log n)$

Interval trees: $Q(n) = O(\log^2 n)$ $S(n) = O(n)$

Is optimal query time $Q(n) = O(\log n)$ with space $S(n) = O(n)$ possible?

Optimal Planar Point Location ?

Segment tree + fractional cascading: $Q(n) = O(\log n)$ $S(n) = O(n \log n)$

Interval trees: $Q(n) = O(\log^2 n)$ $S(n) = O(n)$

Is optimal query time $Q(n) = O(\log n)$ with space $S(n) = O(n)$ possible?

YES

1978 Lipton and Tarjan using the new planar separator theorem (very complicated, horrible constants)

1979 Kirkpatrick (simple, moderate consants, but specialized)

1984 Edelsbrunner, Guibas, and Stolfi ($Q(n) \leq 3 \cdot \log_2 n$)

1986 Sarnak and Tarjan using persistent search trees

1986 Cole based on searching similar lists

1997 Goodrich, Orletsky, and Ramaiyer ($Q(n) \leq 2 \cdot \log_2 n$)

1998 Adamy and Seidel $Q(n) \leq 1 \cdot \log_2 n + 2\sqrt{\log_2 n} + O(\sqrt[4]{\log} n)$

1990 Mulmuley / Seidel randomized methods

• Planar point location

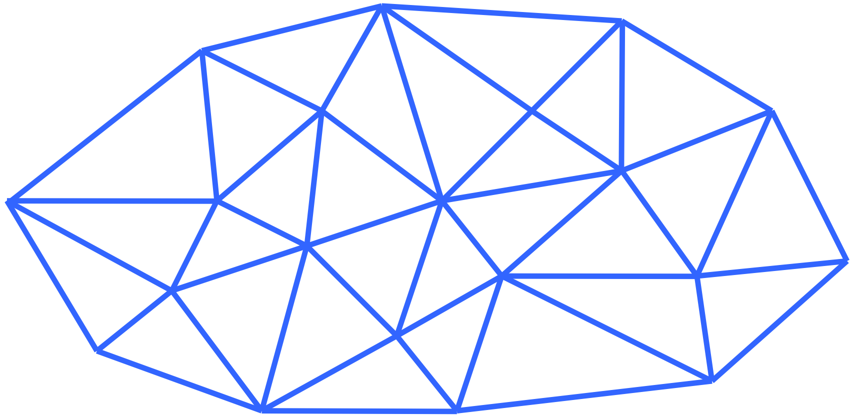
Optimal methods:

- Lipton – Tarjan
- Kirkpatrick
- Edelsbrunner – Guibas - Stolfi
- Cole
- Sarnak – Tarjan
- randomized

Other methods:

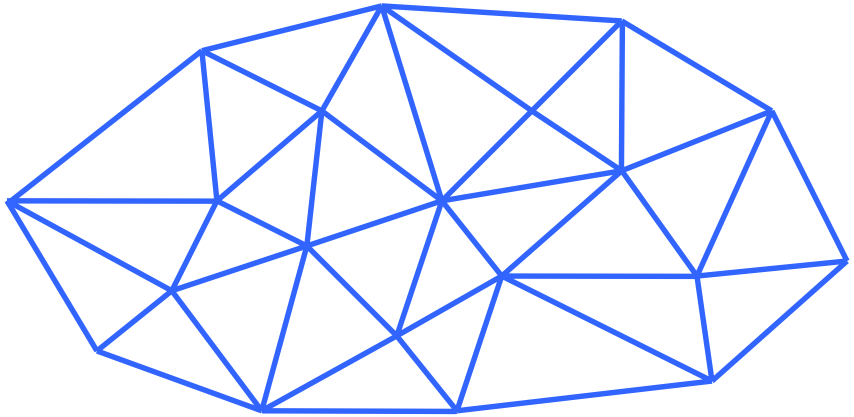
- via segment trees / via interval trees
- trapezoidal search trees
- constant optimal methods
- via cuttings
- distribution adaptive methods
- ...

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

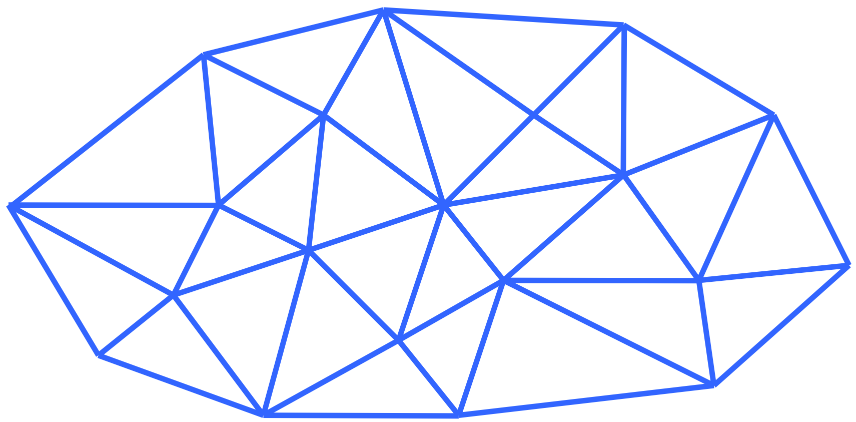
Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

to obtain **smaller** G'

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

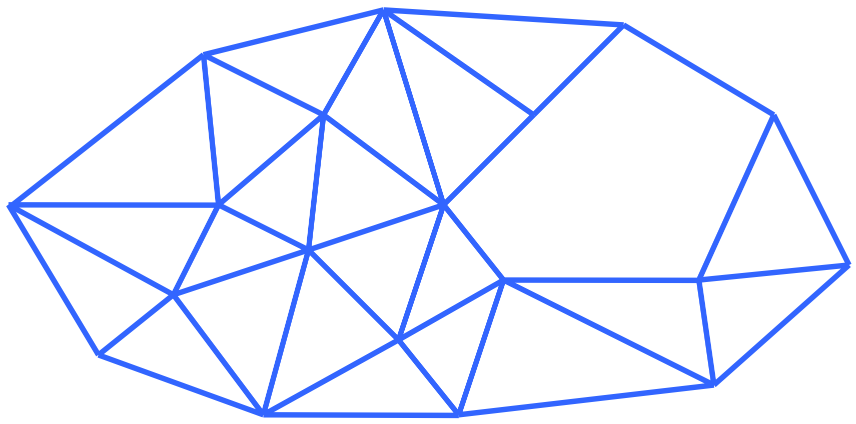


subdivision G

to obtain **smaller** G'

remove low degree vertex
and retriangulate hole

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

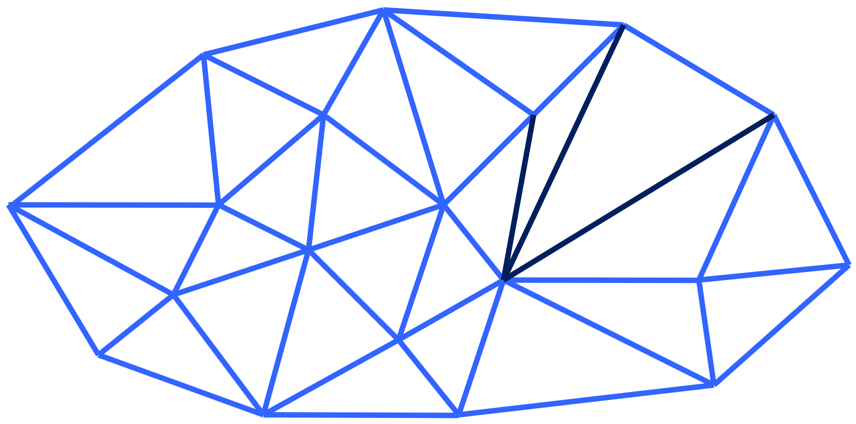


subdivision G

to obtain **smaller** G'

remove low degree vertex
and retriangulate hole

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

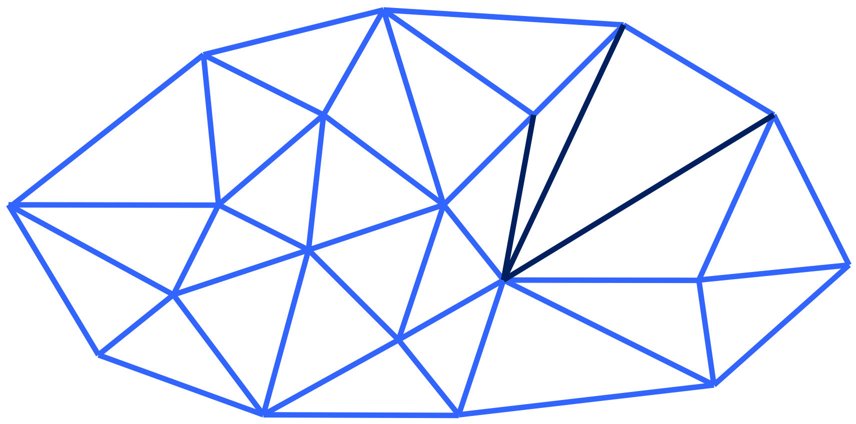


subdivision G

to obtain **smaller** G'

remove low degree vertex
and retriangulate hole

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



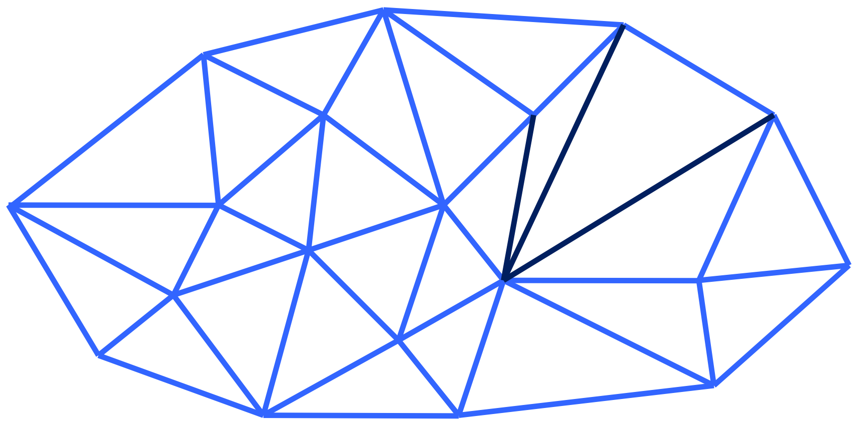
subdivision *G*

to obtain **smaller** *G'*

- remove low degree vertex
  and retriangulate hole

repeat recursively

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

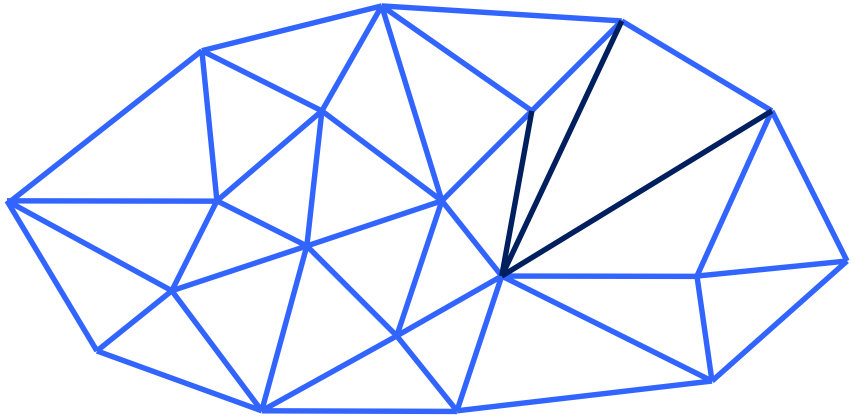
to obtain **smaller** G'

remove low degree vertex
and retriangulate hole

repeat recursively

Query for point q :

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

to obtain **smaller** G'

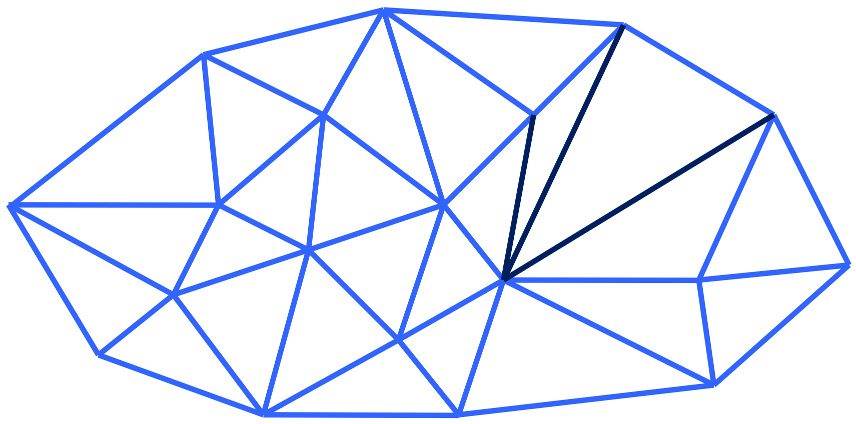
remove low degree vertex and retriangulate hole

repeat recursively

Query for point q :

locate q in G'

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

to obtain **smaller** G'

remove low degree vertex
and retriangulate hole

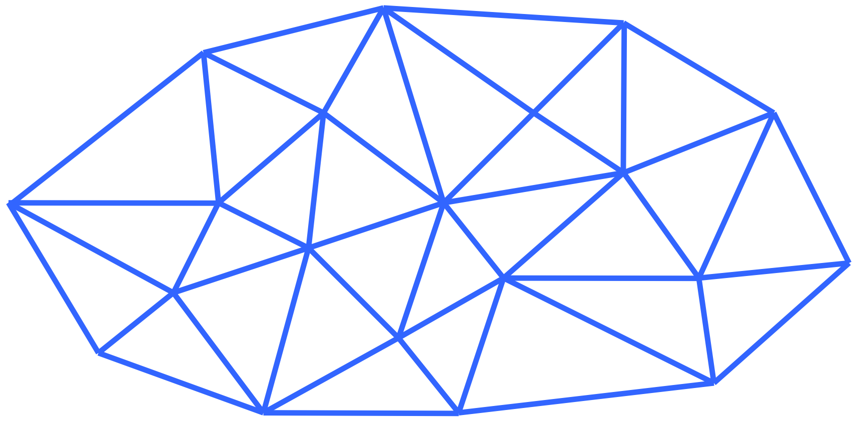
repeat recursively

Query for point q :

locate q in G'

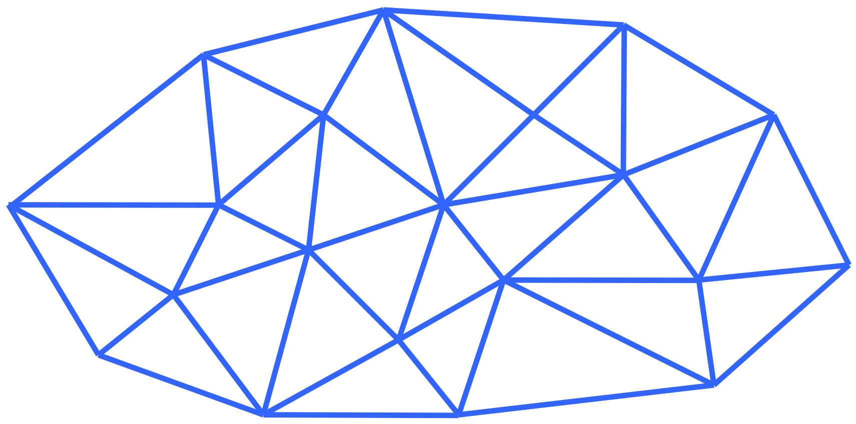
if q in "black" triangle then determine correct triangle of G
else triangle is correct answer already

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



SAARLAND UNIVERSITY
COMPUTER SCIENCE

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

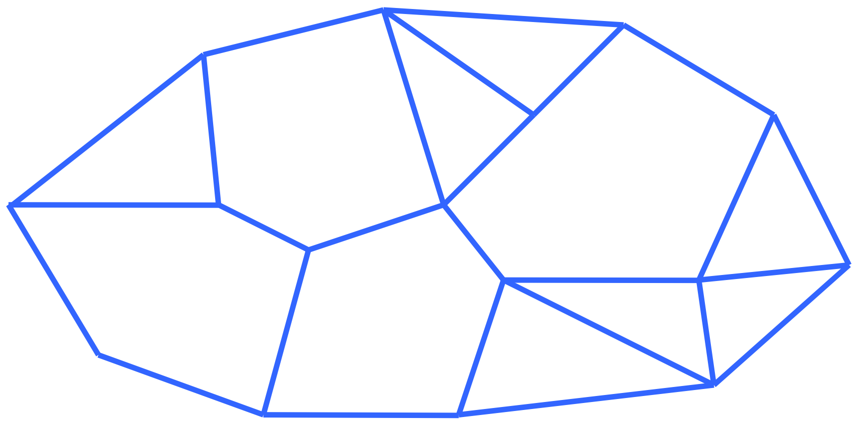


subdivision G

to obtain **smaller** G'

remove large independent set of low degree vertices and retriangulate holes

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

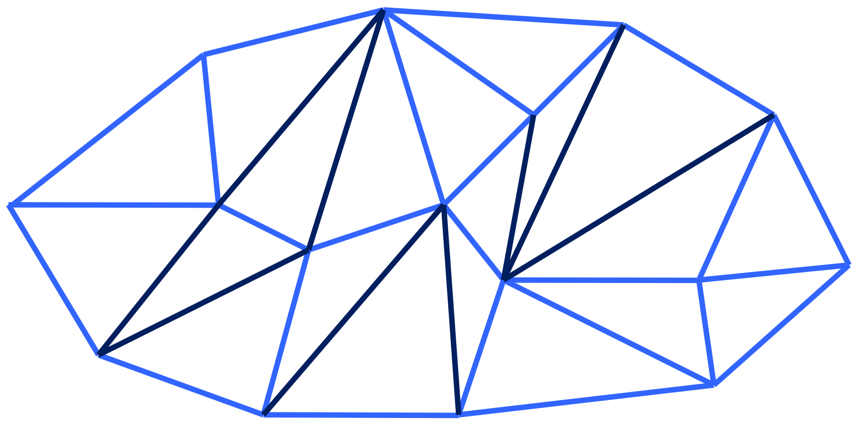


subdivision *G*

to obtain **smaller** *G'*

remove large independent set of low degree vertices and retriangulate holes

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

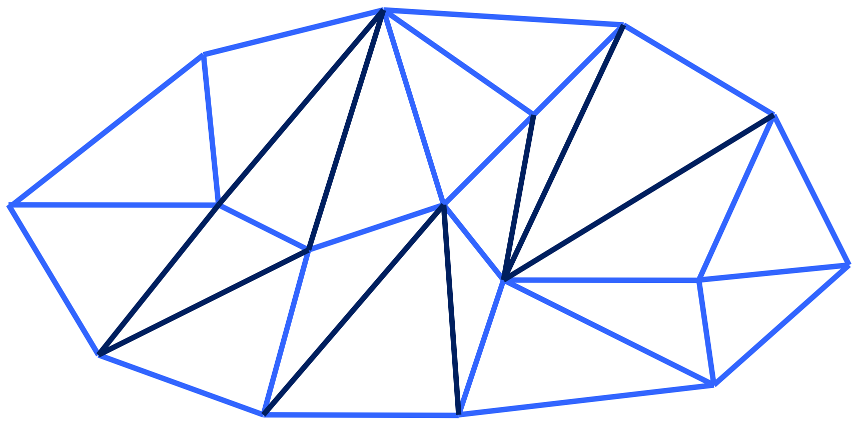


subdivision G

to obtain **smaller** G'

remove large independent set of low degree vertices and retriangulate holes

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



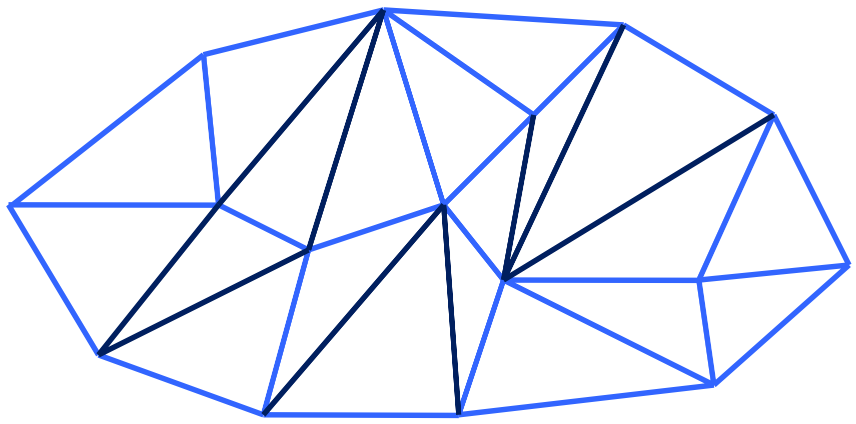
subdivision G

to obtain **<u>smaller</u>** G'

remove large independent set of low degree vertices and retriangulate holes

repeat recursively

Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision G

to obtain **smaller** G'

remove large independent set of low degree vertices and retriangulate holes

repeat recursively

Query for point q :

locate q in G'

if q in "black" triangle then determine correct triangle of G
else triangle is correct answer already

Lemma: For every $d \geq 6$ there exists an $\alpha > 0$ such that every n-vertex planar graph has an independent set of at least αn vertices of degree $\leq d$.

Lemma: For every $d \geq 6$ there exists an $\alpha > 0$ such that every n-vertex planar graph has an independent set of at least αn vertices of degree $\leq d$.

$\Rightarrow$ height of hierarchy of subdivisions can be made $O(\log n)$

Lemma: For every $d \geq 6$ there exists an $\alpha > 0$ such that every n-vertex planar graph has an independent set of at least αn vertices of degree $\leq d$.

$\Rightarrow$ height of hierarchy of subdivisions can be made O(log n)

$\Rightarrow$ Query time O(log n)
Space O(n)
Preprocessing O(n)

Lemma: For every $d \geq 6$ there exists an $\alpha > 0$ such that every n-vertex planar graph has an independent set of at least αn vertices of degree $\leq d$.

$\Rightarrow$ height of hierarchy of subdivisions can be made O(log n)

$\Rightarrow$ Query time O(log n)
Space O(n)
Preprocessing O(n)

Shortcomings:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

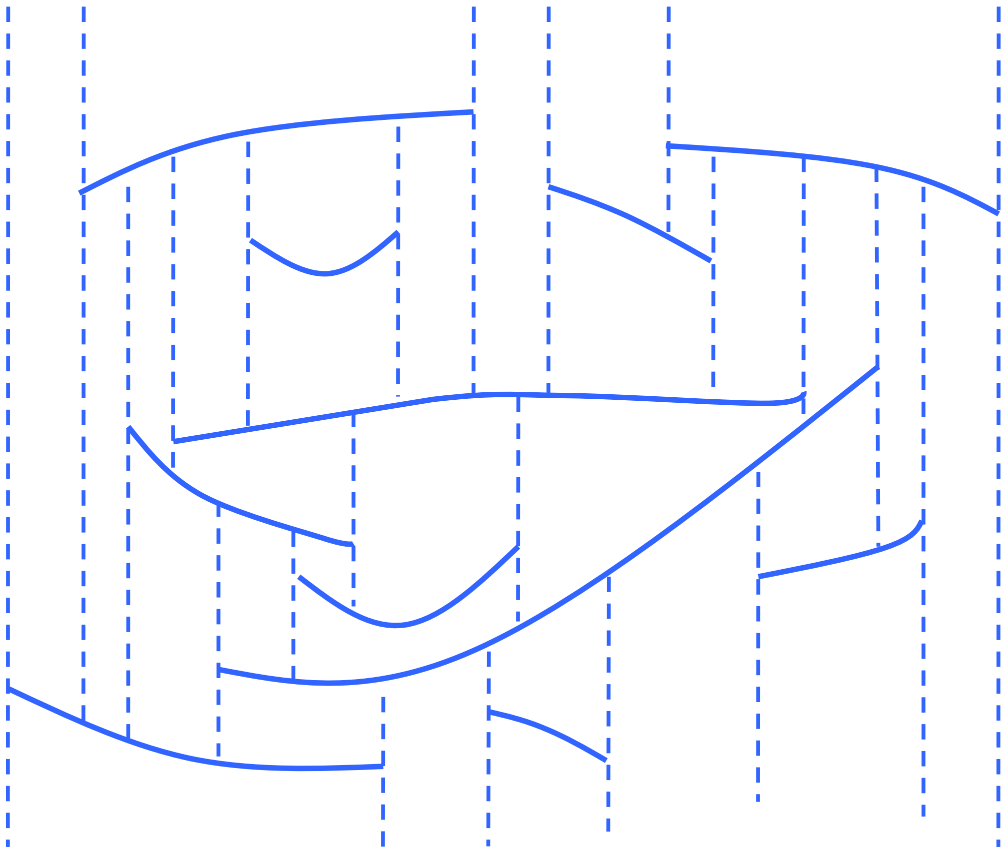
Shortcomings:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

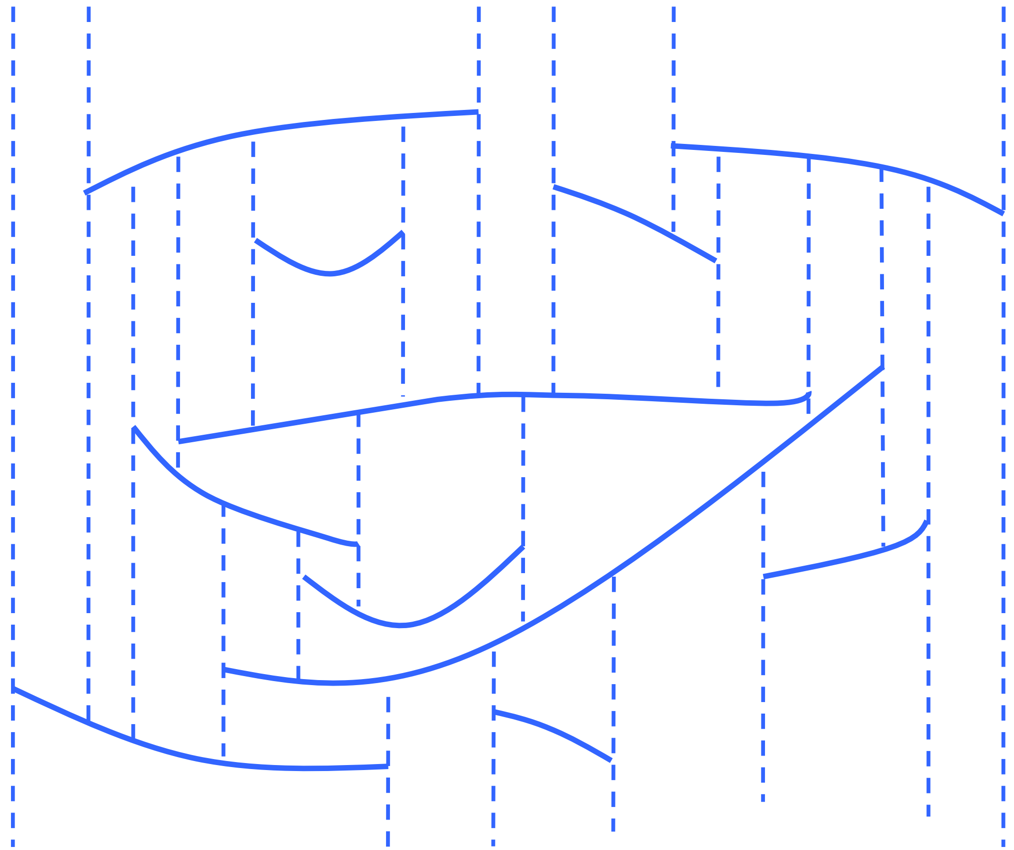
Shortcomings:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

Idea: apply this hierarchical approach to trapezoidations and but remove segments instead of vertices.

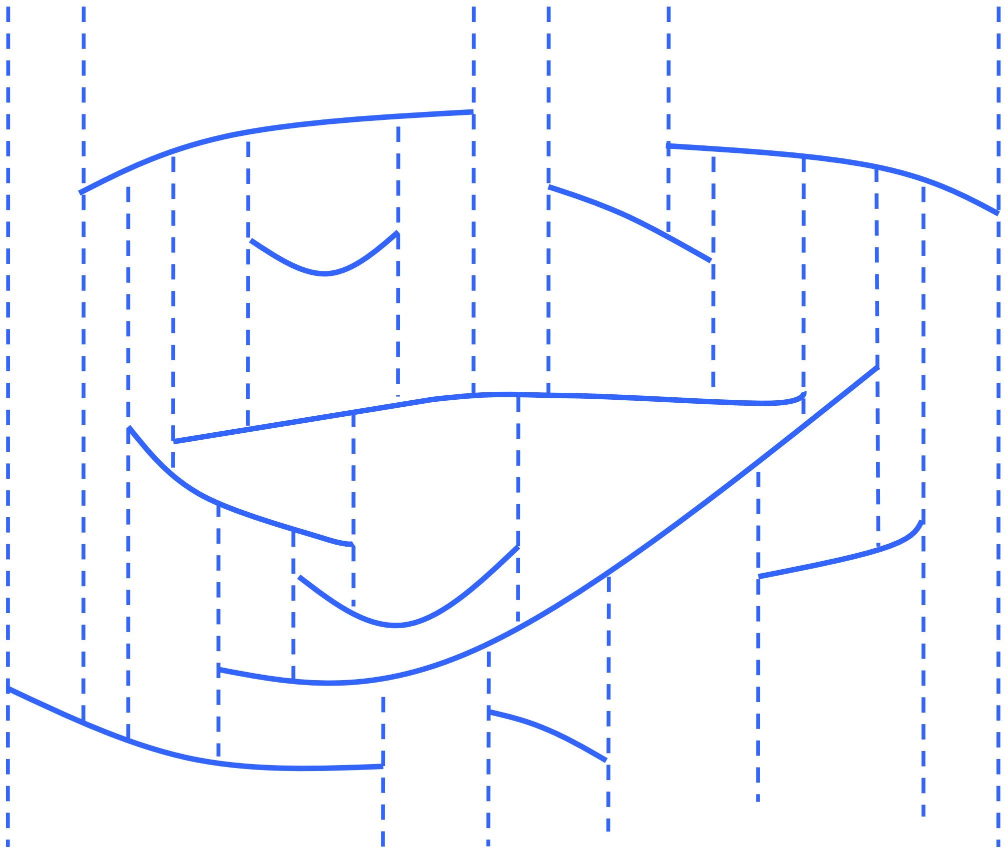


trapezoidation G



trapezoidation *G*

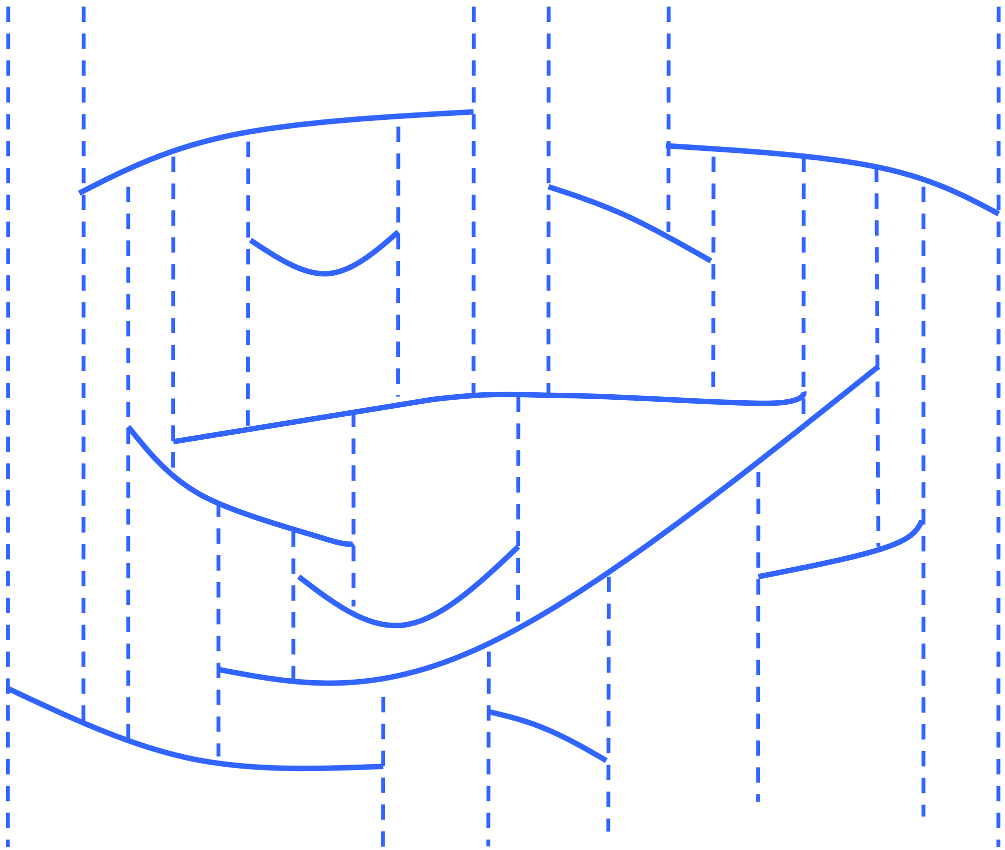
to obtain **smaller** *G'*



trapezoidation G

to obtain **smaller** G'

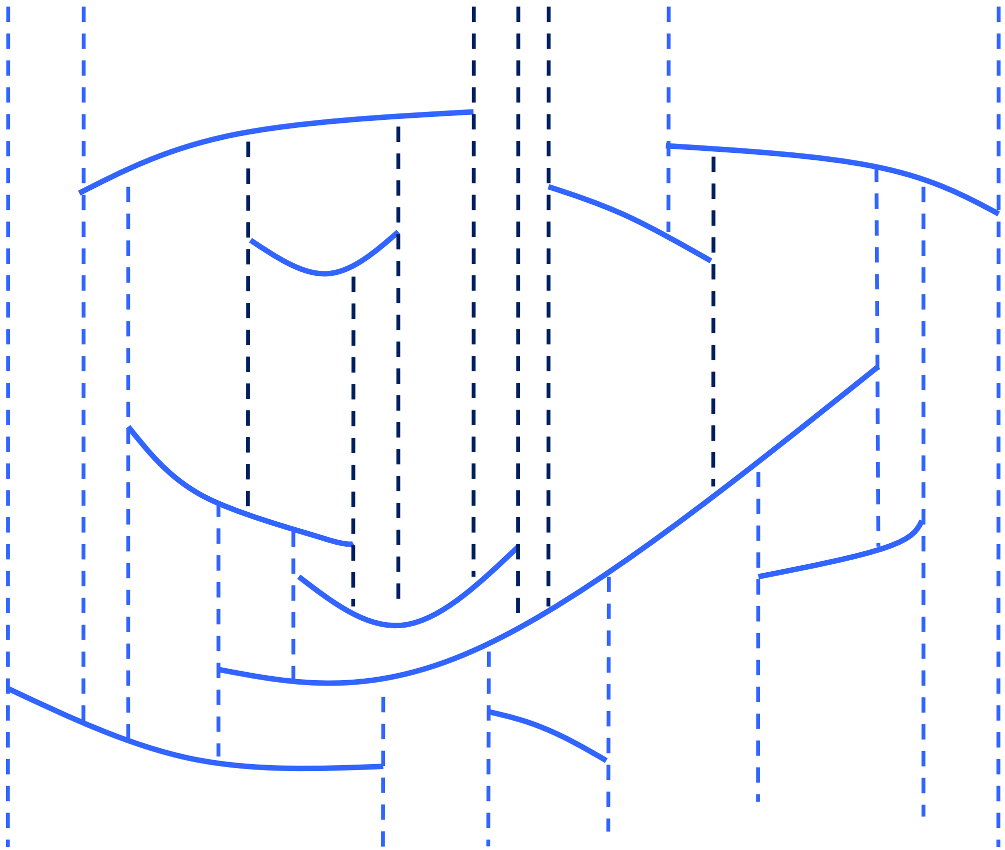
remove **some** segment



trapezoidation *G*

to obtain **smaller** *G'*

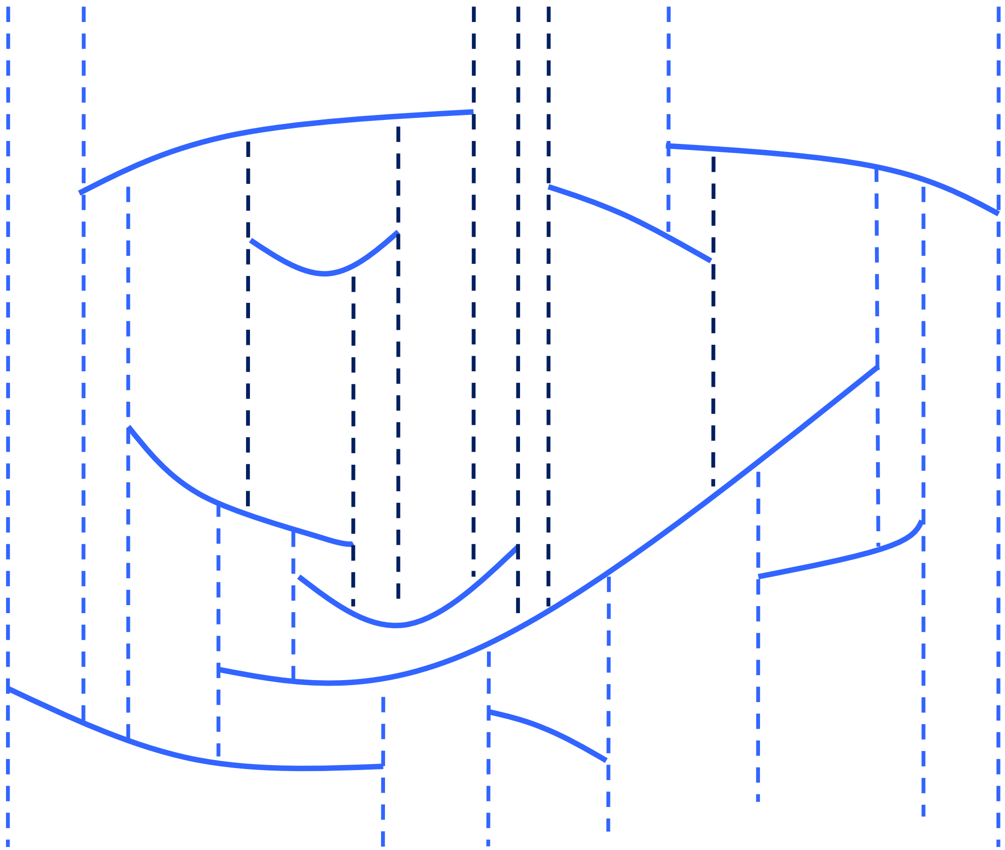
remove **some** segment
and "retrapezoidalize" hole



trapezoidation G

to obtain **smaller** G'

remove **some** segment
and “retrapezoidalize” hole

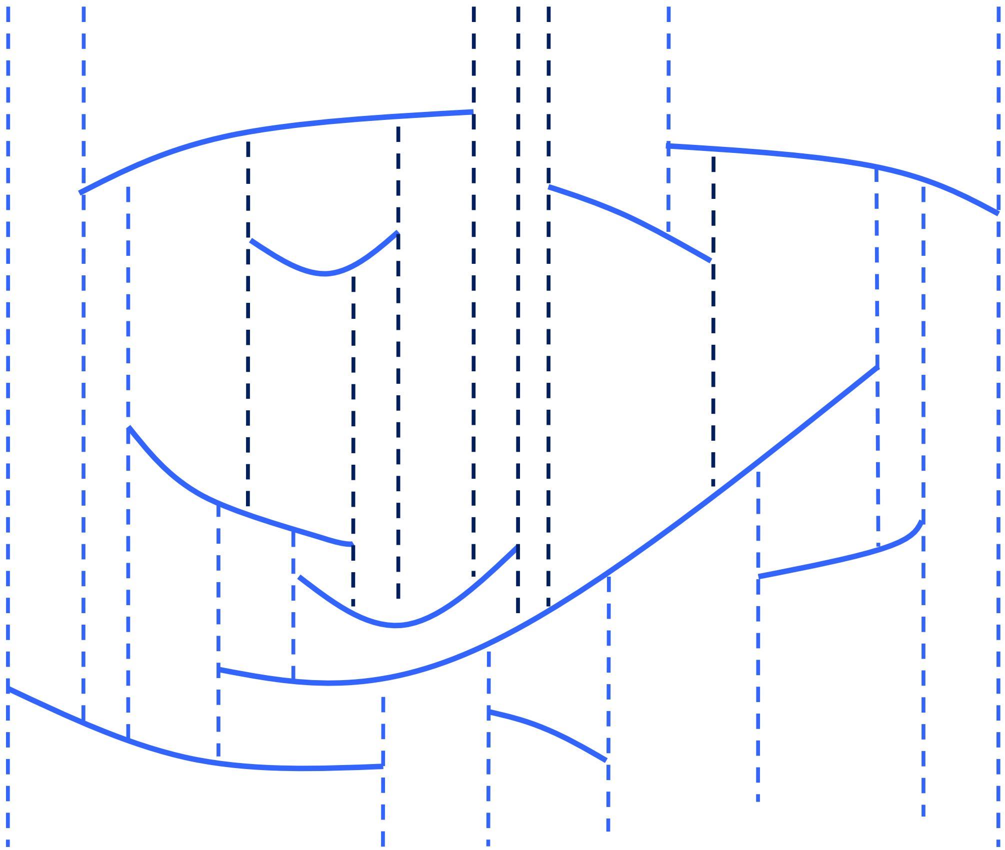


trapezoidation G

to obtain **smaller** G'

remove **some** segment
and “retrapezoidalize” hole

repeat recursively



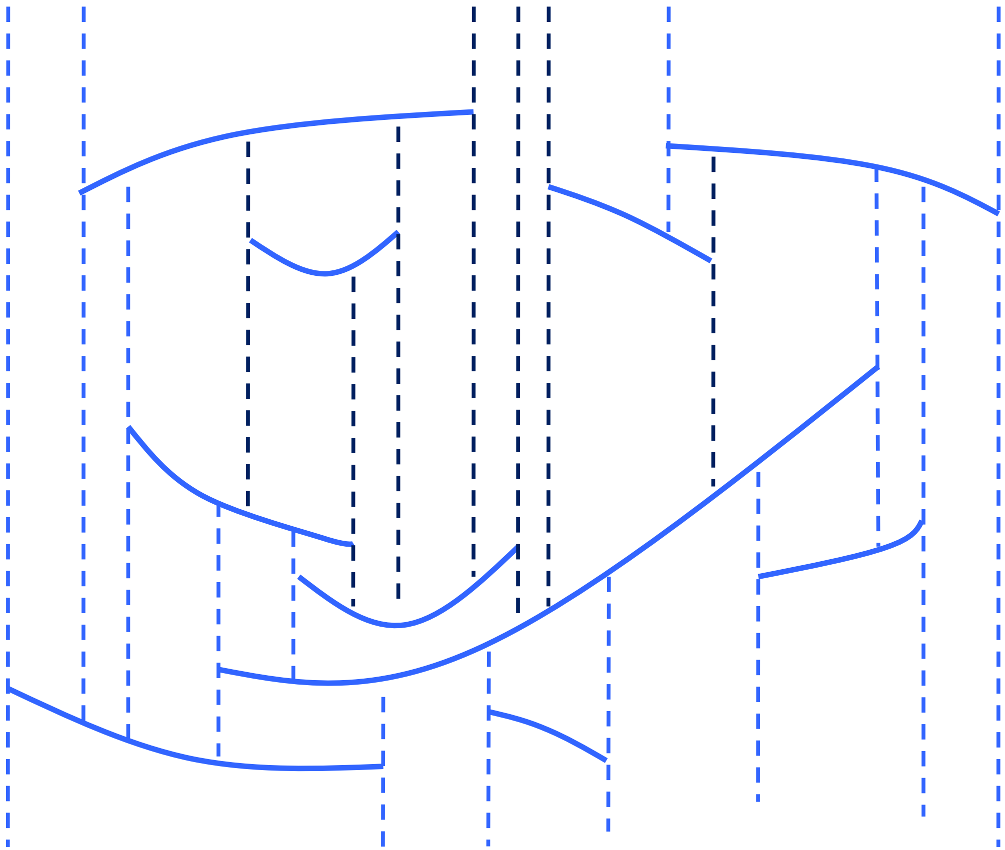
trapezoidation G

to obtain **smaller** G'

remove **some** segment
and “retrapezoidalize” hole

repeat recursively

Query for point q :



trapezoidation G

to obtain **smaller** G'

remove **some** segment
and "retrapezoidalize" hole

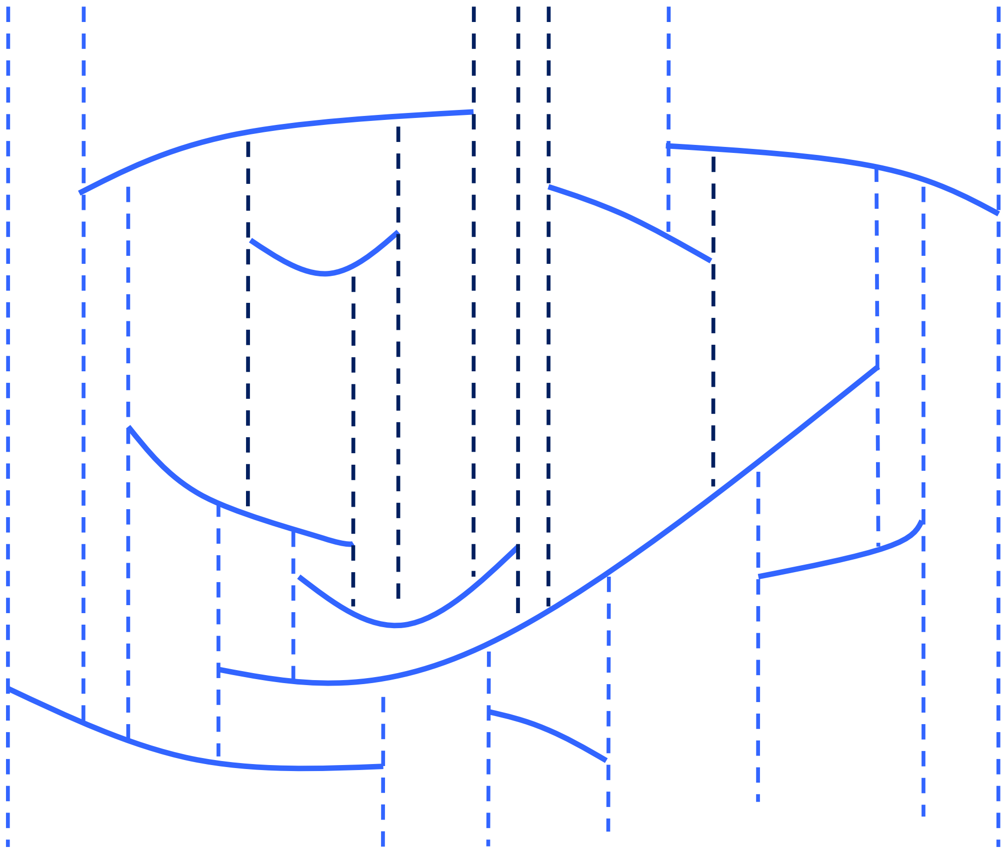
repeat recursively

Query for point q :

locate q in G'

SAARLAND UNIVERSITY

COMPUTER SCIENCE



trapezoidation G

to obtain **smaller** G'

remove **some** segment
and “retrapezoidalize” hole

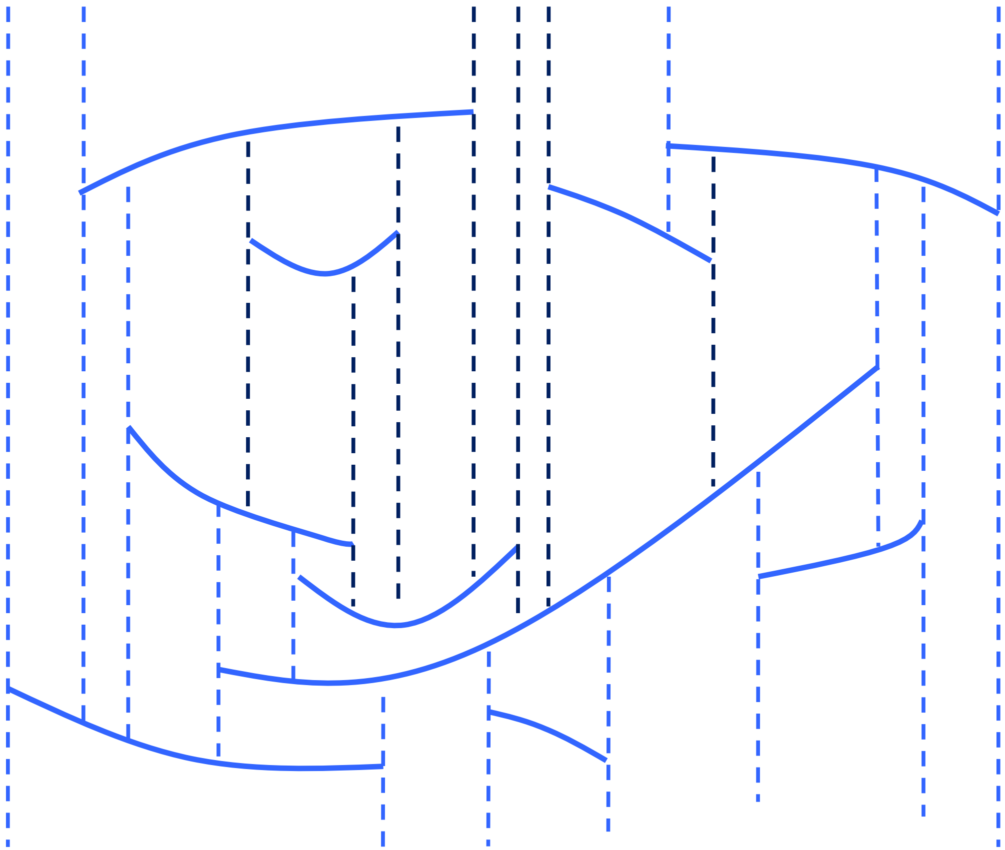
repeat recursively

Query for point q :

locate q in G'

if q in “black” trapezoid then determine correct trapezoid of G
else trapezoid is correct answer already

SAARLAND UNIVERSITY
COMPUTER SCIENCE



trapezoidation G

to obtain **smaller** G'

remove **some** segment
and “retrapezoidalize” hole

repeat recursively

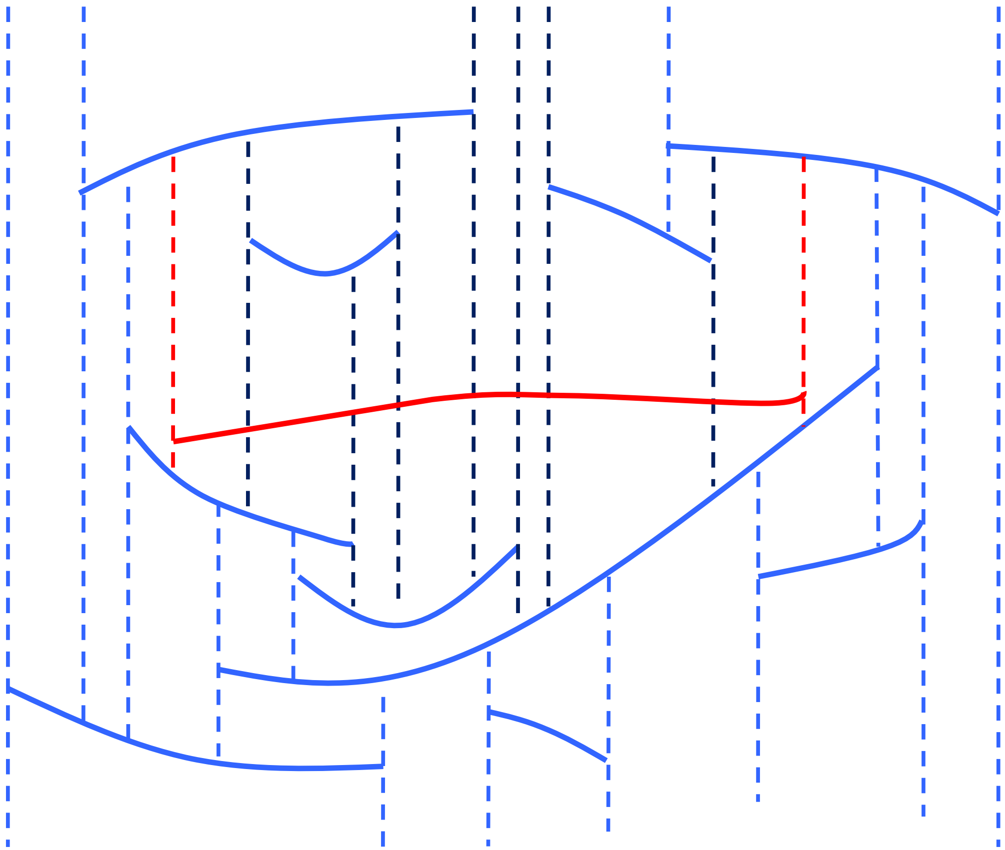
Query for point q :

locate q in G'

1 or 2 comparisons !!

if q in “black” trapezoid then determine correct trapezoid of G
else trapezoid is correct answer already

SAARLAND UNIVERSITY
COMPUTER SCIENCE



trapezoidation G

to obtain **smaller** G'

remove **some** segment
and "retrapezoidalize" hole

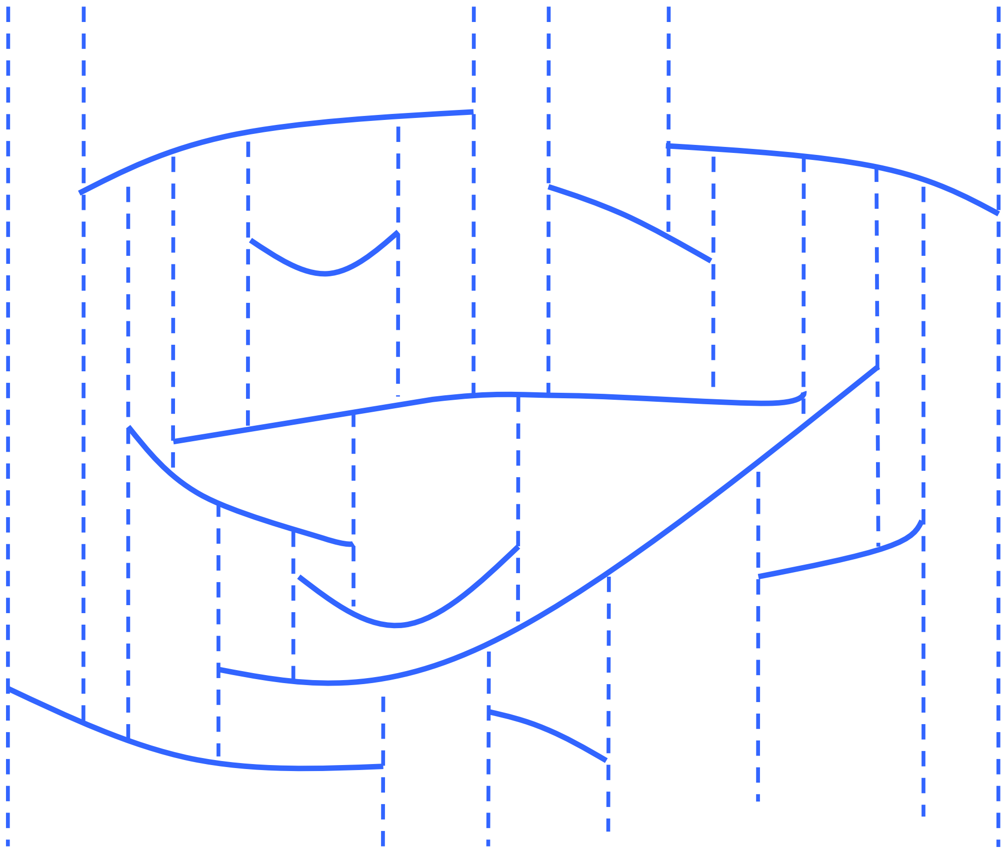
repeat recursively

Query for point q :

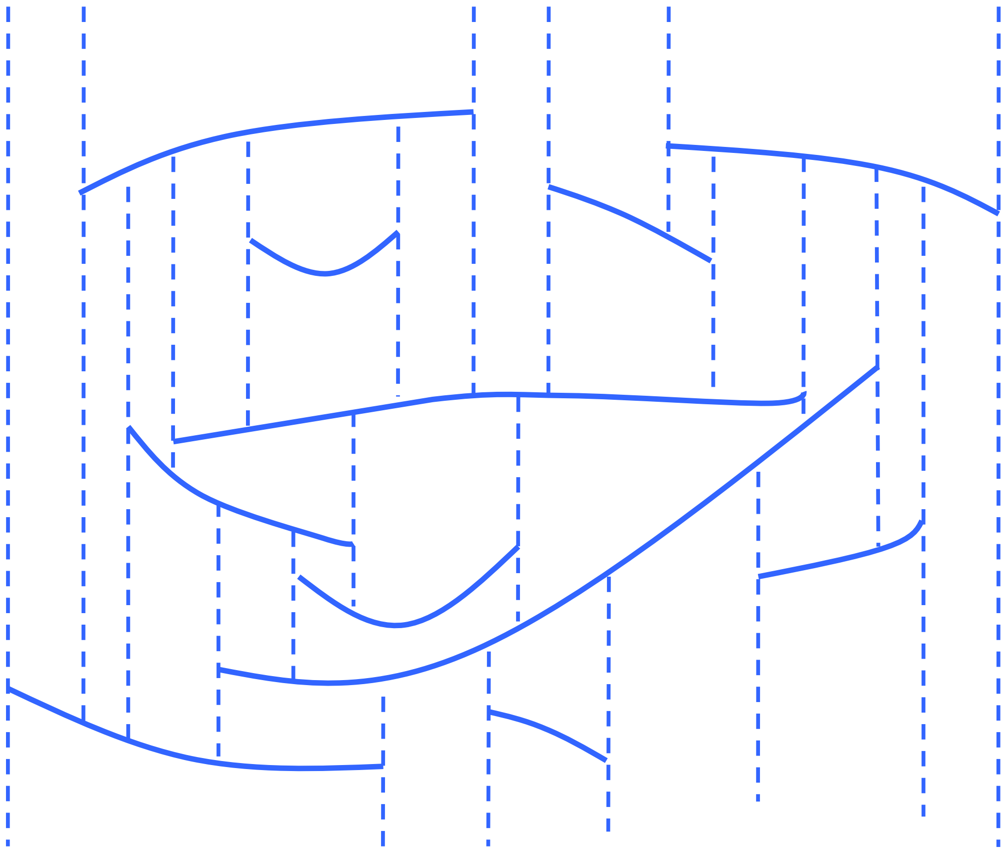
locate q in G'

1 or 2 comparisons !!

if q in "black" trapezoid then determine correct trapezoid of G
else trapezoid is correct answer already

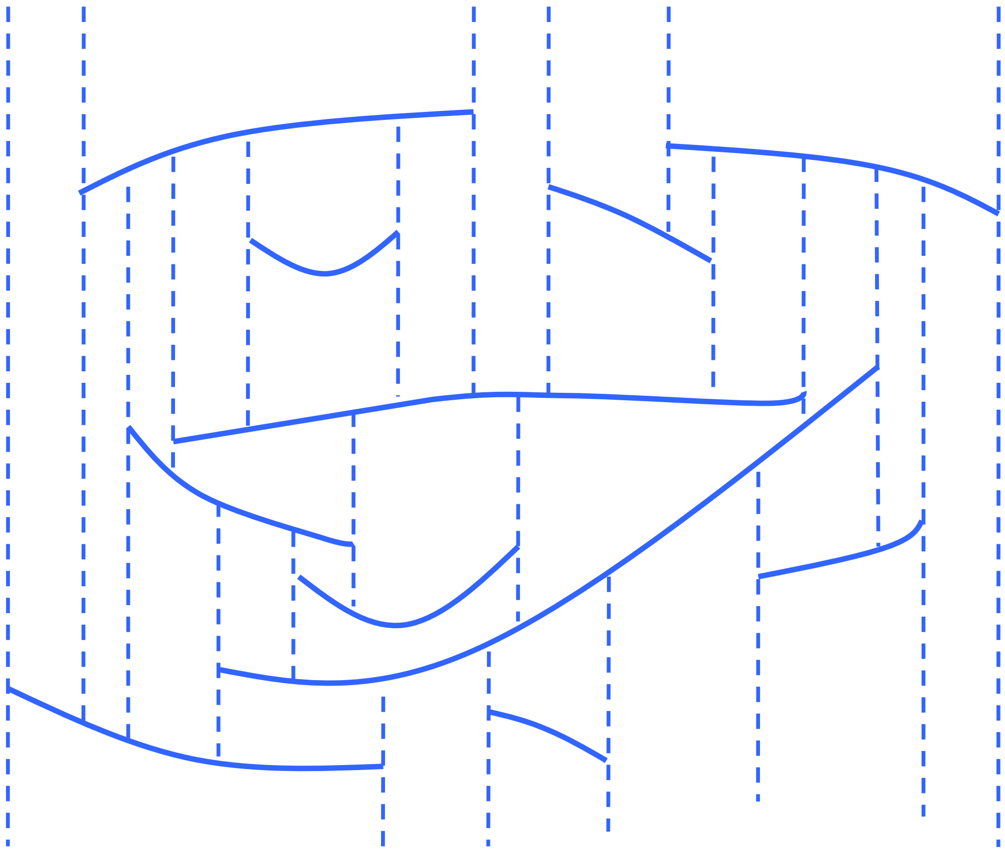


trapezoidation G



trapezoidation G

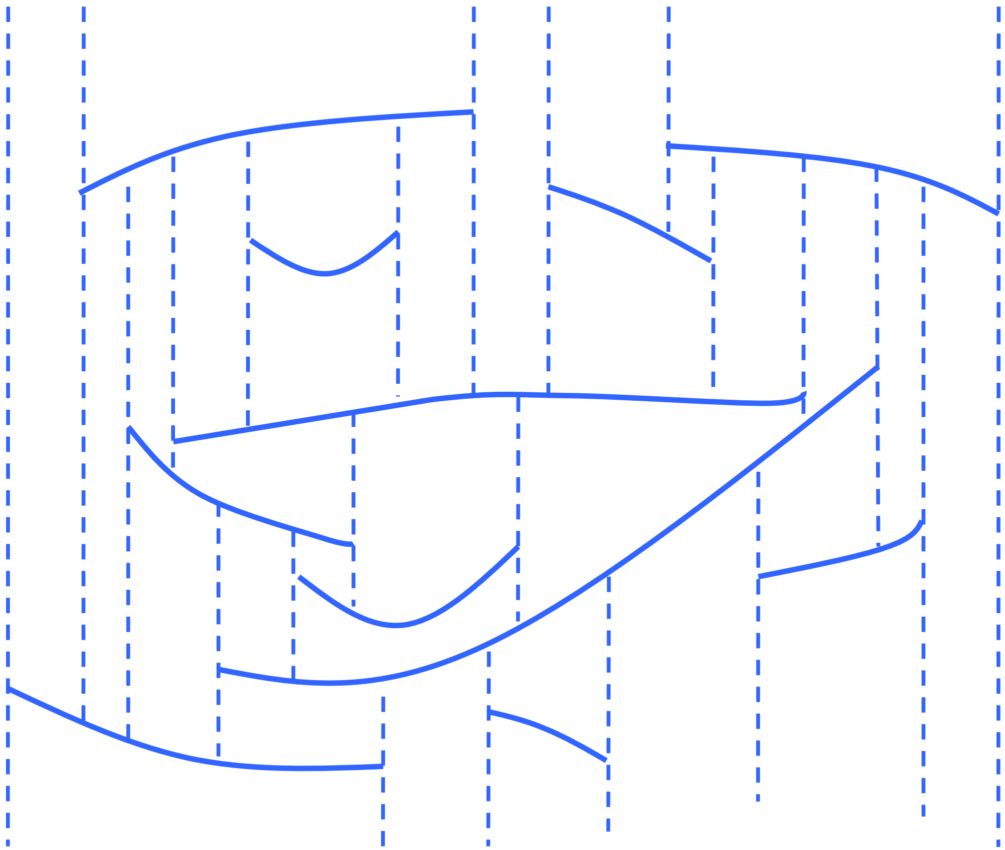
to obtain **<u>smaller</u>** G'



trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

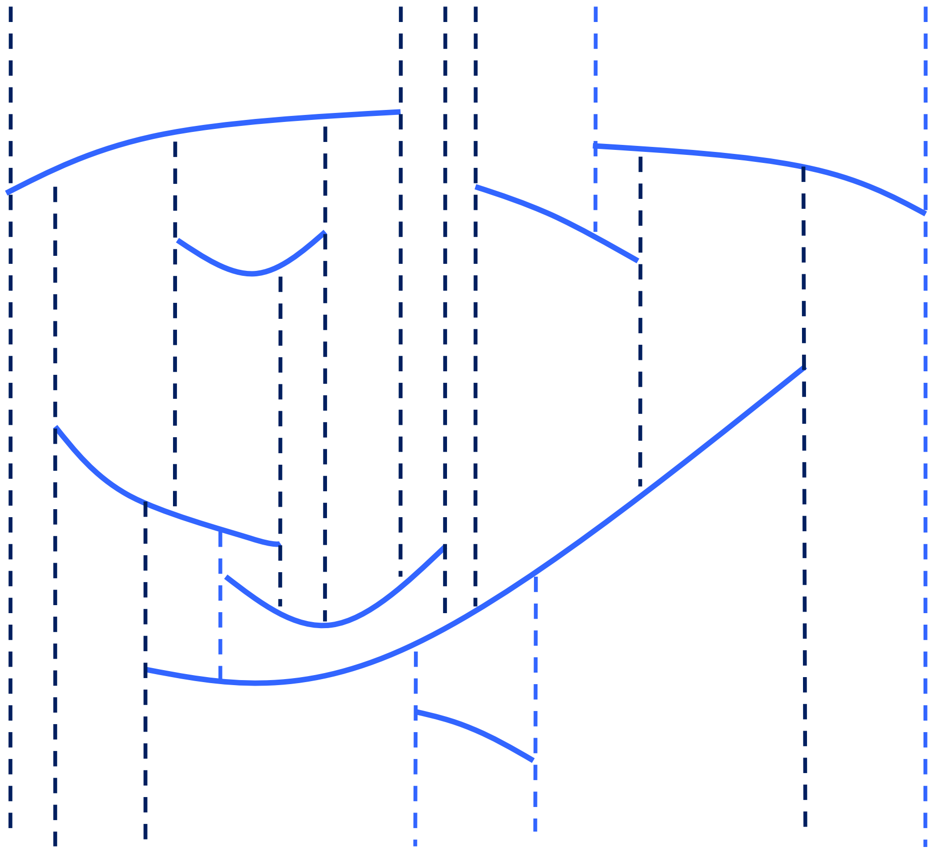


trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

and "retrapezoidalize" holes

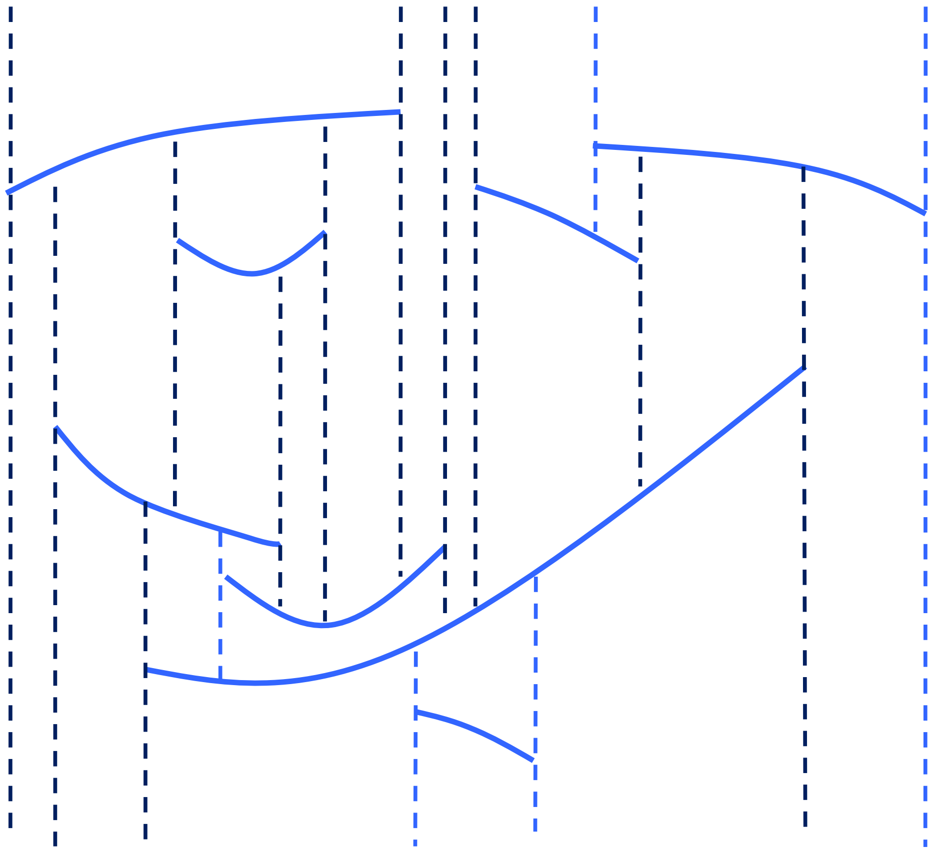


trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

and “retrapezoidalize” holes



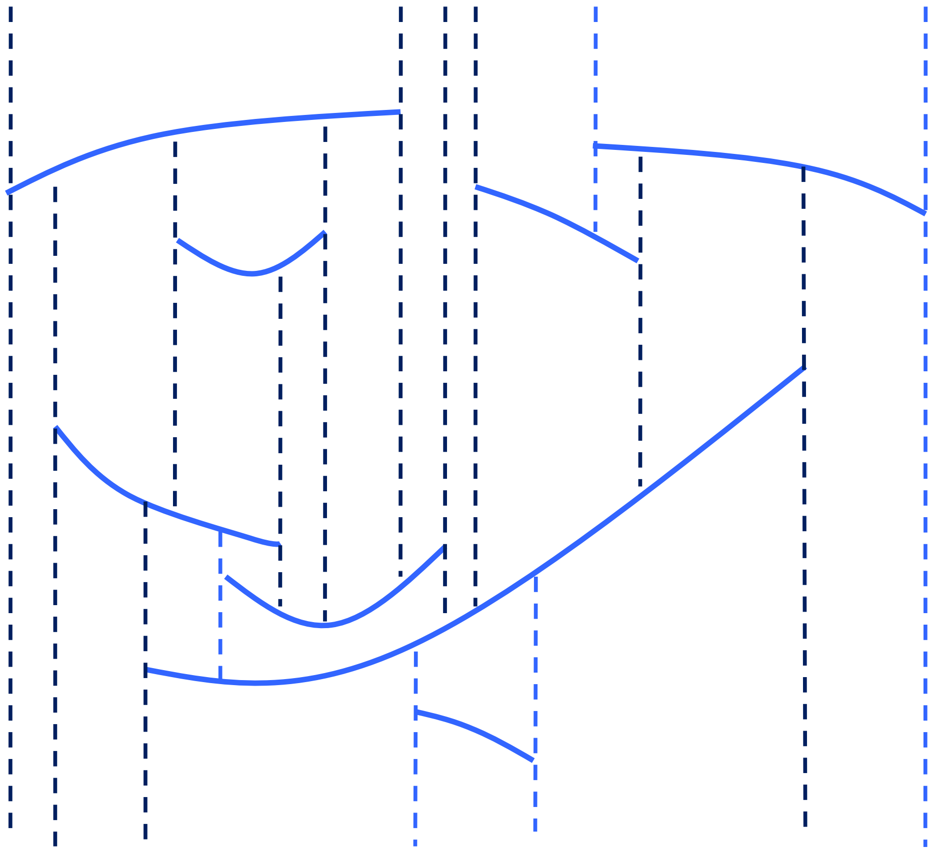
trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

and "retrapezoidalize" holes

repeat recursively



trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

and "retrapezoidalize" holes

repeat recursively

Query for point q :

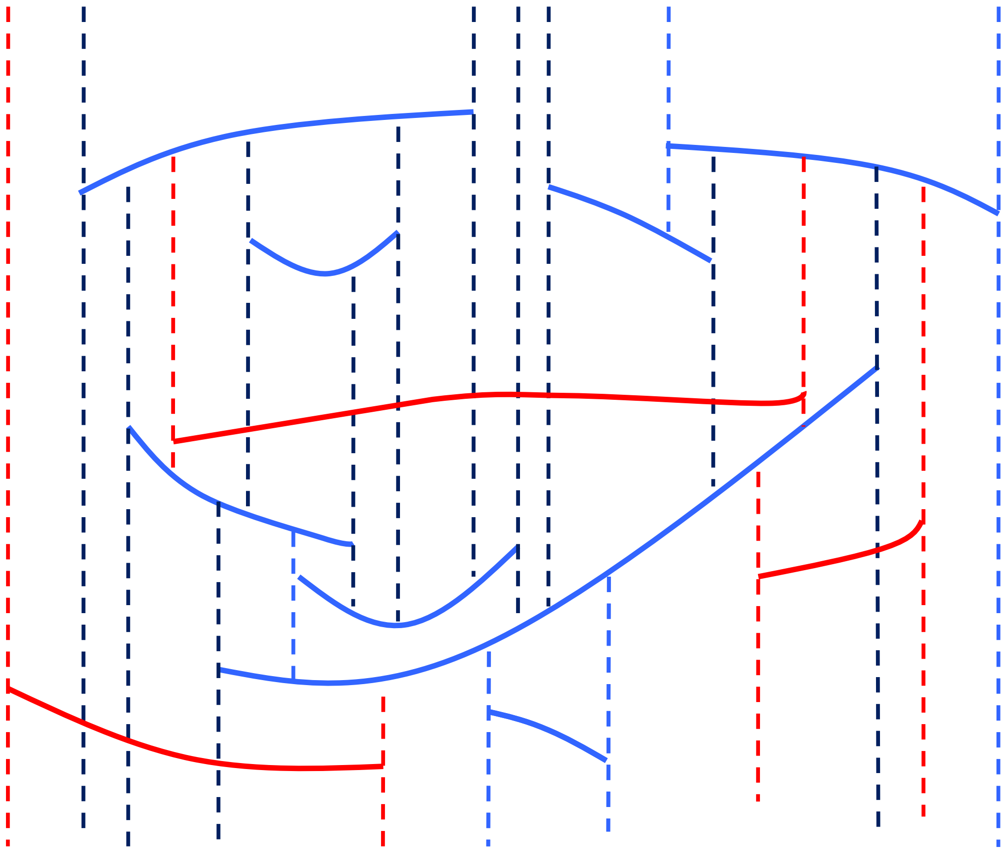
locate q in G'

if q in "black" trapezoid then determine correct trapezoid of G (1 or 2 comparisons !!)

else trapezoid is correct answer already

SAARLAND UNIVERSITY

COMPUTER SCIENCE



trapezoidation G

to obtain **smaller** G'

remove **set of independent segments**

and "retrapezoidalize" holes

repeat recursively

Query for point q :

locate q in G'

1 or 2 comparisons !!

if q in "black" trapezoid then determine correct trapezoid of G

else trapezoid is correct answer already

Lemma 1: In every set of $n \geq 4$ x-monotone segments there exists an "independent" set of size at least $n/4$.

Lemma 1: In every set of $n \geq 4$ x-monotone segments there exists an "independent" set of size at least $n/4$.

Lemma 2: In every set of m "exposed" vertical segments there exists an "independent" set of size at least $m/2$.

Lemma 1: In every set of $n \geq 4$ x-monotone segments there exists an "independent" set of size at least $n/4$.

Lemma 2: In every set of m "exposed" vertical segments there exists an "independent" set of size at least $m/2$.

$\Rightarrow$ height of hierarchy of trapezoidations can be made $O(\log n)$

Lemma 1: In every set of $n \geq 4$ x-monotone segments there exists an "independent" set of size at least $n/4$.

Lemma 2: In every set of m "exposed" vertical segments there exists an "independent" set of size at least $m/2$.

$\Rightarrow$ height of hierarchy of trapezoidations can be made $O(\log n)$

$\Rightarrow$ Query time $O(\log n)$ $\leq 3.5 \log_2 n$

Space $O(n)$

Preprocessing $O(n)$

$O(n \log n)$

Shortcomings of Kirkpatrick's original method:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

Shortcomings of Kirkpatrick's original method:

- ~~only works for straight edge subdivisions~~
- constants are large
- "complicated" (needs to find independent sets)

Shortcomings of Kirkpatrick's original method:

- ~~only works for straight edge subdivisions~~
- ~~constants are large~~
- "complicated" (needs to find independent sets)

Shortcomings of Kirkpatrick’s original method:

- ~~only works for straight edge subdivisions~~
- ~~constants are large~~
- “complicated” (needs to find independent sets)

Use randomization !!!

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Randomized Planar Point Location

Idea: Use single segment removal, but remove a random segment, each with equal probability ($1/n$ for each of the n segments)

$\mathcal{T}(S)$ . . . trapezoidation for segment set S
$\mathcal{Q}(S)$ . . . query structure for segment set S

trapezoids of $\mathcal{T}(S)$ correspond 1-1 with sinks of $\mathcal{Q}(S)$.

Randomized Planar Point Location

Creating $\mathcal{T}(S)$ and $Q(S)$ from S:

1. choose a random s from S, let $S' = S \setminus \{s\}$
2. recursively construct $\mathcal{T}(S')$ and $\mathcal{Q}(S')$
3. use $\mathcal{Q}(S')$ to locate the endpoints a and b of s in $\mathcal{T}(S')$
4. split those two trapezoids verticallly by the vertical lines through a and b respecdtively
5. make the corresponding nodes in $\mathcal{Q}(S')$ to x-comparison nodes (w.r.t. a and b)
6. "Thread" segment s from a to b in $\mathcal{T}(S')$:
7. for each trapezoid cut by s make the corresponding node in $\mathcal{Q}(S')$ to a y-comparison node w.r.t s
8. Generate a sink node of $\mathcal{Q}(S')$ for each new trapezoid in the resulting trapezoidation and connect the newly created y-comparison nodes to the appropriate sink node

Randomized Planar Point Location

1. For each query point q the expected search time for q is $O(\log n)$
2. The expected size of the structures constructed is $O(n)$.
3. The expected preprocessing time is $O(n \log n)$.