CS 164 & CS 266: Computational Geometry

Lecture 11

The locus method, point location, and trapezoidal decomposition

David Eppstein
University of California, Irvine

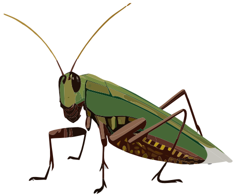
Fall Quarter, 2023

The locus method

The locus method

"Locus": Latin word for "place".

In geometry, used to describe a set of points satisfying some condition.

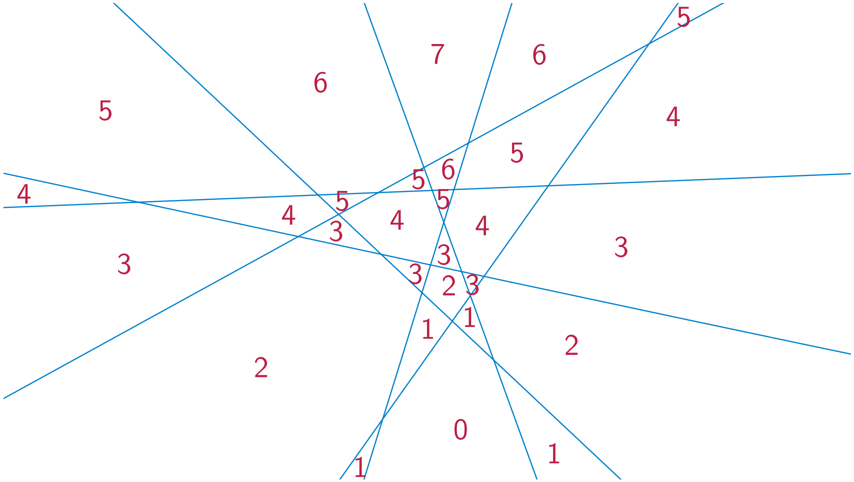

"Locust": kind of insect

General method for building data structures to answer questions about geometric data

- Formulate questions as calls to a "query" subroutine with a point as its argument
- Divide the space of points into regions in which the answer is the same or similar
- Build a "point location" data structure to find the region containing a query point

Example: Lines below a point

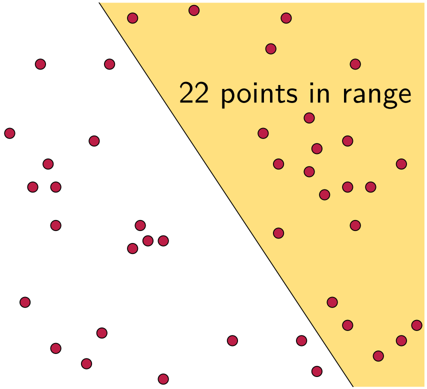
Problem: Given n lines, count lines below each query point



Solution: Construct an arrangement of lines (week 2)
and label each cell with how many lines are below it

Example: Points in a halfplane

Problem: Given n points, count points in a query halfplane



Solution: Projective duality transforms the boundary line of a query into a point and data points into lines; use previous example

Example: Nearest larger value

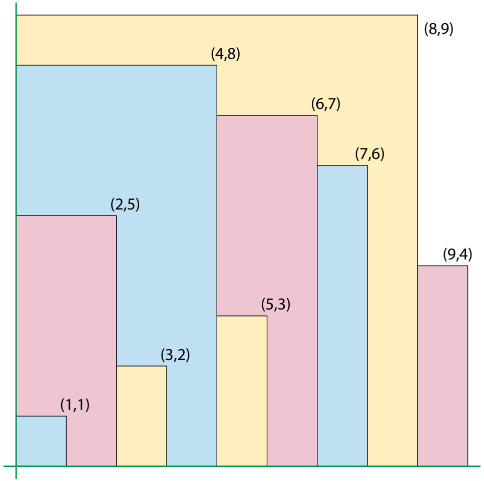
Input: List of numbers

(Here: 1, 5, 2, 8, 7, 6, 9, 4)

Query: Given query x, y, find a number in the list that occurs after position x, has value greater than y, and is as close to position x as possible

Solution: For each input value y in position i, make a quadrant below and to the left of (i, y)

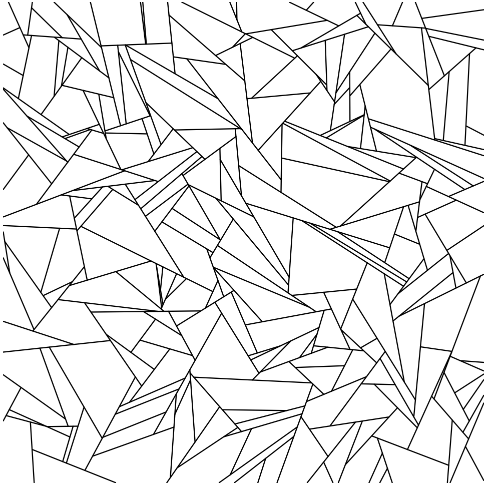
Overlay these quadrants, in left-to-right order



The point location problem

The point location problem

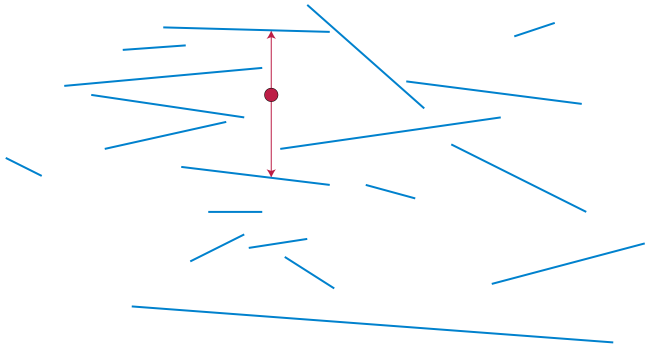
Input: Subdivision of plane by non-crossing line segments



Goal: Data structure that can quickly look up the region containing query point

Vertical ray shooting

Input: Non-crossing and non-vertical line segments,
might not divide the plane into regions



Goal: Look up segments immediately above and below query point

Solving point location by ray shooting

Given a subdivision of the plane:

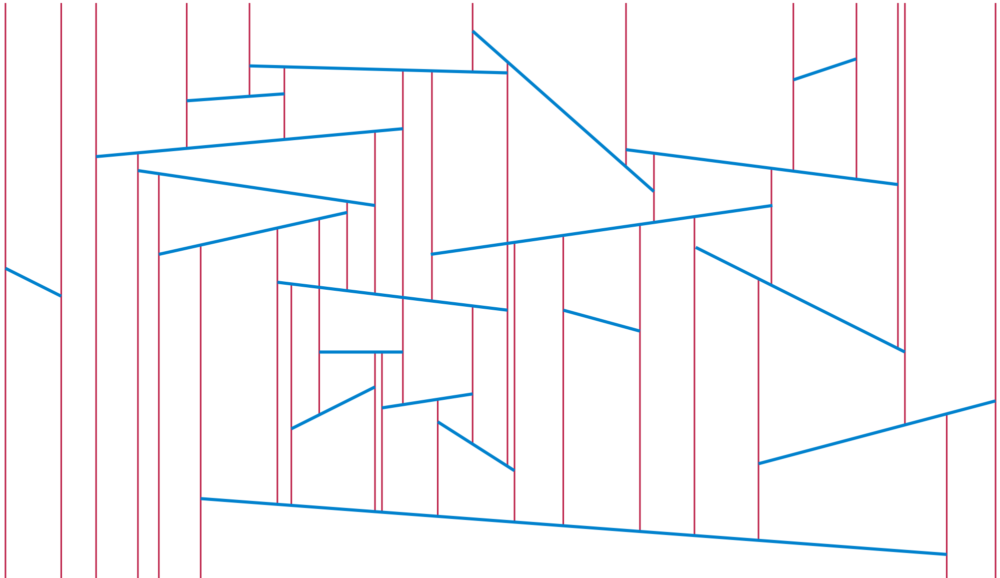
Label each segment by the region below it

To find region containing a query point q:

- Find the line segment s directly above q
- If s does not exist, then q is in the (unique) unbounded region
- Otherwise, look up the region below s, stored as the label for s, and return it as the region that contains q

The locus method for ray shooting

Regions where answer is the same partition the plane into trapezoids, by vertical walls through the endpoints of the segments



Summary (so far)

Many geometric data structures can be split into two steps:

- Construct a partition of the plane (or of a higher dimensional space) so that, in each region, the answer is constant
- Build a point location data structure to find the region containing each query

Point location part can be solved by transforming into ray shooting

But ray shooting can be approached by the locus method $\Rightarrow$ trapezoidal decomposition + point location

We will resolve this loop by finding an algorithm that constructs both the trapezoidal decomposition and a point location data structure in it, simultaneously

Randomized incremental trapezoidal decomposition

Main ideas of algorithm

Add segments one by one in a random order

Because we are constructing a trapezoidal decomposition, not a point location structure for the arrangement of segments, it doesn't matter that they won't meet up to form polygons

Maintain trapezoidal decomposition of the segments added so far

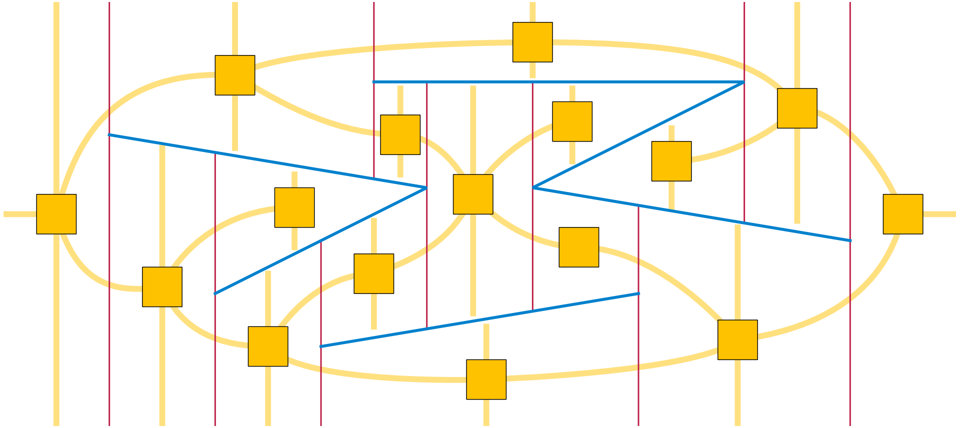
Maintain a point location data structure for the current trapezoidal decomposition to help find where to insert each new segment

When we are done the same structure can also be used for point location or ray shooting for the original input segments

How to represent the trapezoidal decomposition

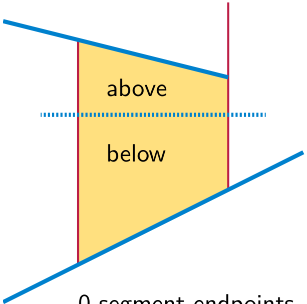
Allow segments to share endpoints but break ties so all endpoints have distinct x-coordinates

Each trapezoid stores: endpoints on left and right walls (or ∞ if unbounded in that direction), segments above and below (or ∞), pointers to ≤ 2 neighboring trapezoids across each wall

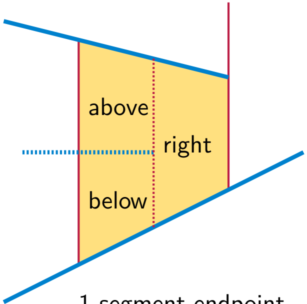


When a new segment is added

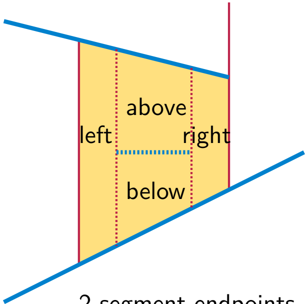
Each trapezoid that it crosses can split into ≤ 4 parts, depending on how many of the segment endpoints belong to the trapezoid



0 segment endpoints in trapezoid



1 segment endpoint in trapezoid



2 segment endpoints in trapezoid

Some of those parts can merge with parts coming from neighboring trapezoids, because the new segment blocks part of an existing wall

To add a new segment

Use the point location data structure (not yet described) to find the trapezoid containing its left endpoint

Use pointers on trapezoid objects to walk left-to-right through the trapezoids finding all trapezoids crossed by the new segment

Construct new trapezoids for each part formed by splitting these crossed trapezoids, following the case analysis on the previous slide

Merge new trapezoids when the wall separating them is blocked by the added segment

Backwards analysis

After i segments have been added, there are $\leq 2i$ vertical walls, with three trapezoids/wall and two walls/trapezoid, so there are at most $3i + 1$ trapezoids

A trapezoid T was just added if and only if the last segment that was added is one of ≤ 4 segments: the ones above it or below it, or the only segment having an endpoint on its left or right wall

For each trapezoid in the current decomposition, the probability that it was just added is $\leq 4/i \Rightarrow$ the expected number of trapezoids that were just added is $(3i + 1) \times 4/i = O(1)$

Summing over all steps in which we add a segment, the total expected number of trapezoids created or destroyed is $O(n)$

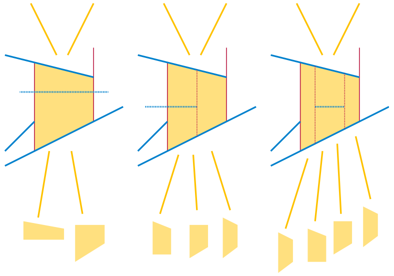
Point location via the history DAG

Main idea

We still need a data structure for finding the trapezoid containing the left endpoint of each added segment

When we break a trapezoid into parts, don't delete it from memory

Instead, flag it as deleted and add links from it to the segment that split it and the ≤ 4 new trapezoids that were formed from it



The result is a directed acyclic graph (not a tree, because of merged trapezoids) whose root is the trapezoid for the empty set of segments (covering the whole plane)

Point location queries

To find the current trapezoid containing a point p:

$T =$ root trapezoid (the one for the whole plane and the empty set of segments)

While T is marked as a deleted trapezoid:

- Find the segment s that caused T to be split
- Compare the x-coordinates of p to the endpoints of s, and test whether p is above or below s, to find which part of T contains p
- Follow the link from T to the trapezoid representing that part

Analysis of point location

For any point p whose location does not depend on the order in which previous trapezoids were added:

After the ith previous segment was added, the probability that it caused a split to the trapezoid containing p is $\leq 4/i$ (same as previous analysis)

If it caused a split, this increases the time for locating p by one more step

Therefore the total expected time for locating p is at most

$$\sum_{i=1}^{n} O(1) \times \frac{4}{i} = O(\log n)$$

(the sum of $1/i$ is the harmonic series)

Summary of analysis

The total expected time for constructing trapezoids and the total expected space for storing the history DAG is $O(n)$

The total expected time for using the history DAG to locate the left endpoint of each new segment is $O(n \log n)$

Once constructed, the history DAG can be used to handle additional point location queries in $O(\log n)$ expected time per query

More careful analysis: with high probability, the longest search path in the history DAG is $O(\log n)$