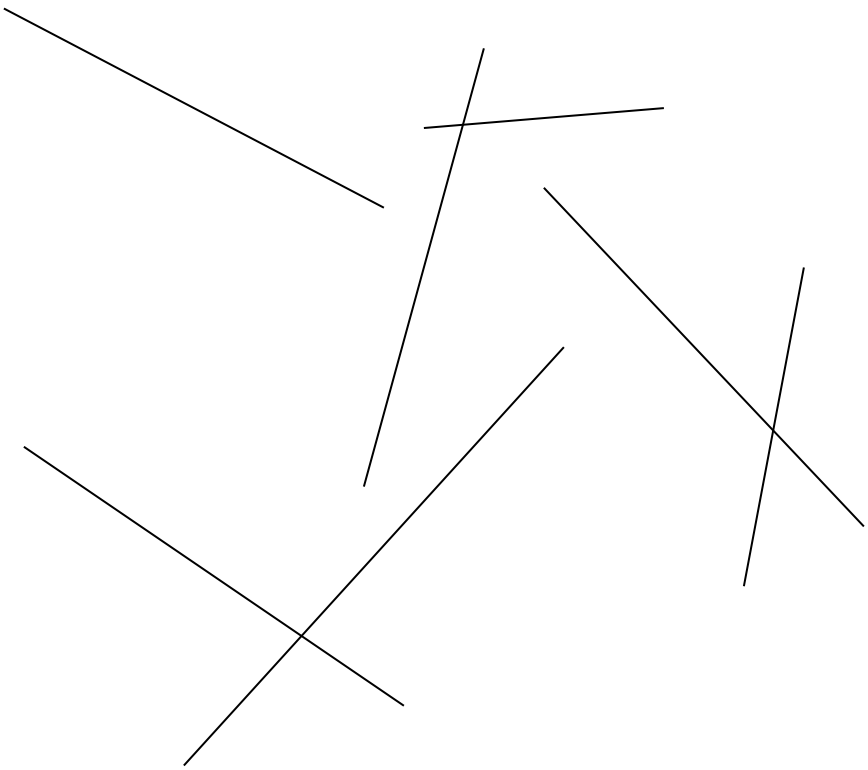
Lecture 2: September 11, 2001

Segment intersection

- Given: a set S of n segments $s_1 \ldots s_n$.
- Goal: find all pairs $s, s' \in S$ of intersecting segments
  (or just detect if any such pair exists)



Motivation

- Collision detection. E.g., make sure that the streets assigned to different rallies do not intersect.
- Map overlay (see Chapter 2 for more details).
- ...

Enables to illustrate *sweep-line technique*:

- an algorithmic paradigm
- sweeping details under the carpet

Naive solution

- Exhaustive search: for each pair of segments, check if they intersect
- $\Theta(n^2)$ running time:
  - worst-case optimal for reporting all intersections
  - ...but usually number of intersecting pairs $P << n^2$, so *output sensitive* algorithm would be better
  - definitely inefficient for detecting existence of an intersection

Simpler problem: 1d case

- Given: n intervals $I_1 \ldots I_n$, $I_j = [l_i, r_j]$
- Goal: report all pairs of intersecting intervals

1d case ctd.

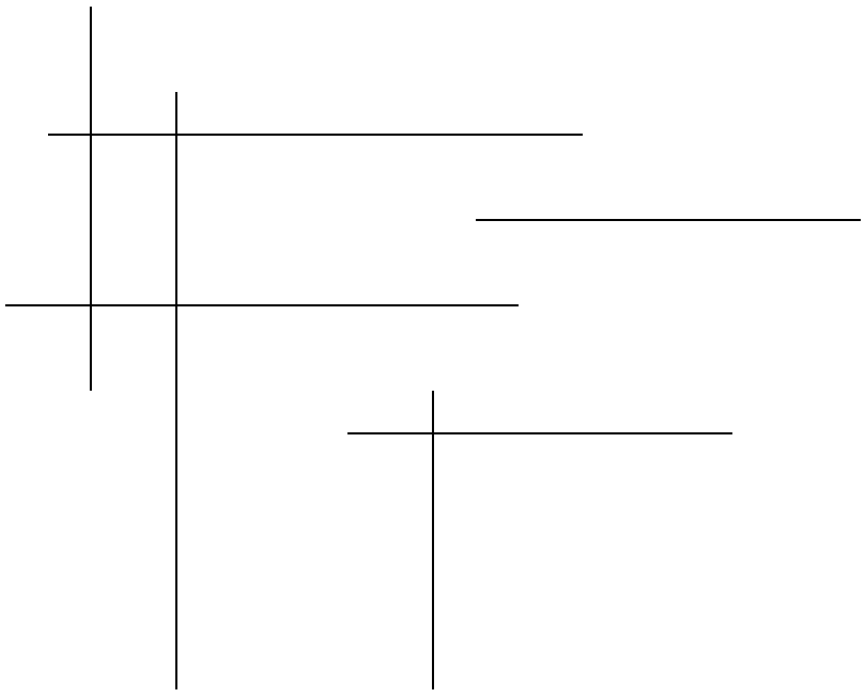
Algorithm:

- Sort all endpoints l_j, r_j
- For each l_j, enumerate consecutive successors of l_j in the sorted order until reaching r_j

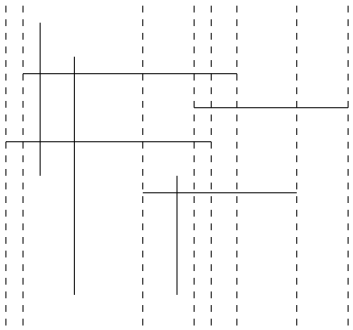
Running time: $O(n \log n + P)$

Orthogonal 2d segments

Assume we have only vertical/horizontal segments.



Algorithm



- Sort the x-coordinates of horizontal segments, inducing vertical "stripes"
- Assign each vertical segment to the corresponding stripe
- "Sweep" the stripes from left to right, and for each stripe:
  - Update the set of y coordinates of the horizontal segments intersecting with the stripe
  - For each vertical segment contained in the stripe, check if it contains any of those y coordinates

Complexity and implementation

- Sorting: $O(n \log n)$
- Assignment of segments to stripes: $O(n)$
- Maintain a dynamic binary search tree enabling $O(\log n)$-time *successor, insert, delete* operations:
  - When moving to the next stripe, update the BST: total cost $O(n \log n)$
  - For each vertical segment, check intersection using the successor operation: total cost $O(n \log n + P)$

Total cost: $O(n \log n + P)$. Can be improved to $O(n \log \log n + P)$ using elaborate techniques.

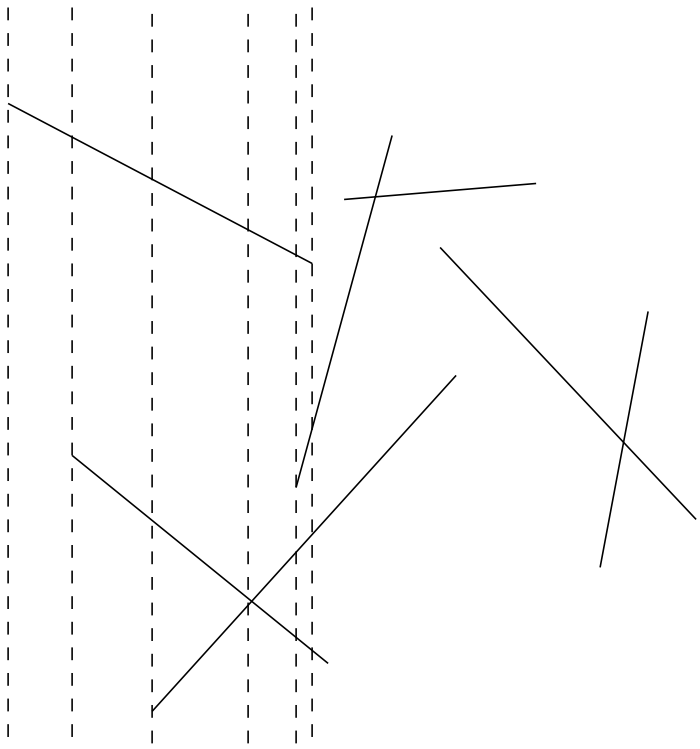
We can also solve the problem of *counting* the number of intersections in $O(n \log n)$ time (as pointed out by a voice in the audience).

Musings

- The algorithm essentially reduces the problem from 2d to 1d; similar technique will be used for range searching later
- Two data structures used:
  - horizontal: the trail of the "sweep line"
  - vertical: BST updated during the sweep

The general case (well, almost)

- Assume no two points (endpoints or intersections) have the same x coordinates
- The horizontal data structure H contains x-coordinates of important points (events) in sorted order:
  - segment endpoints
  - intersection points - cannot determine from the beginning, so must use *dynamic* data structure
- The vertical data structure V maintains the relative order of segments intersecting the sweep line



Details

For each new event p:

1. Handle the event:
   (a) If p is the left endpoint of a segment, add the segment to V
   (b) If p is the right endpoint of a segment, remove the segment from V
   (c) If p is an intersection point of s and s', swap the order of s and s' in V and report p
2. For each segment s with a new neighbor s' in V
   (a) Check if s, s' intersect on the right of the sweep line
   (b) If so, add their intersection point to H

(avoid duplicates in H).

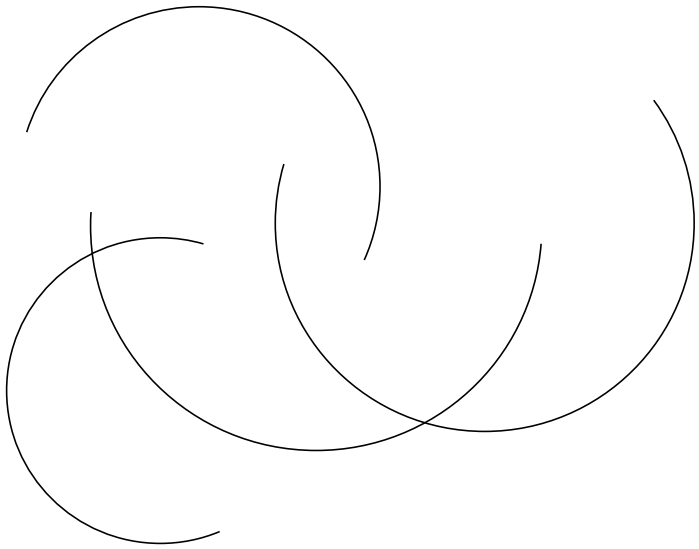
Total complexity: $O((n + P) \log n)$.

Correctness

- All reported intersections are correct.
- Assume there exist an intersection point $p = (x, y)$ not reported by the algorithm. Take the first such intersection (of s, s').
- There must have been a time $x' < x$ when s and s' were neighbors on a sweep line at position x'.
- Since all events before x were handled correctly by the algorithm, s and s' must have been neighbors in V at some point before x.
- Therefore, their intersection should have been reported.

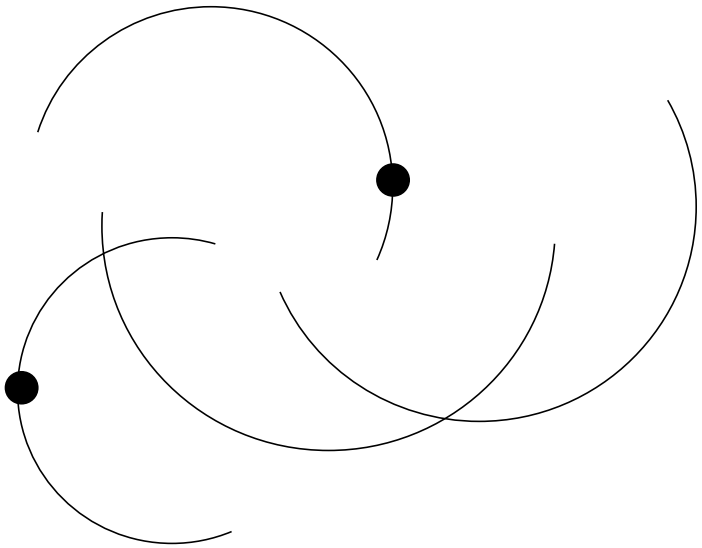
Bonus

What if the segments are not straight ?



Curves

For one, need to decompose into monotone arcs:



The algorithms works !
Axioms

- curves are x-monotone (so that we know where they begin and end)
- curves can be "computationally handled", e.g., we can compute intersection points of two curves in $O(1)$ time
- curves are "smooth"

Summing up

- Segment (or curve) intersections can be found in $O((n + P) \log n)$ time
- Can be improved to $O(n \log n + P)$ (quite difficult)
- Main tool: sweep-line approach, also used for triangulation (Chapter 3) and Voronoi diagrams (Chapter 7)