Intersection detection and crossing listing
Intersection detection problem

Input: A list of line segments (each one: 4 coords of 2 endpoints)

Output: Do any two cross? If so report a crossing (or maybe all)

Application: Test validity of geographic data or circuit designs
Naïve solution

For each pair of input segments:
  test whether they cross \[ O(n^2) \]

Last week:

- Test for crossing using four left-right tests on triples of points \[ O(1) \]
- Find line through two points, and point where lines cross, via projective geometry

But can we do better?
(Unrealistic) general position assumptions

- No two segment endpoints have the same $x$-coordinate
  (Implies: No vertical segments)
- No endpoint lies on another segment
- No three segments cross at a single point

likely to occur
probably ok

should be reported
as being crossings
Plane sweep algorithms
Plane sweep

General approach for designing geometric algorithms

sweep vertical line left-to-right

processed input → unprocessed input

The combinatorial structure of the result will only change at a finite set of discrete “event points” — process these in left-to-right order.
Crossed segment data structure

As sweep line sweeps left-to-right across the input, maintain:

- Set of line segments that cross it
- Vertical ordering of these line segments in a binary search tree

Vertical ordering will help us find crossings when we sweep over them, because it changes at those points.

We can maintain it efficiently using a balanced binary search tree.

Search tree operations only need to know the ordering of the segments, not the precise coordinates of the points where they cross the sweep line.
How the vertical ordering can change

cross left endpoint of segment — add to set of segments crossed by sweep line
cross right endpoint of segment — remove from segments crossed by sweep line
sweep over crossing point — swap in vertical order of crossed segments
Event queue data structure

Keep track of future event points (where vertical ordering changes) in a priority queue, prioritized by $x$-coordinate (position in the left-to-right ordering used by the sweep line).

For crossing detection, we only need sorted list of segment endpoints.

For listing all crossing, we also include the crossing points that we have found so far.

In some algorithms we may also include “potential” events that we think might happen, but that can be removed from the event queue before they actually happen.
Crossing detection pseudocode

- Initialize $T$ to an empty binary search tree
- Initialize $Q$ to be a sorted list of segment endpoints
- For each point $p$ in $Q$:
  - If $p$ is a left endpoint of a segment $S$: Add $S$ to $T$, and check for crossings between $S$ and its neighbors above and below it in the vertical crossing order (found using $T$)
  - Else $p$ is a right endpoint of a segment $S$; remove $S$ from $T$, and check for crossings between the two segments that were above and below it in the vertical crossing order

If we ever find a crossing, stop the whole algorithm and report it

If any two segments cross, they will be adjacent just before the sweep line sweeps over the crossing, and this algorithm will check them and discover the crossing
Listing all crossings

- Initialize $T$ to an empty binary search tree
- Initialize $Q$ to a priority queue of points, prioritized by $x$-coordinate, initially containing all segment endpoints
- While $Q$ is non-empty:
  - Find the minimum-priority point $p$ in $Q$ and remove it from $Q$
  - If $p$ is a left endpoint of a segment $S$: Add $S$ to $T$, and check for crossings between $S$ and its neighbors above and below it in the vertical crossing order (found using $T$)
  - Else if $p$ is a right endpoint of a segment $S$: remove $S$ from $T$, and check for crossings between the two segments that were above and below it in the vertical crossing order
  - Else $p$ is a crossing point of two segments; swap the segments in $T$, and check for crossings between them and the two segments above and below them in $T$

Whenever we find a new crossing point, just insert it into $Q$
Analysis (of both algorithms)

Let $n$ be the number of segments (so there are $2n$ endpoints) Let $k$ be the number of crossing points; $0 \leq k \leq \binom{n}{2}$.

The detection algorithm has $2n$ events; the crossing listing algorithm has $2n + k$.

Each event is performed using a constant number of operations in binary search trees and priority queues.

Total time: $O(n \log n)$ for detection, $O((n + k) \log n)$ for crossing listing.
Arrangements and their representation
Arrangement

Think of any system of segments or curves as barriers to motion. “Cell”: 2d region within which you can get between any two points.

How to find and represent this system of cells and their boundaries?
Some terminology

**Cell**
2d connected region

**Edge**
1d boundary of two cells, separated by part of a segment
Same cell can be on both sides

**Vertex**
An endpoint of a segment, or crossing point of segments
At an intersection point, multiple edges come together

**Flag**
A vertex, edge, and cell that all touch each other
Most representations are centered on the **edges** of an arrangement.

Each edge touches two vertices at its ends, and two cells on its two sides.

There are representations with:

- one object per edge (pointing to all four of these things)
- two objects per edge (one for each of its two sides)
- four objects per edge (one per flag)

**Structure from our text:** two objects per edge
Half-edges

Represent each edge of the arrangement by two directed edges ("half-edges"), like the two directions of a two-way road (But like in England or Japan where they drive on the left)
Doubly-connected edge list

Object-oriented, with objects for vertices, half-edges, and faces

Each half-edge stores:
- Pointer to twin half-edge from same edge
- The vertex it comes from (Can find other vertex from twin)
- The face on its side of the edge
- The next and previous half-edges in the cycle around its face

Each vertex stores:
- Its coordinates
- One of the half-edges it touches

Each face stores:
- A half-edge on its outer boundary
- A list of half-edges, one for each internal boundary
Constructing the arrangement of line segments

Same plane sweep algorithm, maintaining DCEL of the part of the arrangement to the left of the sweep line

Updates to DCEL at sweep events:

- When we sweep over left endpoint of a line segment, use search tree of segments crossed by sweep line to find its cell, start a new interior boundary inside the cell, with a vertex and two half-edges
- When we sweep over a right endpoint, close off twin half-edges and (if they separated two cells) merge into a single cell
- When we sweep over a crossing, make a vertex, close off four half-edges, and start four more half-edges, with a new cell between them