Line Segment Intersection

Given a set $S$ of $n$ line segments in the plane

We'll consider these in detail:

- Do any two segments in $S$ intersect?
- Which pairs of segments intersect?
- Find all points of intersection
- Subdivide $S$ into subsegments that intersect only at endpoints

Arrangement of $S$

Examples:

Given a polygon, is it simple?

Given two polygons, compute their union/intersection

Given two planar maps, compute their overlay

Variants:

Replace segments with boxes, circles, curves, triangles in $\mathbb{R}^3$...

Elementary Cases:

Two segments $pq$ and $rs$ cross if they intersect at a single interior point

$pq$ crosses $rs$ and $pq$ crosses $rs$

We can compute $pq \cap rs$ by solving linear equations

$$\text{orient}(p,q,z) = 0$$
$$\text{orient}(r,s,z) = 0$$
$$\begin{align*}
(b-d)x + (c-g)y &= ad-bc \\
(f-h)x + (g-d)y &= ch-fg
\end{align*}$$

We saw this on Tuesday!
Not all intersections are crossings!
Still detectable in $O(1)$ time but more cases to consider

So we can find all intersections in $O(n^2)$ time by brute force.

Intersection Detection

Given $n$ line segments, do any two intersect?

[Shamos Hoey 1976] $O(\log n)$ time sweep algorithm

Imagine a vertical line $l$ sweeping continuously from left to right.
Maintain sequence of intersections between $l$ and $S$
Watch for crossings between segments

Continuous $\rightarrow$ discrete:
Maintain sequence of segments (indices) crossing $l$, not actual intersection points

Sequence changes only when $l$ reaches
- segment endpoint
- segment crossing

We know these in advance
Always between adjacent pair
Maintain sweep-line status in balanced binary search tree (AVL, red-black, treap, splay, skip list, ...)

Insert/Delete/Pred/Succ in $O(\log n)$ time

Consider endpoints in order from left to right

$O(\log n)$ time to sort

At left endpt $p_i$

\[
\begin{align*}
\hat{i} &< \text{pred}(p_j) \\
\text{if } p_i, q_i \text{ and } p_j, q_j \text{ cross}, \text{return True}
\end{align*}
\]

\[
\begin{align*}
\hat{k} &< \text{succ}(p_j) \\
\text{if } p_i, q_i \text{ and } p_k, q_k \text{ cross}, \text{return True}
\end{align*}
\]

Insert $j$ into BST (using $p_j$ for comparisons)

$O(\log n)$ time per endpoint $\Rightarrow O(\log n)$ time

If we reach last endpt with no crossings

\[\text{return False}\]
Reporting all intersecting pairs \([\text{Bentley, Ottmann}]\)

Keep a priority queue of potential crossings

For each adjacent pair \(ij\) in sweep line sequence

store \(Pi_i \cap Pj_j\)

priority = \(x\)-coord
but ignore if left of \(l\)

Use the priority queue to find next event
(see with endpoints of \(S\))

---

**At left endpoint \(p_i\)**

\(i \leftarrow \text{pred}(j)\)

\(k \leftarrow \text{succ}(j)\)

insert \(j\) into BST

delete \((ik)\) from PQ

insert \((ij)\) into PQ

insert \((jk)\) into PQ

---

**At right endpoint \(q_j\)**

\(i \leftarrow \text{pred}(j)\)

\(k \leftarrow \text{succ}(j)\)

delete \(j\) from BST

insert \((ik)\) into PQ

delete \((ij)\) from PQ

delete \((jk)\) from PQ

---

**At crossing \(S_j \cap S_k\)**

report crossing

\(i \leftarrow \text{pred}(j)\)

\(k \leftarrow \text{succ}(k)\)

swap \(j, k\) in BST

delete \((ij)\) and \((k, l)\) from PQ

insert \((i, k)\) and \((j, l)\) into PQ
\(O(\log n)\) time per event

\#events = \#endpts + \#crossing pairs

\(O((n+k) \log n)\) time

\(k=\text{output size} = \#\text{intersecting pairs}\)

But what if we only want to output the arrangement?

Multiple coincident events:

- Previous algorithm implicitly perturbs segments into general position.
- \(k\) segments meeting at one point \(\Rightarrow \leq \binom{k}{2}\) events

Perturbation depends on how the priority queue breaks ties.

We want this to be one event!

**Modify priority queue**:

- Contains endpoints and crossing points
- Priority = \(x\)-coord, ties broken by \(y\)-coord
- Each PQ record stores set of all segments known to contain that point

Initialize with endpoints

We already need a way to detect duplicates in PQ

Use balanced BST as PQ, Insert, Find, Delete Min all in \(O(\log n)\) time
Operations:

Insert Crossing \((i, j)\) — Record a crossing between \(s_i\) and \(s_j\) if right of sweep line

\[ O(\log n) \text{ time} \]

\[ \text{Compute intersection point } L(x, y) \]

\[ \text{Find event record for } (x, y) \quad \text{if none found} \]

\[ \text{Add } i \text{ and } j \text{ to } X \text{.segments} \]

ExtractMin \(\rightarrow\) event record \(X\)

\[ \text{Split } X \text{.segments into } \text{Left, Right, Thru,} \]

\[ \text{Delete Left from BST} \]

\[ \text{Delete Thru} \]

\[ \text{Insert Thru — reversing order} \]

\[ \text{Insert Right} \]

\[ \text{Update output graph data structure} \]

\[ O(d \log n) \text{ time if vertex has degree } d \]

Every BST delete & insert calls InsertEvent, at most twice

Overall running time is \(O(n \log n) + \sum_{v} \deg(v) \cdot O(\log n)\)

\[ = O(n \log n) + O(E \log n) \]

\[ = O(V \log n) \]

\[^{\uparrow} \text{output edges } \geq n \]

\[^{\uparrow} \text{output vertices} \]

by Euler's formula: \(E = 3V - 6\)