Notes on Computational Geometry
Chapter 2: Line Segment Intersection

Andy Pavlosky

Motivating example: Geographic information systems split maps into several layers, each containing one type of data, and compute the intersections of data in these maps (e.g. intersection of roads and rivers).

First problem formulation: Given a set S of segments in the plane, compute all points where segments in S intersect.

The brute force approach for this is $O(n^2)$, which is only efficient if there are lots of intersections. In general, we can do better.

Idea: if the orthogonal projections of segments onto the y-axis don't intersect, then the lines themselves can't intersect. So only test lines whose projections intersect.

Definition 1. *An algorithm that uses this idea is called a **plane sweep** algorithm (imagine sweeping a horizontal line l, called the **sweep line** down the plane). The set of segments intersecting the sweep line is called the **status**. The status must be updated only at certain points, called **event points**.*

We also want to ensure segments are close on the x-axis when we test them. To do this, only test segments adjacent in the x-direction on the sweep line.

Handling each type of event point:

1. Upper endpoints: The new segment added to the status must be tested for intersection with its neighbors.
2. Intersection points: The two segments which intersect change order, and get (at most) one new neighbor to be tested against.
3. Lower endpoints: The neighbors of the line which is remove from the status are now neighbors and must be tested.

Our algorithm will store the event points in the **event queue** $\mathcal{Q}$, which is a balanced binary search tree where where $p > q$ iff $p_y > q_y$ or $p_y = q_y$ and $p_x < q_x$. We also store the segments intersecting the sweep line in the **status structure** $\mathcal{T}$, which is a binary search tree with the segments intersecting the sweep line in its leaves. At a high level, the algorithm to compute these intersections does the following:

- Insert the segment endpoints into $\mathcal{Q}$.
- Handle the next event point as described above and delete it from the queue.
- If an intersection is found, report it and add it to the event queue if it's not already in it.

The approach described runs in $O(n \log n + I \log n)$ time, where n is the number of segments and I is the number of intersections.

Second problem formulation: We want to compute the overlap of regions in the plane. We'll represent one of these regions as an area enclosed by edges of a graph.

Terminology:

- A **face** is an open polygonal region whose boundary consists of edges and vertices.
- The **complexity** of a subdivision is the sum of the number of its vertices, edges, and faces.
- If a vertex is the endpoint of an edge, they are **incident** (similar for faces and edges, and faces and vertices).

The **doubly-connected edge list** will store information about our region and let us perform operations such as the following:

- Walk around the boundary of a given face
- Access an adjacent face
- Visit all edges around a vertex

It accomplishes this by storing collections of the following information:

- Vertex records, which will store each vertex's coordinates and an arbitrary incident edge.
- Face records, which will store some half-edge on each face's boundary, and a list of arbitrary half-edges for each hole.
- Half-edge records, which will store each half-edge's origin, its twin, its incident face, and its next and prev half-edges.

Now for two of these subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$, we will compute their overlay $\mathcal{O}(S_1, S_2)$. This is essentially the subdivision created by the edges of both $\mathcal{S}_1$ and $\mathcal{S}_2$. Our goal is to compute a doubly-connected edge list for $\mathcal{O}(S_1, S_2)$.

The book describes an $O(n \log n + k \log n)$ approach to this problem, where n is the total number of vertices, and k is the complexity of the output. This allows us to compute the union, intersection, and difference of subdivisions of the plane in quasi-linear time.