Comp 163: Computational Geometry
Tufts University, Spring 2005

Professor Diane Souvaine
Scribe: Kevin Keating

Line Segment Intersection Using a Sweep Line Algorithm

1 Introduction

Given a set of n line segments, how would you compute all intersections between these segments? To naively check each line segment for an intersection with every other line segment would require $O(n^2)$ time. However, this may do unnecessary work, as it is possible that many segments do not intersect. It will also return the intersections in unsorted order

An alternative approach is to use a *plane sweep* algorithm. Such an algorithm moves a line across the plane to find intersection points. Imagine the *sweep line* moving rightward across the plane. All intersections to the left of the sweep line have already been detected. The *status* of this sweep line is the set of segments currently intersecting it. This status changes as the sweep line moves to the right. Any time the sweep line passes over an end point of a segment or an intersection point, the sweep line stops and updates this status. Thus, these points are called the *stopping points*. Because the stopping points are processed in sorted order, the intersections are found in sorted order.

We are concerned with finding intersections of line segments currently on the sweep line. However, two line segments can not intersect unless they are next to each other. Thus, we keep track of the vertical ordering of the line segments and only test neighboring segments for intersection.

2 Overview

Given: n line segments defined by $2n$ endpoints.

Goal: Compute all intersections.

Status Data Structure:

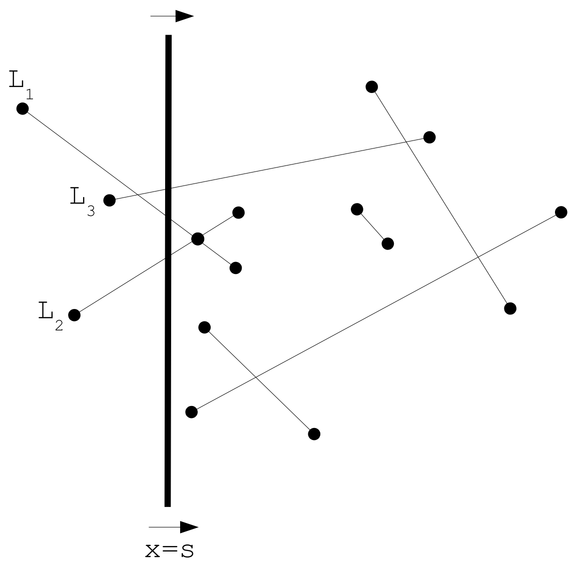

Figure 1: A sweep line at $x = s$.

This stores the cross section of the arrangement of line segments intersecting the sweep line at $x = s$ for some value of s. The invariant is that all intersections to the left of $x = s$ have already been computed and reported. s is initialized at $-\infty$, where the status data structure contains no line segments.

Stopping Points Data Structure:
This data structure contains start and end points of all line segments that are to the right of $x = s$. In addition, it contains intersection points to the right of the sweep line for line segments which are currently adjacent.

3 Data Structure Specifics

3.1 Status Data Structure

This structure stores the slope, y-intercept, start point, and end point for each line segment on the status line. The segments are ordered by their vertical position (their y-coordinate at $x = s$). It is implemented as a balanced binary search tree allowing:

- Insertions in $O(\log n)$ time
- Deletions in $O(\log n)$ time
- Finding the neighbors of a segment in $O(\log n)$ time

In addition, when two segments are no longer neighbors, their intersection point must be removed from the stopping points data structure. To quickly locate any such intersection points, there is a pointer from the segments in the status data structure to their intersection point in the stopping points data structure.

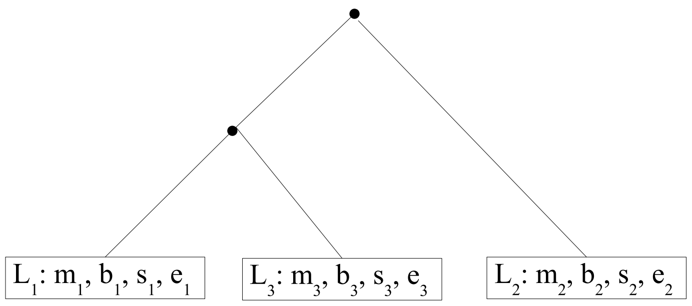
L_1: m_1, b_1, s_1, e_1
L_3: m_3, b_3, s_3, e_3
L_2: m_2, b_2, s_2, e_2

Figure 2: The status data structure stores slope, y-intercept, start point, and end point for each line on the status line.

3.2 Stopping Points Data Structure

This structure stores all stopping points, as well as indicating the type of stopping point: segment start point, segment ending point, and intersection point. This structure uses a priority queue, implemented as a heap, allowing:

- The point with the minimum x-coordinate to be removed in $O(\log n)$ time.
- A new point to be inserted in $O(\log n)$ time.
- An intersection point to be deleted in $O(\log n)$ time. This occurs when two line segments are no longer neighbors.

- The structure to be initialized with all segment start and end points. This can be done in $O(n)$ time if the heap is built bottom up.

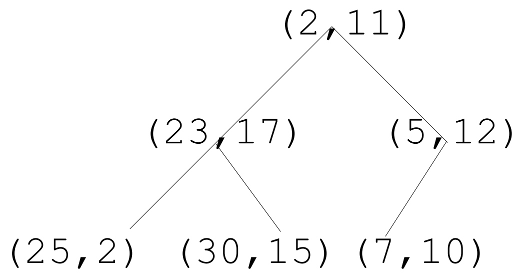

Figure 3: The stopping points data structure implemented as a heap.

4 Plane Sweep

To perform the plane sweep, remove and process the next event point from the stopping points data structure.

At a line segment starting point:

- Do binary search to insert the segment into the status data structure.
- Notify the neighbors that they are no longer adjacent and delete their intersection point (if any) from the stopping points data structure.
- Compute the intersections of this segment with its neighbors (if any) and insert those into the stopping points data structure.

At a line segment ending point:

- Delete the segment from the status data structure.
- Notify the neighbors of the deleted segment that they are now adjacent. Compute their intersection point (if any) and insert it into the stopping points data structure.

At an intersection point:

- Output the point.
- Swap the positions of the intersecting segments in the status data structure.
- Notify the new neighbors of the swapped segments. Insert and delete intersection points from the stopping points data structure as needed.

The algorithm finishes when the stopping points data structure is empty. It outputs all intersection points sorted by x-coordinate.

5 Complexity

5.1 Naive Algorithm

Brute force calculation of all intersections requires $O(n^2)$ space and $O(n^2)$ time. If the output must be later sorted, then the total runtime will be $O(n^2 \log n)$

5.2 Space Complexity

The status data structure contains at most one line for each of the n lines in the problem, so it is $O(n)$. In the stopping points data structure, there can be n starting points and n end points. In addition, because only adjacent lines can have intersection points, there can be at most $n - 1$ intersection points. Thus, stopping points is also $O(n)$. Therefore, the algorithm has $O(n)$ space complexity.

5.3 Time Complexity

The plane sweep algorithm is output sensitive. That is, the running time depends on the number of intersection points in the graph. Let k equal the number of intersection points in the graph. Plane sweep runs in $O((n + k) \log n)$ time, while the naive algorithm would require $O(n^2)$. Thus, plane sweep normally results in a faster runtime. However, k can be as big as $O(n^2)$ if all line segments intersect, resulting in a run time of $O(n^2 \log n)$.

The run time is computed as follows:

The stopping points data structure can be initialized in $O(n)$ time, as the heap is built bottom up. The status data structure is initialized in constant time, as there are no line segments at $x = -\infty$.

Each point in the stopping points data structure must be processed. This includes one starting and one stopping point for each line segment, as well as one point for each intersection point. Thus, there are $O(n + k)$ points that must be processed.

As shown above, finding the next stopping point can be found in $O(\log n)$. This point can be either a starting point, an ending point, or an intersection point.

Starting points can be processed in $O(\log n)$ time:

- The new segment is inserted into the status data structure in $O(\log n)$ time.
- The neighbors of the new segment are no longer adjacent, so their insertion point is deleted. Because there is a pointer from the segments in the status data structure to the intersection point in the stopping points data structure, finding the intersection point takes constant time. Deleting this point takes $O(\log n)$ time.
- The intersection points of the new segment with its neighbors are computed and inserted into the stopping points data structure. These insertions take $O(\log n)$ time each.

Stopping point can be processed in $O(\log n)$ time:

- The segment is deleted from the status data structure in $O(\log n)$ time.
- The intersection point of the neighbors of the deleted segment is inserted into the stopping points data structure in $O(\log n)$ time.

Intersection point can be processed in $O(\log n)$ time:

- The segments are swapped in the status data structure in $O(\log n)$ time.
- Up to two intersection points may be deleted and up to two intersection points may be added to the status data structure. This is done in $O(\log n)$ time.

Thus, any point may be processed in $O(\log n)$ time. As there are $O(n+k)$ points, the overall algorithm runs in $O((n+k)\log n)$ time.

6 Lines

The plane sweep algorithm can be modified to calculate the intersections of lines, rather than line segments.

The status data structure must now be initialized at $x = -\infty$ with all lines in order of decreasing slope. Because lines never need to be inserted or deleted, this data structure may be implemented as an array.

The stopping points data structure no longer contains starting or end points. Instead, it must be initialized with intersections points of lines adjacent at $x = -\infty$.

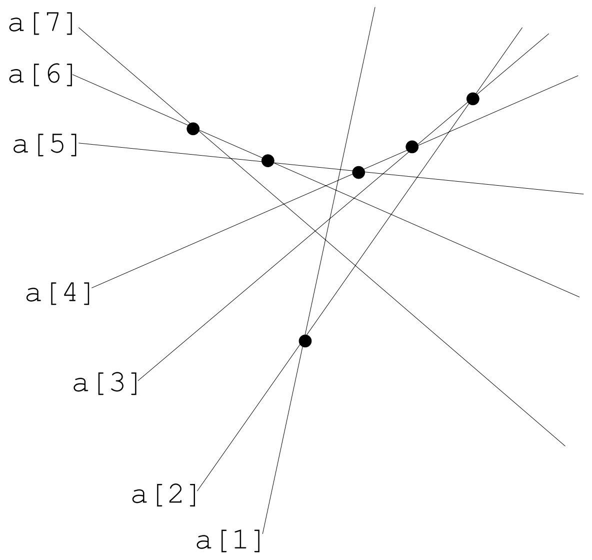

Figure 4: An arrangement of lines and the intersections of neighbors