COMPUTATIONAL GEOMETRY

PROJECT REPORT

Problem Description:

Implement the Bentley-Ottmann sweep to DETECT if a polygonal chain C = (p1, p2, . . . , pn) is simple (it is not simple if it properly crosses itself, at a point interior to two edges, or if it crosses itself at a vertex, in the degenerate case). Report a witness to the crossing if the chain is not simple.
The input is assumed to be either graphical (e.g., allowing the user to mouse-click enter points) or read from a simple text file, with each line consisting of two floating point numbers (the x- and y-coordinates of a vertex) separated by spaces. You should animate the algorithm to show each step during the sweep.
Conduct an experiment to determine the running time in practice for large values of n, for randomly generated inputs. Keep track also of the largest size of the SLS.

Algorithm:

The Bentley–Ottmann algorithm is a sweep line algorithm for listing all crossings in a set of line segments, i.e. it finds the intersection points of line segments. The main idea of the algorithm is to have a sweep line moving from top to bottom. We arrange all the end points in a y-sorted order. We use a variable named as 'sweep line status' which is a binary search tree to keep track of all the line segments in an x-sorted order and we use a variable 'Event Queue' which is a priority queue to determine which is the point to consider next.
When the sweep line touches an endpoint in the sorted list, it could mean 3 things:

1. The point is the start point of the line segment. In this case we insert this line segment into the Sweep line Status. We find the predecessor and successor of this line segment in the binary search tree and check if the current line segment intersects any of them.
2. The point is the end point of the line segment. In this case we remove this line segment from the Sweep line Status. We find the line segments which became adjacent as a result of the removal of this line segment and verify if those line segments intersects.
3. The point is an intersection point. In this case, we swap the positions of the two line segments which caused the intersection points. Now we check if the swap of positions caused any intersections with the new predecessors and successors of the swapped lines.

In all the above steps if there is an intersection discovered, we add it to the Event queue.

In the DETECT mode, we only need to report the first intersection. In this case we do not need to store all the intersections in the priority queue. We can stop the algorithm once it finds the first intersection point. So in this case, the event queue could be just a list.

Now moving on to the problem description, this is a small twist to the original Bentley Ottman sweep. In this problem we need to DETECT if a polygonal chain is simple or not. A polygonal

chain is not considered simple if it crosses itself, at a point interior to two edges, or if it crosses itself at a vertex, in the degenerate case.
Since this is a DETECT problem, we could have the Event Queue to be just a list. Since this is a chain the adjacent line segments would intersect at the end points. Keeping in view of this, I have changed the algorithm in the following way.
The Sweep Line Status is still a binary search tree. The event queue is a list. When the sweep line touches an endpoint in the sorted list, it could mean 3 things:

1. The point is a vertex where 2 line segments start. In this case we insert both the line segments into the sweep line status and check if each of them has intersections with it's predecessor and successor.
2. The point is a vertex where 2 line segments finish. In this case we remove both the line segments from the sweep line status and verify if there are intersections in the new predecessor and successor.

If any intersections were found, we can stop the algorithm since this is run in DETECT mode.
To find the predecessor and successor of a segment in a binary search tree, I find the in-order predecessor and in-order successor. To check for intersections, I use left tests and on-segment tests. To find the intersection point of 2 line segments, I use the determinant approach.
The points are inserted as a linked list so that each point would know which is the next point after it and which is the point before it.

The program has been animated to visualize how the sweep line moves in this algorithm. The end points of each line segment and the intersection point if found is also highlighted. The sweep line status is shown at the top of the program which would change dynamically.

The complexity of the algorithm will be nlogn. This is because there are a maximum of n + 1 end points for n line segments and each of these points would contribute to either insertion or deletion from the sweep line status which is a logn operation. So totally the complexity is O(n log n).

Experiments:

The algorithm was tested on polygonal chains with increasing number of vertices. The results are as follows:

Number of Vertices	Time Taken (Sec)
20	0.00935
40	0.01608
60	0.01933
80	0.0227
120	0.045

Conclusion:

As part of the future work for this project, a REPORT mode could be implemented. It could also be good to highlight which are the line segments currently considered by the sweep line when it is sweeping down.