Advanced Topics in Algorithms
Pratical Project

Pedro Moreno
Rui Fonseca

April 4, 2017

1 Horizontal Partition

1.1 First Approach

The first algorithm developed aimed to compare each new horizontal edge found against the sweep line and detect which of the 8 possible cases was happening. First of all, we sort all a list with all the vertex of the polygon by their Y, then their X, using Bubble Sort taking $O(n^2)$. At each step, we consider all the vertices that have the same Y.

The sweep line will contain the information of what we already know about the polygon in each step. First it is an empty list, but when a section of the polygon starts, its up edges that define the section will be added. The edges will be added taking order into account, taking O(n) time complexity. It is true that we will have always a pair number of edges, and that if we divide the list in pairs of two, the space between the edges of a pair is inside the polygon and that the space between pairs of edges is outside the polygon.

We will iterate through all the points and check if it forms an horizontal edge with the neighbor points that we are considering. If an horizontal edge is detected, we first detect if there are any more consecutive collinear points that form a bigger edge. Then we iterate through the pairs of the sweep line until the matching case is encountered. In the worst case we iterate through all the pairs of the sweep line until the end which has at most n/4 edges pairs, and constant time to find if there is a match. When we find the segmentation cords that need to trace in constant time. We update the sweep line immediately (taking O(n) time) except in cases 2 and 4, were we only update it at the end of the step. This was need in order to use only the sweep line to find the segmentation points, thus only using constant time.

We also considered the case where a point can be collinear in same continuous line of the polygon so those cases were also treated.

There was another difficulty in inserting new vertex in the polygon (as the result of segmentation) and maintaining the sweep line up to date, so new verticies will only be added at the end of

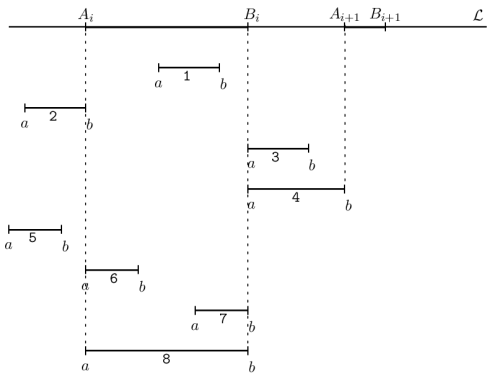

Figure 1: The eight possible cases against the sweep line. Reference [Tom04]

the step.

Also, as segmentation cords would mess with the detection of connected vertices on the same line, the segmentation cords are also only added at the end of the step.

To summarize this algorithm is limited by $O(n^3)$ time.

In terms of space, this algorithm will need space for:

- The Vertex List: contains n vertex pointers.
- The sweep line: will contain at most n/2 edge pointers.
- The sweep line event list: will contain at most n/4 event pointers.
- The Insert vertex list: will contain at most n/4 vertex pointers.
- The Segmentation vertex list: will contain at most n/2 vertex pointers. D

All in all, the additional space needed (excluding the dcel) is O(n).

1.2 Second Approach

In our second approach we will also use the sweep line and do steps by each horizontal line, where we consider all the vertex on the same Y. Again, the first step is to order the vertex list by their Y coordinate. Using bubble sort, it takes $O(n^2)$ time.

For each step we first insert all the new up edges that are formed from the vertices being considered into the sweep line. This inserts at most n/2 edges, the maximum number of vertices on the same step.

Then, for each point, we will try to do two operations: trace left and trace right.
For the left trace we first check if an left edge already exists and if the left face is out. This takes constant time using the dcel. Then, we find the the edge in the sweep line that will have the intersection by finding the edge with the greatest X smaller than the X coordinate of the point. This procedure can last at most n/2 iterations. This process is mirrored for the right trace.

Finally, we remove from the sweep line the edges that ended in that step.

Some changes to the insert vertex function allowed to save some changes on the sweep line.

This algorithm takes $O(n^2)$ time complexity to run.

The space complexity needed by this algorithm is O(n):

- The vertex list, containing n vertices pointers.
- The sweep line, containing at most n/2 edge pointers.

1.3 Update Faces

After the Horizontal partition is done, we update the faces that were created with the segmentation edges.

This is done by first adding all the edges in the dcel to a list. Then, choose one edge and remove all the next ones until it completes the circuit of the face. Those edges are removed from the list and the process is repeated.

This procedure takes $O(n^2)$ time complexity because removing from the list takes O(n) time complexity.

2 Grid Partition

In the grid partition we take the horizontal partition and apply a very similar method to the second approach. The list of vertices is ordered by X using bubble sort. Then we consider all the vertices in the same vertical line. New edges that start at that line are added to the sweep line.

Then, for each point, we will run two methods: trace up and trace down. These methods behave in the same way as trace right and left except that when we insert a new edge, the same method is recursively on the destination edge. This way the method will trace segmentation edges until it encounters the out face or an edge already exists. This recursion adds at most n to the time complexity. The edges that end in the vertical line are removed from the sweep line.

This algorithm has $O(n^3)$ time complexity and the uses the same space as the second approach, so O(n) space complexity.

3 Improvements to data structures and sort

The time complexity of the algoithms could be improved if:

1. We use heap sort instead of bubble sort to order the vertices list.
2. We use a balanced tree (e.g. avl tree, red black tree, ...) instead of a link list for the sweep line.

Using this data structures and algorithms, the vertex sort will take O(n log n) time complexity, and the sweep line search and insertions will take O(log n) instead of O(n) time complexity.

This means that the first approach will have $O(n^2 \log n)$ time complexity and that the second approach will take O(n log n) time complexity.

The grid algorithm complexity bound could be improved to $O(n^2 \log n)$ if we use a balanced data structure for the sweep line, instead of a linked list.

4 Vertex Visibility

The algorithm developed follows the theorem that a face is only totally visible to a vertex if that vertex can see every vertex that the face contains.

- We receive a list with all out edges and another with all faces. The list with the out edges is found in $O(n^2)$ and the faces list is calculated while we do the partition.
- Iterate through all the available faces.
- In each face, we iterate through every vertex that it contains.
- In each vertex, we check if the line segment from the original vertex to that vertex intersects with any on the edges on the out edges list.

If any of the points in a face is not visible then that means that face is not visible. This algorithm has a time complexity of $O(n^3)$.

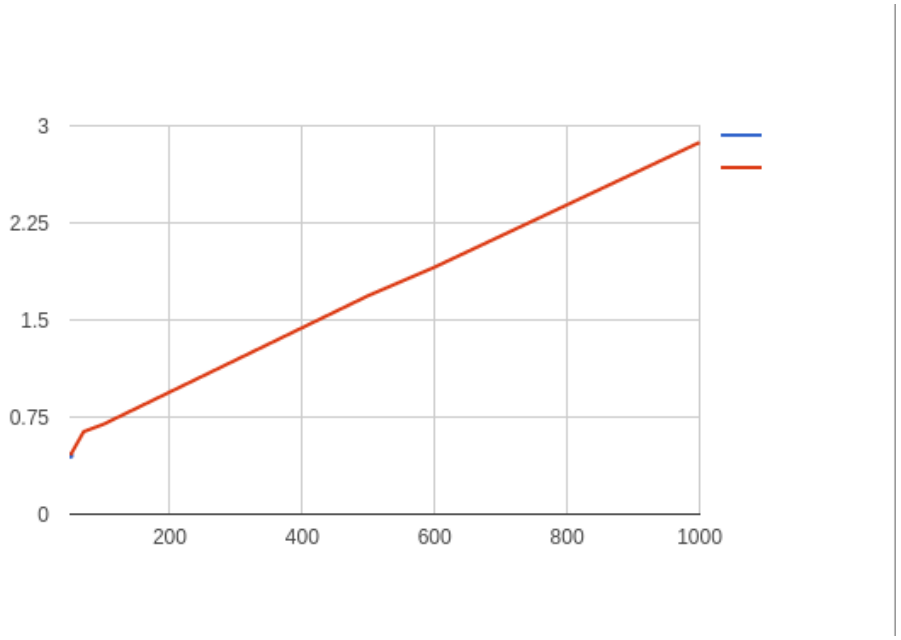

Figure 2: The 4th root of time by input

5 Minimum Vertex Guard

To find the solution for the minimum vertex guard problem, we find the visibility matrix for each vertex. That is, we construct a matrix where the rows are the faces and the columns are the vertices. A value in the matrix is 1 if the vertex can see that face, 0 otherwise. This procedure has a time complexity of $O(n^4)$.

Next, we solve the problem using glpk library, the simplex optimizer and integer optimizer. The objective is set to minimize and the defined variable k is set as a lower bound to the rows.

6 Results

input size	50	70	100	500	600	1000
time	0.043	0.165	233	8.13	13.26	67.999
$\sqrt[4]{time}$	0.4553728292	0.6373397212	0.694766677	1.68858387	1.90825268	2.871611154

References

[Tom04] Ana Paula Tomás. Análise de alguns problemas geométricos em poligonos ortogonais. *Technical Report Series: DCC-04-03*, page 4, 2004.