

Regularizing a Planar Subdivision

1 Introduction

Motivation: Non-monotonic polygons can be hard to do computations with. Thus, we attempt to regularize such polygons by creating monotonic sub-regions inside the original simple polygon. This may be accomplished in $O(n \log n)$ time.

2 Algorithm:

2.1 General Idea:

The algorithm uses a sweepline approach that sweeps the polygon horizontally and adding edges between vertices that introduce edges that makes the region non-monotonic to create monotonic sub-divisions inside the non-monotonic polygon. We realize that at every vertex of the polygon, we need process the point and determine whether to split the current region into sub-divisions. This leads to the need of the Stopping Points data structure to keep track of where the next "splitting" decision is located and the Status data structure to add and remove potential stopping points throughout the algorithm.

The algorithm keeps track of the information of the resulting sub-regions that could enable easier computation of various purposes such as point inclusion in $O(\log n)$ querying time by further regulating the sub-regions in to triangular regions.

2.2 Data Structure:

2.2.1 Status

The status data structure keeps track of the adjacency status of edges in the current x-coordinate position of the polygon using a BBST of edges. With

every update of the Status data structure, two lines could become new neighbors of each other. Edges could also stop neighboring each other when edge reaches it's ending point early.

Each edge that has the interior of the polygon directly above it also keeps a pointer to the point closest to the sweep line between this interior edge besides its own vertex, which is the edge to it's right in the Status data structure. We call this point its helper. (This is used later in the section on critical points).

2.2.2 Stopping Points

To sweep across the polygon and process the necessary procedures, which we realize occurs with the encountering of every vertex. Thus we need a stopping points data structure that represents the points we need to execute sub-procedures on in the order that we sweep across the plane. With these points, pointers to the edges they are directly connected to should also be kept. During each step of processing a stopping point, their edges' status in the Status data structure is updated.

2.2.3 Result Data Structure

The resulting regularized polygon of the monotonic subdivisions may be kept in a Doubly-Connected-Linked-List, which has the following properties:

1. Any vertex has access to one edge that it is on.
2. Any face has access to one edge that surrounds it.
3. Any edge has access to 2 vertices and the face to it's right and the face to it's left.

2.3 Algorithm Description:

1. We begin by inserting all vertices into the Stopping Point data structure. Then with every stopping point:
2. There are three possible classifications of points:

- (a) The point is the starting point of 2 edges and thus introduces multiple edges into the status data structure.
 - i. In this case, if the stopping point occurs inside the polygon, this point is a critical point and breaks the monotonicity of the polygon. In this case, we connect the point to the helper of the line to the left in status where the critical point is found
- (b) The point is the ending point of one edge and the starting point of another edge. In this case, the point replaces one edge with another.
 - i. This is not a critical point. We don't need to perform any fix ups, so we simply update the Status data structure.
- (c) The point is the ending point of 2 edges and thus takes 2 edges out of the status data structure.
 - i. This is also a critical point that requires the addition of another edge. However, we won't be able to easily connect it to a helper. Instead, whenever replacing a helper pointer that was not the endpoint of the line itself, we can connect the previous and new values of helper, we will connect the critical points together.

There are $O(n)$ stopping points. For every stopping point, we search and update the Status data structure, which takes $O(\log n)$ time as there will never be more than n edges in the status data structure, one for each edge in the simple polygon. The Stopping Point data structure will have at most n points, each corresponding to a vertex of the polygon. Therefore, the algorithm will be overall $O(n \log n)$ in time and $O(n)$ in space.