LAB 4: Incremental triangulation

Joseba Sierra
21/11/2020

To compute and save the triangulation, a DCEL (doubly connected edge list) will be used. The variant implemented uses **2 half edges for every edge**, each one going against the other. For instance if we look at the following figure 1

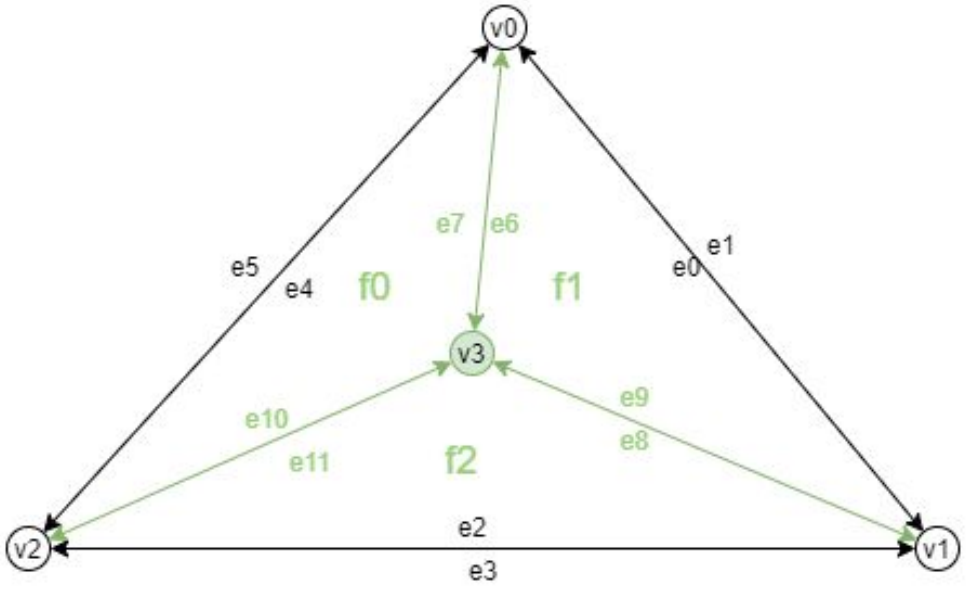

Figure 1: Standard case when adding a new point

when adding vertex v3 to an existent triangulation, we need to add the new half edges e6,e7,e8,e9,e10,e11, faces f0,f1,f2 and update the new state of the old edges e0,e2,e4 (the new face they're pointing and the next edge of each one).

In this implementation, a face belongs to a half edge if it's on its right, so, in the image above, those edges with their names inside a face represent a half edge oriented in such a way that have the face on its right side.

We also need to take care of the degenerate case, where a new the point falls on an edge:

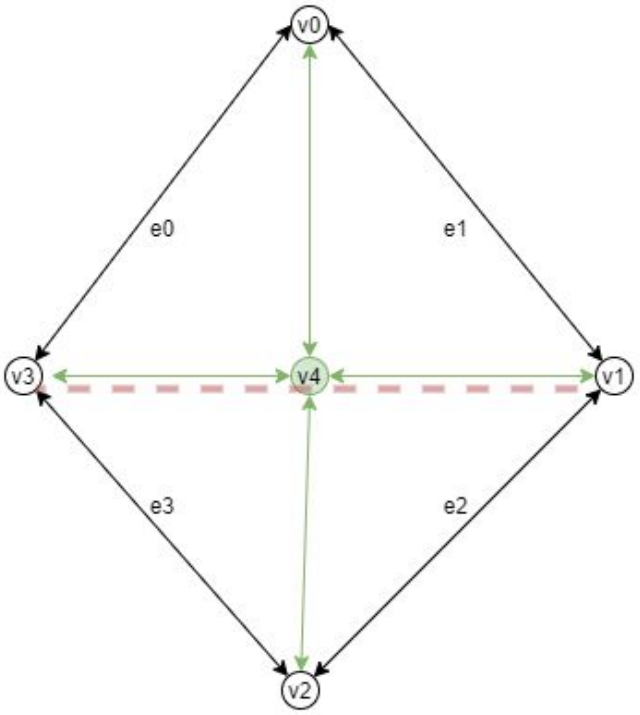

Figure 2: Degenerate case, point falls on an edge

In this case, we need to remove the edge where the point falls (red edge), and add the edges represented in green together with the new created faces. Then we need to update the new state of e0,e1,e2,e3.

Once we know this, we only need to **initialize the DCEL** with a correct state, and **identify the face** or edge where a new point falls.

(If we have a case where we receive an existent point, we add it as another new vertex but don't triangulate it).

DCEL initialization:

To initialize the DCEL we compute an enclosing triangle of all the points, to do this, the bounding box is computed, and a triangle that enclose this obtained bounding box is chosen. Then we initialize the DCEL data with the correct values of this selected triangle.

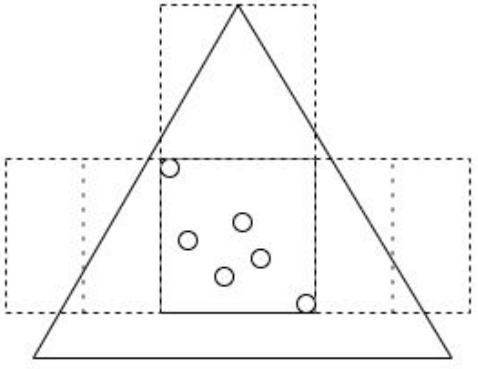

Figure 3: Enclosing triangle

Face identification

A tree data structure will be used to search the face in an efficient way. Every time a face is triangulated in new smaller triangles, this face becomes the parent of these children triangles. So, in the standard case, we have nodes pointing to 3 other nodes, and in the degenerate case 2 nodes are modified with 2 new children each.

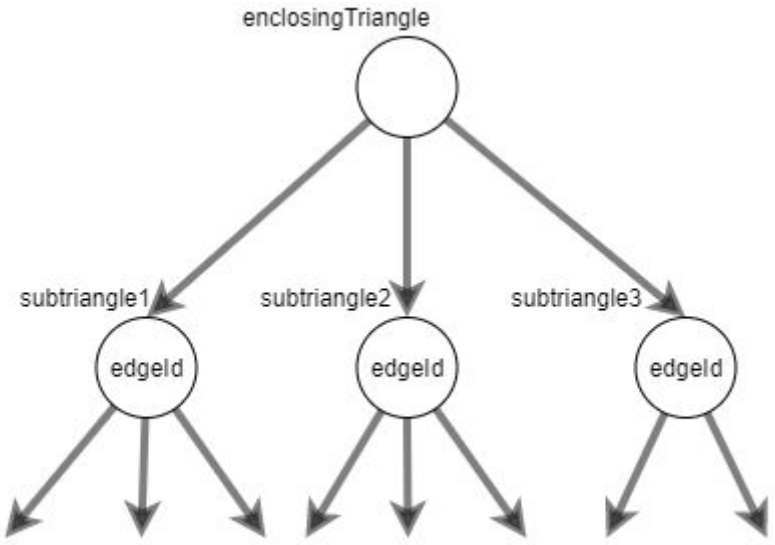

Figure 4: Tree data structure overview

This data structure requires the points to enter in a random order of coordinates, if this is not the case we should randomize the order in which they enter if possible, to achieve something similar to a balanced tree.

For each node, we only save 1 edge identifier and the children pointers. The edge selected for every child node is the one that separates the face of one child from a previous child (following a cyclic order). It's an edge that will allow us to know where to continue the tree search, doing an orientation test with the point.

Then, searching a face will have time complexity O(logn) assuming the tree is balanced.

Some observations

The time improvements when using the tree data structure from a naive search iterating all the faces goes from ~10000ms to 20-80ms.
The first time the triangulation is executed, it lasts longer (50-80ms) than next executions (20-30ms). (Because of how the browser engine manages resources under the hood and then reuses them?).

References

https://dccg.upc.edu/wp-content/uploads/2020/06/GeoC-Triangulating-polygons.pdf
http://dccg.upc.edu/wp-content/uploads/2020/06/GeoC-Delaunay.pdf
http://www.holmes3d.net/graphics/dcel/