COMP308 lab6

03/09/20

1) Calculate time complexity of void triangulate (...)

Sketch of the method:

void triangulate (...) {

⋮

I → for (vertices.begin → .end) { }

$\Delta x, y$ calculation

⋮

create supertriangle and add it to the triangulation

II → for (each vertex in vertices.begin → .end) {

A → - modify iterator range to "remove" 'completed' triangles

B → - modify same iterator to "remove" triangles containing vertex v, keep edges

C → - erase from workset with new iterator range

III → for (each edge kept ; edges.begin → . end) {

- create new triangles from the edges

}

}

D → - remove triangles of the super triangles and move the rest to the output triangleset

}

Assume $|V| = n$ (i.e. n vertices)

By I we loop through a list of vertices to find the min and max coordinates. This takes $O(n)$

By II we also loop through the same vertices list, the loop alone takes also $O(n)$

By A we modify an iterator over workset which initially contains the super triangle. We use remove-if which takes linear time depending on the range of workset. This range is dependent on n and always smaller. Consider this $O(m_1)$ where m = range of workset

By B we modify the same range to exclude triangles which contain the current vertex. Again it depends on n but is always smaller. $O(m_2)$

By C we actually use the resultant range from A and B and transform our workset. erase is linear $O(m_3)$

By III: In step B when we modified the range we kept track of the affected edges; now we loop through the list of these and use them to create new triangles and insert them in the workspace. The number of edges is in the digits of n thus this takes $O(\log n)$

By D we prepare the output or triangle set. Use of remove-copy-if is linear and is applied to the range of the workset $O(m_4)$

To sum it up:

$$O(n + n(m_1 + m_2 + m_3 + \log n) + m_4)$$

We can simplify this by recognizing that n is the dominant term:

$$O(n + n(\log n))$$

$$O(2n \log n)$$

Do away with the constants

$$O(n \log n) \square$$