Computational Methods in Simulation hand-in week 3

KXS806

1 COMMON MESH QUALITY MEASURES

1.1 theoretical part

In order to represent a meshing of a polygon we need to introduce the V array and the T array. In 3d meshing T array contains all the tetrahedron point indexes, and in 2d meshing, it contains all triangular point indexes, each line represents an element. And each line in a V array represents an element, inside are the points arranged in counterclockwise.

This week we will do some meshing to turn a 2D shape into triangles or turn a 3D model into tetrahedrons. However, there are several ways of doing it and the result may differ from each other. I'll use the 2D example to explain. We want the resulting triangles to be as close to a square triangle as possible and we don't want them to be too slim or too long, because in that case, we may counter some numerical problems in computing, as when dealing with float numbers computer will have errors. In order to get rid of these badly shaped triangles and to optimize the mesh function we need a way to measure the mesh we generated.

We implemented two quality measures for tetrahedra. We found them in Shewchuk's note and they are both easy to understand and implement so we choose them.
The first quality measure:

$$6\sqrt{2}\frac{V}{\ell_{\text{rms}}^3}$$

Here V is the volume of each element which in this case is each tetrahedron. And ℓ_{rms} is the root-mean-square edge length of an element, $\sqrt{\frac{1}{e}\sum_{i=1}^{e}\ell_i^2}$.

This quality measure calculates the element's shape by dividing the volume by the edge's length. If the element is tall and slim then this measure will be very close to 0. As we want the shape to be more regular, so this measurement is very useful.

The second quality measure:

$$2\sqrt{6}\frac{r_{\text{in}}}{\ell_{\text{max}}}$$

Here r_{in} is the signed inradius of an element, and l_{max} is the max length of the edges. This quality measure also calculates the shape of a tetrahedra and get the quality of a mesh.

1.2 practical part

In this part, we first implemented the two functions to calculate the quality measure of a mesh. They all take the V array and F array as input and calculate all tetrahedra quality, and return a list of all quality values. Then we implement another function to plot the histogram of the quality measure so we can visualize the quality of the mesh.

Then we load the object "armadillo" and plot it as fig 1

Fig. 1. amadillo

Then we use the wildmeshing to tetrahedral the object and get the plot as fig 2

Fig. 2. amadillo mesh

Then we calculate the mesh's quality and plot the two histograms as follows:

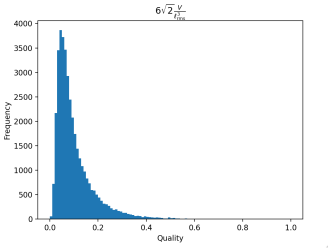

Fig. 3. quality1

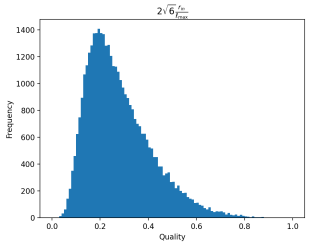

Fig. 4. quality2

From the histogram plot we can observe that the quality of the armadillo is not very good, we think it's because the shape of the armadillo is not regular and it's so hard to express some of its shapes using tetrahedron. Also when we zoom in we can see there are a lot of thin and slim tetrahedrons on its surface.

Then we also applied the histogram to another object:

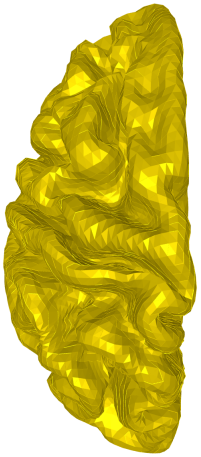

Fig. 5. left brain

Quality histograms:

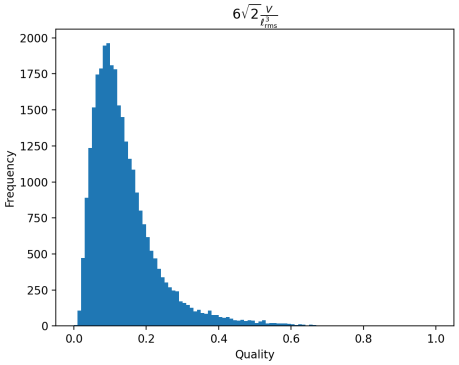

Fig. 6. quality1

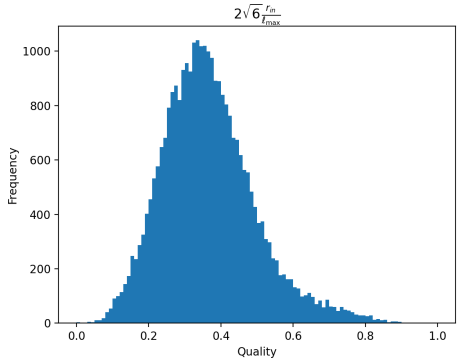

Fig. 7. quality2

obj 2:

2 IMPLEMENT A SIMPLE COMPUTATIONAL MESH GENERATOR

2.1 theory part

In this part, we will implement the marching triangle method to do meshing on a 2D polygon.

To do the meshing we need to do the following steps:

- Make a grid
- compute the values of the signed distance field at the nodes. A signed distance field is a scalar field and each node's value is its nearest distance to the boundary of the polygon. If the point is outside of the polygon then the value will be positive, and vice versa. On the boundary, the points' values are zero. Using this field we can easily find out if a node is inside or outside a polygon. And also it will help to calculate for the new points when doing marching trangles.
- go through each node with pre-defined cases There are 4 cases that shows how to deal with each of the points:
  - Check if any of the edges intersect with the shape, if they are all outside then just remove all points.

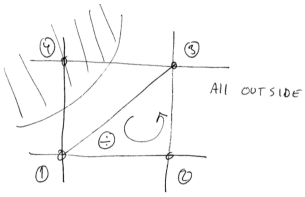

Fig. 8. Case 1

  - If They are all inside the polygon then keep all of them

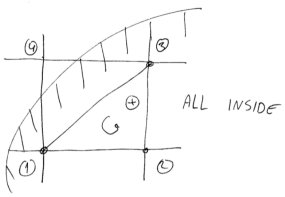

Fig. 9. Case 2

– If two nodes are outside and one is inside then we need to connect them and find the two points on the boundary in which the signed distance is 0. Then we need to use these two new points to form a new triangle.

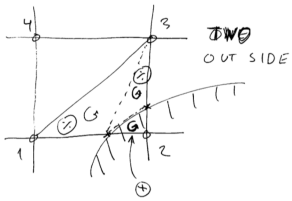

Fig. 10. Case 3

– If one node is outside and two nodes are inside, we can select one inside node to connect to the outside node and get a new boundary point, and connect it with two inside nodes.

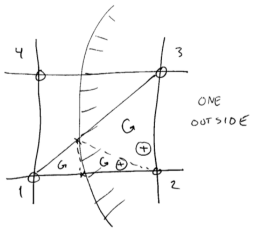

Fig. 11. Case 4

- Run through these cases and get a mesh

2.2 implementation part

We implemented the cases and runned the method on a test polygon and got the following result:

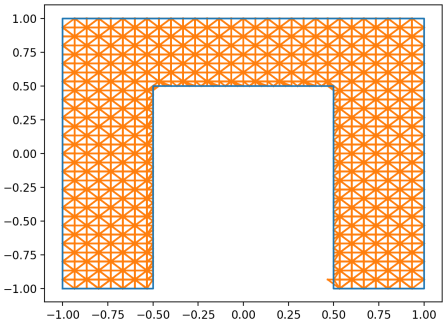

Fig. 12. mesh result

From the result, we can see in the corner the method did a good job of triangulating but it still has some problems, and in the middle, all triangles are the same as expected.

Then we used distmesh to calculate the mesh of the polygon and got the following figure:

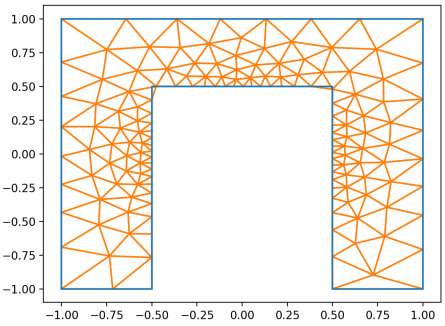

Fig. 13. distmesh

We can observe it gives a much better mesh on the polygon.