Assignment 3

CASPER BRESDAHL, whs715, University of Copenhagen, Denmark

1 INTRODUCTION

This week we will begin by looking at a description of different quality measures which we will later use for our experiment. We will then look into how we can create tetrahedral meshes of .obj files with the python library *wildmesh*. Next we look at the Marching Triangles algorithm and how we can use it to implement a mesh generator. Lastly we will perform an experiment exploring how the *stop_quality* parameter used when creating meshes influences the quality of the generated mesh.

2 THEORY

2.1 Quality measures

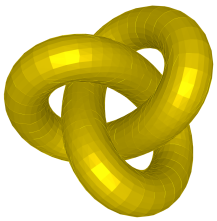
In simulation we are interested in producing meshes which consists of regular tetrahedrals or same sided triangles, which implies all sides and angles are equal. This means we are interested in creating uniform meshes which consists of correctly shaped tetrahedrals or triangles. We can use quality measures to assess our produced meshes and to compare which of two generated meshes are best. Simple quality measures could be the volume of our tetrahedrals, the surface area of our tetrahedrals / triangles or the inscribed sphere / circle of our tetrahedrals / triangles. These measure are quite simple but gives an indication to the uniformity of our mesh. However, these simple measures does not say anything about how we want the geometry to be, only if it is uniform. We thus use the paper [Shewchuk 2002] to find some better quality measures. As the title implies he lists quality measures for both interpolation and conditioning. We are not interested in the interpolation measures as we will not use our grids for numerical analysis, we will instead focus on conditioning measures. For our first experiment we will use two quality measure from [Shewchuk 2002], namely what we will call 'volume over RMS of area' and 'radius ratio'. Both defined in table 7 and 8 on page 54-55. 'Volume over RMS of area' is defined as $\frac{3^{7/4}}{2\sqrt{2}} \frac{V}{A_{rms}^{3/2}}$ where A_{rms} is the root mean square of the surface area of the tetrahedral. This gives a volume to surface area ratio. 'Radius ratio' is defined as $3\frac{r_{in}}{r_{circ}}$ where r_{in} is the radius of the inscribed sphere radius and r_{circ} is the circumscribed sphere radius. If this ratio is close to zero, it means the tetrahedral is very flat or very tall, and thus this measure can tell us about the angles in the tetrahedral. The last quality measure we will use has been found during study group work and will be called 'perfect regular tetrahedrals'. To get this measure we take a tetrahedral in our mesh and compute its volume. We then find the optimal side lengths for a tetrahedral of this volume, and we then compute the inscribed sphere radius of this optimal tetrahedral. We then compute the inscribed sphere radius of the original tetrahedral and compute the ratio between these two radii. This measure describes how close the side lengths in our tetrahedrals are to be optimal for their volume, i.e. how close the tetrahedrals are to be regular. All of these quality measures return values in the range 0 to 1 and can be considered fair as they return 0 for degenerate tetrahedrals.

Author's address: Casper Bresdahl, whs715, University of Copenhagen, Copenhagen, Denmark, whs715@alumni.ku.dk.

2.2 How to use wildmesh to generate meshes

Wildmesh is a python library which we can use to create meshes from .obj files. We first create a tetrahedralizer object which most notably takes as parameters *stop_quality* and *max_its* which impact the quality of the mesh which is gonna be generated. After creating the tetrahedralizer object we can load a .obj file and then compute the V and T matrices which encodes the coordinates of the vertices and the connectivity of the tetrahedrals. With these we can use the library *meshplot* to show the meshes. The result 'tetrahedralizing' the 'dragon.obj' and 'knot.obj' files can be seen in Figure 1 and the Armadillo can be seen in experiment 1.

(a) Wildmesh generated mesh for 'dragon.obj'.



(b) Wildmesh generated mesh for 'knot.obj'.

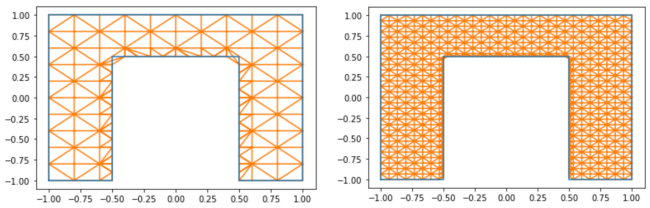
Fig. 1. Wildmesh generated meshes.

3 MARCHING TRIANGLES

To create a mesh for an object we can use the simple Marching Triangles algorithm. The algorithm begins by creating a structured grid which completely contains the object. This structured grid is then split into triangles to form an unstructured grid. In this process we obtain a V array containing the coordinates for the vertices and a T array storing the connectivity of the triangles in the mesh. Each element in T corresponds to a triangle, and thus T is n by 3 in size. Each of the three elements in a triangle encodes a vertex which when looked up in V gives the coordinate to that vertex, and thus V is m by 2. The grid nodes are then assigned values depending on their distance to the object, thus creating a signed distance field. Without making any assumptions about our object, we will now most likely have several intersections between the grid lines and the object. What we are interested in now, is to find these intersections and recreate the mesh such that the triangles in our mesh match the borders of the object. When working in 2D we have eight possible cases which our triangles can 'be in'. The first scenario is all three vertices of a given triangle are outside the object. In this case we can simply remove the triangle from our mesh. The second case is all vertices in the triangle is inside the object in which case we just keep the triangle as it is. We then have three permutations where

one vertex is outside the object but the other two vertices are inside. In this case we need to find the two intersection and interpolate the point outside onto the object towards the intersections. This results in a square which we will then have to split into two triangles. Lastly we have three permutations of having one point inside the object and two points outside. Here we again have to interpolate the outside vertices towards the intersections of the object. But in contrast to the previous case we get a new triangle here as we 'move' the two points outside the object onto the border of the object. To determine which case a triangle belongs to we look at the value of the signed distance to the object. If it is negative we are inside the object and if it is positive we are outside. We use the formula $sign(\phi(v1)) + sign(\phi(v2)) \cdot 2 + sign(\phi(v3)) \cdot 4$ to determine which of the eight cases we are in. One should look at this like it being a bit mask where we find the sign of each vertex in a triangle. Vertex 1 is then responsible for the smallest bit, vertex 2 for the middle bit and vertex 3 for the last bit. This gives a unique mapping to the eight cases and we know exactly which vertices are inside and which are outside.

As we now know which vertices are inside the object and which are not, and we know what to do with these vertices, we now only missing how to find the intersections of the object and the grid lines. This can be done is several ways. One way, and the way I have implemented is the following. We use the mapping system described above to identify a triangle which has one or two vertices outside the object. We then create a vector from a point inside to a point outside. We then get the signed distance for the point inside to the object and the distance of the vector we created. We then create a ratio between these two distances and multiply this ratio with the vector we created. The idea being this ratio tells us how long we need to travel along the vector to get to the intersection. Moving a copy of our point inside along this our vector with the specified length we get closer to the intersection point. But as the signed distance we compute is likely shorter than diagonally distance we need to travel we repeat this process in a loop with the copied point. When we have iterated a few times we end up fairly close to the intersection point. This method has it shortcomings as very flat triangles would require a lot of iterations, and there are no guarantee the closest point of the object coincides with the vector we created and move along which means we might measure a wrong distance to move, however, this only measures too short a distance and so, in later iterations we will measure the distance to the correct border. The results using this approach to create a mesh can be seen in Figure 2. We see that the meshes are not perfect, lacking some triangles in some corners and 'taking too much corner' in others. These are unfortunately side effects of the Marching Triangles algorithm. Another perhaps simpler approach we could have used to find the intersections is a bisection. We here identify the two vertices which cross the border of our object and then find the midpoint between these two points. If the signed distance form the object to this midpoint is positive we are still outside and we can move the outside point to the midpoint. Otherwise we are inside and we can move the inside point to the midpoint. We can repeat this process in a loop until we are sufficiently close to the intersection.

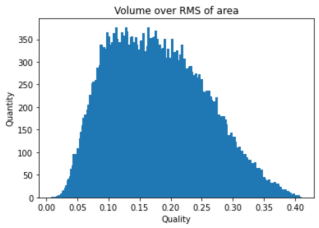


(a) Triangle mesh generated with 10 by 10 nodes.

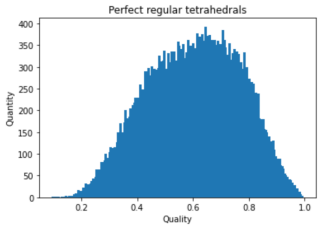
(b) Triangle mesh generated with 30 by 30 nodes.

Fig. 2. Two generated triangle meshes.

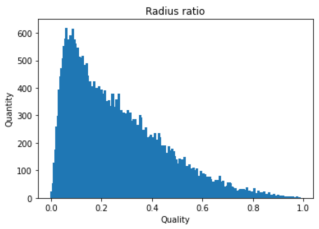
(a) Armadillo tetrahedralized with *stop_quality* of 500.



(b) Histogram of volume over RMS area measure.



(c) Histogram of perfect tetrahedrals measure.



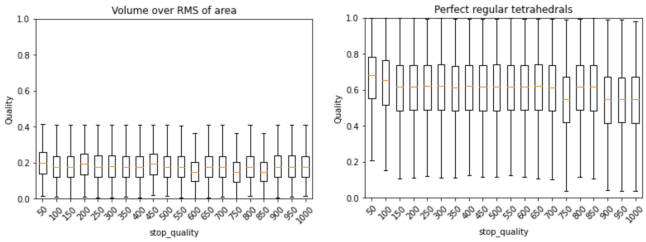
(d) Histogram of radius ratio measure.

Fig. 3. Histograms of quality measures for Armadillo tetrahedralized with *stop_quality* of 500.

4 EXPERIMENT 1 - HOW MESH QUALITY CHANGES WHEN TUNING *STOP_QUALITY*

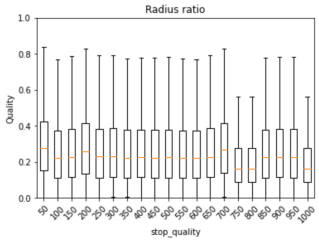
In this experiment we would like to examine how the quality of meshes generated by the library *wildmeshing* changes when we adjust the parameter *stop_quality*. For this we will use the three quality measures described in the theory, 'volume over RMS of area', 'perfect regular tetrahedrals' and 'Radius ratio'. To establish some initial idea of the quality measure we measure the quality of the Armadillo object when tetreahedralized with a *stop_quality* of *500* which can be seen in Figure 3. As we see, the quality of this mesh is rather poor, as there are no peaks in the histograms. This implies the tetrahedrals bein measured are not uniform which is what we would like them to be. More specifically we see from Figure 3c that there are very few actual regular tetrahedrals, and the average value being around 0.6, implying quite poor optimal shape.

With this initial look at the tetrahedralized mesh, we would expect to be able to increase the mesh quality by decreasing the *stop_quality* parameter. The *stop_quality* adjusts how much AMIPS energy is



(a) Box plots of volume over RMS of area measures.

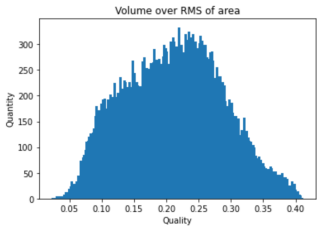
(b) Box plots of perfect tetrahedrals measures.



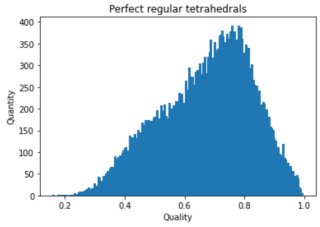
(c) Box plots of radius ratio measures.

Fig. 4. Box plots of quality measures for Armadillo mesh with *stop_quality* parameter setting in range 50 to 1000.

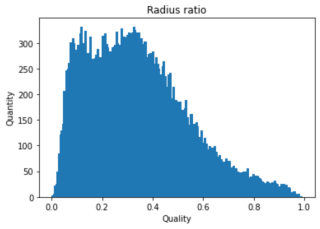
(a) Armadillo tetrahedralized with *stop_quality* of 50.



(b) Histogram of volume over RMS area measure.



(c) Histogram of perfect tetrahedrals measure.



(d) Histogram of radius ratio measure.

Fig. 5. Histograms of quality measures for Armadillo tetrahedralized with *stop_quality* of 50.

tolerated in the mesh, which from Google searches seems to be a measure of distortion. This, when decreasing the amount of tolerated distortion in the mesh, we would expect the quality to rise. As it is not quite feasible to compare a range of histograms, we instead create box plots which can tell us about the spread or variance of the data and where the mean is. Using the Armadillo, we can now compute the tetrahedral mesh and perform the quality measure and plot the result as a box plot for *stop_quality* values in the range 50 to 1000. The results can be seen in Figure 4. In Figure 4a we do not see too much change, but there seem to be a slight tendency for the variance to become smaller and smaller as *stop_quality* increases, however, the mean the data then centers around seem to be lower than at low *stop_quality* values. In Figure 4b we see a tendency of the mean getting lower as *stop_quality* increases, and the variance increasing as *stop_quality* increases. In Figure 4c we again seem to get quite similar results, but both the variance and mean of the data seem to shift downwards as *stop_quality* increases.

4.1 Discussion of results

We get a clear impression that the tetrahedrals become a lot more regular at lower *stop_quality* values as the mean is at the highest and the variance is at the lowest in Figure 4b. However the box plots of the two other measures does not give a clear indication to a best *stop_quality* value. On one hand we want the quality to be the highest, and on the other we want the tetrahedrals to be uniform. As all three box plots agrees on the highest quality is found when *stop_quality* is low we choose to investigate this by plotting the histograms for the Armadillo with a *stop_quality* of 50 which can be seen in Figure 5. Comparing the two Armadillo meshes with the naked eye there is not too big of a difference, however, comparing the histograms in Figure 3 and Figure 5 we see big improvements in the quality of the meshes. We see that the mean of the 'volume over RMS of area' has shifted up without the variances really changing, we see the variance has dropped a lot in 'perfect regular tetrahedrals' and the mean has shifted up, and finally we see the mean has shifted up for the 'radius ratio' although the variance has also increased quite a lot. We thus conclude that the quality can be improved by using lower *stop_quality* values when tetrahedralizing the Armadillo mesh. However, the variance or uniformity of the tetrahedrals suffer to some extend when it comes to the 'volume over RMS of area' measure and suffers quite a bit when it comes to 'radius ratio'.

5 CONCLUSION

We end by concluding the Marching Triangles algorithm is fairly simple and can create fairly good meshes, but not quite perfect. We see the meshes often have 'cut' edges, or edges where we 'steal some of the corner'. These defects are however reduced as we increase the resolution of our grid. The results we have seen were created with a 'home-made' method and could have been simplified by using a bisection algorithm to compute the intersection points. We also end by concluding that *stop_quality* have a quite significant influence on the quality of the generated meshes as we lowered the value. This is especially seen in the 'perfect regular tetrahedrals' measure which improved the most both in quality and in reduced variance. The other two measure we have looked at suggested that the quality rose as *stop_quality* had lower values, but the variance fell as *stop_quality* increased. However, comparing the Armadillo mesh at *stop_quality* 500 and 50 showed big improvements in quality for the *stop_quality* 50.

REFERENCES

J. Shewchuk. *What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures.* Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, 2002.