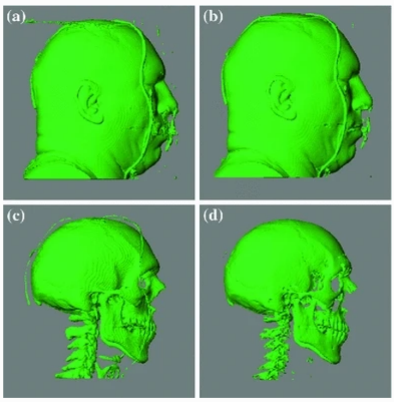
Final Term Project

Garrett McMichael, Arman Hastings

December, 2022



Volume rendering of a male head dataset using marching cubes with different isovalues: **a** 30, **b** 50, **c** 75, **d** 100

1 Abstract

How can we add a visualization technique we learned in class to existing marching cubes visualizations for a new perspective on complex data sets? We decided to take our found interest of these topics and expand on it with the paper we chose on marching cubes. Marching cubes have been used for complex medical applications such as with MRI data sets and we decided that we could use critical points and saddle points to visualize additional information from these data sets. These additions could help with creating new understandings for data sets with a quick visual inspections such as with critical points or saddle points showing something out of the ordinary in a data set such as for a MRI scan.

2 Getting started

In class this term we covered marching squares, critical points and saddle points. For our project we selected option two because our team decided that we are missing a domain expert in most the topics we discussed in class.

Our MRI data set is our complex data set we aimed to apply our work to. Additionally the medical MRI data sets we found associated with our research paper was prepared for us as a set of images that when put together create a 3D model. This interested us in focusing our approach on applying our method to a 3D model first, then when we could be confident that we could apply it to the 3D MRI data set after putting together with marching cubes.

To get started we decided that we wanted to test the application on a much simpler data set that could be easily visually evaluated to then confirm our methods are working properly. We decided to create a simple data set of a sphere, with marching cubes and critical points. This proved to be more difficult than we initially imagined. We were able to apply marching cubes from our paper to our generated sphere giving us a good base for adding our critical and saddle points. Once we decided that we needed to created a list of neighbors for each point to create the critical points.

Once the unordered map of critical points was created we then applied similar methods from class to create our critical points taking every set points greater into a Max and every set of lesser points to a min.

3 Testing our methods

3.1 First tests for accuracy

3.1.1 Reducing our critical points

Once we finally had our first result of critical points applied to our data set based on marching cubes we did a quick visual inspections to find our method was yet to be working as desired.

Our first results yielded us with too many critical points and maximum points. After discussing we figured the issued stemmed from a lack of precision during our finding of the critical points. We then visited our TA Jinta to get his perspective on our situation and he agreed it could be because of a lack of precision.

With that we reduced the issue down to a statement where we converted a string to a floating point. This has been known to case issues in other projects so we proceeded with our first approach of converting our string variable to our Vector3 variable we worked with from class since that would match the Vector3 variable we resulted in thus theoretically eliminating the precision error. To do that we had to make some changes in our code, abandoning our first attempt to handle the precision from the string. After we finished our changes such as overloading the evaluative equals operator to accept a Vector3 to Vector3 comparison we compiled, and ran into run time errors when trying to get our desired result. These errors in our perspectives looked very complicated so this is where we returned to the concept of throttling our precision error with the string to Vector3 statement. We returned to a working version of this, which compiled and we still found errors in our visual inspection. In our perspectives it seemed though we had made some sort of progress in reduction the amount

of minimum and maximum critical points to something more inline with what we expected in a fully working situation however we had some additional issues after this with our max critical points.

We decided that this issue with maximum critical points could be resolved with some working our our statements around the finding of the critical points and started focusing our attention there.

Even though we have yet to apply our additions with the marching cubes method to the complex MRI data we are proud to say we have a partially working product with our generated 3D data set. We believe that with a little bit more time we could fix our final issues and get this working fully on that generated data set and then the complex data set.

From here we continued we changed quiet a bit of the project to address the issues we were having with the critical points and we decided to explore new approaches to solving our issue after our work on the precision with throttling the string to float conversion for accuracy. We also added list to keep track of the unique vertices found.

Other than precision, we discussed and figured that our issue could be coming from the our neighbors evaluation of vertices.

We came to the conclusion that were only looking and calculating our results for our min and Max points with some of our neighbor triangles instead of all of them. We found that we were evaluating the first vertex of the triangle instead of looking at each vertex and adding all the triangles neighbors.

Again we went and altered our evaluative equals function for both Vector3 and Vector4 changing it to work with the Epsilon value we added. We then changed our unordered map of vertices for our neighbors into a new set of two vectors of vertices that use Vector and Vector3 variables. This these two vectors created from our initial unordered map allowed us to eliminate the string variable and move forward with variables that will be consistent and precise all the way through.

To add the missing neighbors we decided that we needed to do more searching, by creating a couple additional nested for loops in the function where we analyze vertices (AnalyzeVertices( )) to find each vertex of the triangle to find any vertices of triangles we may have missed in the neighbor map. These were then added to our new two lists This allowed us to add some new missing vertices to get a result that is closer to our expected result.

With that an extra loop also needed to be added when we added any newly found vertices to our neighbors vector list and a couple additional checks to make sure we were only adding new points we needed.

Then for finding critical points (FindCriticalPoints( )) we again had to alter it to work with our new variables to search through our new vector list of analyzed Vertices for the critical points

3.1.2 Removing the extra point

After doing this we compiled, visually evaluated it and found that we got a much more desired outcome for our mins and maxs. With our maxs fixed,

being closer to our desired outcome we found another one off issue, an extra critical point, that found was created from a missing neighbor vertex creating that extra point in an incorrect place. To test this theory we hard coded that point in to our neighbors list and it resolved our issue so we took that as a good sign to proceed with finding that problem and getting it to work by finding the critical polygons.

Additionally before finding the critical polygons, we were looping through each triangle for vertices and then then we made another loop to find each triangle. We were checking the initial vertex list by looking through the vertex list with the triangle incrementing to get the other values and we were going out of bounds to the next triangle in the loop. To fix this we made a vector of three elements for all three vertices so that the loop would be bound more clearly and added a mod to assert that everything stayed within scope during the looping.

Secondly with the hard-coding of the missing vertex we did a closer inspection and we found the specific issue of an incorrect looping variable was used, which once we correct it fixed our problem. This came up most likely because of missing a correction after additional for loops that were added.

To resolve this we hard coded visual outputs to aid our debugging process and found the exact point that is needed to be included in the evaluation of the incorrect point to have it removed. We are then left we the min and max polygon for a visual solution one step closer to our desired visualization of critical points.

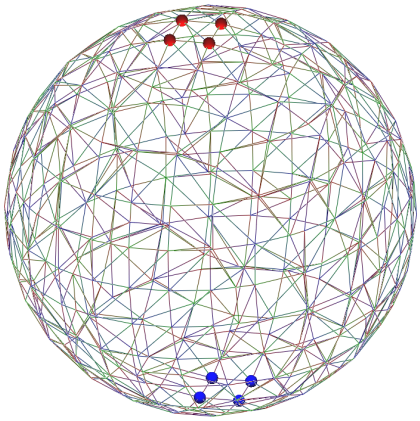
3.1.3 From Critical Polygons to Critical Points

At this we had a working result that was of multiple critical points associated to the vertices of the object in shape of the max and min polygon at the max and min most position. This is correct, if we wanted to find the critical polygons because all of the found points are the highest vertices in the mesh all together outlining the min and max polygon of the mesh. We at this point had a critical Polygon, though we desired to have a result of only having 1 point then the next steps we sought after was to find the neighbor critical points from the critical polygon to then make a single critical point.

We eventually were able to reduce our 4 points into two points at the single center of the min and max points of the data set giving our very close double result of critical points. Our code went through all the critical points gets neighbors then it looped through them all again and then checked if any neighbors are critical points. Then it added them to list called critical neighbors.

Before we were testing if there was any neighbors (greater than zero) , we would find a mid point and create a neighbor, this was creating an issue because two of the triangles were missing so it made two extra points on top of the desired critical point position. Then we solved that added an extra incremental step to find that other neighbor triangle which then resulted in use having the two points in the one critical point place mentioned previously. From that point, with the two points, stacked on top of the same position, all we needed to do is removed one of those two critical points.

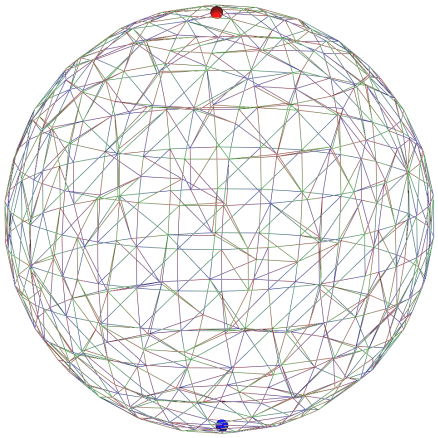
Figure 1: Visualizing the critical polygons of our generated data set



The next step would be to eliminate one of those two points at the same position. We fixed this by checking if the critical point exists, and then only adding one. We found the same point twice and there for made two points. Now with that evaluation we just have our final result of one critical point for min and max on the sphere. This was our desired result for our visual inspection

At this point there would be no saddle points for a sphere so we then decided we would move on to our next generated data set, the more complex, noisy sphere.

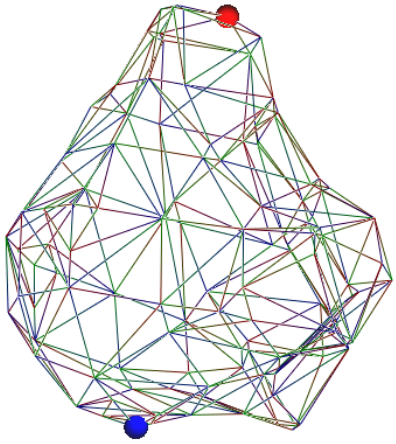
Figure 2: Visualizing the critical points of our generated data set



3.2 Second test for accuracy

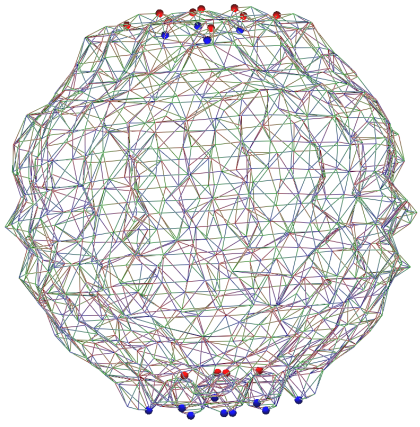
We then tested this with a sphere data sphere with noise, an irregular shape and it works as expected on first visual inspection as it worked on the sphere data set. When testing on an irregular shape it only finds one absolute critical point, instead of a critical polygon.

Figure 3: Visualizing the critical points of our noisy generated data set



We also can represent multiple local minima and local maxima between the absolute minima and maxima.

Figure 4: Visualizing the critical points of our noisy generated data set



We also can represent multiple local minima and local maxima between the absolute minima and maxima.

4 Next steps

From here we can move on to finding saddle points or finding local minima and maxima points.

5 Use our added visualizations

To use our program open the solution in visual studio with build tools v143, you can zoom in and out with the scroll wheel and rotate the camera my dragging the mouse while holding right click. Pressing the number **1** on the keyboard will display the mesh in a flat shaded view

To visualize the critical points **hit A** to prepare the visualization by making the neighbors map and finding the unique vertices, **hit B** to prepare the critical points for a critical polygon, then **hit C** to prepare the reduced critical points down to one critical point, and then then **hit D** to cycle through the critical points display modes.

This critical polygon however could be used with the critical point we plan to find for added visual aid for an individual that may want to test that a scan is level such as with a file that was created from a scan that may have had an issue resulting in an off center critical point in the critical polygon. For example this visualization of the critical polygon and point could be helpful for an individual scanning in an physical object to an 3D printable file who is looking to inspect the top and bottom polygon such as to make it level enabling both the critical points and polygon for a comparison.

6 Conclusion

We were able to create critical points, and critical polygons for our test data set and our more complex test set. We expect this could be applied to the MRI data sets if we could create the model from the collection of imagines provided then we could see how our application fits to that model.

7 Our Proposal

7.1 what about option 2?

Marching cubes technique for volumetric visualization accelerated with graphics processing units[1]. We decided on this paper to focus on volume visualization and rendering with the marching cubes technique.

7.2 Data sets

We plan on getting our main data sets from OpenNeuro. OpenNeuro is a free and open platform for sharing multiple types of data including MRI scans.

Our secondary data sets will be coming from the Stanford 3D Scanning Repository[2]

We plan on using a sphere as a testing data set and we will get that sphere with the function sampleValue = distanceFromCenter - sphereRadius.

Our first data set is an MRI of a mouse from OpenNeuro[3]. To get our data into a format we can use, we plan on taking the MRI data and turning it into images we can sample as textures in OpenGL.

Here is an example image from one of our data sets where black is 0 and white is 1:

 [3]

7.3 How we will evaluate correctness

To evaluate correctness we will start on more simple objects to assess how it is working before applying them to more complex objects such as from [2]. We plan on using a sphere and simple objects that can be visually verified to test that the marching cubes algorithm is working, after we verify the output mesh is correct we can use the same simple objects to verify the critical points and contour algorithms are working.

7.4 Our scope considering the paper

We plan to focus on the marching cubes technique for the volumetric visualization and adding features we learned about in class such as contour lines and critical points. Other topics such as volume rendering data structures, CPU and GPU acceleration are further out of scope for our focus.

References

[1] Marcos Vinicius Mussel Cirne & Hélio Pedrini. Marching cubes technique for volumetric visualization accelerated with graphics processing units.

[2] The stanford 3d scanning repository. `https://graphics.stanford.edu/data/3Dscanrep/3Dscanrep.html`. Accessed: 2022-11-18.

[3] Openneuro. `https://openneuro.org/`. Accessed: 2022-11-18.