CSE 190: Assignment 2—Geometric Modeling and Mesh Simplification

Ravi Ramamoorthi

1 Introduction

This assignment is about triangle meshes as a tool for geometric modeling. As the complexity of models becomes larger, and better techniques of acquiring them from the real world (such as by range scanning or nowadays depth maps from Kinect-like sensors) become prevalent, polygon meshes are becoming an important representation for modeling. This assignment explores one aspect of triangle meshes—mesh simplification. Mesh simplification is important for making the complexity of large meshes manageable, allowing transmission on the internet, and providing control for level of detail operations. We will be implementing classic and widely used mesh simplification methods based on papers by Hoppe and Garland.

The specific goal of the assignment is to write a mesh simplification system (based on the Quadric error metric paper by Garland and Heckbert in SIGGRAPH 97) and show a demo of progressive meshes [Hoppe, SIGGRAPH 96] (so you can transition progressively from a coarse model with a small number of faces to a detailed dense model). Clearly, an implementation of the entire system can have numerous uses (such as internet transmission, low resolution models for games etc.) and makes an impressive demo (for instance, see the geomorph demo on Hoppe's website).

Warning: This assignment can be challenging, primarily because it may be hard to debug the data structures and edge collapse methods. The assignment is structured in strict order of increasing difficulty. Some groups may not be able to get all the way, but if you follow the various steps here, we can grade based on what you accomplished.

Support: We do not provide skeleton code for this assignment; this work is more open-ended and involves to some extent, building a real system. For an example of some of the ideas involved and what you can achieve, you can download code and demos from Hoppe (progressive meshes) and Garland (the qslim package). Clearly, it would be wrong to simply (or otherwise) copy code from those or other sources, though I have no objection to your looking at the source code in them for inspiration or understanding. We do provide geometric models for you to test your code, linked off the website (the original models are courtesy of a number of sources, including Stanford and Viewpoint). This assignment will require some knowledge of OpenGL to write a user interface.

Hints: Try to read the entire assignment before starting. While the components are in increasing order of difficulty, and should be implemented incrementally starting early, it is worth thinking about the overall system as well. Besides this, there are two important hints:

- **Choice of mesh data structure:** This is critical in terms of ease of implementation and efficiency. Think through the algorithms before determining the best choice. There is no one answer here. An indexed face set, along with a vertex-to-face adjacency list (for each vertex, store the list of adjacent faces) can enable a relatively simple to implement, but slightly sub-optimal algorithm. Fancier data structures like winged-edge or half-edge can lead to slightly more efficient algorithms, but implementation can be quite complex, especially if you take into account special cases in order to reject edge collapses that would lead to a non-manifold topology.
- **Use of data structures:** This assignment can make a lot of use of auxiliary data structures like vectors, lists, priority queues and so on. If you use C++, you should take advantage of the STL container classes if possible, to make your life simpler. You should bear in mind the cost of operations

though if you use these types of data structures that you did not implement yourself. For instance, insertion and erasing of an element are $O(n)$ for a vector, but *push_back* and *pop_back* are amortized to be constant time complexity.

2 Submitting the Assignment and Logistics

You will turn in the assignment by creating a website (or making a PDF writeup). Please also link a video off your website, showing progressive meshes in action. Your website should include a complete description or writeup showcasing your results. Since this project is slightly less structured, part of your task is to figure out how best to present your results. In general, as a researcher or even in industry, one goal is to document your results well and to meet deadlines.

Failure to fully document the results of your program will make the grader very unhappy. Being honest, noting what works and what doesn't work for all the features will make the assignment easy to grade and make us very happy. That said, the purpose of the assignment is to have fun; we will try not to be too harsh on technicalities. Your website should also include an executable (if we can't get it to run, we may need to ask you to do a demo session; we may also request to look at the source code). As before, please do not include source code on your website, or on public sites like github repos. Do not modify the website after the due date of the assignment. E-mail a link to your website (or PDF) to the instructor/TA before the due date.

You may (and are encouraged to) work in groups of two. Only one assignment needs to be handed in. The requirements remain the same if you work alone, although we will try to informally take into account the situation in assigning final grades if you are forced to work alone.

3 What you need to implement

You should go through the following sections in a step by step fashion, gradually implementing more complex features.

3.1 Mesh viewer

The first thing you need to be able to do is read in the models we provide, and work them into a viewer. The viewer should at the very least allow you to perform basic operations (rotate the model, translate the viewer or model and scale and/or zoom in and out). You should also compute some kind of nice shading on the mesh, by computing the normals (average neighboring face normals for each vertex to compute its normal). It is helpful to have a number of other modes for debugging. Since you are dealing with flat triangles, it is worthwhile for debugging to use flat shading. Other useful things are to color each triangle with a distinct color that depends on coordinates or chosen in some other way not to overlap its neighbors etc. Finally, when you read in a mesh, the program should automatically compute a default scale for display so the mesh is centered on the screen. Your writeup should state briefly what you implemented, and show 3 or 4 screenshots of your program running with different models.

The format of the models you are given is OFF. This is a very simple format. The first line simply says OFF. The second line has 3 numbers. The first two are the numbers of vertices and faces, respectively. Then, there are as many lines for vertices as specified above. Each line has 3 coordinates for the vertex position. Finally, there are as many lines as faces in the model. Each line starts with 3 (the number of vertices in a face—since we are dealing with triangles, this will always be 3). The next 3 numbers are the vertex numbers in that face. You can assume that triangles are always specified by naming the vertices in counterclockwise order. The numbering of vertices starts at 0. There are no normals specified. Your program should compute at least face normals for some kind of lighting, and may need to compute vertex normals (by averaging the normals of adjacent faces) for smooth shading.

To do this part, it is worth thinking about what mesh data-structures you will need for the later part of the assignment. You can define a Mesh class, filling in the appropriate information for triangles, faces and possibly edges, by reading the model geometry. You can define a draw method to actually draw the mesh.

3.2 Mesh connectivity data structure

You will now implement a mesh connectivity data structure. The point is to allow you to perform an edge collapse (required next) in constant time (or at least in time proportional to the number of neighboring edges). An edge collapse should not require you to iterate over the whole mesh, and should be a local operation that leaves most of the mesh completely unchanged.

The data structure needs to be able to answer questions like "What vertices neighbor my current vertex?" or "What faces neighbor my current vertex?". You will need to think carefully about what information you will need for the next part of the assignment.

There are a number of data structures possible like indexed face set with vertex adjacency lists, half-edge, quad-edge, winged-edge and so forth. Some of these will be discussed in lecture. Garland's 97 paper says they explicitly represent vertices, faces and edges, along with adjacency information regarding all of them. That is not necessarily the easiest to deal with; see the hints at the beginning of the assignment.

You also need to think about whether the data structure is complete, and how it can be updated when you do mesh decimation or edge collapses. Remember you also need to build up the data structure from the raw vertex and face information in the OFF file. You will probably write a buildconnectivity method for your Mesh class to implement this.

Your writeup should describe briefly the data structure you chose, and some rationale for why you chose that structure, and how you can efficiently update it for edge collapses. You should also describe how you create the data structure from the vertex and face information supplied in the OFF file.

3.3 Mesh Decimation

For this part of the assignment, you will implement an edge-collapse method in your mesh class that takes as input two vertices and collapses the edge between them (assuming they do have an edge between them). This is the basic operation in mesh simplification algorithms like Garland's 97 paper. In principle, this is straightforward, since in doing the previous part, you should already have thought about this part. Choose the new vertex to replace the two vertices being collapsed as lying at the midpoint of the two for now.

In practice, you need to make sure you tie in all the loose ends. You need to think about what happens to adjacent faces, vertices and edges, and how you will update your data structure. You should already have thought about (and documented) this in your previous writeup.

Note that when you collapse an edge, you must make sure to remove any degenerate faces that are produced. Refer to figure 1. This test, *testpatch.off* is available in the models directory (but **not** in the zip file, if you download that). Collapsing the edge between v0 and v1 makes faces X and Y degenerate (two vertices will coincide) and those should definitely be removed from the model. You should also try (though it is not absolutely mandatory) to remove any fins (pairs of mirror-image polygons), such as Z and W in that example. In your writeup, document your results on this simple test case.

The hardest aspect about this part of the project is probably just debugging it. For debugging, incidentally, you might be better off placing the new vertex at v_1, rather than at the midpoint. In general, some time spent thinking about the problem may save you lots of effort in actually writing the code. For debugging, the easiest thing is to start with a very simple mesh. The plane model *plane.off* is a good one. You can also make your own very simple mesh. Sketch out what should happen on a piece of paper, and verify (step through your code and examine data structures if necessary) that your mesh connectivity data structure does get updated correctly.

For the writeup, the following is one option, besides the required mandatory documentation on the test in figure 1. If you think you have a better way of doing this, such as examples equivalent to figures 1 and 2 in Garland's paper, that is fine too. For the plane model, collapse vertices 11 and 23, leaving vertex 11 where is. Show the corrected mesh (zooming in on the affected region would help, as would adding labels to note the vertex number). Also, it would be nice if you included verification of the connectivity updates for neighboring vertices or faces. The main point of this exercise is not to be harsh on you, but to convince yourself primarily, and me, that your mesh decimation is debugged.

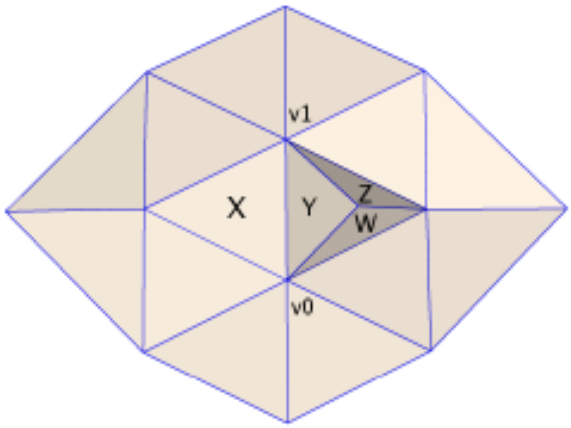

Figure 1: Consider this example, in *testpatch.off* Collapsing the edge between v0 and v1 will cause faces X and Y to become degenerate (two vertices will coincide). You should definitely remove X and Y from your model representation. Further, you should try to remove any fins (pairs of mirror-image polygons). In this example, you should try to remove Z and W.

3.4 Quadric Simplification

Now, go ahead and implement Garland's 97 paper on quadric mesh simplification. That might sound daunting, but here are a few simple steps to go about it (as described in section 4.1 of the paper): You should be warned that this part may be difficult for some of you. If you feel you won't be able to do this, you might be better off reading the next part of the assignment, and implementing progressive meshes (with dummy examples) first.

1. Compute his quadric Q matrices for all initial vertices. Read section 5 of the paper.
2. Select all valid pairs of vertices. For now, it suffices if you deal only with edge collapses, though you will eventually want to incorporate thresholds, as described in section 3.2.
3. For now, don't worry about choosing the contracted vertex to be at the optimal location with respect to the error quadric, but just place it at the midpoint. Compute the quadric for contracting a given paper per bullet 3 in Garland's algorithm.
4. It's easy enough to sort the pairs by the cost. The tricky thing is that when you go through one contraction, you need to update the costs of all related pairs (though this shouldn't be too hard, given your edge collapse structure). Then, you need to re-insert things in the list of pairs. For that, you'll need an efficient sorting method to allow fast insertion. Garland recommends using a heap. It is permissible to use outside code for simply the implementation of a suitable data structure for this.

You should show in your writeup, models simplified with different numbers of faces, as well as a brief description of what you did, and experiences with the assignment. It may be helpful for demo and debugging purposes to implement a simple control, like on the keyboard to progressively simplify your model.

3.5 Progressive meshes

Finally, you need to implement progressive meshes. The Hoppe 96 paper is a useful reference, but you don't really need to read it. Once you've got this far, this is actually quite trivial. You simply need to write out the list of edge collapses to a file, as you simplify your model down to some small number of faces. Then, you can read in this file, and invert the order (you now need to perform vertex splits to do the opposite of the edge collapse) to create a progressively more detailed mesh. As above, you should include some form of control (simple keyboard control is fine) to make your mesh coarser or more detailed. For full credit, you should also implement Hoppe's geomorph, to create a smooth animation. You should include a video of the geomorph, along with screenshots on your website.

If you are unable to get quadric simplification to work, you can still get most of the points for this part [progressive meshes] by showing a few examples where you manually or in some other way chose the simplification order (possibly easy enough for simple shapes like plane or torus).

3.6 Grading:

The tentative grading scheme (subject to change) provides 7 points for the writeup, 3 points for the mesh viewer, and 10 points each for the mesh connectivity data structure, mesh decimation, quadric simplification and progressive meshes. If you do anything you think merits extra credit, please include it in your writeup.

Acknowledgement: This assignment has been adapted from a number of sources, including Szymon Rusinkiewicz and Igor Guskov.