CS 184: Computer Graphics and Imaging, Spring 2019

Project 2: Mesh Editor

Gefen Kohavi

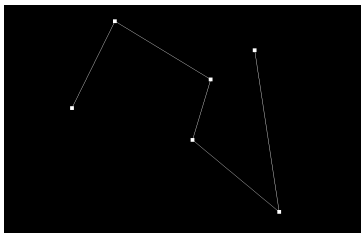
Overview

In this project, I implemented methods to deal with 3D meshes. First, I worked on 2D and 3D bezier curves to smoothly interpolate between points. Then I worked on surface normals to make smooth shading of meshes. Finally, I worked on mesh modification algorithms like half-edge flipping, edge splitting, and mesh upsampling. I learned a lot in this project and now understand why the half edge mesh data structure is so useful.

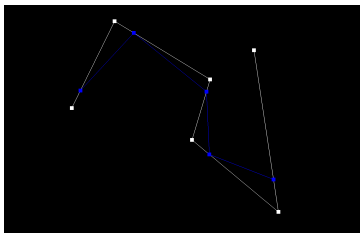
Section I: Bezier Curves and Surfaces

Part 1: Bezier curves with 1D de Casteljau subdivision

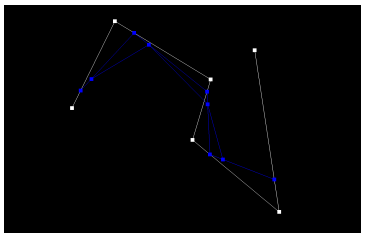
De Casteljau's algorithm is used to implement Bezier curves. The algorithm happens as follows. First, given a list of vectors, we can add a level by taking a weighted sum of each pair of points that are consecutive in the list. The equation in this case is $(1-t)*p_i + t*p_{i+1}$ where t is between zero and one. This subdivition process can be repeated until there is only one point in the new list we have. This point is the point on the actual bezier curve.



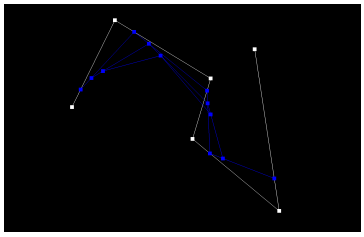
Normal 6 control points



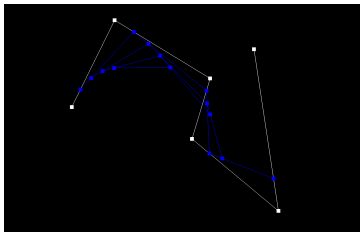
1 Evaluation level



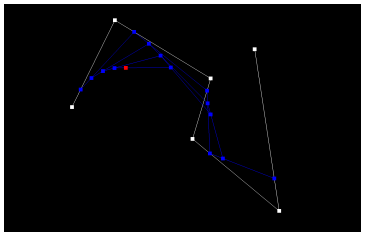
2 Evaluation levels



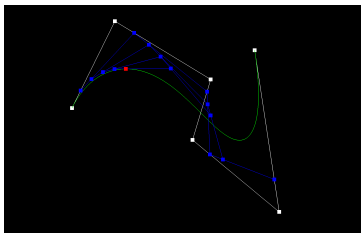
3 Evaluation levels

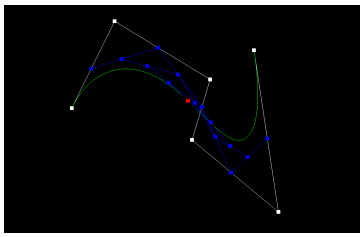


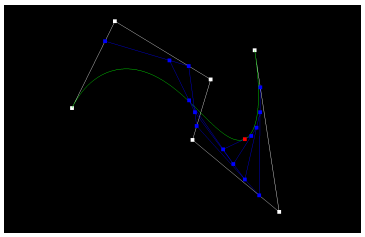
4 Evaluation levels



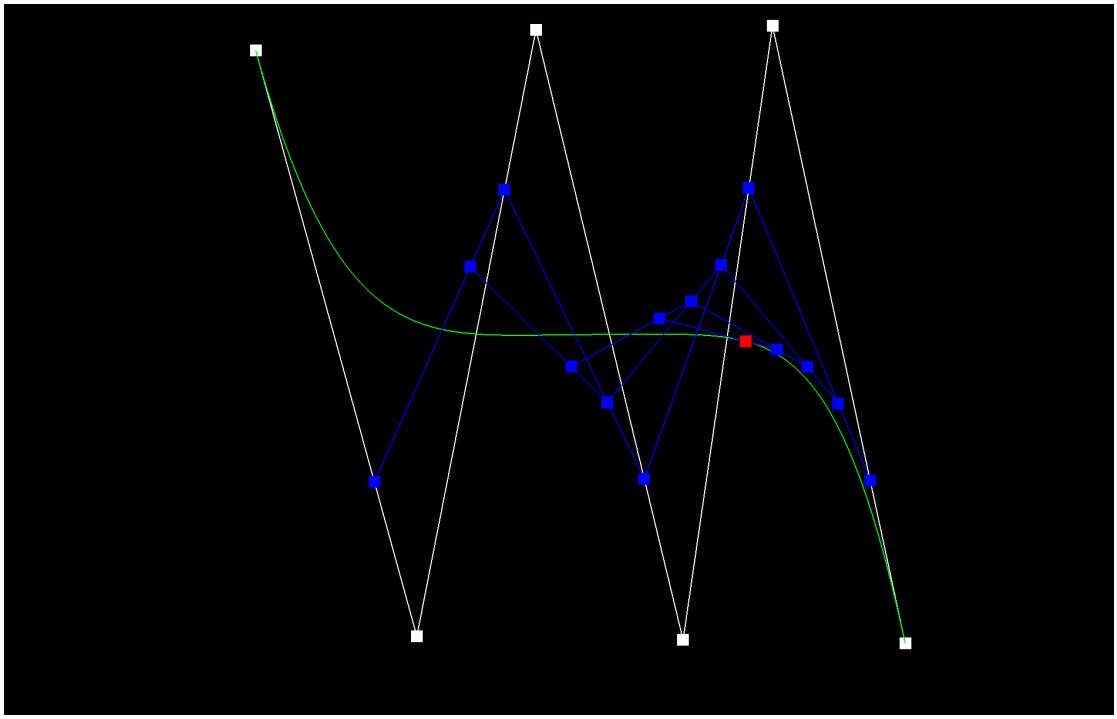
5 Evaluation levels







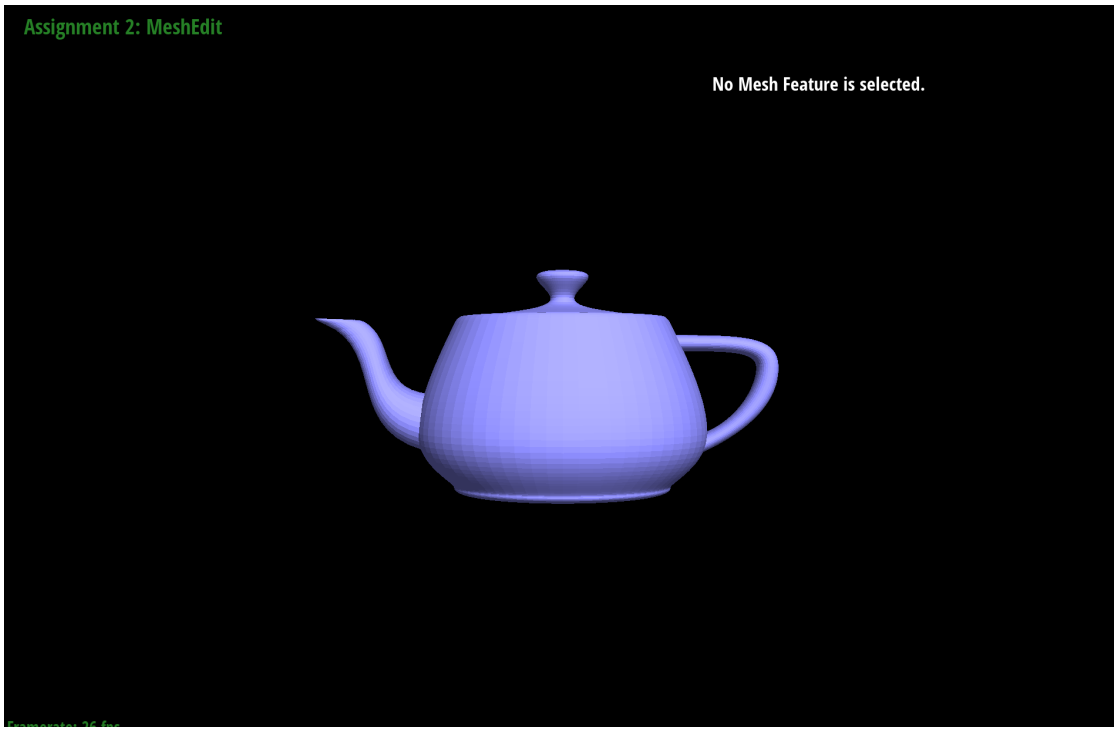
t < .5 t ≈ .5 t > .5



Another Bezier curve

Part 2: Bezier surfaces with separable 1D de Casteljau subdivision

In this part, I extended the simple 1D bezier curve to 2D. This can simply be done on a 4x4 grid of 3D vertices by creating 4, 1D Bezier curves along one direction (like all rows) and then passing the points into a final 1D Bezier curve in the other direction (all columns) to get one final point. A lot of this part can reuse the 1D evaluation from part 1, basically we do 5 total 1d bezier evalutions.



Bezier curve teapot

Section II: Sampling

Part 3: Average normals for half-edge meshes

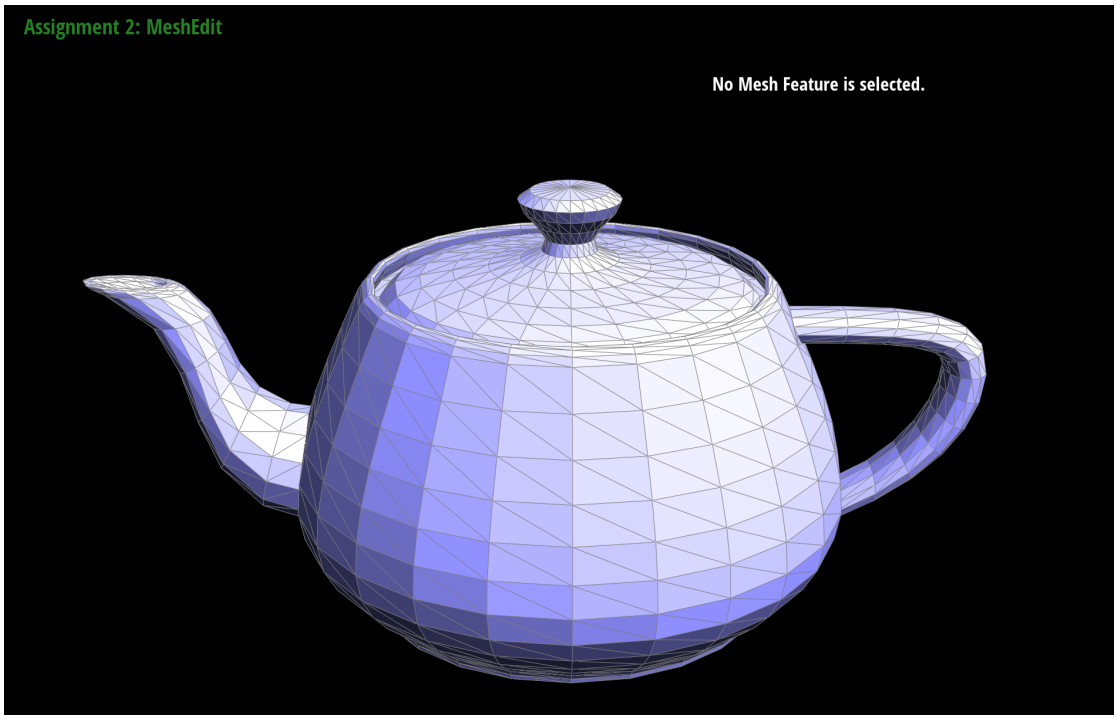
To smooth the faces of the mesh when doing shaing, we can compute the average surface normals of the neighboring vertices. To do this, we iterate using halfedges to get to the next vertex around the origin vertex via $h \rightarrow next() \rightarrow twin()$. We iterate until we get back to the start vertex. Because each vertex is a point and we want a vector, we subtract each vertex with is complement to get two vectors. We can then take the cross product to yield a normal vector. These vectors are already area weighted by definition of the cross product so we sum them and make it unit norm.

No smoothing

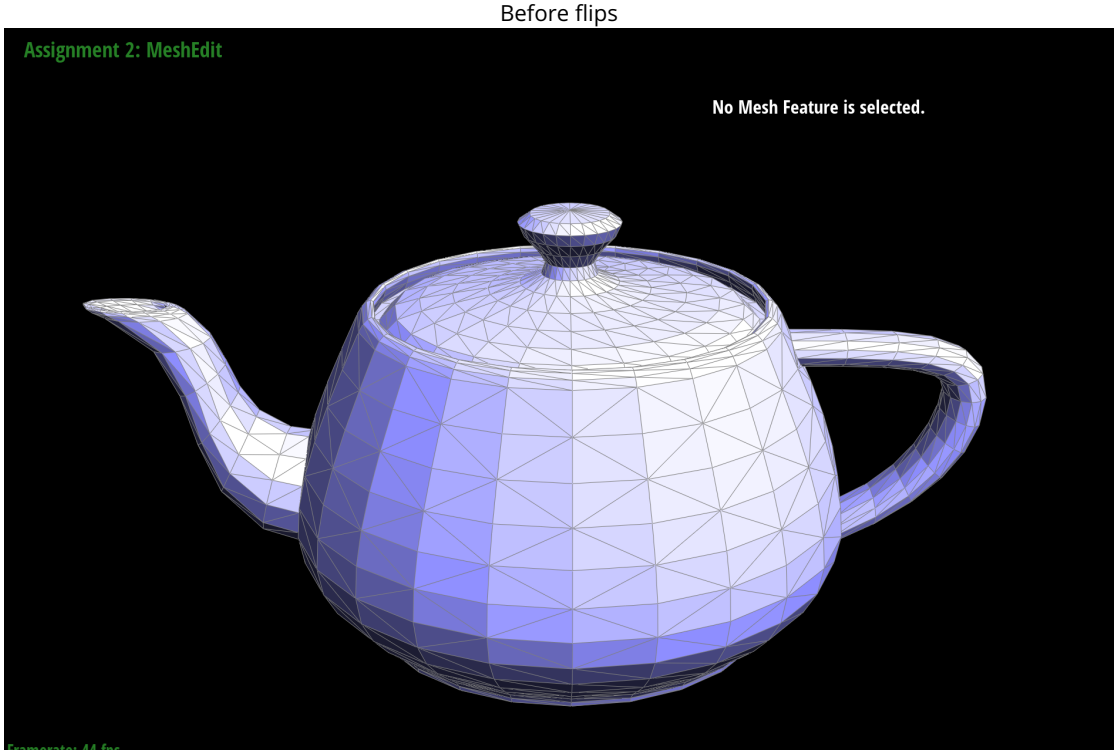
With average normals

Part 4: Half-edge flip

Basicaly for half edge flip, we have to change the pointers for all of the half edges, vertices, and faces so that it is consistent with the flipped edge. Generally we don't have to change much for the half edges on the outside of the two triangles we are flipping. There are 9 half-edges that need to be reassigned but also 2 faces, and 4 vertices that need their corresponding half edges to be reassigned.



Before flips

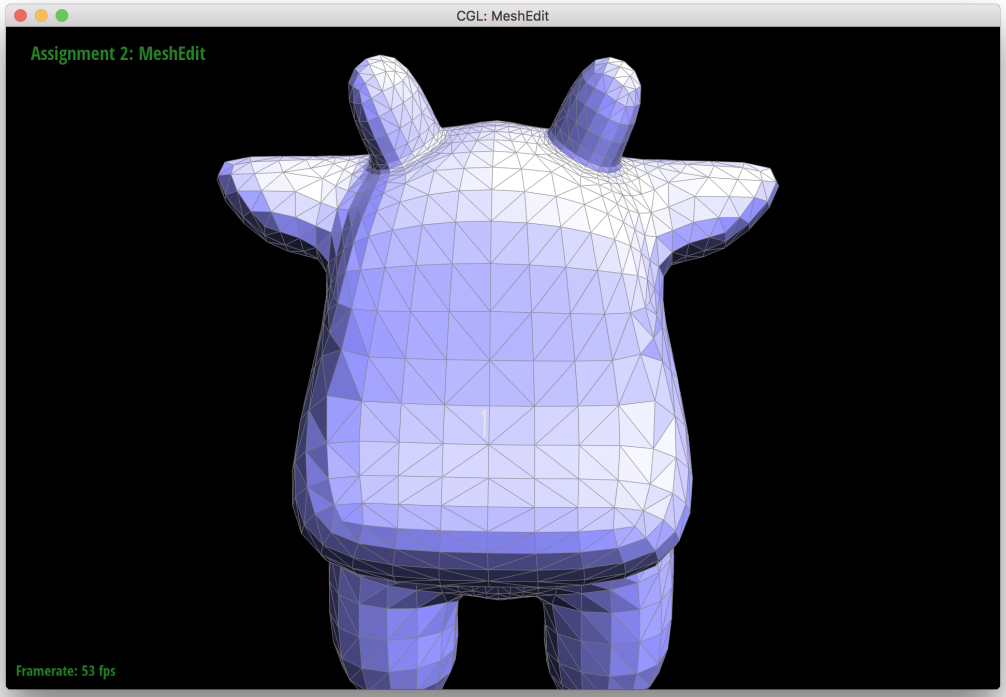


Lots of flips

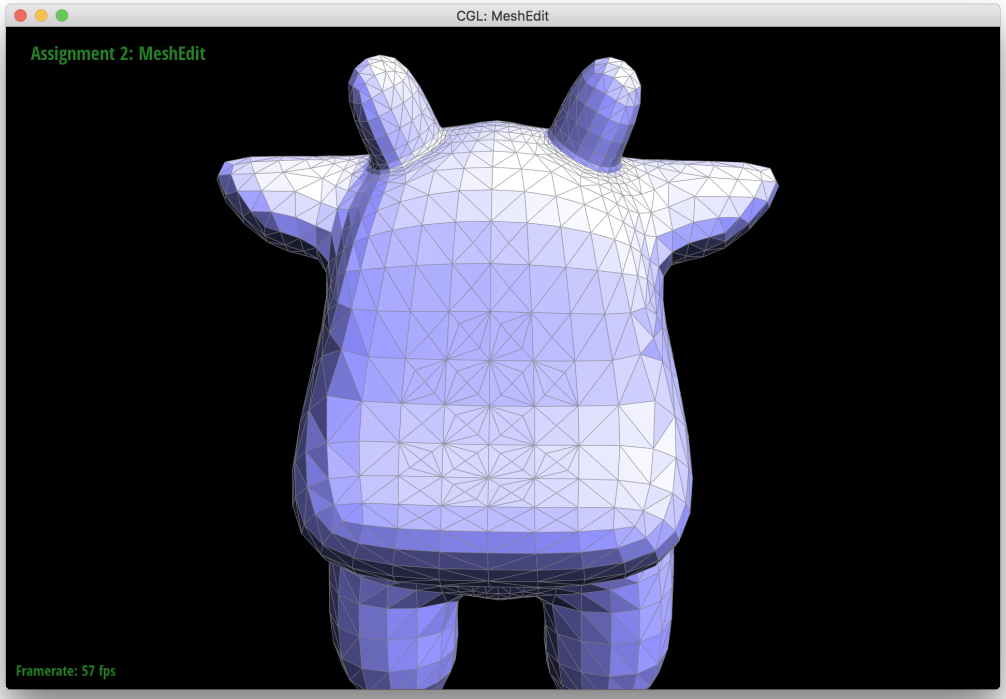
Part 5: Half-edge split

For half edge split, we need to create 3 new edges, 1 new vertex, 2 new faces, and 6 new half edges. We can then just reassing pointers to make the structure consistent that of two triangle that have been split. I used pencil and paper to make sure that my algorithm was right...

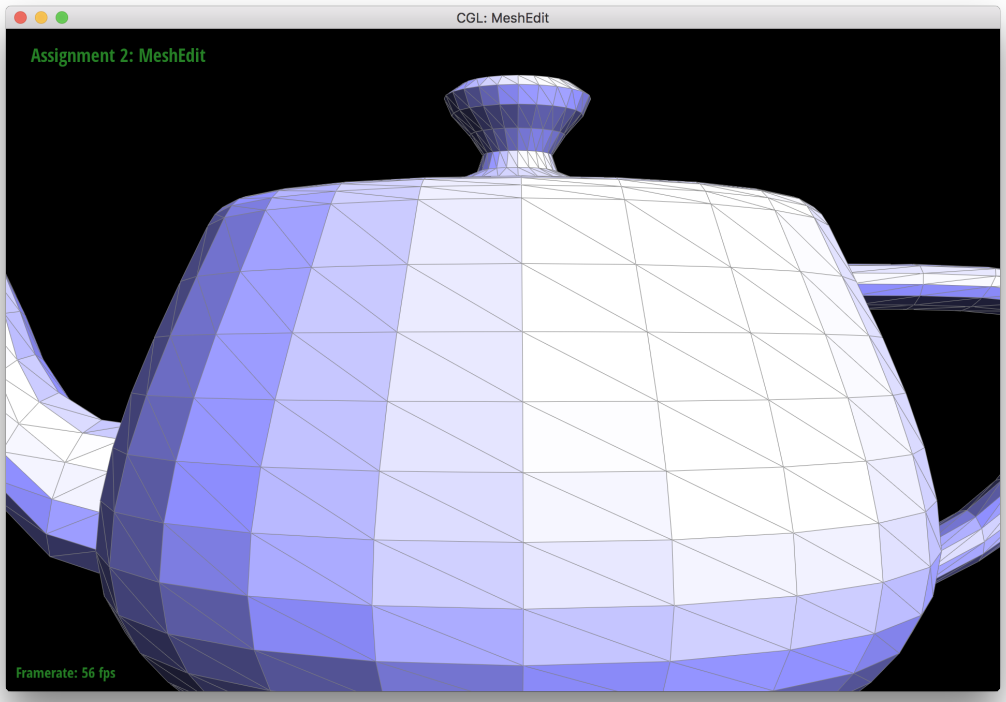
Basically for debugging, it ended up taking me two hours to find three misplaced pointer. I had to walk through each individual pointer assignment and after I found one that was misplaced I kept trying to run the code and see if it was working. Unfortunately it will just hang if it doesn't work so I knew for the first two times that I did something wrong.



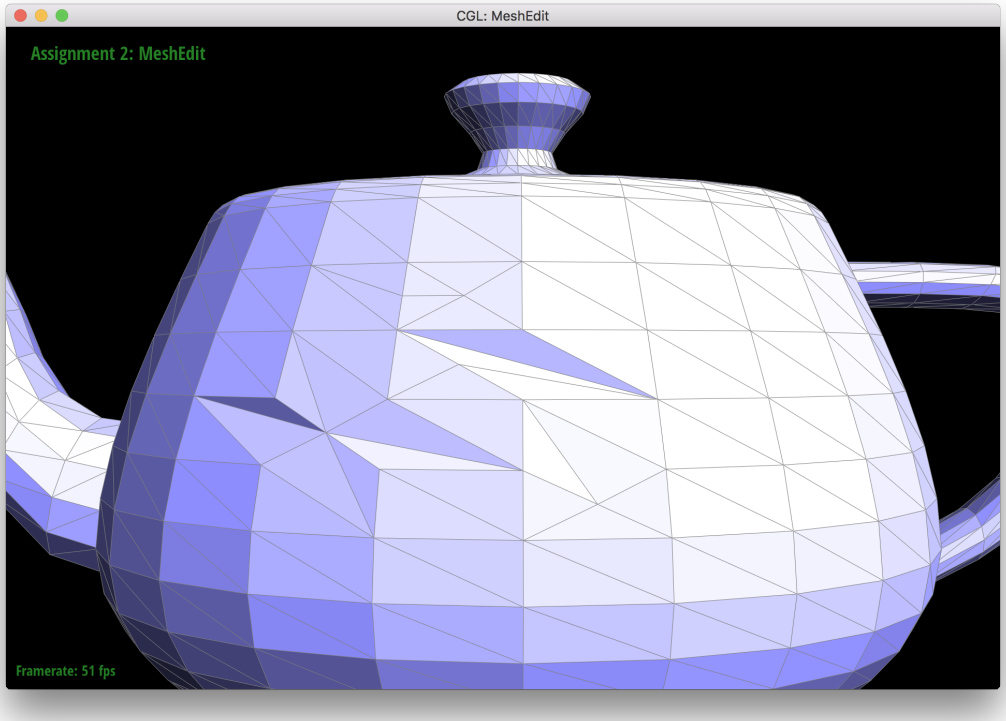
Before splits



Lots of splits



Before flips and splits



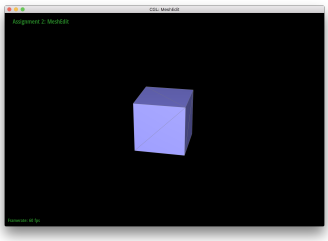
Lots of flips and splits

Part 6: Loop subdivision for mesh upsampling

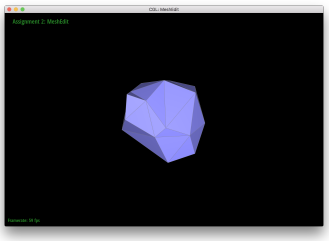
The process for mesh upsampling is as follows. First, I compute the new positions for all the old vertices by doing a weighted sum of the nigboring vertices. If $n = 3$ (where n is the vertex degree), then the weights for the other vertices are $u = \frac{3}{16}$, otherwise the weights are $u = \frac{3}{8n}$. For the vertex we update, we weight

the new position as $(1 - n * u)$ the old position and sum all the values up. We store this new position in a temporary variable. We then update all the old edge midpoints to be a sum of $\frac{3}{8}$ of the 2 vertices on either end of the line and $\frac{1}{8}$ of the vertices at the end of the line if it was flipped. We also store this in a temporary variable. For both edges and vertices, we set is_new to be false and we also change the code for edge split to set is_new to be false for 1 edge (part of the old "edge") and is_new to be true for 2 new edges.

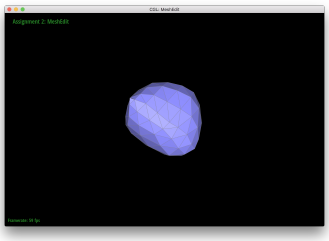
We then iterate over all of the edges we modified previously and split them, setting is_new to be true and putting the vertex position at the newly cacluated midpoint of the edges. We then flip all edges that are connected to both a old vertex and a new vertex. Finally we update all vertex positions using the temporary variable and set all is_new attributes to be false.



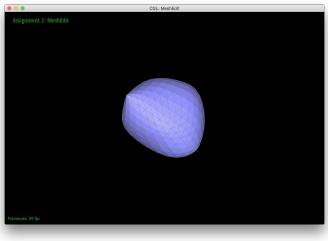
Normal cube



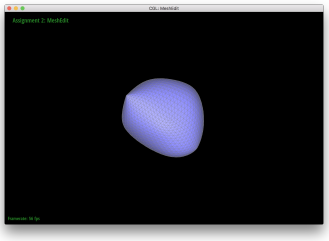
1 upsample



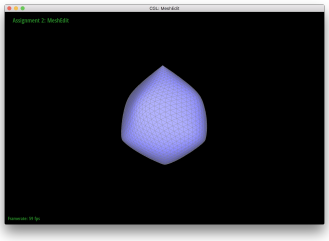
2 upsamples



3 upsamples

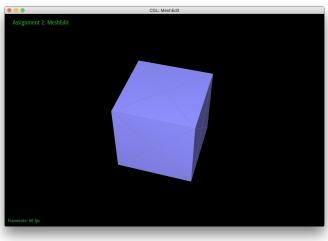


4 upsamples

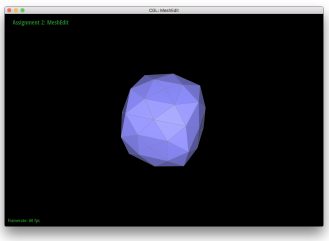


4 upsamples, rotated

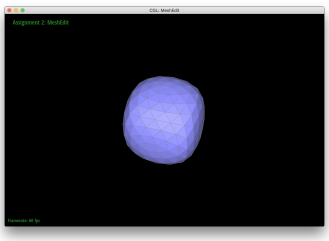
As shown above, the result does not always divide symetrically, even when the original mesh shape is symetric. This happens because the mesh itself may not be symmtric. In the case of the cube, the corners that don't have face edges touching it usually become spiky whereas the other parts of the cube are smooth. This is caused by the fact that we are upsampling in some places that are smooth but then we do the same amount of upsampling in places that have a large change in distances. We can resolve this by splitting all edges on the cube so that each corner has 3 face edges touching it.



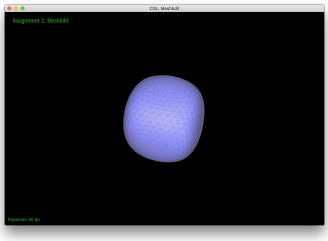
Cube with all face edges split



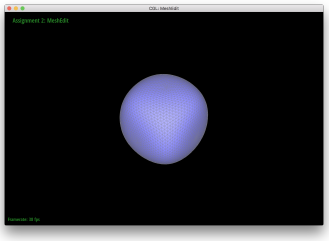
1 upsample



2 upsamples



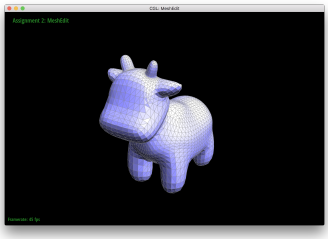
3 upsamples



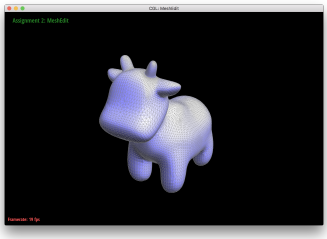
4 upsamples

As seen, the results look a lot better and much more symmetrical.

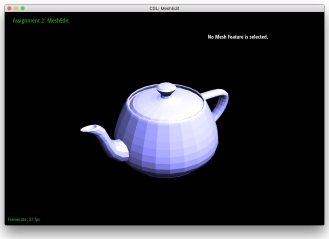
Extra examples below. It seems to be common that sharp corners and edges when upsampled still stay sharp, even when the rest of the mesh is smoothed. A simple solution to this is presplitting the edges around these sharp points before upsampling.



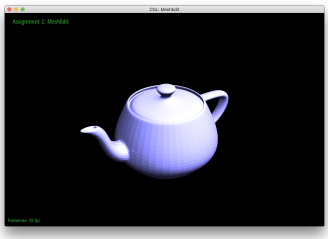
Original



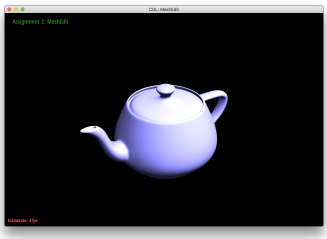
1 upsample



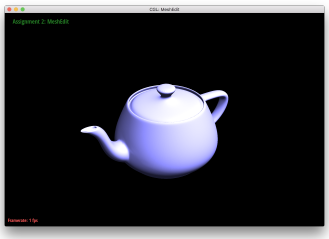
Original



1 upsample



2 upsamples



3 upsamples

Section III: Mesh Competition

If you are not participating in the optional mesh competition, don't worry about this section!

Part 7: Design your own mesh!

No designs here!!