Sam Zhou

CS184 Project 2: MeshEdit

The goal of this project was to create a tool which can render and modify 3D meshes of surfaces. First we will look at Bezier curves/surfaces as a means of rendering sufaces. And then we will move onto general triangle meshes by making use of the half edge structure. With these half edges, we will implement flipping and splitting edges as well as upsampling via loop subdivision.

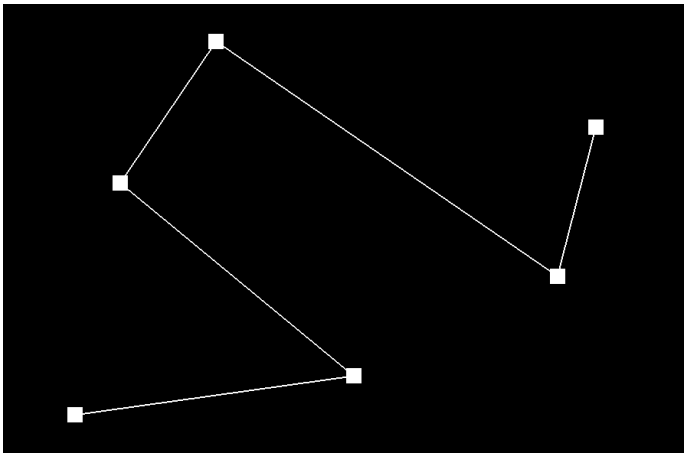
Section 1: Bezier Mesh

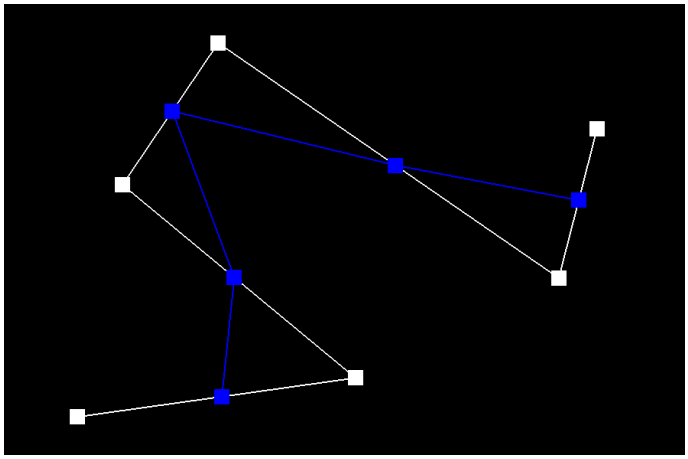
de Casteljau's Algorithm

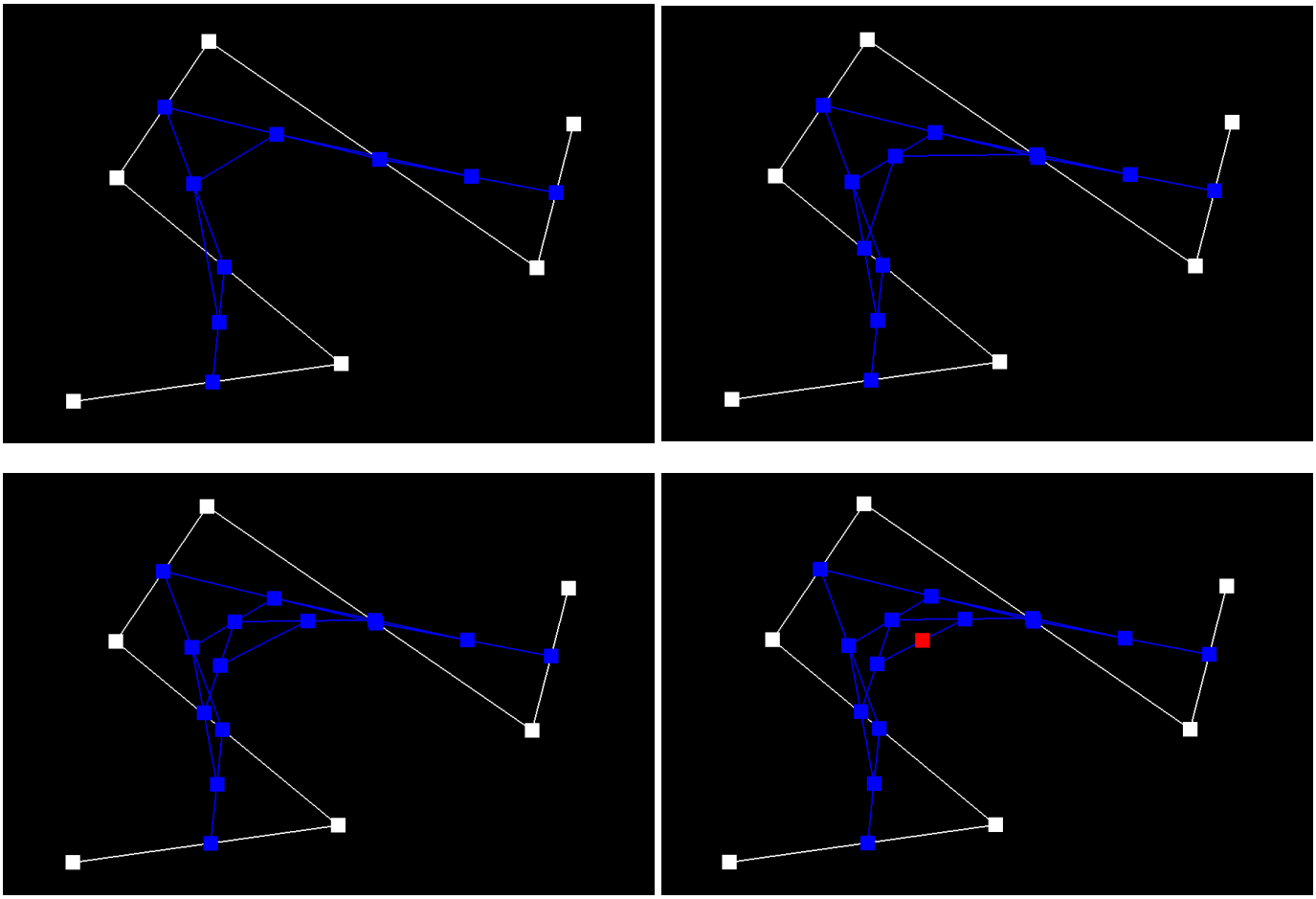
First, we will just consider rendering splines correctly given a set of control points that define the curve. Using de Casteljau's algorithm, we can define the curve as a function of **t**, a variable ranging from 0 to 1.

For a given **t**, we calculate its position by iteratively decreasing the number of intermediate points by 1 until there is only 1 left. In each iteration, we take every pair of adjacent points and linearly interpolate between them using t as the parameter for the interpolation. Doing this gives us one fewer point than the previous iteration and we continue this process using our new set of intermediate points. Once there is only 1 point remaining, this is the point on our curve that is **t** percent of the way from the start.

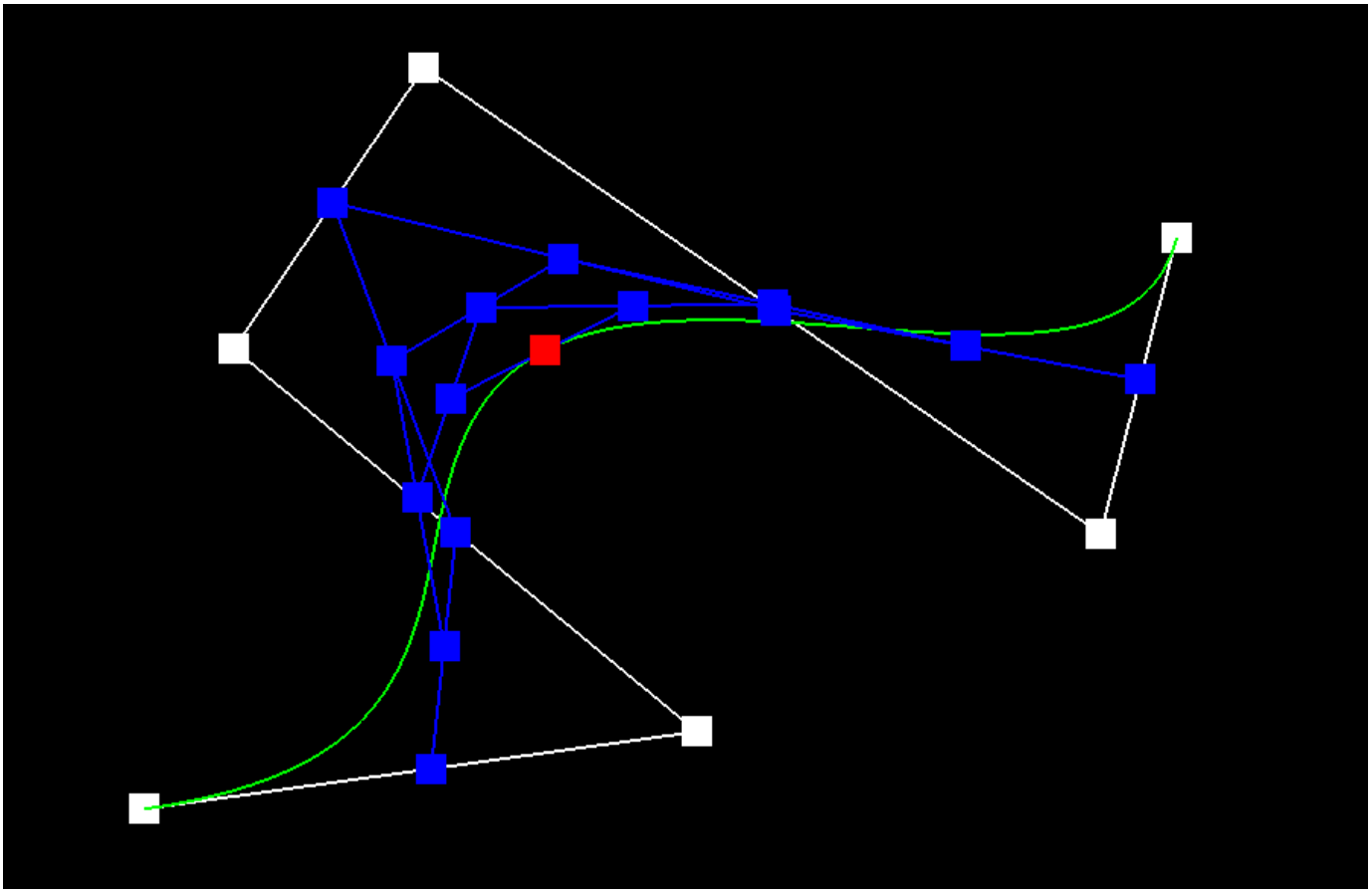
Here is an example on a Bezier Curve with 6 original control points. In each step you can see one fewer intermediate point is added until we converge on the red point which is on our curve.



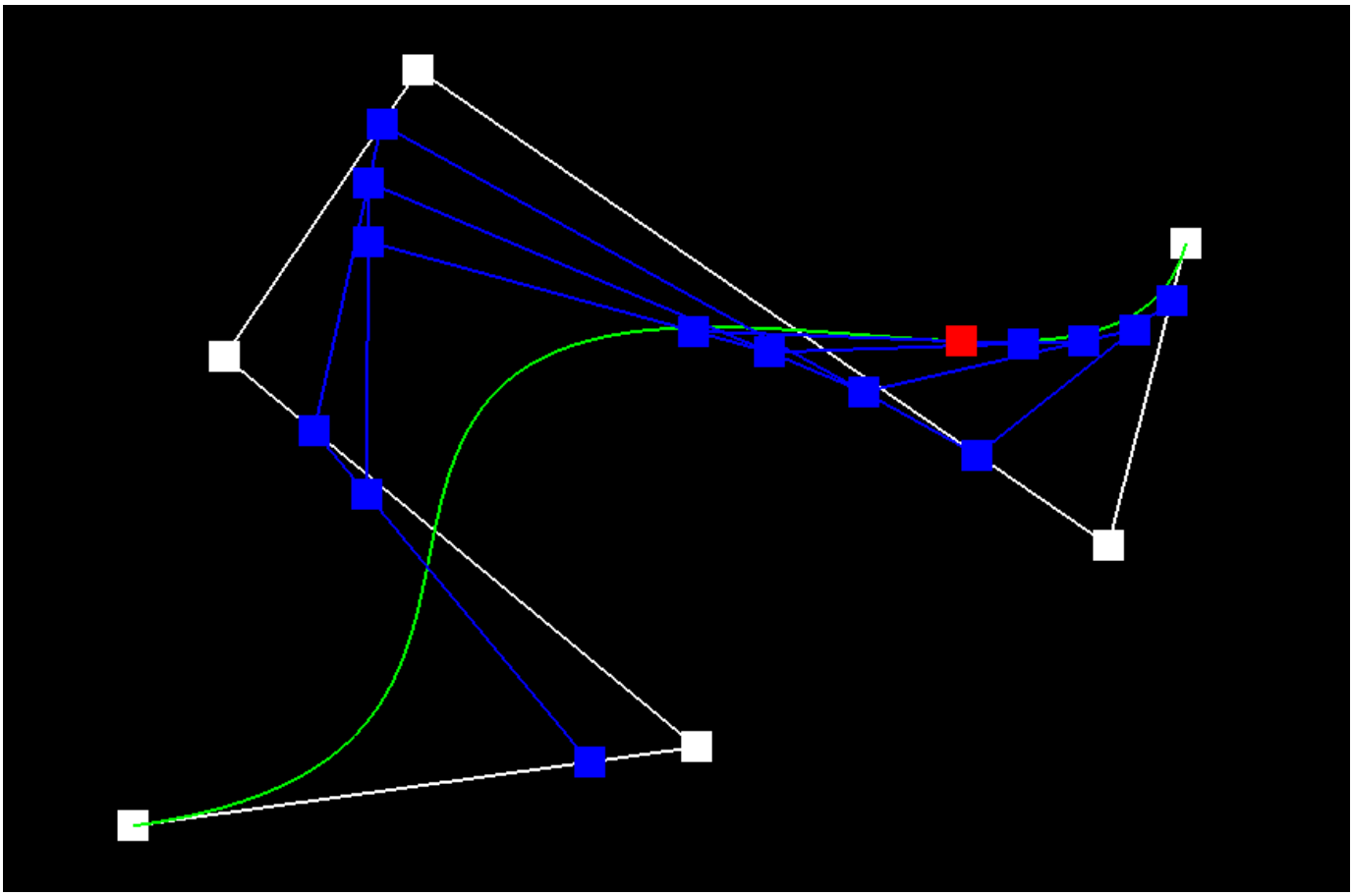




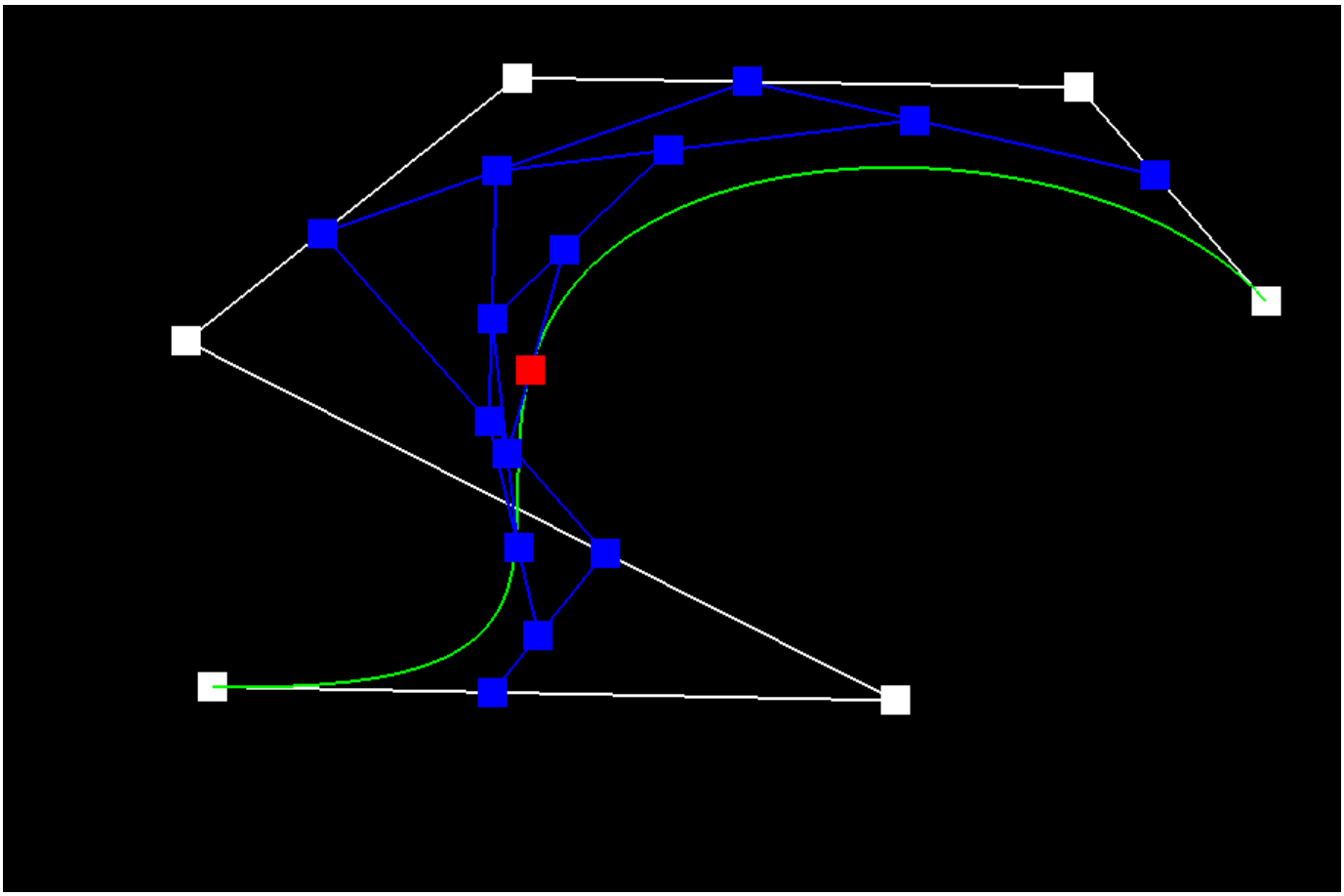
And finally here is the full image including all the intermediate points as well as our curve.



We can scroll through different values of **t** to move the red point and see how the intermediate points change along with it.



Another example of the full curve drawn out but with slightly different locations.

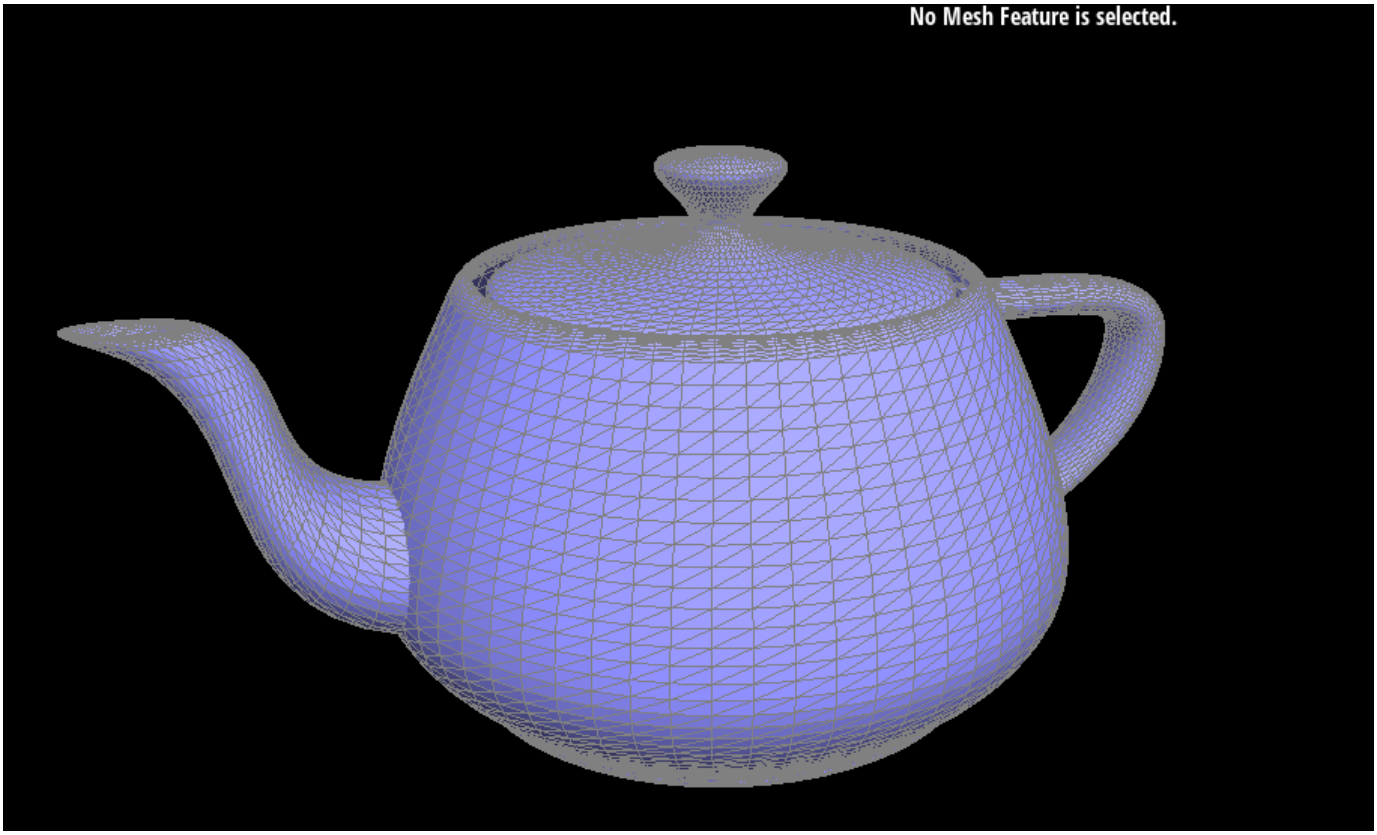


Bezier Surfaces

The usage of de Casteljau's algorithm extends to surfaces as well. In this case, we have two parameters to define the surface instead of one, **u** and **v**. We also require a grid of 3D control points instead of a list of 2D control points. In this part, we are only going to work with 4x4 grids of control points.

Applying de Casteljau's algorithm to 3D space just requires multiple applications of the 2D case. We break apart the grid into rows first and use the same algorithm as before except we are using **u** as the parmeter. Instead of giving us a point on our surface, this will give us 4 new control points which we can use in another application of de Casteljau's. This time we use **v** as our parameter and get one point which is on our surface.

This technique allows us to render surfaces using Bezier control points. Here is an example of a teapot rendered using this technique.



Section 2: General Triangle Mesh

Moving on from Bezier Meshes, we will support general triangle meshes by using the half edge data structure. This structure is useful for implementing operations on the mesh as it gives us a good way to define relations between elements in our mesh.

Average Normal Smoothing

First, we will implement a method for better local shading. To achieve this, we need to calculate the area weighted average normal vector at every vertex. For a given vertex, we can calculate by walking around the vertex and summing the norm given by every triangle which has this vertex as one of its points. We do this in a loop moves from triangle to triangle using calls to twin() and next(). For each triangle, we grab the two vertices opposite the central vertex and create a two vectors by subtracting the central vertex from the opposing verticies. Then we can take the cross product of these two vectors and add it to a running total. After we iterate over every triangle, we can normalize the vector and we have the area weighted average normal vector.

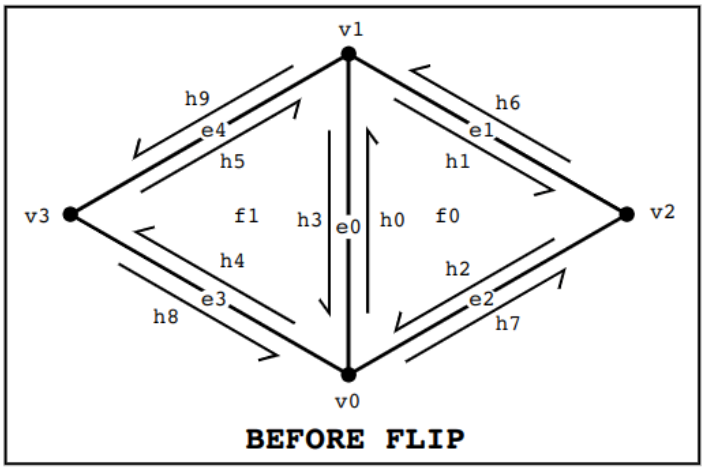
Here is an example of the improved shading on our teapot. It smooths out all of the sharp ridges produced by the edges between triangles in our mesh.

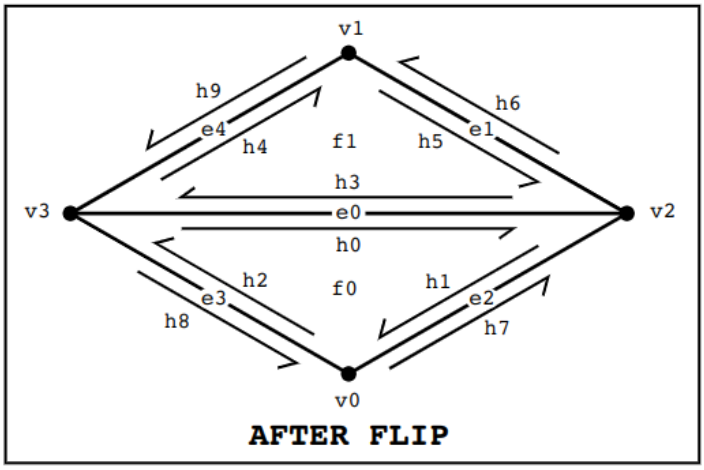
Original Shading

Improved Shading

Half-edge Flip

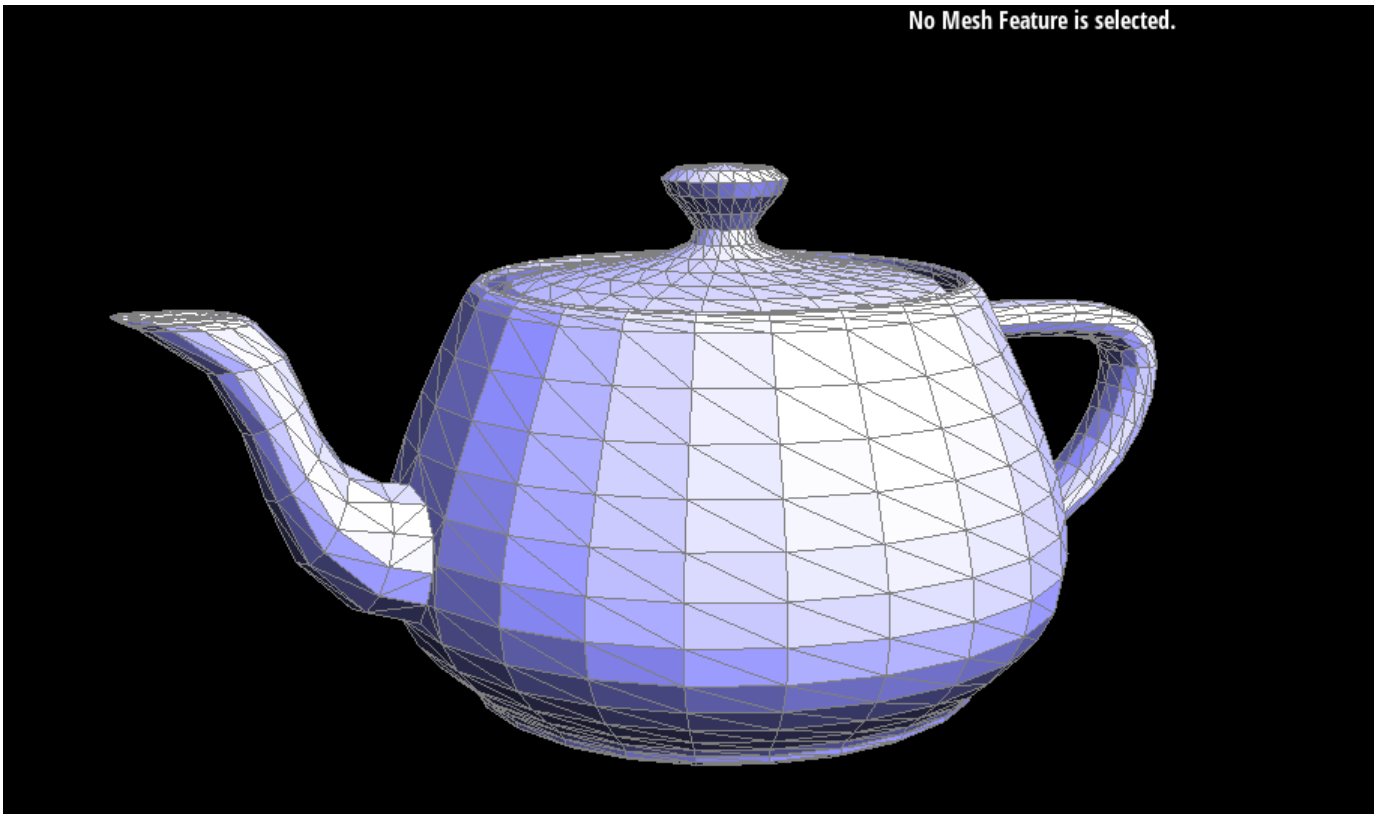
The first operation we will implement is an edge flip. To perform the flip, we only need to reassign points so that the relationships between half edges, edges, vertices, and faces reflect what would happen after a flip. Using the provided diagram (shown below) as a reference, every reference was explicitly reassigned even if it was unnecessary. For this part, I did not have to debug anything after just being very careful while typing in the references.



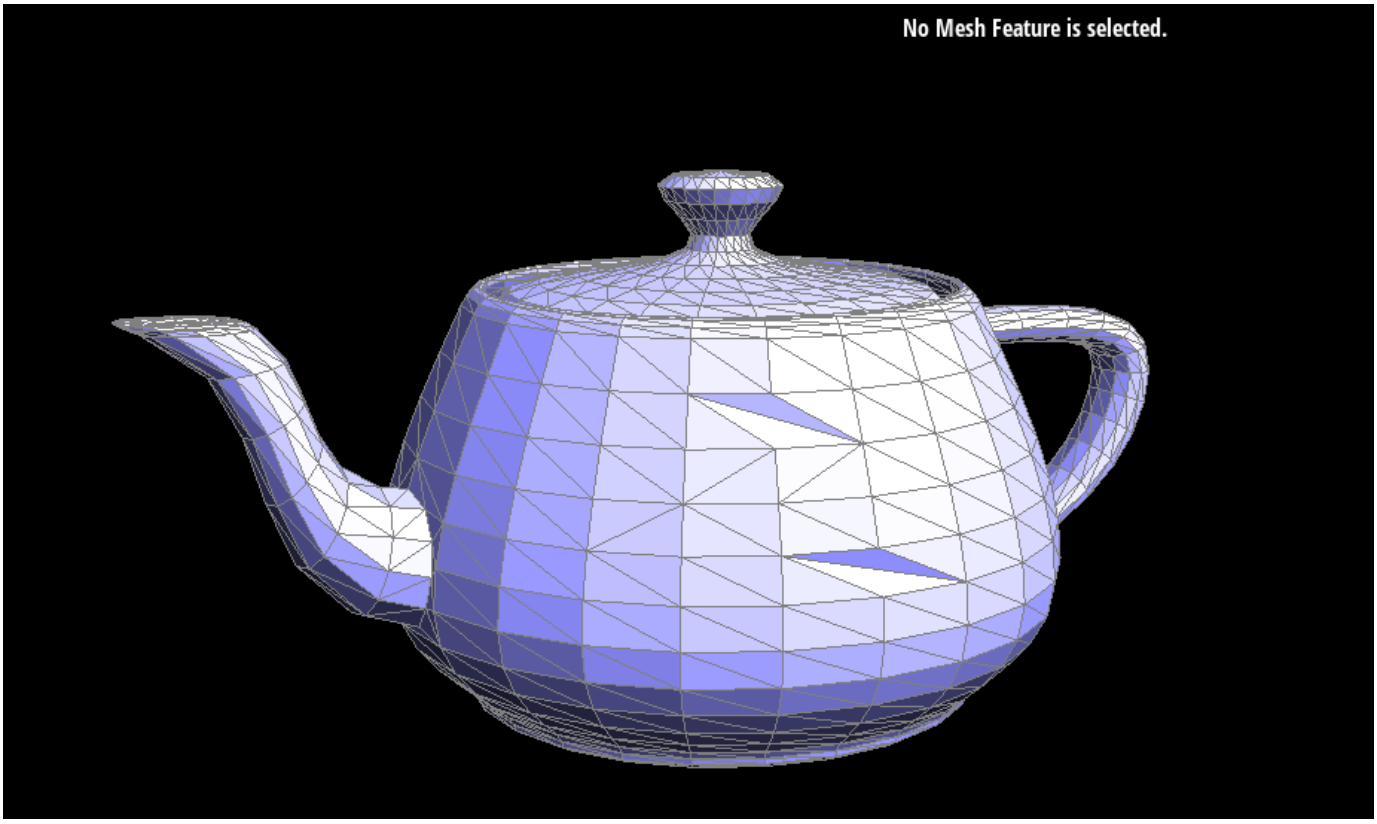


Here is an example of some edge flips. On the surface of the teapot you can see the mesh lines which were flipped. Some flips do not change the topology of the teapot significantly at all, but you can also see some examples here which caused indents in the teapot because of flipping an edge.

Original



A few flips

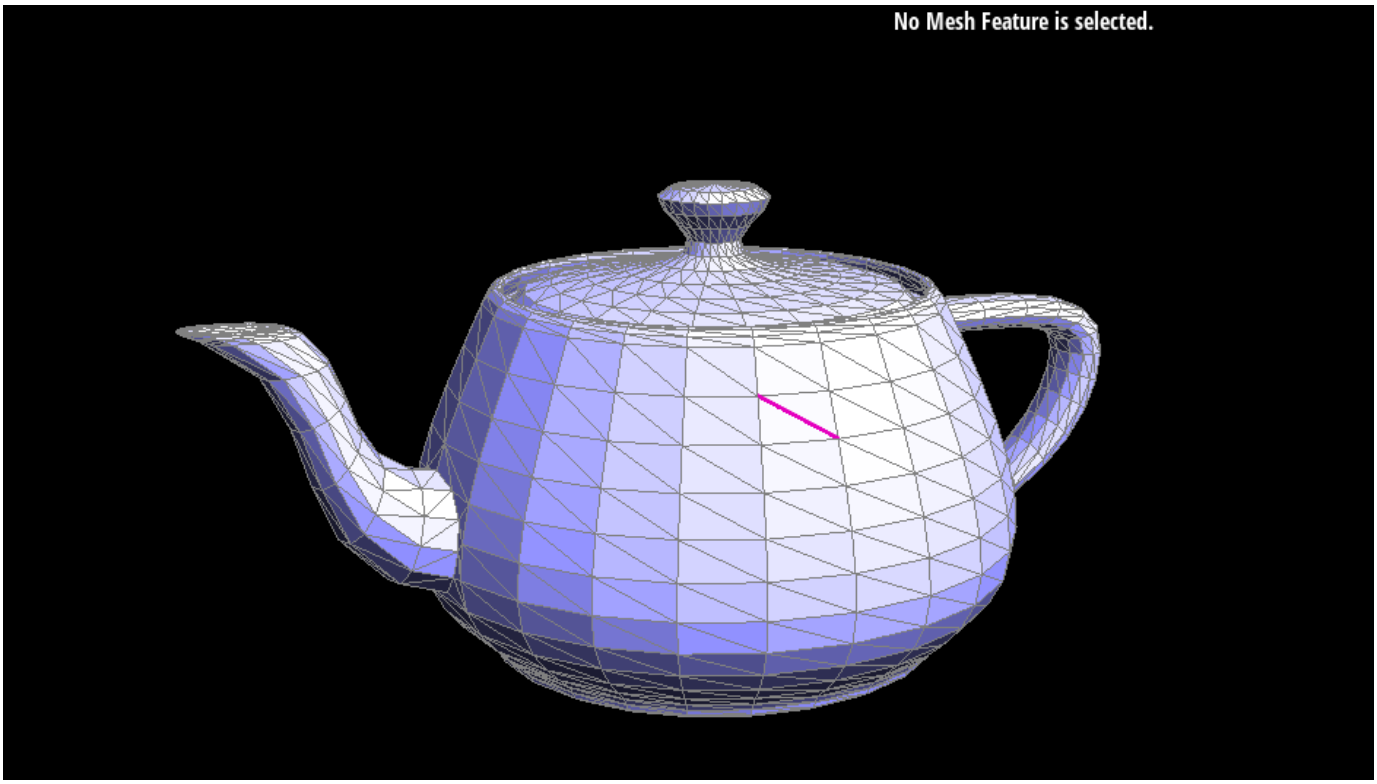


Half-edge Split

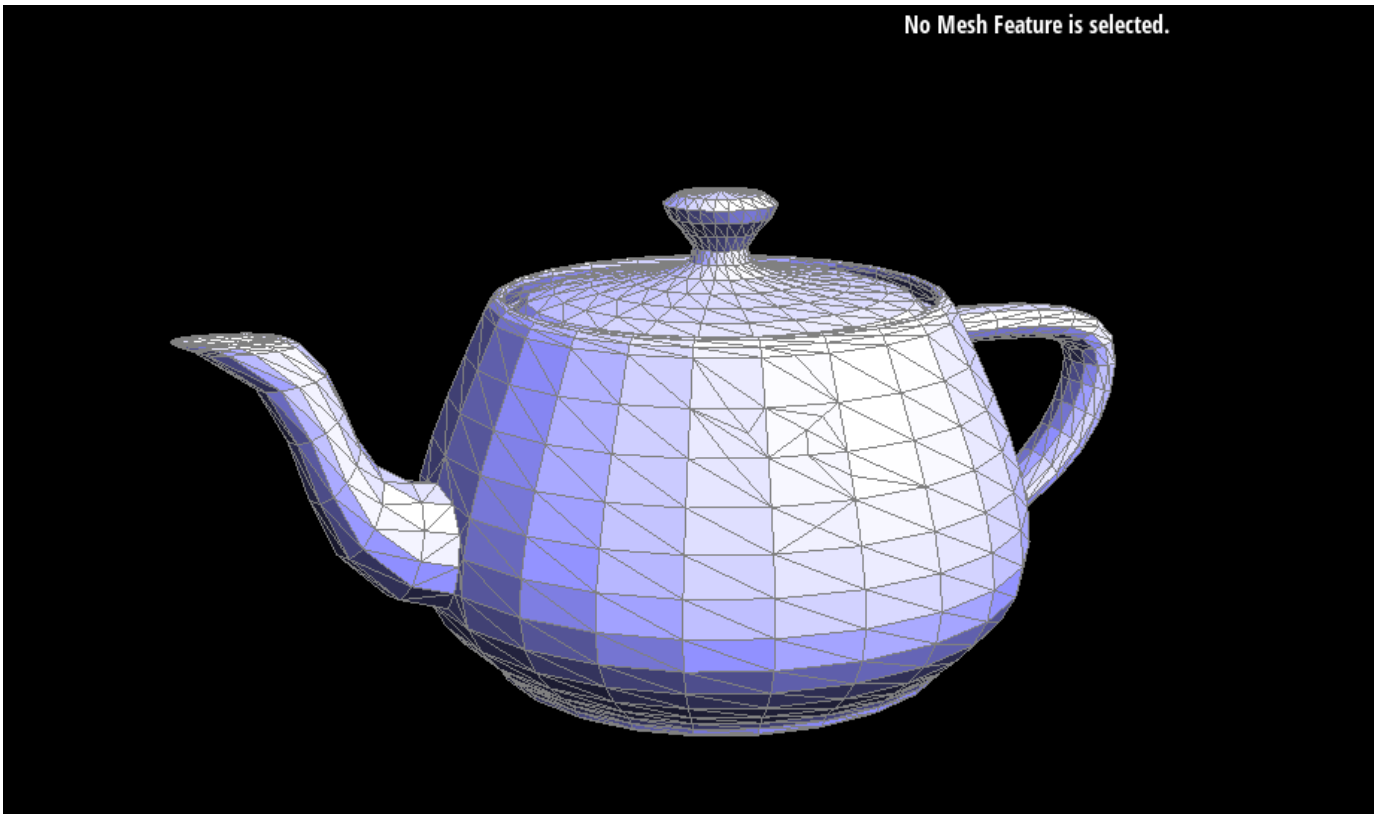
The split operation was implemented very similarly to the flip operation. In this case, however, I had to draw my own diagram and there are many more mesh elements involved. Also this time, six new half edges, one new vertex, three new edges, and two new faces had to be created. Again, I did not have to do any debugging after just being very very careful while typing in the references.

Here is an example on our trusty teapot. First we just split some edges on the face of the teapot as seen in the second photo. Then we perform a few flips and a few more splits/flips to get to our third image. This is to double check our two operations work in conjunction with each other.

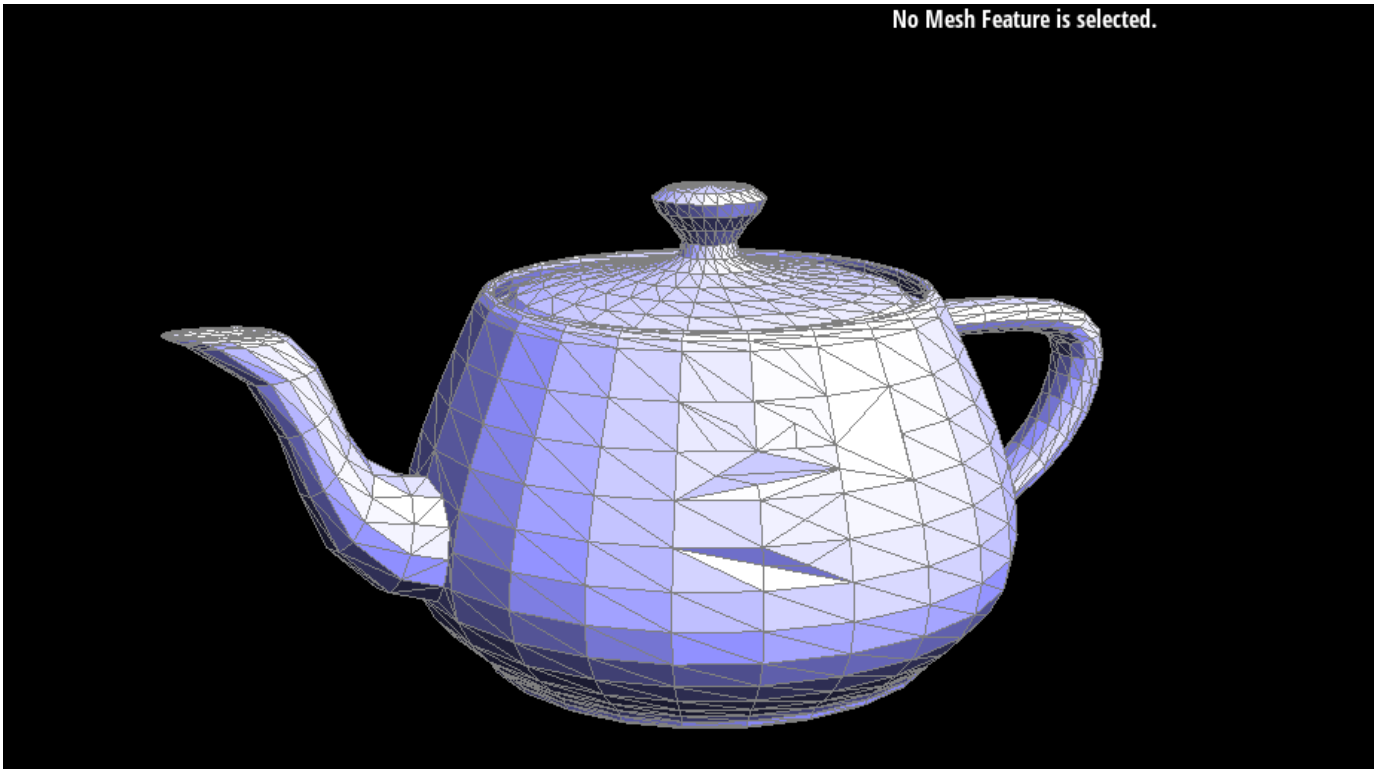
Original



A few splits



Some flips and more splits

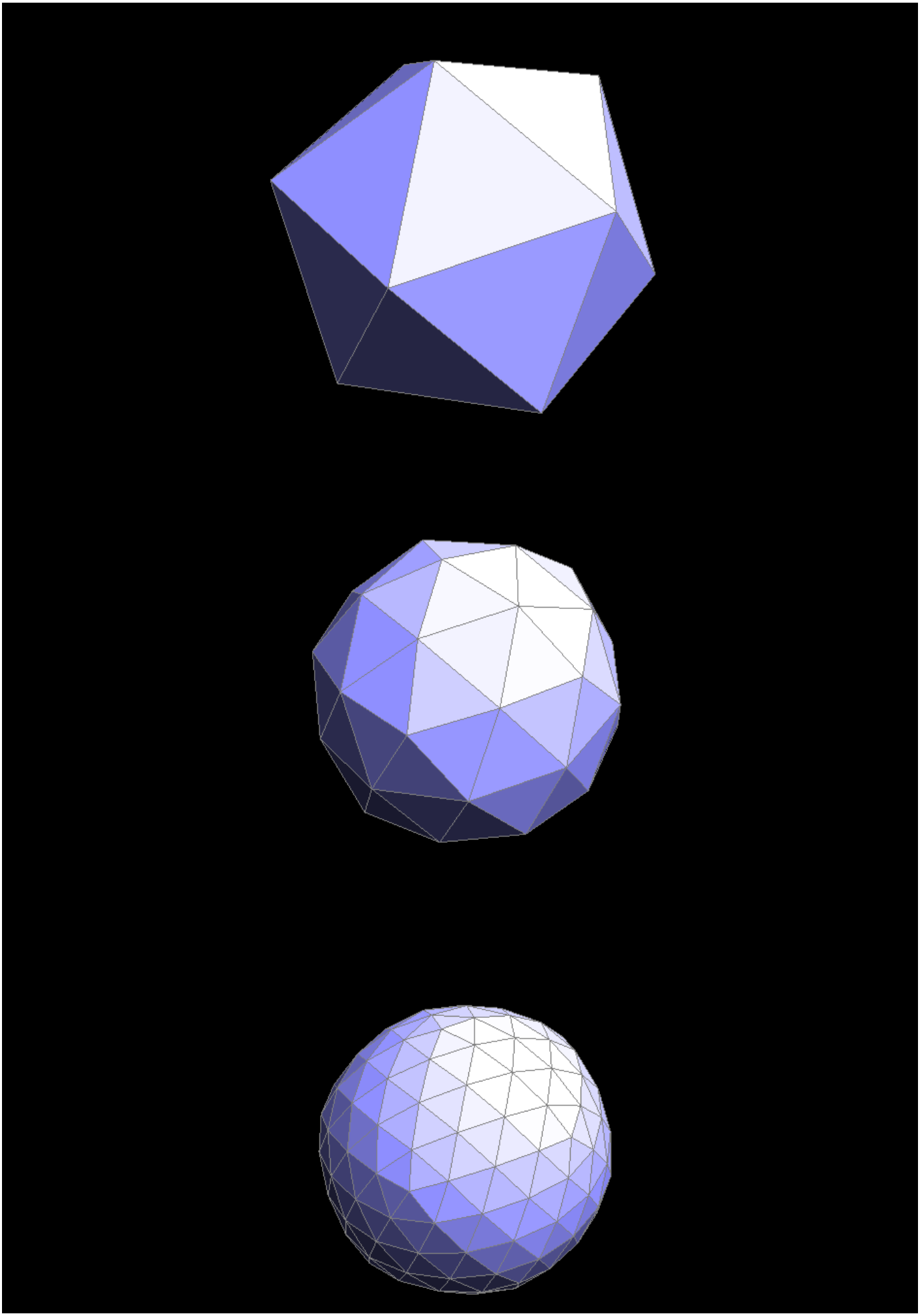


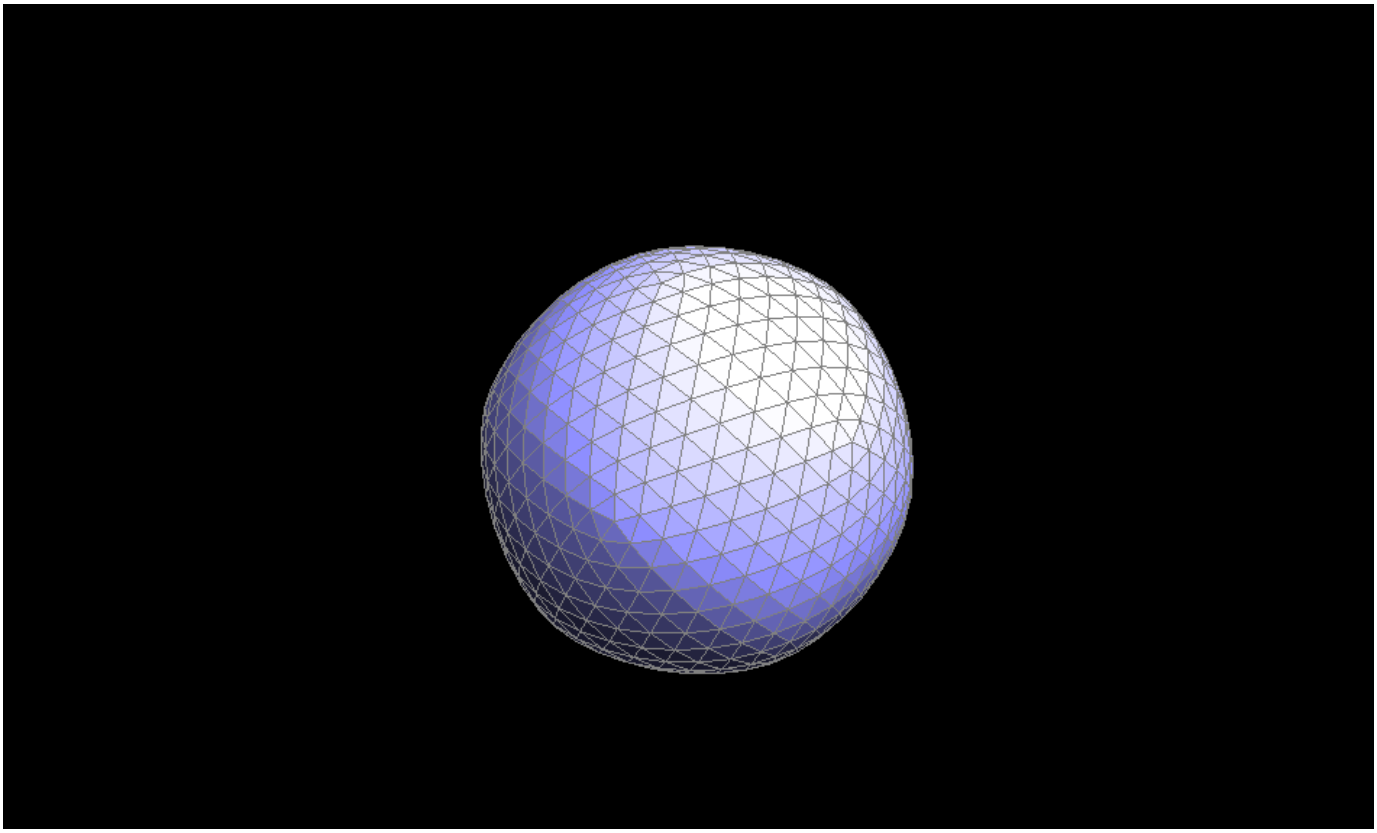
Loop Subdivision

To achieve upsampling, we will take our triangles and perform a 4-1 subdivision on each of them so that our mesh is composed of 3 times as many triangles. Following the procedure given in the project description, I used 5 total for loops.

The first just marked every vertex as an old vertex and also calculated their new position and saved it for later.
The second iterated through every edge, calculated the new position for their midpoints, marked them as old, and added them to a separate vector for later.
The third iterated through the vector of saved edges and called split on them. For this step, I also modified split to mark vertices and edges as new as appropriate. After the split call returned the new vertex, I also took the precalculated edge midpoint positions and saved them in the new vertices.
The fourth iterated through every edge and called flip on them if they were a new edge which connected a new vertex and an old vertex.
The fifth iterated through every vertex and updated their position using the precalculated new positions.

Here is an example of upsampling on an icosahedron. As we upsample more and more, it becomes rounder and rounder. By the end it almost looks like a sphere.

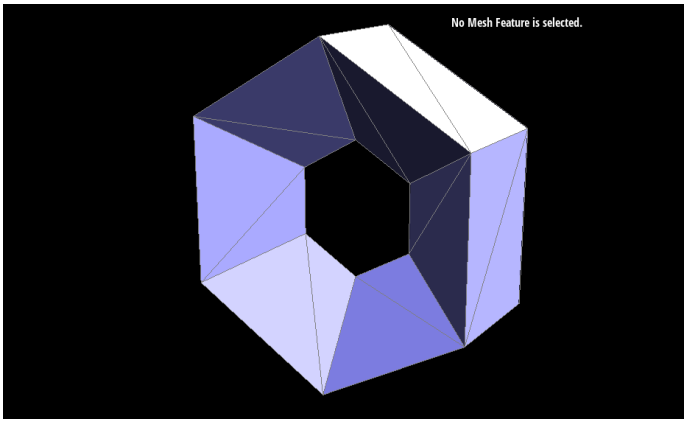


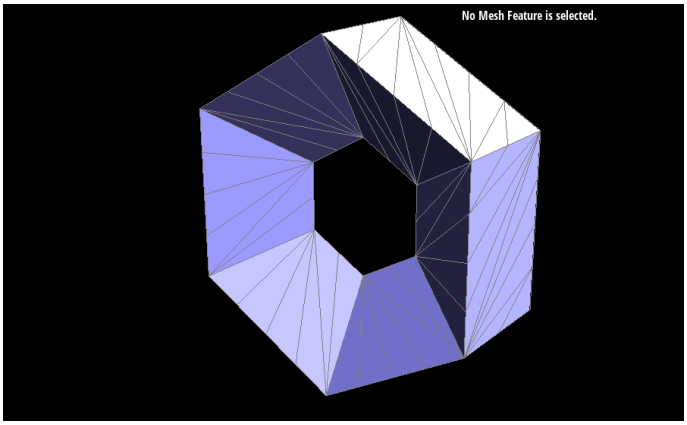


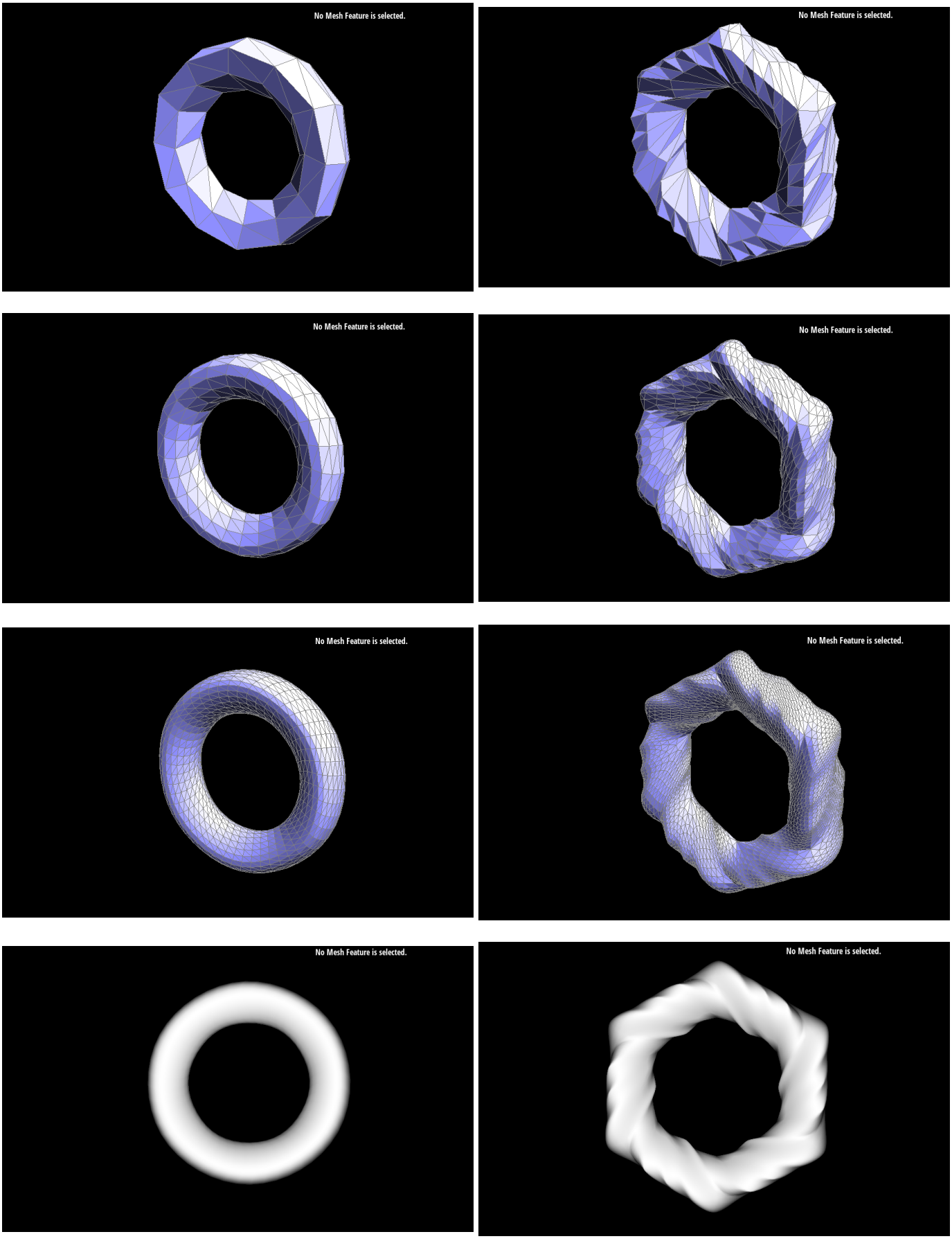
Loop Subdivision Observations

Loop subdivision for upsampling smooths out the mesh as a whole. This gives the appearance of a more detailed/cleaner mesh for some, but it does this by doing a lot of averaging. This leads to the loss of sharp corners. This can be seen very clearly with the original torus example pictured below. With 3 applications of upsampling, we get a smooth ring even though we started with a clunky hexagon. If we want to reduce this effect, we need the original mesh to have more triangles in it so that even with upsampling, the original shape is preserved. On the right column, you can see that even though many edges are still smoothed over, the ring maintains its overall hexagonal shape in the end.

Original ---------- **Preprocessed**



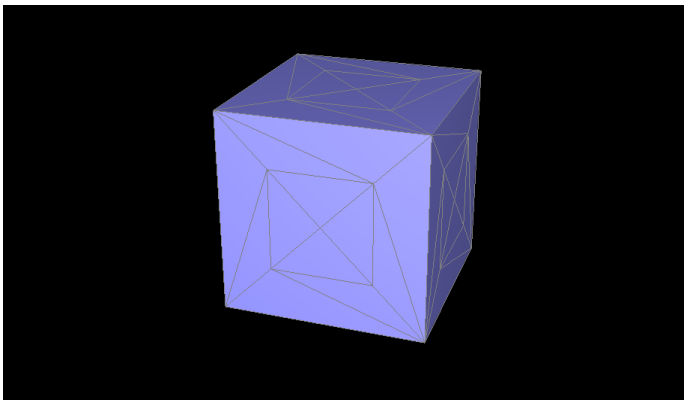
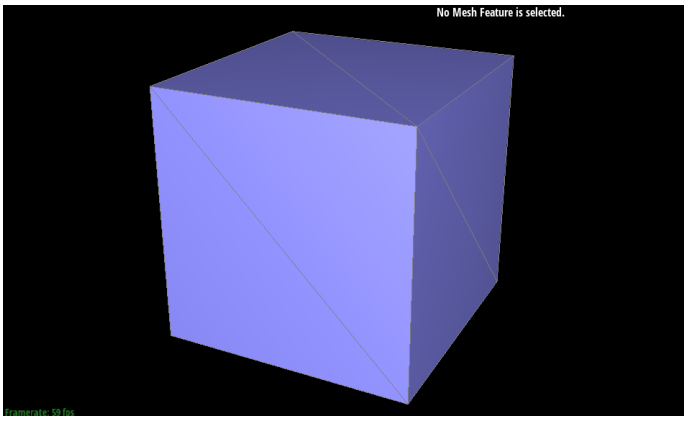


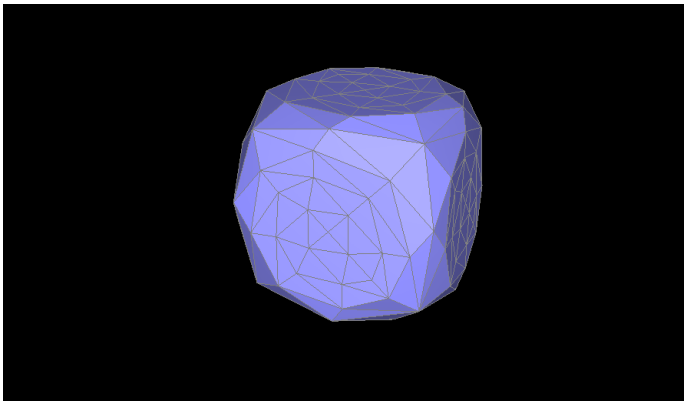
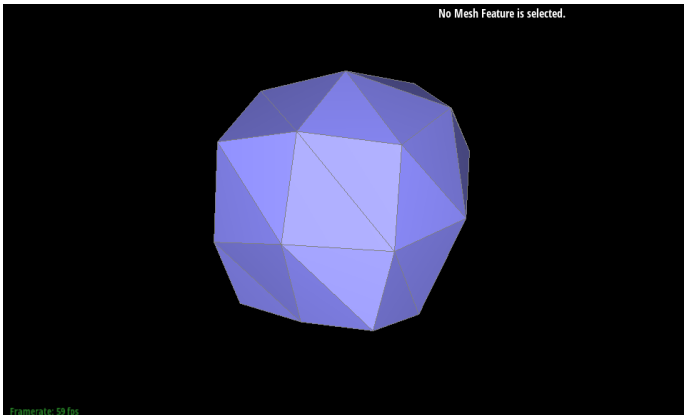


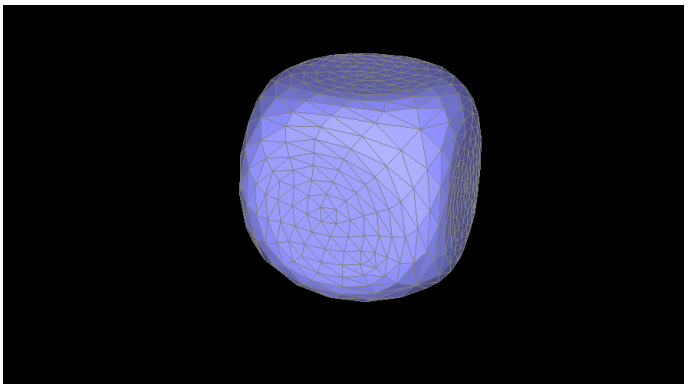
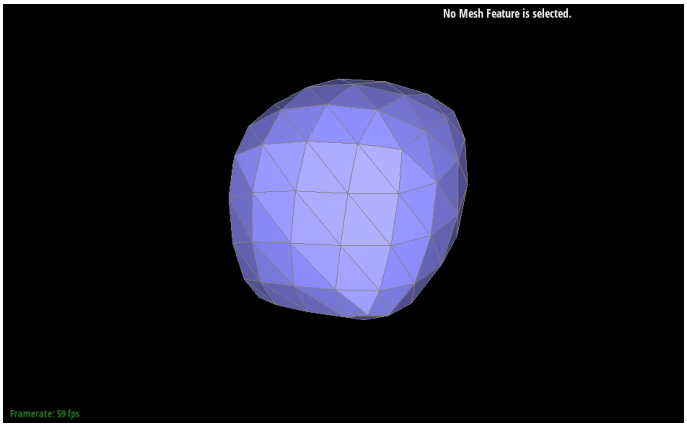
Another point of interest is in how the symmetry of the original mesh affects the symmetry of the resulting upsampled mesh. On the right hand column below, the cube slowly becomes asymmetric and looks like a strange pillow. This is because the cube is composed of triangles

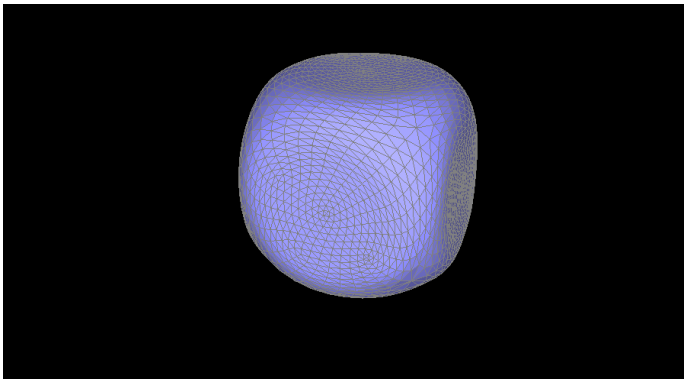
which are not radially symmetric on each face. The resulting shape will not have symmetry on any face either because of this. On the left, you can see the effect of preprocessing the mesh. By using split to make the mesh radially symmetric, the overall shape of the cube is better preserved and the symmetry of the shape is preserved.

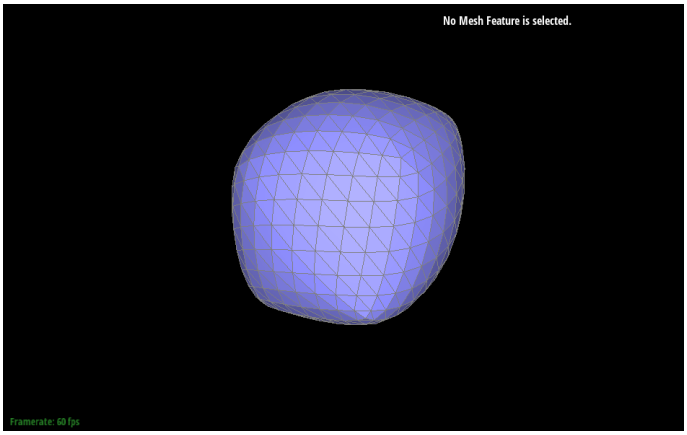
Preprocessed ---------- **Original**

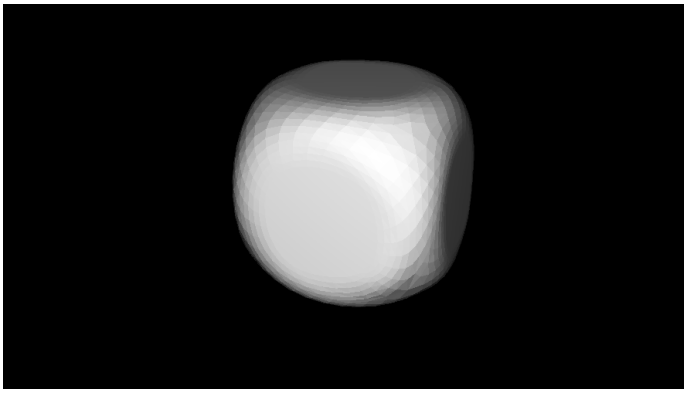



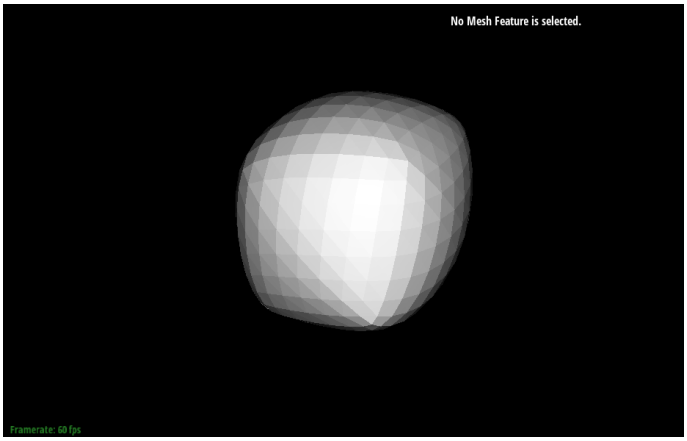






No Mesh Feature is selected.
Framerate: 60 fps




No Mesh Feature is selected.
Framerate: 60 fps