CS 184: Computer Graphics and Imaging, Spring 2019

Project 2: Mesh Editor

YOUR NAME

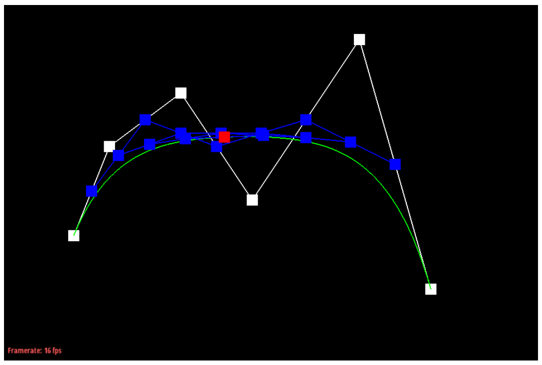
Overview

Give a high-level overview of what you implemented in this project. Think about what you've built as a whole. Share your thoughts on what interesting things you've learned from completing the project.

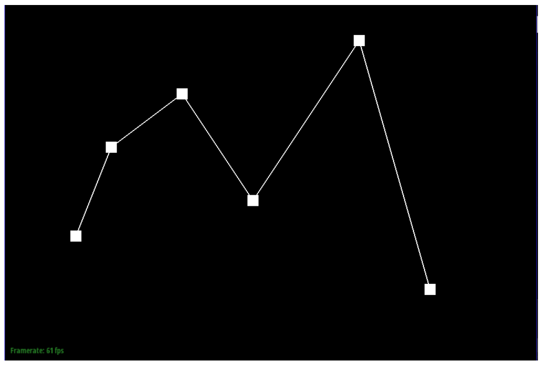
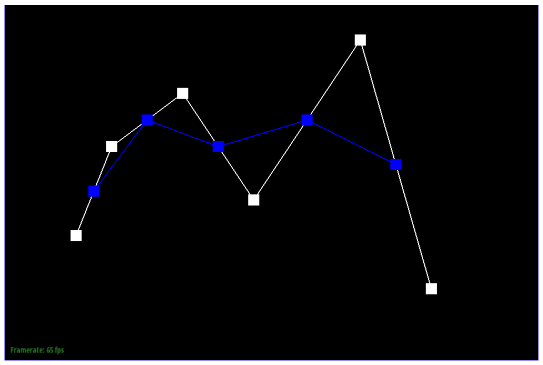
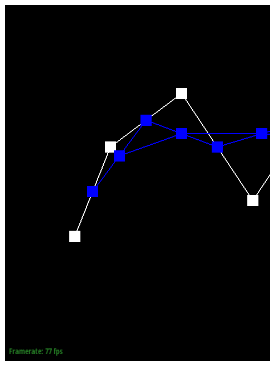
Section I: Bezier Curves and Surfaces

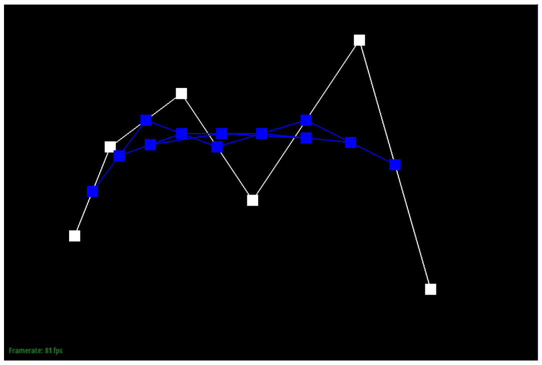
Part 1: Bezier curves with 1D de Casteljau subdivision

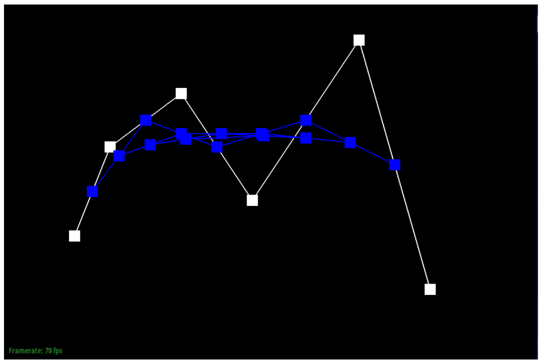
Here is an example 2x2 gridlike structure using an HTML table. Each **tr** is a row and each **td** is a column in that row. You might find this useful for framing and showing your result images in an organized fashion.

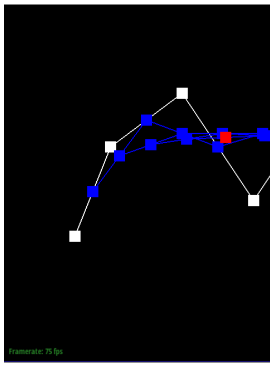


Six point curve.

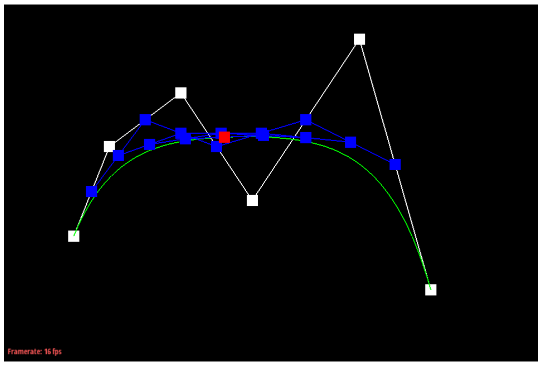
 Bezier Curve with six points.	 Step 2:	 Ste



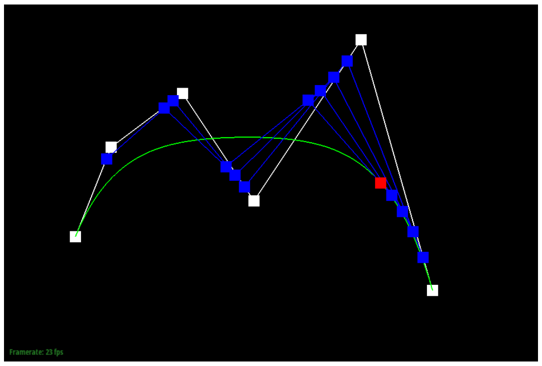




Step 4: Step 5: Ste



Normal Bezier Curve.

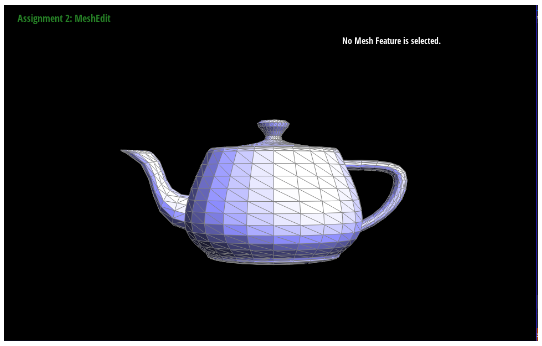


Curve with modified t.

De Casteljau's Algorithm is a method used to evaluate Bezier Curves using a recursive relation. It starts by connecting adjacent points with lines and choosing some value **t** between 0 and 1. With this value, they split all lines between points where the first half of the line had length **(1 - t)** and the second half is **t**. There is a point put on this split and every adjacent point created through this process is connected by a new line and the algorithm repeats until the process is down to one point formed through this process and the point traces the curve.

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

De Casteljau's Algorithm can also be applied to surfaces. Given two paramaters **u and v**, we first take all of the points and evaluate the Bezier curve for all pairs of points using **u** as our parameter t. After that, we have a group of points that we then use to rerun the algorithm with parameter **v**. In my own implementation, I ran the 1D evaluation algorithm on all of the control points and stored the result in a vector. I then reran the algorithm on the storage vector using parameter v.

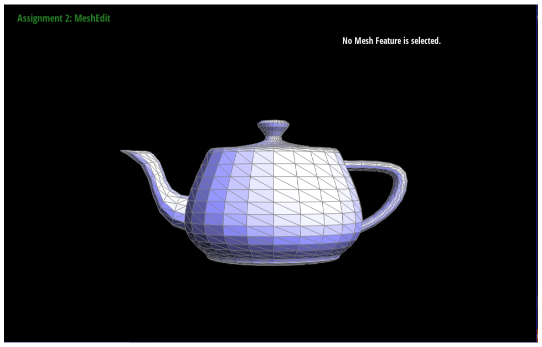


Teapot (.dae because .bez is freezes).

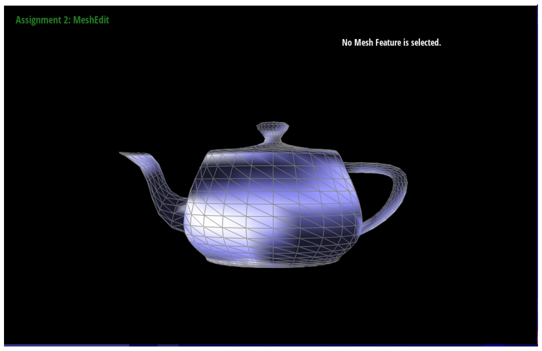
Section II: Sampling

Part 3: Average normals for half-edge meshes

Given a vertex, I look at all th halfegdes connected to the vertex in adjacent pairs, take the cross product, and sum them all up and return the normalized sum.



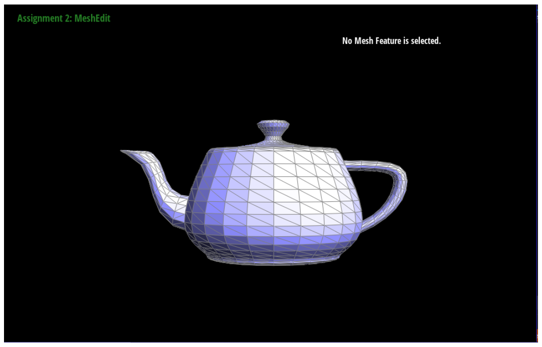
Teapot with smoothed normals.



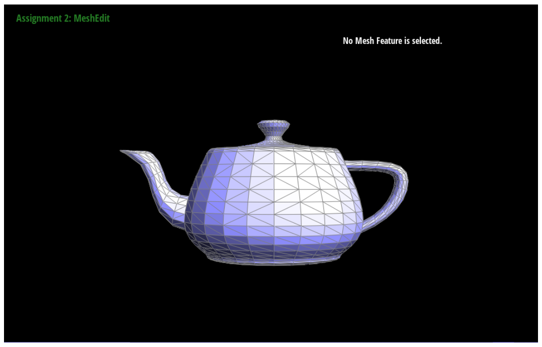
Teapot without smoothed normals.

Part 4: Half-edge flip

Given an EdgeIter, I set variables to all relevant halfedges and vertices in the local, relevant halfedge mesh. I drew a picture and for every reassignment I made in the picture, I made it in my code.



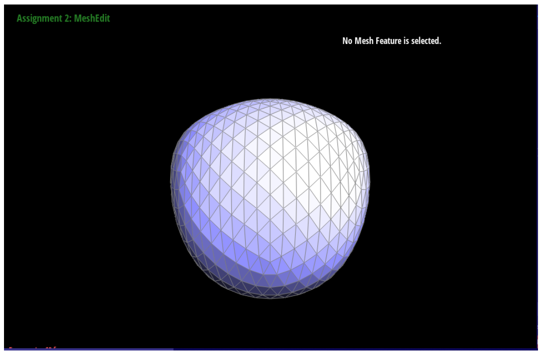
Before Flip.



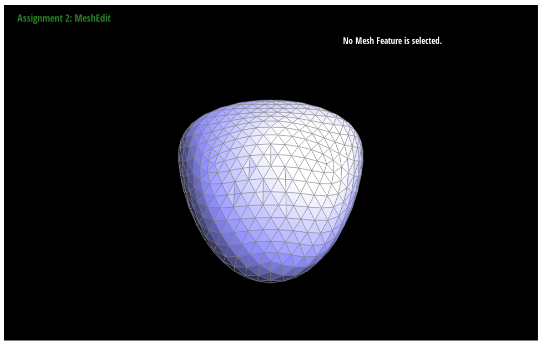
After Flip.

Debugging was tricky, mostly because I spent a lot of time not knowing what a (insertComponent)Iter was for most of the project, but as soon as I figured it out, pointer reassignment was fine. I had a really hard time with pointers in CS61A, but over the summer I finally started to understand them and I make it a point to draw pictures when dealing with them.

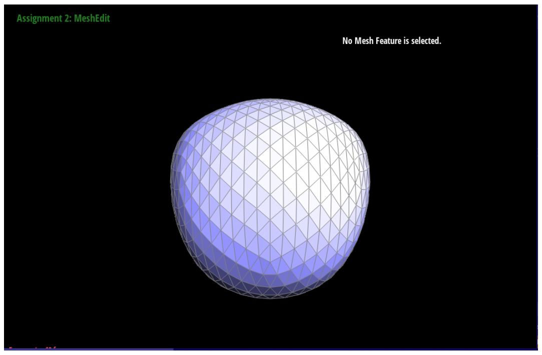
Part 5: Half-edge split



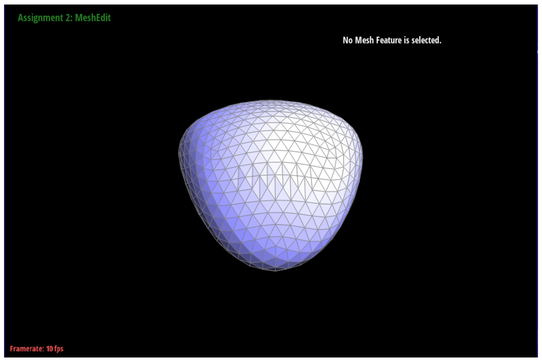
Before Edge Split.



After Edge Split.



Before Flip and Split.



After Flip and Split.

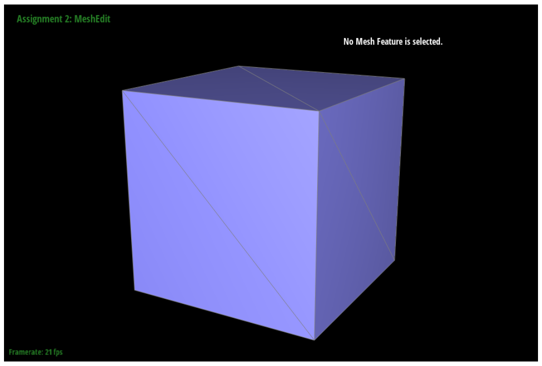
This was pretty quick and there were no real issues in debugging. It was one of the fastest parts of the project. I drew the halfedge mesh, labeled everything, drew what was supposed to happen when it was finished, and reassigned pointers accordingly.

Part 6: Loop subdivision for mesh upsampling

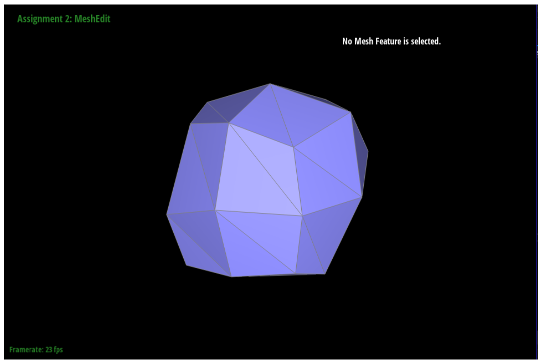
I marked all of the old vertices and edges for later use, recomputed the new position of the old vertices, then split every edge. Afterwards, I traversed every edge and checked to see if it was connecting a new and old vertex. If so, I flipped it. Otherwise, I skipped over it. Afterwards, I reassigned the position of old vertices to their new positions.

In Loop subdivision, the mesh often gets smaller and sharp edges get dull and sink into the more accurate geometry of the mesh. In some cases, the sharp edges put holes in the mesh and completely destroy the geometry of the entire figure.

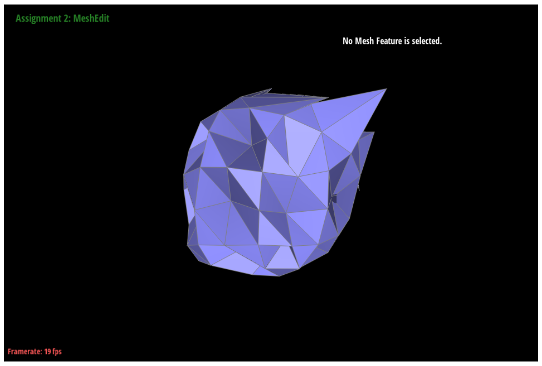
To alleviate the effects, it helps to do a series of splits so that there are more triangles. It is noticeable in most meshes when you have too few triangles and you attempt to upsample. The more triangles there are, the more accurate the upsampling process becomes. However, the sharp edges still show a few issues that may be fixable by not updating the vertices at the edges.



Normal.



With Upsampling (1).



With Upsampling (2).

I had some issues with getting the upsample completely right. I still suspect that the issue is in the Split operation, but I (and several of TAs) were not able to find it. Still looks pretty, though!

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!