# CS 184: Computer Graphics and Imaging, Spring 2023

# Project 2: Mesh Edit

Matthew Sun

## Overview

This assignment tasked me with making an implementation of the de Casteljau algorithm for bezier curves and surfaces as well as various basic mesh operation for a mesh editor using a specified data structure. A proper implementation of these features is vital to the processing of geometry. The project was a bit tough, as doing the edge flips and splits on paper was difficult, but overall, the project was very insightful and allowed me to see the process in which programs soften geometry.

## Section I: Bezier Curves and Surfaces

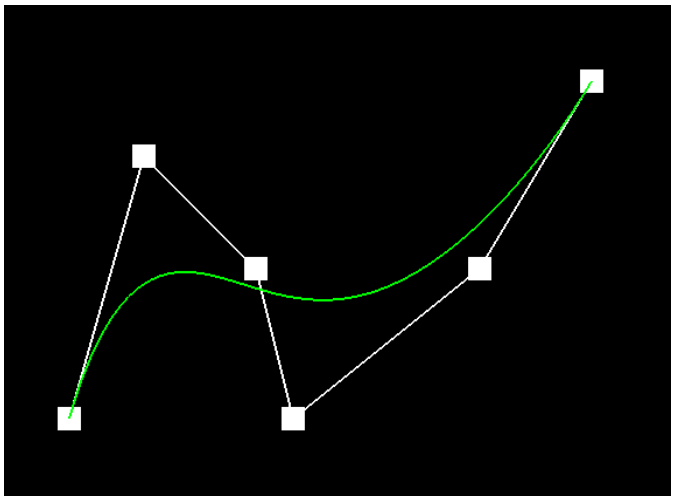
### Part 1: Bezier Curves with 1D de Casteljau Subdivision

**Briefly explain de Casteljau's algorithm and how you implemented it in order to evaluate Bezier curves.**

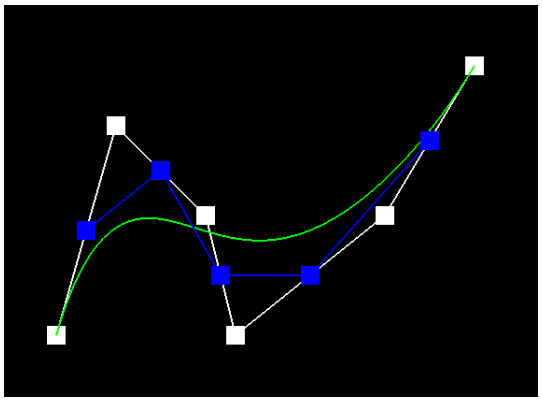
The de Casteljau algorithm is an algorithm that computes the position of a point on a Bezier curve that relies depends on the parameter $t$. Given a list of control points, the algorithm defines points on a Bezier curve by recursively performing linear interpolations using the previously mentioned parameter $t$. The implementation is simple as follows: If the list of points is of length $1$, simply return the list. Otherwise, perform linear interpolation on each pair of points using the parameter $t$ with the equation $(1-t)p_i + tp_{i+1}$ and return a vector of all the results.

**Take a look at the provided .bzc files and create your own Bezier curve with 6 control points of your choosing. Use this Bezier curve for your screenshots below.**

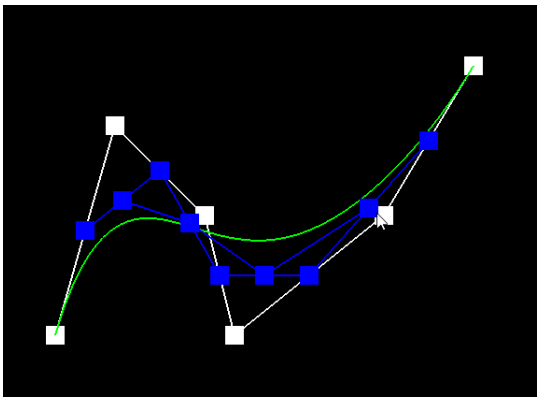
The following images are a new Bezier curve and the result after the poitns are passed through the implemented de Casteljau algorithm.



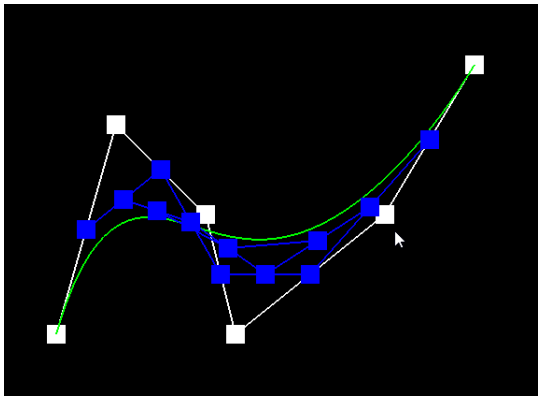
**Show screenshots of each step / level of the evaluation from the original control points down to the final evaluated point. Press `e` to step through. Toggle `c` to show the completed Bezier curve as well.**



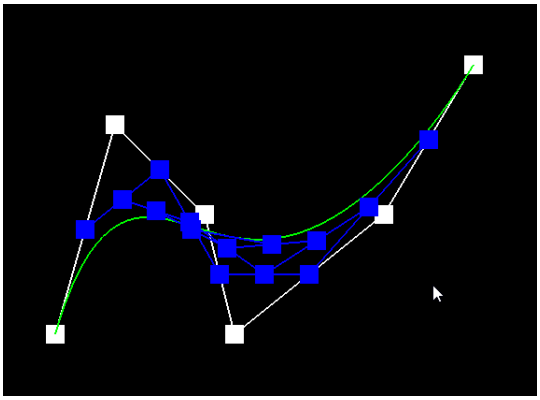
Level 0



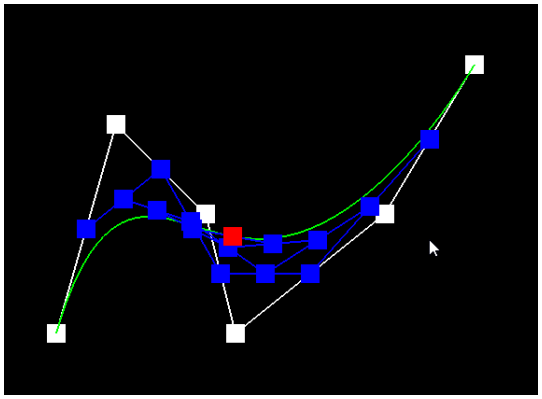
Level 1



Level 2

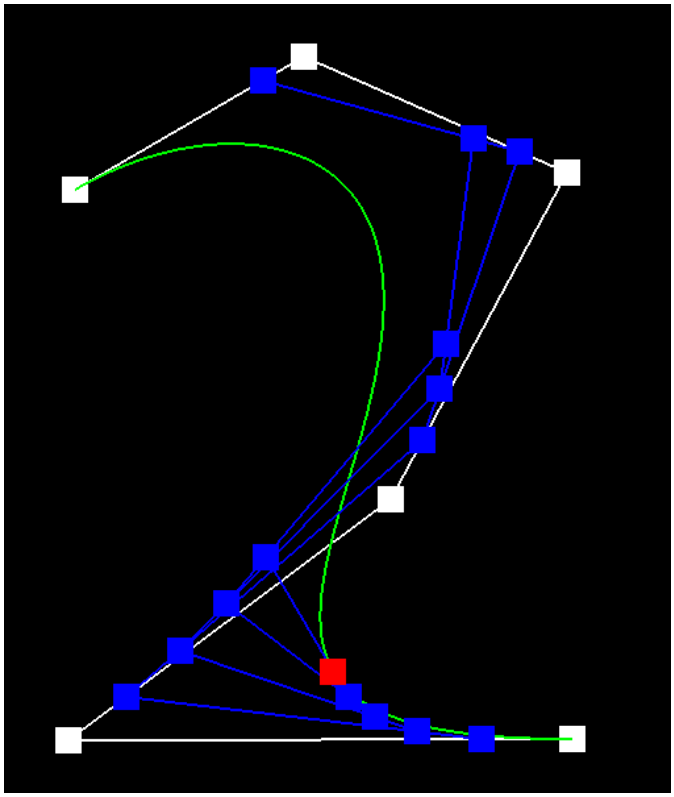


Level 3



Level 4

Show a screenshot of a slightly different Bezier curve by moving the original control points around and modifying the parameter $t$ via mouse scrolling.



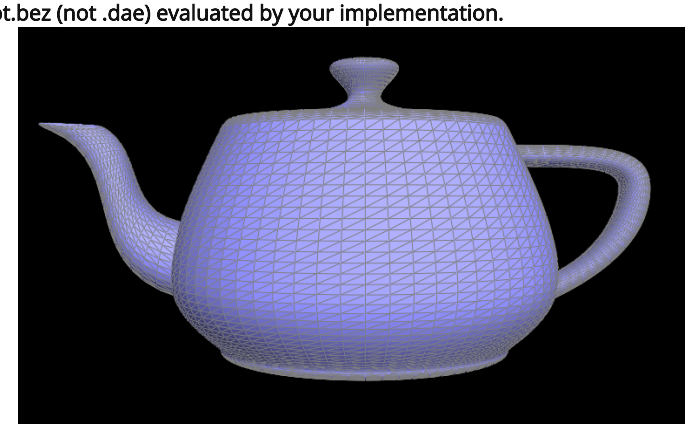
### Part 2: Bezier Surfaces with Separable 1D de Casteljau

**Briefly explain how de Casteljau algorithm extends to Bezier surfaces and how you implemented it in order to evaluate Bezier surfaces.**

The 1D de Casteljau algorithm extends rather straightforwardly to 2D surfaces. Given a collection of $m$ Bezier curves in 3D space, defined using $n$ control points and parametrized by a scalar $u$, a Bezier surface can be defined as a locus of points reached by the Bezier curves parametrized by a scalar $v$ that is defined through the control points for a constant $u$.

Using an $m \times n$ 2D vector of control points and the parametric coordinates $(u, v)$, first evaluate the 1D de Casteljau algorithm described in part 1 along the rows of the vector at $u$, resulting in a set of $m$ Bezier curves. Then, using these resulting $m$ points, evaluate the Bezier curves at $v$ using the de Casteljau algorithm again.

**Show a screenshot of bez/teapot.bez (not .dae) evaluated by your implementation.**



## Section II: Triangle Meshes and Half-Edge Data Structure

### Part 3: Area-Weighted Vertex Normals

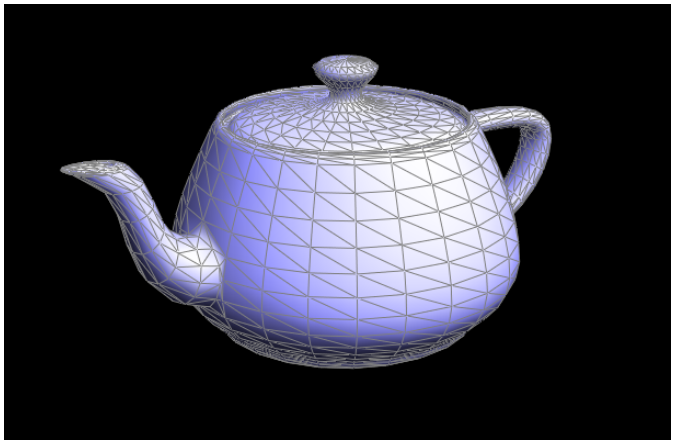
**Briefly explain how you implemented the area-weighted vertex normals.**

The implementation I went with is as follows:

1. Declare and initialize a normal vector and a `HalfedgeCIter` iterator.

2. Given a vertex, traverse the faces of the mesh that surround it, access the normals, and weigh them based on the area of that face.
    a. This is done by retrieving the `halfedge` associated with the vertex, reach every outgoing `halfedge`, make a call to `twin()->next()`.
    b. The area of the face is found using the fact that the magnitude of the vector produced by the cross product between two vectors is equal to the parallelogram that they determine ($|u \times v| = |u||v| \sin\theta$). Since the faces are triangular, simply halving this cross product results in the correct area, and hence the correct weight.
    c. The normal is generated by creating a weighted sum of the normals of each face.

**Show screenshots of dae/teapot.dae (not .bez) comparing teapot shading with and without vertex normals. Use `q` to toggle default flat shading and Phong shading.**

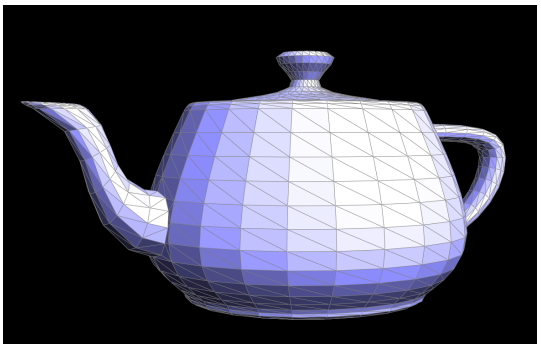


## Part 4: Edge Flip

**Briefly explain how you implemented the edge flip operation and describe any interesting implementation / debugging tricks you have used.**

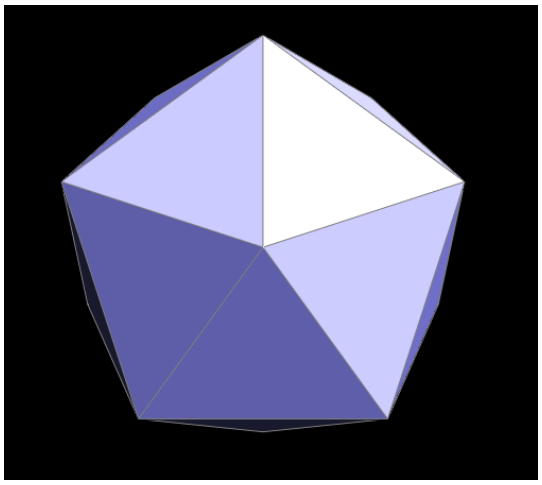
I conceptualized the pointer reassignment that needed to be done by performing the operation on paper. This took... quite a while, but I had a good grasp of what the pointers needed to be reassigned to. Performing these reassignments were as simple as calling `setNeighbors` on the respective `Halfedge`s and assigning the `Halfedge` pointers to the correct `Halfedge`.

**Show screenshots of the teapot before and after some edge flips.**

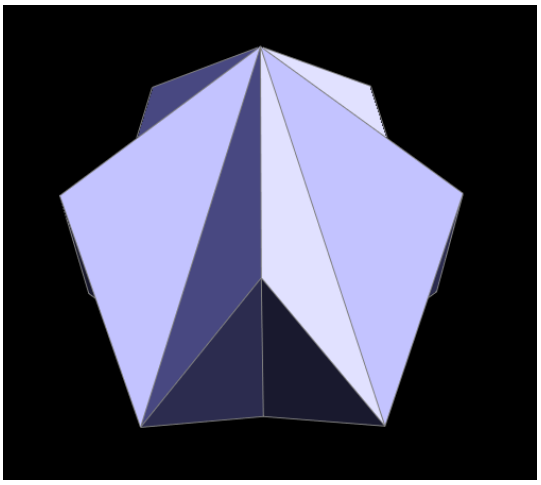


A normal metal teapot.

I may have dented it a little...



A normal d20.



A not-so-normal d20.

**Write about your eventful debugging journey, if you have experienced one.**

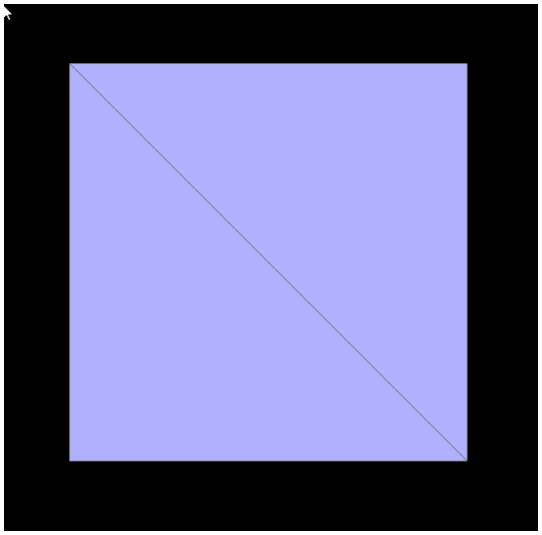
Thankfully, I have not had too much debugging. Because I did everything on paper first, I did not actually encounter any bugs and was lucky to be successful first try.

## Part 5: Edge Split

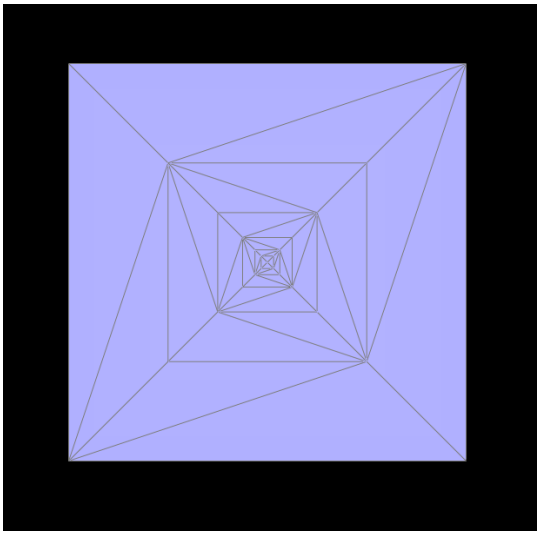
**Briefly explain how you implemented the edge split operation and describe any interesting implementation / debugging tricks you have used.**

My implementation of edge-splitting was similar, in that I performed the operation on paper. However, this took a significantly longer time, as it required the additions of a new vertex, 3 new edges, and 6 new `Halfedge`s. After reassigning the pointers properly, I also gave an update to the new vertex's position.

**Show screenshots of a mesh before and after some edge splits.**

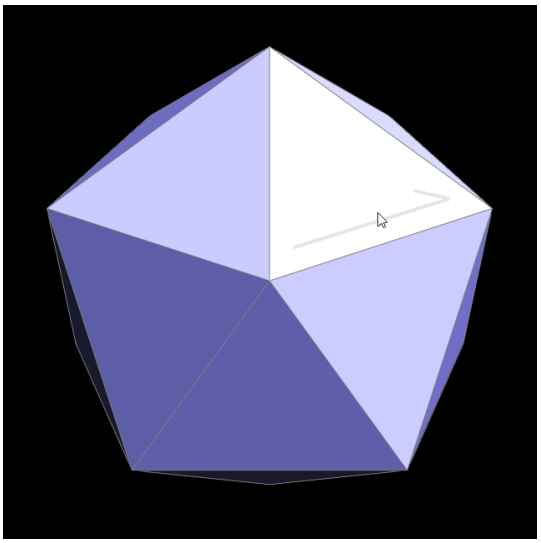


A cube face.

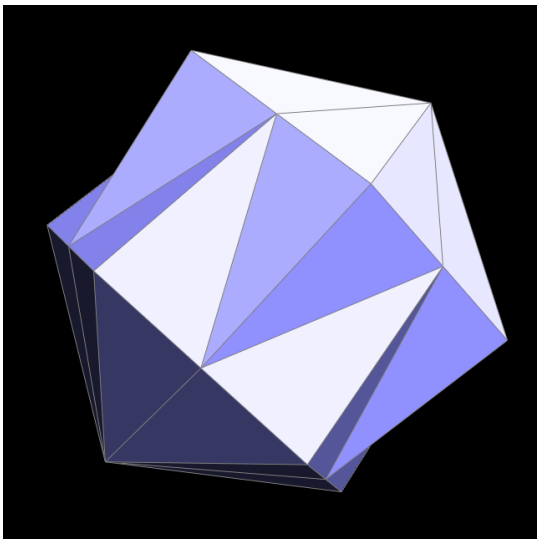


A cube face with a fractal on it.

**Show screenshots of a mesh before and after a combination of both edge splits and edge flips.**



A d20. (Mouse cameo?)



Not sure what this is, but it's pretty.

**Write about your eventful debugging journey, if you have experienced one.**

Thankfully, this function was also a lucky first try. However, this is only within the vacuum of this problem, and unfortunately, a very important detail caused some issues with loop subdivision.

## Part 6: Loop Subdivision for Mesh Upsampling

**Briefly explain how you implemented the loop subdivision and describe any interesting implementation / debugging tricks you have used.**

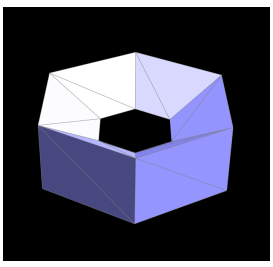
I followed the recommendations in the spec as best I could. I preprocessed the weighted vertes location for all vertices in the target mesh. The old vertices' `newPosition` parameter was directly set, while the positions of the new vertices are set into the `newPosition` parameter of the old edges instead for accessibility post-splits.

I went back to part 5 and tried to perform a loop subdivision manually to grasp the procedure and determine which edges needed to be flipped. I looped through each old edge and split them (the `if`-statement prevents infinite loops by checking for the `isNew` parameters of the vertices). Then, I looped through each edge and checked if exactly one of the vertices was a new vertex and flipped it if it was.

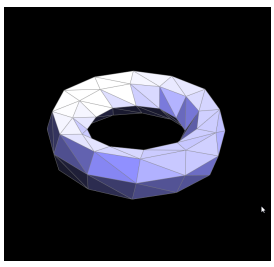
As far as debugging goes, an oversight in part 5 had caused some severe issues in this part. I had failed to reassign the `isNew` parameter. Originally, I thought the error had resided in my paper-and-pencil model, so I redid it twice over to make sure I didn't make any mistakes there. Once I was certain, I checked over the code again to make sure I hadn't typed anything wrong. From then on out, I filtered through all the files once again to make sure I wasn't missing something important. Lo and behold, I had forgotten to assign the `isNew` parameter properly. I breathed a large sigh of relief when that was done.

**Take some notes, as well as some screenshots, of your observations on how meshes behave after loop subdivision. What happens to sharp corners and edges? Can you reduce this effect by pre-splitting some edges?**

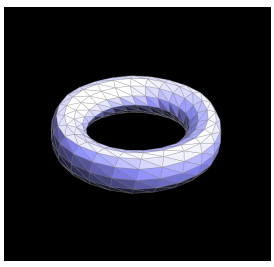
Sharp edges "melt," for lack of a better term. They sink down and become rounder and rounder. However, even at 256x supersampling (my poor laptop graphics card couldn't handle any more than that), the approximate locations of the original sharp edges can still be seen; it just makes them really rounded.



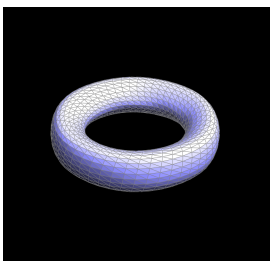
Torus, 1x upsample.



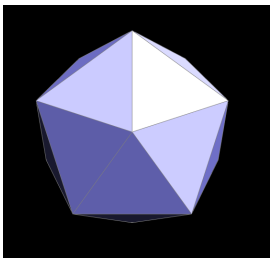
Torus, 4x upsample. (Mouse cameo!)

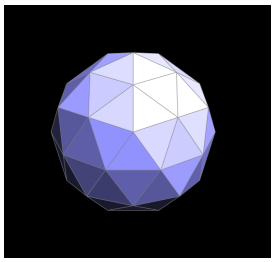


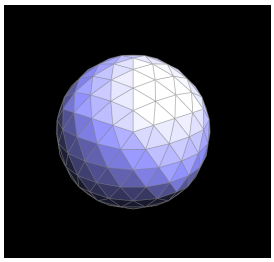
Torus, 16x upsample.

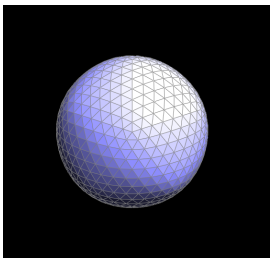


Torus, 64x upsample


Icosahedron, 1x upsample.


Icosahedron, 4x upsample.

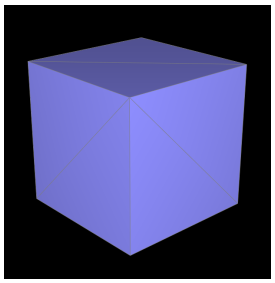

Icosahedron, 16x upsample.

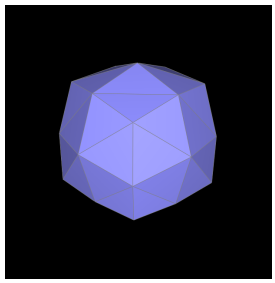

Icosahedron, 64x upsample.

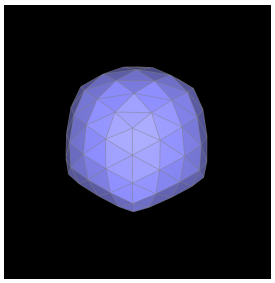
**Load dae/cube.dae. Perform several iterations of loop subdivision on the cube. Notice that the cube becomes slightly asymmetric after repeated subdivisions. Can you pre-process the cube with edge flips and splits so that the cube subdivides symmetrically? Document these effects and explain why they occur. Also explain how your pre-processing helps alleviate the effects.**

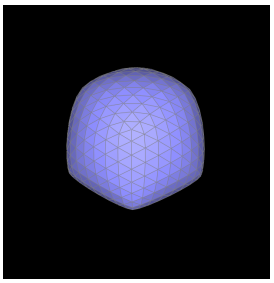
The asymmetry comes from the fact that the original cube has faces divided in a way that cannot be kept symmetrical over all faces. Since the topology of the asymmetrical cube has much less sample, with each face being made of just 2 triangles, the upsampling results in a division along the diagonal edge.

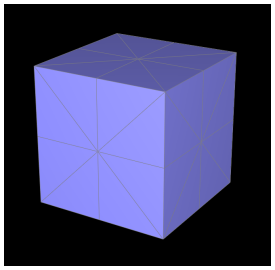
Creating a symmetrical upsampling is as simple as splitting the edge on each face of the cube. The results are as follows:

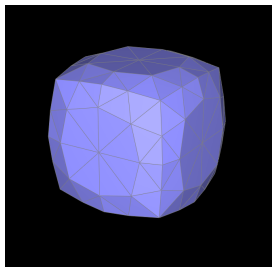

Asymmetrical cube, 1x upsample.

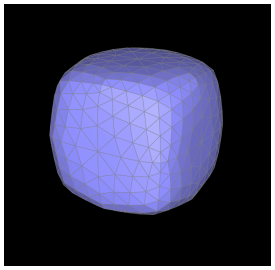

Asymmetrical cube, 4x upsample. (Mouse cameo!)

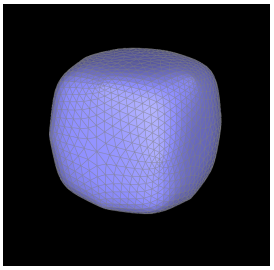

Asymmetrical cube, 16x upsample.


Asymmetrical cube, 64x upsample


Symmetrical cube, 1x upsample.


Symmetrical cube, 4x upsample.


Symmetrical cube, 16x upsample.


Symmetrical cube, 64x upsample.

**Website**

Follow this link to reach the gitpage hosting this project write-up.