Project 2: Mesh edit

Note: Please try to see the images in my web-page repo. My website is not loading images and I even tried to convert straight from Markdown to png, but it won't work.

Section I: Bezier Curves and Surfaces

Part 1: Bezier Curves with 1D de Casteljau Subdivision

de Casteljau's algorithm is a recursive method that compute the polynomials in the Bezier curves. For iteration level:

- For each Bezier curve `i`: split it into 2 segments by inserting a new point `b_i`, such that the 2 segments are proportional to the original line by parameter `t` and `1-t`.
- Connect all the added points from curve `i` to `i+1`. (don't connect the first and last curves' point, just all other intermediates)
- Repeat until only have 1 curve left and split it with 1 final point `b_0`.

In the total curves, the first, last and `b_0` point will make up the final linearly interped curve. Shown here as example image from lecture: de Casteljau algo

Screenshots of each step / level of the evaluation from the original control points down to the final evaluated point: B1 B2 B3 B4 B5 B6

Screenshot of a slightly different Bezier curve by moving the original control points around and modifying the parameter `t` via mouse scrolling:

Same final curve with much lower `t` value: Lower t value

Final curve with 1 control point moved to the far right, which shifts the final curve: Shift 1 control point

Part 2: Bezier Surfaces with Separable 1D de Casteljau

de Casteljau algorithm can extend to Bezier surfaces. A 3D surface is defined by 2 vectors `(u,v)`, while 2D line just uses `t`. A Bezier surface has 3D control points. We can think of the Bezier surface as infinitely many Bezier curves one next to another. First, we interpolate `u` on all of those Bezier curves. After getting the final point of those curves, we can use `v` to get the whole Bezier surface.

Screenshot of `bez/teapot.bez`: Teapot 1 Teapot 2

Section II: Triangle Meshes and Half-Edge Data Structure

Part 3: Area-Weighted Vertex Normals

My implementation of normal:

- Start at the half-edge of some first vertex
- Find that half-edge's origin vertex and vertex that it points towards. (Name it `u,v`)
- Take the cross product of those 2 vectors.

- Add it to the total `output` vector
- Move to its twin's next half-edge (we are looking at a new face now)
- Repeat above steps, until we examine all faces of the mesh and get back to the original half-edge.
- Finally, return normalized vector Teapot Shade

Part 4: Edge Flip

I follow the exact instruction:

- List out all the vertices via all the half-edges
- Draw a remesh/reassign the old list's elements with the new list of elements
- Do the half-edge reassignment via `Halfedge::setNeighbors(...)` and other methods listed in step 4 of the instruction. Also set all points, even though they don't change.

For debugging, I made a typo in the edge name, `bc` but instead I typed `cb` 😦 . So the program runs through the wrong face and mess up the image a bit. I had to fix that.

Before Flip After Flip

Part 5: Edge Split

Same as part 4, I draw 2 diagrams of the 2 triangles given. Add a new vertex to the midpoint of the original edge. Assign/re-assign so many elements to finally get 8 half-edges after adding the middle vertex.

Screenshots of a mesh before and after some edge splits: Before Split After Split

Screenshots of a mesh before and after a combination of both edge splits and edge flips: Before Combo After Combo

Again, one of my fatal flaw was messing up the names of the half-edges I assigned in my code. I name the half-edges according to their start-end direction (i.e. `bc` or `cb`), and got mixed up some names in my own naming convention. So debugging was painful, even though it was a typo 😦

Part 6: Loop Subdivision for Mesh Upsampling

Loop subdivision is a form of upsampling. It will divide each shape/triangle of the object into severall smaller shapes/triangles.