Writeup

Overview

The project I implemented aim to linear interpolate Bezier Curves and Bezier Surface by control points. Also, I learn how to flip and split edges in order to upsampling meshes and make the meshes smoother and render better with shadows. The project gives me a more deep understanding of the engineering logic behind graphis.

Part1 Bezier Curves with 1D de Casteljau Subdivision

De Casteljau’s algorithm is a recursive method to evaluate linear interpolations in Bezier Curve. Bezier curve is defined by N+1 control points. There is a single parameter t, ranging between 0 and 1. Finding N intermediate control points in N segments using linear interpolations based on t, and then I find N-1 intermediate control points in N-1 segments in the next level. I repeat the steps recursively until only 1 intermediate control point can be found. One smooth curve is generated by these control points of different levels.

Part2 Bezier Surfaces with Separable 1D de Casteljau

In order to evaluate the points lies on the Bezier Surface, I evaluate the Bezier Surface with De Casteljau’s algorithm firstly which is similar to what I did in part1.

The control points of Bezier Surface is a M x N grid, each row of the control points in the grid generates a Bezier Surface based on the parameter u. After that, I select the point on each Bezier Surface at the same v coordinate by BezierPatch::evaluate1D() function which calls BezierPatch::evaluateStep() recursively and get the final single control point. I use De Casteljau’s algorithm to evaluate point v on the moving curve generated by these points.

Part3 Area-Weighted Vertex Normals

In this part, I compute the vertex normal vectors of the mesh in order to provides better shading for the surface smooth.

Firstly, I initialize a vector to store the sum of normal.

Secondly, I select a half-edge connected with the vertex and traverse to the twin edge of this half-edge by using the twin() pointer. I compute the area-weighted normal vector of the face connected with the twin edge and add it to the initialized vector. By looping this process, I am able to iterate all the normal vectors of the faces and add them together. Finally, I normalize the result.

Part4 Edge Flip

Firstly, I list all the existing pointers of half-edges, vertices, edges and faces of the meshes.

Secondly, I check whether the half-edges are on the boundary, if yes, I return these edges immediately.

Thirdly, I reset all the half-edge pointers to the updated edges, vertices and faces after remeshing with Halfedge::setNeighbors() function and also update the half-edges of vertices, edges and faces if they are changed.

I finish part4 very successfully without debugging.

Part5 Edge Split

Firstly, similar to what I did in Part 4, I list all the existing pointers and check whether half-edges are on the boundary.

Secondly, since after splitting the meshes, there are new half-edges, edges, vertices and faces and they are initialized.

Thirdly, I reset the pointers with Halfedge::setNeighbors() function as I did in the part 4.

The main bug I have is that although I set the half-edge of the new vertex to the new half-edge, I don’t set the position of it. Therefore, when I am going to split the edge, the face will disappear.

Part 6 Loop Subdivision for Mesh Upsampling

Loop Subdivision provides a data structure to smooth the sharp meshes.

Firstly, I calculate the position of the new vertex following 3/8 \* (A + B) + 1/8 \* (C + D) and update the position of the old vertex following (1 - n \* u) \* original\_position + u \* original\_neighbor\_position\_sum. I set bool isNew of all existing edges and vertices to be false.

Secondly, I split all the old edges using HalfedgeMesh::splitEdge(). The way I determine whether the edges are old or the newly subdivided is whether two of their vertices both are old. Then, I set the newPosition to the new vertices after splitting and also set bool isNew of these vertices to be new.

Thirdly, I flip the edges that are newly subdivided and one of their two connected vertices are new and one are old using HalfedgeMesh::flipEdge(). I also set bool isNew of these edges to be false in order for the next step.

Finally, I update newPosition of all the newly subdivided vertices.

I have main three bugs when I am implementing the function:

1. I didn’t determine whether the edges were old or newly subdivided before I split the edges, causing it runs infinitely.
2. I didn’t set all the variables to double or float.
3. I didn’t set bool isNew of newly subdivided edges to be true, thus, although when I was going to flip the edges, there weren’t any new edge and nothing was input into HalfedgeMesh::flipEdge(). Therefore, when I run the program, only lines were shown.