CSC4140 Assignment 5

Computer Graphics

April 5, 2022

Geometry

This assignment is 9% of the total mark.

Student ID: 120090266

Student Name: Feng Yutong

This assignment represents my own work in accordance with University regulations.

Signature: Feng Yutong

1 Overview

This project covers several aspects of geometric topcis. In part 1, I implemented Bezier Curves with 1D de Casteljau subdivision, then I extend it into Bezier surfaces in 3D. In part 2, I caculated the area-weighted vertex normals for phong shading, implemented the mainipulation of half-edge meshes, such as split and filp. Then I use loop subdivision to upsample the mesh. I learn to express an object implicitly and explicitly with finite given information.

2 Implement and Results

2.1 Task 1: Bezier curves with 1D de Casteljau subdivision

De Casteljaus algorithm, consists of repeated bilnear and possibly subsequent repeated linear interpolation, is a method to calculate Bezier curves recursively. Process of 1D de Casteljau subdivision:

1. Insert a intermediate point using linear inter polation ($p_i^{'} = lerp(p_i, p_{i+1}, t) = (1-t)p_i, +tp_{i+1}$) on every edge separated by control points.

2. Repeat step 1 on intermediate points until only one intermediate point is inserted.

3. Run step 1 and 2 for every t in [0,1].

BezierCurve:: evaluateStep (...) implements step 1. The different levels of evaluation from the orignal control points down to the final evaluated point with 6 control points (mycurve.bzc) are shown in Figure 1.

A screenshot of a slightly different Bezier curve by moving the original control points around and modifying the parameter t via mouse scrolling is shown in Figure 2.

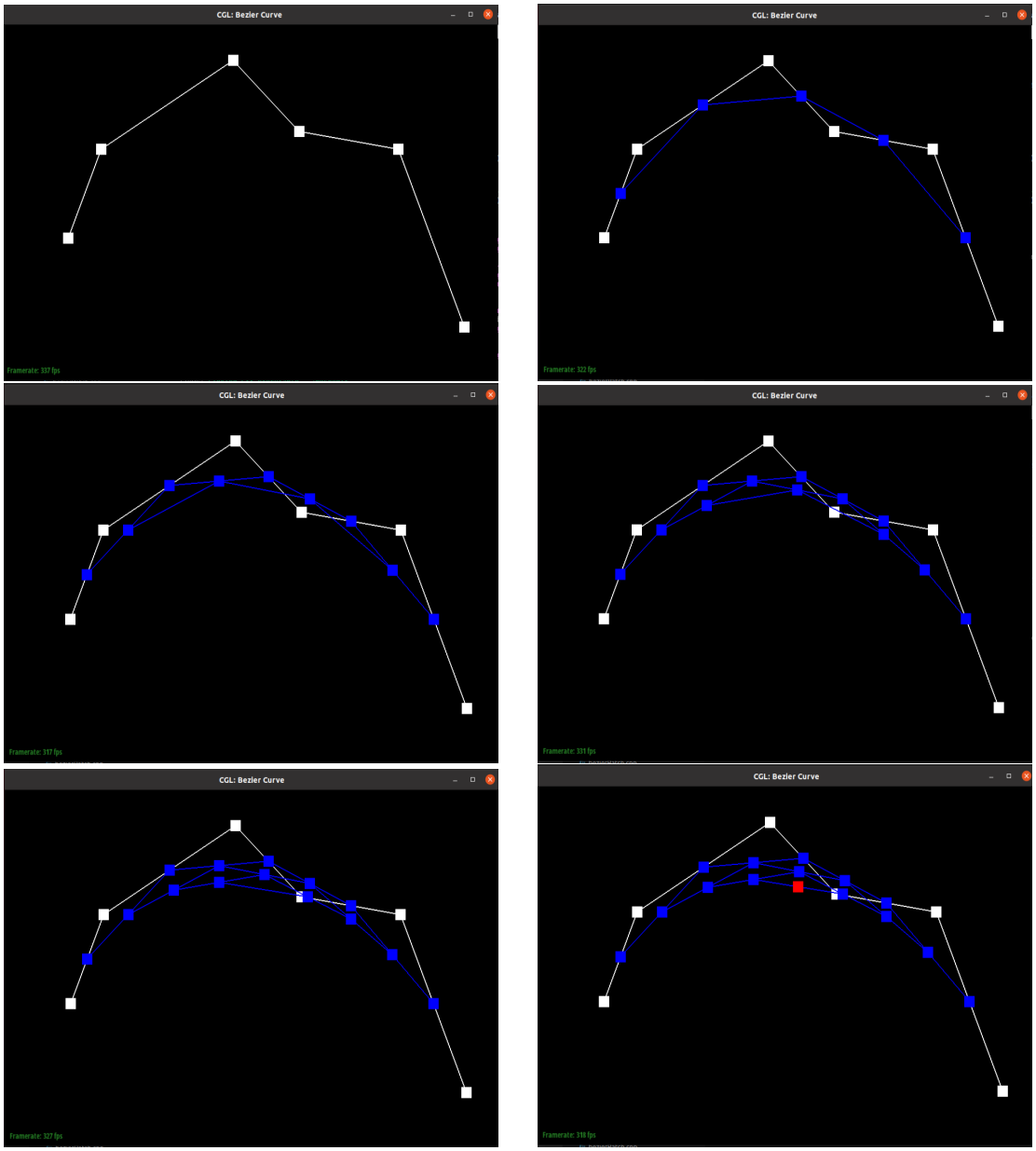

Figure 1: Different levels of evaluation by de Casteljau

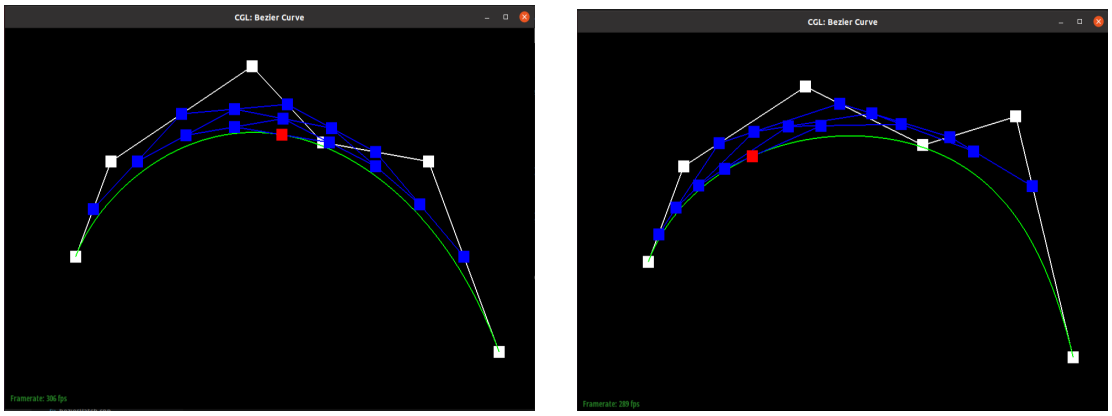

Figure 2: Final evaluation of two slightly different Bezier curves with different parameter t

2.2 Task 2: Bezier Surfaces with Separable 1D de Casteljau

Extending de Casteljau on surface is like applying this algorithm on two dimensions. We can apply it separately on the given parameter (u,v) to evaluate surface points. Process:

1. Use de Casteljau to calculate point u on each of the n Bezier curves in u. This gives n new control points.

2. Use de Casteljau to calculate point v with n new control points.

The result is shown in Figure 3.

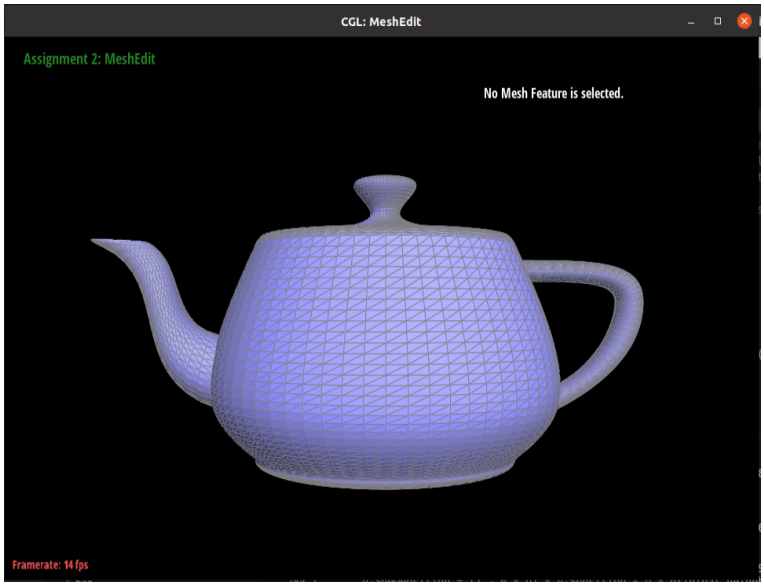

Figure 3: Bezier surface

2.3 Task 3: Area-Weighted Vertex Normals

The Bezier surface is difficult to control and render since changing the postition of one control point affects the entire mesh. Thus, triangle meshes are used to represent 3D geometric models. To better iterate the neighbouring triangles, half-edge mesh, a data structure storing mesh entities and connectivity information, is introduced.

Since flat shading is not good for smooth surface, we improve it by replacing vertex normals with area-weighted vertex normals. Process:

1. Define h = halfedge().

2. Start a while loop that ends when h == halfedge(). Inside loop iteration: calulate triangle area by cross product of its two edges; add the product of triangle area and its corresponding normals to the area-weighted normal; traverse to next triangle by updating h=h->twin()->next().

3. Return the re-normalized normals by Norm.normalize().

Comparsion between flat-shading and phong shading are shown in Figure 4.

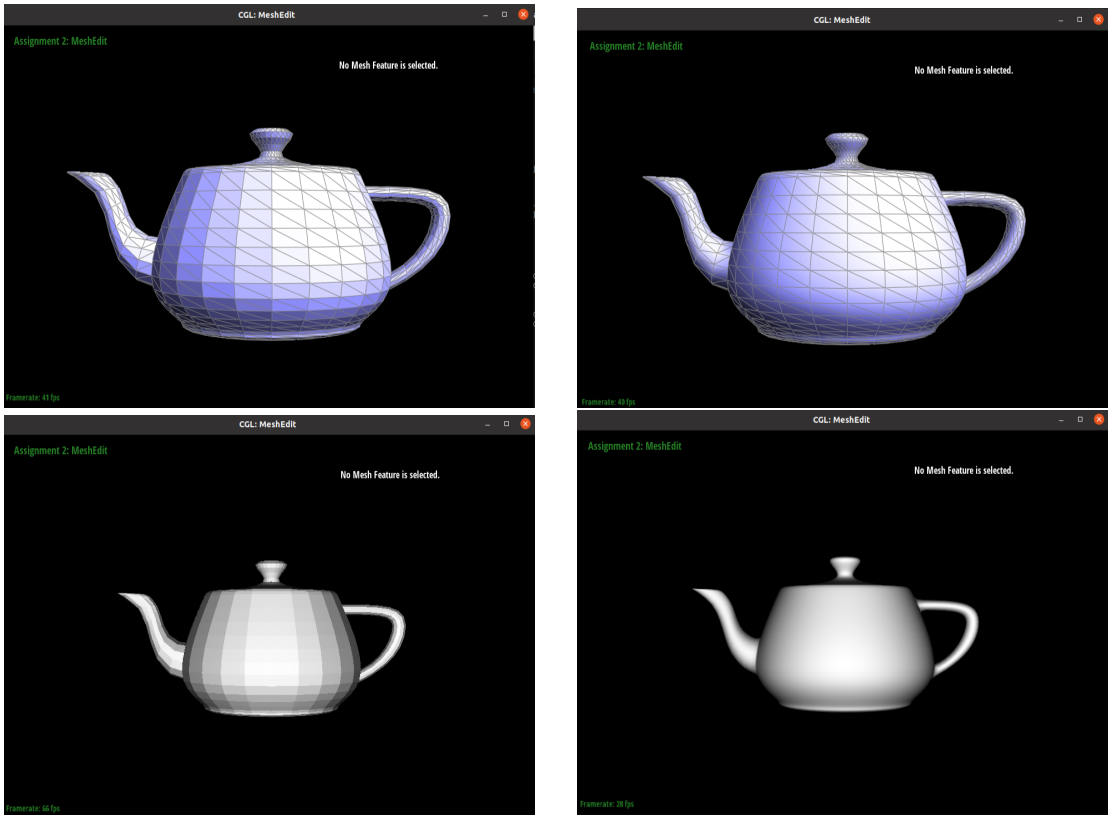

Figure 4: Comparsion between flat-shading and phong shading

2.4 Task 4: Edge flip

Given a pair of triangles (a,b,c) and (c,b,d), a flip operation on their shared edge (b,c) converts the original pair of triangles into a new pair (a,d,c) and (a,b,d), as shown in Figure 5.

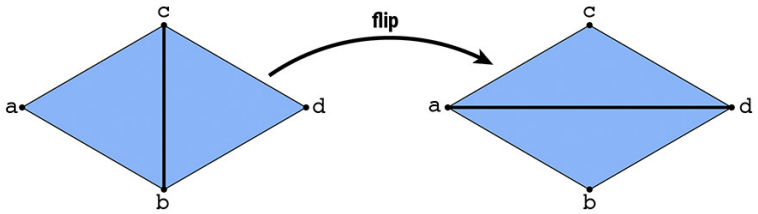

Figure 5: Demostration for edge flip

This operation is challenging in ensuring all pointers are in the right place and have correct connectivitiy. Process:

1. Draw a simple mesh before and after edge flip. Write down a list of all elements, i.e., half-edges, vertices, edges, and faces. Since I first failed this task, I referred to article provided in the guidance and reconstructed my picture. (see Figure 6).

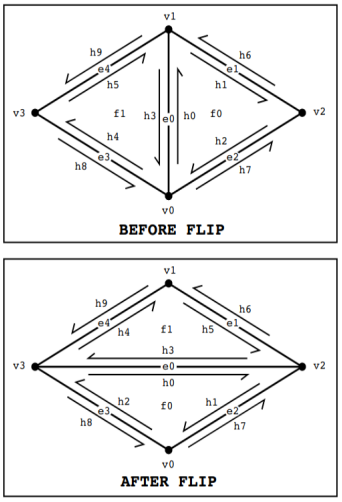

Figure 6: Draft for edge flip

2. If is a boundary edge, just return itself. Otherwise , set pointers for all elements in the mesh even if the element is not changed. This can be extremely important, since it is very easy to miss or repeat some pointers. Corretness is more important than effciency.

3. According the picture in step 1, reset the connectivity by Halfedge::setNeighbors(...).

Results are shown in Figure 7. Note the left part reach a degerate case.

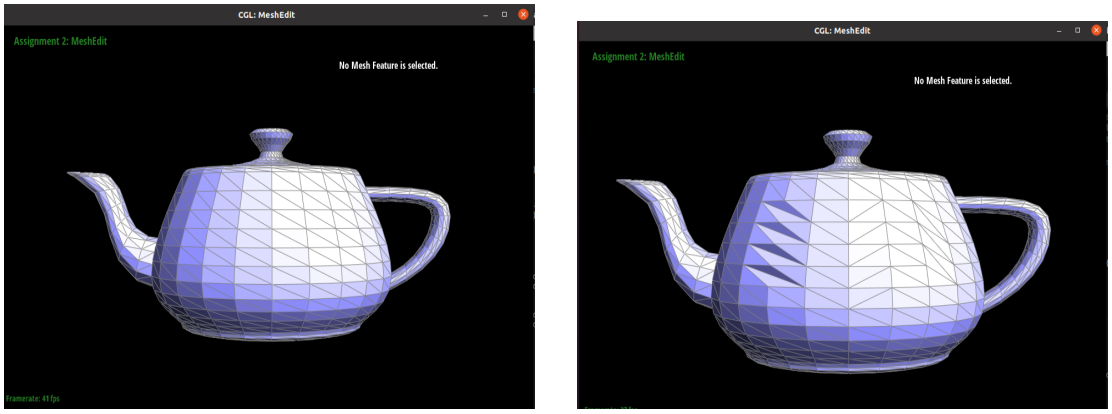

Figure 7: Comparsion between before and after edge filp

2.5 Task 5: Edge Split

Given a pair of triangles (a,b,c) and (c,b,d) , a split operation on their shared edge (b,c) inserts a new vertex mm at its midpoint and connects the new vertex to each opposing vertex a and d, yielding four triangles as shown in Figure 8.

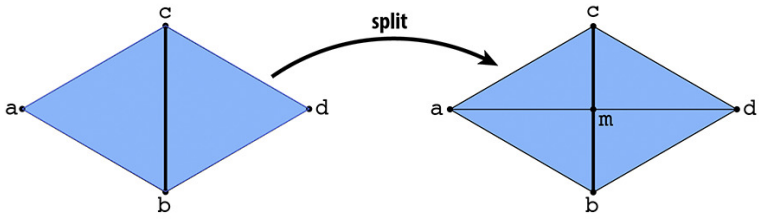

Figure 8: Demonstration for edge split

Process:

1. Draw a simple mesh before and after edge split. Write down a list of all elements, i.e., half-edges, vertices, edges, and faces.

2. If is a boundary edge, just its vertex. Otherwise, set pointers for all element in the mesh even if the element is not changed. Assign new face, new edge, and new vertex to the midpoint. Modify the property isNew = true for newly added elements.

3. According the picture in step 1, reset the connectivity of rest halfedge by Halfedge::setNeighbors(...).

Results are shown in Figure 9.

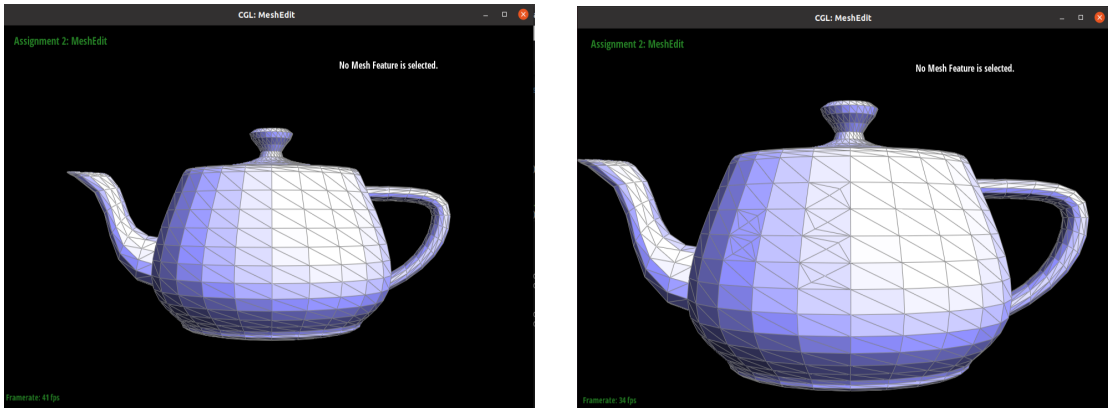

Figure 9: Comparsion between before and after edge split

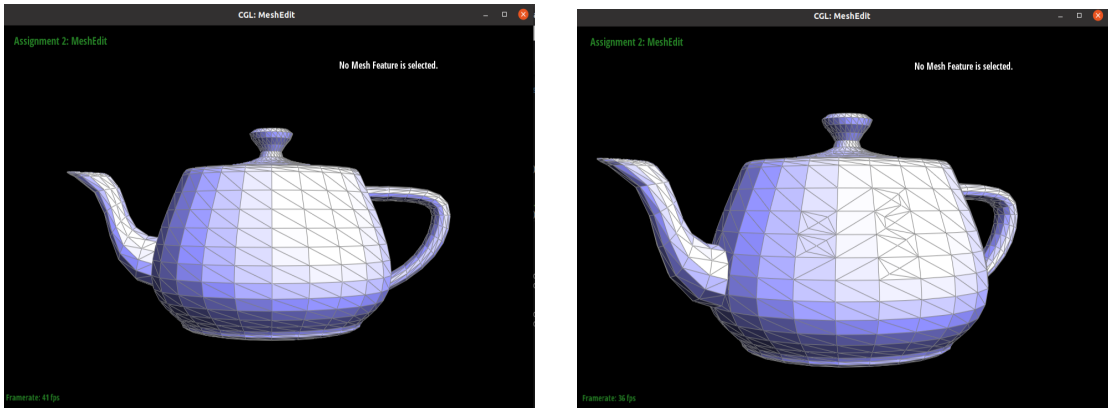

Figure 10: Comparsion between before and after a combination of edge flip and edge split

2.6 Task 6: Loop Subdivision for Mesh Upsampling

Sometimes we need to upsample a low-resolution polygon mesh for better display, simulation, and etc. Simply splitting each polygon into smaller piceces may result in blocky or chunky characteristics. As a result, we use Loop subdivision that nicely interpolates or approximates the original mesh data. Process:

1. Compute new positions for all vertices in the input mesh and store them in v->newPosition. The new position of an original vertex is

$$(1 - n \times u) \times v->postion + u \times orignal\_neighbor\_position\_sum$$

$$u = \frac{3}{16} \text{if vertex degree is 3,} u = \frac{3}{8n} \text{otherwise}$$

where n is vertex degree and orignal_neighbor_position_sum can be get by traversing neighbors. Set v->isNew = false to mark the vertex as being a vertex of the original mesh.

2. Compute the updated vertex positions associated with edges and store it in e->newPosition. The new position is

$$\frac{3}{8} \times (A + B) + \frac{1}{8} \times (C + D)$$

where A and B is the closer neighbour vertex positions, C and D is the farther ones.

3. For every edge, if it is not a new edge and it connects two new old vertices, split the edge and store its midpoint in v->newPosition.

4. For every edge, if it is a new edge and it connects an old and new vertex, flip the edge.

5. Update all vertex positions with newPosition.

At first, I only marked one of newly added edges (I pasted some similar pattern code and forgot to modify the name of the parameters). I compare the picture with only split operation and found the two picture looks almost the same (see Figure 11). Since I couldn't find any bug in edge flip, I located the bug in edge split and found it by counting the edge filpped.

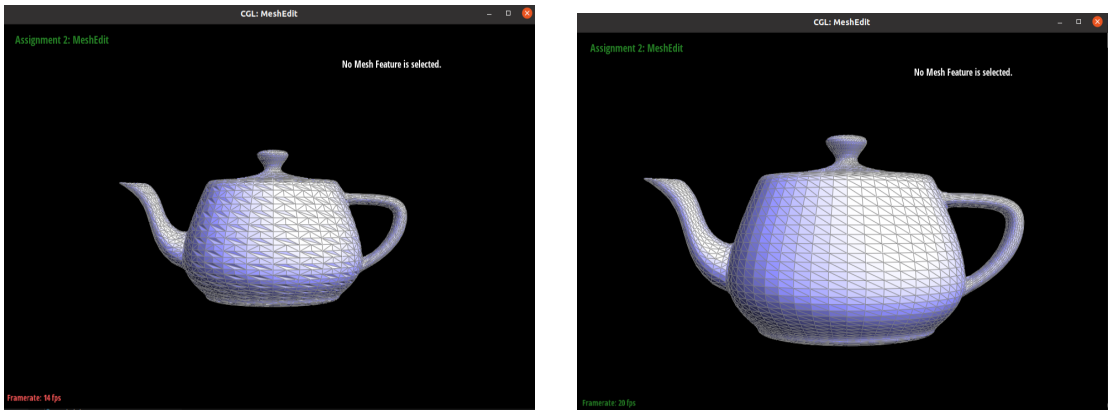

Figure 11: Comparsion between wrong(left) and correct(right) loop sudivsion

Obersvations Loop subdivision can smooth the curves (see Figure 12). Upsampling predicts details of an object on the condition that objects has a low frequency (in most cases). This results in less correctness when an object has high frequency or sharp corners or edges. Consequently, upsmapling can serve as a method to smooth objects.

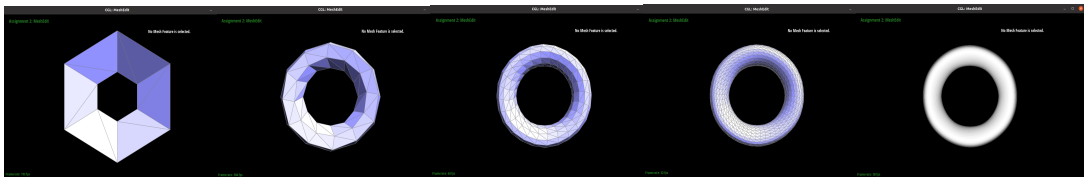

Figure 12: Loop subdivison for torus

To preserve the features, we can add more information on the mesh. One simple way is to pre-split the edges close to the corner to increase points around it (see Figure 13).

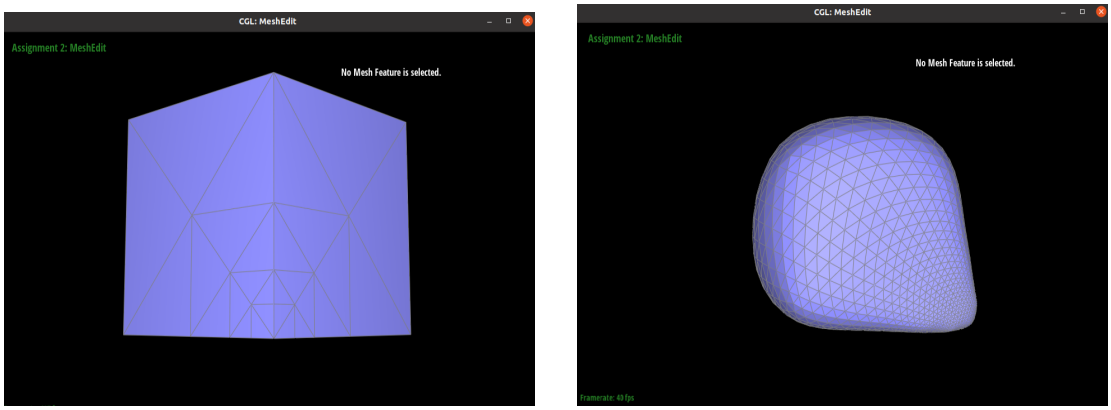

Figure 13: Orignal corner(upper left); Pre-split corner(bottom right)

Subdivision results in a asymmetirc mesh for asymmetric lined edges (see Figure 14). In order to get a symmetric mesh, we can pre-split the edges to make the mesh have a symmetric structure , i.e., splitting every diagonals (see Figure 15).

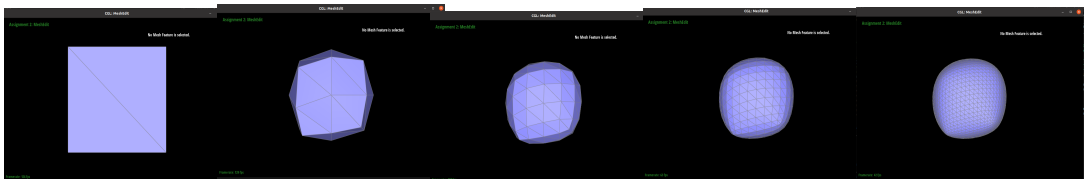

Figure 14: Loop subdivision of cube

Indication: more information on original mesh leads to more accurate modeling. Symmetric linked edge leads to symmetric objects. We can also use Figure 16 to show it.

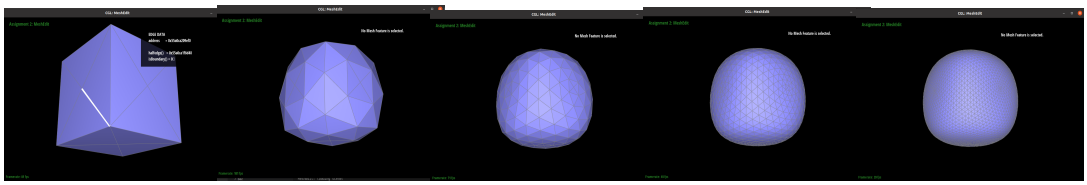

Figure 15: Loop subdivision of cube

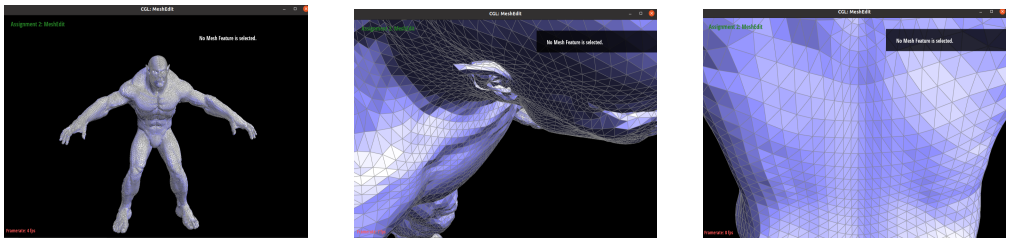

Figure 16: Orignal picture(left); High point density near sharp ear(middle); Symmetry(right)