CS 184 HW2 Report

Catherine Gai, Wenhan Sun

2022 Spring

Overview

In this project, we implement the following features:

- Section 1: Compute Bezier curves and surfaces.
  - Part 1: Compute Bezier curve using 1D de Casteljau's algorithm.
  - Part 2: Compute Bezier curve using 2D de Casteljau's algorithm, that is, performing 1D de Casteljau's algorithm for each row of the grid and performing another 1D de Casteljau's algorithm for the resulting column.
- Section 2: Implementing triangle mesh operations with halfedge data structure.
  - Part 3: Implement the area-weighted average normal of each vertex.
  - Part 4: Implement the edge flip operation, that is, given 4 original points A, B, C, D that makes up 2 triangles: ABC, BCD, the triangle are rearranged to be ABD, ACD.
  - Part 5: Implement the edge split operation, that is, given 4 original points A, B, C, D that makes up 2 triangles: ABC, BCD, create a new vertex M in the middle of B and C and reconstruct the local triangle mesh to ABM, BMD, DMC, CMA. Moreover, AM and CM are assigned to be new edges.
  - Part 6: Implement the upsampling of the triangle mesh using the following algorithm:
    1. Split all original edges in random order.
    2. Flip all new edges as defined in the edge split operation.
    3. Update all new vertices and old vertices according to a specific rule (see Part 6 in Section 2).

Section 1: Bezier Curves and Surfaces

Suppose V is a vector space, $n \in \mathbb{N}_+$. Suppose $\{m_i \in \mathbb{N} | i \in \mathbb{N}_+(n)\}$ and suppose a is an n-sequence in $\prod_{i=1}^{n} \mathbb{N}(m_i + 1) \to V$. Then the **$n$-dimensional Bezier structure** of degree

$\{m_i \in \mathbb{N} | i \in \mathbb{N}_+(n)\}$ with controls points a is defined as the function in $[0,1]^n \to V$:

$$\{u_i | i \in \mathbb{N}_+(n)\} \mapsto \sum_{\{k_i | i \in \mathbb{N}_+(n)\} \in \prod_{i=1}^{n} \mathbb{N}(m_i+1)} a(\{k_i | i \in \mathbb{N}_+(n)\}) \prod_{i=1}^{n} \binom{m_i}{k_i} u_i^{k_i} (1-u_i)^{m_i - k_i}$$

A 1-dimensional Bezier structure is defined as a **Bezier curve**. A 2-dimensional Bezier structure is defined as a **Bezier surface**.

Part 1: Bezier Curves with 1D de Casteljau's Subdivision

Suppose a is a 1-dimensional sequence, that is $n = 1$. Then the Bezier curve evaluated at $t \in [0,1]$ could be calculated by the 1-dimensional de Casteljau's algorithm:

input : The input control points a;
input : The evaluated point $t \in [0,1]$;
Initialize $b \leftarrow$ copy of a;
while *length of* $b > 1$ **do**
  $l_b \leftarrow$ length of b;
  $b \leftarrow \{(1-t)b_i + tb_{i+1} | i \in \mathbb{N}(l_b - 1)\}$;
end
output: The point b_0

The new curve is stored in `bzc/curve3.bzc`. The control points are given by Tab.1. The complete rendering of the curve is given by Fig.1. For each level, the rendering of the curve `bzc/curve3.bzc` is given by Fig.2. A modified curve based on `bzc/curve3.bzc` is given by Fig.3.

Index	Point in $\mathbb{R}^2$
0	(0.200, 0.350)
1	(0.250, 0.540)
2	(0.300, 0.600)
3	(0.800, 0.750)
4	(1.000, 0.650)
5	(0.900, 0.450)

Table 1: The control points of `bzc/curve3.bzc`

Part 2: Bezier Surfaces with Separable 1D de Casteljau's

Suppose a is a 2-dimensional sequence of size $(m_1+1) \times (m_2+1)$, that is $n = 2$. Then the Bezier surface evaluated at $(u,v) \in [0,1]^2$ could be calculated doing separate 1-dimensional de Casteljau's algorithm:

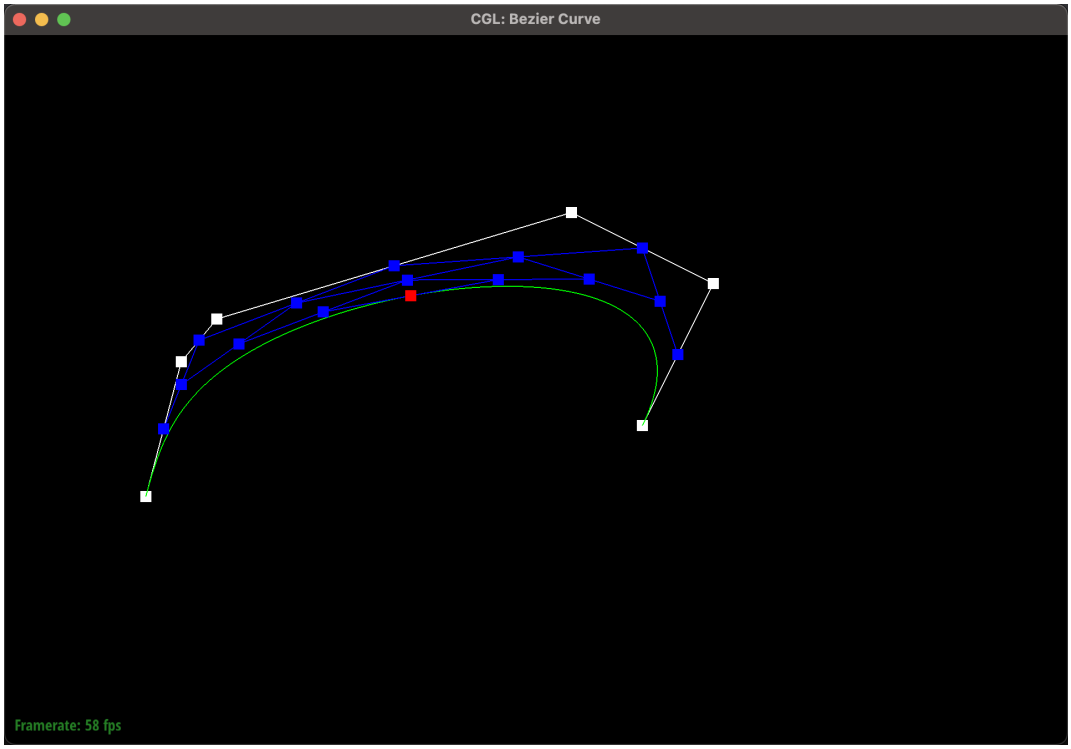


Figure 1: Complete rendering of the Bezier curve given by `bzc/curve3.bzc`

input : The input control points a;
input : The evaluated point $(u, v) \in [0, 1]^2$;
$b \leftarrow \{\texttt{de\_Casteljaus\_1d}(a_i, u) | i \in \mathbb{N}(m_1 + 1)\}$;
$c \leftarrow \texttt{de\_Casteljaus\_1d}(b, v)$;
output: The point b_0

The rendering of `bez/teapot.bez` is given by Fig.4.

Section 2: Triangle Meshes and Half-Edge Data Structure

Part 3: Area-Weighted Vertex Normals

Suppose $\mathbf{v}$ is a vertex and F is a set of all triangles such that one of the vertices of f_i is $\mathbf{v}$. Suppose $\forall f \in F$ the area vector (oriented outward) is $\mathbf{a}_f$. Then the area-weighted vertex unit normal at vertex $\mathbf{v}$ is mathematically given by:

$$\mathbf{n_v} = \frac{\sum_{f \in F} \mathbf{a}_f}{\left\| \sum_{f \in F} \mathbf{a}_f \right\|}$$

$\mathbf{n_v}$ also orients outward in this manifold.

$\forall f \in F$, suppose the vertices in the positive orientation of f is given by $\mathbf{v}, \mathbf{v}_{1,f}, \mathbf{v}_{2,f}$. Then by the definition of cross product in $\mathbb{R}^3$, the area vector oriented outward is $\mathbf{a}_f$ is given by:

$$\mathbf{a}_f = (\mathbf{v}_{1,f} - \mathbf{v}) \times (\mathbf{v}_{2,f} - \mathbf{v})$$

Therefore,

$$\mathbf{n}_\mathbf{v} = \frac{\sum_{f \in F} (\mathbf{v}_{1,f} - \mathbf{v}) \times (\mathbf{v}_{2,f} - \mathbf{v})}{\left\| \sum_{f \in F} (\mathbf{v}_{1,f} - \mathbf{v}) \times (\mathbf{v}_{2,f} - \mathbf{v}) \right\|}$$

The renderings of `bez/teapot.bez` with flat shading and Phong shading are given by Fig.5.

Part 4: Edge Flip

The edge flip operation is the following operation: Suppose a, b, c, d are four vertices such that 2 triangles are formed between the four vertices: abc, bcd. The operation results in the triangles to be abd and acd.

The edge flip operation preserves the number of object in each class, for instance: Edge bc can be used as Edge ad; halfedge bc and cb can be used as halfedge ad and da respectively. Therefore, the operation is just a reassigning of pointers of each object (vertex, edge, face, halfedge). Pre-existing vertice, edge, face, and halfedge objects remain representing the same thing. Therefore, there are some tricks that is to be noticed:

- Because halfedges on the boundaries do not change their representation, their twins do not need to be modified.
- Vertex a and d has no halfedge removed from them, so their halfedges do not need to be modified.
- The relation between all edges and halfedges remains the same, specifically, $(bc, cb) = (ad, da) \iff bc = ad$ (halfedge on the left hand sde and edge on the right).

The table of object representation is given as Tab.3.

Halfedge	Vertex	Edge	Face
$ab \to ab$	$a \to a$	$ab \to ab$	$abc \to abd$
$bc \to ad$	$b \to b$	$bc \to ad$	$bcd \to acd$
$ca \to ca$	$c \to c$	$ca \to ca$	
$cb \to da$	$d \to d$	$bd \to bd$	
$bd \to bd$		$dc \to dc$	
$dc \to dc$			

Table 2: Object representation in edge flip operation.

The rendering of `dae/teapot.dae` with or without edge flips is included as Fig.6.

There is no eventful debugging journey in this part. We have succeeded implementing that on the first build. However, my partner and I share different perspective on visualizing this operation: my partner view it as rotating only the edge between bc to ad; I view it as rotating both the triangle together by 90 degrees without rotating the boundary.

Part 5: Edge Split

The edge split operation is the following operation: Suppose a, b, c, d are four vertices such that 2 triangles are formed between the four vertices: abc, bcd. The operation results in a new vertex being created as m, and the triangles between a, b, c, d, m are now: amb, bmd, dmc, cma where m is the midpoint between b and c.

The edge split operation requires 6 additional halfedges, 3 additional edges, 1 additional vertex, and 2 additional vertices. Therefore, the table of object representation is given as Tab.**??**. Pre-existing vertex, edge, face, and halfedge objects remain representing the same thing. Some tricks are to be noticed:

- Because halfedges on the boundaries does not change their representations, their twins do not need to be modified.
- Vertex a, c, d have no halfedge removed from them, so their halfedges do not need to be modified.
- The relation between all pre-existing edges and halfedges objects remains the same, specifically, $(bc, cb) = (bm, mb) \iff bc = bm$.
- The new edge am and md are flagged as `isNew = true` to facilitate upsampling.

Halfedge	Vertex	Edge	Face
$bc \to bm$	$b \to b$	$bc \to bm$	$abc \to abm$
$ca \to ca$	$c \to c$	$ca \to ca$	$bdc \to bdm$
$ab \to ab$	$a \to a$	$ab \to ab$	new dcm
$cb \to mb$	$d \to d$	$bd \to bd$	new cam
$bd \to bd$	new m	$dc \to dc$	
$dc \to dc$		new ma	
new ma		new dm	
new dm		new cm	
new md			
new cm			
new mc			
new am			

Table 3: Object representation in edge flip operation.

The rendering of `dae/teapot.dae` with or without edge flips or edge splits is included as Fig.7.

There is no eventful debugging journey in this part. We have succeeded implementing that on the first build.

Part 6: Loop Subdivision for Mesh Upsampling

The loop subdivision method is an upsampling method such that each triangle is topologically subdivided into 4 triangles, where the positions of the vertices are updated. The algorithm is described as follows:

- Suppose for all vertices v, the position of the vertex is P_v.
- For each vertex v, suppose the set of all neighbor vertices of v is U. Then the new position of v, P_v, is:

  $$P_v \leftarrow (1 - |U|\alpha)P_v + \alpha \sum_{u \in U} P_u$$

  where

  $$\alpha = \begin{cases} \frac{3}{16} & , |U| = 3 \\ \frac{3}{8|U|} & , |U| > 3 \end{cases}$$

  Moreover, upon iteration through the vertices, set all vertex to be pre-existing ones.
- For each edge in the old mesh, the edge would be split into two, and a new vertex would be created as v. Suppose u_1 and u_2 are the two endpoints of the original edge and u_1' and u_2' are the vertices opposite to the original edge that was split in the two adjacent triangles. Then the position of v is given by:

  $$P_v \leftarrow \frac{3}{8}(P_{u_1} + P_{u_2}) + \frac{1}{8}(P_{u_1'} + P_{u_2'})$$

  Note that P_{u_1}, P_{u_2}, $P_{u_1'}$, $P_{u_2'}$ refers to the position of the vertices previous to the update of the pre-existing vertices. To get away from that, the newly computed vertex positions are stored separately before they are transferred into the vertex at the end of the algorithm. Because no new vertex has yet been created, it is stored temporarily as the position linked to the edge.

  Moreover, upon iteration through the edges, set all edges to be pre-existing ones.

  Debugging journey: We did not thought of that at first, but it gives us error in the case where split is done before upsampling, or upsampling twice. Suppose two triangles abc and bcd. The edge split of bc results in amb, bmd, dmc, cma as in the edge split part. ma and md are flagged as new, so when we split them in upsampling, the midpoint of ma, denoted as m', would be in the following situation: either $m'a$ or $m'm$ is new, inherited from ma. This would lead that edge to be split, but it should not be the case. Therefore, we force all edges and vertices flagged as `isNew = false` *before edge flip and split.*
- Split all pre-existing edges in any order. A new vertex is created as this moment, so the position of the vertex is given by the position linked to the edge computed in the last step. The new
- Flip all edges that are flagged as new edges in the edge split operation that links a newly created vertex to a pre-existing vertex.

- Copy the computed value in the first step to the position of the pre-existing vertices. Flag all edges and vertices as pre-existing (`isNew = false`).

For a regularly defined mesh, the new mesh is simply a denser version that shares the same shape with the original mesh (minified by a factor of 2). This well preserves the uniformity of the mesh since the relative structural difference between triangles is not changed significantly.

Moreover, sharp corners are rounded because the upsampling algorithm tends to make the mesh behave more smoothly. Therefore, if a corner wants to be preserved, the mesh has to be dense at the corner order to detail the structure of the object at that location.

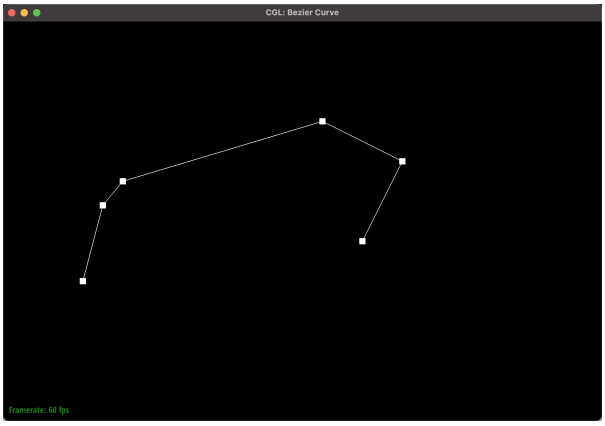
The rendering of the original `dae/teapot.dae` and with upsampling is included as Fig.8.

For the cube in `dae/cube.dae`, the previous mesh is not symmetric under all symmetry of a cube.

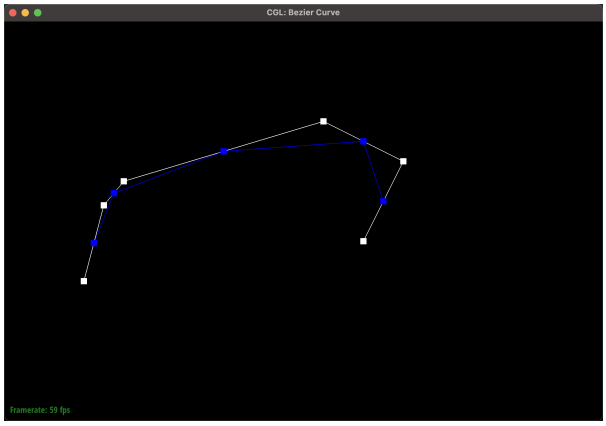
For mathematical establishment, suppose the center of the cube is positioned at the origin of the coordinate system with face normals in $\mathbf{e}_x$, $\mathbf{e}_y$, and $\mathbf{e}_z$ directions where $\mathbf{e}_x$, $\mathbf{e}_y$, and $\mathbf{e}_z$ forms a positive oriented orthonormal basis in $\mathbb{R}^3$.

The group of all operation that preserves the symmetry of the cube is given by the following subgroup in $O(\mathbb{R}^3)$ (the rotation group in $\mathbb{R}^3$): the subgroup generated by $R\left(\frac{\pi}{2}\mathbf{e}_x\right)$, $R\left(\frac{\pi}{2}\mathbf{e}_y\right)$, and the inversion I where R is the axis-angle representation of proper rotation in $\mathbb{R}^3$.

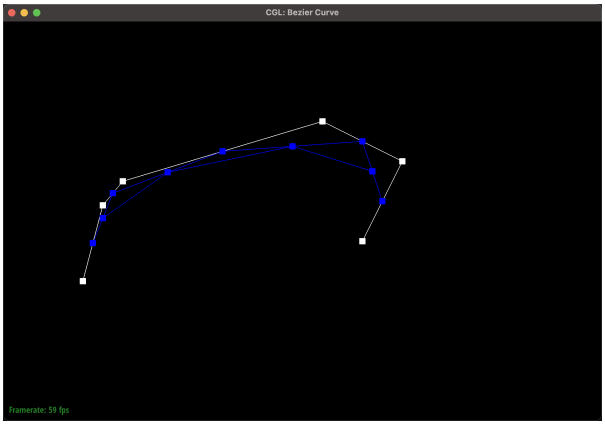
Therefore, the topology of the mesh must also satisfies the symmetry specified above. An easy way to do this is to split all diagonal edges. Therefore, all faces looks the same: a cross produced by two diagonals, which maintains a rotation symmetry by an angle of $\frac{\pi}{2}$ in the direction of the surface normal. Therefore, this would produce a symmetric mesh after upsampling. Fig.9 shows it is really the case.



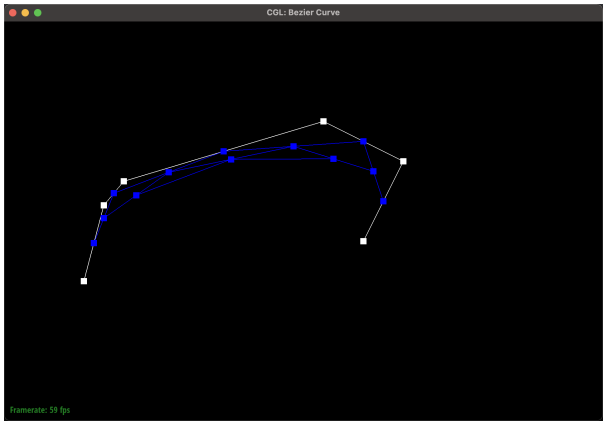
(a) Level 0



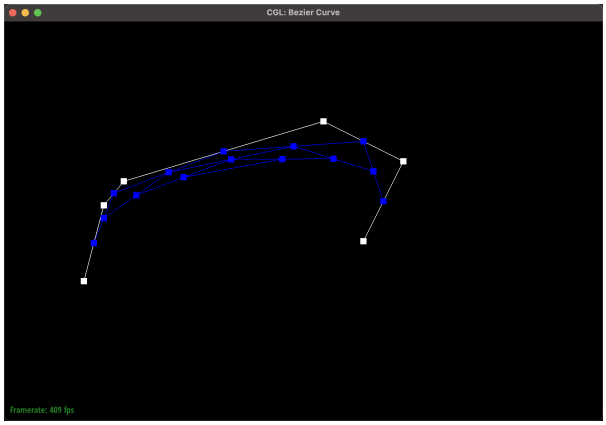
(b) Level 1



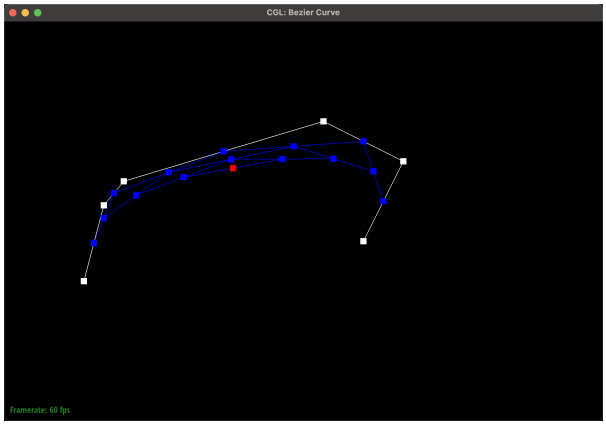
(c) Level 2



(d) Level 3



(e) Level 4



(f) Level 5

Figure 2: Levels in which the Bezier curve given by `bzc/curve3.bzc` is rendered.

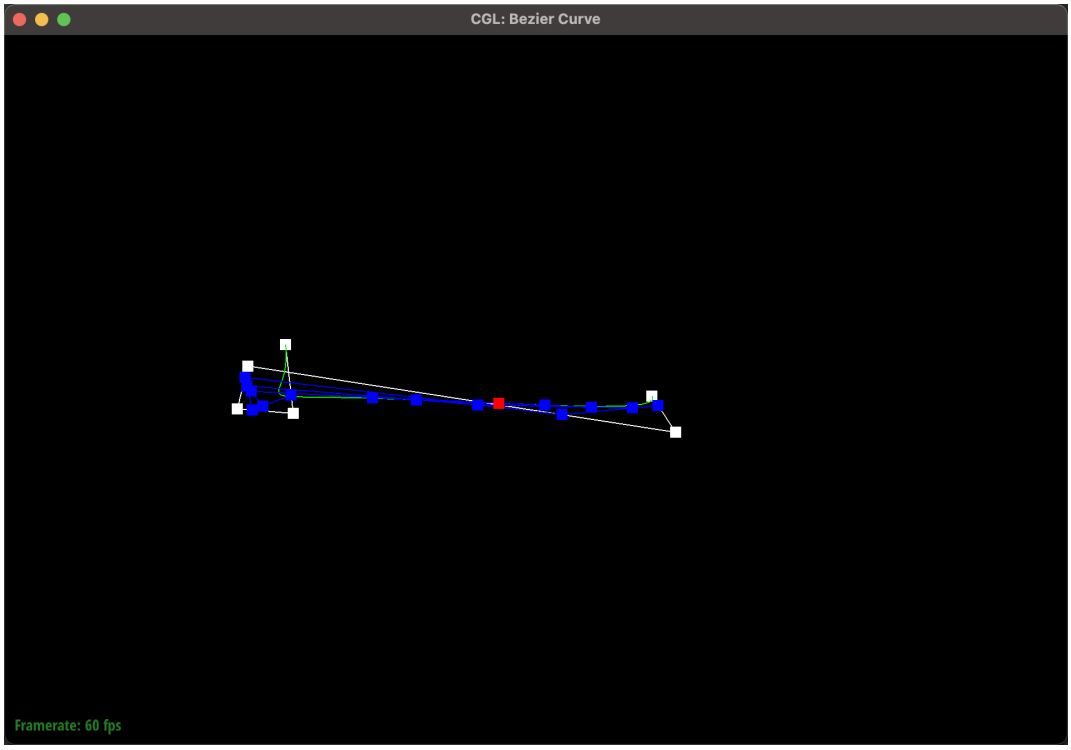


Figure 3: Modified rendering of the Bezier curve given by `bzc/curve3.bzc`

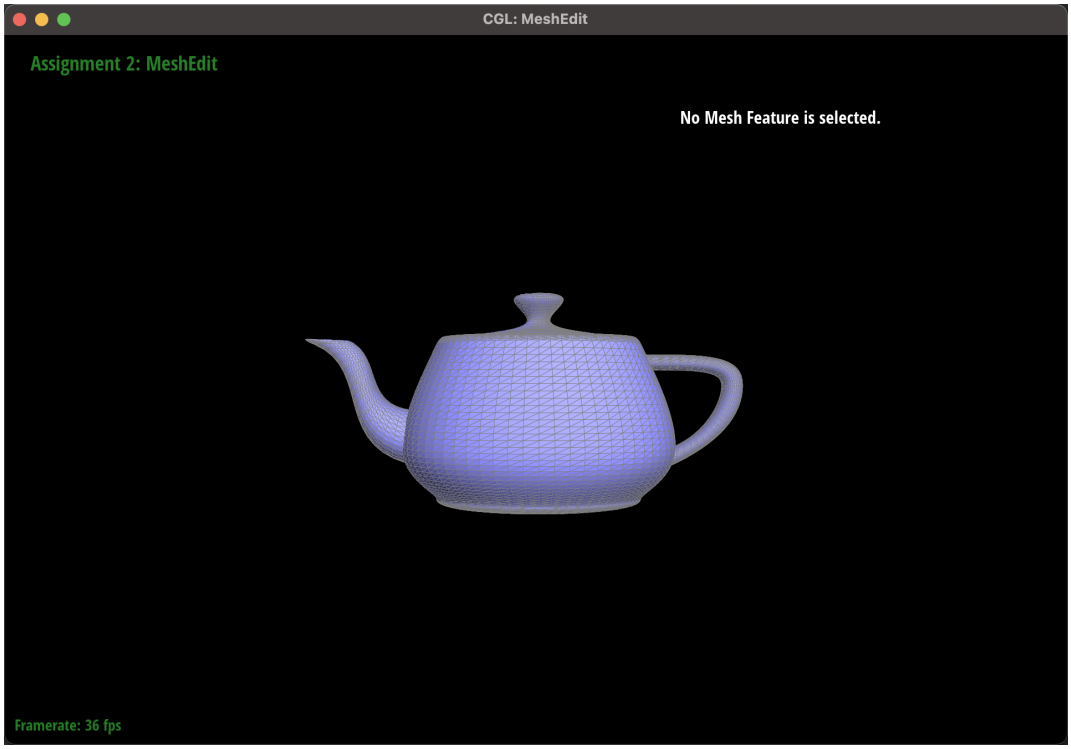
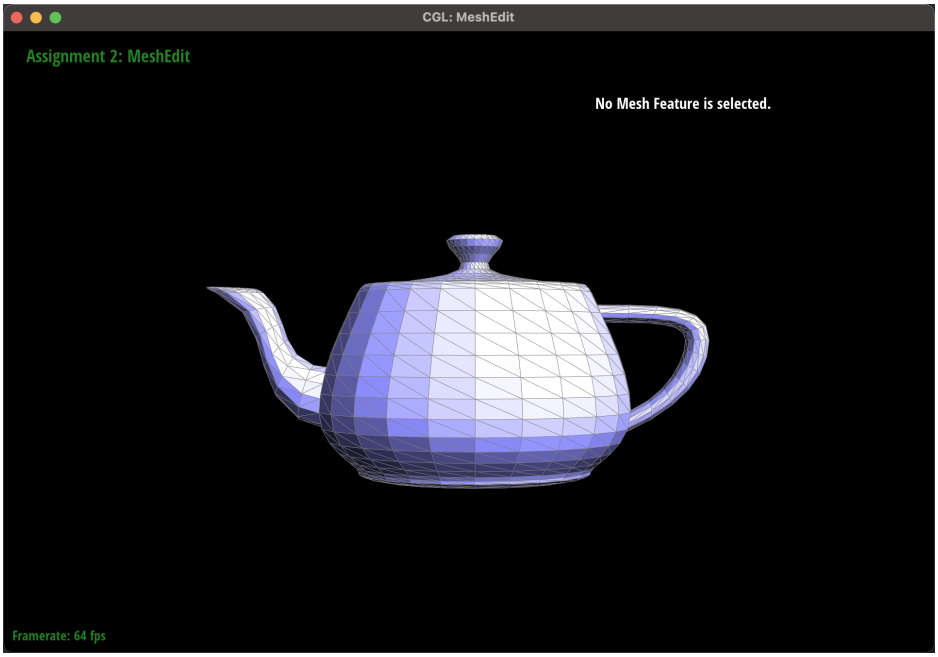
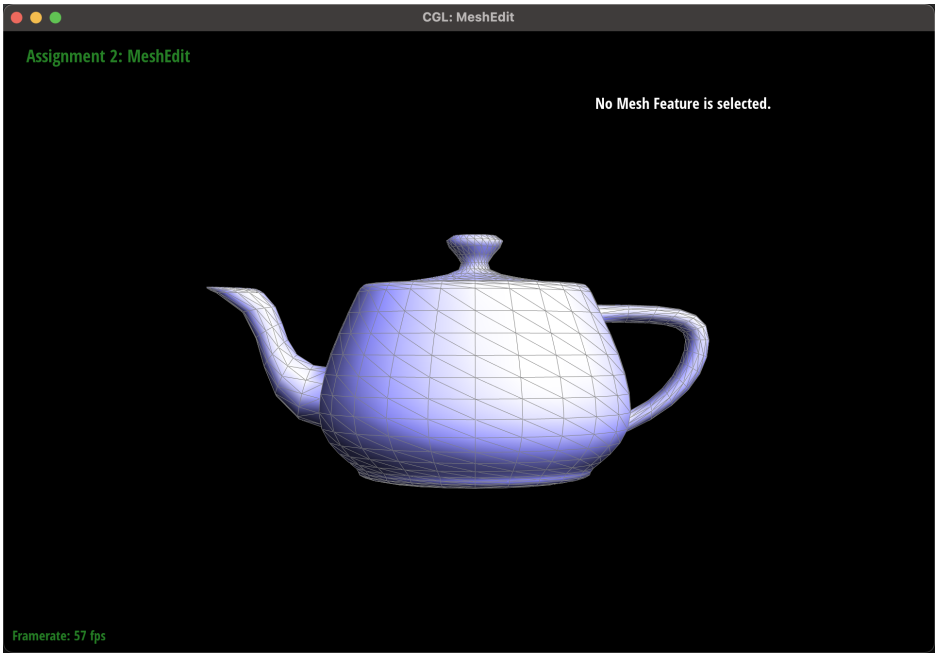


Figure 4: Rendering of `bez/teapot.bez`.

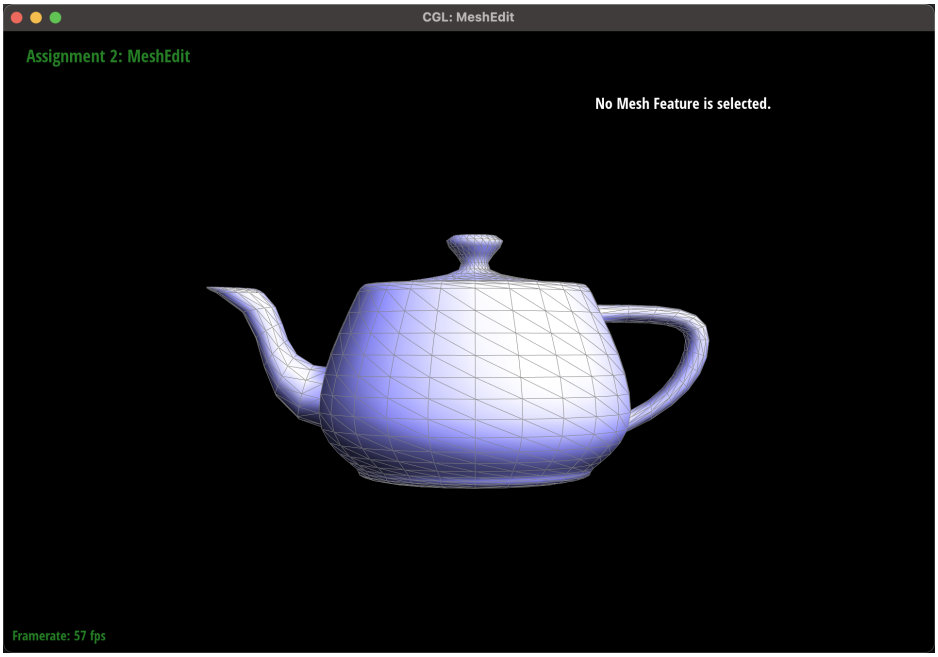


(a) Flat shading

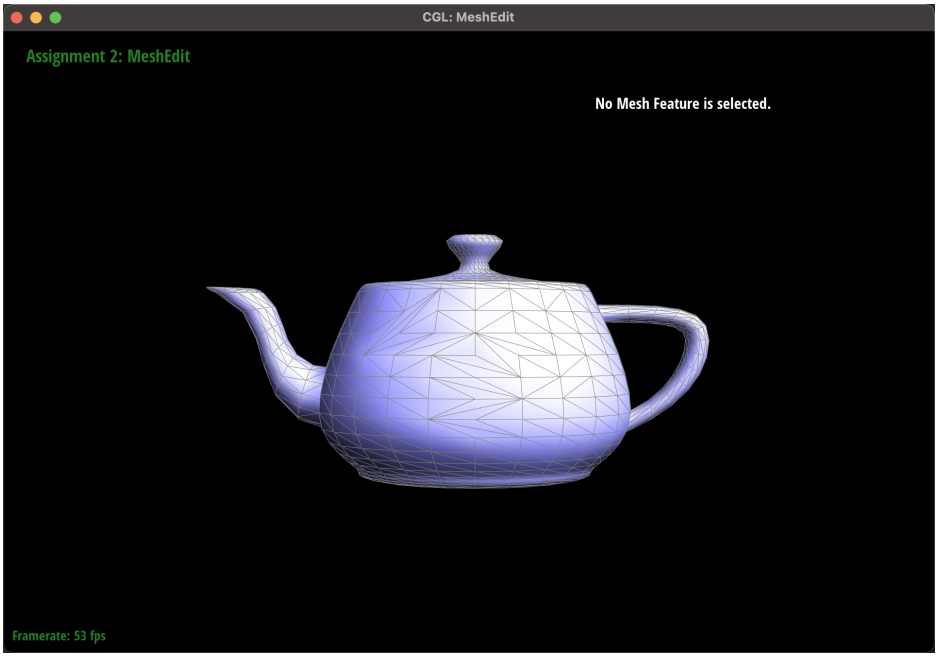


(b) Phong shading

Figure 5: Rendering of `dae/teapot.dae` with different shading.

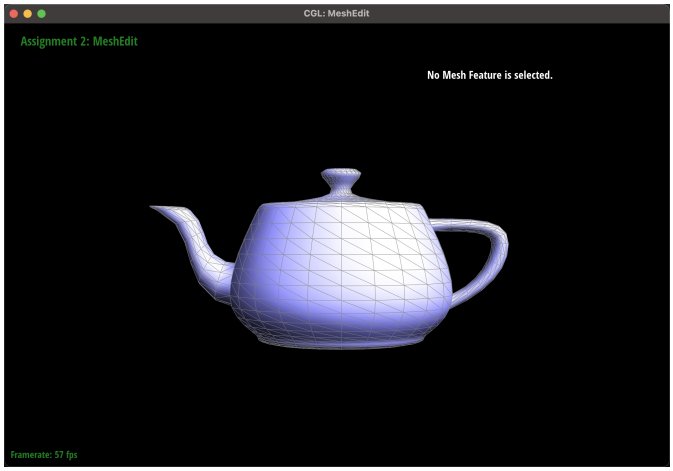


(a) Original mesh

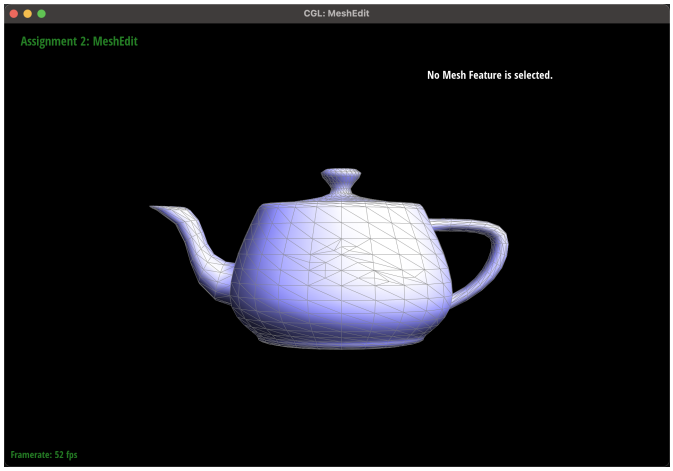


(b) Mesh with some edge flips

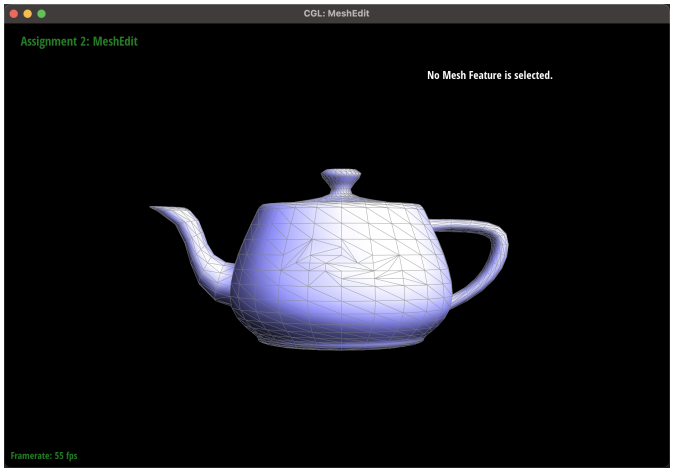
Figure 6: Rendering of `dae/teapot.dae` with or without edge flips.



(a) Original mesh

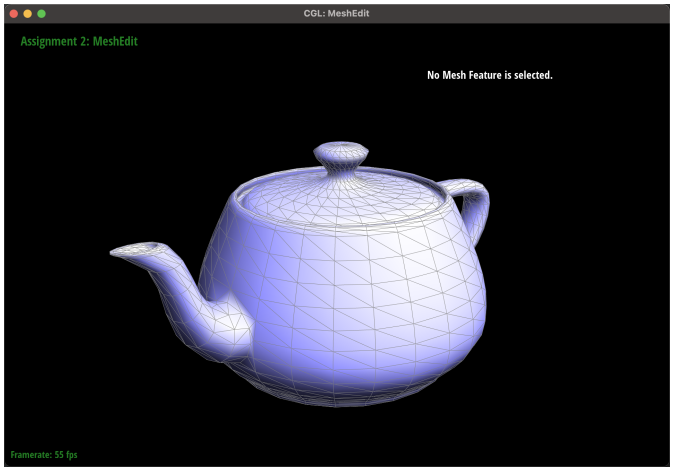


(b) Mesh with only edge splits

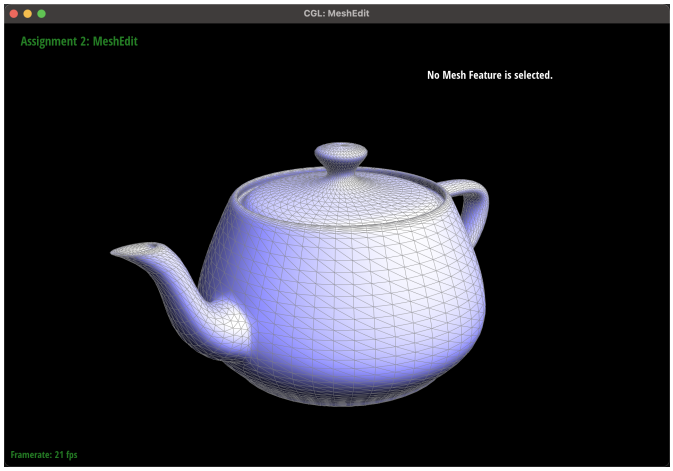


(c) Mesh with both edge flips and edge splits

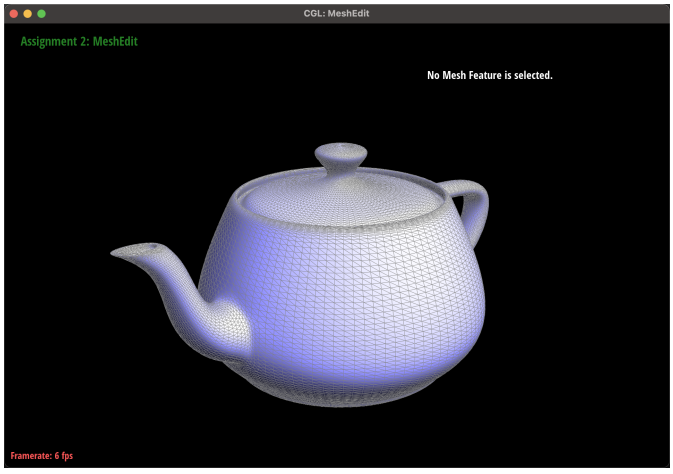
Figure 7: Rendering of `dae/teapot.dae` with or without edge flips or edge splits.



(a) Original mesh

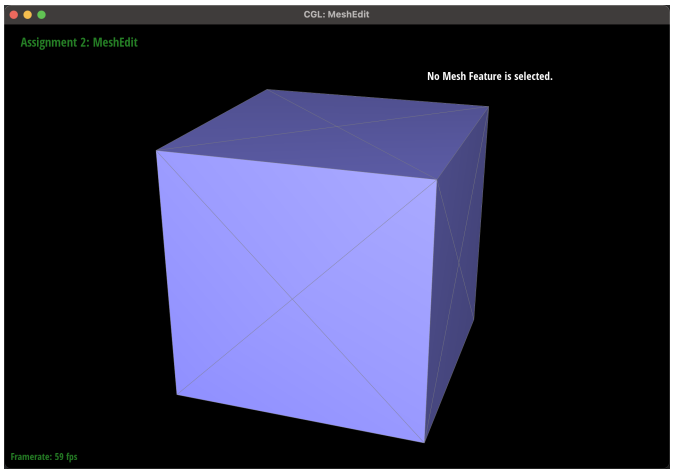


(b) Upsampled once

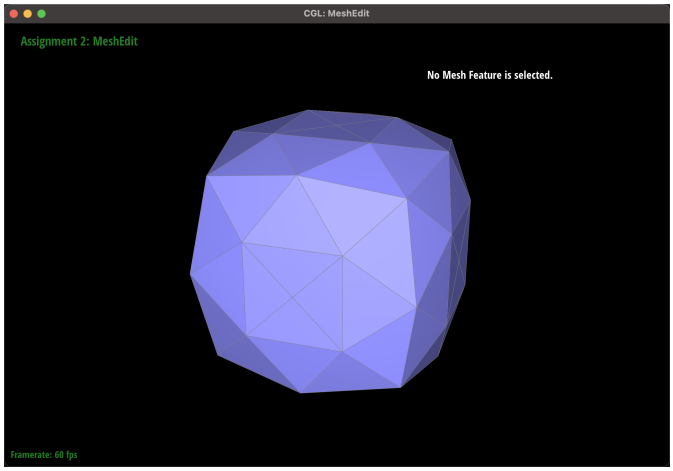


(c) Upsampled twice

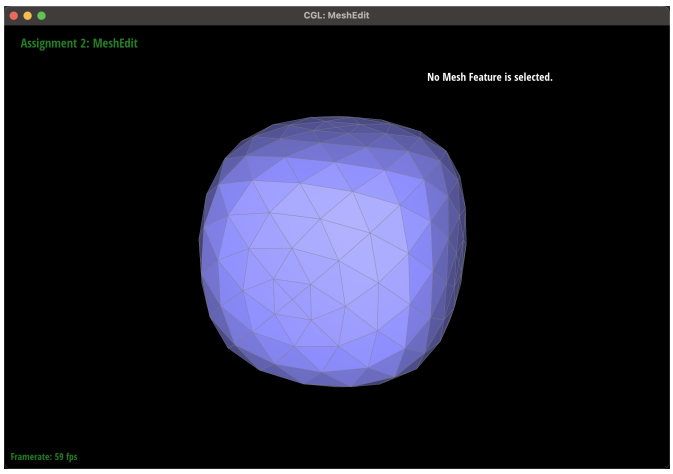
Figure 8: Rendering of the original `dae/teapot.dae` and with upsampling.



(a) Original mesh where all diagonal edges are split



(b) Upsampled once



(c) Upsampled twice

Figure 9: Rendering of the original `dae/cube.dae` and with upsampling after preprocessing: edge split all diagonal edges on each face of the cube.