Advanced Computer Graphics
Assignment I Report

Tianyu Huang, 2021010656
软件 21, School of Software
huang-ty21@mails.tsinghua.edu.cn

October 17, 2023

1 Curves

1.1 Proof.

$b_{i,n}(t)$ can be expanded as

$$b_{i,n}(t) = \frac{n!}{(n-i)!i!}t^i(1-t)^{n-i} \tag{1}$$

Computing the derivative of $b_{i,n}(t)$ we get:

$$\frac{d}{dt}b_{i,n}(t) = \begin{cases} n(b_{i-1,n-1}(t) - b_{i,n-1}(t)), & 0 < i < n \\ -nb_{i,n-1}(t), & i = 0 \\ nb_{i-1,n-1}(t), & i = n \end{cases} \tag{2}$$

So the derivative of $\boldsymbol{B}(t)$ can be written as

$$\begin{aligned} \boldsymbol{B}'(t) &= \sum_{i=0}^{n} b'_{i,n}(t)\boldsymbol{P}_i \\ &= b'_{0,n}(t)\boldsymbol{P}_0 + b'_{1,n}(t)\boldsymbol{P}_1 + \cdots + b'_{n,n}(t)\boldsymbol{P}_n \\ &= n\Big[- b_{0,n-1}(t)\boldsymbol{P}_0 + (b_{0,n-1}(t) - b_{1,n-1}(t))\boldsymbol{P}_1 + \cdots + b_{n-1,n-1}(t)\boldsymbol{P}_n\Big] \\ &= n\Big[b_{0,n-1}(t)(\boldsymbol{P}_1 - \boldsymbol{P}_0) + \cdots + b_{n-1,n-1}(t)(\boldsymbol{P}_n - \boldsymbol{P}_{n-1})\Big] \\ &= n\sum_{i=0}^{n-1} b_{i,n-1}(t)(\boldsymbol{P}_{i+1} - \boldsymbol{P}_i) \quad \square \end{aligned} \tag{3}$$

1.2 Proof.

It is easy to see that

$$\sum_{i=0}^{n} b_{i,n}(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = [t + (1-t)]^n = 1, \forall t \in [0,1] \tag{4}$$

Meanwhile,

$$b_{i,n}(t) > 0, \forall t \in [0,1] \tag{5}$$

In this view, $\boldsymbol{B}(t)$ becomes a **convex combination** of all $\boldsymbol{P}_i$ with Bernstein polynomials as base:

$$\boldsymbol{B}(t) = \sum_{i=0}^{n} b_{i,n}(t)\boldsymbol{P}_i \in \mathbf{C}, \forall t \in [0,1] \tag{6}$$

It implies that

$$\boldsymbol{B} \subset \boldsymbol{C} \tag{7}$$

where $\mathbf{C}$ is the convex hull of all control points $\{\boldsymbol{P}_i\}_{i=0}^{n}$. □

2 Surfaces

Before I begin, I want to make it clear:

According to the Q&A information publicly available on Piazza, the surfaces involved in this assignment are simple geometries **homeomorphic to a sphere**. Therefore all implementations below use this premise.

2.1 Mesh Subdivision

2.1.1 Basic Implementation

The mesh subdivision algorithm itself is quite trivial, I will not repeat the algorithm introduced on the ppt here. The core problem of mesh subdivision is how to efficiently query the geometric structures around a certain edge or a certain vertex.

2.1.2 Optimization: The *Half-edge* Data Structure

The half-edge data structure has following characteristics:

- Each edge in the mesh structure is represented as 2 opposite directed edges (half-edges).
- Each face in the mesh structure is represented as an island composed of a closed loop of half-edges, where the direction of rotation of the directed edges conforms to the normal direction.

- In each half-edge, the other half-edge *opposite* to the half-edge is stored. (Actually, it is the corresponding half-edge in the adjacent face.)
- It is not difficult to prove that for all mesh structure that is homeomorphic to a sphere, all directed half-edges have a unique opposite half-edge.

In order to utilize the half-edge data structure, the following two operations need to be completed in programming implementation:

- Creation of half-edge data structure. To build the half-edge data structure, we need to iterate over each face and add the half-edges to the half-edge list in counterclockwise order. After the face traversal is completed, we also need to traverse the half-edge list to determine the opposite half-edge of each half-edge.
- Update of half-edge data structure. Updating the half-edge data structure is much trickier, so I'll explain that in a new paragraph.

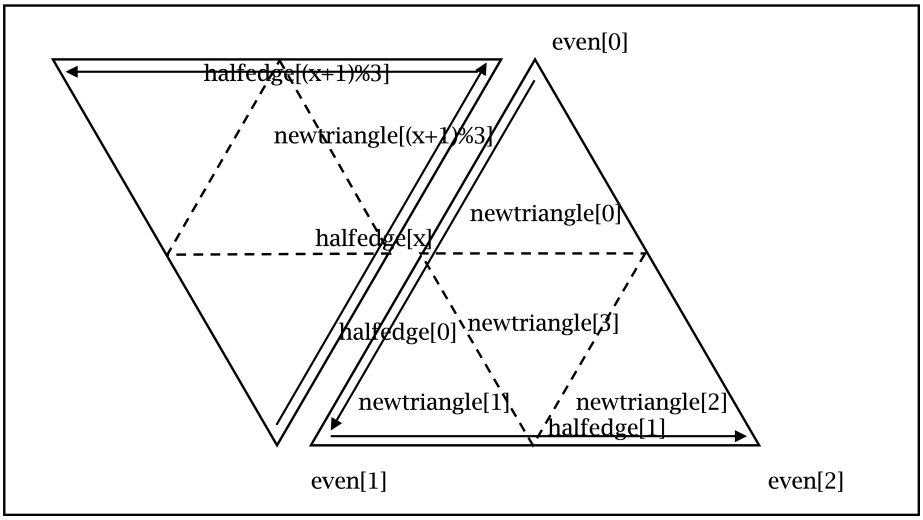

Figure 1: Schematic used to show half-edge data structure update.

The core problem of updating the half-edge data structure is **how to determine the opposite half-edge of the new half-edge**. If the traversal algorithm used to create the half-edge data structure is used, the time complexity will be very poor, so a method that conforms to the characteristics of Loop Subdivision must be used.

During the creation of the half-edge data structure, we placed the three half-edges of the same face at three consecutive positions in the half-edge list. We can use the same idea in the update of the half-edge data structure caused by Loop Subdivision. Specifically, an old face corresponds to 4 new faces, that is,

12 half-edges. Then we can store all the new half-edges generated by an old face in 12 consecutive positions in the half-edge list. From this, we can calculate the new half-edges based on the old faces. The opposite half-edge entries of the old half-edges of the surface determines the opposite half-edge entries of the new half-edges (just do some modulo and multiplication and division to derive the array index).

2.2 Mesh Simplification

In this section, I was instructed to implement two functions.

2.2.1 The Q Matrix

This function computes the Q matrix for a single plane.

Following the work of M Garland et al., in this function I only need to determine the plane equation based on the three vertices of the triangle:

$$\boldsymbol{p}^\intercal \boldsymbol{v} = \begin{pmatrix} a & b & c & d \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \tag{8}$$

where

$$\sqrt{a^2 + b^2 + c^2} = 1 \tag{9}$$

Then we compute the fundamental error quadric

$$\boldsymbol{Q} = \boldsymbol{p}\boldsymbol{p}^\intercal \tag{10}$$

2.2.2 Generate Merged Point

~~In this function, I need to implement equation 1 from the work of M Garland et al:~~

$$\bar{\boldsymbol{v}} = \begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{11}$$

~~If the matrix is not invertible, I was instructed to return the midpoint.~~

The **CORRECT** implementation of this part is:

- First, try to calculate $\bar{\boldsymbol{v}}$ using the following formula (M Garland et al.):

$$\bar{\boldsymbol{v}} = \begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{12}$$

- If the matrix is not invertible, then try to optimize on line segments.

  Note that the following method was adopted in the demo program of M Garland et al. (`http://www.cs.cmu.edu/~./garland/quadrics/qslim.html`) However, it seems that M Garland et al. did NOT give a mathematical derivation in their paper or other documents(?) - at least I did not find that. And most implementations of QEM on GitHub does not implement this part correctly.

  In the function I was given two end vertices: $\vec{\boldsymbol{p}}_0$, $\vec{\boldsymbol{p}}_1$. Let:

  $$\vec{\boldsymbol{d}} = \vec{\boldsymbol{p}}_0 - \vec{\boldsymbol{p}}_1 \tag{13}$$

  So any point between the 2 end vertices can be expressed as:

  $$\vec{\boldsymbol{p}} = a\vec{\boldsymbol{d}} + \vec{\boldsymbol{p}}_1, a \in [0, 1] \tag{14}$$

  Our goal is to determine

  $$\min \begin{pmatrix} \vec{\boldsymbol{p}}^\intercal & 1 \end{pmatrix} \mathbf{Q} \begin{pmatrix} \vec{\boldsymbol{p}} \\ 1 \end{pmatrix} = \min \begin{pmatrix} \vec{\boldsymbol{p}}^\intercal & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \vec{\boldsymbol{b}} \\ \vec{\boldsymbol{b}}^\intercal & d^2 \end{pmatrix} \begin{pmatrix} \vec{\boldsymbol{p}} \\ 1 \end{pmatrix} \tag{15}$$

  Do some expansion:

  $$\begin{aligned} \begin{pmatrix} \vec{\boldsymbol{p}}^\intercal & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \vec{\boldsymbol{b}} \\ \vec{\boldsymbol{b}}^\intercal & d^2 \end{pmatrix} \begin{pmatrix} \vec{\boldsymbol{p}} \\ 1 \end{pmatrix} = a^2 \vec{\boldsymbol{d}}^\intercal \mathbf{A} \vec{\boldsymbol{d}} + a\vec{\boldsymbol{d}}^\intercal \mathbf{A} \vec{\boldsymbol{p}}_1 + a\vec{\boldsymbol{p}}_1^\intercal \mathbf{A} \vec{\boldsymbol{d}} + \vec{\boldsymbol{d}}_1^\intercal \mathbf{A} \vec{\boldsymbol{p}}_1 \\ + 2a\vec{\boldsymbol{d}}^\intercal \vec{\boldsymbol{b}} + \vec{\boldsymbol{v}}_0^\intercal \vec{\boldsymbol{b}} + \vec{\boldsymbol{b}}^\intercal \vec{\boldsymbol{p}}_1 + d^2 \end{aligned} \tag{16}$$

  Calculate the derivative of RHS and make it equal to 0, we get

  $$a_0 = \frac{-\vec{\boldsymbol{d}} \cdot \mathbf{A}\vec{\boldsymbol{p}}_1 - \vec{\boldsymbol{p}}_1 \cdot \mathbf{A}\vec{\boldsymbol{d}} - 2\vec{\boldsymbol{b}} \cdot \vec{\boldsymbol{d}}}{2\vec{\boldsymbol{d}} \cdot \mathbf{A}\vec{\boldsymbol{d}}} \tag{17}$$

  Note that the denominator $\neq 0$, we will deal with the $= 0$ case later.

  Then we clamp a_0 between 0 and 1.

  $$\bar{\boldsymbol{v}} = a_0\vec{\boldsymbol{d}} + \vec{\boldsymbol{p}}_1 \tag{18}$$

- If the denominator above $= 0$. We calculate the errors of the two end vertices respectively and the error of the midpoint. Then we choose the one with the smallest error.

3 Results

The test environment is Apple Macbook Air M1 (2020), so it may have better running results on desktop devices.

3.0.1 Standard Output

```
% ./mesh_main meshes/icosahedron_input.obj results/
    icosahedron_subdivided_1.obj subdivide 1
Loaded 20 faces and 12 vertices
Execution took 0 milliseconds.
% ./mesh_main meshes/icosahedron_input.obj results/
    icosahedron_subdivided_3.obj subdivide 3
Loaded 20 faces and 12 vertices
Execution took 16 milliseconds.
% ./mesh_main meshes/bunny_200.obj results/
    bunny_200_subdivided_1.obj subdivide 1
Loaded 200 faces and 102 vertices
Execution took 5 milliseconds.
% ./mesh_main meshes/bunny_200.obj results/
    bunny_200_subdivided_3.obj subdivide 3
Loaded 200 faces and 102 vertices
Execution took 99 milliseconds.
% ./mesh_main meshes/bunny.obj results/
    bunny_simplified_10000.obj simplify 10000
Loaded 17416 faces and 8710 vertices
Execution took 599 milliseconds.
% ./mesh_main meshes/bunny.obj results/
    bunny_simplified_17000.obj simplify 17000
Loaded 17416 faces and 8710 vertices
Execution took 885 milliseconds.
% ./mesh_main meshes/cow.obj results/cow_simplified_5000
    .obj simplify 5000
Loaded 5804 faces and 2904 vertices
Execution took 251 milliseconds.
% ./mesh_main meshes/cow.obj results/cow_simplified_5600
    .obj simplify 5600
Loaded 5804 faces and 2904 vertices
Execution took 274 milliseconds.
```

3.0.2 Screenshots

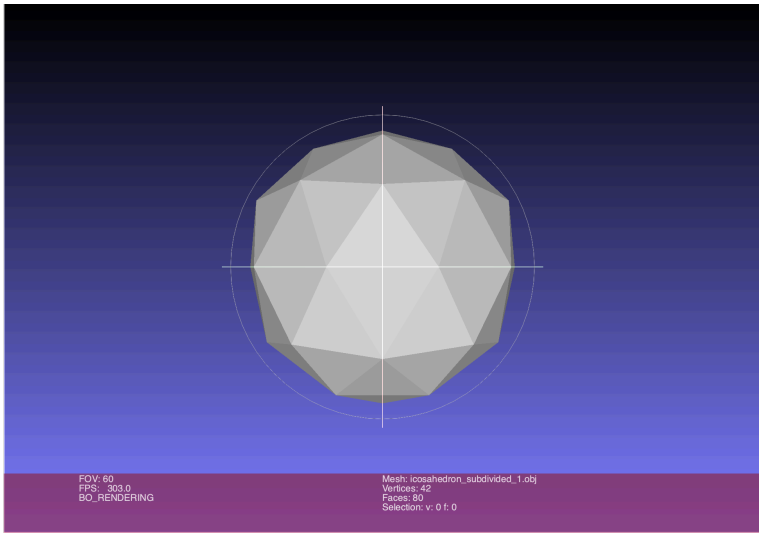

Figure 2: Icosahedron Subdivided 1

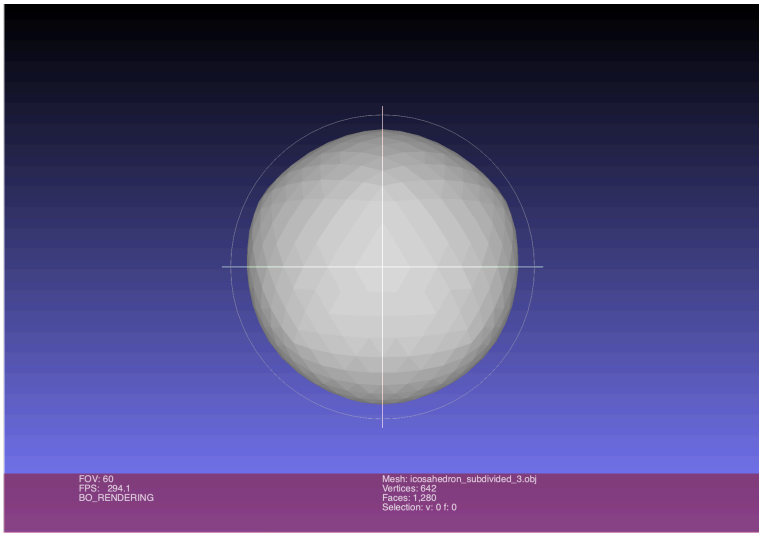

Figure 3: Icosahedron Subdivided 3

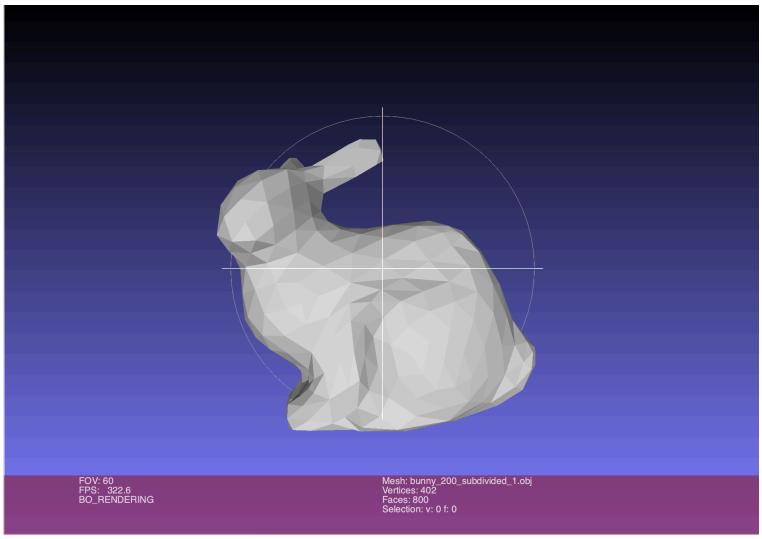

Figure 4: Bunny_200 Subdivided 1

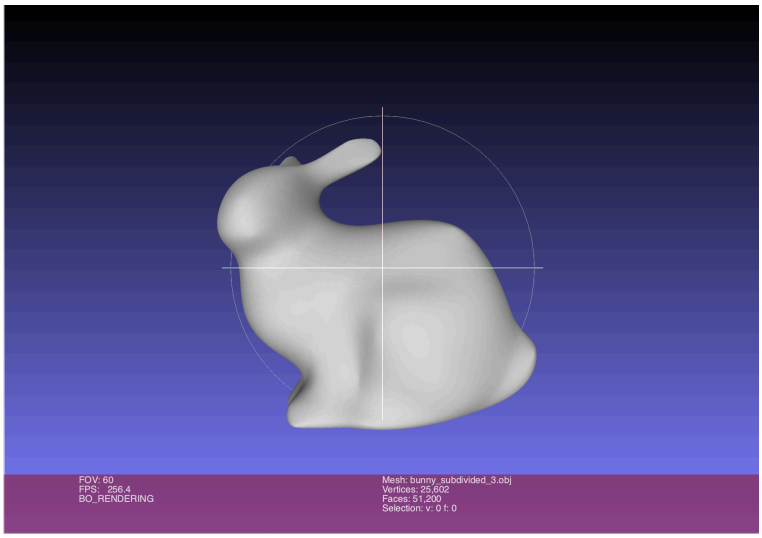

Figure 5: Bunny_200 Subdivided 3

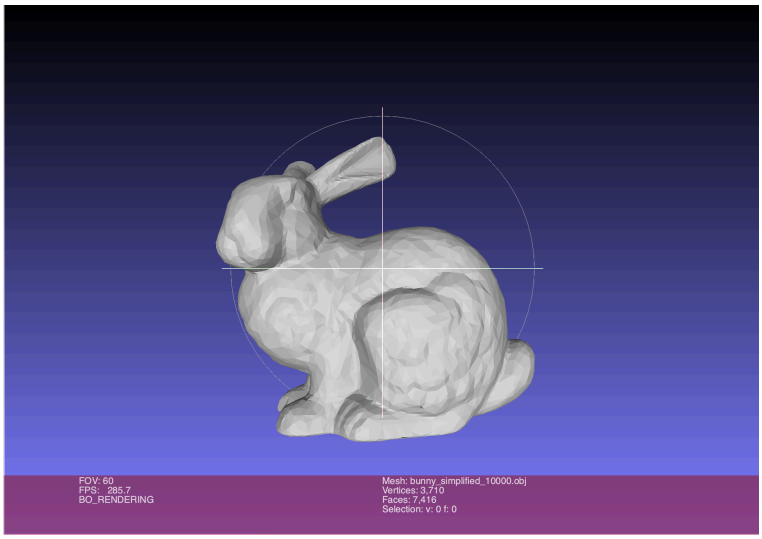

Figure 6: Bunny Simplified 10000

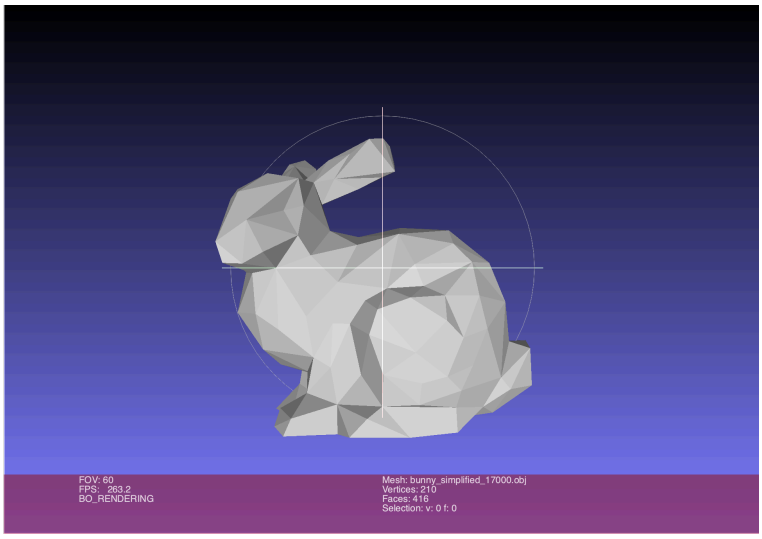

Figure 7: Bunny Simplified 17000

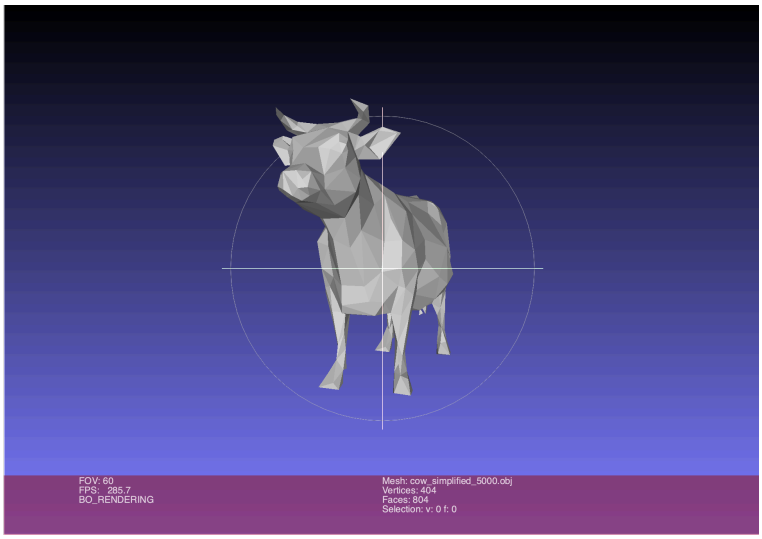

Figure 8: Cow Simplified 5000

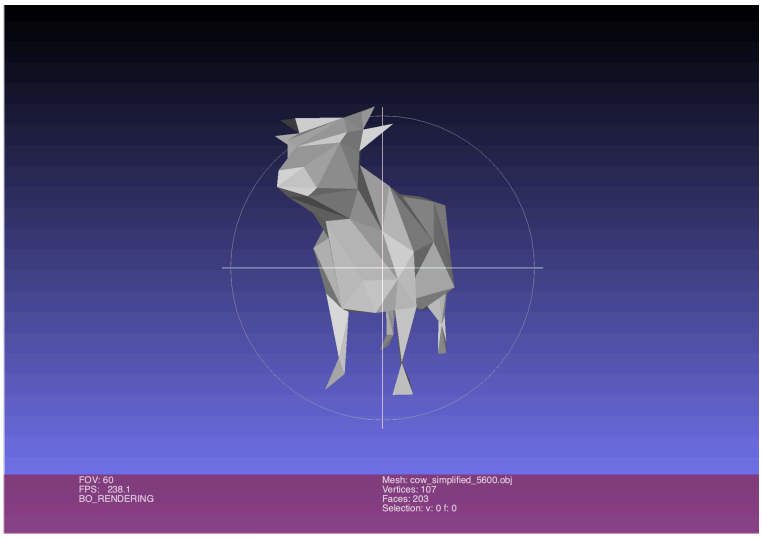

Figure 9: Cow Simplified 5600

4 Acknowledgement

Loop Subdivision only refers to the content of the course ppt. The half-edge data structure is a commonly used data structure in CG. I did not refer to more literature, and the half-edge data structure update in Loop Subdivision is my

original work (but I did not pay attention to whether anyone has adopted this method. I think there should be, because this method is trivial). For the QEM method, this assignment refers to the work of M Garland et al.

It is worth noting that for the edge merging part of QEM, M Garland et al. implemented a similar multi-fallback algorithm in their example program. But they did not seem to give a detailed mathematical derivation in the paper or other documents (?). In my report, the detailed mathematical derivation is provided.