MIXAR HIRING ASSIGNMENT

Vedant Acharya

Github Repository Link:
https://github.com/Ninjacoder-vedant/mixar_hiring_assignment

TABLE OF CONTENTS

Task 1 ..... **3**
Task 2 ..... **6**
Task 3 ..... **7**
Option 1: Seam Tokenization Prototype **9**

Main tasks

Task 1

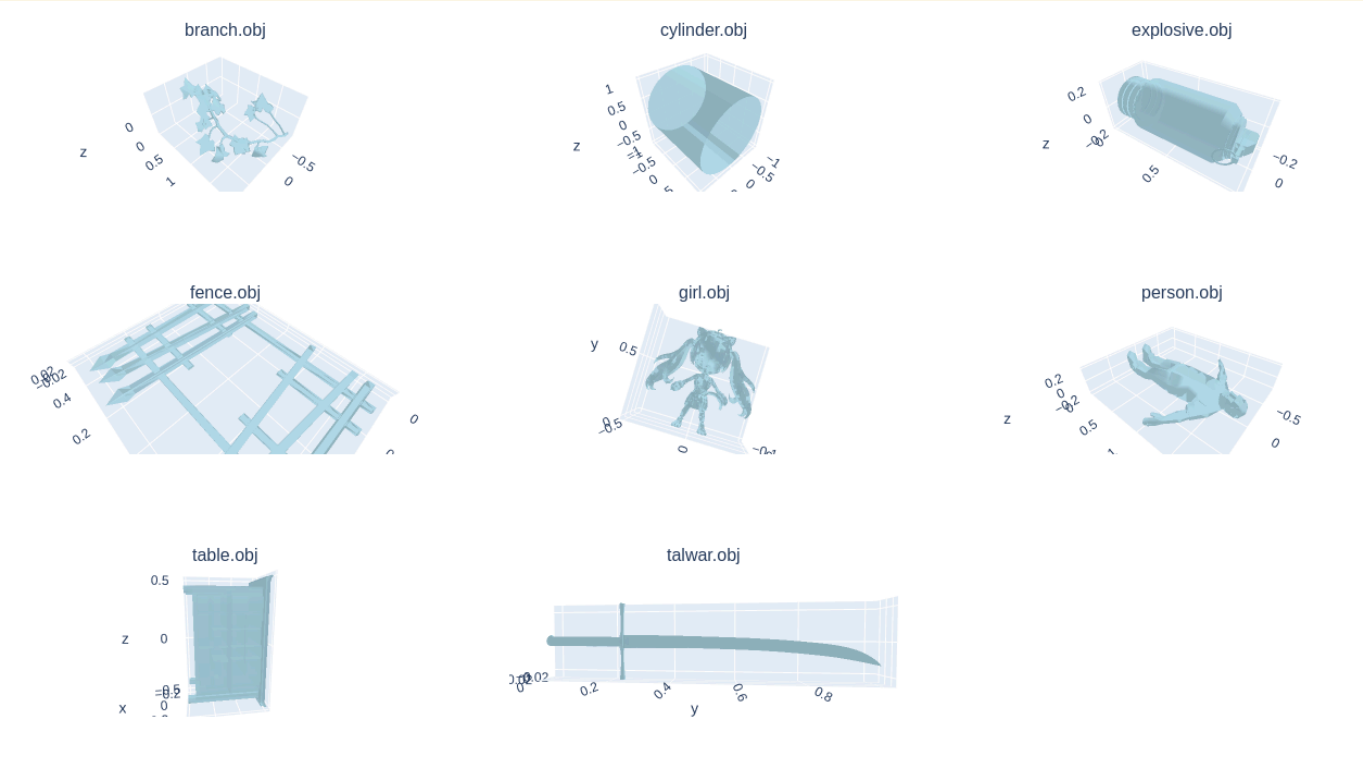
1. Statistics for all 8 .obj files are shown below:

```
========== branch.obj ==========

--- Basic Shapes ---
Vertices (v):        (2767, 3)
Faces (f):           (1960, 3)
Normals (vn):        (2767, 3)
Texture coords (vt): (2767, 2)

--- Vertex Statistics ---
Number of vertices: 2767
Min (x, y, z): [-0.851562,  0.000000, -0.464844]
Max (x, y, z): [ 0.849609,  1.900391,  0.462891]
Mean (x, y, z): [ 0.075443,  1.087390,  0.121967]
Std  (x, y, z): [ 0.343380,  0.456991,  0.200067]

 ========== cylinder.obj ==========

--- Basic Shapes ---
Vertices (v):        (192, 3)
Faces (f):           (124, 3)
Normals (vn):        (192, 3)
Texture coords (vt): (192, 2)

--- Vertex Statistics ---
Number of vertices: 192
Min (x, y, z): [-1.000000, -1.000000, -1.000000]
Max (x, y, z): [ 1.000000,  1.000000,  1.000000]
Mean (x, y, z): [-0.000000,  0.000000,  0.000000]
Std  (x, y, z): [ 0.707107,  1.000000,  0.707107]

 ========== explosive.obj ==========

--- Basic Shapes ---
Vertices (v):        (2812, 3)
Faces (f):           (2566, 3)
Normals (vn):        (2812, 3)
Texture coords (vt): (2812, 2)
```

```
--- Vertex Statistics ---
Number of vertices: 2812
Min (x, y, z): [-0.199625, -0.000000, -0.197126]
Max (x, y, z): [ 0.199625,  1.000000,  0.197126]
Mean (x, y, z): [ 0.042888,  0.529113, -0.003446]
Std  (x, y, z): [ 0.115096,  0.389941,  0.094676]

 ========== fence.obj ==========

--- Basic Shapes ---
Vertices (v):        (1088, 3)
Faces (f):           (684, 3)
Normals (vn):        (1088, 3)
Texture coords (vt): (1088, 2)

--- Vertex Statistics ---
Number of vertices: 1088
Min (x, y, z): [-0.500000,  0.000000, -0.022500]
Max (x, y, z): [ 0.500000,  0.843170,  0.022500]
Mean (x, y, z): [-0.003508,  0.410475, -0.000441]
Std  (x, y, z): [ 0.345794,  0.254046,  0.010981]

 ========== girl.obj ==========

--- Basic Shapes ---
Vertices (v):        (8284, 3)
Faces (f):           (8475, 3)
Normals (vn):        (8284, 3)
Texture coords (vt): (8284, 2)

--- Vertex Statistics ---
Number of vertices: 8284
Min (x, y, z): [-0.500000,  0.000000, -0.181411]
Max (x, y, z): [ 0.500000,  0.904419,  0.181411]
Mean (x, y, z): [ 0.002113,  0.403385,  0.014002]
Std  (x, y, z): [ 0.178756,  0.214389,  0.061790]

 ========== person.obj ==========

--- Basic Shapes ---
Vertices (v):        (3103, 3)
Faces (f):           (2251, 3)
Normals (vn):        (3103, 3)
Texture coords (vt): (3103, 2)

--- Vertex Statistics ---
```

```
Number of vertices: 3103
Min (x, y, z): [-0.843750, -0.000000, -0.212891]
Max (x, y, z): [ 0.841797,  1.900391,  0.210938]
Mean (x, y, z): [ 0.004887,  1.159460, -0.003614]
Std  (x, y, z): [ 0.395132,  0.511882,  0.095144]

 ========== table.obj ==========

--- Basic Shapes ---
Vertices (v):        (3148, 3)
Faces (f):           (4100, 3)
Normals (vn):        (3148, 3)
Texture coords (vt): (3148, 2)

--- Vertex Statistics ---
Number of vertices: 3148
Min (x, y, z): [-0.208906,  0.000000, -0.500000]
Max (x, y, z): [ 0.208906,  0.611761,  0.500000]
Mean (x, y, z): [-0.013190,  0.386374, -0.003587]
Std  (x, y, z): [ 0.153119,  0.191922,  0.346052]

 ========== talwar.obj ==========

--- Basic Shapes ---
Vertices (v):        (1668, 3)
Faces (f):           (1922, 3)
Normals (vn):        (1668, 3)
Texture coords (vt): (1668, 2)

--- Vertex Statistics ---
Number of vertices: 1668
Min (x, y, z): [-0.031922,  0.000000, -0.117146]
Max (x, y, z): [ 0.031922,  1.000000,  0.117146]
Mean (x, y, z): [ 0.021694,  0.302795, -0.004365]
Std  (x, y, z): [ 0.011166,  0.236869,  0.046789]
```

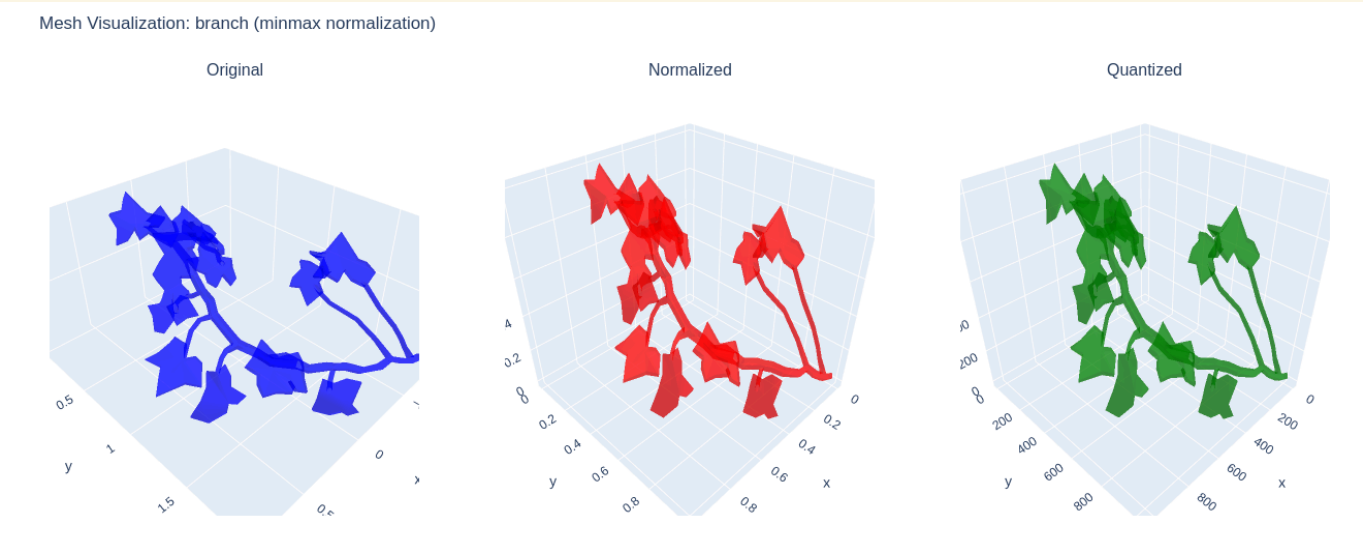
2. 3D interactive visualization using the plotly library of each mesh given in an .obj file is shown below:



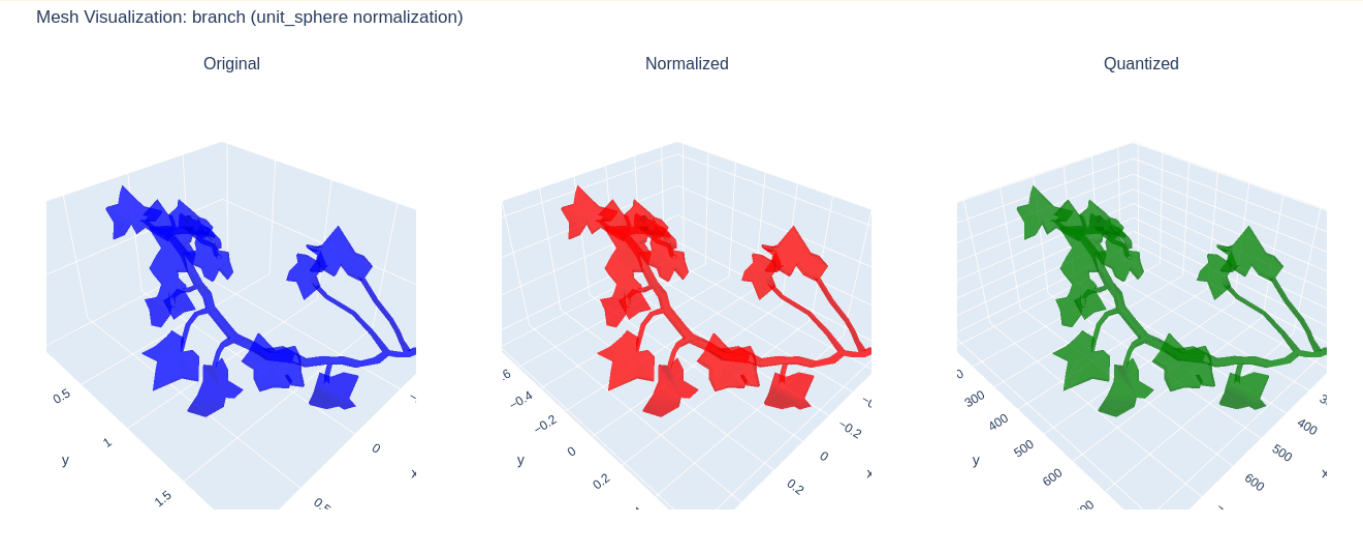
Task 2

I have chosen the min-max and the unit-sphere normalization techniques for this task. For each normalization method, quantization is done using a bin size of 1024.

1. Min-max results for the branch.obj:



2. Unit sphere results for the branch.obj:

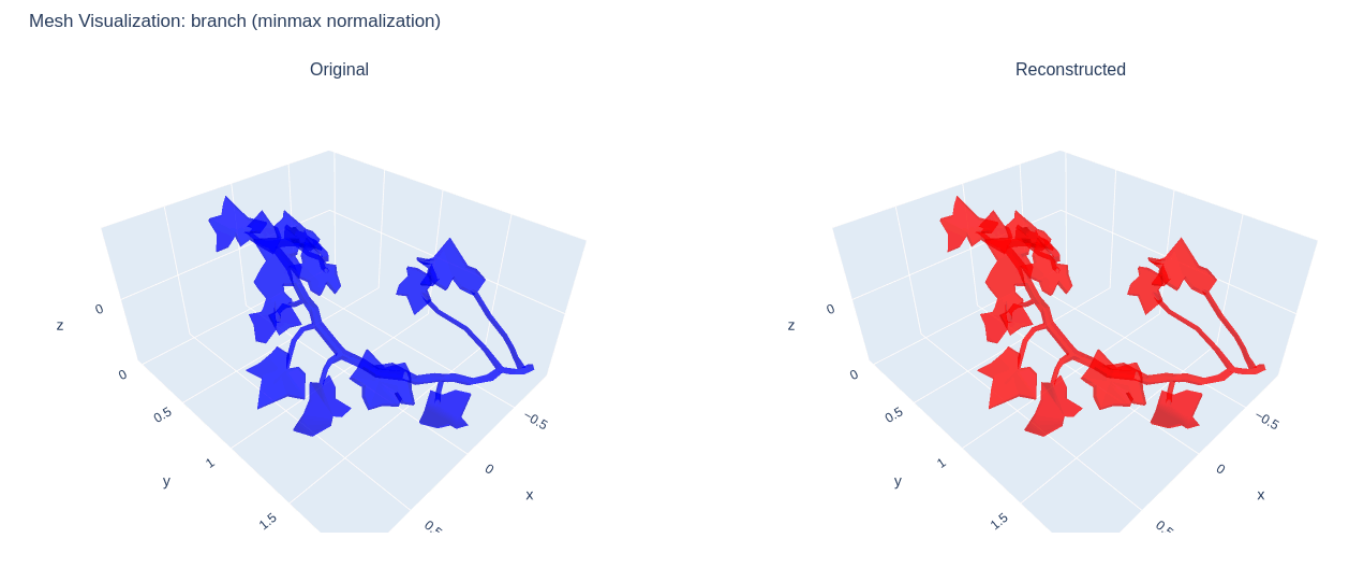


As we can see in the plots for min-max normalization, the scale is not preserved after normalization, while unit-sphere normalization preserves the shape. This is because min−max normalization stretches each axis independently, but unit-sphere normalization scales all axes by the same amount, so the original proportions remain intact. So, the unit-sphere normalization method preserves the mesh structure better.

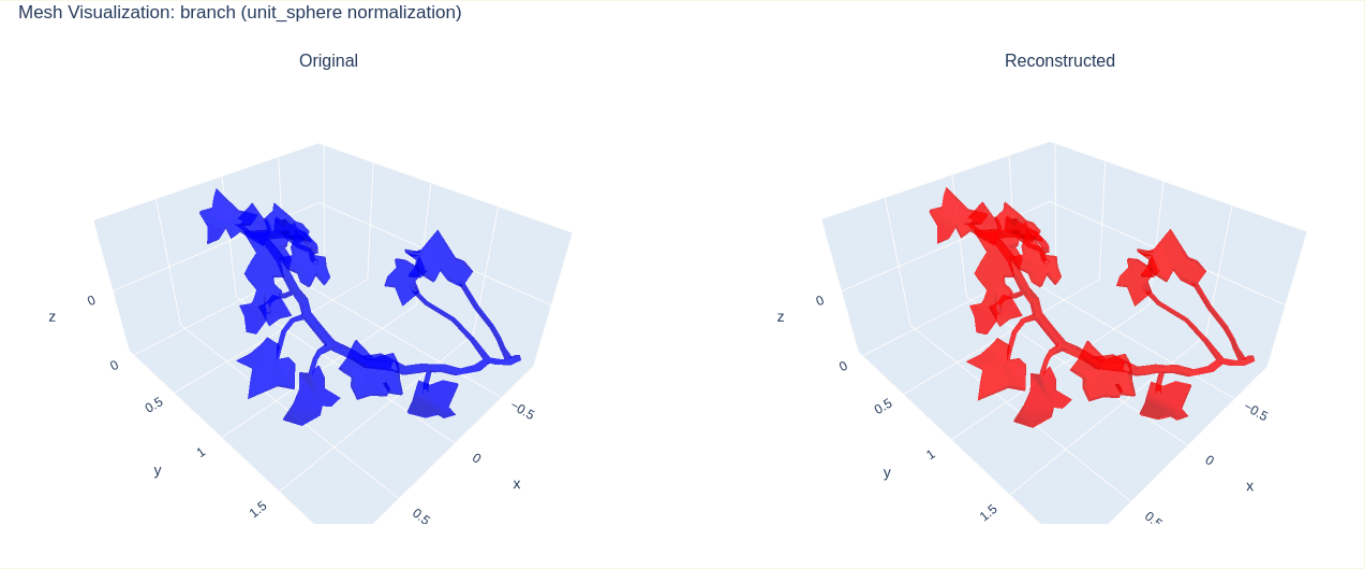
Task 3

To reconstruct the mesh from its normalized and quantized form, I first de-quantized it and then de-normalized the mesh.

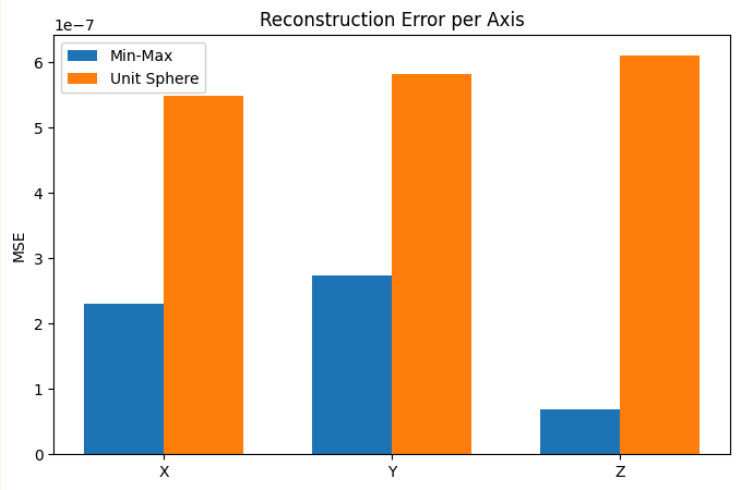
1. Reconstruction of mesh from min-max normalization:



2. Reconstruction of mesh from unit sphere normalization:



3. Plot for reconstruction error metrics (MSE)



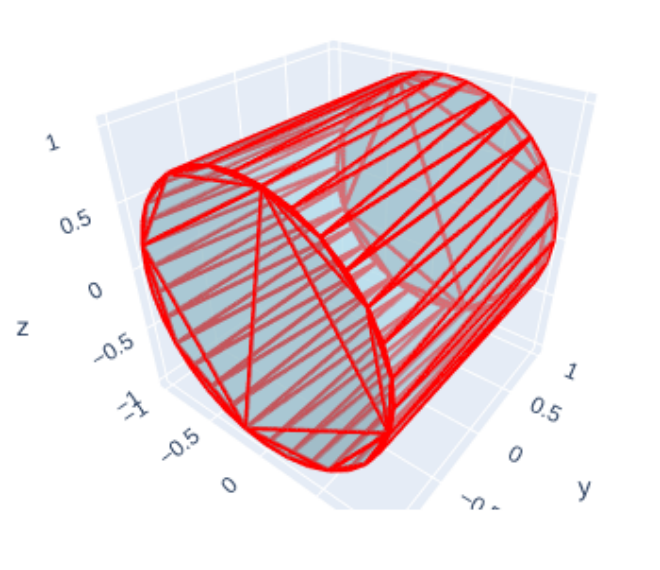
As shown in the above bar graph, min-max normalization combined with 1024-bin quantization gives the least reconstruction error across all axes, with MSE values around 2.2e-7 to 0.68e-7. In contrast, unit-sphere normalization shows consistently higher errors (around 5.4e-7 to 6.1e-7). The pattern suggests that min−max normalization is more quantization-friendly because it spreads the vertex coordinates uniformly within each axis range, allowing finer precision. Unit-sphere normalization preserves the shape better visually, but because all coordinates are scaled together, some axes lose resolution after quantization, resulting in slightly higher MSE. Overall, min−max normalization gives lower numerical error, while unit-sphere normalization gives better geometric consistency.

Bonus task

Option 1: Seam Tokenization Prototype

To identified seam edges, first I identified seam edge vertices, using the property that seam vertices have different UV mappings, so I first grouped by vertices and identified vertices which has atleast 2 different UV mappings, then traversed the unique edges that faces contain, and identified the edges which has both the vertices in the seam vertices, these edges are seam edges.

Red lines shows the seam edges for cylinder.obj:



A seam is basically a vertex pair (v1, v2). We could encode it directly using vertex indices, but if the object is very large, the vertex indices will also be large, which increases the vocab size. The vocab size should remain fixed. Also, vertex indices don't carry any geometric meaning, so encoding based on them doesn't make much sense.

To include geometric meaning, we can represent v1 as (x1, y1, z1) and v2 as (x2, y2, z2), and encode the pair ((x1, y1, z1), (x2, y2, z2)). But x, y, z coordinates can be of arbitrary range, so we need to normalize them. I'll use unit-sphere normalization from the previous task because it preserves geometric structure. I'll also apply quantization using bin size 1024, as done earlier, to keep the vocab size fixed.

To separate different seam segments, I'll add <EDGE> and </EDGE> tokens. In this encoding, each seam has 8 tokens: 6 quantized tokens and 2 boundary tokens. During decoding, I'll simply dequantize and denormalize the sequence based on the <EDGE> and </EDGE> tokens.

Below is the Encoding-decoding example:

1. Token Encoder:

```
Original sequence: [(0.0, -1.0, -1.0, 0.0, 1.0, -1.0), (0.0, -1.0, -1.0, 0.19509, -1.0, -0.980785), (0.0, -1.0, -1.0, 0.19509, 1.0, -0.980785)]
Encoding: ['<EDGE>', 512, 150, 150, 512, 873, 150, '</EDGE>', '<EDGE>', 512, 150, 150, 582, 150, 157, '</EDGE>', '<EDGE>', 512, 150, 150, 582, 873, 157, '</EDGE>']
```

2. Token Decoder:

```
Token sequence: ['<EDGE>', 512, 150, 150, 512, 873, 150, '</EDGE>', '<EDGE>', 512, 150, 150, 582, 150, 157, '</EDGE>', '<EDGE>', 512, 150, 150, 582, 873, 157, '</EDGE>']
Decoded sequence: [(0.001382, -0.999488, -0.999488, 0.001382, 0.999488, -0.999488), (0.001382, -0.999488, -0.999488, 0.194921, -0.999488, -0.980134), (0.001382, -0.999488, -0.999488, 0.194921, 0.999488, -0.980134)]
Original sequence: [(0.0, -1.0, -1.0, 0.0, 1.0, -1.0), (0.0, -1.0, -1.0, 0.19509, -1.0, -0.980785), (0.0, -1.0, -1.0, 0.19509, 1.0, -0.980785)]
```