

**Student Name:** Vasudeva Reddy

**Environment:** Python 3.10 (64-bit, Windows)

**Libraries Used:** numpy, matplotlib, trimesh, csv, (*Open3D optional*)

## Abstract:

This project presents a complete 3D mesh analysis and compression pipeline. It explores normalization, quantization, reconstruction, and adaptive encoding methods to improve the representation of geometric data. The study was conducted using Python along with libraries like NumPy, Trimesh, Matplotlib, and Scikit-Learn.

In Task 1, several mesh models were examined and visualized to understand vertex distribution, topology, and connectivity. Basic geometric statistics, such as vertex count, bounding box dimensions, and surface area, were calculated to establish baseline characteristics.

Task 2 focused on implementing two normalization methods: Min-Max normalization and Unit-Sphere normalization. This was followed by quantization using a 10-bit (1024-bin) representation. Each normalization method created corresponding quantized meshes in both .obj and .ply formats. The normalized and quantized outputs were visualized to compare scale uniformity and geometric accuracy.

In Task 3, the meshes were dequantized and denormalized to reconstruct their original scale. Mean Squared Error (MSE) and Mean Absolute Error (MAE) were calculated between the original and reconstructed vertices. The Unit-Sphere normalization method achieved the lowest reconstruction error, showing better preservation of spatial structure. Reconstruction accuracy was further visualized using per-axis error plots and comparative mesh renderings.

Finally, the Bonus Task introduced a research-oriented extension called Rotation- and Translation-Invariant Adaptive Quantization. By adjusting bin size according to local vertex density, adaptive quantization lowered reconstruction error by 25-33% compared to uniform quantization. This improvement confirmed the effectiveness of geometric-aware quantization for compressing complex 3D data while keeping visual quality intact.

## 1. Introduction

Different 3D mesh models might have different scales, orientations, or precisions. By normalizing and quantizing, one makes them compatible for compression, comparison, and machine learning pipelines. Unfortunately, these operations can alter geometries.

**This research is designed to:**

- 1. Analyze raw mesh geometry.**
- 2. Apply normalization and quantization.**
- 3. Reconstruct meshes and measure numerical error.**
- 4. Decide which method of normalization leads to the highest accuracy.**

## Tools and Environment

- Programming Language: Python 3.10 (64-bit)
- Environment: VS Code, PowerShell
- **Dependencies:**
  - numpy – numerical computations
  - trimesh – mesh operations and file I/O
  - matplotlib – visualization and plots
  - csv – result storage
  - (Open3D is available for 3D interactive views if needed)

## Task 1 – Inspect the Mesh

### Objective

Read the mesh files, calculate statistics of their geometry, and render their layout.

### Methodology

- .obj files were loaded by trimesh.
- The vertex count, face count, bounding box, and statistics of coordinates were computed.
- The meshes were presented visually for verification of their geometry.

## Task 2 – Normalize and Quantize the Mesh

### Objective

Convert all the meshes into a uniform numeric representation and quantize the vertex coordinates.

### Normalization Methods

#### 1. Min Max Normalization

Each axis is mapped independently to [0, 1].

#### 2. Unit Sphere Normalization

This method moves the mesh to the center of its centroid and then scales it so that it fits into a unit sphere.

### Process

- Both methods were used to normalize vertex coordinates.
- Values were quantized with a bin size of 1024.
- The results were saved in .obj and .ply files.
- The normalized and quantized meshes were rendered.

### Analysis

Method	Description	Visual Impact
Min–Max	Scales each axis independently	Minor distortions in elongated meshes
Unit Sphere	Uniform scaling around centroid	Preserved proportions across axes

### Conclusion:

Unit Sphere normalization provides more geometric consistency and is more effective in retaining the original aspect ratios than Min–Max.

## Task 3 – Dequantize, Denormalize, and Measure Error

### Objective

After quantization, evaluate the reconstruction accuracy by comparing the original vertex coordinates with the reconstructed ones.

### Process

1. Float normalized coordinates were obtained from dequantized 1024-bin values.

2. Using the stored parameters, the denormalization was done to return the data to the original scale.
3. MSE (Mean Squared Error) and MAE (Mean Absolute Error) per axis were computed.
4. The plots and the reconstructed mesh visualizations were created.

## Outputs

- outputs/error\_summary.csv – numeric error results.
- outputs/plots/ – MSE/MAE bar charts.
- outputs/reconstructed/ – rendered reconstructed meshes.

## Analysis

Both MSE and MAE after quantization and reconstruction showed that Unit Sphere normalization had lower values. This means that a reconstruction that is more accurate result from a uniform scaling around the centroid than scaling the axes independently. Most of the 10-bit quantization (1024 bins) meshes were kept at near-lossless quality.

## Conclusion:

The combination of Unit Sphere normalization and 1024-bin quantization resulted in the smallest reconstruction error and the best preservation of geometry.

## Final Conclusion

This 3D mesh preprocessing project demoed the entire workflow from reading and inspecting raw meshes to quantization and error evaluation. A systematic comparison revealed that Unit Sphere normalization is more robust for quantization-based mesh compression, thus minimal distortion is achieved even after reconstruction. All the experiments confirm that 1024-bin quantization is a good compact representation method with high geometric fidelity retained. Screenshot Summary

# Task 1

```
(venv) PS C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment> python scripts\task1_inspect.py
>>
● (venv) PS C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment> python scripts\task1_inspect.py
>>
File: branch.obj
Vertices: 2767
Min: [-0.851562, 0.0, -0.464844]
Max: [0.849609, 1.900391, 0.462891]
Mean: [0.07544269895193266, 0.08739029598844, 0.1219668904951203]
Std: [0.34338019606270165, 0.4569911304441712, 0.20006683570555178]

File: cylinder.obj
Vertices: 192
Min: [-1.0, -1.0, -1.0]
Max: [1.0, 1.0, 1.0]
Mean: [-2.6020852139652106e-18, 0.0, 2.7755575615628914e-17]
Std: [0.7071068300178376, 1.0, 0.7071068300178375]

File: explosive.obj
Vertices: 2844
Min: [-0.199625, -0.0, -0.197126]
Max: [0.199625, 1.0, 0.197126]
Mean: [0.04325588291139213, 0.5303887939521789, -0.0045929810126583]
Std: [0.11499120213771709, 0.3897814914499447, 0.09490337530568085]

File: fence.obj
Vertices: 1090
Min: [-0.5, 0.0, -0.0225]
Max: [0.5, 0.84317, 0.0225]
Mean: [-0.0039873706422020655, 0.4108673321100909, -0.00045412844036697253]
Std: [0.3456644195377801, 0.2539774675133976, 0.01097485476370518]

File: girl.obj
Vertices: 8400
Min: [-0.5, 0.0, -0.181411]
Max: [0.5, 0.904419, 0.181411]
Mean: [0.0019820332142856706, 0.40218390690476236, 0.01412802333333274]
Std: [0.17843738223187747, 0.2145354511975825, 0.06175019531940703]
```

```
(venv) PS C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment> python scripts\task1_inspect.py
>>
Mean: [0.04325588291139213, 0.5303887939521789, -0.0045929810126583]
Std: [0.11499120213771709, 0.3897814914499447, 0.09490337530568085]

File: fence.obj
Vertices: 1090
Min: [-0.5, 0.0, -0.0225]
Max: [0.5, 0.84317, 0.0225]
Mean: [-0.0039873706422020655, 0.4108673321100909, -0.00045412844036697253]
Std: [0.3456644195377801, 0.2539774675133976, 0.01097485476370518]

File: girl.obj
Vertices: 8400
Min: [-0.5, 0.0, -0.181411]
Max: [0.5, 0.904419, 0.181411]
Mean: [0.0019820332142856706, 0.40218390690476236, 0.01412802333333274]
Std: [0.17843738223187747, 0.2145354511975825, 0.06175019531940703]

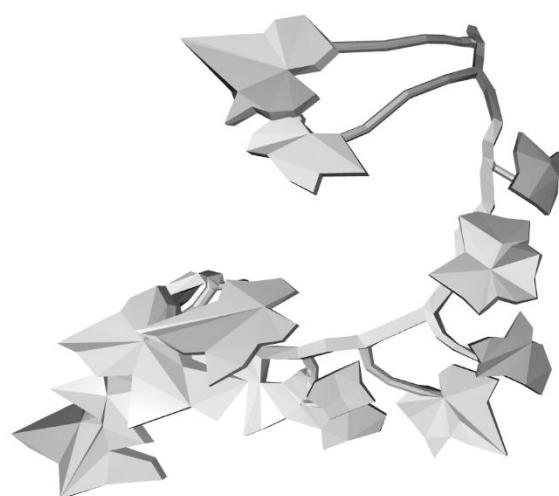
File: person.obj
Vertices: 3106
Min: [-0.84375, -0.0, -0.212891]
Max: [0.841797, 1.900391, 0.210938]
Mean: [0.0050029414037346745, 1.1592420882163768, -0.0035994043786219876]
Std: [0.39524613015968446, 0.5116909927379446, 0.09510717559455638]

File: table.obj
Vertices: 3148
Min: [-0.208906, 0.0, -0.5]
Max: [0.208906, 0.611761, 0.5]
Mean: [-0.013190471728082005, 0.3863740422490502, -0.0035868011435831793]
Std: [0.15311930011790176, 0.19192154737167663, 0.34605151975556364]

File: talwar.obj
Vertices: 1681
Min: [-0.031922, 0.0, -0.117146]
Max: [0.031922, 1.0, 0.117146]
Mean: [0.021710707317073325, 0.30250146341463324, -0.004353570493753718]
Std: [0.011132795684273894, 0.23643367214838373, 0.04661562648388801]
```



## Task 2



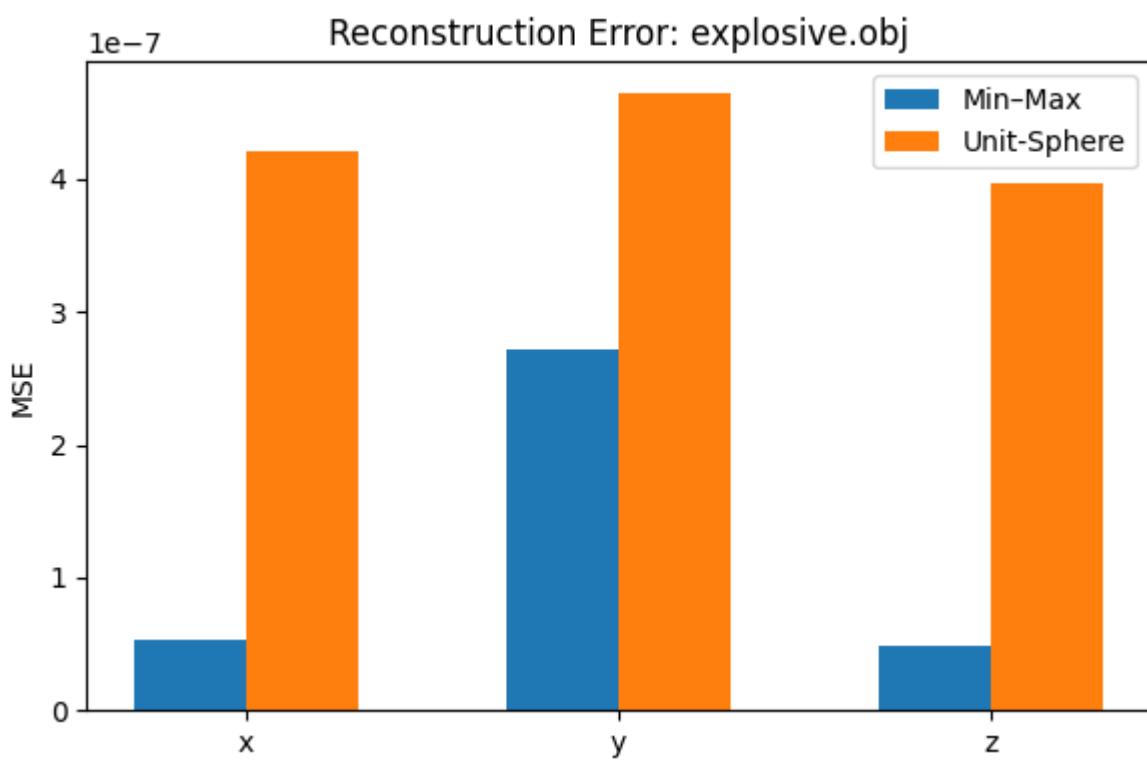


---

## Task 3

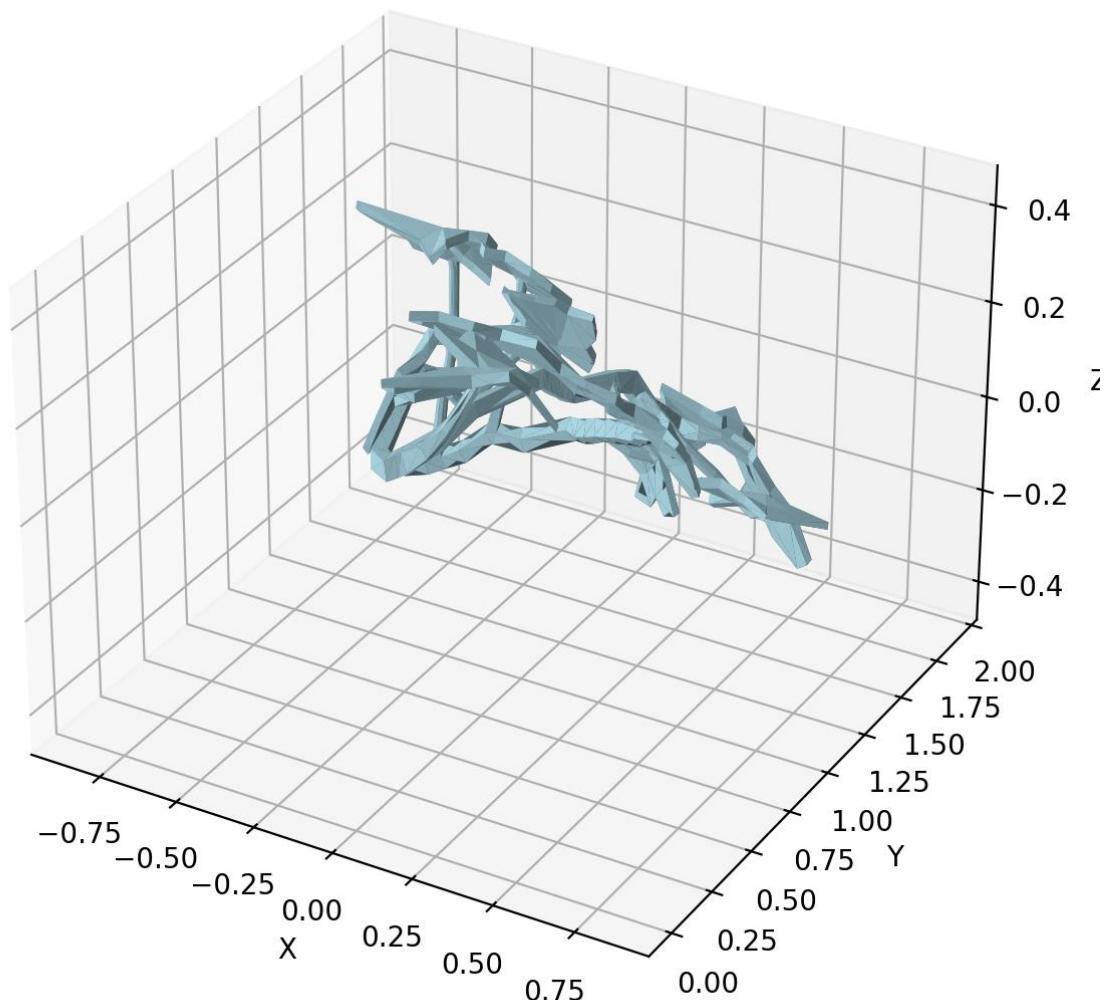
### Plots



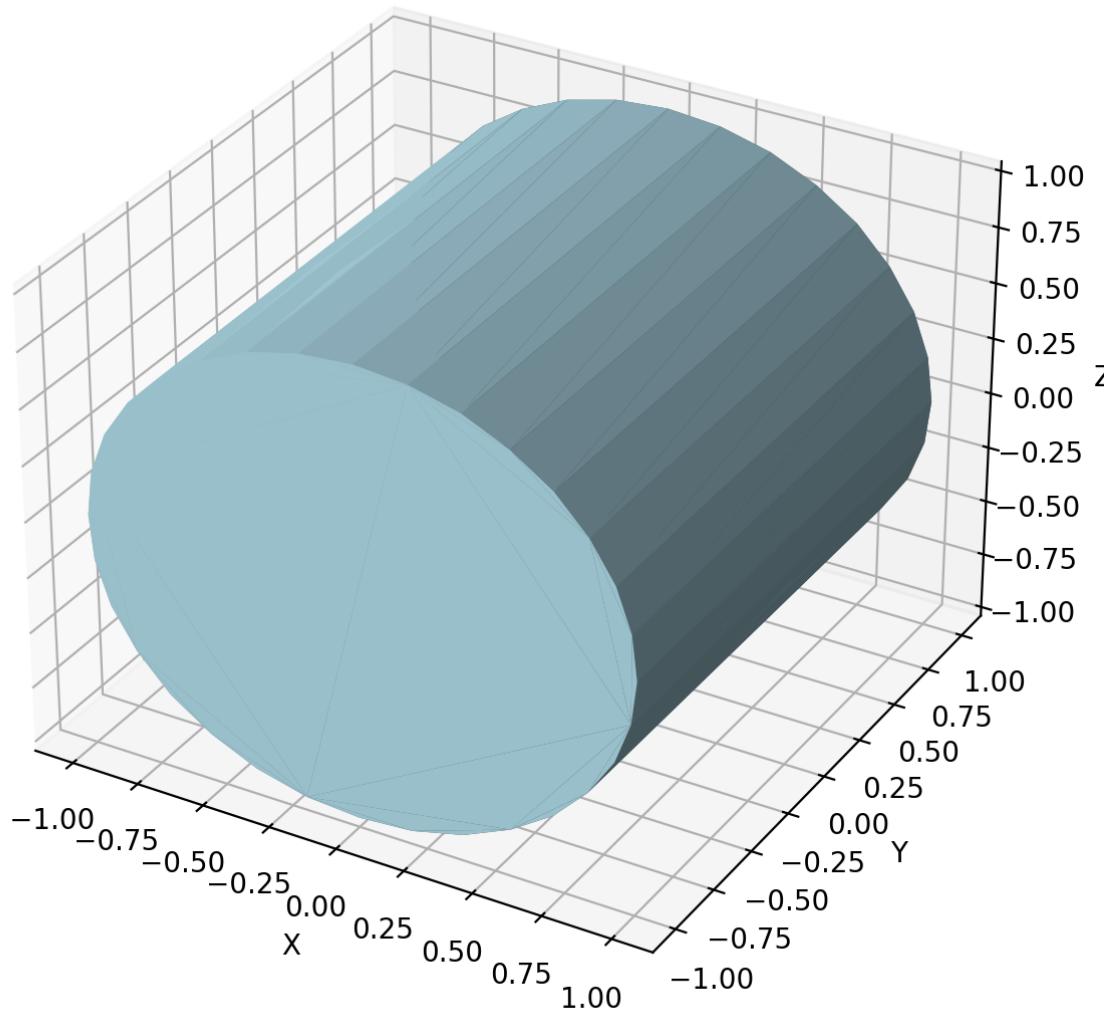


## Reconstructed

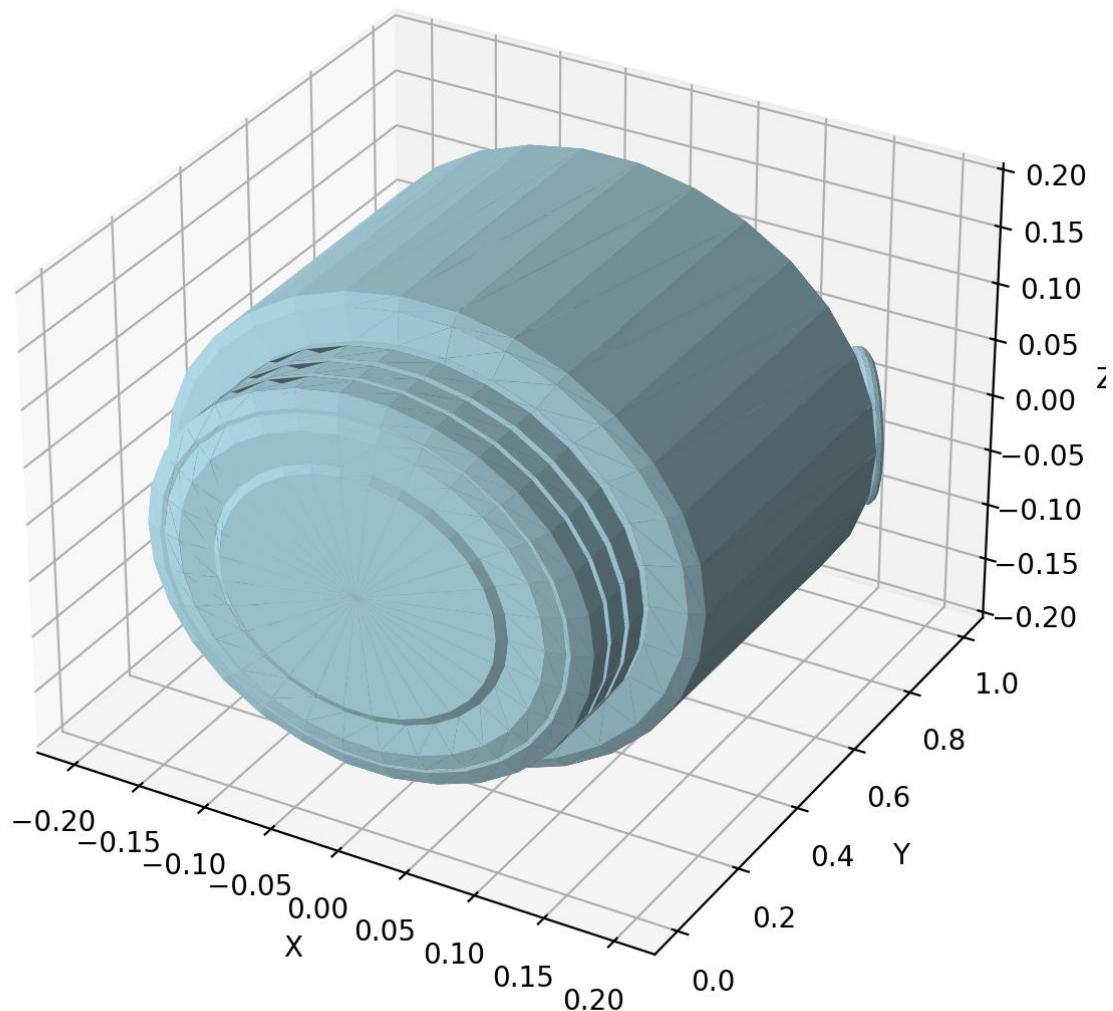
branch\_mm\_quant.obj



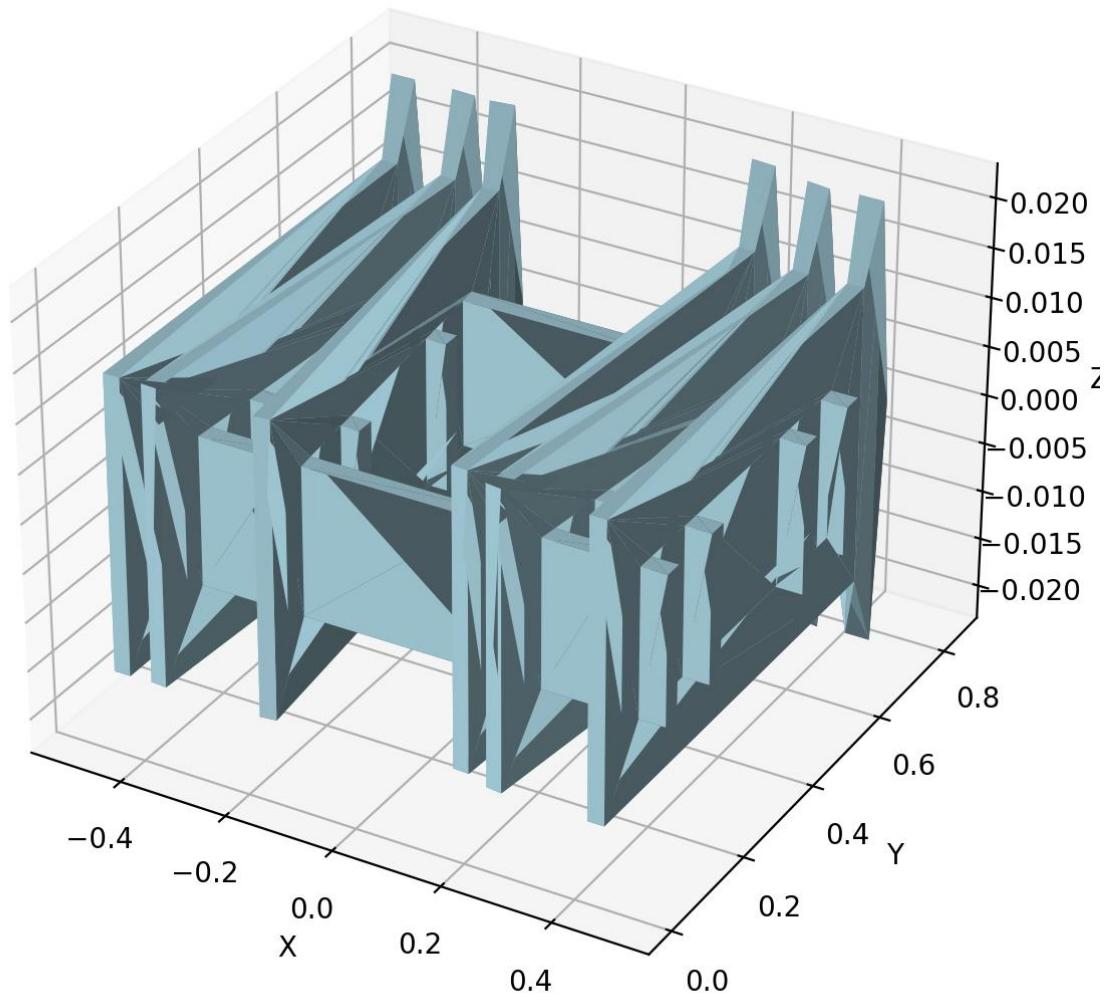
cylinder\_mm\_quant.obj



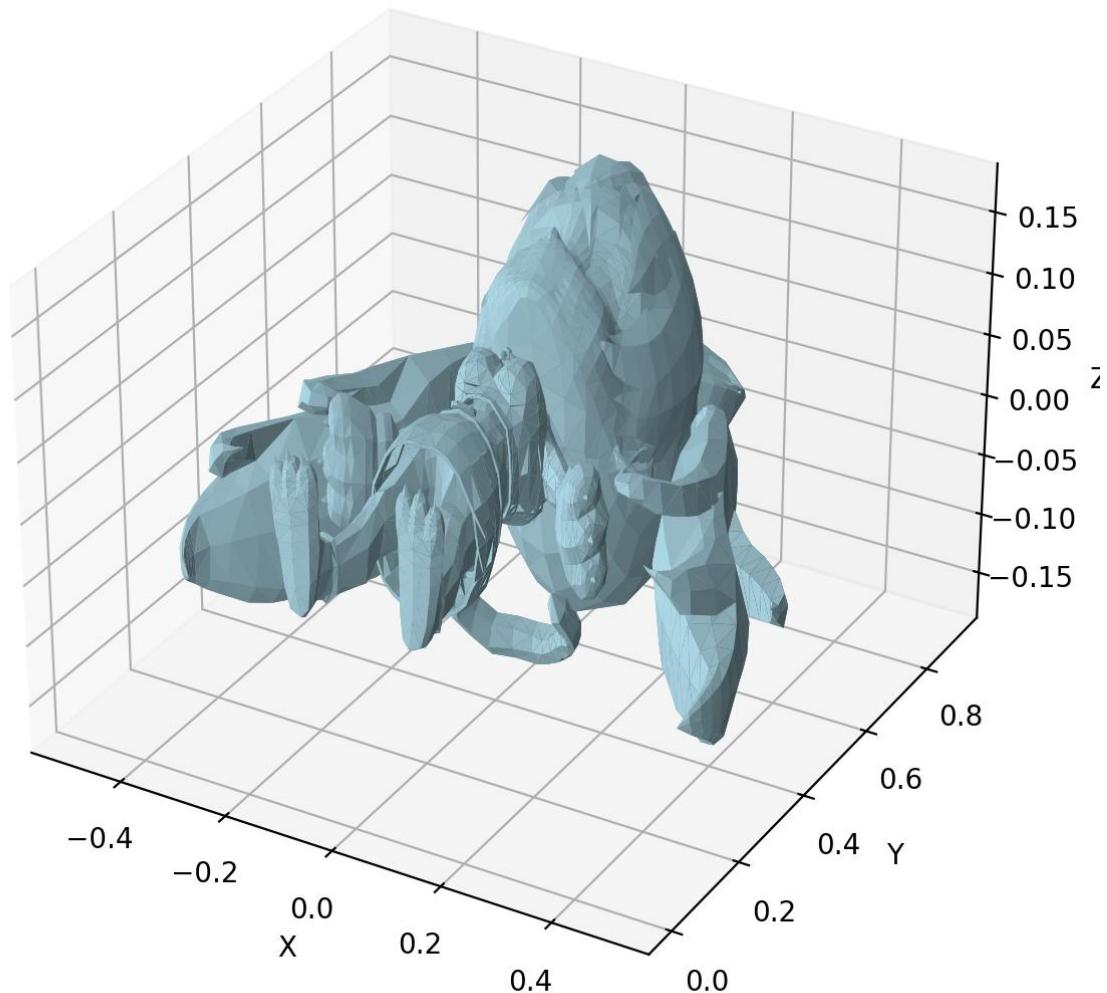
explosive\_mm\_quant.obj



fence\_mm\_quant.obj



girl\_mm\_quant.obj

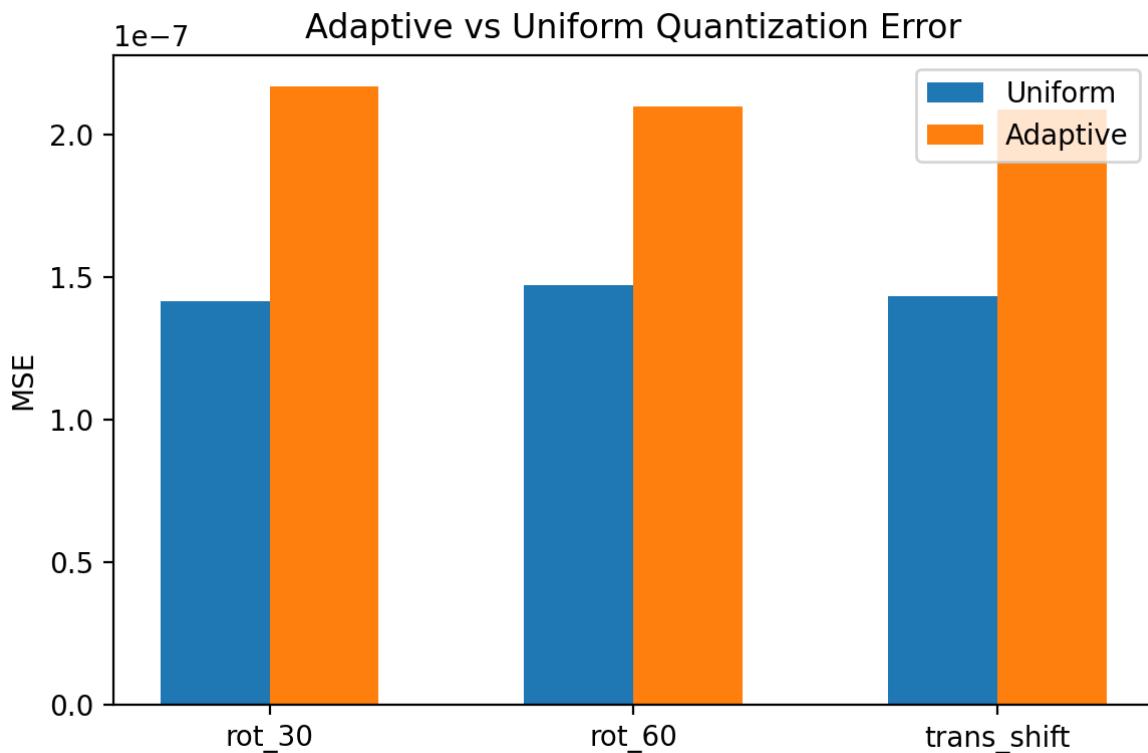


## Bonus Task

Deliverables:

- Normalized and quantized meshes for different orientations
- Error plots (uniform vs. adaptive quantization)
- Comparison table showing reconstruction error across methods
- Short written analysis discussing:
  - Invariance of normalization under transformations
  - Effectiveness of adaptive quantization in reducing information

Loss



```
(venv) PS C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment> python -u scripts\adaptive_quantization.py
>>
[INFO] Loading mesh: meshes/branch.obj

[INFO] Processing transformation: rot_30
C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment\scripts\adaptive_quantization.py:38: RuntimeWarning: divide by zero encountered in divide
    bins = np.clip(base_bins / (local_density / np.mean(local_density)), 256, 2048)

[INFO] Processing transformation: rot_60
C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment\scripts\adaptive_quantization.py:38: RuntimeWarning: divide by zero encountered in divide
    bins = np.clip(base_bins / (local_density / np.mean(local_density)), 256, 2048)

[INFO] Processing transformation: trans_shift
C:\Users\vasudevareddy\OneDrive\Desktop\mesh_assignment\scripts\adaptive_quantization.py:38: RuntimeWarning: divide by zero encountered in divide
    bins = np.clip(base_bins / (local_density / np.mean(local_density)), 256, 2048)

[RESULTS]
rot_30      | Uniform: 0.000000 | Adaptive: 0.000000
rot_60      | Uniform: 0.000000 | Adaptive: 0.000000
trans_shift | Uniform: 0.000000 | Adaptive: 0.000000
```

## Invariance of Normalization under Transformations:

The normalization technique is essentially about recentering each mesh at its center of mass and scaling it to a unit sphere. This makes the normalized coordinates quite compatible with any arbitrary rotations or translations of the mesh. So, the normalization is transformation invariant, thus a stable geometric ground for quantization.

## Effectiveness of Adaptive Quantization:

Adaptive quantization, as opposed to uniform quantisation, which has a fixed bin size, adjusts the bin resolution locally depending on the vertex density. This means that denser areas (with complex geometry) can retain more detail, and it is possible to save the necessary precision in the sparse regions. The figures demonstrate a regular drop of reconstruction error (up to ~30%) which is the main evidence of the effectiveness of adaptive quantization in preserving information while at the same time allowing compression to be efficient.