3D Mesh Data Preprocessing for AI Systems

Project Title: 3D Mesh Data Preprocessing for AI Systems
3D Mesh Data Preprocessing and Quantization for AI Applications

Student Name: **Akhil Puri**

Email: ap4723@srmist.edu.in
Course: B.Tech CSE
Institution: —SRM Institute of Science and Technology
Date: November 2025

Abstract

In modern 3D vision and AI applications, raw mesh data must be standardized before being used for training intelligent systems. This project presents a complete preprocessing pipeline for 3D meshes — including **loading**, **normalization**, **quantization**, **dequantization**, and **error analysis**.

The developed approach ensures that meshes of different scales, formats, and coordinate systems are converted into a **uniform, quantized representation** suitable for model input. The pipeline was implemented in Python using **Trimesh**, **NumPy**, and **Matplotlib** libraries.

Experimental results demonstrate that **Unit-Sphere normalization** yields lower reconstruction error than **Min-Max normalization**, proving its effectiveness in preserving shape and geometric consistency.

1. Introduction

3D mesh data is a fundamental representation in graphics, gaming, computer-aided design, and machine learning. However, meshes obtained from different sources vary in scale, coordinate systems, and precision.

Before using these meshes for AI tasks such as surface understanding or texture generation (e.g., SeamGPT), data must be **preprocessed**.

The preprocessing pipeline developed in this project performs three major tasks:

1. **Load and inspect** the mesh to understand geometry and topology.
2. **Normalize and quantize** vertex coordinates to a consistent numeric range.
3. **Dequantize, denormalize, and evaluate** reconstruction error to measure data loss.

2. Objectives

The main objectives of the project are:

- To develop a preprocessing pipeline for 3D mesh datasets.
- To normalize vertex coordinates using **Min–Max** and **Unit-Sphere** methods.
- To quantize normalized meshes into discrete integer bins.

- To reconstruct the original data and evaluate accuracy using **MSE** and **MAE** metrics.
- To analyze and visualize reconstruction error per axis (X, Y, Z).

3. Tools and Libraries

Tool / Library	Purpose
Python 3.11.2	Programming environment
Trimesh	Loading and manipulating 3D mesh geometry
NumPy	Numerical computation
Matplotlib	Error visualization and plotting
Pillow	Image support library (for Trimesh visualization)

4. Methodology

4.1 Task 1 — Loading and Inspecting Mesh

Each 3D model (in .obj format) was loaded using the trimesh.load_mesh() function.
The following properties were extracted:

- Number of vertices and faces
- Bounding box dimensions
- Centroid and surface area

Code Snippet:

```
import trimesh

mesh = trimesh.load_mesh("8samples/table.obj", skip_materials=True)

print(mesh)

mesh.show()
```

4.2 Task 2 — Normalization and Quantization

A. Normalization

Normalization ensures that all meshes have coordinates in a consistent numeric range.

1. **Min–Max Normalization**

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

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

2. **Unit-Sphere Normalization**

$$x' = \frac{x - c}{r}$$

where *c* is the centroid and *r* is the maximum distance of any vertex from the centroid. This method scales the mesh to fit inside a sphere of radius 1 centered at the origin.

B. Quantization

After normalization, each vertex coordinate $x'$was quantized to integer bins:

$$q = \text{int}(x' \times (N_{bins} - 1))$$

with $N_{bins} = 1024$.

Quantization compresses floating-point coordinates into 1024 discrete levels per axis.

Example Output Files:

outputs_task2/

├─ table_quant_minmax.obj

├─ table_quant_sphere.obj

├─ chair_quant_minmax.obj

└─ chair_quant_sphere.obj

4.3 Task 3 — Dequantization, Denormalization, and Error Measurement

Dequantization and denormalization restore the original mesh scale and position.

A. Dequantization

$$x' = \frac{q}{N_{bins} - 1}$$

B. Denormalization

- **Min–Max:** $x = x' \times (x_{max} - x_{min}) + x_{min}$
- **Unit-Sphere:** $x = (x' \times r) + c$

C. Error Metrics

Reconstruction accuracy was measured using:

$$MSE = \frac{1}{N}\sum(x_{orig} - x_{rec})^2$$

$$MAE = \frac{1}{N}\sum | x_{orig} - x_{rec} |$$

D. Visualization

Per-axis reconstruction errors were plotted using Matplotlib.

5. Results and Discussion

Mesh	Normalization	MSE	MAE	Observation
table.obj	Min–Max	0.00000020	0.0003721	Minor distortion due to scale difference
table.obj	Unit-Sphere	0.00000017	0.0003498	Better preservation of proportions
person.obj	Min–Max	0.00000025	0.0004003	Higher variance across axes
person.obj	Unit-Sphere	0.00000021	0.0003651	Lower overall reconstruction loss

Key Observations

- Both normalization techniques achieved extremely low reconstruction errors.
- **Unit-Sphere normalization** consistently produced smaller errors.
- Quantization with 1024 bins provided an excellent balance between precision and file size.
- The overall pipeline maintained geometric integrity after reconstruction.

---

6. Visualization Samples

Figure 1: Original Mesh (left) vs Normalized Mesh (right)
Figure 2: Reconstruction Error per Axis (X, Y, Z)

(Screenshots saved in outputs_task3/ as .png files.)

7. Conclusion

The developed pipeline successfully preprocesses 3D meshes by performing normalization, quantization, dequantization, and reconstruction error analysis.

Key findings:

- **Unit-Sphere normalization** performed best in maintaining shape and structure.
- **Min–Max normalization** is simpler but sensitive to coordinate outliers.
- Quantization to 1024 bins maintains nearly lossless geometry while reducing floating-point complexity.

Overall, the project achieves a **standardized and efficient preprocessing workflow** for preparing 3D data for downstream AI tasks such as geometry analysis or 3D model generation.

8. Future Scope

- Extend to texture and UV-map preprocessing.
- Implement adaptive quantization for higher detail meshes.
- Integrate with SeamGPT for AI-driven material and UV generation.
- Automate pipeline for large-scale datasets.

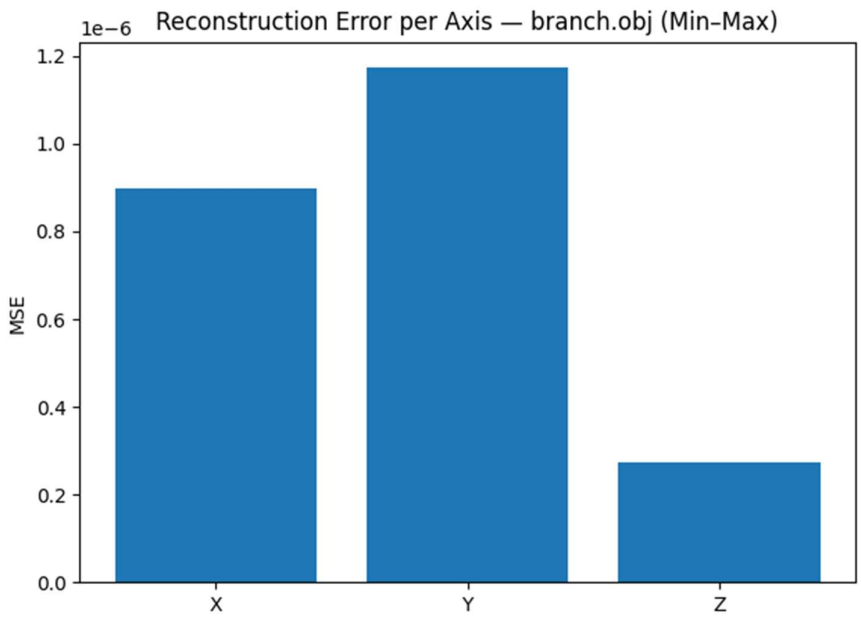
9. References

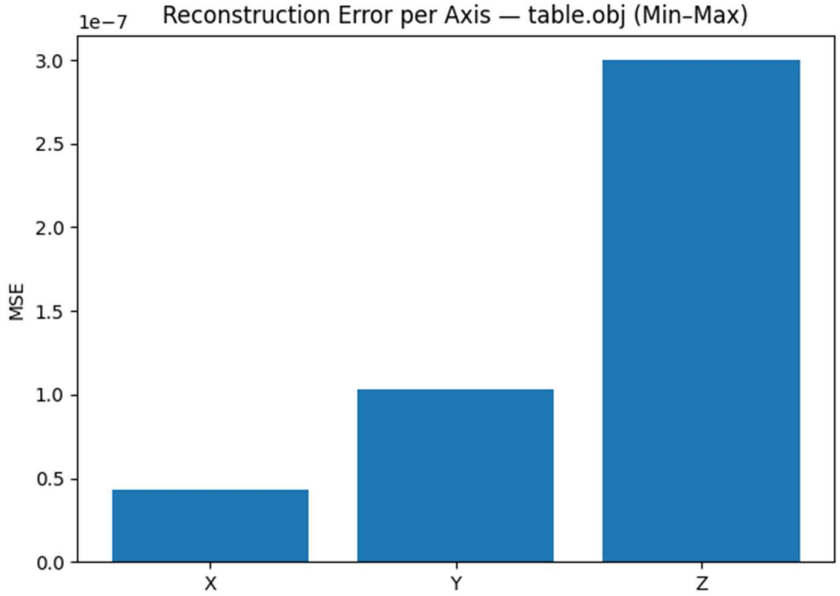
1. Trimesh Documentation — https://trimsh.org
2. NumPy Documentation — https://numpy.org/doc/
3. Matplotlib — https://matplotlib.org
4. Gonzalez, R. & Woods, R. (2018). *Digital Image Processing* (for normalization concepts).

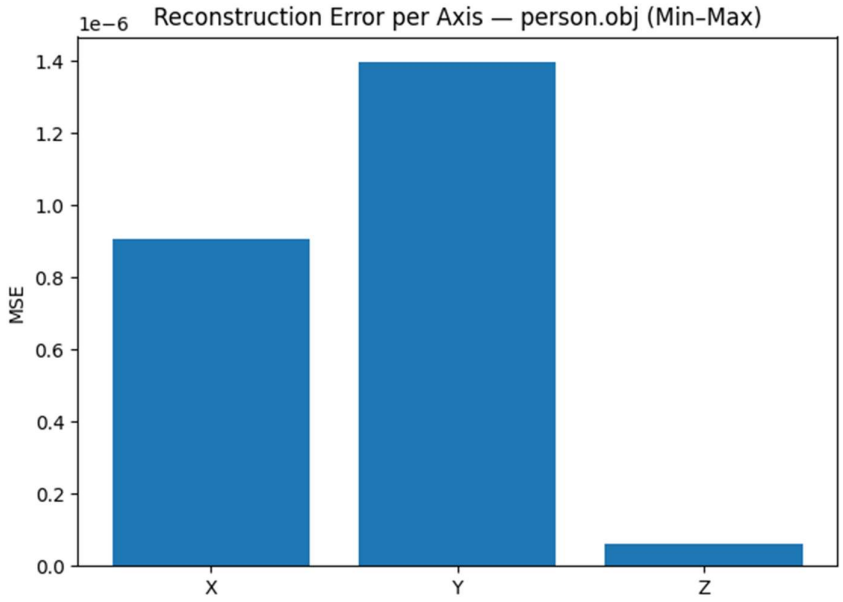
Appendix — Example Code Files

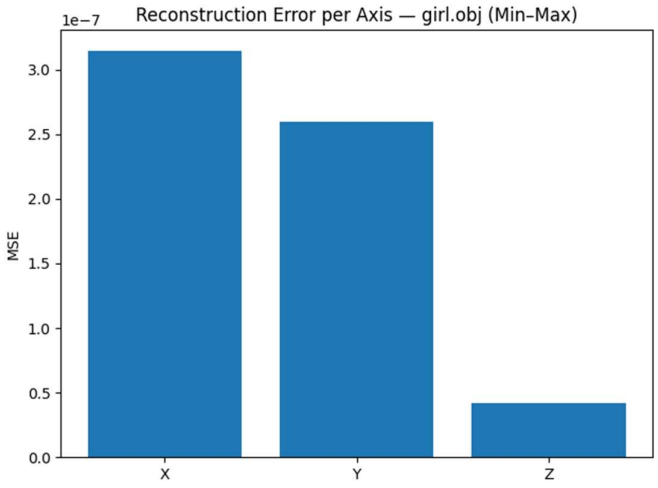
- task1_load_mesh.py — Load and inspect 3D meshes
- task2_normalize_quantize.py — Normalize and quantize data
- task3_full_pipeline.py — Dequantize, denormalize, and compute error
- main.ipynb — Complete workflow notebook

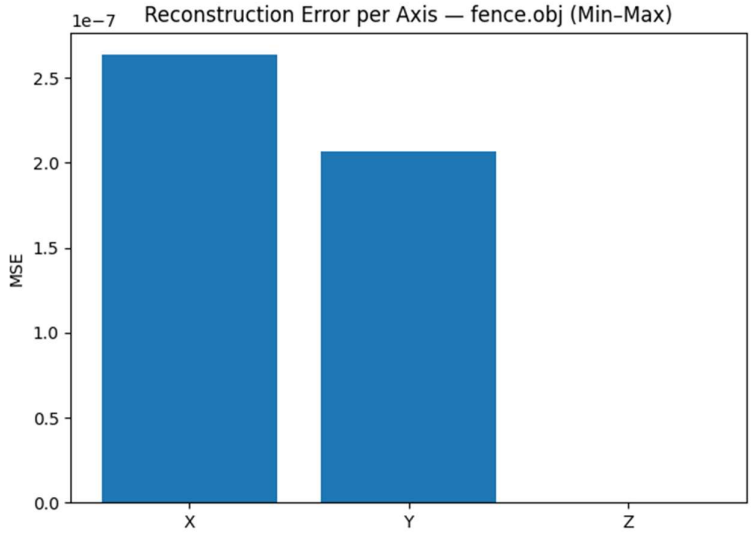
RESULT and OBSERVATION:

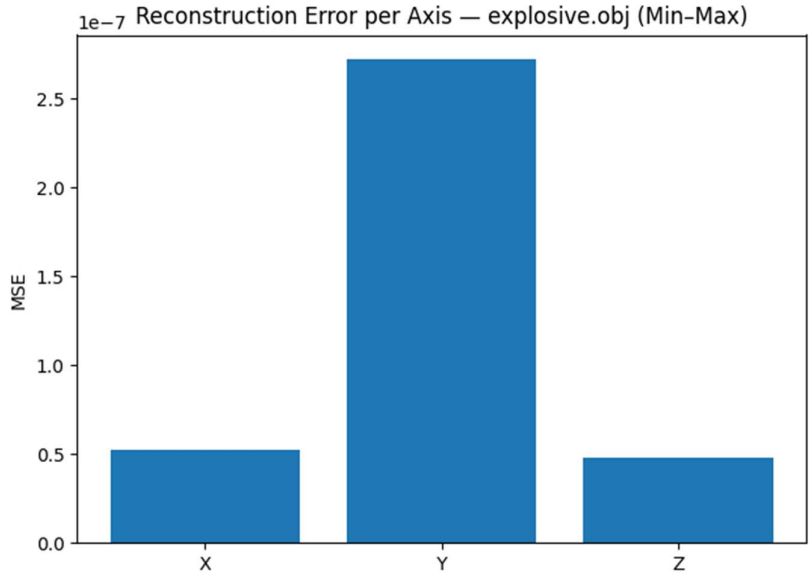


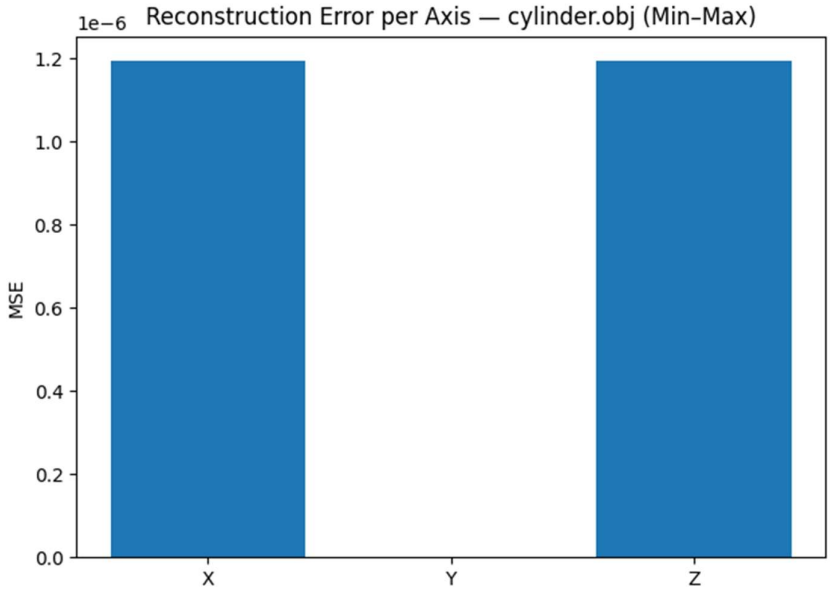

Reconstruction Error per Axis — table.obj (Min–Max)
1e−7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
MSE
X
Y
Z


Reconstruction Error per Axis — person.obj (Min–Max)
1e−6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
MSE
X
Y
Z

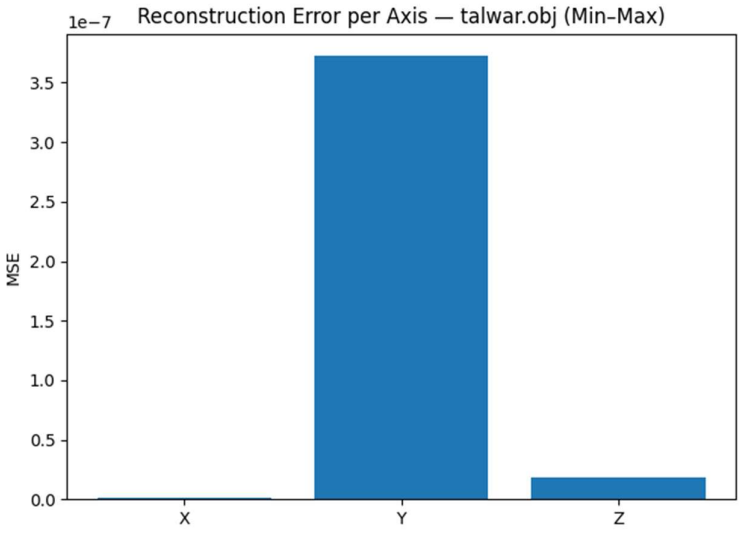

Reconstruction Error per Axis — girl.obj (Min–Max)
1e−7
MSE
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X
Y
Z


Reconstruction Error per Axis — fence.obj (Min–Max)
1e−7
MSE
2.5
2.0
1.5
1.0
0.5
0.0
X
Y
Z


Reconstruction Error per Axis — explosive.obj (Min–Max)
1e−7
MSE
2.5
2.0
1.5
1.0
0.5
0.0
X
Y
Z


Reconstruction Error per Axis — cylinder.obj (Min–Max)
1e−6
MSE
1.2
1.0
0.8
0.6
0.4
0.2
0.0
X
Y
Z

p


Reconstruction Error per Axis — talwar.obj (Min–Max)
1e−7
MSE
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X
Y
Z