Lab Exercise: Implementing a Basic Rendering Pipeline in C#

Objective

The objective of this lab is to build a basic 3D rendering pipeline from scratch in C#. You will implement key components of a rendering pipeline, including object representation, transformations, projection, and shading. By the end of this lab, you should have a functioning 3D renderer capable of rendering simple scenes with basic lighting.

Part 1: Setting up the Renderer

1. Define Vector and Matrix Classes

- Objective: Create classes for Vector3 (3D vector) and Matrix4x4 (4x4 transformation matrix).

- Details: Implement basic vector operations (addition, subtraction, dot/cross products) and matrix operations (multiplication, transformations). Support 4D homogeneous coordinates.

- Expected Results: Unit tests to verify vector and matrix operations; no visual output.

2. Create a Basic Object Class (Mesh)

- Objective: Implement a Mesh class to represent 3D objects using vertices and faces.

- Details: Initialize the mesh with a simple 3D object (e.g., a cube).

- Expected Results: Unit tests to verify correct vertex and face representation; no visual output.

Part 2: Transformations and Projection

1. Implement Transformations

- Objective: Add methods for translation, rotation, and scaling in the Matrix4x4 class.

- Expected Results: Unit tests to verify transformations on sample vectors; no visual output.

2. Define the Camera and Projection

- Objective: Create a Camera class with view and perspective projection matrices.

- Expected Results: Unit tests to verify correct transformation of coordinates to screen space; no

visual output.

Part 3: Rasterization and Shading

1. Rasterize Triangles

- Objective: Implement a rasterizer to convert 3D triangles into 2D pixels.

- Expected Results: Visual output of a rendered 3D mesh (e.g., a cube). Verify triangle filling and screen mapping.

2. Implement a Basic Lighting Model

- Objective: Add Phong shading (ambient, diffuse, and specular) for a single point light source.

- Expected Results: Visual output of a shaded 3D object. Unit tests to validate color calculations.

3. Depth Buffer (Z-Buffer) Implementation

- Objective: Implement Z-buffering to resolve overlapping triangles.

- Expected Results: Visual output showing correct depth rendering in scenes with overlapping objects. Unit tests to verify depth comparisons.

Part 4: Rendering and Display

1. Render to Bitmap

- Objective: Render the scene onto a 2D Bitmap object.

- Expected Results: Visual output of the final rendered scene in a Windows Form application.

2. Add Interactivity

- Objective: Enable user interaction (rotation, zoom, etc.) using keyboard or mouse inputs.

- Expected Results: Visual output showing dynamic transformations in real-time.

Extra Credit (Optional)

1. Texture Mapping

- Add support for texture mapping to 3D objects.

- Expected Results: Visual output showing textured 3D objects.

2. Multiple Light Sources

- Add support for multiple light sources with distance-based attenuation.

- Expected Results: Visual output showing multiple light effects on the scene.

3. Additional Shading Models

- Experiment with Gouraud shading and compare it to Phong shading.

- Expected Results: Visual comparison of shading effects.

Submission Requirements

- Code files for all implemented classes (Vector3, Matrix4x4, Mesh, Camera, Rasterizer, etc.).

- Screenshots of rendered outputs at different stages (e.g., static render, shading, Z-buffer test).

- A short report explaining the implementation and any challenges faced.