CSE 4208: Computer Graphics Laboratory

3D Airport Terminal Simulation with Interactive Features

By,

Maimuna Chowdhury

Roll:1907013

Date of Submission: 22-01-2025

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

Contents

1 Introduction

2 Objective

3 Project Idea

4 Project Overview

5 Methodology

6 Key Features

7 Limitations

8 Discussion

9 Conclusion

10 References

Introduction

Airports are intricate and dynamic spaces that demand meticulous planning and design to ensure functionality, efficiency, and comfort for travelers. 3D simulations offer a powerful way to visualize and comprehend these spaces, enabling a deeper understanding of their layout and operations. This project, "3D Airport Terminal Simulation with Dynamic and Interactive Features," presents a virtual representation of an airport terminal designed to feel both realistic and engaging.

Developed using tools like OpenGL, the simulation incorporates detailed features such as check-in points, lounges, vending machines, flight information displays, and outdoor elements like runways, trees, and a spherical world sculpture. Interactive elements enhance user engagement by allowing camera movement, object transformations, and lighting adjustments, offering a comprehensive exploration of the simulated environment.

Objectives

- Develop a realistic 3D simulation of an airport terminal with detailed facilities like lounges, counters, and runways.
- Incorporate interactive features such as camera navigation, object transformations, and dynamic lighting.
- Demonstrate the potential of 3D technology in architectural visualization and design.
- Provide an engaging and immersive user experience.

Project Idea

The core concept of this project is to create a 3D simulation of an airport terminal using OpenGL, emphasizing realism and interactivity. By utilizing OpenGL's real-time rendering capabilities, the simulation will include detailed representations of airport elements such as check-in counters, lounges, vending machines, flight information displays, runways, and umbrellas. Features like dynamic lighting, camera navigation, and object transformations will allow users to interact with

the virtual environment, making the experience engaging and immersive. This project highlights the potential of OpenGL in developing visually appealing and functional simulations for architectural visualization and design purposes.

Project Overview

This project leverages OpenGL to develop a 3D airport terminal simulation with the following key features:

1. Employing Bezier curves to create smooth and realistic shapes for objects and structures.
2. Designing complex airport components such as check-in areas, lounges, vending machines, runways, and planes.
3. Incorporating animations for moving objects, such as planes and check-in areas within the terminal.
4. Applying detailed textures and dynamic lighting to enhance the visual realism of the environment.
5. Creating fractal-based objects like trees to add intricate visual details and improve environmental aesthetics.

Methodology

1 Keyboard Functionality

The following keyboard functionalities are implemented in the project to enhance interactivity and control:

Camera Movement:

- **W**: Move the camera forward.
- **S**: Move the camera backward.
- **A**: Move the camera left.
- **D**: Move the camera right.

Object Translation:

- **I**: Translate the object along the +Y axis.
- **K**: Translate the object along the -Y axis.
- **L**: Translate the object along the +X axis.
- **J**: Translate the object along the -X axis.
- **O**: Translate the object along the +Z axis.
- **P**: Translate the object along the -Z axis.

Object Scaling:

- **C**: Scale the object along the +X axis.
- **V**: Scale the object along the -X axis.
- **B**: Scale the object along the +Y axis.
- **N**: Scale the object along the -Y axis.
- **M**: Scale the object along the +Z axis.
- **U**: Scale the object along the -Z axis.

Object Rotation:

- **X**: Rotate the object clockwise along the X-axis.
- **Y**: Rotate the object clockwise along the Y-axis.
- **Z**: Rotate the object clockwise along the Z-axis.
- **R**: Rotate the object counterclockwise based on the active axis.

Predefined Camera Positions:

- **0**: Move the camera to a predefined position (Position 1).
- **7**: Move the camera to a predefined position (Position 2).
- **8**: Move the camera to another predefined position (Position 3).
- **9**: Move the camera to a predefined position (Position 4).

Light Control:

- **1**: Toggle point lights (on/off).

Eye Position Adjustments:

- **H**: Adjust eye position along the +X axis.
- **F**: Adjust eye position along the -X axis.

- **T**: Adjust eye position along the +Z axis.
- **G**: Adjust eye position along the -Z axis.
- **Q**: Adjust eye position along the +Y axis.
- **E**: Adjust eye position along the -Y axis.

Movement Toggle:

- **M**: Toggle object movement.

Exit Functionality:

- **ESC**: Close the application window.

Features

Use of geometric primitives for realistic 3D Modeling:

- Use of geometric primitives like cubes, trapezoids, and spheres, cylinder, cone, half sphere for object modeling.

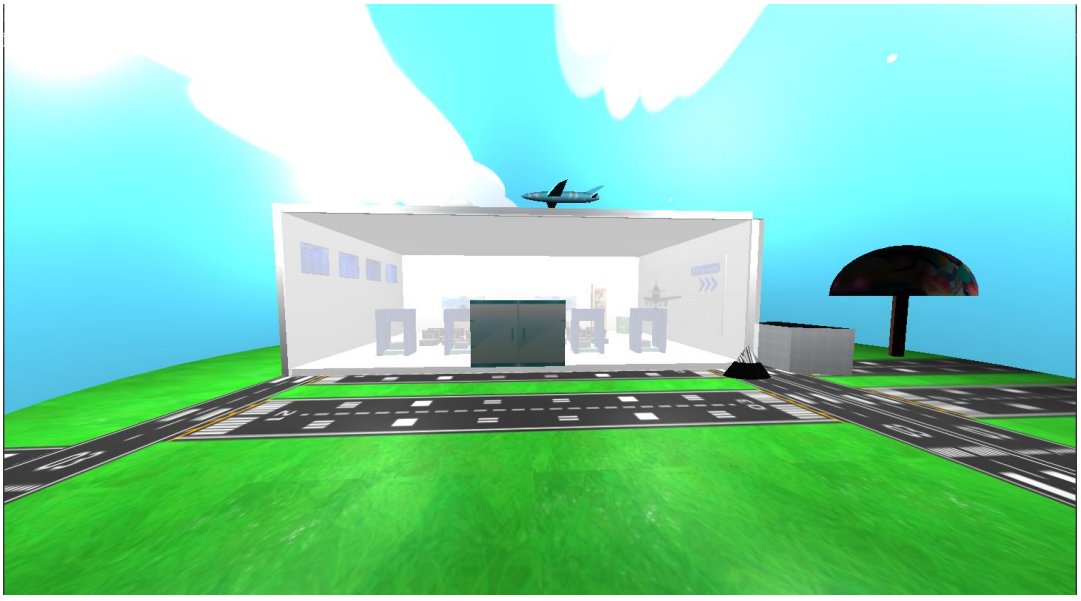

Figure 1: 3D modeling of the airport terminal.

Bezier Curves:

- Bezier curves are implemented for modeling planes and helicopter components.

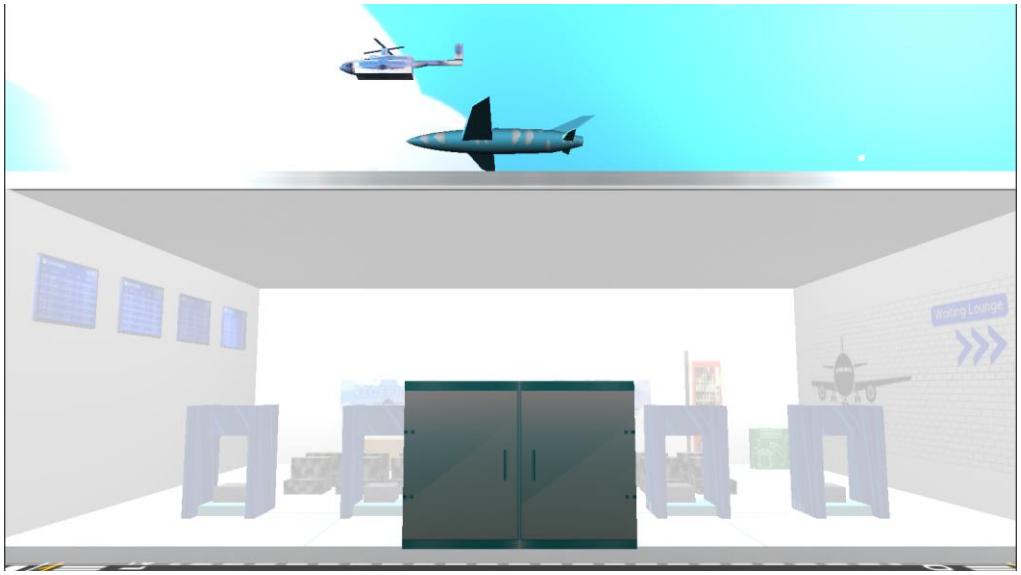

Figure 2: Complex objects by Bezier curves (Plane and Helicopter).

Lighting System:

- Integration of point lights, spotlights, and ambient lighting to enhance visual realism.
- Keyboard toggles to enable/disable lights dynamically.

Textures and Materials:

- Texture is used for creating a real looking environment.
- Option to toggle between textured and material-based rendering.

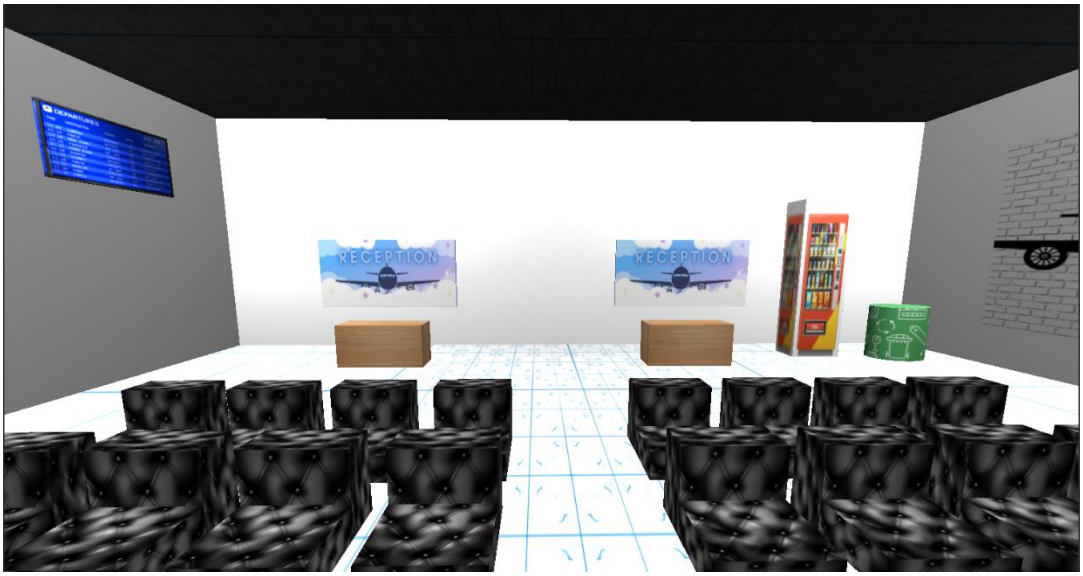

Figure 3: Textured Objects (Cube and Cylinder).

Figure 4: More examples of textured Objects.

Dynamic Object Interaction:

- Moving objects include plane, helicopter, and luggages on the checking points.

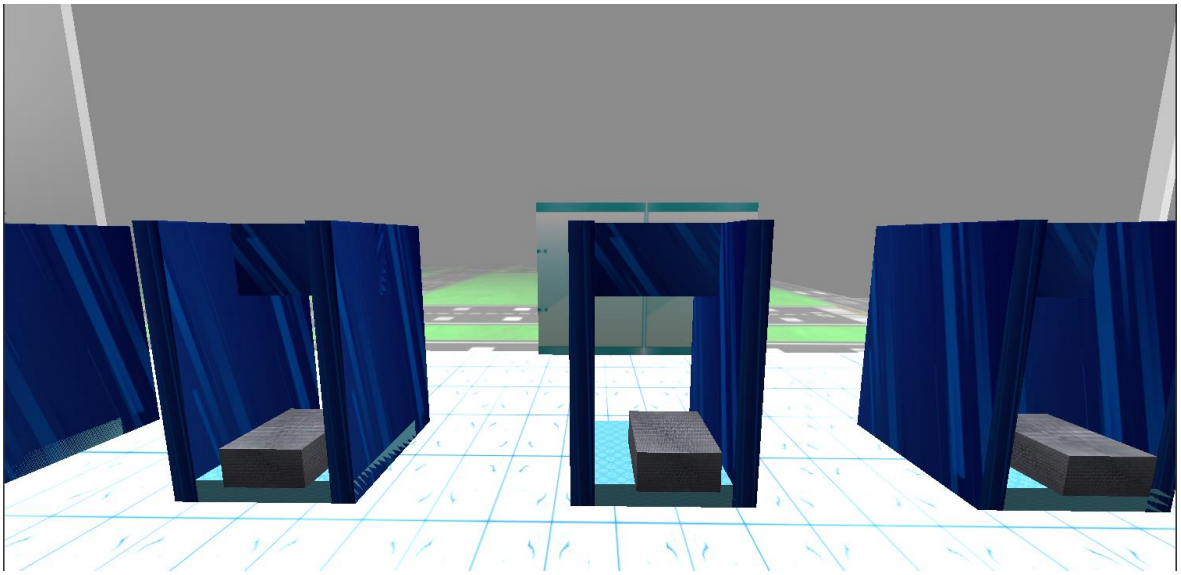

Figure 5: Moving Objects (Luggages on the checking points).

Fractal-Based Objects:

- Creation of fractal trees to enhance environmental detail.
- The provided function implements a recursive algorithm to draw a tree-like structure using fractal principles. The tree is constructed by drawing cubes (basic building blocks) with transformations applied at each recursion level to simulate branching.

Pseudo code:

Function: drawTreeWithFractilesToFile

Input Parameters:

- **cube**: The cube object representing the tree segment.

- **lightingShader**: Shader used to render the cube.
- **alTogether**: Cumulative transformation matrix applied to the entire tree.
- **L, H, W**: Length, height, and width of the current tree segment.
- **N**: Depth of recursion (determines the fractal level).
- **file**: Output file stream to save transformation data.

Steps:

1. Initialize Matrices

- Create identity matrices for model, translate, scale, rotate, and next.
- Set model to the identity matrix.

2. Base Case (N == 0)

- **Set Dimensions**:
  - Initialize length = 0.5, height = 2.0, width = 0.5.
- **Compute Translation Values**:
  - Calculate mvx, mvy, mvz to position the base segment.
- **Apply Transformations**:
  - Set translate using glm::translate with mvx, mvy, and mvz.
  - Combine translate with alTogether to compute next.
  - Apply scaling using glm::scale with length, height, and width.
  - Compute model as alTogether * scale.

- **Save and Render**:
  - Save model, dimensions, and depth to the file using saveTreeFractilesToFile.
  - Render the segment using cube.drawCubeWithTexture(lightingShader, model).
- **Recurse**:

- Call drawTreeWithFractilesToFile with updated parameters (next, length, height, width, N + 1).

3. Recursive Case (0 < N < 5)

- **Adjust Dimensions**:
  - Scale down length, height, and width by 60%.
- **For Each Branch**:
  - Loop through rotation angles -45° and 45°.
  - For each axis (X and Z):
    - Apply scaling using glm::scale with the adjusted dimensions.
    - Apply rotation using glm::rotate with the current angle and axis.
    - Compute model as alTogether * rotate * scale.
    - Save model and dimensions using saveTreeFractilesToFile.
    - Render the segment using cube.drawCubeWithTexture(lightingShader, model).
- **Recurse for New Branches**:
  - Compute new translation values (mvx, mvy, mvz) to position branches.
  - Set translate using glm::translate.
  - Compute next as translate * alTogether.
  - Call drawTreeWithFractilesToFile recursively with updated parameters.

4. Stopping Case (N >= 5)

- **Set Dimensions**:
  - Scale down length, height, and width by 60%.
- **Apply Transformations**:
  - Use glm::scale with the adjusted dimensions.
  - Apply a fixed rotation (-45°) using glm::rotate.
  - Compute model as alTogether * rotate * scale.
- **Save and Render**:

- Save model and dimensions using saveTreeFractilesToFile.
    - Render the segment using cube.drawCubeWithTexture(lightingShader, model).

- Return to the caller when recursion completes.

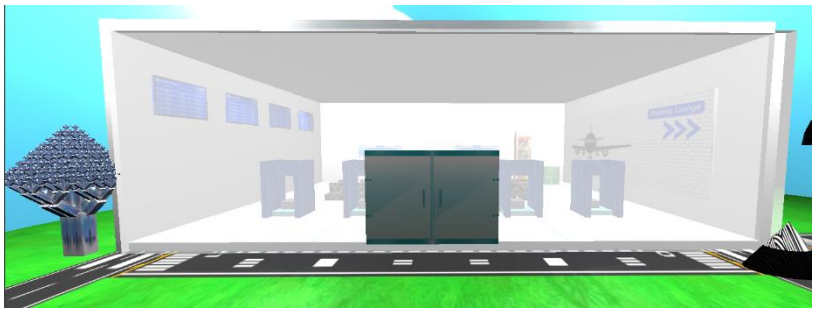

Figure 6: A tree like structure using recursive calls.

Discussion

The 3D airport terminal simulation project represents an innovative application of OpenGL in creating visually immersive and interactive environments. By leveraging the power of OpenGL's real-time rendering capabilities, the project achieves a high level of realism and functionality, demonstrating how advanced graphics techniques can be used in architectural visualization and simulation. As the complexity of the simulation increases, rendering large numbers of objects, including detailed tree fractals and terminal components, can become computationally intensive. Optimizing the rendering pipeline is crucial to maintain real-time performance.

Conclusion

This project highlights the importance of 3D simulations in simplifying the visualization of complex spaces, making them an invaluable tool in fields like architecture, education, and entertainment. While challenges such as performance optimization and limited interactivity remain, the project lays a strong foundation for future enhancements, such as incorporating more detailed animations, real-time events, and VR integration. Overall, this simulation serves as both a technical accomplishment and a practical demonstration of how 3D technology can bridge the gap between conceptual designs and real-world applications, making it a valuable contribution to the field of computer graphics.

References

[1] "Mathematics for 3D Game Programming and Computer Graphics" By: Eric Lengyel

[2] "Computer Graphics: Principles and Practice" By: John F. Hughes, Andries van Dam, Morgan McGuire, et al.