Technical Report on the Implementation of a 3D future city

Module Code: CPT205

Name : Chenrui Zhu

ID: 2361387

Degree Programme: Information and Computer Science

1 Introduction

This project showcases "Future City", an interactive 3D graphics simulation developed using c++ and the FreeGLUT library. The main purpose of this application is to demonstrate the practical application of basic computer graphics principles - including 3D conversion, lighting models, texture mapping and program generation - in a real-time environment.

This project visualizes a future featuring an upper-level floating city system with huge billboard modules, a middle-level rail transportation system, and a lower-level ground forest system. It can also control the day and night changes through a keyboard, greatly transforming the visual atmosphere from a misty and serene daylight environment to a high-contrast, luminous night scene. Users can navigate this procedurally generated world using a third-person aircraft, experiencing a blend of layered modeling animations and physics-based movements. This report provides an overview of the design concept, features and the user instructions for future city.

2 Design concept and features

This section outlines the architectural philosophy of the Future City, detailing the scene composition and the specific graphics techniques implemented using the OpenGL (FreeGLUT) library.

2.1 Overall Design concept

The project visualizes a procedural Cyberpunk metropolis. The design philosophy is centered on modularity and hierarchical modeling. The entire world—from the floating islands to the flying vehicles are constructed mathematically at runtime using OpenGL geometric primitives

*The application employs a State Machine approach to managing environmental transitions. A global state variable (**isNight**) controls the rendering pipeline, dynamically switching lighting parameters, fog density, and geometry styles (solid vs. neon wireframe) to create two distinct visual atmospheres within the same scene graph.*

2.2 Scene Architecture

The environment is divided into several logical subsystems, each responsible for rendering specific visual elements:

2.2.1 Sky Dome & Ground System

*The world boundary is defined by a large semi-sphere constructed using **GL_TRIANGLE_STRIP** loops. A texture map (sky.bmp) is applied to the interior of this dome. The ground is a disk rendered using **GL_TRIANGLE_FAN**, textured with a repeating grass pattern to provide spatial reference.*

2.2.2 Procedural Forest

*Surrounding the city is a generated forest. Trees are placed using polar coordinate randomization (**r * cos(angle)**). To maintain performance while rendering hundreds of trees, they are implemented as Billboards that automatically orient themselves towards the camera.*

2.2.3 Floating City & Buildings

The core city consists of Floating Blocks. Each block acts as a parent node in the hierarchy, containing a platform and multiple child building nodes. The buildings are randomly generated in height and width, creating a unique skyline every session.

2.2.4 Orbital Transport System

The transport system is implemented using Concentric circles. TransportRing acts as the parent, rotating around the city center, while child CargoTrain objects move locally along the ring's circumference.

2.2.5 Billboard System

Holographic advertisements float in the sky. These are rendered as textured quads with additive blending to simulate light projection. They feature a "projector" base and a semi-transparent light beam connecting the base to the screen.

2.2.6 User Vehicles

The user controls a flying vehicle constructed from compound geometric shapes. It features a central body, a transparent outer shell, and particle-like engine trails. The car's position drives the camera logic via linear interpolation.

2.3 Features

2.3.1 Lighting and materials

*To establish a realistic 3D environment, the simulation utilizes the standard OpenGL Phong lighting model and activates it through **glEnable (GL_LIGHTING)**. The visual atmosphere is dynamically controlled through a state machine, in which a specific call to **glLightfv** updates the ambient light, diffused light and mirror light components in real time. This enables the scene to transition seamlessly between the bright diffused lighting during the day and the sharp high-contrast lighting at night. In addition, the material properties vary among objects. Use **glMaterialfv** to assign high mirror brightness **GL_SHININESS** to the player's vehicles and metal structures, simulating the real response of the surface to constantly changing light sources.*

2.3.2 Alpha blending and transparency

*The future city I designed is mainly in the style of transparent materials. **glEnable(GL_BLEND)** is used in different functions depending on the material context. For standard transparency, such as the body of a car, the system uses **glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)** to blend the color of the object with the background. In contrast, the luminous holographic beam combines additive blending (**GL_SRC_ALPHA, GL_ONE**) with selective depth masking (**glDepthMask**), so that the light effect appears bright and semi-transparent without obscuring the geometry behind them.*

2.3.3 Texture mapping

*In my design, bmp texture mapping was mainly adopted for billboards, the ground, skyboxes and trees. By using **glEnable (GL_TEXTURE_2D**), bitmap images were mapped onto geometries to enhance surface details. UV coordinates are assigned to vertices through **glTexCoord2f** to correctly align the image. The system uses **glTexParameteri** to configure texture parameters to handle specific boundary conditions. **GL_REPEAT** is applied to the ground plane to create an infinite grassland, while **GL_CLAMP** is used for the sky dome to prevent seam artifacts at the texture edges and ensure a continuous horizon.*

2.3.4 Forest rendering

*For programmatic Forest, in order to create a three-dimensional visual experience, I use billboard rendering technology to optimize performance. The current model-view matrix is obtained through **glGetFloatv**. The system extracts the "right" and "top" vectors of the camera and constructs a rotation matrix to ensure that the tree quadrilateral is always perpendicular to the observer. And to display the tree as realistically as possible, I chose a bmp file with transparent channels and used **glEnable (GL_ALPHA_TEST)** and **glAlphaFunc** to cut the shape of the tree from the texture. This discards pixels with low alpha values.*

2.3.5 Fog appears

*Use **glEnable (GL_FOG)** to simulate atmospheric depth, blend distant objects into the sky color, and mask the cropped plane in the distance. The system sets the fog mode to **GL_EXP2** (exponential squared) for natural density decline rather than linear cut-off. Use **glFogfv** to dynamically update the fog color to match the background **glClearColor**, ranging from light blue-gray during the day to deep purple at night, thereby creating a cyberpunk atmosphere.*

2.3.6 Wireframe overlay and neon light effects

*The "Cyberpunk" visual style is achieved through multi-channel rendering technology, where objects such as buildings are first drawn as shadow entities and then covered with neon-colored wireframe grids using "glutwireccube". And in the wireframe, use **glEnable (GL_POLYGON_OFFSET_LINE)** and **glPolygonOffset**. This function slightly shifts the depth value of the line towards the camera, effectively eliminating the Z-fighting (flickering) that may occur when the solid surface and the wireframe occupy exactly the same depth.*

2.3.7 Animation technology

*The entire scene is presented through continuous mathematical animations driven by the global time variable (**gTime**). Trigonometric functions like sin () are applied to the vertical positions of urban blocks to create a gentle floating effect.*

3 Instructions

For car movement

W: Move forward (moves in the direction the car faces)

S: Move backward (moves in the opposite direction the car faces)

A: Rotate the car left

D: Rotate the car right

Spacebar: Ascend (fly vertically upwards)

Q: Descend (fly vertically upwards)

For camera

Mouse move: Adjust Camera Pitch (look up / down)

Mouse wheel: Zoom In / out (Adjust distance from the vehicle)

For system

N: Toggle Day / Night cycle (implemented by switching lighting, fog and neon effects)

Esc: Exit the application

4 Screen Shots

Serveral typical screen shots while excution

Figure 1 : The Future City in daytime

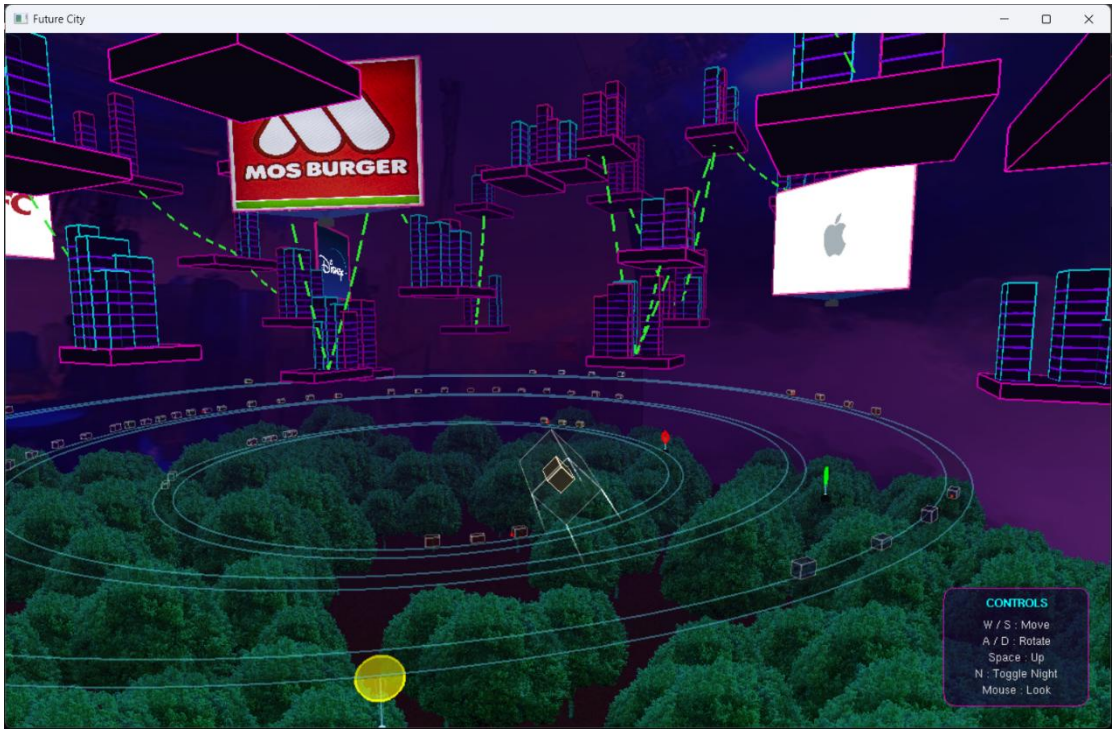

Figure 2 : The Future City in nighttime