Project report on,

BUBT Smart City Simulation

Course Name: Computer Graphics Lab
Course Code: CSE-342

Submitted By,

Name	ID	Intake/Sec
Shahadat Hossain	21225103227	49/5
Nowshin Tabassum	21225103389	49/5
Aysha Siddika Esha	21225103495	49/5
Fairuz Tasnim	21225103460	49/5
Md. Rafi Afsan	21225103228	49/5

Submitted To,
Name : Md. Khairul Islam
Lecturer, Dept. of CSE
Bangladesh University of Business and Technology

Date: 20/05/2025

BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY
Committed to Academic Excellence

Objective

The primary objective of this project is to design and develop a 2D animated city environment using the OpenGL graphics library in the C++ programming language. The project aims to simulate a realistic urban scene that includes essential visual elements such as buildings, roads, vehicles, trees, grass, and environmental details. A major highlight of this simulation is the integration of a moving BUBT-themed bus, symbolizing the university's presence in the virtual city.

This project serves both as a creative application of computer graphics principles and as a practical exercise in working with OpenGL's 2D rendering pipeline. It allows students to understand how complex scenes are broken into layers and drawn using primitive shapes, colors, and transformations. The objective also includes implementing animation techniques that simulate real-world movement, such as buses moving along a road, using mathematical logic and frame updates.

Furthermore, this project demonstrates:

- The use of modular programming techniques to divide graphical components like buildings, vehicles, and trees.
- Application of coordinate-based drawing systems to accurately place and align objects on the screen.
- The integration of text rendering and bitmap fonts to label elements like buses and buildings.

- Developing a structured, maintainable, and scalable graphics project that could be further expanded in the future.

Through this project, team members enhance their technical proficiency in C++ and OpenGL, strengthen their problem-solving abilities, and gain hands-on experience in 2D animation, real-time rendering, and graphics programming—key areas in fields like game development, simulation systems, and interactive visualization.

Tools & Technologies

To build a visually interactive and animated 2D city simulation, a set of specific tools, libraries, and technologies were used. Each tool was selected based on its relevance to graphical programming and real-time rendering. The project required efficient low-level control over drawing primitives, event handling, and scene management, which was made possible through the use of OpenGL and supporting libraries in the C++ environment.

This section outlines the development platform, programming tools, and the header files included in the project, along with clear explanations of why each was essential. These technologies collectively enabled us to create a smooth, realistic, and well-structured city environment complete with buildings, animated buses, trees, roads, and labels.

Programming Language

- C++ : Used for writing the core logic and integrating OpenGL functions for rendering 2D graphical scenes.

Graphics Library

- OpenGL (Open Graphics Library): A cross-platform API used to render 2D and 3D graphics. In this project, OpenGL is used for drawing all visual components such as buildings, roads, buses, and trees.
- GLUT (OpenGL Utility Toolkit) : A utility toolkit for OpenGL that simplifies window creation, handling keyboard/mouse inputs, and managing the display loop.

Development Environment

- Platform: Windows 10/11 (64-bit)
- IDE: CodeBlocks for compiling and debugging OpenGL applications.

Header Files used in the Project:

In Code::Blocks, a header file is used to store function declarations, constant definitions, and reusable code. It usually has a `.h` extension and is included in the main `.cpp` file using `#include`. Header files help keep the code organized and allow functions to be reused across multiple files. They separate the declaration (what something is) from its implementation (how it works).

1. **#include <windows.h>** : Purpose: Required for Windows-specific functions and for initializing the OpenGL context in Windows environments. It helps in managing the display window, timing, and system-level operations like delays or window properties.
2. **#include <stdio.h>** : Purpose: Standard Input/Output header in C/C++. Used in OpenGL applications primarily for debugging purposes (e.g., printing values to the console during development).
3. **#include <GL/glut.h>** : Purpose: Main graphics library for OpenGL. Contains all the functions used to draw shapes, handle buffers, manage keyboard input, and render scenes.
4. **#include <math.h>** : Purpose: Contains mathematical functions such as sin(), cos(), and pi which are essential for drawing circles and curved objects. Used specifically in this project to draw trees, wheels, and clouds using trigonometric calculations.
5. **#define pi 3.142857** : Purpose: Defines the value of π for use in drawing circular/elliptical shapes. Used in the circle() function to calculate angles when drawing shapes using sine and cosine.
6. **#include<iostream>** : This is essential for input and output operations. It allows the program to use standard input and output streams like `std::cin` and `std::cout`.

Team Roles and Responsibilities

To successfully complete a graphics project consisting of over 4000 lines of code, it was essential to divide responsibilities across the team efficiently. Each team member was assigned a specific area of the project based on their strengths and interests. The goal was to ensure high-quality development while promoting collaboration, specialization, and a smoother debugging and testing process.

This report outlines the detailed breakdown of team roles and responsibilities. Each person handled a unique part of the simulation from initial setup and geometric logic to building rendering, environmental design, animations, and final integration. This division allowed us to manage complexity and create a dynamic, functional, and visually engaging city simulation using OpenGL.

Member 1: Nowshin Tabassum(Core Setup and Geometry Functions):

- Responsibilities:
    - Included required libraries and platform-specific headers.
    - Wrote the myInit() function for background setup and 2D orthographic projection.
    - Developed the circle() function using trigonometric math for dynamic object rendering like wheels, trees, and clouds.
- Code Coverage: Lines 1–100
- Skills: OpenGL initialization, reusable graphics logic, 2D coordinate setup.

Member 2: Ayesha Siddika Esha (Environment Design : Sky, Road, Grass, and Trees)

- Responsibilities:
    - Created sky and background using GL_QUADS with color gradients.
    - Designed a detailed road system with white stripes and layering.
    - Implemented realistic trees using a combination of quads and triangles.
    - Added grass, dividers, and nature elements for aesthetic realism.
- Code Coverage: Lines 101–600, Trees and Decor ~800–1300
- Skills: GL primitives, aesthetic design, nature modeling.

Member 3: Fairuz Tasnim (Complex Building Architecture and Scene Layout)

- Responsibilities:
    - Designed multiple buildings:
        - BUBT Main Building
        - Building Two
        - Building Three (formerly Hospital)
        - Additional buildings with multi-floor, multicolor structures
    - Positioned elements symmetrically and used glRasterPos() for text.
- Code Coverage: ~600–1500
- Skills: 2D building design, OpenGL quads, text labeling, structural balance.

Member 4 : Shahadat Hossain (Animation and BUBT Bus Design)

- Responsibilities:
  - Created animation logic using time/position variables (j, k, etc.).
  - Designed BUBT-themed bus in red and white to match the campus brand.
  - Added realistic bus features: doors, windows, front glass, and wheels.
  - Displayed animated motion across the road using variable updates.
- Code Coverage: ~1300–2000
- Skills: Animation logic, character rendering, bus and vehicle modeling.

Member 5 : Rafi Afsan (Additional Buses, Final Touches, and Main Execution)

- Responsibilities:
  - Added second bus (blue) and managed dual object movement.
  - Included light poles, clouds, banners, signs, and finishing elements.
  - Controlled main() function to initialize and execute the OpenGL scene.
  - Ensured integration and polish across all components.
- Code Coverage: ~2000–4000+ (including final animations and render logic)
- Skills: Scene integration, visual enhancements, main loop logic, debug optimization.

Flow of Execution

The flow of execution describes how the program runs, from initialization to final rendering and animation updates. Below is a step-by-step explanation of how this city simulation project is structured and executed using OpenGL and C++:

1. Program Startup

- The program begins with the main() function.
- glutInit() initializes the GLUT (OpenGL Utility Toolkit) environment.
- glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB) sets the display mode (single buffer, RGB color).
- glutInitWindowSize() and glutCreateWindow() configure the OpenGL window's size and title.
- myInit() is called to initialize the background color and the orthographic projection using glOrtho().

2. Coordinate System Setup

- The myInit() function defines a 2D coordinate system with:
  glOrtho(0, 700, 0, 800, -10.0, 10.0);
- This sets up a screen where (0, 0) is the bottom-left corner and (700, 800) is the top-right.
- Background color is set using glClearColor().

3. Display Function Call

- glutDisplayFunc(display) links the rendering logic to the display() function.
- glutMainLoop() enters an infinite loop that listens for window events and keeps refreshing the display.

4. Rendering the Scene: display()

The display() function is the core of this project. It is responsible for drawing all objects in the scene.

a. Background and Environment

- The sky and ground are drawn first using colored rectangles (GL_QUADS).
- Grass and side areas are rendered with green shades.

- Roads and their lane markings (white lines) are created using horizontal lines and rectangles.

b. Natural Elements

- Trees are rendered using the circle() and GL_TRIANGLE_FAN for leaves and GL_QUADS for trunks.
- Clouds are formed by grouping overlapping circles in the sky area.

c. Buildings

- Multiple buildings are constructed using GL_QUADS, each with different shapes, sizes, and colors:
    - BUBT Main Building — contains text labels.
    - Building Two — medium structure near the road.
    - Building Three — renamed from "Hospital", a smaller building.
- Buildings are made with floors, windows, and roofs by stacking shapes.

d. Buses and Animation Logic

- Buses are created using rectangles and circles (for wheels).
- The first bus (BUBT Bus) uses red and white colors.
- The second bus is blue.

Animation Logic:
if (j <= 800)

j += 0.3;

else

- j = -250;
    - This continuously updates the bus's horizontal position.
    - As the bus reaches the end of the screen, it resets to simulate looping movement.

e. Text Labels
Bus labels and building names are drawn using:
glRasterPos2i();

- glutBitmapCharacter();

f. Decorative Objects

- Additional objects like streetlights, banners, and poles are drawn using quads and triangles for enhanced realism.

5. Animation Handling

- Movement is achieved by updating position variables (j, k, etc.).
- These updates are triggered automatically as OpenGL redraws the scene continuously via the display loop.

6. Program Loop

- The program remains in glutMainLoop() until the user closes the window.
- OpenGL automatically calls display() repeatedly, allowing animation to occur smoothly and continuously.

Colors used in this project:

Colors are used in this OpenGL project to visually differentiate between various objects like buildings, trees, sky, roads, and decorative elements. This makes the scene more realistic, attractive, and easy to understand. Colors are applied using the `glColor3ub(r, g, b)` function, where RGB values range from 0 to 255. Before drawing any shape, a specific color is set to define how that shape will appear. Different color shades are used to create depth, highlights, and textures.

Buildings Main Colors:

1. Light Gray (Building Structure): glColor3ub(204, 204, 204)
2. White/Off-white (Building Walls): glColor3ub(242, 242, 242)
3. Reddish Brown (Building Accents): glColor3ub(177, 124, 119)
4. Teal/Blue-Gray (Glass Windows): glColor3ub(85, 119, 119)
5. Dark Blue (Building Partitions): glColor3ub(0, 26, 51)
6. Medium Gray (Floor Dividers): glColor3ub(179, 179, 179)
7. Dark Brown (Door Areas): glColor3ub(123, 88, 71)
8. Pure White (Details): glColor3ub(255, 255, 255)
9. Light Teal (Building Parts): glColor3ub(143, 175, 175)
10. Navy Blue (Doors): glColor3ub(0, 0, 77)

Food Court Colors:

11. Dark Gray-Blue (Food Court Main): glColor3ub(52, 67, 74)
12. Medium Gray-Blue (Food Court): glColor3ub(91, 102, 108)
13. Dark Blue-Gray (Food Court Details): glColor3ub(27, 45, 54)
14. Cream (Food Court Interior): glColor3ub(254, 246, 223)
15. Tan (Food Court Board): glColor3ub(182, 138, 94)
16. Light Tan (Food Court Lines): glColor3ub(250, 201, 143)

Trees & Nature Colors:

17. Brown (Tree Trunks): glColor3ub(75, 35, 5)
18. Olive Green (Tree Leaves): glColor3ub(139, 146, 22)
19. Dark Green (Tree Leaves): glColor3ub(0, 102, 0)
20. Medium Green (Tree Leaves): glColor3ub(0, 99, 47)
21. Reddish Brown (Tree Details): glColor3ub(181, 106, 76)
22. Ground Green: glColor3ub(90, 147, 48)

Sky & Background:

23. Yellow (Sky Bottom): glColor3ub(255, 255, 147)
24. Light Blue (Sky Top): glColor3ub(102, 204, 255)
25. Orange-Yellow (Sun): glColor3ub(253, 183, 77)
26. Light Blue (Clouds): glColor3ub(232, 241, 255)
27. White (Clouds): glColor3ub(252, 254, 255)
28. Light Blue-Gray (Clouds): glColor3ub(221, 229, 247)

Tower & Structures:

29. Gray (Tower Main): glColor3ub(140, 140, 140)
30. Dark Gray (Tower Details): glColor3ub(102, 102, 102)
31. Light Gray (Tower Highlights): glColor3ub(217, 217, 217)

Road & Infrastructure:

32. Dark Gray (Road): glColor3f(0.2, 0.2, 0.2)
33. White (Road Lines): glColor3f(1.0, 1.0, 1.0)
34. Dark Gray-Blue (Lamppost): glColor3ub(83, 131, 131)

Bus & Vehicles:

35. Red (First Bus): glColor3ub(255, 81, 76)
36. Dark Blue (Second Bus): glColor3ub(43, 58, 139)
37. Very Dark Green (Bus Glass): glColor3ub(26, 26, 0)
38. Light Blue (Bus Windows): glColor3ub(230, 255, 255)
39. Black (Bus Wheels): glColor3ub(0, 0, 0)
40. Light Yellow (Bus Wheel Highlights): glColor3ub(255, 255, 204)

Decorative Elements:

41. Pink (Flower Details): glColor3ub(227, 91, 137)

Text Colors:

42. Blue (Text): glColor3ub(0, 51, 204)
43. Gray (Text): glColor3ub(115, 115, 115)
44. Green (Text): glColor3ub(0, 102, 34)
45. Red (Text): glColor3ub(204, 0, 0)
46. Dark Gray (Text): glColor3ub(64, 64, 64)
47. Dark Blue-Gray (Text): glColor3ub(31, 46, 53)

Usage of Functions:

Functions are reusable blocks of code that perform specific tasks. In this project, they are used to organize the drawing process and avoid repeating the same code. A user-defined function is created by the programmer to perform custom tasks (e.g., `circle()` and `myInit()`). A library function or default function comes from built-in libraries like OpenGL or GLUT (e.g., `glBegin()`, `glColor3ub()`, `glVertex2f()`). These built-in functions help handle graphics rendering tasks. Using both types of functions makes the project well-structured and efficient.

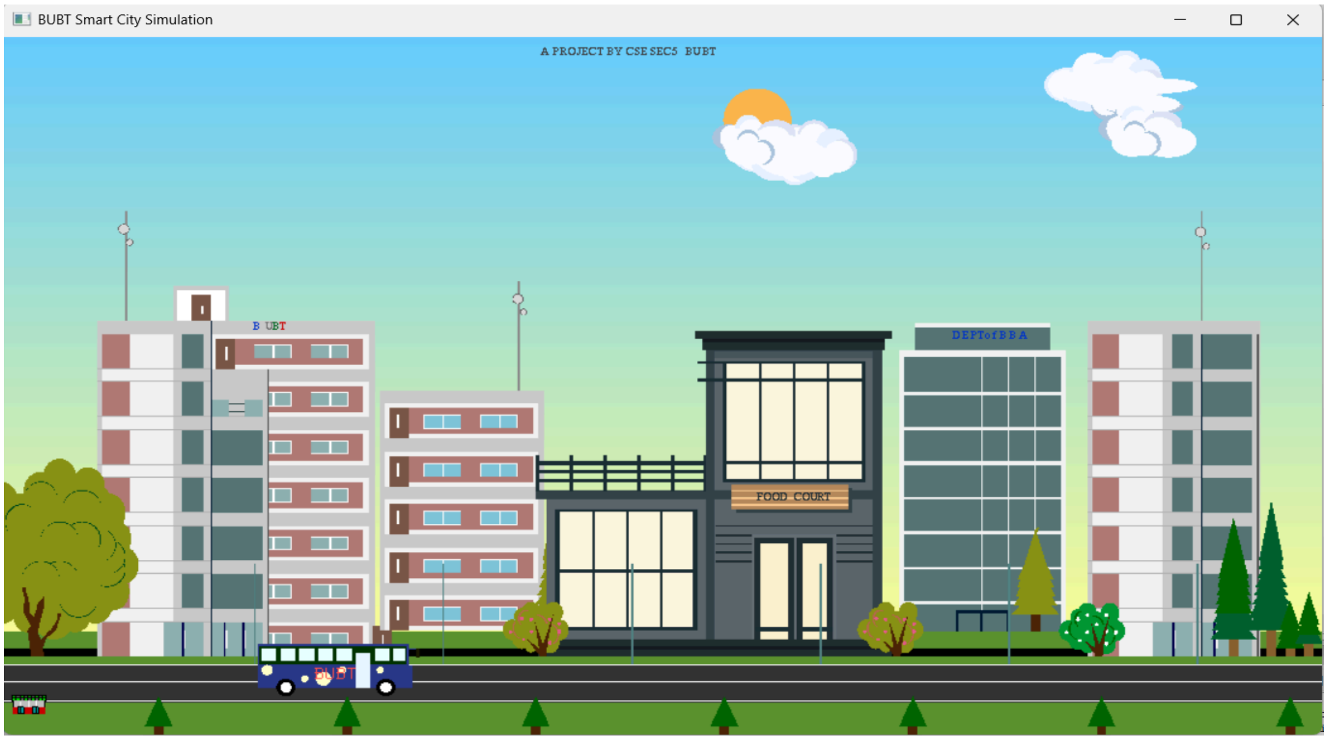
User-Defined Functions:

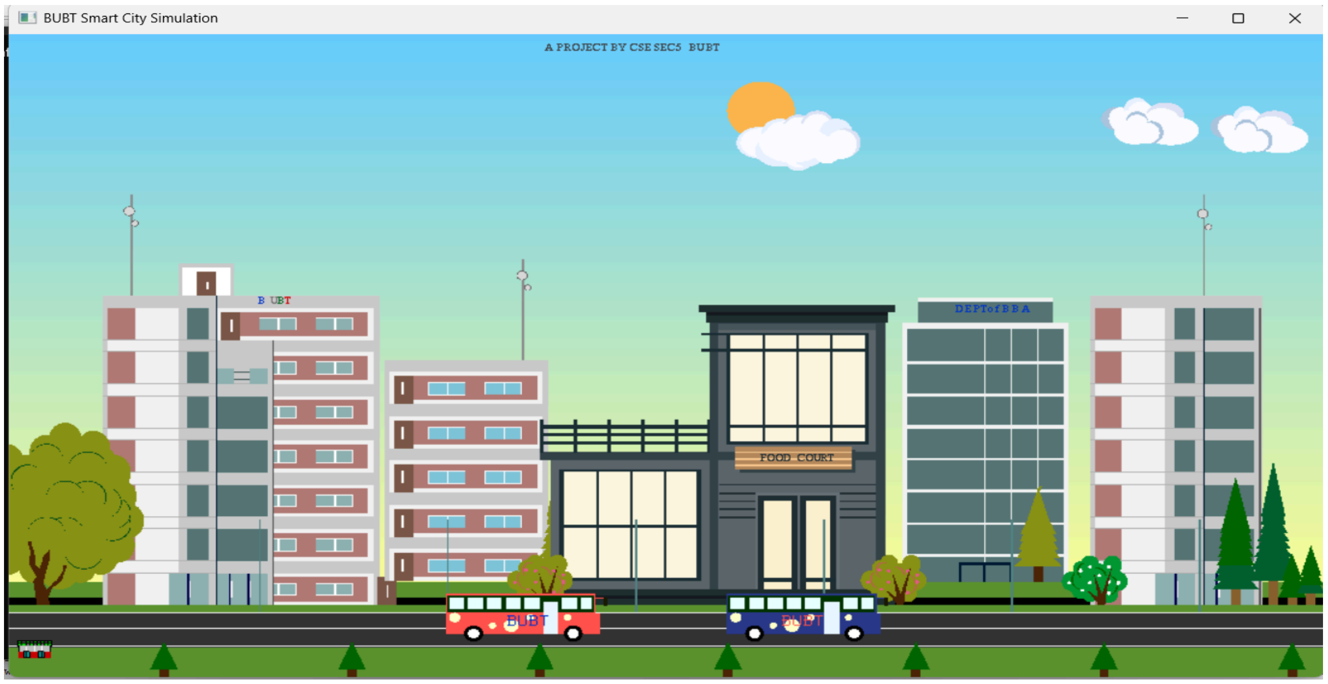
1. myInit() - Initializes the OpenGL environment by setting the background color, projection matrix, and orthographic 2D view using glOrtho().
2. circle(GLfloat rx, GLfloat ry, GLfloat cx, GLfloat cy) - Draws an ellipse or circle using GL_TRIANGLE_FAN by defining the radius and center position.
3. display() - The main rendering function that draws the full scene, including buildings, trees, towers, road, background, decorations, and text.

Default OpenGL/GLUT Functions:

4. glClearColor(r, g, b, alpha) - Sets the background color of the window.
5. glMatrixMode(GLenum mode) - Specifies which matrix stack is the target for subsequent matrix operations (e.g., GL_PROJECTION).
6. glLoadIdentity() - Replaces the current matrix with the identity matrix to reset transformations.
7. glOrtho(...) - Defines a 2D orthographic projection matrix, mapping the specified coordinate range to the viewport.
8. glBegin(GLenum mode) - Begins specifying a geometric primitive such as GL_QUADS or GL_TRIANGLE_FAN.
9. glColor3ub(r, g, b) - Sets the current color for drawing using RGB values from 0 to 255.
10. glColor3f(r, g, b) - Sets the current color using float values from 0.0 to 1.0.
11. glVertex2f(x, y) - Specifies a vertex at coordinate (x, y) in 2D space.
12. glVertex3f(x, y, z) - Specifies a vertex at coordinate (x, y, z) in 3D space (used if needed).
13. glEnd() - Ends the definition of the geometric primitive begun with glBegin().
14. glLineWidth(value) - Sets the width of lines to be drawn.
15. glClear(GL_COLOR_BUFFER_BIT) - Clears the color buffer to prepare for new drawing.
16. glRasterPos2i(x, y) - Sets the raster position for text rendering.
17. glutBitmapCharacter(font, char) - Draws a character at the current raster position using a specified GLUT bitmap font.

Visualization:





Challenges faced

- Understanding the OpenGL coordinate system and adjusting element positions accurately.
- Managing the large number of `glBegin` and `glEnd` blocks without losing track of which shape was being drawn.
- Maintaining visual balance while placing elements like buildings, trees, and towers.
- Debugging misaligned or overlapping elements caused by incorrect vertex coordinates.
- Handling color codes consistently across the different parts of the scene.
- Lack of built-in shape-drawing functions like `circle()`, which required creating custom implementations.
- Working with a deprecated OpenGL feature set (GL_QUADS), which may not be supported in modern environments.
- Ensuring that the scene scales well within the fixed viewport set by `glOrtho`.
- Writing and managing a large amount of repetitive code without abstraction.
- Testing rendering output due to differences in system resolution and window size.

Learnings

Throughout the development of this project, our team gained valuable experience in both technical and collaborative aspects of software development, particularly in the field of computer graphics and OpenGL programming. Below are the key learnings from this project:

1. Hands-on Experience with OpenGL

- We gained practical experience using OpenGL's immediate mode for 2D rendering.
- Learned to draw complex shapes using basic primitives like GL_QUADS, GL_TRIANGLES, and GL_TRIANGLE_FAN.
- Understood the significance of color layering, coordinate scaling, and how to use glOrtho() to define a 2D projection.

2. Animation and Real-Time Graphics

- Learned how to implement basic animation by updating position variables within the rendering loop.
- Gained insight into managing object states for smooth motion (e.g., buses moving across the screen).
- Discovered the importance of frame synchronization and performance considerations when working with live visuals.

3. Scene Composition and Layering

- Developed a deep understanding of how to build scenes layer by layer, starting from the background and progressing to detailed objects in the foreground.
- Learned to manage depth perception in a 2D space using proper element ordering and consistent layout design.

4. Geometry and Coordinate Logic

- Strengthened skills in mathematics, particularly in geometry and trigonometry, by building reusable functions like circle() for drawing circular shapes.
- Learned how to align objects precisely in a 2D coordinate system.

5. Collaboration and Teamwork

- Experienced real-world team collaboration by dividing responsibilities among five members.
- Practiced good coding habits like modular development, commenting, and maintaining consistent coding styles for easier integration.
- Improved our ability to communicate effectively, assign roles, and resolve conflicts during project integration.

6. Debugging and Problem Solving

- Enhanced our debugging skills while resolving visual glitches, misalignments, and animation issues.
- Learned to test components in isolation before combining them into the main project.
- Understood the importance of small iterative changes and version control in large codebases.

7. Limitations of GLUT and OpenGL Immediate Mode

- Gained awareness of the limitations of immediate mode rendering in OpenGL and how modern graphics programming has evolved to use shaders and buffers.
- Motivated to explore advanced frameworks like OpenGL Core Profile, WebGL, and game engines for future projects.

8. Presentation and Documentation

- Developed documentation and report writing skills by preparing structured sections like Objective, Tools, Flow of Execution, and Challenges.
- Understood how to explain technical work clearly for teachers, evaluators, and non-programmers.

Conclusion

In conclusion, this project has provided us with a deep and meaningful understanding of 2D computer graphics using OpenGL. We were able to visualize and implement a complete city environment with realistic features like buildings, animated buses, roads, trees, and labels. Each element was crafted through fundamental graphics programming techniques such as shape rendering, coordinate transformations, and real-time updates.

The project not only strengthened our technical capabilities but also enhanced our teamwork and communication. By dividing tasks effectively among five members, we learned how to manage large codebases, integrate independently developed components, and maintain consistency in design and functionality.

Moreover, this simulation offered a strong foundation in visual design, logical structuring, and user experience, encouraging us to further explore the possibilities of computer graphics in real-world applications. The successful completion of this project stands as a testament to our dedication, collaboration, and continuous learning throughout the course.

We believe the knowledge gained from this project will greatly support our future endeavors in software development, animation, game design, and interactive application development.