Village Scenario: A Realistic OpenGL Simulation

Shafayet Ullah Ramim
ID: 2104010202219

Shihabul Alom Sakib
ID: 2104010202221

Saikat Barua
ID: 2104010202196

Course Instructor: Salman Farsi

Department of Computer Science & Engineering

Premier University , Chittagong

Date: 2nd March, 2025

Abstract

This document presents the development of the *Village Scenario: A Realistic OpenGL Simulation*, an advanced interactive simulation project designed using OpenGL and C++. The goal of this project is to create a dynamic and immersive virtual village environment, where different real-world elements such as houses,roads,waves,boats,man,river and landscapes are simulated using OpenGL's graphical rendering capabilities. The project delves into complex modeling, simulation, and rendering techniques while maintaining a user-friendly interface. This work covers the implementation of dynamic scene generation, real-time user interaction, and optimization methods to create a smooth and engaging experience for users.

Contents

Chapter 1

Introduction

The *Village Scenario: A Realistic OpenGL Simulation* is an advanced interactive simulation project developed using OpenGL and C++. The primary objective of this project is to create a dynamic and immersive virtual village environment that simulates various real-world elements, such as houses, roads, waves, boats, people, rivers, and landscapes. By leveraging OpenGL's powerful graphical rendering capabilities, this simulation aims to bring these diverse elements to life within a virtual setting.

This project explores complex techniques in modeling, simulation, and rendering, while ensuring a user-friendly interface to allow seamless interaction. Key aspects of the implementation include dynamic scene generation, real-time user interaction, and performance optimization methods, all contributing to an engaging and smooth experience for users. The goal is to create an interactive, realistic virtual village environment that offers an immersive experience through advanced simulation techniques.

Chapter 2

Objectives

The primary objectives of this project are:

- Develop a realistic virtual environment that represents a village scenario using OpenGL.
- To implement moving wave in Village Scenerio.
- To implement moving man in Village Scenerio.
- To implement moving boat in village scenerio.

Chapter 3

Motivation

The concept of creating a virtual village simulation stems from the desire to explore and visualize how different environmental and social elements interact within a rural setting. Villages are the backbone of many cultures around the world, and the dynamics of life in these communities offer rich learning opportunities. The motivation for this project can be outlined as follows:

- **Realistic Virtual Environment**: The project aims to simulate the daily life of a village, integrating natural landscapes, human-made structures, and user interaction. This virtual environment will include terrain with hills, roads, rivers, and buildings, demonstrating how these elements shape a community.
- **User Interaction**: The simulation will allow users to interact with and navigate through the environment. This feature helps users better understand the complexities of village life and how different elements interact within the ecosystem.
- **Urban Planning and Education**: The simulation will serve as a tool for urban planning, education, and entertainment. Users can experiment with the environment, make decisions, and observe how these decisions impact the village, offering potential educational benefits for geography and environmental studies.
- **Real-World Applications**: The project merges technology and creativity to create a virtual world that can be used for entertainment, education, or urban development planning. It provides a platform for experimenting with village life and managing resources in a virtual village setting.
- **Immersion and Connection**: The project aims to immerse users in a carefully crafted environment where they can engage with the dynamics of village life. It will help them develop a deeper connection to the places that make up the fabric of rural communities.

Chapter 4

Requirement Specification

The following requirements outline the necessary components and functionalities for the Village Simulation Project:

- **Hardware Requirements**:
  - A computer with at least 4 GB of RAM.
  - A graphics card with support for 3D rendering.
  - Minimum of 2 GHz processor speed.
- **Software Requirements**:
  - Operating System: Windows, macOS, or Linux.
  - Development Environment: Any IDE supporting C++, Python, or JavaScript.
  - Graphics API: OpenGL or similar for rendering the environment.
  - A web browser (for web-based interaction, if applicable).
- **Functional Requirements**:
  - Interacts between man movement and door .
  - Dynamic wave movement .
  - Dynamic Boat movement.
- **Non-Functional Requirements**:
  - The system should run smoothly with a frame rate of at least 30 FPS (frames per second).
  - The system should have a user-friendly interface that is easy to navigate.
  - The simulation should be scalable, with the ability to add new terrain features and buildings as needed.
- **Performance Requirements**:
  - The simulation should load in under 10 seconds for a basic village setup.

Chapter 5

Methodology Feature Description

5.1 Simulation Flow

The simulation operates based on the following sequence:

1. The simulation environment initializes, setting up the village layout, objects, and structures.
2. Users can navigate through the environment, interact with objects, and manipulate the simulation settings.
3. The environment updates in real-time based on user inputs and internal dynamics.
4. The simulation continues to run until the user exits or closes the application.

5.2 Core Components

- **Landscape**: The simulated terrain and surrounding environment, including hills, roads, and rivers.
- **Buildings**: The houses and structures that make up the virtual village.
- **User Interaction**: The control system that allows the user to navigate and manipulate the environment.

5.3 Real-Time Interaction

Users can control their viewpoint within the simulation, moving across the village and interacting with various objects in the environment. The simulation allows users to zoom in on specific objects or areas, adjust the time of day, and switch between different perspectives.

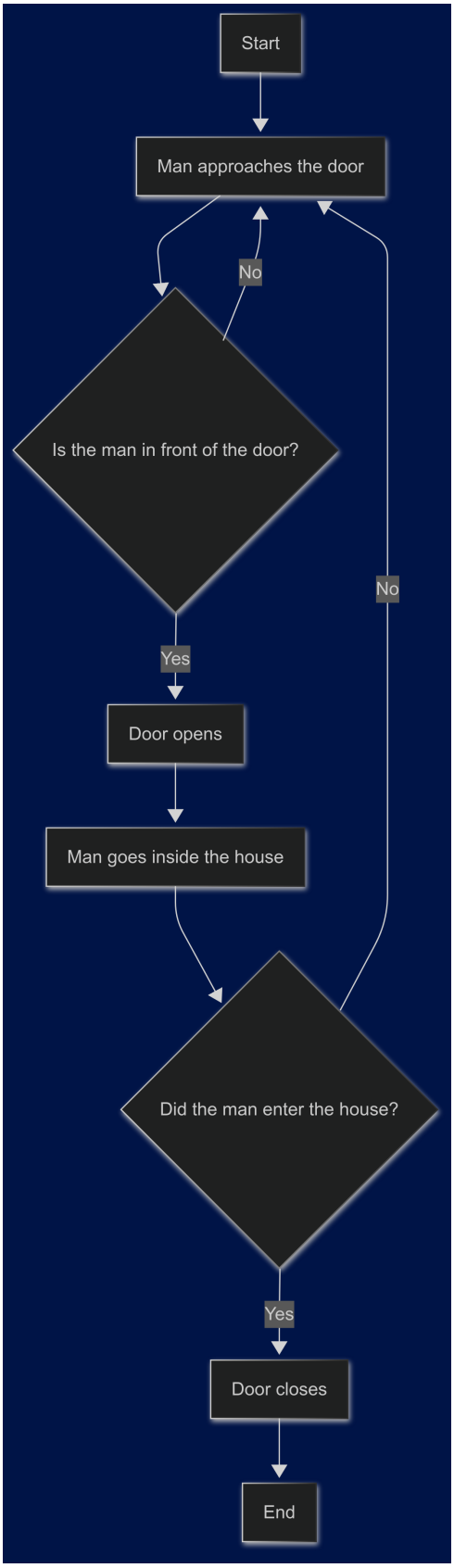

Figure 5.1: Flowchart illustrating the man's interaction with the door as he moves

Chapter 6

Implementation and Pseudocode

6.1 Libraries and Global Variables

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <stdlib.h>
#include <unistd.h>
#include<cmath>
float waveOffset = 1.50f;
float PI = 3.14157;
struct Color {
    float r, g, b;
};
float manX = -0.4f;
float manY = -1.0f;
bool isMoving = true;
float dx=0.6f;
```

Figure 6.1: Libraries and Global Variables

- **Platform-specific inclusion**: The `#ifdef __APPLE__` directive ensures that the correct GLUT library is included based on the platform. If compiling on macOS, it includes `<GLUT/glut.h>`, and otherwise, it includes `<GL/glut.h>` for other systems like Windows or Linux.
- **Standard Libraries**: The code includes libraries like `stdlib.h` for general-purpose functions, `unistd.h` for Unix system calls, and `cmath` for mathematical functions.
- **Global Variables**:
  - `waveOffset`: A variable used to adjust or animate the "wave" movement, initially set to `1.50f`.
  - `PI`: A constant value for pi, used in calculations involving circular or wave-like motion.
  - `Color struct`: A structure to represent RGB color values.
  - `manX` and `manY`: The initial position of a "man" or object in a 2D space, initially set at `(-0.4f, -1.0f)`.
  - `isMoving`: A boolean flag that indicates whether the "man" is moving. It is set to `true` to start with.
  - `dx`: A variable to adjust horizontal movement or animation speed, initially set to `0.6f`.

6.2 House Creation

- **Clearing the Buffer:** The function `glClear(GL_COLOR_BUFFER_BIT);` clears the color buffer before rendering any objects.

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    float offsetX = -0.2f; // Move everything left
    // Background Color
    glClearColor(0.5f, 0.8f, 1.0f, 1.0f);
    // Roof
    glColor3f(0.506f, 0.502f, 0.427f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.53f + offsetX, 0.23f);glVertex2f(0.53f + offsetX, 0.23f);
        glVertex2f(0.53f + offsetX, 0.27f);glVertex2f(-0.53f + offsetX, 0.27f);
    glEnd();
    // Top floor
    glColor3f(0.667f, 0.522f, 0.271f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5f + offsetX, 0.27f);glVertex2f(0.5f + offsetX, 0.27f);
        glVertex2f(0.5f + offsetX, 0.7f);glVertex2f(-0.5f + offsetX, 0.7f);
    glEnd();
    rightWindowTop(offsetX);centerWindowTop(offsetX);leftWindowTop(offsetX);
    glColor3f(0.506f, 0.502f, 0.427f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.53f + offsetX, 0.7f);glVertex2f(0.53f + offsetX, 0.7f);
        glVertex2f(0.53f + offsetX, 0.75f);glVertex2f(-0.53f + offsetX, 0.75f);
    glEnd();
    glColor3f(0.498f, 0.306f, 0.329f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.45f + offsetX, 0.75f);glVertex2f(0.45f + offsetX, 0.75f);
        glVertex2f(0.45f + offsetX, 0.85f);glVertex2f(-0.45f + offsetX, 0.85f);
    glEnd();
    glColor3f(1.0f, 1.0f, 1.0f);
    drawText("Premier University, Chittagong", -0.37f + offsetX, 0.78f);
    stair(offsetX);road(offsetX);
    glColor3f(0.667f, 0.522f, 0.271f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5f + offsetX, -0.2f);glVertex2f(0.5f + offsetX, -0.2f);
        glVertex2f(0.5f + offsetX, 0.23f);glVertex2f(-0.5f + offsetX, 0.23f);
    glEnd();
    rightWindow(offsetX);leftWindow(offsetX);door(offsetX);
    glFlush();
}
```

Figure 6.2: House making

- **Setting the Background Color:** The function `glClearColor(0.5, 0.8, 1.0, 1.0);` sets a sky-blue background.
- **Applying an Offset:** A variable `offsetX = -0.2f;` is introduced to shift all elements slightly to the left.
- **Drawing the Roof:** The roof is drawn using the `GL_POLYGON` primitive with three vertices to form a triangular shape. The function `glColor3f(0.6, 0.5, 0.3);` sets the roof's brownish color.
- **Drawing the Top and Bottom Floors:** Each floor is represented using a quadrilateral (`GL_POLYGON`) with different color values. The top floor has a slightly lighter shade compared to the bottom floor.
- **Drawing Windows:** Several functions such as `rightWindow()`, `centerWindowTop()`, and `leftWindowTop()` are used to render different windows.
- **Drawing the Door:** The function `door(offsetX);` is used to render the main entrance.
- **Displaying Text:** The function `drawText("Premier University, Chittagong", -0.37f, 0.78f);` renders a text label at the top of the scene.
- **Finalizing the Rendering:** The command `glFlush();` ensures that all OpenGL commands are executed and the rendering is displayed on the screen.

6.3 Man Creation

- **Transformation Setup:** The function first pushes the current matrix onto the stack using `glPushMatrix()` and then applies translation (`glTranslatef()`) to po-

```
void drawMan(float xPos, float yPos) {
    glPushMatrix();glTranslatef(xPos, yPos, 0.0f);
    glScalef(0.8f, 0.8f, 1.0f);
    if (yPos>=0.2f && yPos<=0.6){doorOpen = true;}
    else{doorOpen = false;}
    glColor3f(1.0f, 0.8f, 0.6f);
    float cx=0.031f,cy=-0.56f,radius = 0.032f;
    int num_segments = 100;
    glBegin(GL_TRIANGLE_FAN);glVertex2f(cx, cy);
    for (int i = 0; i <= num_segments; i++) {
        float angle = 2.0f * M_PI * i / num_segments;
        float x = cx + radius * cos(angle);
        float y = cy + radius * sin(angle);
        glVertex2f(x, y);
    }
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex2f(-0.001f, -0.63f);glVertex2f(-0.04f, -0.72f);
        glVertex2f(0.06f, -0.63f);glVertex2f(0.1f, -0.72f);
    glEnd();
    glColor3f(0.0f, 0.0f, 1.0f);
    glBegin(GL_QUADS);
    glVertex2f(-0.001f, -0.6f);glVertex2f(0.06f, -0.6f);
        glVertex2f(0.06f, -0.72f);glVertex2f(-0.001f, -0.72f);
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex2f(0.03f, -0.72f);glVertex2f(-0.01f, -0.85f);
        glVertex2f(0.03f, -0.72f);glVertex2f(0.08f, -0.85f);
    glEnd();
    glPopMatrix();
}
```

Figure 6.3: Drawing Man

sition the figure at `(xPos, yPos)`. A uniform scaling transformation (`glScalef()`) is applied to slightly resize the figure.

- **Door Interaction Logic:** If the man's `yPos` is between `0.2f` and `0.6f`, the door is set to open (`doorOpen = true`). Otherwise, the door remains closed (`doorOpen = false`).
- **Drawing the Head:** The head is represented as a filled circle using the `GL_TRIANGLE_FAN` primitive. The center of the circle is at `(cx, cy) = (0.031f, -0.56f)`, and its radius is `0.032f`. A loop iterates `num_segments = 100` times to generate a smooth circular shape.
- **Drawing the Arms:** The arms are drawn using `GL_LINES`, with two line segments extending outward from the body.
- **Drawing the Body:** The body is represented as a rectangular torso using the `GL_QUADS` primitive, colored in blue.
- **Drawing the Legs:** The legs are drawn using two separate `GL_LINES` that extend downward from the torso.
- **Finalizing the Rendering:** After rendering all parts of the human figure, the transformation matrix is restored using `glPopMatrix()` to avoid affecting other OpenGL objects.

6.4 Boat Creation

- **Offsetting the Boat's Position:** The variable `dy = 0.6f` is used to shift the boat vertically.

```
void boat4() {
    float dy = 0.6f; // Offset to move boat to (1.0, 0.7)
    glBegin(GL_POLYGON);
    glColor3ub(0, 0, 0);
    glVertex2f(0.3f + dx, -0.25f + dy);glVertex2f(0.35f + dx, -0.3f + dy);
    glVertex2f(0.65f + dx, -0.3f + dy);glVertex2f(0.7f + dx, -0.25f + dy);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3ub(255, 153, 0);
    glVertex2f(0.37f + dx, -0.25f + dy);glVertex2f(0.39f + dx, -0.17f + dy);
    glVertex2f(0.412f + dx, -0.13f + dy);glVertex2f(0.63f + dx, -0.13f + dy);
    glVertex2f(0.64f + dx, -0.16f + dy);glVertex2f(0.65f + dx, -0.25f + dy);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3ub(255, 25, 25);
    glVertex2f(0.462f + dx, -0.08f + dy);glVertex2f(0.462f + dx, 0.08f + dy);
    glVertex2f(0.465f + dx, 0.1f + dy);glVertex2f(0.564f + dx, 0.08f + dy);
    glVertex2f(0.565f + dx, 0.06f + dy);glVertex2f(0.565f + dx, -0.1f + dy);
    glEnd();

    glBegin(GL_POLYGON);
    glColor3ub(136, 204, 0);
    glVertex2f(0.5f + dx, -0.13f + dy);glVertex2f(0.5f + dx, 0.14f + dy);
    glVertex2f(0.51f + dx, 0.14f + dy);glVertex2f(0.51f + dx, -0.13f + dy);
    glEnd();
}
```

Figure 6.4: Drawing Boat

- **Drawing the Base of the Boat:** - The first `GL_POLYGON` block creates the lower part of the boat. - The boat base is colored black (`glColor3ub(0, 0, 0)`). - The shape is defined using four vertices forming a trapezoidal base.
- **Drawing the Upper Structure of the Boat:** - The second `GL_POLYGON` block creates the deck structure. - The color used is orange (`glColor3ub(255, 153, 0)`). - The polygon is formed by six vertices, giving it a slightly curved shape.
- **Drawing the Cabin:** - The third `GL_POLYGON` block represents the boat's cabin. - The color used is red (`glColor3ub(255, 25, 25)`). - The shape consists of six vertices, forming a rectangular structure.
- **Drawing the Mast:** - The fourth `GL_POLYGON` block represents the mast. - The color used is green (`glColor3ub(136, 204, 0)`). - The shape is a thin vertical rectangle created with four vertices.

6.5 Wave Creation

```
void drawWave(float amplitude, float frequency, float phaseShift, float verticalOffset) {
    glBegin(GL_LINE_STRIP);
    glColor3f(0.0f, 0.5f, 1.0f);
    for (float x = -1.5f; x <= 1.5f; x += 0.05f) {
        float y = amplitude * sin(frequency * (x + waveOffset) + phaseShift) + verticalOffset;
        glVertex2f(x, y);
    }
    glEnd();
}
```

Figure 6.5: Drawing Wave

- **Starting the Line Strip:** - The function begins with `glBegin(GL_LINE_STRIP)`, which connects a series of points with line segments. - The color of the wave is set to a blue shade (`glColor3f(0.0f, 0.5f, 1.0f)`).
- **Looping Through X Values:** - The loop iterates over values of `x` ranging from `-1.5f` to `1.5f` with a step size of `0.05f`. - This ensures a smooth representation of the wave.
- **Computing the Y-Coordinate:** - The y-value of each point is calculated using the sine wave equation:

$$y = \text{amplitude} \times \sin(\text{frequency} \times (x + \text{waveOffset}) + \text{phaseShift}) + \text{verticalOffset}$$

- Here:
  - `amplitude` controls the height of the wave.
  - `frequency` determines how many oscillations occur within the given range.
  - `phaseShift` offsets the wave horizontally.
  - `verticalOffset` shifts the entire wave up or down.

- **Drawing the Wave:** - The computed `x, y` values are passed to `glVertex2f(x, y)`, which plots the wave. - `glEnd()` is called to complete the drawing.

6.6 Interaction between man and door

```
void update(int value){
    waveOffset-=0.01f;
    if(waveOffset<-1.5f)waveOffset=1.5f;
    dx-=0.01f;
    if(dx==1.5f)dx=0.6f;
    if (isMoving) {
        if (manX < -0.05f){
            manX += 0.01f; // Move right
        }
        if (manY < 0.9f) {
            manY += 0.01f; // Move up
        }
        if (manY >= 0.6f) {
            isMoving = false; // Stop moving when reaching the door
        }
    }
    glutPostRedisplay();
        glutTimerFunc(50, update, 0);
}
```

Figure 6.6: Interaction between door and man

- **Updating the Wave Offset:** - The variable `waveOffset` is decremented by `0.01f`, causing the wave to shift left. - If `waveOffset` goes below `-1.5f`, it is reset to `1.5f`, creating a continuous wave animation.

- **Updating the Boat Position:** - The variable `dx` is decremented by `0.01f`, moving the boat left. - If `dx` reaches exactly `1.5f`, it is reset to `0.6f`, ensuring cyclic motion.

- **Animating the Man's Movement:** - The movement of a character (presumably a person) is controlled using `manX` and `manY`. - If `manX` is less than `-0.05f`, it is incremented by `0.01f`, moving the character to the right. - If `manY` is less than `0.9f`, it is incremented by `0.01f`, making the character move upwards. - When `manY` reaches `0.6f`, the boolean variable `isMoving` is set to `false`, stopping further movement.

- **Redisplaying the Scene:** - The function `glutPostRedisplay()` is called to request a redraw of the scene with the updated positions.

- **Setting the Timer for Recursion:** - The function `glutTimerFunc(50, update, 0)` schedules the next call to `update()` after 50 milliseconds, ensuring continuous animation.

Chapter 7

Output

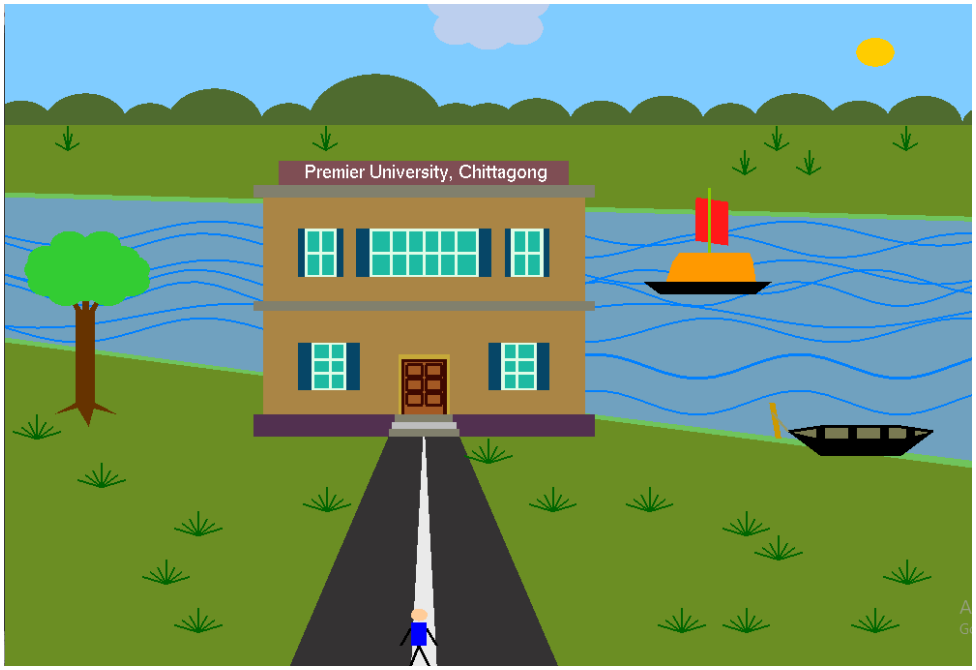

Figure 7.1: Project Output

Chapter 8

Application

The village simulation project can be applied in various fields, offering valuable insights into both practical and theoretical aspects. Some key applications include:

- **Urban Planning and Development:** The simulation can be used by urban planners and developers to visualize and experiment with village layouts, infrastructure development, and the impact of various factors like roads, housing, and environmental features. This can aid in decision-making and help create efficient, sustainable urban spaces.
- **Education and Training:** The simulation offers an interactive environment that can be used in educational settings to teach students about geography, urban development, and environmental science. It can also be used to simulate real-life situations and train individuals for disaster management, construction planning, or environmental studies.
- **Entertainment and Game Development:** The village simulation can be a foundation for developing interactive games or virtual experiences. Game developers can use it to create immersive environments, allowing players to explore and interact with dynamic virtual worlds.
- **Research and Analysis:** Researchers in fields such as environmental science, sociology, or anthropology could use the simulation to study how different variables (e.g., population density, transportation networks, natural resources) impact the development of virtual communities over time.
- **Public Policy Simulation:** The simulation can be employed by policymakers to understand how certain decisions (e.g., resource allocation, construction projects, or social policies) may affect the growth and sustainability of a virtual village. This can help in planning for real-world communities by anticipating challenges and solutions.
- **Environmental Impact Analysis:** It can also be used to simulate the impact of various environmental factors, such as climate change, urban sprawl, and deforestation, on the village's ecosystem. By adjusting variables such as water availability, terrain, and population growth, users can predict how these changes might affect the sustainability of the village.

Chapter 9

Future Scope

While the current village simulation provides a solid foundation, there are several areas where the project can be extended and improved in the future. The following points highlight some of the potential directions for further development:

- **Integration of Artificial Intelligence (AI):** One potential future enhancement is the integration of AI for simulating more realistic behavior in the environment. AI could be used to control the behavior of virtual inhabitants, allowing them to make decisions, interact with each other, and respond to environmental changes. This would add an additional layer of complexity to the simulation, making it more dynamic and lifelike.

- **3D Visualization and Virtual Reality (VR):** The current 2D simulation could be expanded into 3D to provide a more immersive experience. Virtual Reality (VR) integration could allow users to interact with the environment in real-time, providing a more hands-on approach to exploring the village. This could be particularly beneficial for educational purposes, offering students an interactive, virtual environment for learning.

- **Real-Time Data Integration:** Incorporating real-time data into the simulation could enable it to respond to real-world events. For example, environmental data (such as weather patterns or traffic congestion) could be fed into the system to simulate the effect of these events on the village. This could also include integrating data from external sources, such as population growth trends or economic statistics, to provide more accurate projections.

- **Enhanced User Interactivity and Multi-User Support:** Future versions of the simulation could allow multiple users to interact within the same environment. This would open up possibilities for collaborative decision-making, where users could work together to solve problems or plan developments in the virtual village. Enhanced user interactivity could also include the ability to modify the environment, build structures, and influence the growth of the village in real-time.

- **Integration of Environmental Factors:** To make the simulation more realistic, environmental factors such as climate change, natural disasters, and resource depletion could be integrated. These factors could affect the village's infrastructure, economy, and population, allowing users to simulate scenarios like the impact of floods, droughts, or rising temperatures on the community.

- **Expanded Application Domains:** The simulation could be extended to model larger urban areas or even entire cities, allowing for more complex interactions between different neighborhoods and infrastructure systems. Additionally, the simulation could be adapted to explore other domains such as agricultural planning, transportation networks, or public health management.

- **Improved Data Analytics and Reporting:** Future versions could incorporate advanced data analytics tools to track the performance and development of the

virtual village. Users could generate reports on various aspects such as population growth, economic development, infrastructure efficiency, and resource usage. These insights could be valuable for making data-driven decisions and planning for future developments.

These future developments offer exciting opportunities to enhance the functionality, realism, and applicability of the village simulation. By incorporating new technologies such as AI, VR, and real-time data, the project can evolve into a powerful tool for education, urban planning, and research.

Chapter 10

Conclusion

The *Village Scenario: A Realistic OpenGL Simulation* project effectively showcases the capabilities of OpenGL and C++ in developing interactive, real-time graphical simulations. Through the integration of advanced rendering techniques and dynamic user interactions, the simulation provides an engaging and immersive experience, demonstrating its potential applications in areas such as urban planning, education, and virtual environment research.

Looking ahead, future enhancements will focus on increasing the simulation's complexity and realism. Potential improvements include the incorporation of highly detailed landscapes, dynamic weather systems, and AI-driven environmental interactions. Additionally, the implementation of multiplayer functionality could facilitate collaborative exploration, making the simulation a valuable tool for interactive learning and virtual community planning.