A Mini project Report on

“Village Scenery Animation”

Submitted

by

Mr. Prem Gopal Chuniyan (Roll No: 75)
Mr. Shaikh Md.Zain Javed Ahemad(Roll No:74)

submitted in partial fulfillment of the requirements for the

award of the degree of

Bachelor

in

COMPUTER ENGINEERING

For Academic Year 2024-2025
Under the guidance of

Prof. Kanchan Pekhle

DEPARTMENT OF COMPUTER ENGINEERING
MET's Institute of Engineering Bhujbal Knowledge City Adgaon, Nashik-422003

THE MET LEAGUE OF COLLEGES
MET Bhujbal Knowledge City
AS SHARP AS YOU CAN GET

Certificate

This is to Certify that

Mr. Prem Gopal Chuniyan (Roll No: 75)
Mr. Shaikh Md.Zain Javed Ahemad (Roll No: 74)

has completed the necessary mini project work and prepare the report on

"Village Scenery
Animation"

In
satisfactory manner as a fulfilment of the requirement of the award of degree of Bachelor of Computer Engineering in the Academic year 2024-2025

Project Guide
Prof. Kanchan Pekhle

Acknowledgements

Every work is source which requires support from many people and areas. It gives us proud privilege to complete the project on "Village Scenery Animation" under valuable guidance and encouragement of our guide Prof. Kanchan Pekhle.

At last we would like to thank all the staff members and our Colleagues who directly or indirectly supported us without which the Mini project work would not have been completed successfully.

By

Mr. Prem Gopal Chuniyan(Roll No: 75)
Mr.Shaikh MdZain Javed Ahmed(Roll No: 74)

Abstract

The "Village Scenery Animation" project is a captivating 2D graphical simulation created using OpenGL to depict a serene and dynamic village environment. The primary goal of this project is to showcase an interactive, visually appealing animation that represents various elements of a rural setting, such as the sun, moving clouds, houses, trees, boats, and a river. The project aims to create a realistic and immersive experience through smooth animations and vibrant visuals.

Key features include animated movement of clouds and boats, the rising and setting of the sun, and interactive controls for users to influence specific elements, such as repositioning objects using keyboard inputs. The animation leverages OpenGL functionalities like transformations, colour rendering, and single-buffered display for smooth transitions and realistic effects.

The project demonstrates the potential of OpenGL for real-time 2D animations, blending technical implementation with artistic creativity. Future enhancements could include adding sound effects, more complex animations like villagers walking or birds flying, and improved interactivity to enrich the overall experience. The "Village Scenery Animation" is a perfect example of how programming and design can be combined to recreate the charm of rural life in a digital medium

Contents

Chapter 1

1.1 Introduction

The "Village Scenery Animation" project is an engaging 2D animation developed using OpenGL, designed to replicate the tranquil beauty and dynamic elements of a rural village. The primary aim of this project is to create a visually appealing and interactive environment that captures the essence of village life through smooth and realistic animations. Users can observe the movement of natural and man-made elements like clouds, boats, the sun, and trees, along with static structures such as houses, creating an immersive experience.

The project utilizes core OpenGL functionalities, including rendering shapes, applying transformations, and managing user inputs to animate and manipulate various objects in the scene. Keyboard controls allow users to interact with specific aspects of the animation, enhancing engagement. This project serves as a demonstration of OpenGL's potential in crafting real-time 2D simulations and lays a foundation for exploring more advanced animations and interactivity in the future.

Chapter 2

2.1 LITERATURE SURVEY

The development of 2D animations and dynamic environments has been a subject of significant research in computer graphics. OpenGL has emerged as a versatile tool for creating real-time simulations due to its robust rendering capabilities. This literature survey explores the research and methodologies relevant to the "Village Scenery Animation" project, focusing on OpenGL’s functionalities, animation techniques, and interactive design principles

1. **OpenGL for Real-Time 2D Animation**
   OpenGL has been extensively used for real-time 2D and 3D graphics due to its efficiency in rendering and transformations. Research highlights its rendering pipeline, which includes primitives for shape creation, colour handling, and transformations such as translation, scaling, and rotation. These functionalities enable developers to create lifelike animations and dynamic scenes. This project leverages OpenGL to animate elements like moving clouds, the sun, and boats..
2. **Environmental and Scenic Animations**
   Studies in environmental animations emphasize the importance of simulating natural movements to enhance realism. Techniques for animating natural phenomena like flowing water, moving clouds, and swaying trees have been developed using transformations and frame-by-frame rendering. This project incorporates these techniques to depict a realistic village setting with dynamic weather and natural elements.
3. **User Interaction in Animations**
   Research in interactive graphics stresses the significance of user engagement through input controls. Keyboard and mouse inputs are commonly used to allow users to interact with the animation. By implementing keyboard functionalities, this project enables users to manipulate certain scene elements, making the animation more engaging and personalized.
4. **Role of Smooth Transitions and Timing**
   The effectiveness of animations relies on smooth transitions and consistent frame rates. OpenGL’s timer functions and double buffering techniques ensure smooth animations without flickering. Studies also highlight the importance of coordinating animations with real-world timing to enhance visual appeal. These principles are applied in this project to maintain the fluidity of animated elements

Summary

The "Village Scenery Animation" project builds on existing research in real-time rendering, environmental animations, and interactivity. By utilizing OpenGL’s rendering capabilities, natural animation techniques, and user interaction principles, the project delivers a visually engaging and dynamic representation of village life. This serves as both an educational demonstration of OpenGL’s capabilities and a creative exploration of 2D animation development.

2.2 System Requirements:

Hardware Requirements

1. **Processor**: Intel Core i3 or equivalent (minimum); Intel Core i5 or higher recommended for smoother performance.
2. **RAM**: At least 4GB (8GB or more recommended for handling 3D graphics smoothly).
3. **Graphics Card**:
    - Integrated graphics (e.g., Intel HD Graphics) will work for basic functionality.
    - A dedicated GPU (e.g., NVIDIA GeForce GTX 750 or AMD Radeon equivalent) is recommended for enhanced rendering and smoother animation.
4. **Display**: A monitor with at least 1280x720 resolution.

Software Requirements

1. **Operating System**:
    - Windows 10 or higher
    - macOS 10.12 (Sierra) or higher
    - Linux distributions that support OpenGL and GLUT
2. **Development Environment**:
    - A C++ compiler with OpenGL support (e.g., GCC for Linux, MinGW for Windows).
    - An IDE (e.g., Visual Studio, Code::Blocks, or any text editor with build support) is optional but recommended for managing the project.
3. **Graphics Libraries**:
    - **OpenGL**: Version 2.1 or higher for compatibility with 3D rendering functions.
    - **GLUT** (OpenGL Utility Toolkit): FreeGLUT or another GLUT-compatible library, for handling window management and rendering.

Additional Requirements

- **Memory and Storage**: The project itself is lightweight, but ensure at least 500 MB of free disk space to store project files and libraries.
- **Driver Support**: Updated graphics drivers are recommended for optimized OpenGL performance.

These requirements will allow the *"Car Racing Game"* project to run efficiently, delivering smooth animations and an interactive experience.

Chapter 3

3.1 Implementation:

1. initOpenGL() Function

- **Purpose**: Initializes OpenGL settings and prepares the scene for rendering.
- **Key Operations**:

  - glClearColor(0.0, 0.0, 0.0, 1.0) — Sets the background colour of the scene to black.
  - glEnable(GL_DEPTH_TEST) — Enables depth testing to ensure proper rendering of 3D objects based on their distance from the viewer.
  - glEnable(GL_LIGHTING) — Activates lighting in the scene.
  - glEnable(GL_LIGHT0) — Enables the first light source in the scene to illuminate the objects.
  - glEnable(GL_BLEND) — Enables blending to handle transparency effects (e.g., clouds, water).
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA) — Sets the blending function for transparency.

---

2. drawSphere(float radius) Function

- **Purpose**: Draws a sphere with a given radius at the current position in the 3D space
- **Key Operations**:

  - GLUquadric* quad = gluNewQuadric() — Creates a new quadrics object to represent the sphere.
  - gluSphere(quad, radius, 30, 30) — Generates the sphere with the specified radius, using 30 longitudinal and latitudinal segments for smooth rendering.
  - gluDeleteQuadric(quad) — Frees the memory associated with the quadrics object after use.

---

3. drawSun() Function

- **Purpose**: Draws the Sun at the center of the scene as a yellow sphere
- **Key Operations**:

  - glPushMatrix() — Saves the current transformation matrix to allow for isolated transformations.
  - glColour3f(1.0, 1.0, 0.0) — Sets the colour of the Sun to yellow.
  - drawSphere(1.0) — Calls the drawSphere() function to render the Sun with a radius of glPopMatrix() — Restores the previous transformation matrix, ensuring that transformations applied to the Sun do not affect other objects.

---

4. drawCloud(float x, float y, float z) Function

- **Purpose**: Draws a cloud at the specified position in the 3D space.
- **Key Operations**:
  - glPushMatrix() — Saves the current transformation matrix to isolate cloud transformations.
  - glTranslatef(x, y, z) — Translates the object to the given (x, y, z) coordinates.
  - glColour3f(1.0, 1.0, 1.0) — Sets the colour of the cloud to white.
  - drawSphere(0.5) — Draws a small sphere to simulate a part of the cloud.
  - glPopMatrix() — Restores the previous transformation matrix

5. drawHouse(float x, float y, float z) Function

- **Purpose**: Draws a house at the specified coordinates.
- **Key Operations**:
  - glPushMatrix() — Saves the current transformation matrix.
  - glTranslatef(x, y, z) — Moves the house to the given coordinates.
  - glColour3f(0.8, 0.4, 0.0) — Sets the house colour to brown.
  - glPushMatrix() — Starts drawing the body of the house.
  - glScalef(2.0, 2.0, 2.0) — Scales the cube to form the house's body.
  - glutSolidCube(1.0) — Draws the cube representing the house.
  - glPopMatrix() — Restores the transformation state.
  - glColour3f(0.6, 0.2, 0.0) — Sets the roof colour to dark brown.
  - glPushMatrix() — Draws the roof as a cone.
  - glRotatef(45.0, 1.0, 0.0, 0.0) — Rotates the cone to form a pyramid shape.
  - glutSolidCone(1.0, 1.5, 4, 4) — Draws the roof as a cone.
  - glPopMatrix() — Restores the previous transformation matrix.

6. drawRiver() Function

- **Purpose:** Draws a river in the scene as a flat, blue rectangle
- **Key Operations**:
  - glPushMatrix() — Saves the current transformation matrix.
  - glColour3f(0.0, 0.0, 1.0) — Sets the colour of the river to blue.
  - glTranslatef(0.0, -0.5, 0.0) — Moves the river downwards.
  - glPushMatrix() — Scales the river to make it wide and flat.
  - glScalef(15.0, 0.1, 10.0) — Scales the object to represent the river.
  - glutSolidCube(1.0) — Draws the scaled cube representing the river.
  - glPopMatrix() — Restores the transformation matrix.
  - glPopMatrix() — Restores the previous matrix.

7.display() Function

- **Purpose**: The main rendering function responsible for clearing the screen, setting up the camera view, and drawing all objects (sun, houses, river, clouds).
- **Key Operations**:

  - glClear(GL_COLOUR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) — Clears both the colour and depth buffers.
  - glLoadIdentity() — Resets the modelview matrix.
  - gluLookAt(0.0, 5.0, 20.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0) — Sets the camera viewpoint.
  - drawRiver() — Draws the river at the bottom of the scene.
  - drawHouse(-4.0, 0.5, 0.0) — Draws a house at position (-4, 0.5, 0).
  - drawHouse(4.0, 0.5, 0.0) — Draws a house at position (4, 0.5, 0).
  - drawSun() — Draws the Sun at the center of the scene.
  - drawCloud(-3.0, 3.0, 0.0) — Draws a cloud at position (-3.0, 3.0, 0.0).
  - drawCloud(3.0, 3.5, 0.0) — Draws another cloud at a different position.
  - glutSwapBuffers() — Swaps the front and back buffers to display the rendered frame.

8.main(int argc ,char argv) Function

- **Purpose**: Initializes GLUT, sets up the window, and enters the main event loop for rendering
- **Key Operations**:

  - glutInit(&argc, argv) — Initializes GLUT, parsing command-line arguments.
  - glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH) — Sets the display mode for double buffering, RGB colour model, and depth buffering.
  - glutInitWindowSize(800, 600) — Sets the initial window size.
  - glutCreateWindow("Village Scenery Animation") — Creates the window with the given title.
  - init() — Calls the initialization function to set up OpenGL state.
  - glutDisplayFunc(display) — Registers the display function to handle rendering.
  - glutIdleFunc(display) — Ensures continuous updates to the display for animation.
  - glutReshapeFunc(reshape) — Registers the reshape function to handle window resizing.
  - glutMainLoop() — Enters the main event loop to keep the program running.

Chapter 4

4.1 Code:

```
#include<windows.h>

#include <iostream>

#include<GL/gl.h>

#include <GL/glut.h>

#include <math.h>

#include<windows.h>


Float moveC = 0.0f;

Float moveB1 = 0.0f;

Float moveB2 = 0.0f;

Float speed = 0.02f;

Void DrawAllComponents()

{

   glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

   glClear(GL_COLOR_BUFFER_BIT);

   glColor3d(1, 0, 0);

   glLoadIdentity();

   gluOrtho2D(-12, 38, -19, 14);//range

   glMatrixMode(GL_MODELVIEW);

   glTranslatef(1.0, 6.0, 0);
   glutSolidSphere(0.7, 250, 250);

   glPopMatrix();
   glPushMatrix();

   glColor3d(255, 255, 255);
```

```
///sky///

glBegin(GL_POLYGON);

glColor3ub(112, 160, 228);

glVertex2f(38.0, 3.0);

glVertex2f(38.0, 14.0);

glVertex2f(-12.0, 14.0);

glVertex2f(-12.0, 3.0);

glEnd();

/// Sun

glPushMatrix();

glColor3d(255, 0, 0);

glTranslatef(1.0, 7.0, 0);

glutSolidSphere(1.0, 750, 300);

glPopMatrix();

///Cloud-01

glPushMatrix();

glTranslatef(moveC, 0.0f, 0.0f);

glPushMatrix();

glColor3d(255, 255, 255);

   glutSolidSphere(0.7, 250, 250);

glPopMatrix();

glPopMatrix();
```

```
glTranslatef(1.0, 7.0, 0);

glutSolidSphere(0.7, 250, 250);

glPopMatrix();

glPushMatrix();

glColor3d(255, 255, 255);

glTranslatef(2.0, 7.0, 0);

glutSolidSphere(0.7, 250, 250);

glPopMatrix();

glPushMatrix();

glColor3d(255, 255, 255);

glTranslatef(2.0, 6.0, 0);

glutSolidSphere(0.7, 250, 250);

glPopMatrix();

glPushMatrix();

glColor3d(255, 255, 255);

glTranslatef(0.0, 6.5, 0);

glutSolidSphere(0.7, 250, 250);

glPopMatrix();

glPushMatrix();

glColor3d(255, 255, 255);

glTranslatef(3.0, 6.5, 0);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(14.0, 8.5, 0);
glutSolidSphere(0.8, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
```

```
///cloud-02

glPushMatrix();

glTranslatef(moveC, 0.0f, 0.0f);
glPushMatrix();

glColor3d(255, 255, 255);
glTranslatef(15.0, 9.0, 0);
glutSolidSphere(0.8, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(15.0, 8.0, 0);
glutSolidSphere(0.8, 250, 250);
glPopMatrix();

glPushMatrix()

glColor3d(255, 255, 255)

glTranslatef(16.0, 8.0, 0);
glutSolidSphere(0.8, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(16.0, 9.0, 0);
glutSolidSphere(0.8, 250, 250);
glTranslatef(17.0, 8.5, 0);

glutSolidSphere(0.8, 250, 250);
glPopMatrix()
glPopMatrix();

///cloud-03
glPushMatrix();
glTranslatef(moveC, 0.0f, 0.0f);
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(22.0, 8.0, 0)
glutSolidSphere(0.75, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(22.0, 7.0, 0);
glutSolidSphere(0.75, 250, 250);
glPopMatrix();
```

```
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(23.0, 7.0, 0);
glutSolidSphere(0.75, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(24.0, 7.5, 0);
glutSolidSphere(0.75, 250, 250);
glPopMatrix();
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(21.0, 7.5, 0);
glutSolidSphere(0.75, 250, 250);
glPopMatrix();
glPopMatrix();
///See portion

glPushMatrix();

glBegin(GL_POLYGON);

glColor3ub(65, 105, 225);

glVertex2f(40.0, -19.0); /// right-down

glVertex2f(38.0, -11.0); /// right-top

glVertex2f(-12.0, -11.0); /// left-top

glVertex2f(-12.0, -19.0); /// left-down

glEnd();

///Ground portion

glBegin(GL_POLYGON);

glColor3ub(25, 128, 0);
glPushMatrix();
glColor3d(255, 255, 255);
glTranslatef(23.0, 8.0, 0);
glutSolidSphere(0.75, 250, 250);
glPopMatrix();

glVertex2f(90.0f, -12.5f); /// angle
glVertex2f(38.0f, -12.0f); /// right-down
glVertex2f(38.0f, 3.0f); /// right-top
glVertex2f(-12.0f, 3.0f); /// left-top
```

```
    glVertex2f(-12.0f, -11.0f);  /// left-
down
   glEnd();
/// House Drawing
   ///1st House
   /// FEG
   glColor3ub(162, 116, 36);
   glBegin(GL_TRIANGLES);
   glVertex2f(-8, -2);
   glVertex2f(-6, 0);
   glVertex2f(-4, -2);
   glEnd();
   /// FGIH
  glColor3ub(214, 42, 50);
  glBegin(GL_QUADS);
   glVertex2f(-8.023, -5.184);
   glVertex2f(-8, -2);
   glVertex2f(-4, -2);
   glVertex2f(-3.999, -7.219);
   glEnd();

   ///ROPQ
   glColor3ub(19, 105, 51);
   glBegin(GL_QUADS);
   glVertex2f(-6.73, -5.67);
   glVertex2f(-6.76, -3.58);
   glVertex2f(-5.19, -3.58);
   glVertex2f(-5.19, -6.62);
   glEnd();
   ///EGKJ
   glColor3ub(124, 85, 17);
   glBegin(GL_QUADS);
   glVertex2f(-6, 0);
   glVertex2f(4, 0);
   glVertex2f(6, -2);
   glVertex2f(-4, -2);
   glEnd();
   ///GLMI
  glColor3ub(156, 9, 16);
  glBegin(GL_QUADS);
  glVertex2f(-4, -2);
   glVertex2f(5.22, -2);
   glVertex2f(5.26, -5.11);
glVertex2f(-3.99, -7.22);
/SZA1V
   glColor3ub(19, 105, 51);
   glBegin(GL_QUADS);
   glVertex2f(-0.42, -3.62);
   glVertex2f(0.38, -4.38);
  glVertex2f(0.38, -5.53);
```

```
glVertex2f(-0.39, -6.40);
glEnd();
glEnd();
  ///TB1C1U

glColor3ub(19, 105, 51);

glBegin(GL_QUADS);

glVertex2f(1.95, -3.58);

glVertex2f(1.21, -4.38);

glVertex2f(1.21, -5.46);

glVertex2f(1.97, -5.86);

glEnd();
///2nd House

///S1R1W1

glColor3ub(0, 0, 0);

glBegin(GL_TRIANGLES);

glVertex2f(8.09, -5.01);

glVertex2f(10.96, -2.02);

glVertex2f(14.10, -5.01);

glEnd();

///U1V1Z1W1

glColor3ub(145, 0, 12);

glBegin(GL_QUADS);

glVertex2f(9.15, -5.01);

glVertex2f(9.14, -9.02);

glVertex2f(12.85, -9.04);

glVertex2f(12.84, -5.01);

glEnd();
```

```
   ///VSTU

   glColor3ub(216, 215, 111);
   glBegin(GL_QUADS);
   glVertex2f(-0.39, -6.40);
   glVertex2f(-0.42, -3.62);
   glVertex2f(1.95, -3.58);
   glVertex2f(1.97, -5.86);
   glEnd();
   ///A2B2C2D2
   glColor3ub(245, 10, 49);

   glBegin(GL_QUADS);
  glVertex2f(10.36, -6.79);
   glVertex2f(10.35, -8.91);
   glVertex2f(11.59, -8.99);
   glVertex2f(11.58, -6.79);
   glEnd();
///NWJ1H1
   glColor3ub(191, 158, 24);
   glBegin(GL_QUADS);
   glVertex2f(19.02, -6.06);
   glVertex2f(20.96, -6.09)
   glVertex2f(21, -10.08);
   glVertex2f(19.01, -10.08);
   glEnd();
   ///1st  Tree
   glColor3ub(22, 208, 70);
   glBegin(GL_QUADS);
   glVertex2f(14.88, -6.00);
   glVertex2f(20, 0);
   glVertex2f(24.98, -6.16);
   glVertex2f(21.05, -4.96);
   glVertex2f(19.09, -4.95);
   glEnd();

   /// 2nd Tree
   ///L1K1M1
   glColor3ub(17, 218, 45);
   glBegin(GL_TRIANGLES);
   glVertex2f(27.02, -6.16);
   glVertex2f(30, 0);
  glVertex2f(32.93, -6);
   glEnd();
///E1F1P1Q1
   glColor3ub(191, 158, 24);
   glBegin(GL_QUADS);
   glVertex2f(29.09, -6.10);
   glVertex2f(31.03, -6.05);
   glVertex2f(31.05, -10.03);
```

```
 glVertex2f(29.06, -9.97);
 glEnd();
 ///Rail Line
///N1B
 glColor3ub(155, 29, 29);
 glBegin(GL_QUADS);
 glVertex2f(-12, 2);
 glVertex2f(38, 3);
 glVertex2f(37.98, 2.69);
 glVertex2f(-11.98, 1.71);
 glEnd()
 glColor3ub(155, 29, 29);
 glBegin(GL_QUADS);
 glVertex2f(-12.07, 0.72);
glVertex2f(-12.07, 0.48);
 glVertex2f(38.02, 1.52);
 glVertex2f(38, 1.70);
 glEnd();

 glColor3ub(0, 0, 0);
 glBegin(GL_QUADS);
 glVertex2f(-8, 2.08);
 glVertex2f(-7.64, 2.08);
 glVertex2f(-7.63, 0.63);
 glVertex2f(-8, 0.62);
 glEnd();

 glColor3ub(0, 0, 0);
 glBegin(GL_QUADS);
 glVertex2f(0.97, 2.31);
 glVertex2f(1.28, 2.27);
 glVertex2f(1.34, 0.76);
 glVertex2f(0.97, 0.75);
 glEnd();
 glColor3ub(0, 0, 0);
 glBegin(GL_QUADS);
 glVertex2f(11.98, 2.48);
 glVertex2f(12.48, 2.49);
 glVertex2f(12.5, 1);
 glVertex2f(11.98, 0.98);

 glEnd();
 glColor3ub(0, 0, 0);
 glBegin(GL_QUADS);
 glVertex2f(25.45, 2.75);
 glVertex2f(26.09, 2.76);
 glVertex2f(26.09, 1.27);
 glVertex2f(25.45, 1.26);
 glEnd();
 glColor3ub(0, 0, 0);
```

```
 glBegin(GL_QUADS);
 glVertex2f(33.99, 2.92);
 glVertex2f(34.48, 2.93);

 glVertex2f(34.58, 1.45);
 glVertex2f(34.03, 1.44);
 glEnd();
 glColor3ub(0, 0, 0);
glBegin(GL_QUADS)
glVertex2f(18.39, 2.69);
 glVertex2f(19.16, 2.62);
 glVertex2f(19.18, 1.13);
 glVertex2f(18.42, 1.11);
 glEnd();
 ///boat-1 speed left to right
 glPushMatrix();

 glTranslatef(moveB1, 0.0f, 0.0f);
 glBegin(GL_QUADS);
 glColor3ub(139, 69, 19);
 glVertex2f(1.0f, -14.0f);
 glVertex2f(1.50f, -13.0f);
 glVertex2f(-2.0f, -13.0f);
 glVertex2f(-3.0f, -14.0f);
 glEnd();

 glBegin(GL_QUADS);
glColor3ub(0, 0, 0);
glVertex2f(2.0f, -14.5f);
glVertex2f(2.0f, -14.0f);
glVertex2f(-3.0f, -14.0f);
 glVertex2f(-3.0f, -14.5f);
 glEnd();
 glVertex2f(1.5f, -13.0f);
glVertex2f(1.0f,  -14.0f);
glEnd();
 glPopMatrix();
 glBegin(GL_QUADS);
 glColor3ub(0, 0, 0);
 glVertex2f(22.0f, -18.0f)
 glVertex2f(22.0f, -17.5f);
 glVertex2f(17.0f, -17.5f);

 glVertex2f(17.0f, -18.0f);

 glEnd();
```

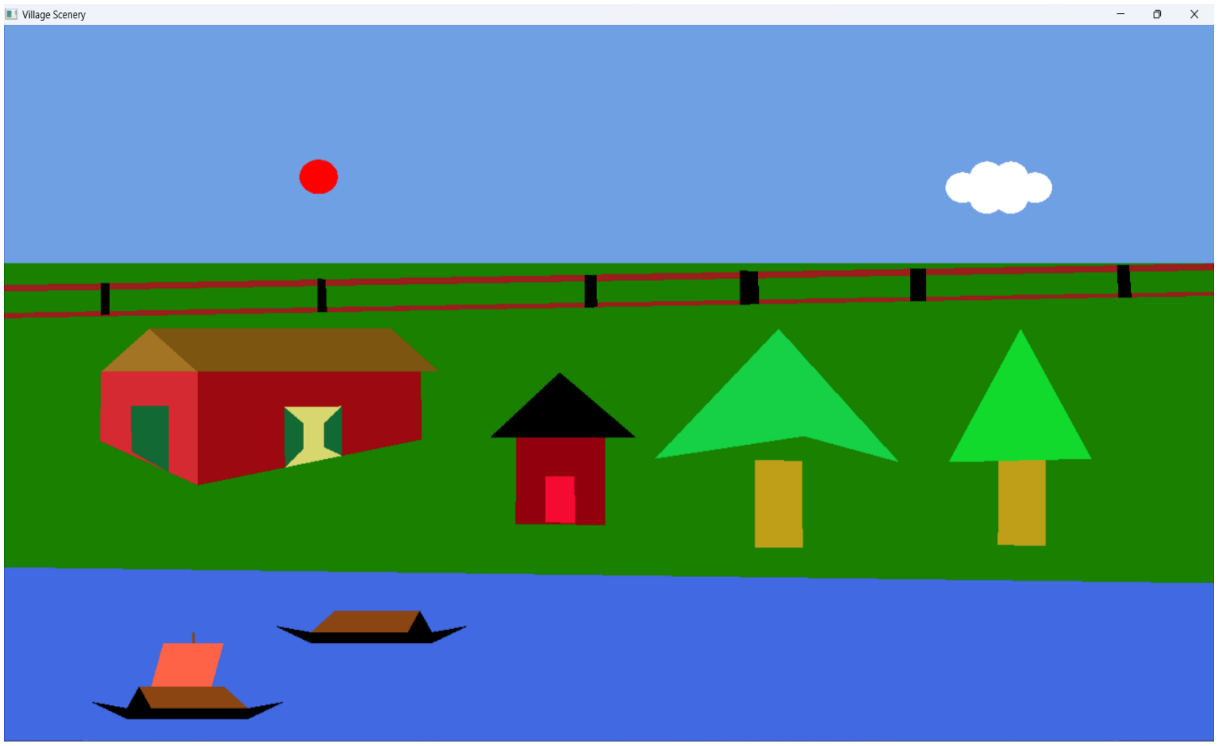
```
    glBegin(GL_QUADS);
    glColor3ub(255, 99, 71);
    glVertex2f(20.5f, -16.5f);
    glVertex2f(21.0f, -14.5f);
    glVertex2f(18.5f, -14.5f);
    glVertex2f(18.0f, -16.5f);
    glEnd();
    glBegin(GL_QUADS);

    glColor3ub(139, 69, 19);
   glVertex2f(19.8f, -14.5f);
   glVertex2f(19.8f, -14.0f);
   glVertex2f(19.7f, -14.0f);
   glVertex2f(19.7f, -14.5f);
   glEnd();
   glBegin(GL_TRIANGLES);
   glColor3ub(0, 0, 0);
   glVertex2f(17.0f, -18.0f);
   glVertex2f(17.0f, -17.5f);
   glVertex2f(15.5f, -17.2f);
   glEnd();
   glBegin(GL_TRIANGLES);
   glColor3ub(0, 0, 0);
   glVertex2f(22.0f, -17.5f);
   glVertex2f(22.0f, -18.0f);
   glVertex2f(23.5f, -17.2f);
   glEnd();
   glBegin(GL_TRIANGLES);
   glColor3ub(0, 0, 0);
   glVertex2f(17.0f, -17.5f);
   glVertex2f(18.0f, -17.5f);
   glVertex2f(17.5f, -16.5f);
   glEnd();
    glPopMatrix();
    //end
    glPopMatrix();
    glutSwapBuffers();
}
Void updateC(int value)
{
    If (moveC > +36)
    {
        moveC = -38;
    }
    moveC += 0.15;
    glutTimerFunc(20, updateC, 0);
    glutPostRedisplay();
}
```

```
Void updateB1(int value)
{
    moveB1 += speed;
    if (moveB1 > 38)
    {
        moveB1 = -38;
    }
    // moveB1 += 0.17;
    glutTimerFunc(20, updateB1, 0);
//Notify GLUT to call update again in
25 milliseconds
    glutPostRedisplay();
}
Void updateB2(int value)
{
    If (moveB2 < -36)
    {
moveB2 = +38;
    }
    //Notify GLUT that the display has
changed
    moveB2 -= 0.13;
 glutTimerFunc(20, updateB2, 0);
//Notify GLUT to call update again in
25 milliseconds
    glutPostRedisplay();

}    }

}
Void handleMouse(int button, int state,
int x, int y)
{
   If (button ==
GLUT_LEFT_BUTTON)
{    Speed += 0.05f;
 }Void display(void)
{
    DrawAllComponents();
    glFlush();
}    Else if (button ==
GLUT_RIGHT_BUTTON)
    {
     Speed -= 0.05f;
    }    glutPostRedisplay();

Void init()
{
   glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
}
```

```
Void handleKeypress(unsigned char key, int x, int y)
{
  Switch (key)
{
    Case ‘s’://stops
    Speed = 0.0f;
        Break;
    Case ‘r’://runs
        Speed = 0.02f;
        Break;
        glutPostRedisplay();


Int main(int argc, char** argv)
{
  glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE |
GLUT_RGB);
    glutInitWindowSize(1300, 700);

    glutInitWindowPosition(100, 100);

    glutCreateWindow(“ Village Scenery “);

    in
    glutTimerFunc(100, updateC, 1);    // cloud speed

    glutTimerFunc(100, updateB1, 0); //Boat 1

    glutTimerFunc(100, updateB2, 0); //boat 2



    glutKeyboardFunc(handleKeypress);

    glutMouseFunc(handleMouse);

    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
```

4.2 Results:

Figure 1



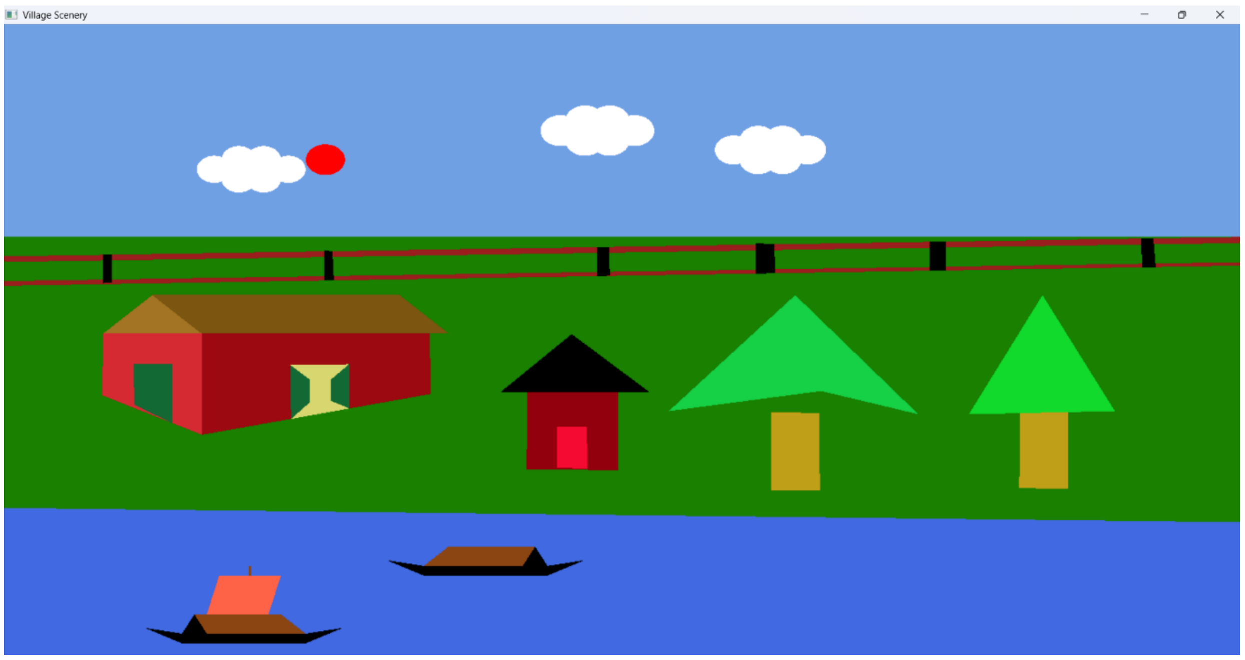


Figure 2

5.Conclusion:

The **Village Scenery Animation** project effectively demonstrates the use of OpenGL for creating a visually appealing and immersive 3D environment. By incorporating essential elements like the Sun, houses, a river, and trees, the project showcases how computer graphics principles can be applied to render natural scenes with realistic lighting and transformations. The integration of various OpenGL techniques, such as object rendering, camera manipulation, and smooth animations, highlights the potential of 3D graphics programming in creating interactive and dynamic visual experiences.

This project serves as a practical application of OpenGL's capabilities, emphasizing fundamental concepts such as lighting, texture mapping, and scene management to bring the village to life. The inclusion of features like object movement and camera positioning enhances the user experience, making the environment feel more immersive.

The **Village Scenery Animation** project not only showcases the core functionalities of OpenGL but also provides a foundation for future improvements. Potential enhancements, such as adding more dynamic weather effects, animated characters, or a day-night cycle, can further enrich the scene. Overall, the project demonstrates the power of OpenGL in visualizing 3D environments and serves as an excellent learning experience in applying computer graphics concepts to create realistic and interactive simulations.

6.References :

1. **OpenGL Utility Toolkit (GLUT)**
   Reference: Official GLUT Documentation
2. **Collision Detection in 2D Games**
   Reference: Ericson, Christer. *"Real-Time Collision Detection."* CRC Press, 2004.
3. **OpenGL Programming Guide**
   Reference: Shreiner, Dave, et al. *"OpenGL Programming Guide: The Official Guide to Learning OpenGL."* Addison-Wesley, 9th edition, 2013.