A Project Report for

COMPUTER GRAPHICS
(UCS505)
on
VILLAGE SCENERY

Submitted by:

ASEEM GOEL 102103729
KARTIK KOUL 102103819

Group: 3CO26
BE Third Year

Submitted to:

Dr. Harpreet Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,

PATIALA, PUNJAB, INDIA

Jan-May 2024

TABLE OF CONTENTS

INTRODUCTION TO THE PROJECT

"Village Scenery" is a delightful project that transports you to the idyllic charm of rural life through computer-generated graphics. Using OpenGL, we've crafted a serene countryside landscape featuring elements like lush greenery, quaint houses, and a meandering river. The scene captures the essence of tranquility and natural beauty, inviting viewers to immerse themselves in the peaceful ambiance of village life. Whether you're seeking relaxation or simply appreciate the timeless allure of rural settings, "Village Scenery" promises to captivate your senses and evoke a sense of nostalgia for simpler times.

Here is a glimpse of what awaits you in our village landscape:

1. **Nature's Beauty:** Picture a canvas painted with a soothing sky adorned with gentle, drifting clouds and a radiant sun casting its warm glow. As if embracing the scene, a meandering river adds a touch of serenity, completing the picturesque setting.

2. **Verdant Surroundings:** Our village is nestled amidst lush greenery, with trees standing tall and proud, their branches adorned with vibrant foliage. The fields, with their varying shades of green, offer a visual feast for the eyes, inviting you to lose yourself in their tranquility.

3. **Quaint Architecture:** Explore the quaint charm of our village houses, each one adorned with a sloping roof and welcoming wooden doors and windows. These cozy abodes exude the timeless appeal of countryside living, beckoning you to step inside and experience their warmth.

4. **Riverside Serenade:** Embark on a journey down the river as a graceful boat glide through its shimmering waters, adding a touch of life and movement to the tranquil scene. It is a reminder of the vibrant pulse that flows through even the most serene of landscapes.

Overall, the "Village Scenery" project aims to evoke a sense of nostalgia and tranquility, inviting viewers to immerse themselves in the idyllic charm of rural life. Through the integration of computer graphics techniques, it presents a visually captivating representation of a timeless landscape that resonates with audiences of all ages.

COMPUTER GRAPHICS CONCEPTS USED

Sure, here are the computer graphics concepts used in the "Village Scenery" project:

1. **OpenGL:** The project utilizes OpenGL, a widely used graphics library, for rendering 2D and 3D graphics.

2. **Primitive Drawing:** Various OpenGL primitive drawing functions such as `glBegin()`, `glEnd()`, and `glVertex2i()` are used to draw basic geometric shapes like points, lines, and polygons.

3. **Color Specification:** The project specifies colors using the `glColor3ub()` function, which sets the color for subsequent drawing operations.

4. **Polygon Filling: Colors can be filled inside any polygon using the glColor3f function, specifying the RGB values.**

5. **Lines:** Lines are drawn by specifying their starting and ending points (x, y coordinates) in space.

6. **Transformations:** Translation transformations are applied to objects using `glTranslatef()` to move them across the screen, creating animations and positioning elements in the scene.

    - **Translation:** Objects are moved across the screen or in space without deformation by adding to their x, y, and z coordinates.
    - **Rotation:** Objects can be rotated about any pivot point along any axis or plane, changing their angle.
    - **Scaling:** The size of objects can be changed by scaling their x and y coordinates.

7. **2D Drawing:** The project focuses on 2D graphics, utilizing functions like `glRecti()` to draw rectangles and `glRectf()` to fill them with colors.

8. **Looping:** Loops are used to iterate over elements in the scene, such as drawing multiple instances of clouds, trees, or fences.

9. **Animation:** Animation is achieved by continuously updating the positions of objects, such as the clouds drifting across the sky and the boat moving along the river.

10. **Polygon Drawing:** the process of rendering a closed shape with straight sides and defined vertices, forming a 2D geometric figure.

    - **Circle:** Circles are drawn by constructing a polygon around the center with a particular radius, using a loop over angles from 0 to 360 degrees.
    - **Sphere:** Spheres are created using the glutSolidSphere library function.
    - **Through Points:** Polygons are constructed by looping over different points in space.

These concepts come together to create a visually appealing and immersive village scenery experience.

USER DEFINED FUNCTIONS

User-defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs. They are also commonly known as “tailor-made functions” which are built only to satisfy the condition in which the user is facing issues meanwhile reducing the complexity of the whole program**.**

In the picturesque landscape of the village scenery, each user-defined function plays a vital role in bringing the idyllic setting to life. The "cloud" function delicately crafts fluffy clouds that lazily drift across the azure sky, adding a touch of whimsy to the tranquil scene. Meanwhile, the "sun" function paints a radiant orb of golden light, casting its warm glow upon the rolling fields and tranquil riverbanks. As for the "fence" function, it meticulously constructs rustic fences that crisscross the landscape, defining boundaries and adding a rustic charm to the pastoral setting. Finally, the "display" function serves as the master orchestrator, seamlessly weaving together these elements to create a captivating tableau of rural serenity. Each User Defined Functions [UDF’s] are explained as follows: -

- **cloud(x, y):** This function takes two arguments, x and y, representing the centre coordinates of the cloud. It likely uses a loop to draw a circle using triangle fans. Triangle fans are a way to efficiently draw multiple triangles with a common center vertex. This creates a fluffy, cloud-like shape.
- **sun(x, y):** Similar to cloud, this function takes coordinates and presumably draws a yellow circle to represent the sun.
- **fence(x):** This function likely takes a single argument x for the starting position and then draws a series of brown rectangles to create a fence.
- **display():** This is the main rendering function. It's responsible for calling all the other drawing functions to assemble the complete village scene. It likely includes logic for:

    a) Clearing the screen
    b) Drawing the sky, field, ground, sun, & river using basic shapes and colours
    c) Calling fence(x) in a loop to create multiple loops.
    d) Drawing the trees, tubewell, houses and boat using the combination of shapes & colours.

CODE

```
#include<windows.h> // Managing window creation, handling of messages and other tasks specific to the Windows operating system.
#include<GL/glut.h> // Necessary header file to use OPENGL and GLUT
#include<stdlib.h>
#include<math.h>
double  r = .2, s = .3;
int i;
float  tx = 10, bx = 10, dy = 0.5;
float angle = 0.0f;
int cw = 0;

void init()
{
   glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Sets the clear colour for the colour buffer. Arguments of the format (R, G, B, Alpha). Alpha represents the measure of transparency. Hence, the clear buffer colour here is opaque white.
   glOrtho(-210, 210, -220, 310, -210, 310); // Defines the size of the orthographic projection. Basically decides the size of the viewing frame, in which we view our drawing. Any area outside the specified region is clipped out. So, here, the size of the window will be a width of 420 units and a height of 530 units.
   // glOrtho(left, right, bottom, top, near, far)
}

// Defining the clouds

void cloud(double x, double y)
{
   glBegin(GL_TRIANGLE_FAN); // Arranges triangles in a fan-like manner, with the first drawn vertex being the common "central" vertex. Using this technique to draw triangles along the defined circumference of a circle is a good technique to draw circles.
   for (i = 0; i < 360; i++)
   {
      x = x + cos((i * 3.14) / 180) * r;
      y = y + sin((i * 3.14) / 180) * r;

      glVertex2d(x, y);

   }

   glEnd();

}
```

```
// Defining the sun

void sun(double x, double y)
{
   glBegin(GL_TRIANGLE_FAN); // This creates more of an ellipse than a circle
   for (i = 0; i < 360; i++)
   {
      x = x + cos((i * 3.14) / 180) * s;
      y = y + sin((i * 3.14) / 180) * s;

      glVertex2d(x, y);

   }
   glEnd();
}

// Defining a fence

void fence(int x)
{
   glBegin(GL_POLYGON); // Specifies that a polygonal shape is to be drawn
   glColor3ub(75, 54, 30); // Sets the colour of the polygon that is to be drawn

   // Specifying the vertices of the rectangles to be drawn for the fence. The width of each individual fence segment is 3 units, while the height is 60 units.

   glVertex2i(-190 + x, 130);
   glVertex2i(-190 + x, 70);
   glVertex2i(-187 + x, 70);
   glVertex2i(-187 + x, 130);
   glVertex2i(-190 + x, 130);

   glEnd();

}

void display()
{

   glClear(GL_COLOR_BUFFER_BIT);

   // Drawing the sky

   glColor3ub(135, 206, 200); // Giving the light blue colour
   glRecti(-200, 300, 200, 100); // Draws a rectangle with the bottom left vertex of (-200,100) and the top right vertex of (200,300).
```

```
    // Drawing the field

    glBegin(GL_POLYGON); // Beginning to draw a polygon
    glColor3ub(0, 100, 0); // Giving the colour of dark green

    glVertex2i(-200, 100);
    glVertex2i(-100, 160);
    glVertex2i(0, 100);
    glVertex2i(50, 70);
    glVertex2i(100, 180);
    glVertex2i(200, 100);
    glColor3ub(255, 215, 0);
    glVertex2i(200, -200);

    glVertex2i(-200, -200);
    glColor3ub(255, 215, 0); // Adding a golden colour to give a yellow-ish transitional colour to the
field
    glVertex2i(-200, 100);

    glEnd();

    // Drawing the sun

    glColor3ub(255, 174, 66);
    sun(70, 260);

    // Drawing the fences
    int x = 0;
    for (int i = 0; i < 39; i++) // Drawing a total of 39 fences
    {
        fence(x);
        x += 10; // Distance of 10 horizontal units between each individual fence
    }

    glColor3ub(75, 54, 30);
    glRecti(-200, 120, 200, 115);
    glRecti(-200, 100, 200, 95);
    glRecti(-200, 85, 200, 80);
```

```
// Drawing the smaller clouds

glPushMatrix();
glColor3ub(220, 220, 220);
glTranslatef(tx, 0, 0);
cloud(0, 250);
cloud(15, 245);
cloud(10, 240);
cloud(-2, 243);

cloud(-80, 250);
cloud(-95, 245);
cloud(-90, 240);
cloud(-90, 243);
cloud(-75, 243);

sun(-165, 260);
sun(-185, 245);
sun(-180, 240);
sun(-152, 243);

glPopMatrix();
tx += .03;
if (tx > 200)
    tx = -200;

// Drawing the smaller tree, using the sun() to draw the leaves of the tree

glColor3ub(139, 69, 19); // Setting the colour for the trunk of the tree
glRecti(-20, 200, -13, 140); // Drawing the trunk using a rectangle
glColor3ub(0, 140, 0); // Setting the colour for the leaves
sun(-30, 190);
sun(0, 190);
sun(-10, 210);
sun(-30, 175);
sun(0, 170);

// Drawing the big tree, with individual components

glBegin(GL_POLYGON); // First part, drawing the trunk
glColor3ub(139, 69, 19);
glVertex2i(-170, 160);
// glVertex2i(-168, 120);
glVertex2i(-178, 40);
glVertex2i(-145, 40);
glVertex2i(-153, 120);
glVertex2i(-150, 160);
glVertex2i(-170, 160);
glEnd();
```

```
glBegin(GL_POLYGON);  // Second part, drawing the rightmost branch of the tree
glColor3ub(139, 69, 19);
glVertex2i(-153, 100);
glVertex2i(-100, 200);
glVertex2i(-95, 200);
glVertex2i(-153, 80);
glVertex2i(-153, 100);
glEnd();

glBegin(GL_POLYGON);  // Third part, drawing the leftmost branch of the tree
glColor3ub(139, 69, 19);
glVertex2i(-170, 160);
glVertex2i(-185, 210);
glVertex2i(-190, 210);
glVertex2i(-168, 90);
glVertex2i(-170, 160);
glEnd();

glBegin(GL_POLYGON);  // Fourth part, drawing the middle branch of the tree
glColor3ub(139, 69, 19);
glVertex2i(-160, 160);
glVertex2i(-150, 210);
glVertex2i(-140, 210);
glVertex2i(-150, 160);
glVertex2i(-160, 160);
glEnd();

glColor3ub(0, 128, 0); // Drawing the leaves
sun(-95, 200);
sun(-85, 180);
sun(-110, 180);
sun(-150, 200);
sun(-130, 210);
sun(-140, 230);
sun(-190, 210);
sun(-185, 200);
sun(-195, 190);
```

```
// Drawing the base of the tube well, with individual components

glBegin(GL_POLYGON);  // Drawing the larger upper base of the tube well
glColor3ub(50, 50, 50);//
glVertex2i(115, 65);
glVertex2i(95, 5);
glVertex2i(145, 5);
glVertex2i(165, 65);
glVertex2i(115, 65);
glEnd();

glBegin(GL_POLYGON);  // Drawing the smaller upper base of the tube well
glColor3ub(100, 100, 100);
glVertex2i(120, 58);
glVertex2i(104, 10);
glVertex2i(140, 10);
glVertex2i(155, 58);
glVertex2i(120, 58);
glEnd();

glColor3ub(0, 0, 0); // Front view of the lower base
glRecti(95, 5, 145, -6);

glBegin(GL_POLYGON);  // Side view of the lower base
glColor3ub(0, 0, 0);
glVertex2i(165, 65);
glVertex2i(166, 55);
glVertex2i(145, -6);
glVertex2i(145, 5);
glVertex2i(165, 65);
glEnd();

// Drawing the tube well itself

glBegin(GL_POLYGON);  // Part 1
glColor3ub(139, 128, 0);
glVertex2i(120, 85);
glVertex2i(120, 30);
glVertex2i(125, 28);
glVertex2i(130, 30);
glVertex2i(130, 85);
glVertex2i(125, 87);
glVertex2i(120, 85);
glEnd();
```

```
glBegin(GL_POLYGON); // Part 2
glColor3ub(255, 215, 0);
glVertex2i(120, 85);
glVertex2i(125, 80);
glVertex2i(130, 85);
glVertex2i(125, 87);
glVertex2i(120, 85);
glEnd();

glColor3ub(139, 128, 0); // Part 3
glRecti(123, 100, 126, 85);

glPushMatrix();
glTranslatef(128, 102, 0);
glRotatef(angle, 0, 0, 1);
glTranslatef(-128, -102, 0);

glBegin(GL_POLYGON);  // Part 4
glColor3ub(139, 128, 0);
glVertex2i(126, 100);
glVertex2i(128, 102);
glVertex2i(128, 110);
glVertex2i(126, 113);
glVertex2i(124, 111);
glVertex2i(100, 80);
glVertex2i(90, 70);
glVertex2i(90, 65);
glVertex2i(100, 73);
glVertex2i(126, 100);
glEnd();
glPopMatrix();

// Adding a code to rotate the tubewell pump handle
if (angle >= 30.0f) {
   cw = 1;
}
else if (angle <= 0.0f) {
   cw = 0;
}
if (cw == 0) {
   angle += 0.1;
}
else if (cw == 1) {
   angle -= 0.1;
}
```

```
// Adding water drops from the tube well
glPushMatrix();
glTranslatef(0, dy, 0);
glColor3ub(14, 135, 204);
glRecti(137, 55, 138, 50);
glPopMatrix();

dy -= 0.5;
if (dy < -30)
   dy = 10;

glPushMatrix();
glTranslatef(83, 10, 0);
glScalef(0.4,0.4,0.4);
glColor3ub(30, 144, 255);
sun(138, -20);
glPopMatrix();

glBegin(GL_POLYGON);  // Part 5
glColor3ub(255, 215, 0);
glVertex2i(130, 70);
glVertex2i(140, 70);
glVertex2i(140, 50);
glVertex2i(136, 50);
glVertex2i(136, 60);
glVertex2i(130, 60);
glVertex2i(130, 70);
glEnd();

glColor3ub(255, 215, 0); // Part 6
glRecti(123, 29, 127, 20);

glColor3ub(139, 128, 0); // Part 7
glRecti(118, 22, 132, 14);

// The first house -- Red Roof

glBegin(GL_POLYGON);  // First part of the roof
glColor3ub(128, 0, 0); // Setting a dark red colour for this part
glVertex2i(-58, 115);
glVertex2i(-75, 145);
glVertex2i(-115, 150);
glVertex2i(-90, 100);
glVertex2i(-62, 100);
glVertex2i(-58, 115);
glEnd();
```

```
    glBegin(GL_POLYGON);  // Second part of the roof
    glColor3ub(120, 0, 0); // Seting a darker shade of red in order for the two parts of the roof to be visually distinguishable
    glVertex2i(-115, 150);
    glVertex2i(-130, 100);
    glVertex2i(-120, 100);
    glVertex2i(-108, 137);
    glEnd();

    glBegin(GL_POLYGON);  // Front view of the house
    glColor3ub(46, 139, 87);
    glVertex2i(-108, 137);
    glVertex2i(-120, 100);
    glVertex2i(-120, 45);
    glVertex2i(-90, 40);
    glVertex2i(-90, 100);
    glVertex2i(-108, 137);
    glEnd();

    glBegin(GL_POLYGON);  // Side view of the house
    glColor3ub(143, 188, 143);
    glVertex2i(-90, 40);
    glVertex2i(-60, 45);
    glVertex2i(-60, 100);
    glVertex2i(-90, 100);
    glEnd();

    glColor3ub(120, 0, 0); // Adding a maroon door to the house
    glRecti(-75, 80, -65, 60);

    glColor3ub(120, 0, 0); // Adding a window to the house
    glRecti(-110, 90, -100, 70);

    glBegin(GL_POLYGON);  // Front view of the base of the house
    glColor3ub(50, 50, 50);
    glVertex2i(-90, 40);
    glVertex2i(-123, 45);
    glVertex2i(-123, 35);
    glVertex2i(-90, 30);
    glVertex2i(-90, 40);
    glEnd();

    glBegin(GL_POLYGON);  // Side view of the base of the house
    glColor3ub(80, 80, 80);
    glVertex2i(-90, 40);
    glVertex2i(-55, 45);
    glVertex2i(-55, 35);
    glVertex2i(-90, 30);
    glVertex2i(-90, 40);
    glEnd();
```

```
// The second house -- Blue Roof

glBegin(GL_POLYGON); // Front view of the roof
glColor3ub(25, 25, 150);
glVertex2i(-50, 140);
glVertex2i(0, 149);
glVertex2i(-12, 88);
glVertex2i(-65, 89);
glVertex2i(-50, 140);
glEnd();

glBegin(GL_POLYGON);  // Front view of the house
glColor3ub(70, 130, 180);
glVertex2i(-60, 90);
glVertex2i(-60, 30);
glVertex2i(-10, 25);
glVertex2i(-10, 95);
glEnd();

glColor3ub(25, 25, 112); // Adding a Window
glRecti(-45, 70, -33, 45);

glBegin(GL_POLYGON);   // Side view of the house
glColor3ub(95, 158, 160);
glVertex2i(-10, 25);
glVertex2i(18, 35);
glVertex2i(18, 100);
glVertex2i(0, 148);
glVertex2i(-10, 100);
glVertex2i(-10, 25);
glEnd();

glBegin(GL_POLYGON); // Side view of the roof
glColor3ub(25, 25, 112);
glVertex2i(-1, 150);
glVertex2i(20, 100);
glVertex2i(17, 90);
glVertex2i(-4, 140);
glVertex2i(-1, 150);
glEnd();

glBegin(GL_POLYGON);  // Adding a Door
glColor3ub(25, 25, 112);
glVertex2i(0, 70);
glVertex2i(10, 73);
glVertex2i(10, 45);
glVertex2i(0, 43);
glEnd();
```

```
glBegin(GL_POLYGON);  // Front view of the base of the house
glColor3ub(50, 50, 50);
glVertex2i(-10, 25);
glVertex2i(-62, 30);
glVertex2i(-62, 20);
glVertex2i(-10, 15);
glVertex2i(-10, 25);
glEnd();

glBegin(GL_POLYGON);  // Side view of the base of the house
glColor3ub(80, 80, 80);//
glVertex2i(-10, 25);
glVertex2i(-10, 15);
glVertex2i(20, 27);
glVertex2i(20, 37);
glVertex2i(-10, 25);
glEnd();

// The third house -- Green Roof
glPushMatrix();
glTranslatef(-130, -40, 0);
glBegin(GL_POLYGON); // Front view of the roof
glColor3ub(1, 50, 32);
glVertex2i(-50, 140);
glVertex2i(0, 149);
glVertex2i(-12, 88);
glVertex2i(-65, 89);
glVertex2i(-50, 140);
glEnd();

glBegin(GL_POLYGON);  // Front view of the house
glColor3ub(255, 127, 127);
glVertex2i(-60, 90);
glVertex2i(-60, 30);
glVertex2i(-10, 25);
glVertex2i(-10, 95);
glEnd();

glColor3ub(1, 50, 32); // Adding a Window
glRecti(-45, 70, -33, 45);

glBegin(GL_POLYGON);   // Side view of the house
glColor3ub(255, 100, 100);
glVertex2i(-10, 25);
glVertex2i(18, 35);
glVertex2i(18, 100);
glVertex2i(0, 148);
glVertex2i(-10, 100);
glVertex2i(-10, 25);
glEnd();
```

```
glBegin(GL_POLYGON); // Side view of the roof
glColor3ub(1, 50, 32);
glVertex2i(-1, 150);
glVertex2i(20, 100);
glVertex2i(17, 90);
glVertex2i(-4, 140);
glVertex2i(-1, 150);
glEnd();

glBegin(GL_POLYGON);  // Adding a Door
glColor3ub(1, 50, 32);
glVertex2i(0, 70);
glVertex2i(10, 73);
glVertex2i(10, 45);
glVertex2i(0, 43);
glEnd();

glBegin(GL_POLYGON);  // Front view of the base of the house
glColor3ub(50, 50, 50);
glVertex2i(-10, 25);
glVertex2i(-62, 30);
glVertex2i(-62, 20);
glVertex2i(-10, 15);
glVertex2i(-10, 25);
glEnd();

glBegin(GL_POLYGON);  // Side view of the base of the house
glColor3ub(80, 80, 80);//
glVertex2i(-10, 25);
glVertex2i(-10, 15);
glVertex2i(20, 27);
glVertex2i(20, 37);
glVertex2i(-10, 25);
glEnd();
glPopMatrix();

// The Fourth house -- Black Roof
glPushMatrix();
glTranslatef(-30, -50, 0);
glBegin(GL_POLYGON); // Front view of the roof
glColor3ub(60, 60, 60);
glVertex2i(-50, 140);
glVertex2i(0, 149);
glVertex2i(-12, 88);
glVertex2i(-65, 89);
glVertex2i(-50, 140);
glEnd();
```

```
glBegin(GL_POLYGON);  // Front view of the house
glColor3ub(255, 253, 208);
glVertex2i(-60, 90);
glVertex2i(-60, 30);
glVertex2i(-10, 25);
glVertex2i(-10, 95);
glEnd();

glColor3ub(1, 50, 32); // Adding a Window
glRecti(-45, 70, -33, 45);

glBegin(GL_POLYGON);   // Side view of the house
glColor3ub(235, 233, 188);
glVertex2i(-10, 25);
glVertex2i(18, 35);
glVertex2i(18, 100);
glVertex2i(0, 148);
glVertex2i(-10, 100);
glVertex2i(-10, 25);
glEnd();

glBegin(GL_POLYGON); // Side view of the roof
glColor3ub(70, 70, 70);
glVertex2i(-1, 150);
glVertex2i(20, 100);
glVertex2i(17, 90);
glVertex2i(-4, 140);
glVertex2i(-1, 150);
glEnd();

glBegin(GL_POLYGON);  // Adding a Door
glColor3ub(1, 50, 32);
glVertex2i(0, 70);
glVertex2i(10, 73);
glVertex2i(10, 45);
glVertex2i(0, 43);
glEnd();

glBegin(GL_POLYGON);  // Front view of the base of the house
glColor3ub(50, 50, 50);
glVertex2i(-10, 25);
glVertex2i(-62, 30);
glVertex2i(-62, 20);
glVertex2i(-10, 15);
glVertex2i(-10, 25);
glEnd();
```

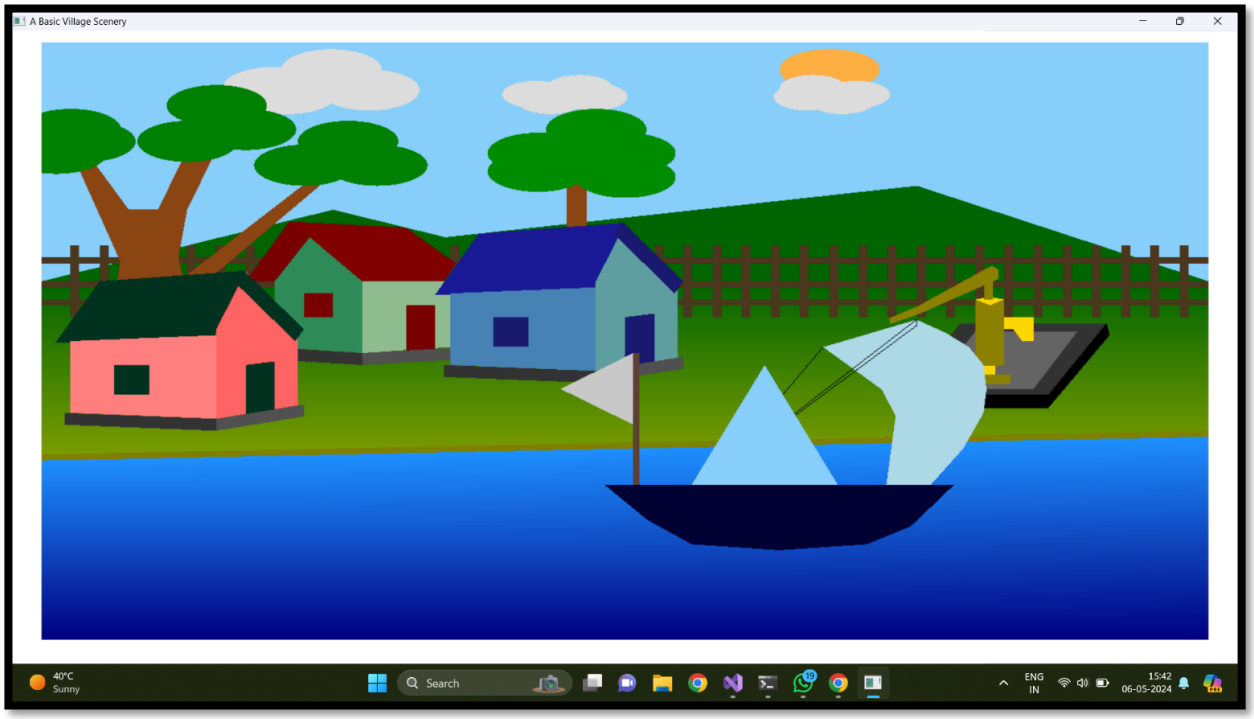
```
glBegin(GL_POLYGON);  // Side view of the base of the house
glColor3ub(80, 80, 80);//
glVertex2i(-10, 25);
glVertex2i(-10, 15);
glVertex2i(20, 27);
glVertex2i(20, 37);
glVertex2i(-10, 25);
glEnd();
glPopMatrix();


// Adding a river below the field
glBegin(GL_POLYGON);
glColor3ub(30, 144, 255);
glVertex2i(-200, -50);
glVertex2i(200, -30);
glColor3ub(0, 0, 128);
glVertex2i(200, -200);
glVertex2i(-200, -200);
glVertex2i(-200, -50);
glEnd();

// Adding a border between the field and the river
glBegin(GL_POLYGON);
glColor3ub(128, 128, 0);
glVertex2i(-200, -45);
glVertex2i(200, -25);
glVertex2i(200, -30);
glVertex2i(-200, -50);
glVertex2i(-200, -45);
glEnd();

glColor3ub(30, 144, 255);
glRecti(135, 15, 143, -40);

// Adding a moving boat on the river, part by part
glPushMatrix();
glTranslatef(bx, 0, 0);

// Adding a front sail to the boat
glBegin(GL_POLYGON);
glColor3ub(173, 216, 230);
glVertex2i(-57, -40);
glVertex2i(-50, -10);
glVertex2i(-49, 10);
glVertex2i(-50, 30);
glVertex2i(-55, 45);
glVertex2i(-63, 57);
glVertex2i(-73, 68);
```

```
glVertex2i(-105, 45);
glVertex2i(-50, -10);
glEnd();
glBegin(GL_POLYGON);
glColor3ub(173, 216, 230);
glVertex2i(-68, -70);
glVertex2i(-57, -40);
glVertex2i(-85, 10);
glVertex2i(-68, -70);
glEnd();
glBegin(GL_POLYGON);
glColor3ub(173, 216, 230);
glVertex2i(-85, -100);
glVertex2i(-68, -70);
glVertex2i(-80, -10);
glVertex2i(-85, -100);
glEnd();

// Making the base of the boat
glBegin(GL_POLYGON);
glColor3f(0.0f, 0.0f, 0.2f);
glVertex2i(-180, -70);
glVertex2i(-165, -100);
glVertex2i(-150, -120);
glVertex2i(-150, -100);
glVertex2i(-180, -70);
glVertex2i(-150, -100);
glVertex2i(-150, -120);
glVertex2i(-120, -125);
glVertex2i(-90, -120);
glVertex2i(-150, -100);
glVertex2i(-85, -100);
glVertex2i(-90, -120);
glVertex2i(-75, -105);
glVertex2i(-60, -70);
glEnd();

// Adding a flag to the boat
glColor3ub(92, 64, 51);
glRecti(-170, 40, -168, -70);

glBegin(GL_POLYGON);
glColor3ub(200, 200, 200);
glVertex2i(-170, 40);
glVertex2i(-195, 10);
glVertex2i(-170, -20);
glEnd();
```

```
    // Adding some connecting ropes for the front sail of the boat
    glBegin(GL_LINE_STRIP);
    glColor3f(0.0f, 0.0f, 0.0f);//Black
    glVertex2i(-142, -62);
    glVertex2i(-73, 68);
    glVertex2i(-73, 63);

    glVertex2i(-142, -62);
    glVertex2i(-105, 45);
    glEnd();

    // Adding a triangular sail to the boat

    glBegin(GL_POLYGON);
    glColor3ub(135, 206, 250);
    glVertex2i(-100,-70);
    glVertex2i(-125,30);
    glVertex2i(-150,-70);
    glEnd();

    // Finishing the translation matrix portion for the boat
    glPopMatrix();
    bx += .07;
    if (bx > 270)
        bx = -180;

    glutPostRedisplay(); // Marks the current window as needing to be redisplayed
    glColor3ub(255, 255, 255);
    glRecti(-210, 310, -200, -210);
    glRecti(200, 310, 210, -210);

    glFlush(); // Empties all the required buffers in the program back-end, causing all the issued
commands to be executed quickly
}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv); // To initialize the GL function
    glutInitWindowSize(1200, 800); // To decide the size of the window
    glutInitWindowPosition(10, 10); // Initial position of the window on the screen
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE); // Determining the OPENGL display mode
for the created top-level or sub-top windows
    glutCreateWindow("A Basic Village Scenery"); // The name under which the created window will
appear
    init(); // Calling our initial function to set up the clear buffer colour and the orthographic scaling of
the drawing
    glutDisplayFunc(display); // Displaying our drawing on the screen
    glutMainLoop(); // Creating the basic loop for the drawing to keep displaying on the screen
    return 0;
}
```

SCREENSHOTS OF OUTPUT

BEFORE EVAL



AFTER EVAL