Computer Graphics (UCS505)

Project on <u>Simulation of Movable Bridge</u>

Submitted By

Akul Vaishnavi 10210387

Paridhi Maheshwari 102103491

Akshat Mathur 102103492

B.E. Third Year – COE

Submitted To:

Dr. Jasmine Kaur

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Introduction to Project

The project **"Simulation of Movable Bridge"** is used to demonstrate the use of OpenGL functions. OpenGL can be used for interaction with the hardware. Then its result can be seen on the screen. Our objective is to develop a simple project to demonstrate the simulation of a movable bridge, and incorporating various Transformations which we have learned in our practical labs in the subject. We need to press the 'p' or 'P' to start the simulation. We can press the 's' or 'S' key to stop the simulation in between.

Initially both the ship and train were at halt. When 'P' or 'p' is pressed then the train signal turns red and the bridge starts moving upwards. When the bridge reaches the top, the ship starts moving and crosses the bridge. Now the bridge comes back to initial position and the train signal turns green, signaling the train to cross the bridge. After crossing the bridge, the train signal turns red. We also have an element of a windmill in the simulation, whose wings start rotating as soon as we start the simulation – but also pause when we pause the simulation. We can press 'S' or 's' any time to pause the simulation. There is also a feature of providing stars in the night sky, which sparkle in random locations, and can be toggled on/off using the 'r' or 'R' key – irrespective of the status of the simulation.

Computer Graphics Concepts Used

OpenGL is a cross language, cross platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit to achieve hardware-accelerated rendering. As a software interface for graphics hardware, OpenGL's main purpose is to render two and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.
To draw a primitive on the screen first we call glBegin, specifying what kind of primitive it is that you want to draw in the mode parameter of glBegin, and then list all vertices one by one (by sequentially calling glVertex3f) and finally call glEnd to let OpenGL know that we're done drawing a primitive.

2D-Transformations:

Transformation is a process of modifying and re-positioning the existing graphics. 2D Transformations take place in a two-dimensional plane. Transformations are helpful in changing the position, size, orientation, shape etc. of the object.

Translation Function: A translation process moves every point a constant distance in a specified direction. It can be described as a rigid motion. A translation can also be interpreted as the addition of a constant vector to every point, or as shifting the origin of the coordinate system. You can translate a point in 2D by adding the translation coordinate (tx, ty) to the original coordinate X, Y to get the new coordinate X′, Y′.

Scaling: It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors, i.e., Sx in x direction, Sy in y-direction. If the original position is x and y. Scaling factors are Sx and Sy then the value of coordinates after scaling will be x1 and y1.

Animation: It is a method, where objects are manipulated to appear as moving, without the user input. It requires a form of infinite loop, which is usually achieved by a set of functions or class methods that describe the changes to each object in a small unit of time- called update or move.

RGB: The RGB Color Model is used for color representation. It is a color coordinate system having three primary colors (red, green and blue), each primary color having its intensity value ranging from 0 to 1. Mixing these three primary colors at varying intensities produces a variety of colors.

Rotation: This transformation is used to adjust orientation of objects and pivots them around a fixed point, determined by an angle of rotation and a center. Object rotation is achieved by specifying the angle of rotation and the axis around which the object rotates. This is done using transformation matrices or functions like glRotated(). The rotation operation modifies the object's vertices, altering its orientation relative to the coordinate system.

User Defined Functions

Function Name	Description
void ocean()	To set the coordinates of the waterbody.
void windmill()	To draw and set the coordinates of windmill
void star()	To make blinking stars in the night sky
void update()	To update the movement values of bridge, ship, train and water waves.
void update_windmill()	To update the rotational movement of the windmill
void keyboard_func()	To get user input through the keyboard.
void call1(), void call2(), void call3()	To control the movements of the bridge, ship, train respectively.
void track_base_front()	To set the coordinates of the front part of the railway track base.
void track_base()	To set the coordinates of the railway track base.
void gate_pillar()	To set the coordinates of the bridge gate supporting pillars and the bridge gate pillar supporters.
void bridge()	To set the coordinates of the bridge gate, the railway track on the bridge, and the control room on the bridge.
void rail_track()	To set the coordinates of the railway track.
void ship()	To set the coordinates of the ship and its components.
void train()	To set the coordinates of the train and its components.
void signal()	To set the coordinates of the train signal.
void light()	To set the colors of the train signal according to different conditions.

void reshape()	Called through glutReshapeFunc() which sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever anoverlay for the window is established.

Screenshots

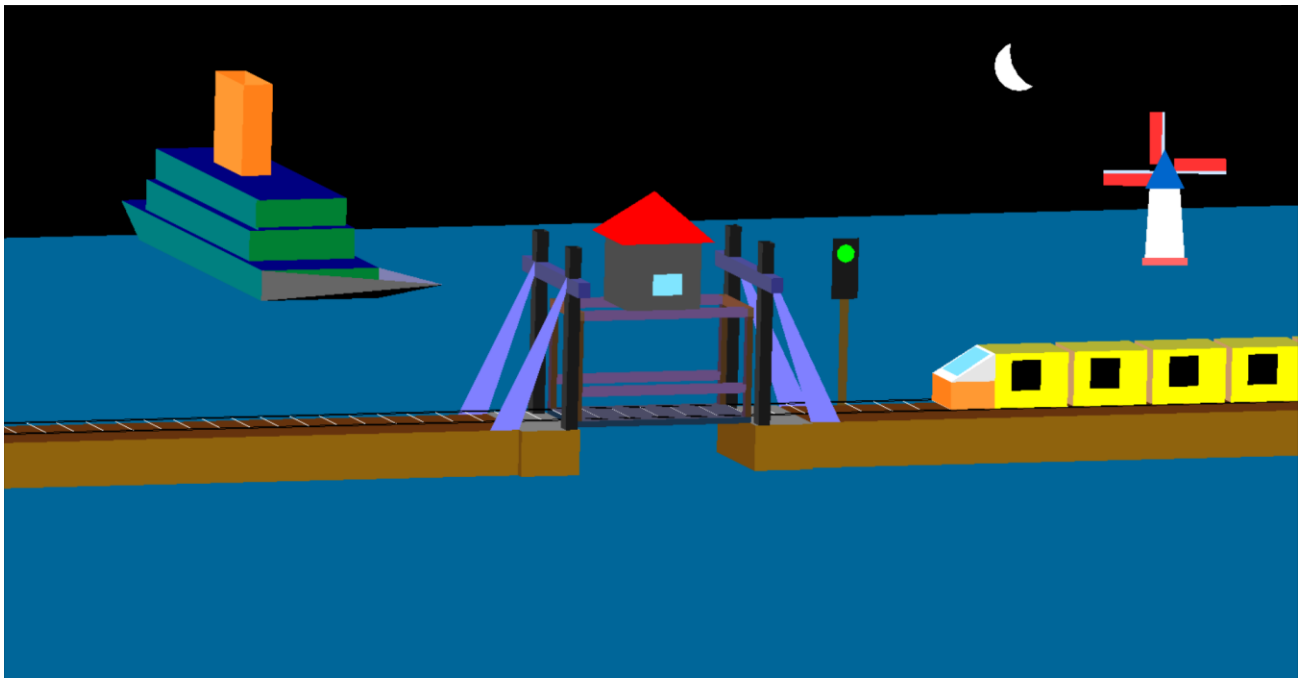

Fig. 1 - Initial Position

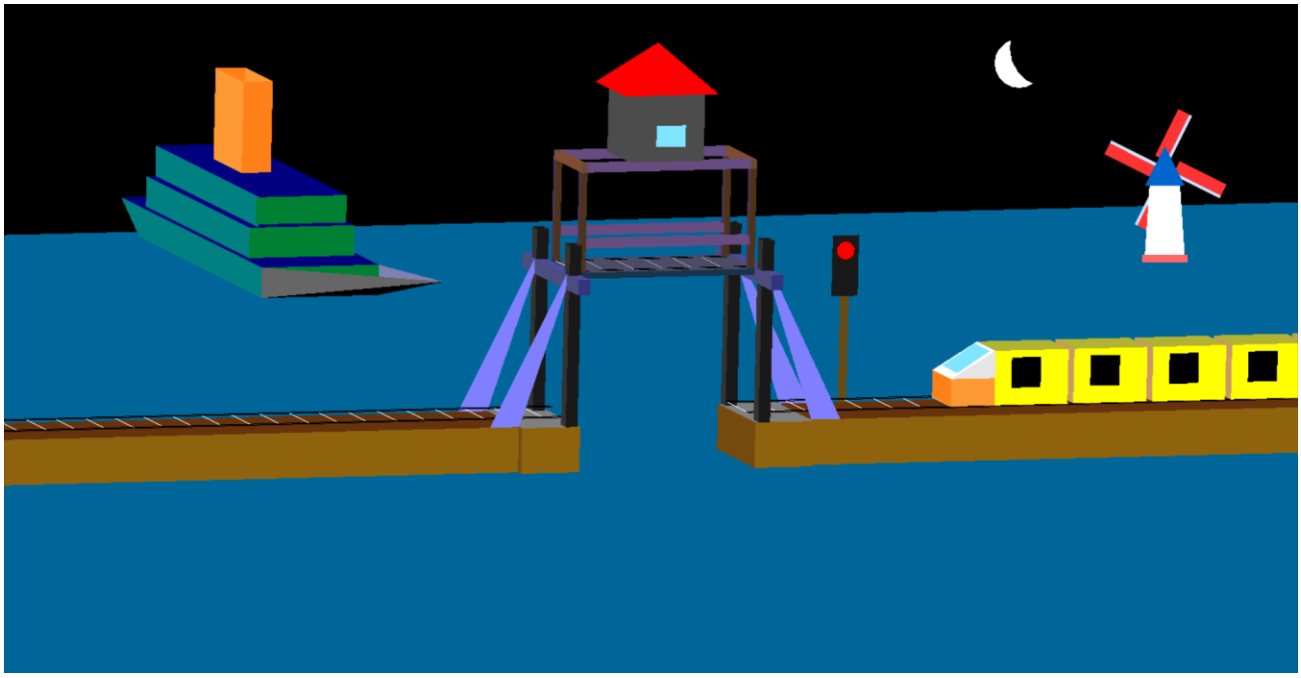

Fig. 2 - Bridge Rises for ship to cross and signal turns red.

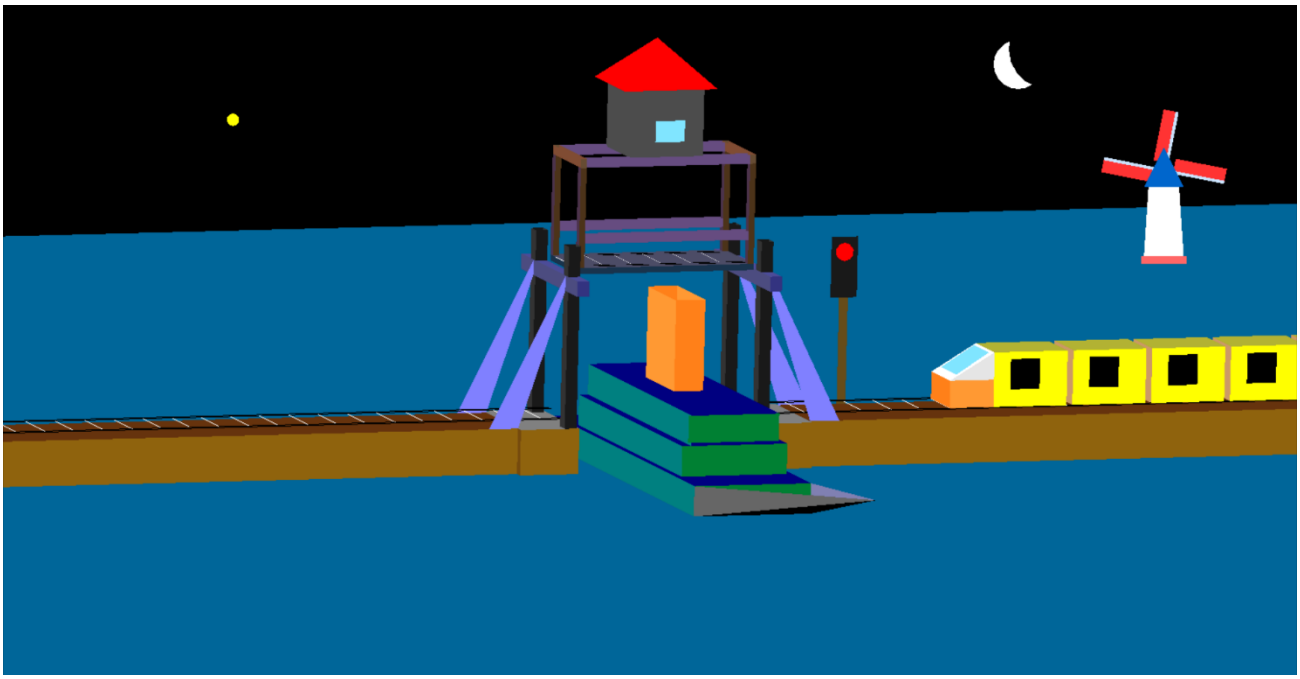

Fig. 3 - Ship is under the risen bridge.

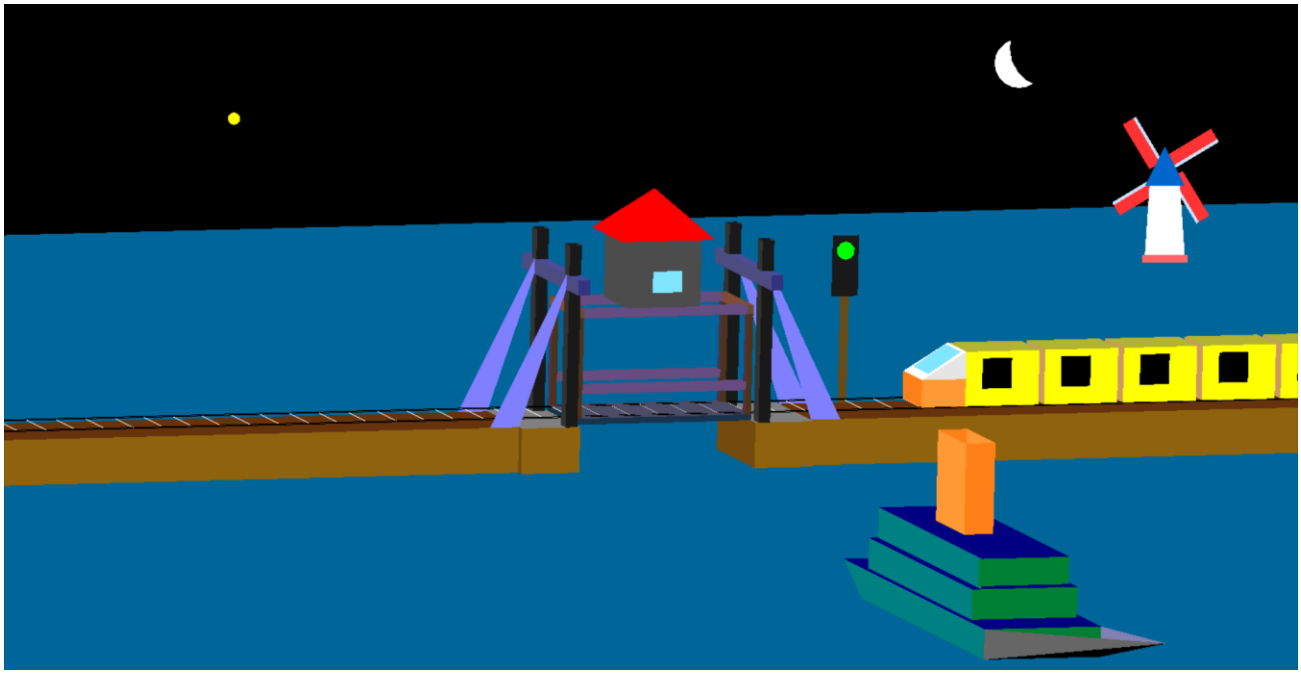

Fig. 4 - Ship has crossed, bridge has come back down and signal is now green.

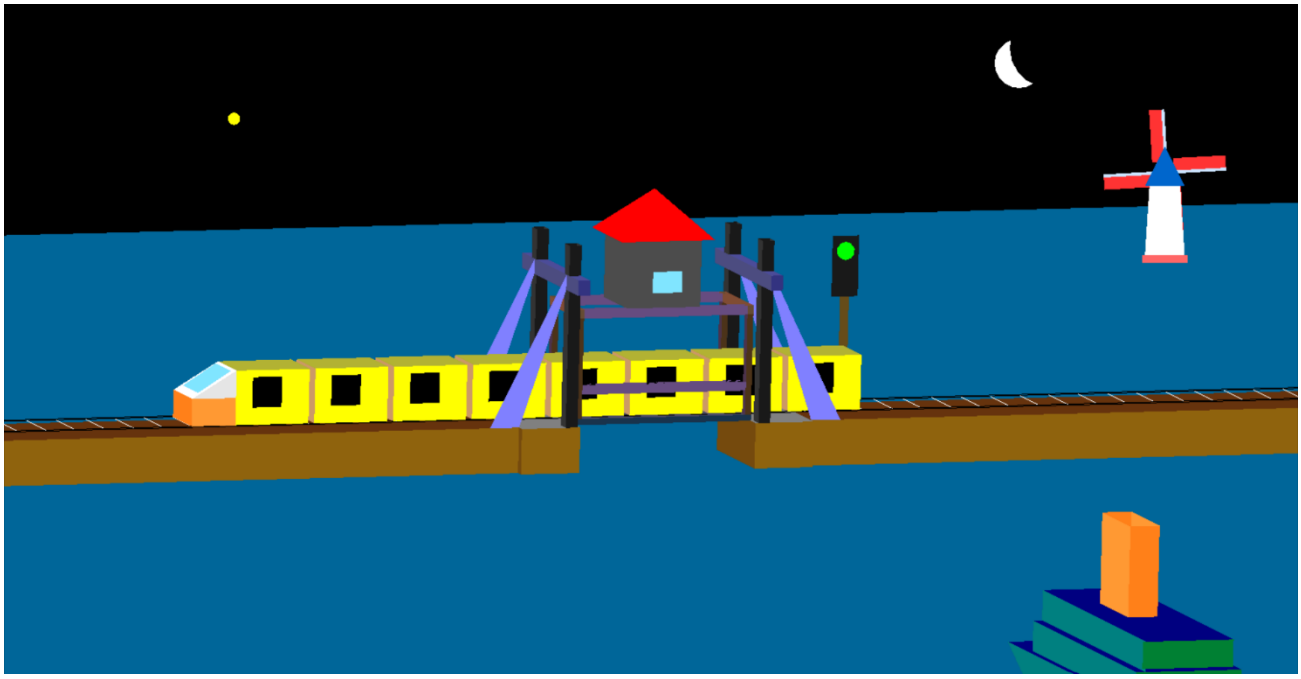

Fig 5. Train is crossing through the bridge.

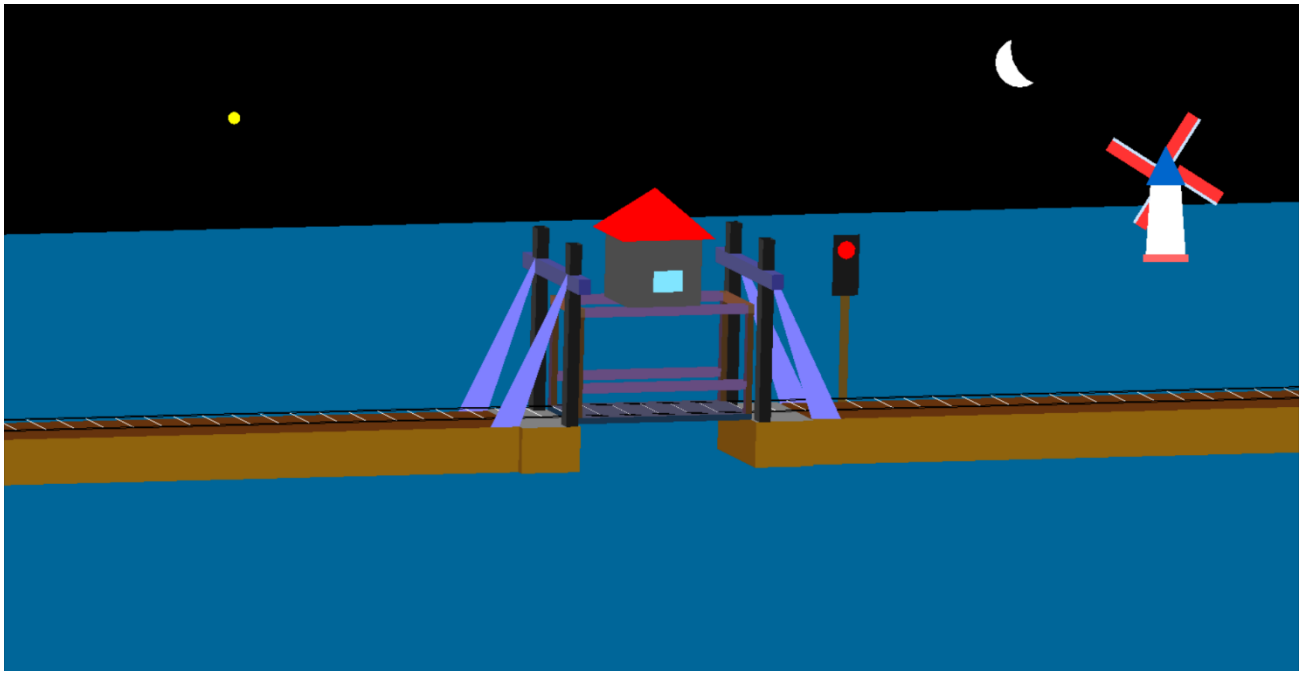

Fig 6. Both Ship and Train have crossed, Light turns red.

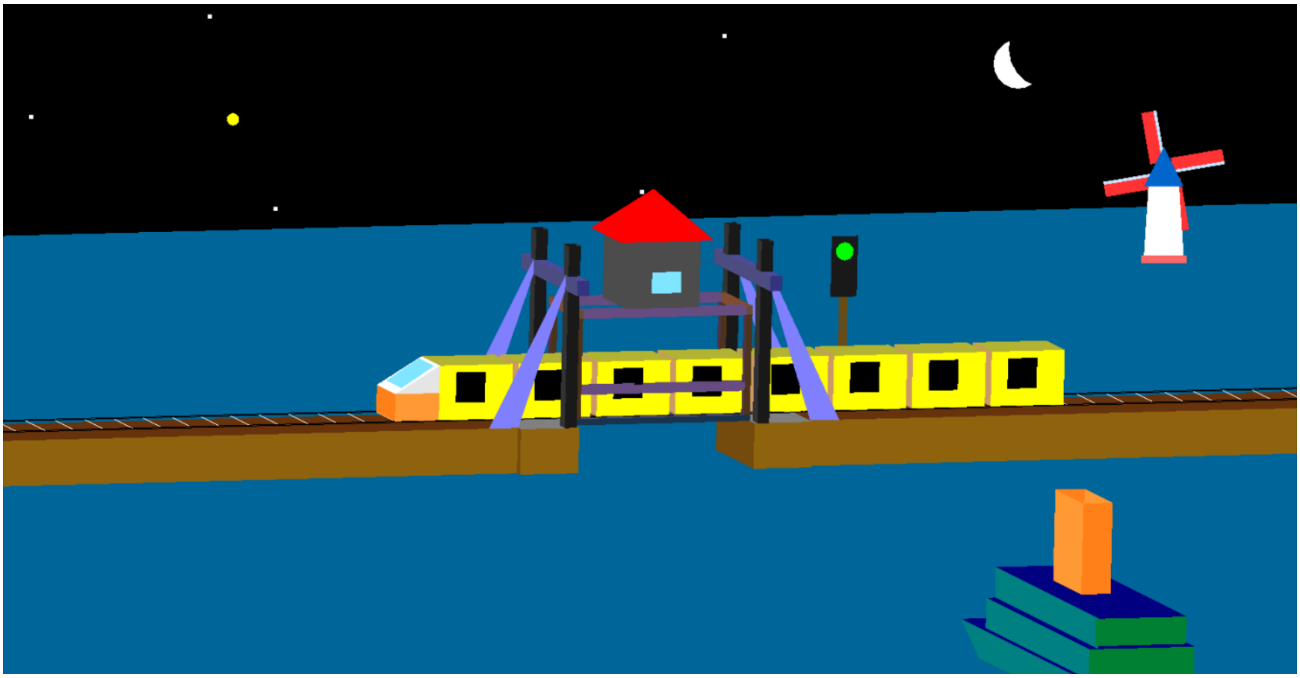

Fig 7. Sparkling stars during the whole process (Toggable)

Code

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <windows.h>
#include <GL/glut.h>
#endif
#include <iostream>
#define GL_SILENCE_DEPRECATION
#include <iostream>
#include <cmath>
#include <string.h>
#include <stdlib.h>

const float PI = M_PI;
int i, flag = 0, flagb = 1, flags = 0, flagt = 0, flagw =1, flagp = 0, flagx = 0,
flagr = 0;

float a = 0.0f, b = 0.0f, c = 0.0f, m = 0.0f, n = 0.0f, o = 0.0f, p = 0.0f,q = 0.0f,
r = 0.0f, x = 0.0f, y = 0.0f, z = 0.0f, a1= 0.0,a2 = 0.0, a3 = 0.0;
float windmill_angle = 0.0f;
float radius, twicePi, triangleAmount; // for circle
int frameNumber = 1;
bool is_star_active = false; // Variable to control star effect

void ocean()
{
glBegin(GL_QUADS);
glColor3f(0.0, 0.4, 0.6);
glVertex3f(-5.0, -0.415, 5.0);
glVertex3f(5.0, -0.415, 5.0);
glVertex3f(5.0, -0.415, -5.0);
glVertex3f(-5.0, -0.415, -5.0);
glEnd();
}
void star() {
    if (is_star_active) {
        glPointSize(5.0); // Set the size of the stars

        glBegin(GL_POINTS);
        glColor3f(1.0, 1.0, 1.0); // Set color for stars

        // Generate random star positions within the scene
        for (int i = 0; i < 100; i++) {
            float x = static_cast <float> (rand()) / static_cast <float> (RAND_MAX)
* 10 - 5; // x-coordinate
            float y = static_cast <float> (rand()) / static_cast <float> (RAND_MAX)
* 5; // y-coordinate
```

```
            float z = static_cast <float> (rand()) / static_cast <float> (RAND_MAX)
* 10 - 5; // z-coordinate

            glVertex3f(x, y+0.415, z); // Draw star
        }

        glEnd();
    }
}

void windmill(){
    // Load identity matrix to reset transformations
    glLoadIdentity();
    // Set the current matrix mode to modelview
    glMatrixMode(GL_MODELVIEW);
    // Translate to position the windmill base
    glTranslatef(2.4, 0.0, 0.0);
    // Draw the base of the windmill
    glColor3ub(255, 102, 102); //base
    glBegin(GL_POLYGON);
        glVertex2f(-0.80f, 0.275f);
        glVertex2f(-0.80f, 0.25f);
        glVertex2f(-0.65f, 0.25f);
        glVertex2f(-0.65f, 0.275f);
    glEnd();

    // Draw the body of the windmill
    glColor3ub(255, 255, 255); //body
    glBegin(GL_POLYGON);
        glVertex2f(-0.775f, 0.5f);
        glVertex2f(-0.79f, 0.275f);
        glVertex2f(-0.66f, 0.275f);
        glVertex2f(-0.675f, 0.5f);
    glEnd();

    // Draw the door of the windmill
    glColor3ub(51, 153, 255); //door
    glBegin(GL_POLYGON);
        glVertex2f(-0.71f, 0.35f);
        glVertex2f(-0.71f, 0.275f);
        glVertex2f(-0.74f, 0.275f);
        glVertex2f(-0.74f, 0.35f);
    glEnd();

    // Draw the window of the windmill
    glColor3ub(51, 153, 255); //window
    glBegin(GL_POLYGON);
        glVertex2f(-0.71f, 0.45f);
        glVertex2f(-0.71f, 0.4f);
        glVertex2f(-0.74f, 0.4f);
```

```
        glVertex2f(-0.74f, 0.45f);
    glEnd();

    // Draw the head of the windmill
    glColor3ub(00, 102, 204); //head
    glBegin(GL_TRIANGLES);
        glVertex2f(-0.79f, 0.5f);
        glVertex2f(-0.725f, 0.63f);
        glVertex2f(-0.66f, 0.5f);
    glEnd();

    // Translate and rotate the windmill blades
    glTranslatef(-0.725f, 0.55f, 0.0f);
    glRotated(frameNumber * (-180.0/500), 0, 0, 1);

    // Draw the first wing of the windmill
    glColor3ub(204, 229, 255);//wing1
    glBegin(GL_POLYGON);
        glVertex2f(0.00f, 0.01f);
        glVertex2f(0.00f, 0.00f);
        glVertex2f(0.20f, 0.00f);
        glVertex2f(0.20f, 0.01f);
    glEnd();

    // Draw the blade of the first wing
    glColor3ub(255, 51, 51);
    glBegin(GL_POLYGON);
        glVertex2f(0.03f, 0.05f);
        glVertex2f(0.03f, 0.01f);
        glVertex2f(0.20f, 0.01f);
        glVertex2f(0.20f, 0.05f);
    glEnd();

    // Draw the second wing of the windmill
    glColor3ub(204, 229, 255); //wing2
    glBegin(GL_POLYGON);
        glVertex2f(0.00f, 0.01f);
        glVertex2f(0.00f, 0.00f);
        glVertex2f(-0.20f, 0.00f);
        glVertex2f(-0.20f, 0.01f);
    glEnd();

    // Draw the blade of the second wing
    glColor3ub(255, 51, 51);
    glBegin(GL_POLYGON);
        glVertex2f(-0.03f, -0.04f);
        glVertex2f(-0.03f, -0.00f);
        glVertex2f(-0.20f, -0.00f);
        glVertex2f(-0.20f, -0.04f);
    glEnd();
```

```
// Rotate the coordinate system to draw the third and fourth wings
glRotatef(90,0.0f, 0.0f, 1.0f);
// Draw the third wing of the windmill
glColor3ub(204, 229, 255); //wing3
glBegin(GL_POLYGON);
    glVertex2f(0.00f, 0.01f);
    glVertex2f(0.00f, 0.00f);
    glVertex2f(0.20f, 0.00f);
    glVertex2f(0.20f, 0.01f);
glEnd();

// Draw the blade of the third wing
glColor3ub(255, 51, 51);
glBegin(GL_POLYGON);
    glVertex2f(0.03f, 0.05f);
    glVertex2f(0.03f, 0.01f);
    glVertex2f(0.20f, 0.01f);
    glVertex2f(0.20f, 0.05f);
glEnd();

// Draw the fourth wing of the windmill
glColor3ub(204, 229, 255); //wing4
glBegin(GL_POLYGON);
    glVertex2f(0.00f, 0.01f);
    glVertex2f(0.00f, 0.00f);
    glVertex2f(-0.20f, 0.00f);
    glVertex2f(-0.20f, 0.01f);
glEnd();

// Draw the blade of the fourth wing
glColor3ub(255, 51, 51);
glBegin(GL_POLYGON);
    glVertex2f(-0.03f, -0.04f);
    glVertex2f(-0.03f, -0.00f);
    glVertex2f(-0.20f, -0.00f);
    glVertex2f(-0.20f, -0.04f);
glEnd();

// Draw the circle representing the windmill head
x=-0.0f; y=0.0f; radius =.02f;
triangleAmount = 50;
twicePi = 2.0f * PI;
glColor3ub(255, 51, 51);
glBegin(GL_TRIANGLE_FAN);
    glVertex2f(x, y);
    for(int i = 0; i <= triangleAmount;i++) {
        glVertex2f(
            x + (radius * cos(i *  twicePi / triangleAmount)),
            y + (radius * sin(i * twicePi / triangleAmount))
```

```
            );
        }
    glEnd();
    // Reset transformations
    glLoadIdentity();
}


void track_base_front()
{
for (float i = 0.0; i < 1.0; i += 0.8)
{
glBegin(GL_QUADS);
glColor3f(0.56, 0.39, 0.05); // front
glVertex3f(-0.5 + i, -0.4, 0.2);
glVertex3f(-0.3 + i, -0.4, 0.2);
glVertex3f(-0.3 + i, -0.25, 0.2);
glVertex3f(-0.5 + i, -0.25, 0.2);

glColor3f(0.46, 0.29, 0.05); // right front
glVertex3f(-0.5 + i, -0.4, 0.2);
glVertex3f(-0.5 + i, -0.4, -0.2);
glVertex3f(-0.5 + i, -0.25, -0.2);
glVertex3f(-0.5 + i, -0.25, 0.2);

glColor3f(0.5, 0.5, 0.5); // top
glVertex3f(-0.5 + i, -0.25, 0.2);
glVertex3f(-0.3 + i, -0.25, 0.2);
glVertex3f(-0.3 + i, -0.25, -0.2);
glVertex3f(-0.5 + i, -0.25, -0.2);
glEnd();
}
}

void track_base()
{
for (float i = 0.0; i < 4.0; i += 3.5)
{
glBegin(GL_QUADS);
glColor3f(0.56, 0.39, 0.05);
glVertex3f(-3.0 + i, -0.4, 0.17); // side front
glVertex3f(-0.5 + i, -0.4, 0.17);
glVertex3f(-0.5 + i, -0.25, 0.17);
glVertex3f(-3.0 + i, -0.25, 0.17);

glColor3f(0.4, 0.2, 0.05); // top
glVertex3f(-3.0 + i, -0.25, 0.17);
glVertex3f(-0.5 + i, -0.25, 0.17);
glVertex3f(-0.5 + i, -0.25, -0.17);
glVertex3f(-3.0 + i, -0.25, -0.17);
glEnd();
```

```
}
}


void gate_pillar()
{
glBegin(GL_QUADS);
// front left pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(-0.35,-0.25, 0.2);
glVertex3f(-0.31,-0.25, 0.2);
glVertex3f(-0.31, 0.2, 0.2);
glVertex3f(-0.35, 0.2, 0.2);
glColor3f(0.2, 0.2, 0.2); // side
glVertex3f(-0.35,-0.25, 0.2);
glVertex3f(-0.35,-0.25, 0.15);
glVertex3f(-0.35, 0.2, 0.15);
glVertex3f(-0.35, 0.2, 0.2);

// back left pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(-0.35,-0.25,-0.15);
glVertex3f(-0.31,-0.25,-0.15);
glVertex3f(-0.31, 0.2,-0.15);
glVertex3f(-0.35, 0.2,-0.15);
glColor3f(0.2, 0.2, 0.2); // side
glVertex3f(-0.35,-0.25,-0.2);
glVertex3f(-0.35,-0.25,-0.15);
glVertex3f(-0.35, 0.2,-0.15);
glVertex3f(-0.35, 0.2,-0.2);

// top bar left
glColor3f(0.2, 0.2, 0.5); // front
glVertex3f(-0.35, 0.2, 0.3);
glVertex3f(-0.31, 0.2, 0.3);
glVertex3f(-0.31, 0.25, 0.3);
glVertex3f(-0.35, 0.25, 0.3);
glColor3f(0.3, 0.3, 0.5); // side
glVertex3f(-0.35, 0.2, 0.3);
glVertex3f(-0.35, 0.2,-0.3);
glVertex3f(-0.35, 0.25,-0.3);
glVertex3f(-0.35, 0.25, 0.3);
glColor3f(0.3, 0.3, 0.6); // top
glVertex3f(-0.35, 0.25, 0.3);
glVertex3f(-0.31, 0.25, 0.3);
glVertex3f(-0.31, 0.25,-0.3);
glVertex3f(-0.35, 0.25,-0.3);

// front left pillar top pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(-0.35, 0.25, 0.2);
```

```
glVertex3f(-0.31, 0.25, 0.2);
glVertex3f(-0.31, 0.35, 0.2);
glVertex3f(-0.35, 0.35, 0.2);
glColor3f(0.2, 0.2, 0.2); // left
glVertex3f(-0.35, 0.25, 0.2);
glVertex3f(-0.35, 0.25, 0.15);
glVertex3f(-0.35, 0.35, 0.15);
glVertex3f(-0.35, 0.35, 0.2);
glColor3f(0.1, 0.1, 0.1); // back
glVertex3f(-0.35, 0.25, 0.15);
glVertex3f(-0.31, 0.25, 0.15);
glVertex3f(-0.31, 0.35, 0.15);
glVertex3f(-0.35, 0.35, 0.15);
glColor3f(0.2, 0.2, 0.2); // right
glVertex3f(-0.31, 0.25, 0.2);
glVertex3f(-0.31, 0.25, 0.15);
glVertex3f(-0.31, 0.35, 0.15);
glVertex3f(-0.31, 0.35, 0.2);

// back left pillar top pillar
glColor3f(0.1, 0.1, 0.1); // back
glVertex3f(-0.35, 0.25,-0.2);
glVertex3f(-0.31, 0.25,-0.2);
glVertex3f(-0.31, 0.35,-0.2);
glVertex3f(-0.35, 0.35,-0.2);
glColor3f(0.2, 0.2, 0.2); // left
glVertex3f(-0.35, 0.25,-0.2);
glVertex3f(-0.35, 0.25,-0.15);
glVertex3f(-0.35, 0.35,-0.15);
glVertex3f(-0.35, 0.35,-0.2);
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(-0.35, 0.25,-0.15);
glVertex3f(-0.31, 0.25,-0.15);
glVertex3f(-0.31, 0.35,-0.15);
glVertex3f(-0.35, 0.35,-0.15);
glColor3f(0.2, 0.2, 0.2); // right
glVertex3f(-0.31, 0.25,-0.2);
glVertex3f(-0.31, 0.25,-0.15);
glVertex3f(-0.31, 0.35,-0.15);
glVertex3f(-0.31, 0.35,-0.2);

// front right pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(0.35,-0.25, 0.2);
glVertex3f(0.31,-0.25, 0.2);
glVertex3f(0.31, 0.2, 0.2);
glVertex3f(0.35, 0.2, 0.2);
glColor3f(0.2, 0.2, 0.2); // side
glVertex3f(0.31,-0.25, 0.2);
glVertex3f(0.31,-0.25, 0.15);
```

```
glVertex3f(0.31, 0.2, 0.15);
glVertex3f(0.31, 0.2, 0.2);

// back right pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(0.35,-0.25,-0.15);
glVertex3f(0.31,-0.25,-0.15);
glVertex3f(0.31, 0.2,-0.15);
glVertex3f(0.35, 0.2,-0.15);
glColor3f(0.2, 0.2, 0.2); // side
glVertex3f(0.31,-0.25,-0.2);
glVertex3f(0.31,-0.25,-0.15);
glVertex3f(0.31, 0.2,-0.15);
glVertex3f(0.31, 0.2,-0.2);

// top bar right
glColor3f(0.2, 0.2, 0.5); // front
glVertex3f(0.35, 0.2, 0.3);
glVertex3f(0.31, 0.2, 0.3);
glVertex3f(0.31, 0.25, 0.3);
glVertex3f(0.35, 0.25, 0.3);
glColor3f(0.3, 0.3, 0.5); // side
glVertex3f(0.31, 0.2, 0.3);
glVertex3f(0.31, 0.2,-0.3);
glVertex3f(0.31, 0.25,-0.3);
glVertex3f(0.31, 0.25, 0.3);
glColor3f(0.3, 0.3, 0.6); // top
glVertex3f(0.35, 0.25, 0.3);
glVertex3f(0.31, 0.25, 0.3);
glVertex3f(0.31, 0.25,-0.3);
glVertex3f(0.35, 0.25,-0.3);

// front right pillar top pillar
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(0.35, 0.25, 0.2);
glVertex3f(0.31, 0.25, 0.2);
glVertex3f(0.31, 0.35, 0.2);
glVertex3f(0.35, 0.35, 0.2);
glColor3f(0.2, 0.2, 0.2); // left
glVertex3f(0.35, 0.25, 0.2);
glVertex3f(0.35, 0.25, 0.15);
glVertex3f(0.35, 0.35, 0.15);
glVertex3f(0.35, 0.35, 0.2);
glColor3f(0.1, 0.1, 0.1); // back
glVertex3f(0.35, 0.25, 0.15);
glVertex3f(0.31, 0.25, 0.15);
glVertex3f(0.31, 0.35, 0.15);
glVertex3f(0.35, 0.35, 0.15);
glColor3f(0.2, 0.2, 0.2); // right
glVertex3f(0.31, 0.25, 0.2);
```

```
glVertex3f(0.31, 0.25, 0.15);
glVertex3f(0.31, 0.35, 0.15);
glVertex3f(0.31, 0.35, 0.2);

// back right pillar top pillar
glColor3f(0.1, 0.1, 0.1); // back
glVertex3f(0.35, 0.25,-0.2);
glVertex3f(0.31, 0.25,-0.2);
glVertex3f(0.31, 0.35,-0.2);
glVertex3f(0.35, 0.35,-0.2);
glColor3f(0.2, 0.2, 0.2); // left
glVertex3f(0.35, 0.25,-0.2);
glVertex3f(0.35, 0.25,-0.15);
glVertex3f(0.35, 0.35,-0.15);
glVertex3f(0.35, 0.35,-0.2);
glColor3f(0.1, 0.1, 0.1); // front
glVertex3f(0.35, 0.25,-0.15);
glVertex3f(0.31, 0.25,-0.15);
glVertex3f(0.31, 0.35,-0.15);
glVertex3f(0.35, 0.35,-0.15);
glColor3f(0.2, 0.2, 0.2); // right
glVertex3f(0.31, 0.25,-0.2);
glVertex3f(0.31, 0.25,-0.15);
glVertex3f(0.31, 0.35,-0.15);
glVertex3f(0.31, 0.35,-0.2);

// hangers
glColor3f(0.5, 0.5, 1); // mauve
// front left
glVertex3f(-0.6,-0.25, 0.17);
glVertex3f(-0.5,-0.25, 0.17);
glVertex3f(-0.35, 0.2, 0.17);
glVertex3f(-0.35, 0.25, 0.17);
// back left
glVertex3f(-0.6,-0.25,-0.17);
glVertex3f(-0.5,-0.25,-0.17);
glVertex3f(-0.35, 0.2,-0.17);
glVertex3f(-0.35, 0.25,-0.17);
// front right
glVertex3f(0.6,-0.25, 0.17);
glVertex3f(0.5,-0.25, 0.17);
glVertex3f(0.35, 0.2, 0.17);
glVertex3f(0.35, 0.25, 0.17);
// back right
glVertex3f(0.6,-0.25,-0.17);
glVertex3f(0.5,-0.25,-0.17);
glVertex3f(0.35, 0.2,-0.17);
glVertex3f(0.35, 0.25,-0.17);
glEnd();
}
```

```
void bridge()
{
glBegin(GL_QUADS);
glColor3f(0.1, 0.2, 0.3); // bridge base front
glVertex3f(-0.3,-0.25, 0.15);
glVertex3f(0.3,-0.25, 0.15);
glVertex3f(0.3,-0.23, 0.15);
glVertex3f(-0.3,-0.23, 0.15);
glColor3f(0.1, 0.2, 0.3); // bridge base left
glVertex3f(-0.3,-0.25, 0.15);
glVertex3f(-0.3,-0.25,-0.15);
glVertex3f(-0.3,-0.23,-0.15);
glVertex3f(-0.3,-0.23, 0.15);
glColor3f(0.3, 0.3, 0.4); // bridge base top
glVertex3f(-0.3,-0.23, 0.15);
glVertex3f(0.3,-0.23, 0.15);
glVertex3f(0.3,-0.23,-0.15);
glVertex3f(-0.3,-0.23,-0.15);
glEnd();

// bridge rail lines
glBegin(GL_LINES); // rail lines horizontal
glColor3f(0.0, 0.0, 0.0);
glVertex3f(-0.3,-0.23, 0.12);
glVertex3f(0.3,-0.23, 0.12);
glVertex3f(-0.3,-0.23, 0.1);
glVertex3f(0.3,-0.23, 0.1);
glVertex3f(-0.3,-0.23,-0.12);
glVertex3f(0.3,-0.23,-0.12);
glVertex3f(-0.3,-0.23,-0.1);
glVertex3f(0.3,-0.23,-0.1);
glEnd();
glBegin(GL_LINES); // rail lines vertical
glColor3f(1, 1, 1);
for (float j = 0.0; j <= 0.6; j += 0.1)
{
glVertex3f(-0.3 + j,-0.23, 0.1);
glVertex3f(-0.3 + j,-0.23,-0.1);
}
glEnd();
glBegin(GL_QUADS);
// left front pillar
glColor3f(0.3, 0.2, 0.1); // front
glVertex3f(-0.3,-0.23, 0.15);
glVertex3f(-0.28,-0.23, 0.15);
glVertex3f(-0.28, 0.1, 0.15);
glVertex3f(-0.3, 0.1, 0.15);
glColor3f(0.3, 0.2, 0.3); // side
glVertex3f(-0.3,-0.23, 0.15);
```

```
glVertex3f(-0.3,-0.23, 0.12);
glVertex3f(-0.3, 0.1, 0.12);
glVertex3f(-0.3, 0.1, 0.15);

// right front pillar
glColor3f(0.3, 0.2, 0.1); // front
glVertex3f(0.3,-0.23, 0.15);
glVertex3f(0.28,-0.23, 0.15);
glVertex3f(0.28, 0.1, 0.15);
glVertex3f(0.3, 0.1, 0.15);
glColor3f(0.3, 0.2, 0.3); // side
glVertex3f(0.28,-0.23, 0.15);
glVertex3f(0.28,-0.23, 0.12);
glVertex3f(0.28, 0.1, 0.12);
glVertex3f(0.28, 0.1, 0.15);

// left back pillar
glColor3f(0.3, 0.2, 0.3); // side
glVertex3f(-0.3,-0.23,-0.15);
glVertex3f(-0.3,-0.23,-0.12);
glVertex3f(-0.3, 0.1,-0.12);
glVertex3f(-0.3, 0.1,-0.15);
glColor3f(0.3, 0.2, 0.1); // front
glVertex3f(-0.3,-0.23,-0.12);
glVertex3f(-0.28,-0.23,-0.12);
glVertex3f(-0.28, 0.1,-0.12);
glVertex3f(-0.3, 0.1,-0.12);

// right back pillar
glColor3f(0.3, 0.2, 0.1); // front
glVertex3f(0.3,-0.23,-0.12);
glVertex3f(0.28,-0.23,-0.12);
glVertex3f(0.28, 0.1,-0.12);
glVertex3f(0.3, 0.1,-0.12);
glColor3f(0.3, 0.2, 0.3); // side
glVertex3f(0.28,-0.23,-0.15);
glVertex3f(0.28,-0.23,-0.12);
glVertex3f(0.28, 0.1,-0.12);
glVertex3f(0.28, 0.1,-0.15);

// bridge top left bar
glColor3f(0.4, 0.3, 0.2); // front
glVertex3f(-0.3, 0.1, 0.15);
glVertex3f(-0.28, 0.1, 0.15);
glVertex3f(-0.28, 0.13, 0.15);
glVertex3f(-0.3, 0.13, 0.15);
glColor3f(0.5, 0.3, 0.2); // left
glVertex3f(-0.3, 0.1, 0.15);
glVertex3f(-0.3, 0.1,-0.15);
glVertex3f(-0.3, 0.13,-0.15);
```

```
glVertex3f(-0.3, 0.13, 0.15);
glColor3f(0.4, 0.3, 0.2); // back
glVertex3f(-0.3, 0.1,-0.15);
glVertex3f(-0.28, 0.1,-0.15);
glVertex3f(-0.28, 0.13,-0.15);
glVertex3f(-0.3, 0.13,-0.15);
glColor3f(0.5, 0.3, 0.2); // top
glVertex3f(-0.28, 0.1, 0.15);
glVertex3f(-0.28, 0.1,-0.15);
glVertex3f(-0.28, 0.13,-0.15);
glVertex3f(-0.28, 0.13, 0.15);

// bridge top right bar
glColor3f(0.4, 0.3, 0.2); // front
glVertex3f(0.3, 0.1, 0.15);
glVertex3f(0.28, 0.1, 0.15);
glVertex3f(0.28, 0.13, 0.15);
glVertex3f(0.3, 0.13, 0.15);
glColor3f(0.5, 0.3, 0.2); // right
glVertex3f(0.3, 0.1, 0.15);
glVertex3f(0.3, 0.1,-0.15);
glVertex3f(0.3, 0.13,-0.15);
glVertex3f(0.3, 0.13, 0.15);
glColor3f(0.4, 0.3, 0.2); // back
glVertex3f(0.3, 0.1,-0.15);
glVertex3f(0.28, 0.1,-0.15);
glVertex3f(0.28, 0.13,-0.15);
glVertex3f(0.3, 0.13,-0.15);
glColor3f(0.5, 0.3, 0.2); // left
glVertex3f(0.28, 0.1, 0.15);
glVertex3f(0.28, 0.1,-0.15);
glVertex3f(0.28, 0.13,-0.15);
glVertex3f(0.28, 0.13, 0.15);

// lower front middle bar
glColor3f(0.4, 0.3, 0.5);
glVertex3f(-0.28,-0.15, 0.15); // front
glVertex3f(0.28,-0.15, 0.15);
glVertex3f(0.28,-0.12, 0.15);
glVertex3f(-0.28,-0.12, 0.15);
glVertex3f(-0.28,-0.15, 0.12); // top
glVertex3f(0.28,-0.15, 0.12);
glVertex3f(0.28,-0.12, 0.12);
glVertex3f(-0.28,-0.12, 0.12);

// lower back middle bar
glVertex3f(-0.28,-0.15,-0.15); // top
glVertex3f(0.28,-0.15,-0.15);
glVertex3f(0.28,-0.12,-0.15);
glVertex3f(-0.28,-0.12,-0.15);
```

```
glVertex3f(-0.28,-0.15,-0.12); // front
glVertex3f(0.28,-0.15,-0.12);
glVertex3f(0.28,-0.12,-0.12);
glVertex3f(-0.28,-0.12,-0.12);

// upper front middle bar
glColor3f(0.4, 0.3, 0.5); // front
glVertex3f(-0.28, 0.1, 0.15);
glVertex3f(0.28, 0.1, 0.15);
glVertex3f(0.28, 0.13, 0.15);
glVertex3f(-0.28, 0.13, 0.15);
glVertex3f(-0.28, 0.1, 0.12); // top
glVertex3f(0.28, 0.1, 0.12);
glVertex3f(0.28, 0.13, 0.12);
glVertex3f(-0.28, 0.13, 0.12);

// upper back middle bar
glColor3f(0.4, 0.3, 0.5); // top
glVertex3f(-0.28, 0.1,-0.15);
glVertex3f(0.28, 0.1,-0.15);
glVertex3f(0.28, 0.13,-0.15);
glVertex3f(-0.28, 0.13,-0.15);
glColor3f(0.4, 0.3, 0.5);
glVertex3f(-0.28, 0.1,-0.12); // front
glVertex3f(0.28, 0.1,-0.12);
glVertex3f(0.28, 0.13,-0.12);
glVertex3f(-0.28, 0.13,-0.12);
glEnd();

// HOUSE
glColor3f(0.3, 0.3, 0.3);
glPushMatrix();
glTranslatef(0.0, 0.25, 0.0);
glutSolidCube(0.25);
glPopMatrix();
glBegin(GL_QUADS); // WINDOW
glColor3f(0.5, 0.9, 1.0);
glVertex3f(-0.05, 0.18, 0.16);
glVertex3f(0.05, 0.18, 0.16);
glVertex3f(0.05, 0.25, 0.16);
glVertex3f(-0.05, 0.25, 0.16);
glEnd();
glBegin(GL_TRIANGLES); // ROOF
glColor3f(1, 0, 0);
glVertex3f(-0.16, 0.35, 0.16);
glVertex3f(0.16, 0.35, 0.16);
glVertex3f(0.0, 0.5, 0.0);
glVertex3f(-0.16, 0.35, 0.16);
glVertex3f(-0.16, 0.35,-0.16);
glVertex3f(0.0, 0.5, 0.0);
```

```
glVertex3f(-0.16, 0.35,-0.16);
glVertex3f(0.16, 0.35,-0.16);
glVertex3f(0.0, 0.5, 0.0);
glVertex3f(0.16, 0.35, 0.16);
glVertex3f(0.16, 0.35,-0.16);
glVertex3f(0.0, 0.5, 0.0);
glEnd();
}
void rail_track()
{
glBegin(GL_LINES); // rail lines horizontal
glColor3f(0.0, 0.0, 0.0);
// left
glVertex3f(-3.0,-0.23, 0.12);
glVertex3f(-0.3,-0.23, 0.12);
glVertex3f(-3.0,-0.23, 0.1);
glVertex3f(-0.3,-0.23, 0.1);
glVertex3f(-3.0,-0.23,-0.12);
glVertex3f(-0.3,-0.23,-0.12);
glVertex3f(-3.0,-0.23,-0.1);
glVertex3f(-0.3,-0.23,-0.1);
// right
glVertex3f(3.0,-0.23, 0.12);
glVertex3f(0.3,-0.23, 0.12);
glVertex3f(3.0,-0.23, 0.1);
glVertex3f(0.3,-0.23, 0.1);
glVertex3f(3.0,-0.23,-0.12);
glVertex3f(0.3,-0.23,-0.12);
glVertex3f(3.0,-0.23,-0.1);
glVertex3f(0.3,-0.23,-0.1);
glEnd();
glBegin(GL_LINES); // rail lines vertical
glColor3f(1, 1, 1);
for (float j = 0.0; j <= 2.6; j += 0.1)
{
glVertex3f(-3.0 + j,-0.23, 0.1);
glVertex3f(-3.0 + j,-0.23,-0.1);
}
for (float j = 0.0; j <= 3; j += 0.1)
{
glVertex3f(0.3 + j,-0.23, 0.1);
glVertex3f(0.3 + j,-0.23,-0.1);
}
glEnd();
}
void ship()
{
glBegin(GL_QUADS);
glColor3f(0, 0.5, 0.2); // base front
glVertex3f(-0.2,-0.4,-3.5);
```

```
glVertex3f(0.2,-0.4,-3.5);
glVertex3f(0.2,-0.3,-3.5);
glVertex3f(-0.2,-0.3,-3.5);
glColor3f(0, 0.5, 0.5); // base side
glVertex3f(-0.2,-0.4,-3.5);
glVertex3f(-0.2,-0.4,-4.8);
glVertex3f(-0.2,-0.3,-5.0);
glVertex3f(-0.2,-0.3,-3.5);
glColor3f(0, 0, 0.5); // base top
glVertex3f(-0.2,-0.3,-3.5);
glVertex3f(0.2,-0.3,-3.5);
glVertex3f(0.2,-0.3,-5.0);
glVertex3f(-0.2,-0.3,-5.0);
// top base
glColor3f(0, 0.5, 0.2); // front
glVertex3f(-0.18,-0.3,-3.7);
glVertex3f(0.18,-0.3,-3.7);
glVertex3f(0.18,-0.2,-3.7);
glVertex3f(-0.18,-0.2,-3.7);
glColor3f(0, 0.5, 0.5); // side
glVertex3f(-0.18,-0.3,-3.7);
glVertex3f(-0.18,-0.3,-4.8);
glVertex3f(-0.18,-0.2,-4.8);
glVertex3f(-0.18,-0.2,-3.7);
glColor3f(0, 0, 0.5); // top
glVertex3f(-0.18,-0.2,-3.7);
glVertex3f(0.18,-0.2,-3.7);
glVertex3f(0.18,-0.2,-4.8);
glVertex3f(-0.18,-0.2,-4.8);
// 3rd base
glColor3f(0, 0.5, 0.2); // front
glVertex3f(-0.16,-0.19,-3.7);
glVertex3f(0.15,-0.19,-3.7);
glVertex3f(0.15,-0.1,-3.7);
glVertex3f(-0.16,-0.1,-3.7);
glColor3f(0, 0.5, 0.5); // side
glVertex3f(-0.15,-0.2,-3.7);
glVertex3f(-0.15,-0.2,-4.8);
glVertex3f(-0.15,-0.1,-4.8);
glVertex3f(-0.15,-0.1,-3.7);
glColor3f(0, 0, 0.5); // top
glVertex3f(-0.15,-0.1,-3.7);
glVertex3f(0.15,-0.1,-3.7);
glVertex3f(0.15,-0.1,-4.8);
glVertex3f(-0.15,-0.1,-4.8);
glEnd();
// bow section
glBegin(GL_TRIANGLES);
glColor3f(0.4, 0.4, 0.4); // front triangle
glVertex3f(-0.2,-0.4,-3.5);
```

```
glVertex3f(-0.2,-0.3,-3.5);
glVertex3f(0.0,-0.15,-2.2);
glColor3f(0, 0, 0); // front bottom
glVertex3f(-0.2,-0.4,-3.5);
glVertex3f(0.2,-0.4,-3.5);
glVertex3f(0.0,-0.15,-2.2);
glColor3f(0.5, 0.5, 0.7); // back triangle
glVertex3f(0.2,-0.4,-3.5);
glVertex3f(0.2,-0.3,-3.5);
glVertex3f(0.0,-0.15,-2.2);
glEnd();
// top pillar
glBegin(GL_QUADS);
glColor3f(1.0, 0.5, 0.1); // front
glVertex3f(-0.05,-0.2,-4.2);
glVertex3f(0.05,-0.2,-4.2);
glVertex3f(0.05, 0.2,-4.2);
glVertex3f(-0.05, 0.2,-4.2);
glColor3f(1.0, 0.6, 0.2); // left
glVertex3f(-0.05,-0.2,-4.2);
glVertex3f(-0.05,-0.2,-4.5);
glVertex3f(-0.05, 0.2,-4.5);
glVertex3f(-0.05, 0.2,-4.2);
glColor3f(1.0, 0.5, 0.1); // back
glVertex3f(-0.05,-0.2,-4.5);
glVertex3f(0.05,-0.2,-4.5);
glVertex3f(0.05, 0.2,-4.5);
glVertex3f(-0.05, 0.2,-4.5);
glColor3f(1.0, 0.6, 0.2); // right
glVertex3f(0.05,-0.2,-4.2);
glVertex3f(0.05,-0.2,-4.5);
glVertex3f(0.05, 0.2,-4.5);
glVertex3f(0.05, 0.2,-4.2);
glEnd();
}
void train()
{
glBegin(GL_QUADS);
// Engine
glColor3f(1.0, 0.6, 0.2); // bottom front
glVertex3f(1.0,-0.23, 0.1);
glVertex3f(1.15,-0.23, 0.1);
glVertex3f(1.15,-0.14, 0.1);
glVertex3f(1.0,-0.14, 0.1);
glColor3f(1.0, 0.5, 0.1); // bottom side
glVertex3f(1.0,-0.23, 0.1);
glVertex3f(1.0,-0.23,-0.1);
glVertex3f(1.0,-0.14,-0.1);
glVertex3f(1.0,-0.14, 0.1);
glColor3f(1, 1, 1); // windshield slope
```

```
glVertex3f(1.0,-0.14, 0.1);
glVertex3f(1.15,-0.05, 0.1);
glVertex3f(1.15,-0.05,-0.1);
glVertex3f(1.0,-0.14,-0.1);
glColor3f(0.5, 0.9, 1.0); // windshield
glVertex3f(1.02,-0.12, 0.1);
glVertex3f(1.13,-0.05, 0.1);
glVertex3f(1.13,-0.05,-0.06);
glVertex3f(1.02,-0.12,-0.06);
glEnd();
// Engine Side
glBegin(GL_TRIANGLES);
glColor3f(0.9, 0.9, 0.9);
glVertex3f(1.0,-0.14, 0.1);
glVertex3f(1.15,-0.14, 0.1);
glVertex3f(1.15,-0.05, 0.1);
glEnd();
// bogies
glBegin(GL_QUADS);
for (float j = 0.0; j < 2; j += 0.27)
{
glColor3f(1, 1, 0); // front
glVertex3f(1.15 + j,-0.23, 0.1);
glVertex3f(1.4 + j,-0.23, 0.1);
glVertex3f(1.4 + j,-0.05, 0.1);
glVertex3f(1.15 + j,-0.05, 0.1);
glColor3f(0.8, 0.6, 0.4); // sides
glVertex3f(1.15 + j,-0.23, 0.1);
glVertex3f(1.15 + j,-0.23,-0.1);
glVertex3f(1.15 + j,-0.05,-0.1);
glVertex3f(1.15 + j,-0.05, 0.1);
glColor3f(0.7, 0.7, 0.2); // top
glVertex3f(1.15 + j,-0.05, 0.1);
glVertex3f(1.4 + j,-0.05, 0.1);
glVertex3f(1.4 + j,-0.05,-0.1);
glVertex3f(1.15 + j,-0.05,-0.1);
glColor3f(0.0, 0.0, 0.0); // window
glVertex3f(1.20 + j,-0.17, 0.12);
glVertex3f(1.3 + j,-0.17, 0.12);
glVertex3f(1.3 + j,-0.07, 0.12);
glVertex3f(1.20 + j,-0.07, 0.12);
}
glEnd();
}
void signal()
{
glBegin(GL_QUADS);
glColor3f(0.4, 0.3, 0.1); // pole
glVertex3f(0.7,-0.25,-0.17);
glVertex3f(0.73,-0.25,-0.17);
```

```
glVertex3f(0.73, 0.10,-0.17);
glVertex3f(0.7, 0.10,-0.17);
glColor3f(0.1, 0.1, 0.1); // light mount
glVertex3f(0.67, 0.10,-0.17);
glVertex3f(0.76, 0.10,-0.17);
glVertex3f(0.76, 0.3,-0.17);
glVertex3f(0.67, 0.3,-0.17);
glEnd();
}
void light()
{
if (b > 0.0)
glColor3f(1.0, 0.0, 0.0);
else
glColor3f(0.0, 1.0, 0.0);
if (p < -3.5)
glColor3f(1.0, 0.0, 0.0);
glPushMatrix();
glTranslatef(0.715, 0.25,-0.17);
glutSolidSphere(0.03, 10, 10);
glPopMatrix();
}
void call1()
{
    // Translate to specified position
    glTranslatef(a, b, c);
    // Call the function to draw a bridge
    bridge();
}

void call2()
{
    // Translate to specified position
    glTranslatef(m, n, o);
    // Call the function to draw a ship
    ship();
}

void call3()
{
    // Translate to specified position
    glTranslatef(p, q, r);
    // Call the function to draw a train
    train();
}

void update(int value)
{
    // Check if the flag for automatic animation is enabled
    if (flagx == 1)
```

```
    {
        // Check if the flag for lowering the bridge is enabled
        if (flagb == 1) // bridge down on change
        {
            // Increment the y-translation for lowering the bridge
            b += 0.02f;
            // Check if the bridge is lowered sufficiently
            if (b > 0.5)
            {
                flagb = 2;
                flags = 1;
            }
        }
        // Check if the flag for moving the ship is enabled
        if (flags == 1) // ship
        {
            // Increment the z-translation for moving the ship
            o += 0.07f;
            // Check if the ship has moved past a certain point
            if (o > 2.0)
                flagp = 1;
            // Check if the ship has moved far enough to raise the bridge
            if (o > 6.0)
            {
                flagb = 0;
            }
        }
        // Check if the flag for raising the bridge is enabled
        if (flagb == 0) // bridge up on change
        {
            // Decrement the y-translation for raising the bridge
            b -= 0.02f;
            // Check if the bridge is raised sufficiently
            if (b < 0.01)
            {
                flagb = 1;
                flagt = 1;
            }
        }
        // Check if the flag for moving the train is enabled
        if (flagt == 1) // train
        {
            // Decrement the x-translation for moving the train
            p -= 0.1f;
        }
        // Check if the flag for reversing the wave animation is enabled
        if (flagw == 1) // wave reverse
        {
            // Increment a counter for wave animation
            a1 += 20.6;
```

```
        }
    }
    // Request a redraw
    glutPostRedisplay();
    // Set up the next timer callback
    glutTimerFunc(100, update, 0);
}

void update_windmill(int v) {
    if(flagx == 1)
    {
        frameNumber++;
    glutPostRedisplay();
    }
    glutTimerFunc(30,update_windmill,0);
}

void display()
{
    // Clear the color and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(0.0, 0.0, 0.0, 0.0); // Set clear color to black

    // Draw the track base front
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0); // Rotate the scene
    track_base_front();
    glPopMatrix();

    // Draw the gate pillar
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    gate_pillar();
    glPopMatrix();

    // Draw the track base
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    track_base();
    glPopMatrix();

    // Draw the rail track
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    rail_track();
    glPopMatrix();

    // Draw the bridge
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
```

```
    call1();
    glPopMatrix();

    // Draw the ship
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    call2();
    glPopMatrix();

    // Draw the train
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    call3();
    glPopMatrix();

    // Draw the ocean waves
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    ocean();
    glPopMatrix();

    // Draw the moon
    glPushMatrix();
    glColor3f(1.0, 1.0, 1.0); // Set color to white
    glTranslatef(1.2, 0.9, -5.1);
    glutSolidSphere(0.08, 20, 20);
    glPopMatrix();

    // Draw the moon mask
    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0); // Set color to black
    glTranslatef(1.3, 0.95, -5.0);
    glutSolidSphere(0.13, 20, 20);
    glPopMatrix();

    // Draw the lighthouse light
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    glTranslatef(0.18, -0.05, -5.0);
    glColor3f(1.0, 1.0, 0.0); // Set color to yellow
    glutSolidSphere(0.02, 20, 20);
    glPopMatrix();

    // Draw the signal
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    signal();
    glPopMatrix();

    // Draw the light
```

```
    glPushMatrix();
    glRotatef(20.0, 0.25, 0.5, 0.0);
    light();
    glPopMatrix();

    // Draw the windmill
    glPushMatrix();
    windmill();
    glPopMatrix();
    // Draw the stars
    star();
    // Flush drawing commands
    glFlush();
}

void mykeyboard(unsigned char key, int x, int y)
{
switch (key)
{
case 80: // P to play
flagx = 1;
break;
case 112: // p to play
flagx = 1;
break;
case 83:
flagx = 0; // S to stop
break;
case 115:
flagx = 0; // s to stop
break;
default:
break;
}
if (key == 'r' || key == 'R')
    {
        is_star_active = !is_star_active; // Toggle rain effect on/off using r or R
key
    }
}
void reshape(int w, int h)
{
    // Set viewport to cover the entire window
    glViewport(0, 0, w, h);
    // Switch to the projection matrix mode
    glMatrixMode(GL_PROJECTION);
    // Reset the projection matrix
    glLoadIdentity();
    // Adjust the projection based on window aspect ratio
    if (w <= h)
```

```
    {
        // Adjust projection for portrait or square window
        glOrtho(-1.1, 1.1, 1.1 * (GLfloat)h / (GLfloat)w, 1.1 * (GLfloat)h /
(GLfloat)w, -10.0, 10.0);
    }
    else
    {
        // Adjust projection for landscape window
        glOrtho(-1.1 * (GLfloat)w / (GLfloat)h, 1.1 * (GLfloat)w / (GLfloat)h, -1.1,
1.1, -10.0, 10.0);
    }
    // Switch back to the modelview matrix mode
    glMatrixMode(GL_MODELVIEW);
    // Reset the modelview matrix
    glLoadIdentity();
}

int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(1500, 1000);
glutCreateWindow("Ship Under Bridge Simulation");
glClearColor(0.0, 0.0, 0.0, 0.0);
glEnable(GL_DEPTH_TEST);
glutReshapeFunc(reshape);
glutDisplayFunc(display);
glutKeyboardFunc(mykeyboard);
glutTimerFunc(30, update_windmill,0);
glutTimerFunc(200, update, 0);
glutMainLoop();
}
```