Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Mini Project Report
on

“Spinner”

[Code No: COMP 342]

Submitted by

Rojan Gautam (14)
Ashwin Sapkot (61)

Submitted to

Mr. Dhiraj Shrestha

Assistant Professor

Department of Computer Science and Engineering

Submission Date: 15th June, 2024

Table of Contents

Chapter 1 : Introduction

The spinner mini-project is an interactive graphics application developed using the OpenGL and GLUT libraries. These libraries provide a robust framework for creating 2D and 3D graphics applications, making them ideal for educational projects and prototypes. This project showcases the capabilities of OpenGL and GLUT by rendering a 3D wind spinner that responds to user inputs for rotation, scaling, and sound effects based on its speed.

Libraries Used

1. **OpenGL (Open Graphics Library)**:
    - OpenGL is a cross-platform graphics API that provides a wide range of functions for rendering 2D and 3D vector graphics. It is extensively used in games, simulations, and various applications requiring high-performance graphics.
    - In this project, OpenGL is used to define and render the geometry of the spinner. Functions like `glBegin`, `glEnd`, and `glVertex3fv` are used to specify the vertices and construct the shapes of the spinner's parts. Transformations like scaling and rotation are applied using `glScalef` and `glRotatef` to animate the spinner.
2. **GLUT (OpenGL Utility Toolkit)**:
    - GLUT is a utility library that simplifies the process of creating and managing windows with OpenGL contexts, handling user input, and managing events. It provides an easy-to-use interface for beginners and is often used in conjunction with OpenGL for educational purposes and quick prototyping.

Chapter 2 : Project Description

The spinner project showcases how to create an interactive and dynamic graphical application using OpenGL and GLUT.

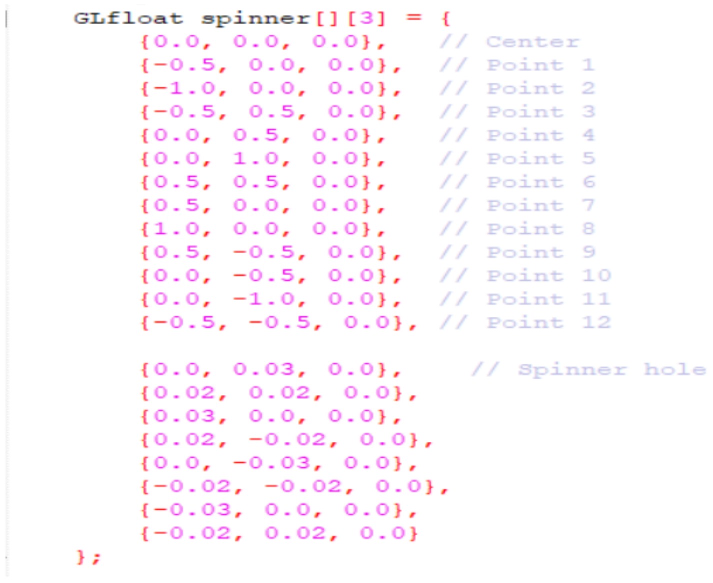
2.1 Geometry Definition

The foundation of the 3D wind spinner lies in its geometric definition. Each part of the spinner is constructed using a series of vertices that outline its shape in a 3D space.

Spinner Coordinates

The spinner's shape is defined by a set of vertices stored in a 2D array. Each vertex is represented by a triplet of coordinates (x, y, z), specifying its position in 3D space. The array captures the following parts:

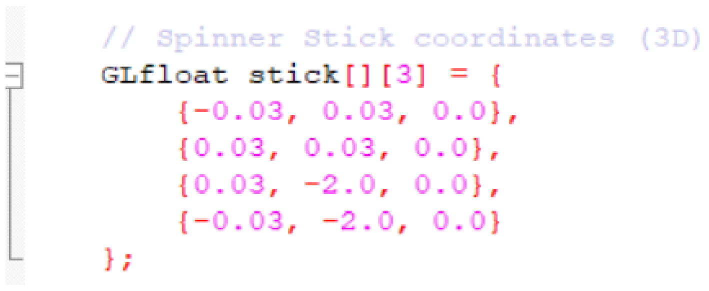
- **Centre Point**: The hub of the spinner, around which all other parts rotate.
- **Leaves**: The spinner has four distinct leaves, each crafted from multiple vertices to form a complex shape. These leaves give the spinner its characteristic look.
- **Hole**: A smaller circular area at the centre of the spinner, adding a realistic touch to the design.



```
GLfloat spinner[][3] = {
    {0.0, 0.0, 0.0},     // Center
    {-0.5, 0.0, 0.0},    // Point 1
    {-1.0, 0.0, 0.0},    // Point 2
    {-0.5, 0.5, 0.0},    // Point 3
    {0.0, 0.5, 0.0},     // Point 4
    {0.0, 1.0, 0.0},     // Point 5
    {0.5, 0.5, 0.0},     // Point 6
    {0.5, 0.0, 0.0},     // Point 7
    {1.0, 0.0, 0.0},     // Point 8
    {0.5, -0.5, 0.0},    // Point 9
    {0.0, -0.5, 0.0},    // Point 10
    {0.0, -1.0, 0.0},    // Point 11
    {-0.5, -0.5, 0.0},   // Point 12

    {0.0, 0.03, 0.0},       // Spinner hole
    {0.02, 0.02, 0.0},
    {0.03, 0.0, 0.0},
    {0.02, -0.02, 0.0},
    {0.0, -0.03, 0.0},
    {-0.02, -0.02, 0.0},
    {-0.03, 0.0, 0.0},
    {-0.02, 0.02, 0.0}
};
```

Stick Coordinates

The supporting stick of the spinner is also defined using a set of vertices, forming a rectangular shape that extends vertically from the spinner:



```
// Spinner Stick coordinates (3D)
GLfloat stick[][3] = {
    {-0.03, 0.03, 0.0},
    {0.03, 0.03, 0.0},
    {0.03, -2.0, 0.0},
    {-0.03, -2.0, 0.0}
};
```

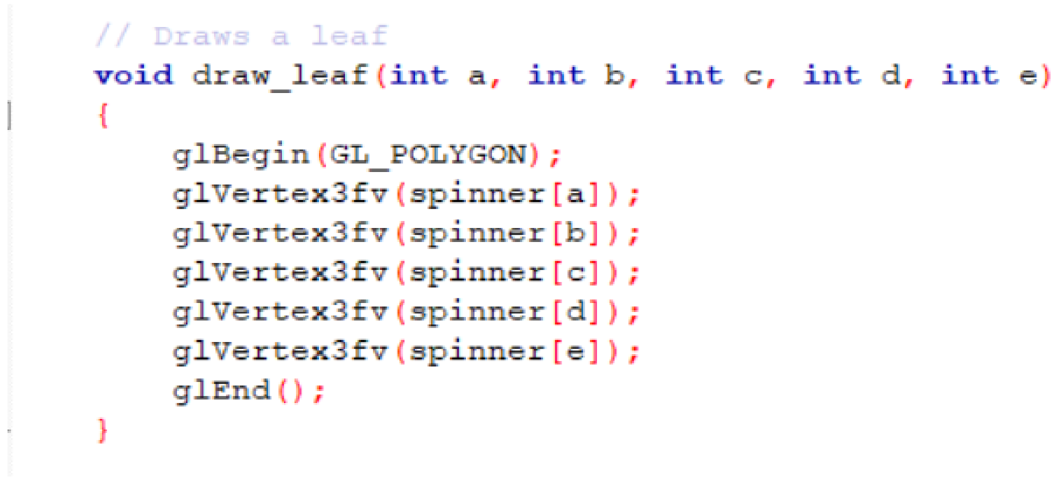
2.2 Rendering the Spinner

Rendering involves converting the geometric definitions into visual shapes on the screen. This is done using OpenGL's immediate mode, where we specify the vertices and colours for each part of the spinner.

Drawing Functions

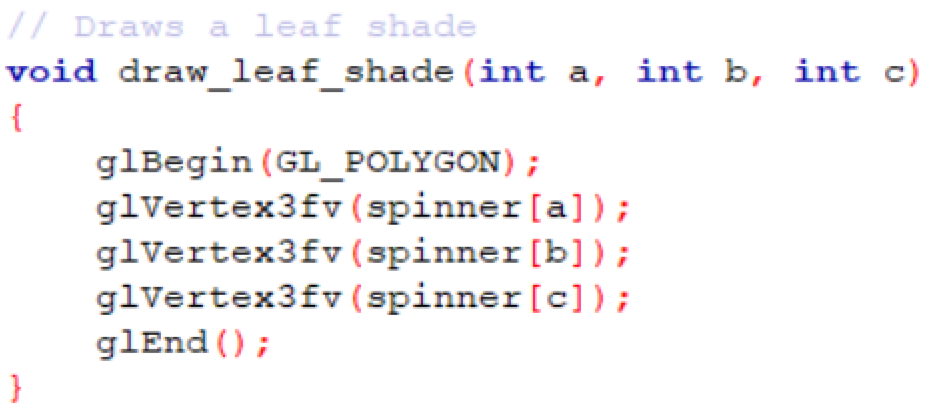
Several custom functions are created to draw different parts of the spinner:

draw_leaf: This function draws a leaf by connecting five vertices, forming a polygon. The colours for each leaf are set before drawing, giving the spinner its colourful appearance.



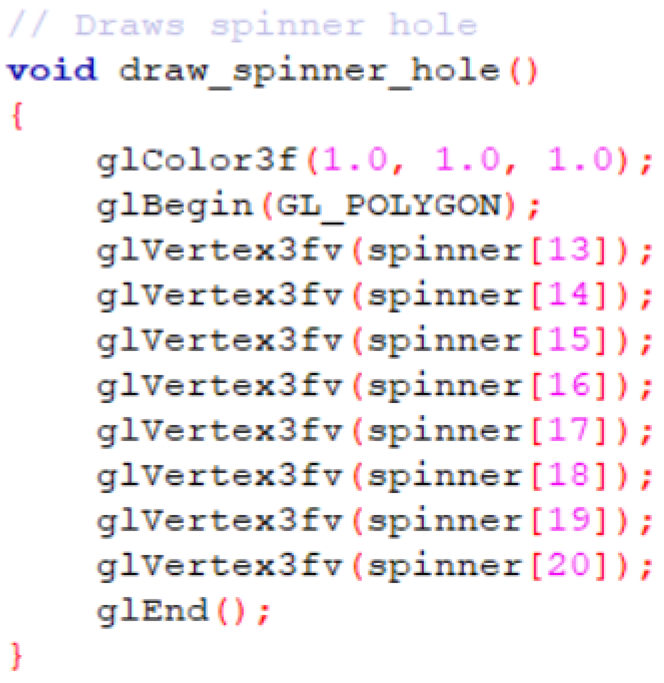
```
// Draws a leaf
void draw_leaf(int a, int b, int c, int d, int e)
{
    glBegin(GL_POLYGON);
    glVertex3fv(spinner[a]);
    glVertex3fv(spinner[b]);
    glVertex3fv(spinner[c]);
    glVertex3fv(spinner[d]);
    glVertex3fv(spinner[e]);
    glEnd();
}
```

draw_leaf_shade: Similar to `draw_leaf`, this function draws a shaded triangle for each leaf, adding depth and visual interest.



```
// Draws a leaf shade
void draw_leaf_shade(int a, int b, int c)
{
	glBegin(GL_POLYGON);
	glVertex3fv(spinner[a]);
	glVertex3fv(spinner[b]);
	glVertex3fv(spinner[c]);
	glEnd();
}
```

draw_spinner_hole: This function draws the circular hole at the centre of the spinner using a series of vertices.



```
// Draws spinner hole
void draw_spinner_hole()
{
	glColor3f(1.0, 1.0, 1.0);
	glBegin(GL_POLYGON);
	glVertex3fv(spinner[13]);
	glVertex3fv(spinner[14]);
	glVertex3fv(spinner[15]);
	glVertex3fv(spinner[16]);
	glVertex3fv(spinner[17]);
	glVertex3fv(spinner[18]);
	glVertex3fv(spinner[19]);
	glVertex3fv(spinner[20]);
	glEnd();
}
```

draw_stick: This function renders the supporting stick of the spinner by drawing a quadrilateral from its four vertices.

```
// Draws spinner stick
void draw_stick()
{
    glColor3f(d2rgb(196), d2rgb(167), d2rgb(141));
    glBegin(GL_POLYGON);
    glVertex3fv(stick[0]);
    glVertex3fv(stick[1]);
    glVertex3fv(stick[2]);
    glVertex3fv(stick[3]);
    glEnd();
}
```

2.3 Applying Transformations

To bring the spinner to life, transformations are applied to animate its rotation and adjust its size. These transformations are handled in the `display` function, which updates the spinner's state on each frame.

Rotation

The spinner rotates around its centre axis, creating a spinning effect. This is achieved using `glRotatef`, which applies a rotation transformation based on the current `spinnerAngle`:

```
glRotatef(spinnerAngle, 0.0f, 0.0f, 1.0f);
```

The angle is updated in each display cycle, with the spinner's speed controlled by the `spinnerSpeed` variable. This continuous update makes the spinner appear to rotate smoothly.

Scaling

The size of the spinner can be adjusted by scaling it up or down. This transformation is applied using `glScalef`, with the scale factor modified based on user input:

```
glScalef(scale, scale, scale);
```

Scaling allows the spinner to change in size while maintaining its proportions, enhancing the visual experience.

2.4 User Interaction

The project provides interactive controls for the spinner's behaviour through keyboard inputs. The `keyboard` function handles these inputs, allowing users to:

- **Adjust Rotation Speed**: Keys 'a' and 'd' decrease and increase the spinner's speed, respectively, causing it to rotate counterclockwise or clockwise.
- **Stop Rotation**: The 's' key stops the spinner's rotation by setting the speed to zero.
- **Change Scale**: Keys 'x' and 'z' increase and decrease the size of the spinner, respectively.

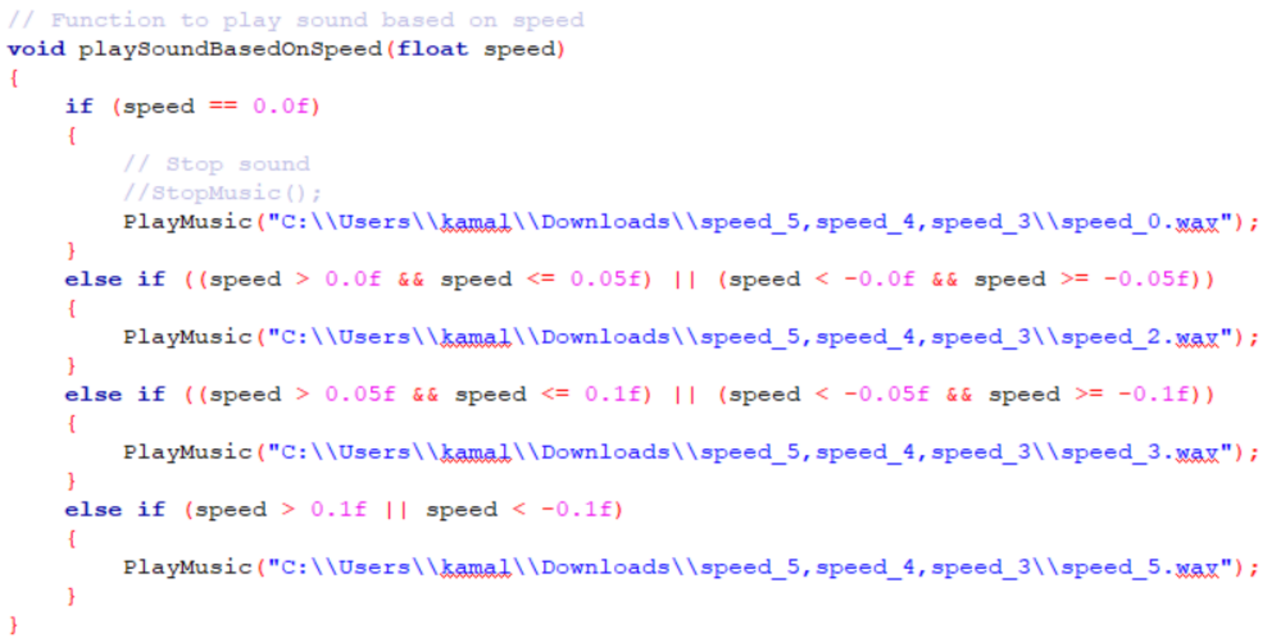
Here's how the `keyboard` function processes these inputs:

```
// Keyboard Controls
void keyboard(unsigned char key, int x, int y)
{
    if (key == 'a')
    {
        spinnerSpeed -= 0.01f;
    }
    if (key == 'd')
    {
        spinnerSpeed += 0.01f;
    }
    if (key == 's')
    {
        spinnerSpeed = 0.0f;
    }
    if (key == 'x')
    {
        scale += 0.01f;
    }
    if (key == 'z')
    {
        scale -= 0.01f;
    }

    // Play sound based on the updated speed
    playSoundBasedOnSpeed(spinnerSpeed);

    glutPostRedisplay();
}
```

2.5 Sound Effects

To enhance the interactive experience, the project includes sound effects that change based on the spinner's rotation speed. The `playSoundBasedOnSpeed` function selects the appropriate sound to play, adding an auditory layer to the visual spinning effect.

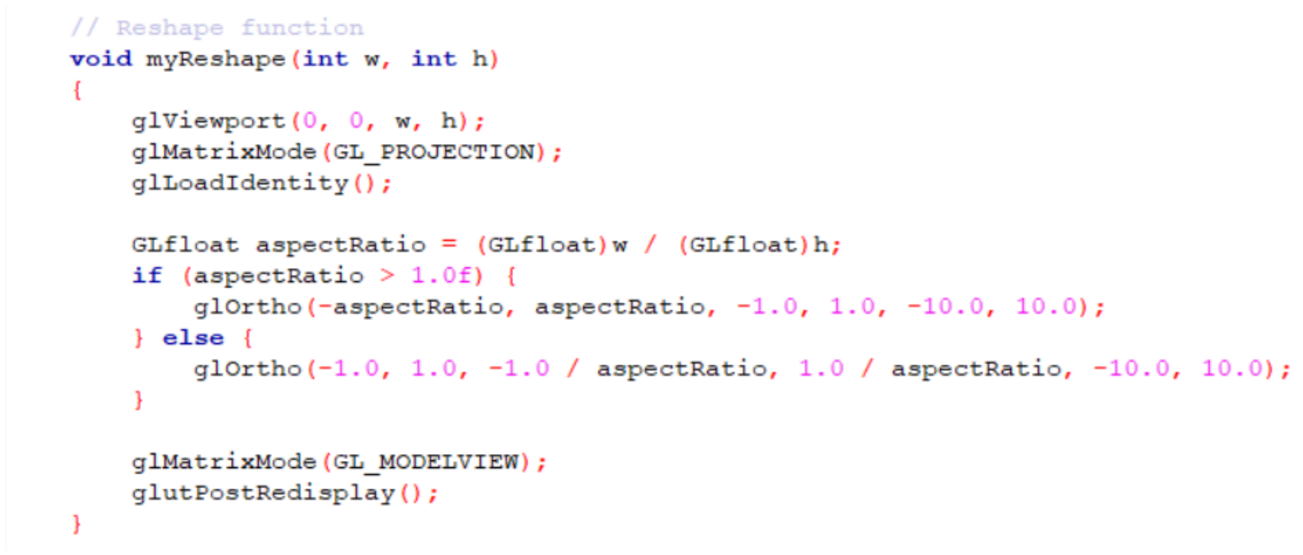
Although the specific implementation for sound playback is indicated with placeholder functions like `PlayMusic`, the concept demonstrates how sound can be integrated into a graphical application to respond dynamically to user actions.



```
// Function to play sound based on speed
void playSoundBasedOnSpeed(float speed)
{
    if (speed == 0.0f)
    {
        // Stop sound
        //StopMusic();
        PlayMusic("C:\\Users\\kamal\\Downloads\\speed_5,speed_4,speed_3\\speed_0.wav");
    }
    else if ((speed > 0.0f && speed <= 0.05f) || (speed < -0.0f && speed >= -0.05f))
    {
        PlayMusic("C:\\Users\\kamal\\Downloads\\speed_5,speed_4,speed_3\\speed_2.wav");
    }
    else if ((speed > 0.05f && speed <= 0.1f) || (speed < -0.05f && speed >= -0.1f))
    {
        PlayMusic("C:\\Users\\kamal\\Downloads\\speed_5,speed_4,speed_3\\speed_3.wav");
    }
    else if (speed > 0.1f || speed < -0.1f)
    {
        PlayMusic("C:\\Users\\kamal\\Downloads\\speed_5,speed_4,speed_3\\speed_5.wav");
    }
}
```

2.6 Window and Display Management

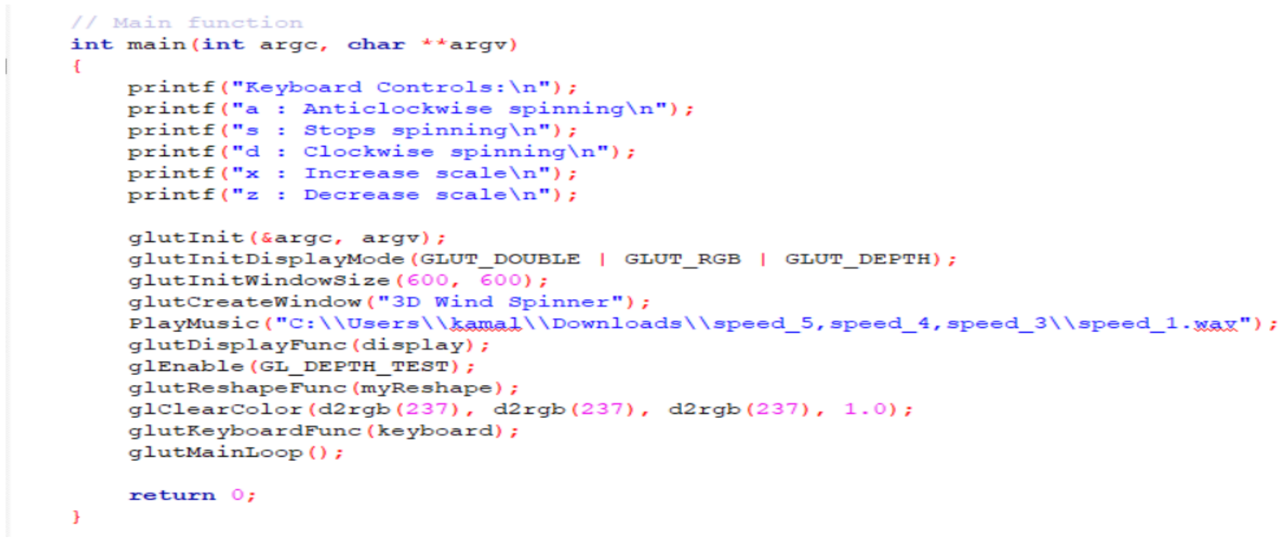
Managing the display and maintaining the spinner's proportions across different window sizes is handled by the `myReshape` function. This function adjusts the viewport and projection settings based on the window's aspect ratio, ensuring that the spinner retains its visual integrity.



```
// Reshape function
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    GLfloat aspectRatio = (GLfloat)w / (GLfloat)h;
    if (aspectRatio > 1.0f) {
        glOrtho(-aspectRatio, aspectRatio, -1.0, 1.0, -10.0, 10.0);
    } else {
        glOrtho(-1.0, 1.0, -1.0 / aspectRatio, 1.0 / aspectRatio, -10.0, 10.0);
    }

    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}
```

2.7 Main Function

The `main` function initialises the GLUT environment, sets up the display mode, creates the window, and enters the main event loop. This setup is crucial for bringing all components together and starting the spinner's animation.



```
// Main function
int main(int argc, char **argv)
{
    printf("Keyboard Controls:\n");
    printf("a : Anticlockwise spinning\n");
    printf("s : Stops spinning\n");
    printf("d : Clockwise spinning\n");
    printf("x : Increase scale\n");
    printf("z : Decrease scale\n");

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutCreateWindow("3D Wind Spinner");
    PlayMusic("C:\\Users\\kamal\\Downloads\\speed_5,speed_4,speed_3\\speed_1.wav");
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glutReshapeFunc(myReshape);
    glClearColor(d2rgb(237), d2rgb(237), d2rgb(237), 1.0);
    glutKeyboardFunc(keyboard);
    glutMainLoop();

    return 0;
}
```

Output

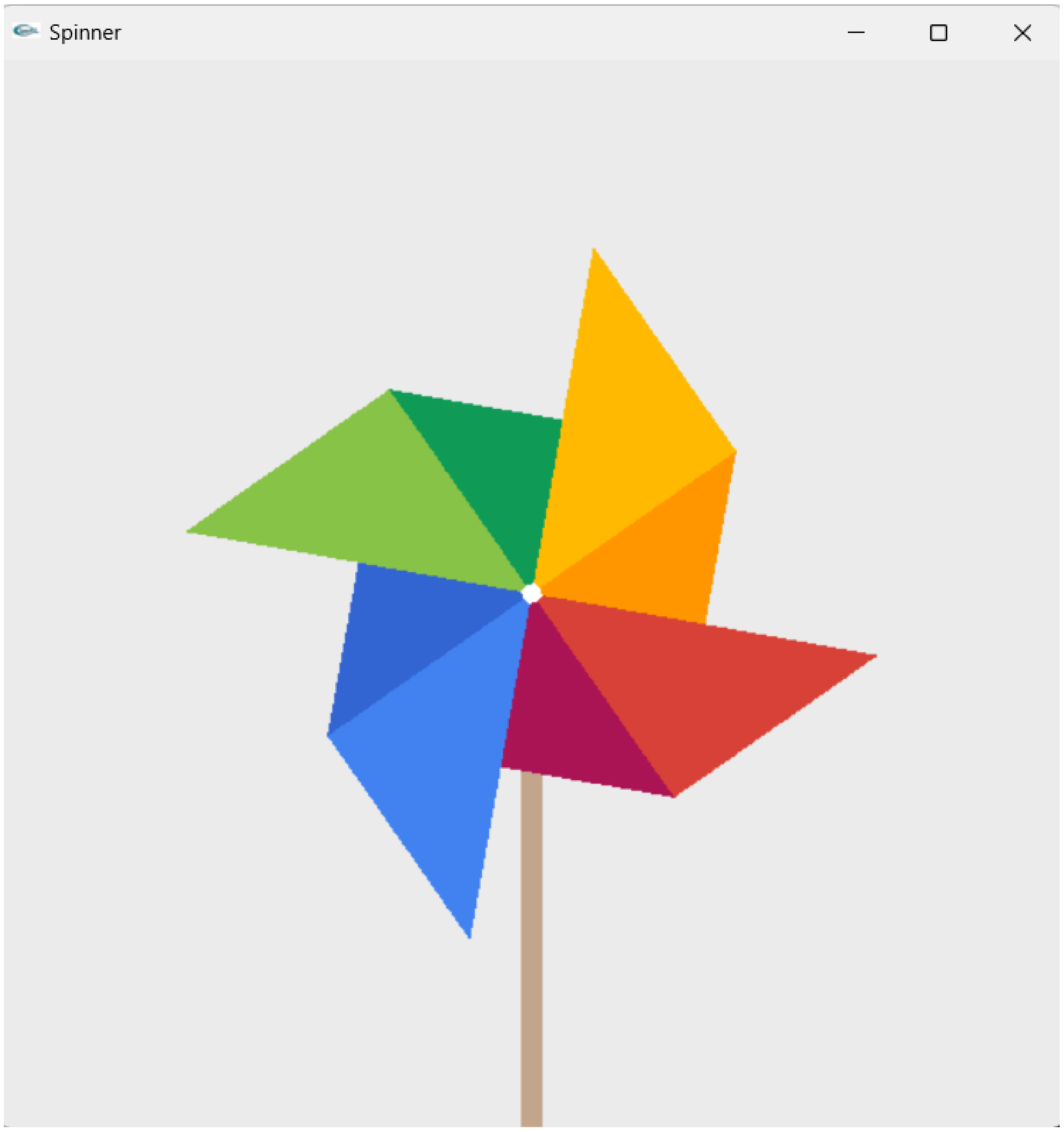


Fig : Final Output Of Spinner

Conclusion

The spinner project effectively showcases the capabilities of OpenGL and GLUT in creating an engaging and interactive graphical application. Through precise geometric modelling, we crafted a visually detailed spinner that animates with realistic motion, controlled by user inputs. The project highlights the power of dynamic transformations and the responsiveness of interactive controls, allowing users to alter the spinner's speed, direction, and scale in real time. Additionally, the integration of sound effects tied to the spinner’s speed adds an immersive auditory dimension. Handling various window sizes gracefully, the spinner maintains its visual integrity, demonstrating robust design principles. This project not only enhances our understanding of OpenGL’s rendering pipeline and real-time graphics but also provides a foundation for future exploration in interactive applications. Overall, the spinner project serves as a testament to the versatility and richness of computer graphics programming.

References

GLUT - The OpenGL Utility Toolkit. (n.d.). Retrieved from https://www.opengl.org/resources/libraries/glut/
This official documentation describes the GLUT library, which is used to create and manage windows, handle user input, and control rendering in OpenGL applications.

GLU - OpenGL Utility Library. (n.d.). Retrieved from https://www.opengl.org/documentation/specs/glu/
This source details the GLU library, which provides higher-level drawing routines built on top of OpenGL, simplifying the process of creating complex shapes and scenes.

Hearn, D. D., & Baker, M. P. (2003). *Computer Graphics with OpenGL* (3rd ed.). Pearson.

This book covers a wide range of topics in computer graphics with a strong emphasis on OpenGL. It provides both theoretical and practical insights that are crucial for developing graphical applications.

Source Code Link

Video Link