Course Title	**:**	Computer Graphics
Course Code	**:**	CSE420
Section	**:**	(02)
Semester	**:**	Spring 2022

Project Report

Project Name: 3D Movable Cube

Submitted To

Dr. Mohammad Rifat Ahmmad Rashid
Assistant Professor
Department of Computer Science and Engineering
East West University

Submitted By

Md. Jannatul Haq	2018-1-60-224
Abdullah Al-Nayeem	2018-1-60-086

Submission Date
27/04/2022

Contents:

► Project Description

► Description of each function

► Implementation

► Code

► Output

► Conclusion

Project Description:

►In this project, we are going to create a spinning 3D cube by using OpenGL library. At the initial stage, the cube will stay at the center of our window. Then we will rotate the cube by controlling the arrow keys of our keyboard. When we rotate the cube using the arrow keys of the keyboard, we will see all the six sides of this cube one by one.

We use C++ language with OpenGL library to implement our project. Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform-independent) API for rendering 2D and 3d Vector Graphics.

Description of each function:

```
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480
```

►In this part, we initialized our required screen.

```
window = glfwCreateWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Moveable cube", NULL, NULL);
```

►We created a windowed mode window and its OpenGL context.

```
glfwSetKeyCallback(window, keyCallback);
glfwSetInputMode(window, GLFW_STICKY_KEYS, 1);
```

►We initialized all keys and then call all the keys.

```
glfwMakeContextCurrent(window);
```

►Make the window's context current.

```
glViewport(0.0f, 0.0f, screenWidth, screenHeight);
```

►Specifies the part of the window to which OpenGL will draw (in pixels), convert from normalized to pixels.

```
glMatrixMode(GL_PROJECTION);
```

►The projection matrix defines the camera’s properties that view the objects in the world coordinate frame. Here you typically set the zoom factor, aspect ratio, and the near and far clipping planes.

```
glLoadIdentity();
```

►Replace the current matrix with the identity matrix and starts us a fresh because matrix transforms such as glOrpho and glRotate cumulate, basically puts us at (0, 0, 0)

```
glOrtho(0, SCREEN_WIDTH, 0, SCREEN_HEIGHT, 0, 1000);
```

►Essentially set coordinate system.

```
glMatrixMode(GL_MODELVIEW);
```

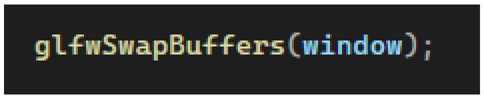
►(Default matrix mode) model view matrix defines how your objects are transformed (meaning translation, rotation, and scaling) in your world.

```
while (!glfwWindowShouldClose(window))
{
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
```

►Loop until the user closes the window.

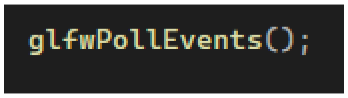
```
glPushMatrix();
glTranslatef(halfScreenWidth, halfScreenHeight, -500);
glRotatef(rotationX, 1, 0, 0);
glRotatef(rotationY, 0, 1, 0);
glTranslatef(-halfScreenWidth, -halfScreenHeight, 500);

DrawCube(halfScreenWidth, halfScreenHeight, -500, 200);

glPopMatrix();
```

►Render OpenGL here



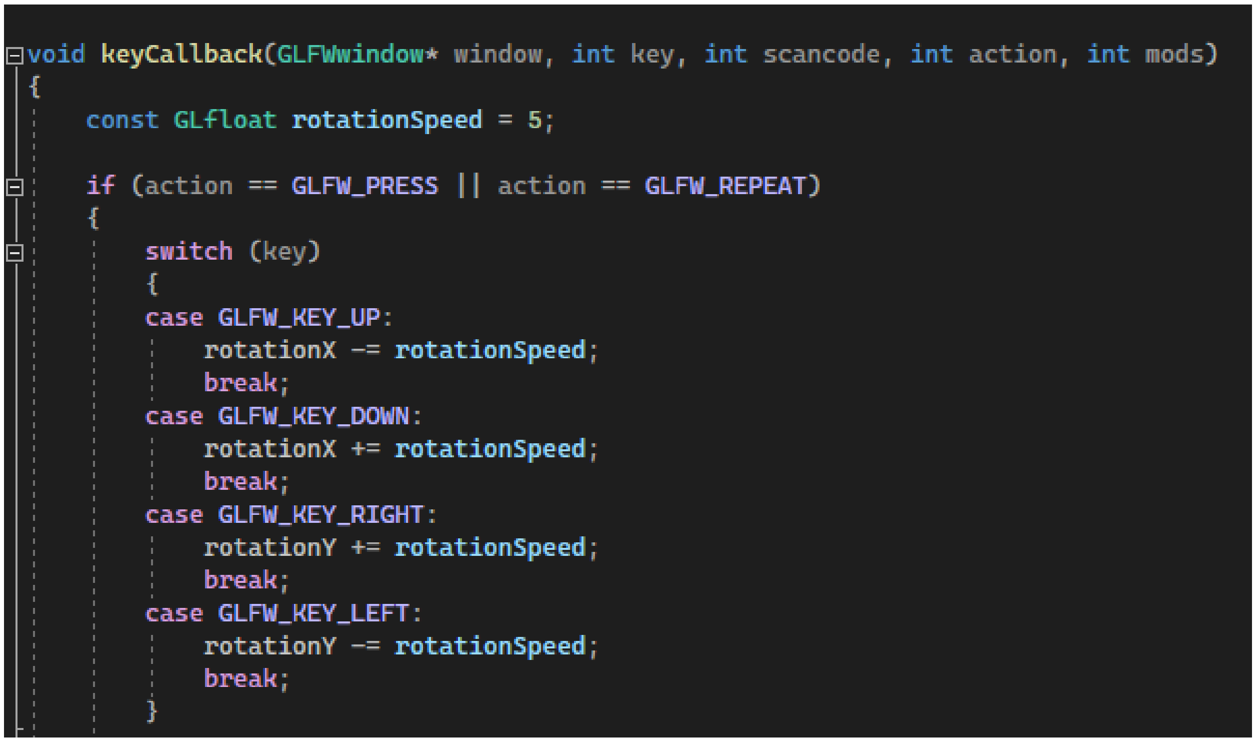
```
glfwSwapBuffers(window);
```

►Swap front and back buffers



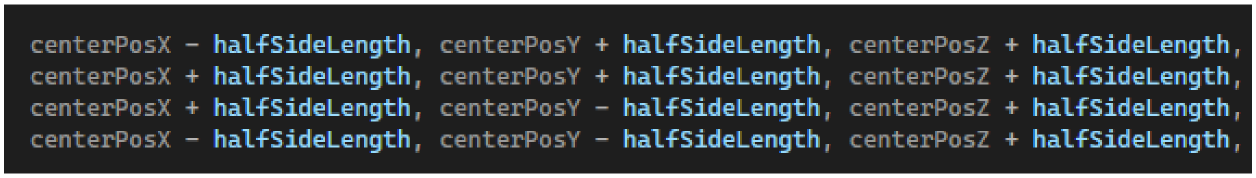
```
glfwPollEvents();
```

►Poll for and process events



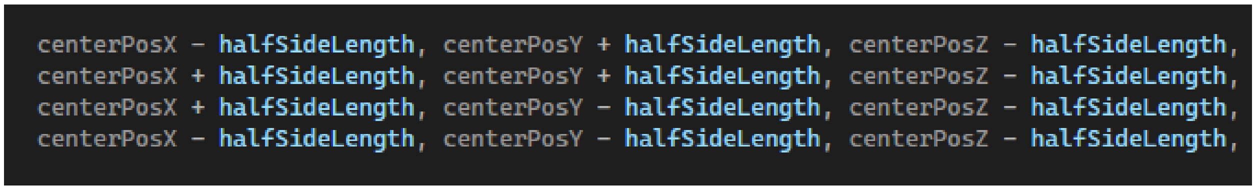
```
void keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    const GLfloat rotationSpeed = 5;

    if (action == GLFW_PRESS || action == GLFW_REPEAT)
    {
        switch (key)
        {
        case GLFW_KEY_UP:
            rotationX -= rotationSpeed;
            break;
        case GLFW_KEY_DOWN:
            rotationX += rotationSpeed;
            break;
        case GLFW_KEY_RIGHT:
            rotationY += rotationSpeed;
            break;
        case GLFW_KEY_LEFT:
            rotationY -= rotationSpeed;
            break;
        }
```

►Actions are GLFW_PRESS, GLFW_RELEASE or GLFW_REPEAT



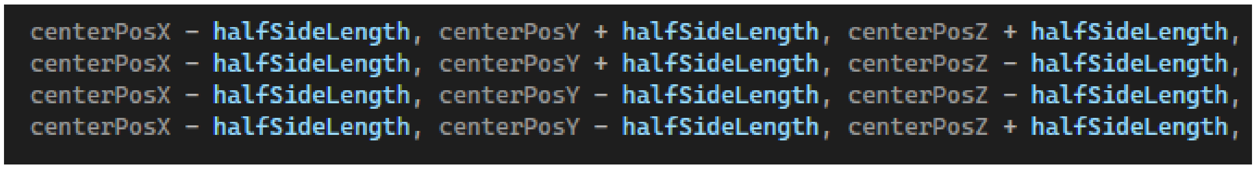
```
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength,
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength,
```

►The front face of the cube



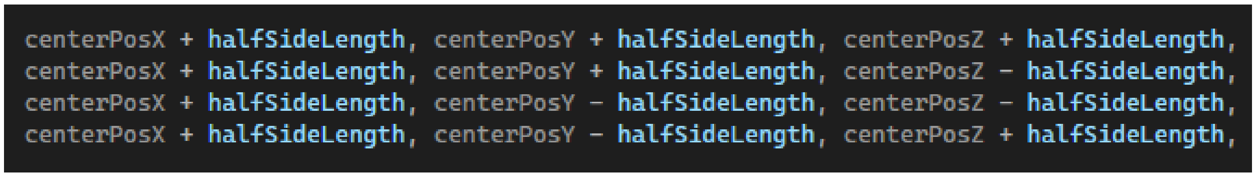
```
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
```

►The back face of the cube



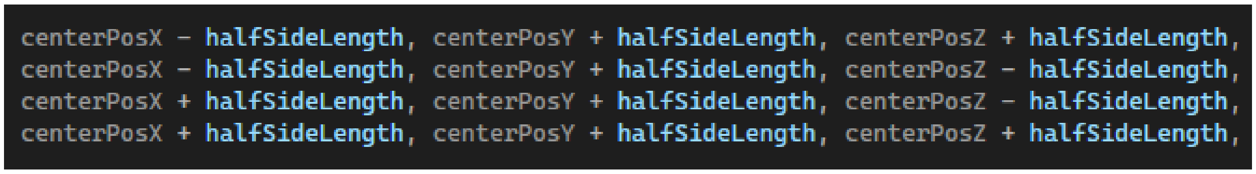
```
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength,
```

►The left face of the cube



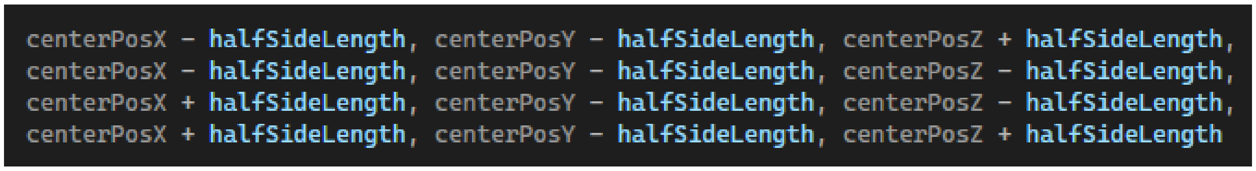
```
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength,
```

►The right face of the cube



```
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength,
```

►The top face of the cube



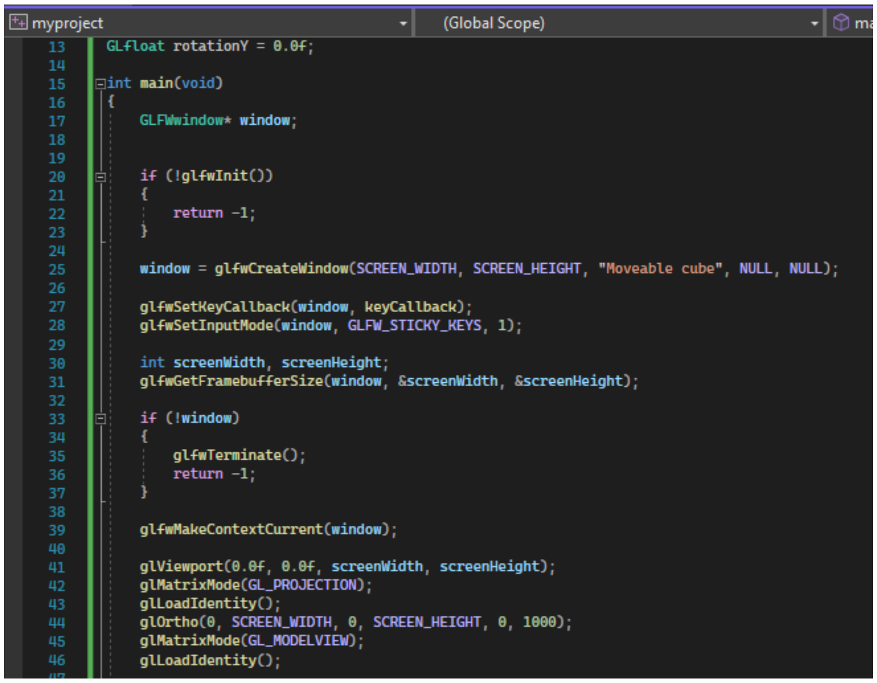
```
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength,
centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength,
centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength
```

►The down face of the cube

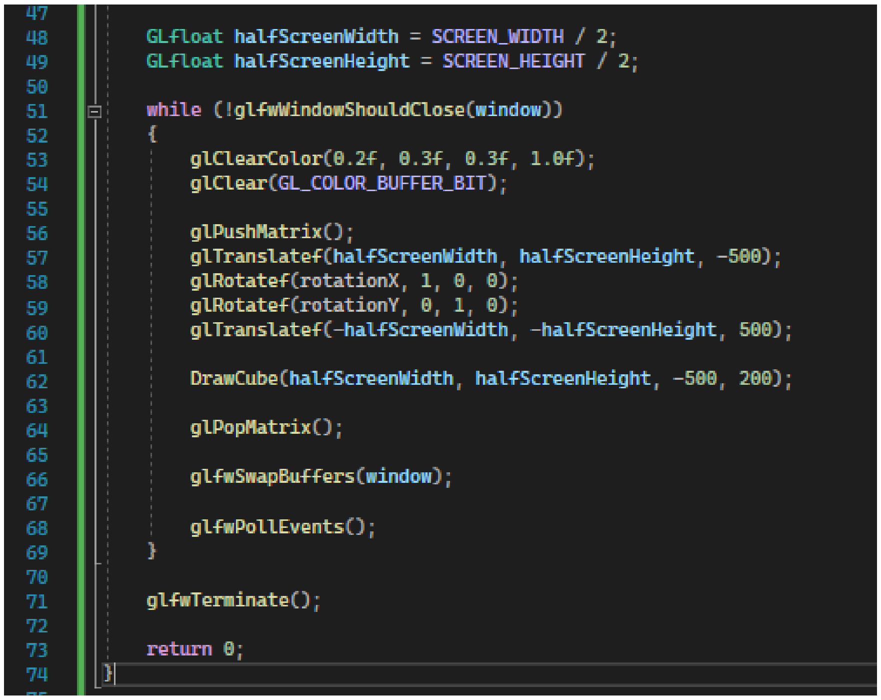
Implementation:

►For implementing our project, we are using the OpenGL library. First of all, we created a window screen with OpenGL context for drawing our cube. Our screen width was 640px and height was 480px. Then we define the projection matrix. The projection matrix defines the camera’s properties that view the objects in the world coordinate frame. In this part, we set the aspect ratio and the near and far clipping planes. After that, we make cubes front, back, left, right, upper, and lower sides. Then we make it a movable cube by using our keyboards with all four arrow keys. By using these 4 keys we can see all the sides of this cube in a single window, and it will be like a movable cube.

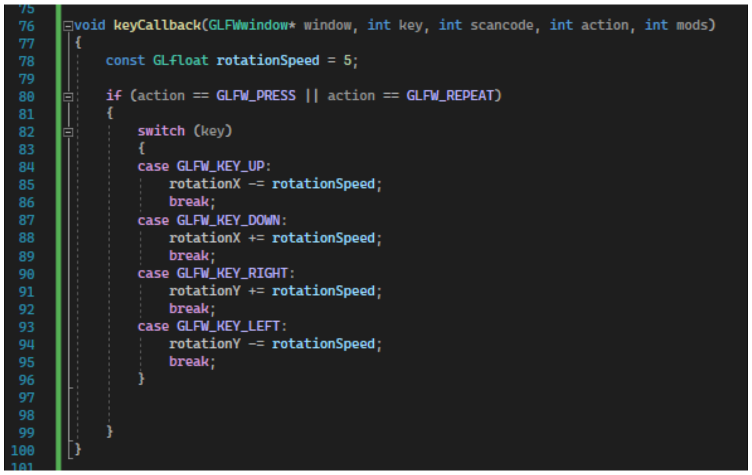
Code:



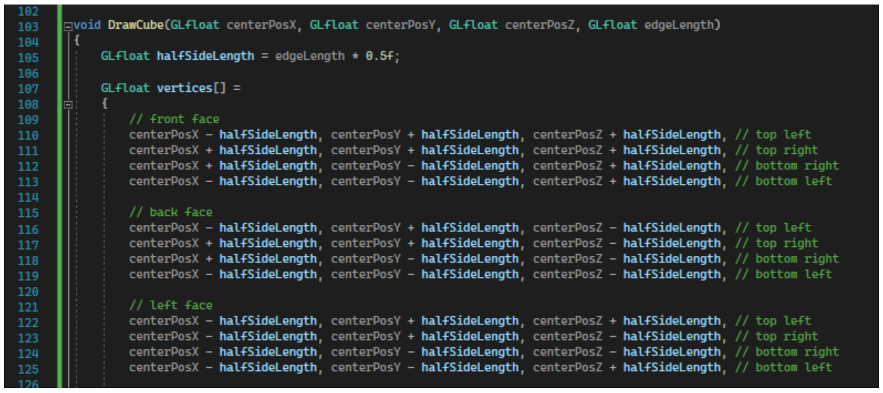
```
GLfloat rotationY = 0.0f;

int main(void)
{
    GLFWwindow* window;


    if (!glfwInit())
    {
        return -1;
    }

    window = glfwCreateWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Moveable cube", NULL, NULL);

    glfwSetKeyCallback(window, keyCallback);
    glfwSetInputMode(window, GLFW_STICKY_KEYS, 1);

    int screenWidth, screenHeight;
    glfwGetFramebufferSize(window, &screenWidth, &screenHeight);

    if (!window)
    {
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(window);

    glViewport(0.0f, 0.0f, screenWidth, screenHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, SCREEN_WIDTH, 0, SCREEN_HEIGHT, 0, 1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
```



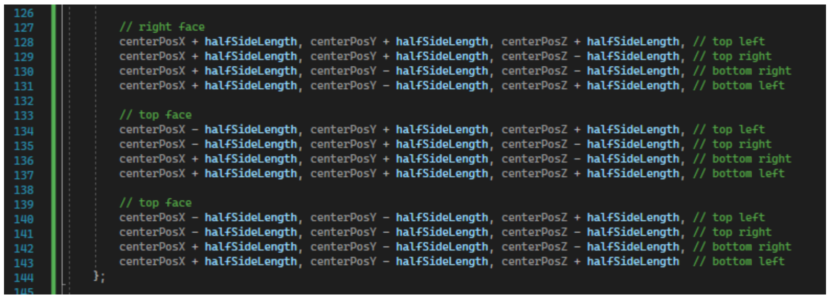
```cpp

    GLfloat halfScreenWidth = SCREEN_WIDTH / 2;
    GLfloat halfScreenHeight = SCREEN_HEIGHT / 2;

    while (!glfwWindowShouldClose(window))
    {
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glPushMatrix();
        glTranslatef(halfScreenWidth, halfScreenHeight, -500);
        glRotatef(rotationX, 1, 0, 0);
        glRotatef(rotationY, 0, 1, 0);
        glTranslatef(-halfScreenWidth, -halfScreenHeight, 500);

        DrawCube(halfScreenWidth, halfScreenHeight, -500, 200);

        glPopMatrix();

        glfwSwapBuffers(window);

        glfwPollEvents();
    }

    glfwTerminate();

    return 0;
}
```



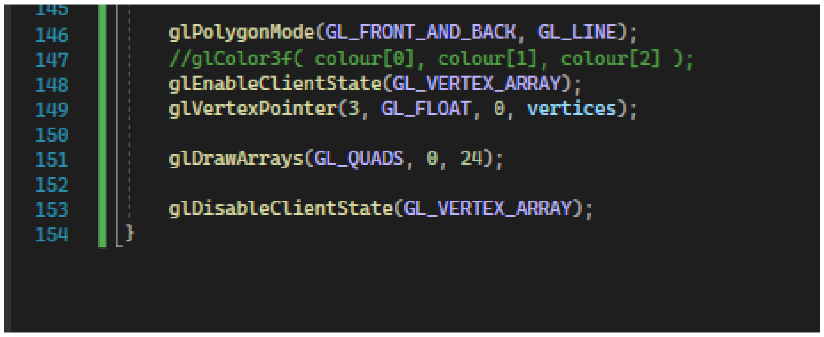
```cpp
void keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    const GLfloat rotationSpeed = 5;

    if (action == GLFW_PRESS || action == GLFW_REPEAT)
    {
        switch (key)
        {
        case GLFW_KEY_UP:
            rotationX -= rotationSpeed;
            break;
        case GLFW_KEY_DOWN:
            rotationX += rotationSpeed;
            break;
        case GLFW_KEY_RIGHT:
            rotationY += rotationSpeed;
            break;
        case GLFW_KEY_LEFT:
            rotationY -= rotationSpeed;
            break;
        }


    }
}
```



```cpp

void DrawCube(GLfloat centerPosX, GLfloat centerPosY, GLfloat centerPosZ, GLfloat edgeLength)
{
    GLfloat halfSideLength = edgeLength * 0.5f;

    GLfloat vertices[] =
    {
        // front face
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // top left
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // top right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength, // bottom right
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength, // bottom left

        // back face
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // top left
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // top right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // bottom right
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // bottom left

        // left face
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // top left
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // top right
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // bottom right
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength, // bottom left

```



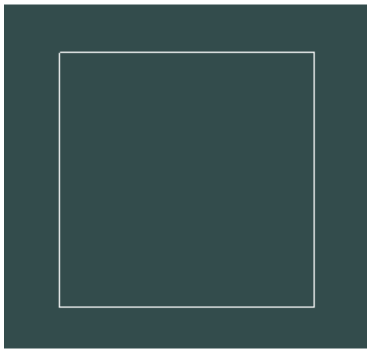
```
        // right face
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // top left
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // top right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // bottom right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength, // bottom left

        // top face
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // top left
        centerPosX - halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // top right
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ - halfSideLength, // bottom right
        centerPosX + halfSideLength, centerPosY + halfSideLength, centerPosZ + halfSideLength, // bottom left

        // top face
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength, // top left
        centerPosX - halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // top right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ - halfSideLength, // bottom right
        centerPosX + halfSideLength, centerPosY - halfSideLength, centerPosZ + halfSideLength  // bottom left
    };
```

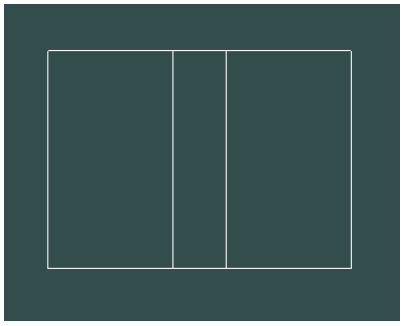


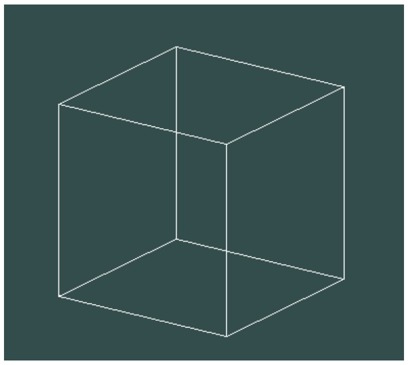
```

    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    //glColor3f( colour[0], colour[1], colour[2] );
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);

    glDrawArrays(GL_QUADS, 0, 24);

    glDisableClientState(GL_VERTEX_ARRAY);
}
```

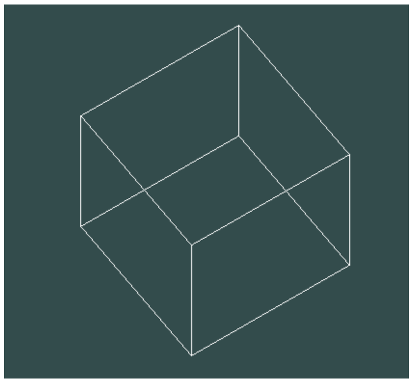
<u>Output:</u>

►Some of the random output screenshots given below-









Conclusion:

►We found designing and developing this 3D Movable Cube a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling, and screen management. Hope we will develop more updated projects like this project.