Scientific Visualization

First OpenGL HomeWork

Anand Kamble
Department of Scientific Computing
Florida State University

1 Introduction

In this OpenGL assignment, we aim to create a dynamic square using the glDrawElements function. Each square vertex will have a distinct color that changes over time, giving it a lively appearance. To add an extra layer of visual interest, we'll use the glm::rotate function to make the square rotate within the x-y plane. The Shader class from "shader.h" will help us implement shaders effectively. Please remember to submit a zipped file containing your CPP file, shader files, and a brief report showcasing your output. This assignment provides hands-on experience using OpenGL to create animated and visually appealing graphics.

2 Prerequisites

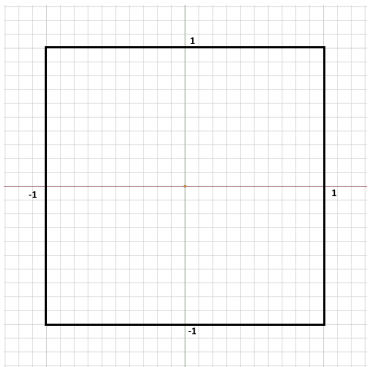
This project is tailored for Linux, and to establish the necessary environment, I've integrated GLFW, GL, and "shader.h" into the include directory for the g++ compiler.

For your convenience, you can download these essential files directly from the LearnOpenGL GitHub repository using the following link:
https://github.com/JoeyDeVries/LearnOpenGL/tree/master/includes

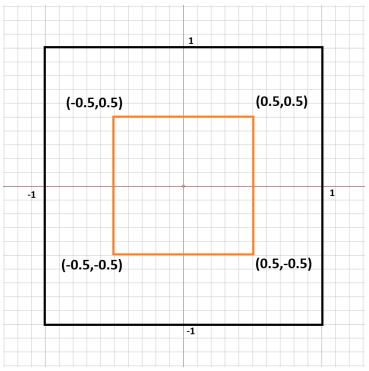
3 Drawing a Square

In OpenGL, we use normalized device coordinates x , y, and z which are in the range $[-1, 1]$. We are drawing the square in the X-Y plane so the coordinates of the 4 vertices of the square we want to draw will be like the following:

$$(0.5f, -0.5f, 0.0f), (-0.5f, -0.5f, 0.0f), (-0.5f, 0.5f, 0.0f), (0.5f, 0.5f, 0.0f)$$



(a) Normalized device coordinates



(b) Square

4 Implementation

First of all, to create a window we are using `glfwCreateWindow` function which is provided by the GLFW library.

```
/** @file main.cpp */
GLFWwindow *window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "First␣
    OpenGL␣Homework", NULL, NULL);
```

Then we define the vertices of our square along with its colors in an array of floats.

```
/** @file main.cpp */
float vertices[] = {
// positions          // colors
0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f,  // bottom right
-0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, // bottom left
-0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 1.0f,  // top left
0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f};  // top right
```

Since We are using two triangles to draw one square, we have to specify the indices for each triangle, this allows us to save memory by declaring the overlapping vertices only once. These indices are defined as follows:

```
/** @file main.cpp */
GLuint indices[] = {
    0, 1, 2,
    3, 2, 0};
```

5 Transformation

In this example, we are using the GLM library to calculate the transformation matrix at each frame, where all the vertices are rotated around the Z-axis with the given angle using the rotate function provided by GLM library.

```
/** @file main.cpp */
    // Update transformation matrix
    trans = rotate(trans, (delta)/radians(angleOfRotation), axis);
    ourShader.setMat4("trans", trans);
```

Here, we are also using the radians function provided by the GLM library to convert the angle of rotation from degree to radian. To calculate this angle of rotation we are using delta time which is calculated by using the values from `glfwGetTime` function. [1] Using this function rather than using a constant ensures that the program will work and look the same on different machines with different rendering speeds such as 60Hz or 144Hz. If we use a constant for rotation the sqaure will rotate much faster on a machine rendering at 144Hz than the machine with 60Hz. This time is calculated as follows:

```
/** @file main.cpp */
now = glfwGetTime(); // Get the current time
delta = now - lastTime; // Calculate the time difference
lastTime = now; // Update the last time
time += delta; // Update the time parameter
ourShader.setFloat("time", time); // Update the time parameter in the
    shader
```

6 Shader Implementation

Both shaders, vertex, and fragment shader are written in separate files named "shader" with extensions ".vs" for vertex and ".fs" for fragment. This files are then used with the OpenGL using the Shader class provided by "shader.h" library.

```
/** @file main.cpp */
    Shader ourShader("shader.vs", "shader.fs");
    ourShader.use();
```

Using this library, we are also defining the uniforms which will be required to share the transformation matrix and the time.

```
ourShader.setFloat("time", time);// Set time parameter in shader
ourShader.setMat4("trans", trans);//Set transformation matrix in shader
```

6.1 Vertex Shader

Inside the vertex shader we have defined the required uniforms which is the transformation matrix and using that to multiply it with the position matrix. From this shader we are also forwarding the color data to fragment shader through a variable named `aColor`.

```
out vec3 ourColor;
uniform mat4 trans;

void main() {
    gl_Position = trans * vec4(aPos, 1.0);
    ourColor = aColor;
}
```

6.2 Fragment Shader

In the fragment shader we are using the time and outColor variables to update the colors of the vertices. As the time progesses we are changing the color using the `sin` and `cos` function. Since the accepatable range of values for color is $[0, 1]$, we need to make sure that we don't exceed these limits to ensure a smooth transition between colors and avoid clipping. For that purpose we are using the `abs` function and also dividing the value by 2. To add some variation we are also dividing the time by 2 and 4.

```
in vec3 ourColor;

uniform float time;

void main() {
    FragColor = vec4((ourColor.x + abs(sin(time))/2),
    (ourColor.y + abs(cos(time/4))/2),
    (ourColor.z + abs(sin(time / 2)))/2, 1.0);
}
```

7 Result

After successful compilation and running the executable, we get the following window with a rotating square.

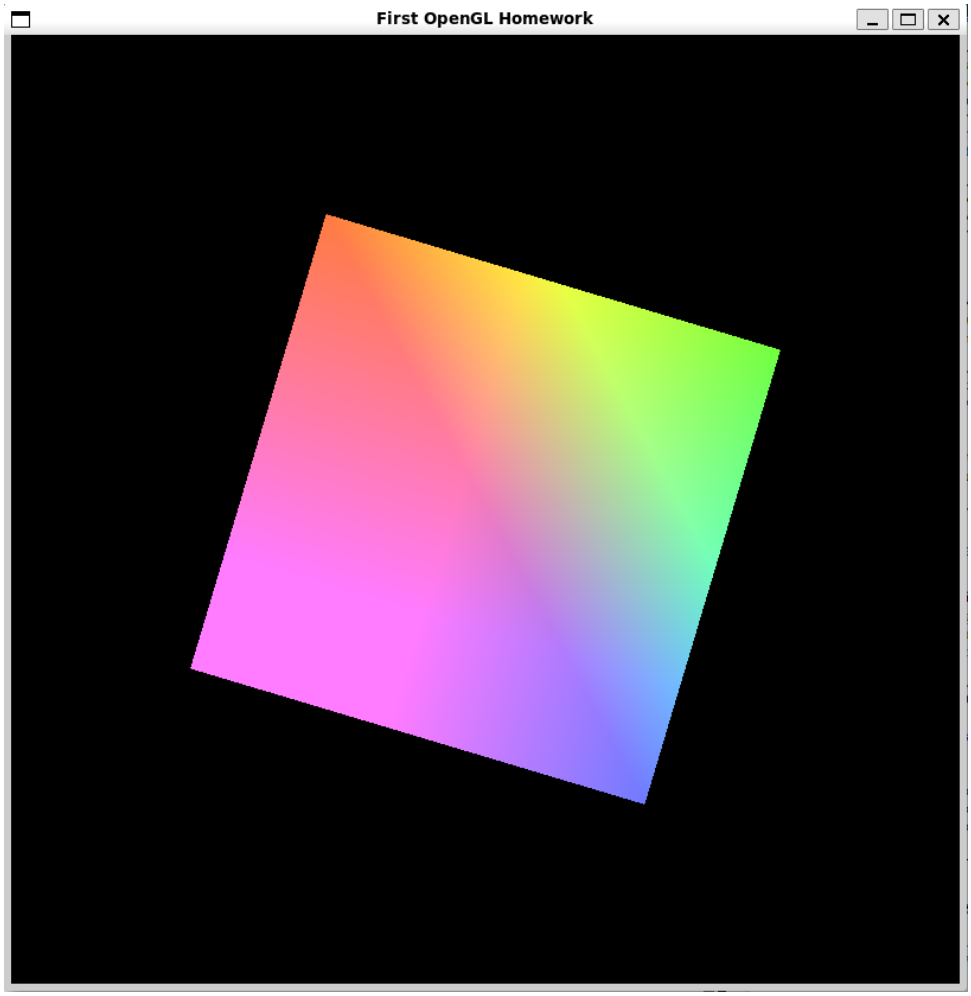


Figure 1: Caption

8 Errors and Debugging

While developing this project, the major challege was to setup and compiling the OpenGL program on linux, I did get a lot of information from ChatGPT to setup a Makefile for this project.[2] Also major bugs which I fixed included clipping of colors, rotation speed of the sqaure.

9 Future Enhancements

Currently, resizing the output window reshapes the square. In other words, the aspect ration of the outtput window directly affects the square which can be fixed by adding some variables which affect the scaling of square and updating those variables using `framebuffer_size_callback` function.

References

[1] Mazatwork. *Glm rotate object by time* , https://stackoverflow.com/a/47114300/22647897. 4 Nov, 2017.

[2] ChatGPT. *Fixing GLFW Init Error* , https://chat.openai.com/share/9bad68ec-2951-44f3-8f98-368d10858d3b. 2 Feb, 2017.

[3] Joey de Vries. *Learn OpenGL* , https://learnopengl.com/book/learnopengl_book_bw.pdf. Jun, 2017.