

Tutorial Report

D11315807

Ardiawan Bagus Harisa

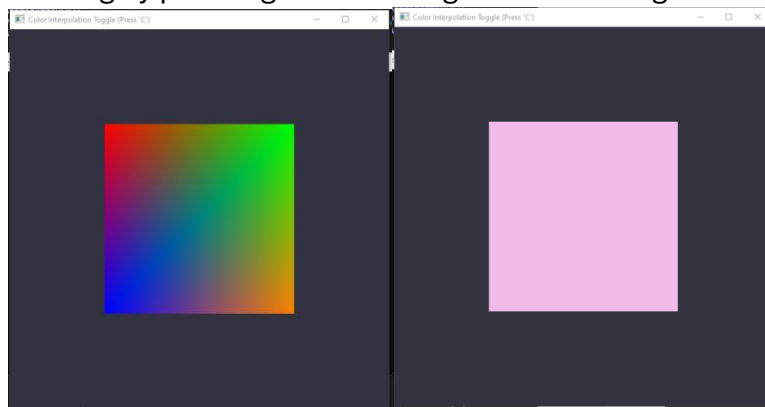
Department of CSIE

Exercise 1

1. Create a rectangle.
2. Interpolate colors, no EBO.
3. Change the color mode using key 'C'.

How to use my program:

1. First, you must have the freeglut and glew library installed.
2. For my convenience, I use VS Studio for debugging.
3. Just run the debug by pressing **F5**. You will get the following result:



Program:

1. Create the rectangle

Declares the shader object, VAO, and VBO, and the vertices (make it global here because we will need it to change the triangle's color). The var isSolidColor is used to determine the color mode.

```
32
33     GLuint shaderProgram;
34     GLuint VAO;
35     GLuint VBO;
36     GLint useSolidColorLoc;
37     GLint solidColorLoc;
38     bool isSolidColor = false;
```

There are two ways to draw the rectangle, first is using the 6 vertices but only 4 unique vertices, second, really just using 4 vertices.

First:

```
39
40 // 4 unique vertices but still using 6 vertices to form 2 triangles
41 float vertices[] = {
42     // positions & colors
43     -0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, // top-left & red
44     0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, // top-right & green
45     0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, // bottom-right & blue
46     -0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, // top-left & red
47     0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, // bottom-right & blue
48     -0.5f, -0.5f, 0.0f, 1.0f, 0.7f, 0.0f // bottom-left & orange
49 };
50
```

And draw

```
113
114 glBindVertexArray(VAO);
115 glDrawArrays(GL_TRIANGLES, 0, 6);
116 glBindVertexArray(0);
117
```

Second:

```
40
41 float vertices[] = {
42     -0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, // red
43     0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, // green
44     -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, // blue
45     0.5f, -0.5f, 0.0f, 1.0f, 0.5f, 0.0f // orange
46 };
47
```

And draw

```
113
114 glBindVertexArray(VAO);
115 //glDrawArrays(GL_TRIANGLES, 0, 6);
116 glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
117 glBindVertexArray(0);
118
```

2. Interpolate color

Because there are two modes of color, I change the coloring in the display() function. The variable isSolidColor is used to check the color mode. If using the solid color mode, I make the flat color to be randomized by randomizing each r, g, b value. And pass the value to the uniform function.

In the glUniform1i(), useSolidColorLoc is the location of the uniform boolean isSolidColor in the shader. The glUniform1i() itself is used to send an int or Boolean value to a uniform. Telling the shader program if the solid color is used or not.

These lines determine which

```

85 // Get uniform locations
86 useSolidColorLoc = glGetUniformLocation(shaderProgram, "useSolidColor");
87 solidColorLoc = glGetUniformLocation(shaderProgram, "solidColor");
88 }

```

The `glUniform3f()` is used to set the RGB value (in which I randomized if the `isSolidColor` is true), in the fragment shader.

```

106 void display() {
107     glClearColor(0.2f, 0.2f, 0.25f, 1.0f);
108     glClear(GL_COLOR_BUFFER_BIT);
109
110     glUseProgram(shaderProgram);
111     glUniform1i(useSolidColorLoc, isSolidColor);
112     float r = rand() % 256 / 255.0f; // Random color
113     float g = rand() % 256 / 255.0f; // Random color
114     float b = rand() % 256 / 255.0f; // Random color
115     //std::cout << "Random color: " << r << std::endl;
116     glUniform3f(solidColorLoc, r, g, b);
117
118     glBindVertexArray(VAO);
119     //glDrawArrays(GL_TRIANGLES, 0, 6);
120     glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
121     glBindVertexArray(0);
122
123     glFlush();
124 }

```

Because I tell the shader program to switch the color being used according to the state of `isSolidColor`.

```

5 const char* vertexShaderSource = R"(
6 #version 330 core
7 layout(location = 0) in vec3 aPos;
8 layout(location = 1) in vec3 aColor;
9 out vec3 vertexColor;
10 uniform bool useSolidColor;
11 void main() {
12     gl_Position = vec4(aPos, 1.0);
13     if (!useSolidColor)
14         vertexColor = aColor;
15     else
16         // will be overridden in fragment, dummy value
17         vertexColor = vec3(0.0);
18 }
19 )";
20
21 const char* fragmentShaderSource = R"(
22 #version 330 core
23 in vec3 vertexColor;
24 out vec4 FragColor;
25 uniform bool useSolidColor;
26 uniform vec3 solidColor;
27 void main() {
28     // use solid color?
29     FragColor = useSolidColor ? vec4(solidColor, 1.0) : vec4(vertexColor, 1.0);
30 }
31 )";

```

3. Switch the color mode using “C”

To detect the if a user press the “C” button on keyboard, regardless of capitalized or not, assign the keyboard event to the `glutKeyboardFunction()`.

```

125
126 void keyPressed(unsigned char key, int x, int y) {
127     if (key == 'C' || key == 'c') {
128         isSolidColor = !isSolidColor;
129         glutPostRedisplay();
130     }
131 }
132
133 int main(int argc, char** argv) {
134     glutInit(&argc, argv);
135     glutInitContextVersion(3, 3);
136     glutInitContextProfile(GLUT_CORE_PROFILE);
137     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
138     glutInitWindowSize(600, 600);
139     glutCreateWindow("Color Interpolation Toggle (Press 'C')");
140
141     glewExperimental = GL_TRUE;
142     if (glewInit() != GLEW_OK) { ... }
143
144     initShaders();
145     initBuffers();
146
147     glutDisplayFunc(display);
148     glutKeyboardFunc(keyPressed);
149
150     glutMainLoop();
151     return 0;
152 }
153
154
155
156

```

Other things I need to do just like the previous project, is to declare the shader program for vertex and fragment.

```

4
5 const char* vertexShaderSource = R"(
6 #version 330 core
7 layout(location = 0) in vec3 aPos;
8 layout(location = 1) in vec3 aColor;
9 out vec3 vertexColor;
10 uniform bool useSolidColor;
11 void main() {
12     gl_Position = vec4(aPos, 1.0);
13     if (!useSolidColor)
14         vertexColor = aColor;
15     else
16         // will be overridden in fragment, dummy value
17         vertexColor = vec3(0.0);
18 }
19 )";
20
21 const char* fragmentShaderSource = R"(
22 #version 330 core
23 in vec3 vertexColor;
24 out vec4 FragColor;
25 uniform bool useSolidColor;
26 uniform vec3 solidColor;
27 void main() {
28     // use solid color?
29     FragColor = useSolidColor ? vec4(solidColor, 1.0) : vec4(vertexColor, 1.0);
30 }
31 )";
32

```

Then, prepare the buffer objects to draw the rectangle statically.

```

90 void initBuffers() {
91     glGenVertexArrays(1, &VAO);
92     glGenBuffers(1, &VBO);
93     glBindVertexArray(VAO);
94     glBindBuffer(GL_ARRAY_BUFFER, VBO);
95     glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
96
97     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
98     glEnableVertexAttribArray(0);
99     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
100    glEnableVertexAttribArray(1);
101
102    glBindBuffer(GL_ARRAY_BUFFER, 0);
103    glBindVertexArray(0);
104 }
105

```

```

48
49 void initShaders() {
50     GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
51     glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
52     glCompileShader(vertexShader);
53     GLint success;
54     glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
55     if (!success) { ... }
56
57     GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
58     glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
59     glCompileShader(fragmentShader);
60     glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
61     if (!success) { ... }
62
63     shaderProgram = glCreateProgram();
64     glAttachShader(shaderProgram, vertexShader);
65     glAttachShader(shaderProgram, fragmentShader);
66     glLinkProgram(shaderProgram);
67     glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
68     if (!success) { ... }
69
70     glDeleteShader(vertexShader);
71     glDeleteShader(fragmentShader);
72
73     // Get uniform locations
74     useSolidColorLoc = glGetUniformLocation(shaderProgram, "useSolidColor");
75     solidColorLoc = glGetUniformLocation(shaderProgram, "solidColor");
76 }
77

```

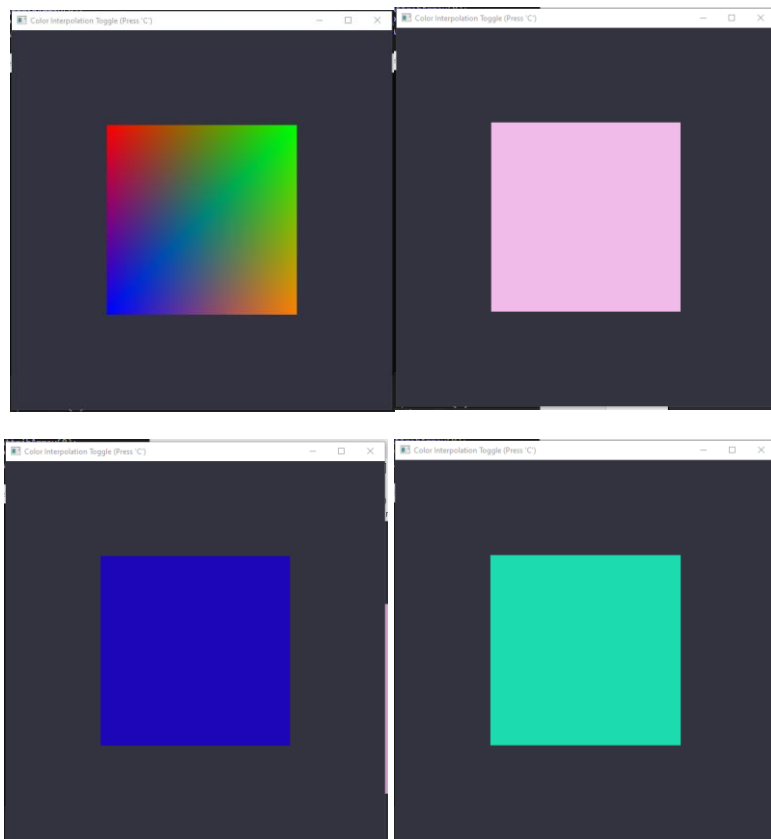
Also don't forget to initialize the shader. Here I assign the color to the associated uniform. I just realize what a uniform is after quite sometimes. Uniform is variable in the GLSL shader program that we can set to define attributes. It is constant in GLSL, but we can change from the main program of c++. It can be used for every vertex or fragment across all shader invocations. The tutorial said that it is usually used for:

- Color
- Positions
- Transformation
- Light
- Texture
- Etc

In short, it is like a global configuration file for our shader.

Term	Scope	Changes per...	Set by
attribute	vertex shader	vertex	your buffer data
varying / out-in	between shaders	interpolated	automatically by GPU
uniform	global	draw call	you (from CPU/OpenGL code)

Results:



Source code:

<https://github.com/ardiawanbagusharisa/cgopengl/tree/main/Tutorial%20Class%20Rectangleangle>

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>

const char* vertexShaderSource = R"(
#version 330 core
```

```

layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aColor;
out vec3 vertexColor;
uniform bool useSolidColor;
void main() {
    gl_Position = vec4(aPos, 1.0);
    if (!useSolidColor)
        vertexColor = aColor;
    else
        // will be overridden in fragment, dummy value
        vertexColor = vec3(0.0);
}
}";

const char* fragmentShaderSource = R"(
#version 330 core
in vec3 vertexColor;
out vec4 FragColor;
uniform bool useSolidColor;
uniform vec3 solidColor;
void main() {
    // use solid color?
    FragColor = useSolidColor ? vec4(solidColor, 1.0) : vec4(vertexColor, 1.0);
}
)";

GLuint shaderProgram;
GLuint VAO;
GLuint VBO;
GLint useSolidColorLoc;
GLint solidColorLoc;
bool isSolidColor = false;

float vertices[] = {
    -0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  0.0f, // red
    0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  0.0f, // green
    -0.5f, -0.5f,  0.0f,  0.0f,  0.0f,  1.0f, // blue
    0.5f, -0.5f,  0.0f,  1.0f,  0.5f,  0.0f // orange
};

void initShaders() {
    GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    GLint success;
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success) {
        char infoLog[512];
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cerr << "Vertex shader error: " << infoLog << std::endl;
    }

    GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
    if (!success) {
        char infoLog[512];
        glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
        std::cerr << "Fragment shader error: " << infoLog << std::endl;
    }

    shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);

```

```

    if (!success) {
        char infoLog[512];
        glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
        std::cerr << "Shader linking error: " << infoLog << std::endl;
    }

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    // Get uniform locations
    useSolidColorLoc = glGetUniformLocation(shaderProgram, "useSolidColor");
    solidColorLoc = glGetUniformLocation(shaderProgram, "solidColor");
}

void initBuffers() {
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}

void display() {
    glClearColor(0.2f, 0.2f, 0.25f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(shaderProgram);
    glUniform1i(useSolidColorLoc, isSolidColor);
    float r = rand() % 256 / 255.0f; // Random color
    float g = rand() % 256 / 255.0f; // Random color
    float b = rand() % 256 / 255.0f; // Random color
    //std::cout << "Random color: " << r << std::endl;
    glUniform3f(solidColorLoc, r, g, b);

    glBindVertexArray(VAO);
    //glDrawArrays(GL_TRIANGLES, 0, 6);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    glBindVertexArray(0);

    glFlush();
}

void keyPress(unsigned char key, int x, int y) {
    if (key == 'C' || key == 'c') {
        isSolidColor = !isSolidColor;
        glutPostRedisplay();
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitContextVersion(3, 3);
    glutInitContextProfile(GLUT_CORE_PROFILE);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Color Interpolation Toggle (Press 'C')");

    glewExperimental = GL_TRUE;
    if (glewInit() != GLEW_OK) {
        std::cerr << "GLEW Initialization Failed!" << std::endl;
        return -1;
    }
}

```



```
    }  
    initShaders();  
    initBuffers();  
  
    glutDisplayFunc(display);  
    glutKeyboardFunc(keyPress);  
  
    glutMainLoop();  
    return 0;  
}
```