**Tutorial Report**

D11315807

Ardiawan Bagus Harisa

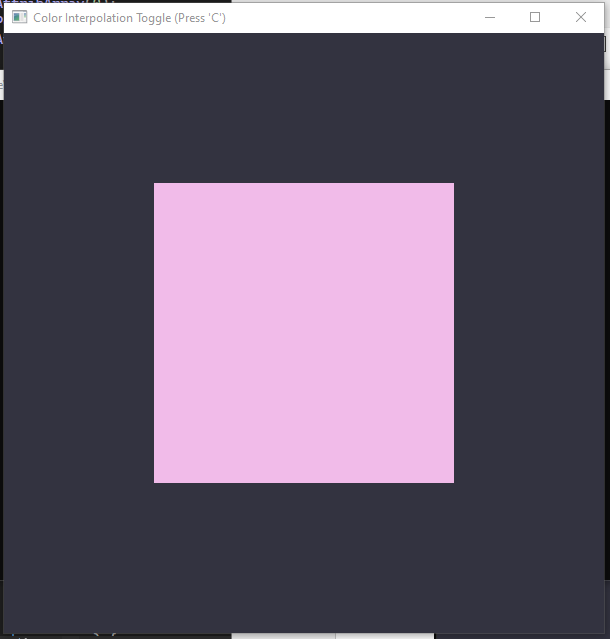
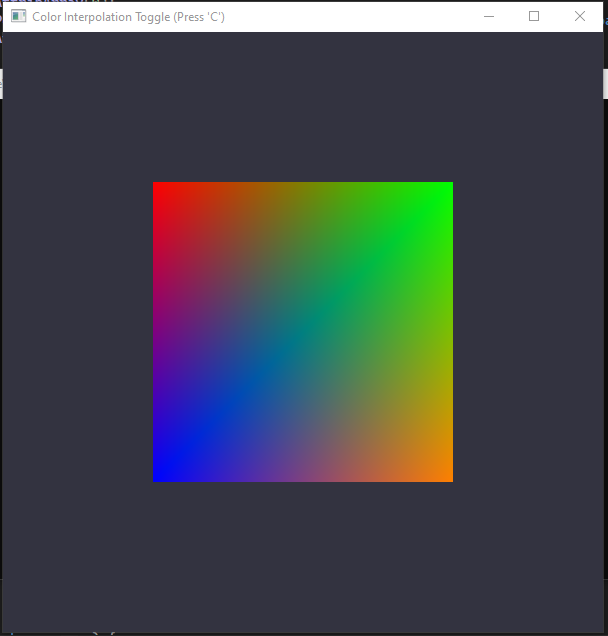
Department of CSIE

**Exercise 1**

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

**How to use my program:**

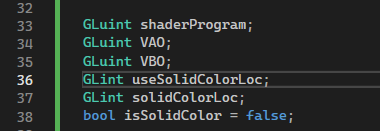
1. First, you must have the freeglut and glew library installed.
2. For my convenience, I use VS Studio for debugging.
3. Just run the debug by pressing **F5**. You will get the following result:

****

**Program:**

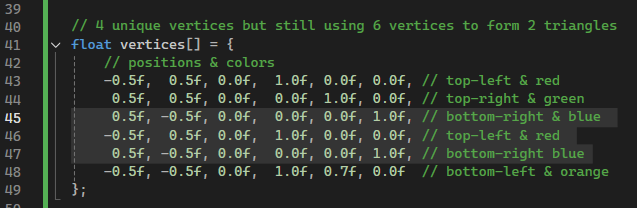
1. **Create the rectangle**

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.

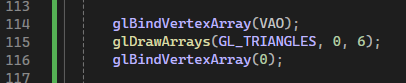


There are two ways to draw the rectangle, first is using the 6 vertices but only 4 unique vertices, second, really just using 4 vertices.

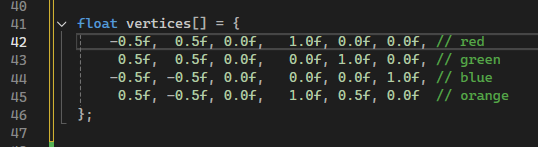
First:



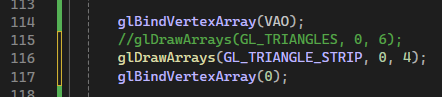
And draw



Second:



And draw

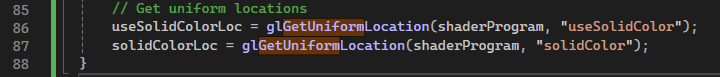


1. **Interpolate color**

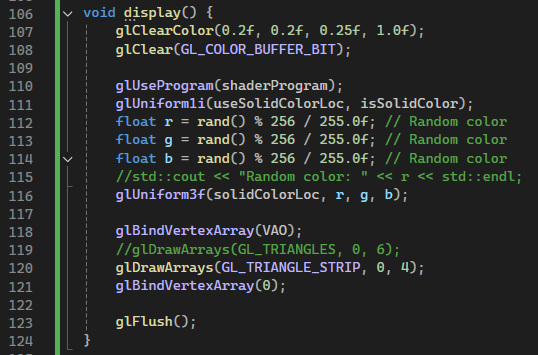
Because there are two modes of color, I change the coloring in the display() function. The variable isSolidColor is used to check the color mode. If using the solid color mode, I make the flat color to be randomized by randomizing each r, g, b value. And pass the value to the uniform function.

In the glUniform1i(), useSolidColorLoc is the location of the uniform boolean isSolidColor in the shader. The glUnifrom1i() 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.

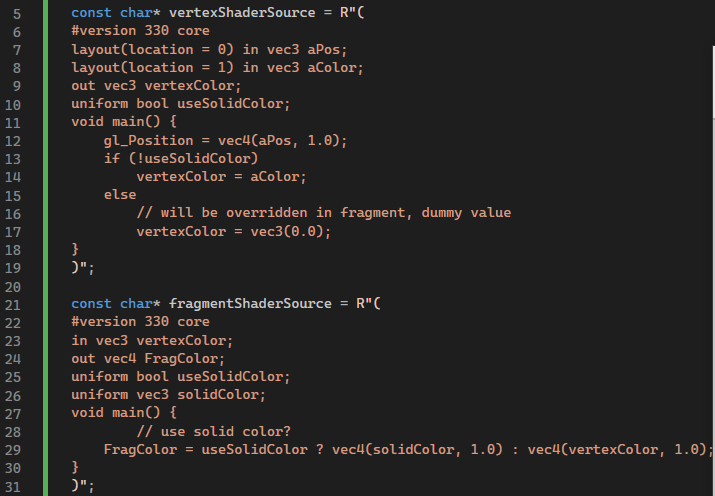
These lines determine which



The glUniform3F() is used to set the RGB value (in which I randomized if the isSolidColor is true), in the fragment shader.



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

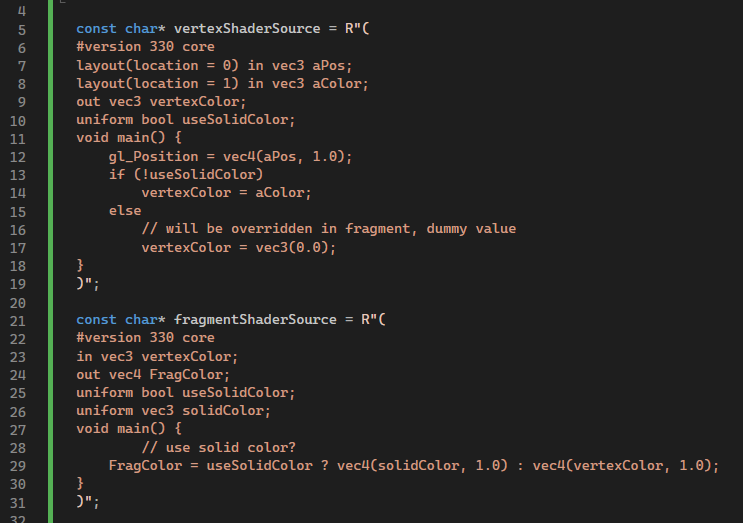


1. **Switch the color mode using “C”**

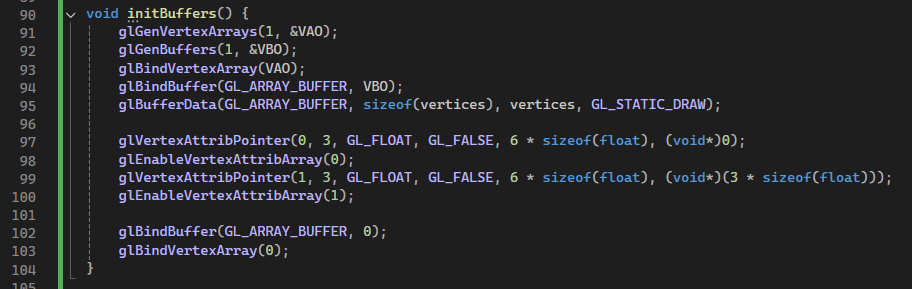
To detect the if a user press the “C” button on keyboard, regardless of capitalized or not, assign the keyboard event to the glutKeyboardFunction().

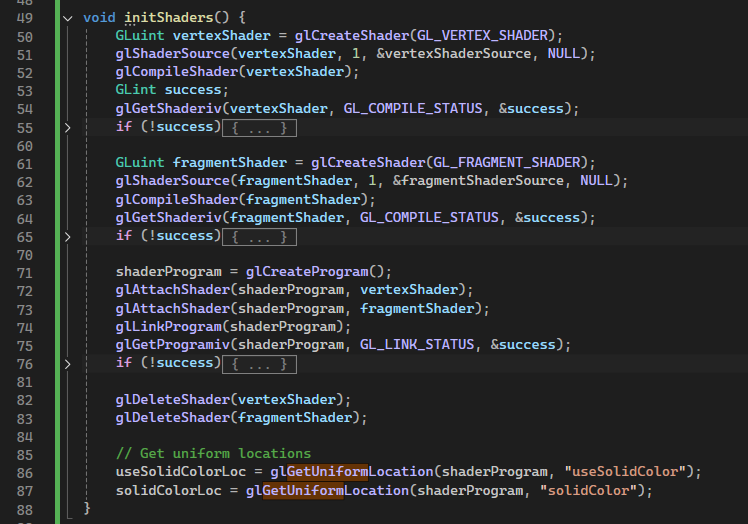


Other things I need to do just like the previous project, is to declare the shader program for vertex and fragment.



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

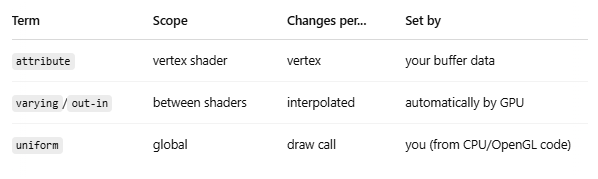




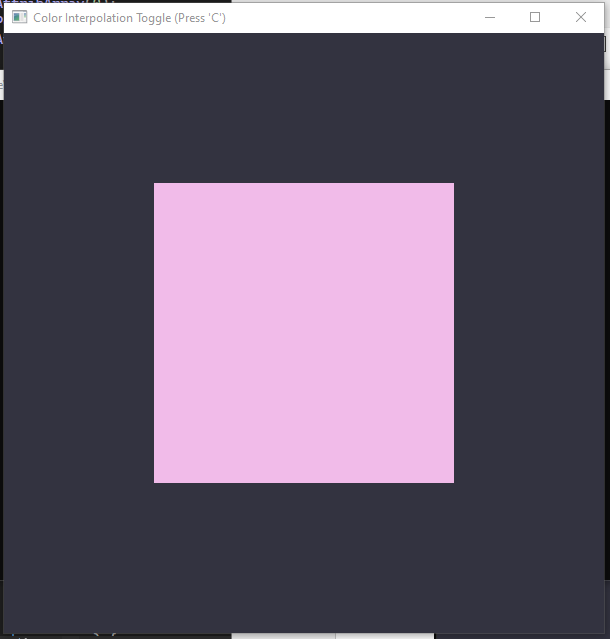
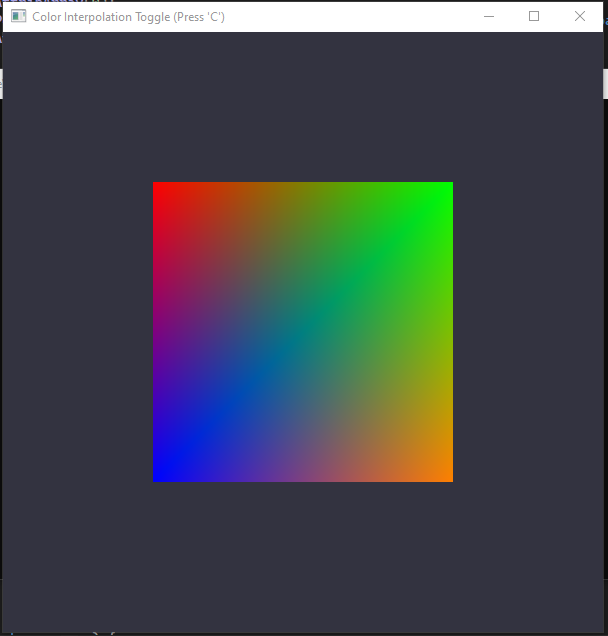
Also don’t forget to initialize the shader. Here I assign the color to the associated uniform. I just realize what a uniform is after quite sometimes. Uniform is variable in the GLSL shader program that we can set to define attributes. It is constant in GLSL, but we can change from the main program of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

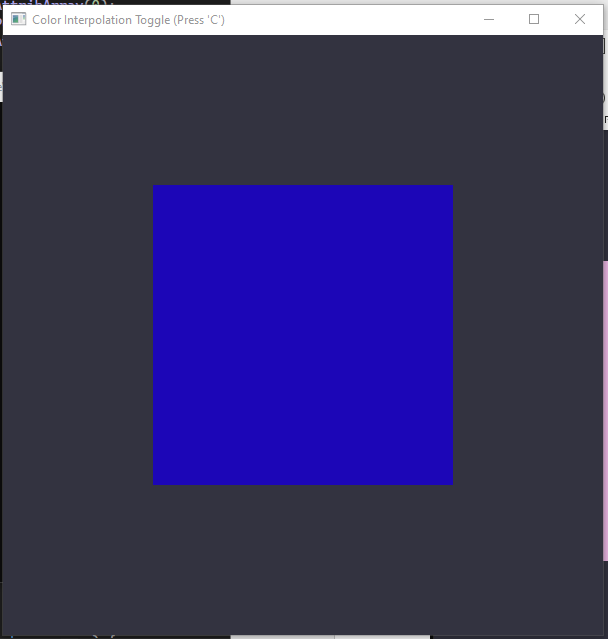
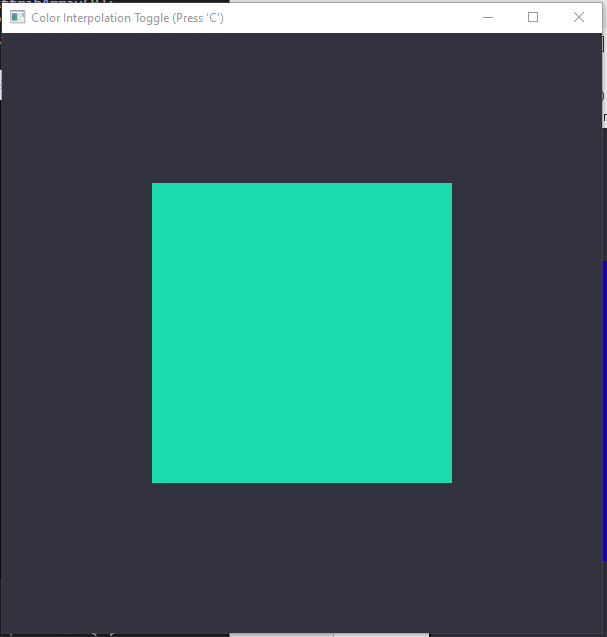
* Color
* Positions
* Transformation
* Light
* Texture
* Etc

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



**Results:**

****

**** ****

**Source code:**

[**https://github.com/ardiawanbagusharisa/cgopengl/tree/main/Tutorial%20Class%20Rectangle**](https://github.com/ardiawanbagusharisa/cgopengl/tree/main/Tutorial%20Class%20Rectangle)

#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;

}