Project 3 Report

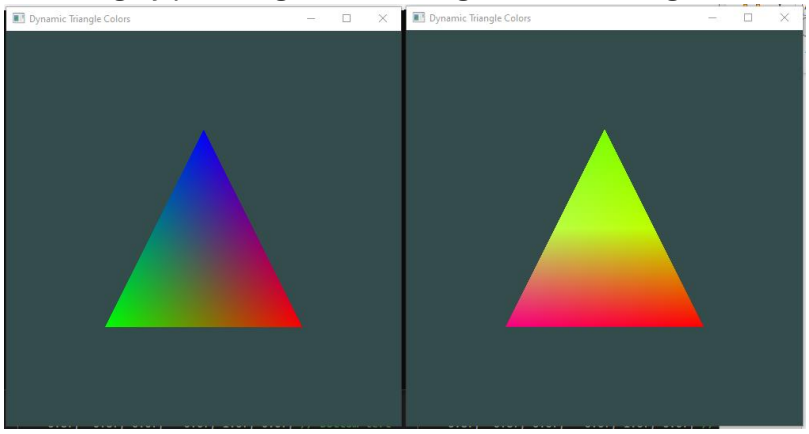
D11315807
Ardiawan Bagus Harisa
Department of CSIE

Exercise 2

1. Create a triangle using VBO + VAO.
2. When the mouse is moved, use its (x, y) position to change the color of one of the triangle's vertices.
3. Update that vertex's color data in the VBO using glBufferSubData() every frame.

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 Triangle**

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).

```
25  GLuint shaderProgram;
26  GLuint VAO;
27  GLuint VBO;
28
29  // Move vertex data as global var
30  float vertices[] = {
31      // positions & colors
32       0.0f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f, // top
33      -0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f, // bottom left
34       0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 0.0f  // bottom right
35  };
36
```

2. **Detect mouse position, use to change color**

After I can detect the mouse event (position), I used that information to update the color of each vertex according to my rule. Because the screen Y is the opposite of the OpenGL coordinate system, we must use 1 - y value. So it is kind of mechanism to normalize the screen device coordinate.

In the loop of vertices,

- Each vertices data store the position and the color (x,y,z,r,g,b). So that's why the base offset is 3 + i*6 (floats) in total.
- For each line of vertices, the color will be calculated as shown in the comment.

```
94
95  void mouseMove(int x, int y) {
96      // Variables for the mouse position
97      int width = glutGet(GLUT_WINDOW_WIDTH);
98      int height = glutGet(GLUT_WINDOW_HEIGHT);
99      float normX = (float)x / width;
100     float normY = 1.0f - (float)y / height;
101
102     // Variables for dynamic colors based on mouse position
103     float r = normX;
104     float g = normY;
105     float b = 1.0f - normX * normY;
106
107     // Update all 3 vertex colors
108     for (int i = 0; i < 3; ++i) {
109         int base = 3 + i * 6; // the color offset in array
110         vertices[base + 0] = r * ((i + 1) % 3); // For each iteration, this should be: 3, 9, 15
111         vertices[base + 1] = g * ((i + 2) % 3);
112         vertices[base + 2] = b * ((i + 3) % 3);
113     }
114     /*
115     summary of the colors:
116     Vertex i | ((i+1)%3) | ((i+2)%3) | ((i+3)%3) | Resulting color (r, g, b)
117           0 |         1 | 2         | 0         | (r, g, 0)
118           1 |         2 | 0         | 1         | (2r, 0, b) → then clamped
119           2 |         0 | 1         | 2         | (0, g, b)
120     since r/g/b are normalized (between 0.0 and 1.0),
121     multiplying by 0, 1, or 2 just scales to the max or min of the channel.
122     */
123
```

3. Update vertex color data in VBO using glBufferSubData()

Create the vertex shader and fragment shader.

```
const char* vertexShaderSource = R"(
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aColor;
out vec3 vertexColor;
void main() {
    gl_Position = vec4(aPos, 1.0);
    vertexColor = aColor;
}
)";

const char* fragmentShaderSource = R"(
#version 330 core
in vec3 vertexColor;
out vec4 FragColor;
void main() {
    FragColor = vec4(vertexColor, 1.0);
}
)";
```

Similar to the previous project's method, I tried to compile the vertex shader and the fragment shader. Then tried to attach them to the shader program. Also, delete the shader because it is not needed as a standalone program.

```
void initShaders() {
    // Vertex Shader
    GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    GLint success;
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success) { ... }

    // Fragment Shader
    GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
    if (!success) { ... }

    // Attach both shaders to a program
    shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success) { ... }

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
}
```

Similarly to the project 3 and 4, initialize our buffers: VAO and VBO. Here, the difference is the GL_DYNAMIC_DRAW, because we want to change the triangle's color dynamically (the vertex data will change a lot.

Line 86 means, we use parameters of location 0 (pos in vertex shader)→ 3 float (x,y,z), interval at 6 floats (3 positions and 3 colors) at the offset 0. While the line 88 means, the generally same idea but start after the first index, index = 1 (color), with the offset = 3xsize of float.

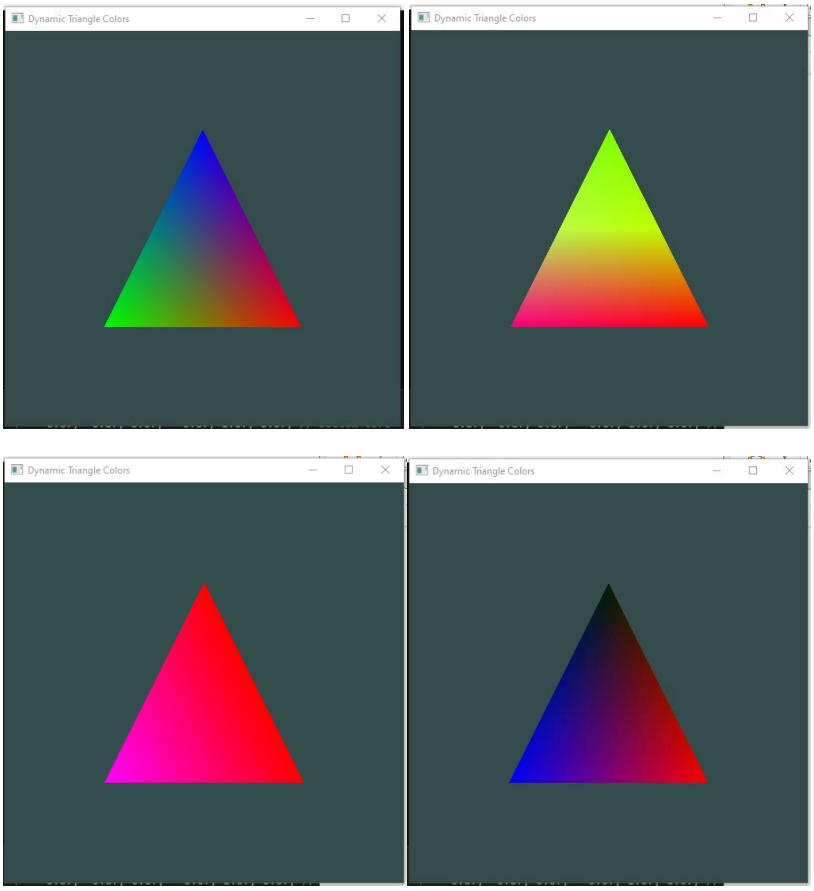
```
76
77  void initBuffers() {
78      // Generate the VAO and VBO, and bind them
79      glGenVertexArrays(1, &VAO);
80      glGenBuffers(1, &VBO);
81      glBindVertexArray(VAO);
82      glBindBuffer(GL_ARRAY_BUFFER, VBO);
83      glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_DYNAMIC_DRAW);
84
85      // Set vertex attribute pointers: position and color
86      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
87      glEnableVertexAttribArray(0);
88      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
89      glEnableVertexAttribArray(1);
90
91      glBindBuffer(GL_ARRAY_BUFFER, 0);
92      glBindVertexArray(0);
93  }
94
```

Just call the drawing function in the display. Then, call display into the main function. And run it.

```
127 void display() {
128     glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
129     glClear(GL_COLOR_BUFFER_BIT);
130
131     // Update all vertex colors (3 vertices × 3 floats)
132     glBindBuffer(GL_ARRAY_BUFFER, VBO);
133     glBufferSubData(GL_ARRAY_BUFFER, 3 * sizeof(float), 3 * 3 * sizeof(float), &vertices[3]);
134     glBindBuffer(GL_ARRAY_BUFFER, 0);
135
136     glUseProgram(shaderProgram);
137     glBindVertexArray(VAO);
138     glDrawArrays(GL_TRIANGLES, 0, 3);
139
140     glFlush();
141 }
142
```

```
143 int main(int argc, char** argv) {
144     glutInit(&argc, argv);
145     glutInitContextVersion(3, 3);
146     glutInitContextProfile(GLUT_CORE_PROFILE);
147     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
148     glutInitWindowSize(500, 500);
149     glutCreateWindow("Dynamic Triangle Colors");
150
151     glewExperimental = GL_TRUE;
152     GLenum err = glewInit();
153     if (err != GLEW_OK) { ... }
157
158     initShaders();
159     initBuffers();
160
161     glutDisplayFunc(display);
162     glutPassiveMotionFunc(mouseMove);
163
164     glutMainLoop();
165     return 0;
166 }
167
```

Results:



Source code:

https://github.com/ardiawanbagusharisa/cgopengl/tree/main/Tutorial%20Class%20OpenGL%20VBO%20VAO%20no%20EBO

```
#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;
void main() {
    gl_Position = vec4(aPos, 1.0);
    vertexColor = aColor;
}
)";

const char* fragmentShaderSource = R"(
#version 330 core
in vec3 vertexColor;
out vec4 FragColor;
void main() {
```

```
    FragColor = vec4(vertexColor, 1.0);
}
)";

GLuint shaderProgram;
GLuint VAO;
GLuint VBO;

// Move vertex data as global var
float vertices[] = {
    // positions & colors
     0.0f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f, // top
    -0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f, // bottom left
     0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 0.0f  // bottom right
};

void initShaders() {
       // Vertex Shader
    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 << "Failed to compile vertex shader\n" << infoLog << std::endl;
    }

       // Fragment Shader
    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 << "Failed to compile fragment shader\n" << infoLog << std::endl;
    }

       // Attach both shaders to a program
    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 << "Failed to link shaders\n" << infoLog << std::endl;
    }

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
}

void initBuffers() {
    // Generate the VAO and VBO, and bind them
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glBindVertexArray(VAO);
```

```
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_DYNAMIC_DRAW);

        // Set vertex attribute pointers: position and color
    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 mouseMove(int x, int y) {
        // Variables for the mouse position
    int width = glutGet(GLUT_WINDOW_WIDTH);
    int height = glutGet(GLUT_WINDOW_HEIGHT);
    float normX = (float)x / width;
    float normY = 1.0f - (float)y / height;

        // Variables for dynamic colors based on mouse position
    float r = normX;
    float g = normY;
    float b = 1.0f - normX * normY;

    // Update all 3 vertex colors
    for (int i = 0; i < 3; ++i) {
        int base = 3 + i * 6; // the color offset in array
               vertices[base + 0] = r * ((i + 1) % 3); // For each iteration, this should be: 3,
9, 15
        vertices[base + 1] = g * ((i + 2) % 3);
               vertices[base + 2] = b * ((i + 3) % 3);
    }
    /*
    summary of the colors:
    Vertex i | ((i+1)%3) | ((i+2)%3) | ((i+3)%3) | Resulting color (r, g, b)
           0 |         1 | 2         | 0         | (r, g, 0)
           1 |         2 | 0         | 1         | (2r, 0, b) → then clamped
           2 |         0 | 1         | 2         | (0, g, b)
    since r/g/b are normalized (between 0.0 and 1.0),
    multiplying by 0, 1, or 2 just scales to the max or min of the channel.
    */

    glutPostRedisplay();
}

void display() {
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // Update all vertex colors (3 vertices × 3 floats)
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferSubData(GL_ARRAY_BUFFER, 3 * sizeof(float), 3 * 3 * sizeof(float), &vertices[3]);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 3);

    glFlush();
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitContextVersion(3, 3);
    glutInitContextProfile(GLUT_CORE_PROFILE);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Dynamic Triangle Colors");

    glewExperimental = GL_TRUE;
    GLenum err = glewInit();
    if (err != GLEW_OK) {
        std::cerr << "GLEW Error: " << glewGetErrorString(err) << std::endl;
        return -1;
    }

    initShaders();
    initBuffers();

    glutDisplayFunc(display);
    glutPassiveMotionFunc(mouseMove);

    glutMainLoop();
    return 0;
}
```