Project 4 Report - EBO

D11315807
Ardiawan Bagus Harisa
Department of CSIE

Tasks:

1. Define four vertices for a rectangle.

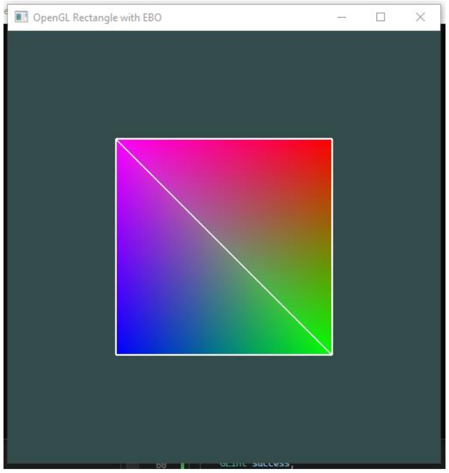
2. Define six indices for two triangles.

3. Bind an EBO with the indices.

4. Use glDrawElements() to draw.

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:

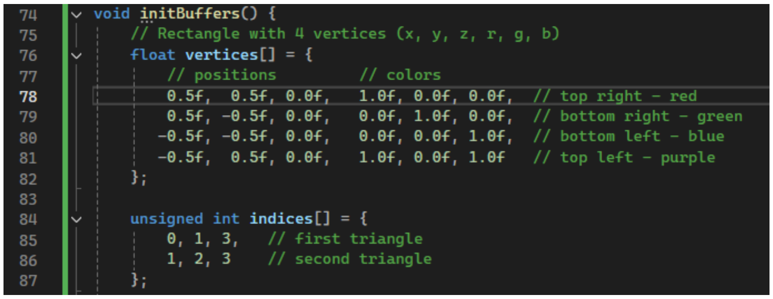


Program:

1. **Define a rectangle**

Similar to the Project 3, we need to provide the points/ vertices to define the rectangle. I am still using this tutorial as reference: https://learnopengl.com/Getting-started/Hello-Triangle.

Here we must define the four vertices that will be used to define the rectangle (which is actually constructed from two triangles). All vertices for the shape is declared in *vertices[]* array, and the triangles are actually declared in the *indices[]* array.

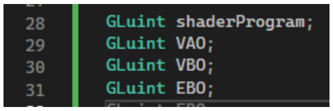


```
void initBuffers() {
    // Rectangle with 4 vertices (x, y, z, r, g, b)
    float vertices[] = {
        // positions         // colors
         0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,  // top right - red
         0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,  // bottom right - green
        -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,  // bottom left - blue
        -0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 1.0f   // top left - purple
    };

    unsigned int indices[] = {
        0, 1, 3,   // first triangle
        1, 2, 3    // second triangle
    };
```

2. **Define six indices**

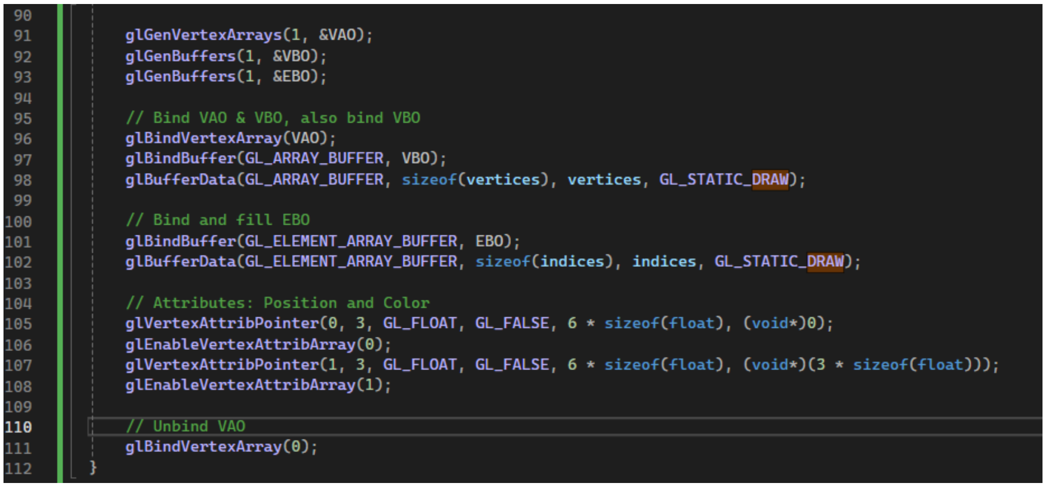
From the code above, we create two triangles from 6 indices. Then declare the buffer objects and the shader program, and bind them.

3. **Bind EBO with rectangle's indices**



```
GLuint shaderProgram;
GLuint VAO;
GLuint VBO;
GLuint EBO;
```

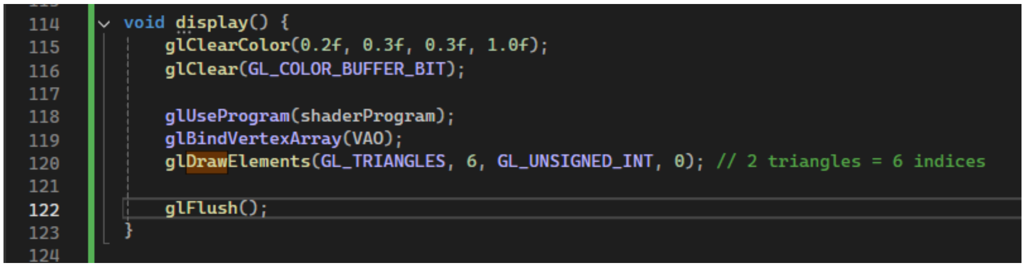
The process really just the same with the project 3, only added EBO. The EBO object that was just created then is bond to the target array buffer (element array buffer), which is buffer type that stores indices, to specify how those vertices are connected to make primitives shape (in this case, triangles). The declaration of those frame objects and the binding processes are declare din the *initBuffer()* function. The static draw is used to optimize the static graphic.



```

    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);

    // Bind VAO & VBO, also bind VBO
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    // Bind and fill EBO
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

    // Attributes: 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);

    // Unbind VAO
    glBindVertexArray(0);
}
```

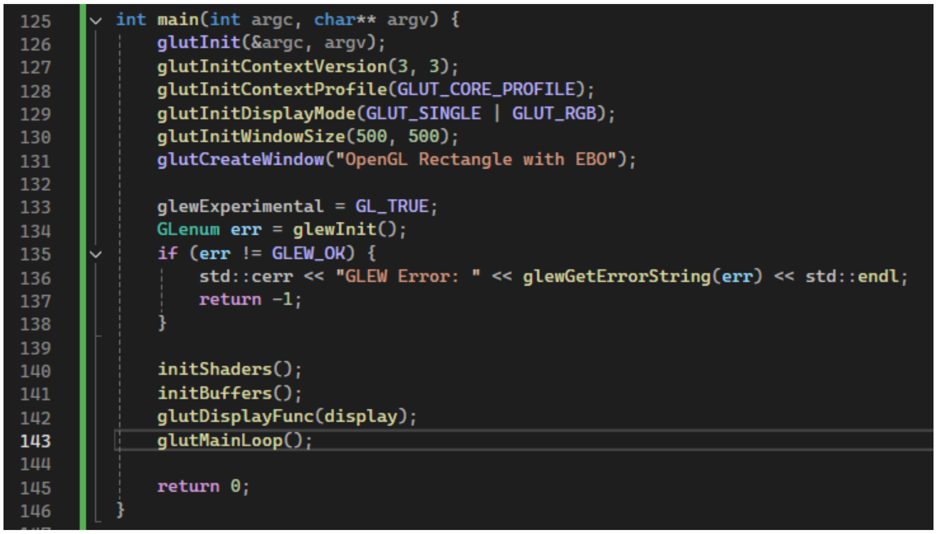
4. **Draw using glDrawElements()**

The drawing function calling occurs in the display which will be called in glut's display function. The line 120th, we draw the rectangles out of triangles using *glDrawElements()*. Here I use indices stored on EBO instead of specifying all vertex directly, to reference vertices in the VBO. No redundant work. Then just command OpenGL to run the commands using glFLush().



```
114  void display() {
115      glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
116      glClear(GL_COLOR_BUFFER_BIT);
117
118      glUseProgram(shaderProgram);
119      glBindVertexArray(VAO);
120      glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0); // 2 triangles = 6 indices
121
122      glFlush();
123  }
124
```

Then just call display on the main function similarly like we call shader initialization and buffer initialization.



```
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("OpenGL Rectangle with EBO");

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

    initShaders();
    initBuffers();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

Here are the shaders initialization including the declarations.

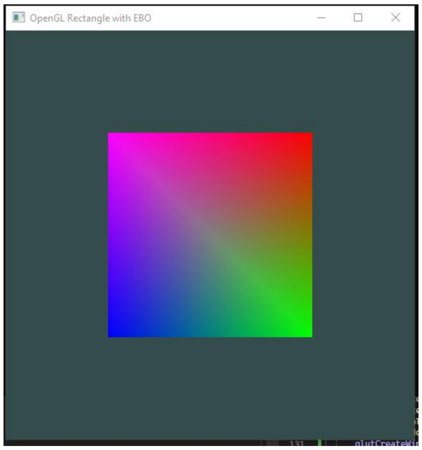
// Shader program declaration

```
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);
}
)";
```

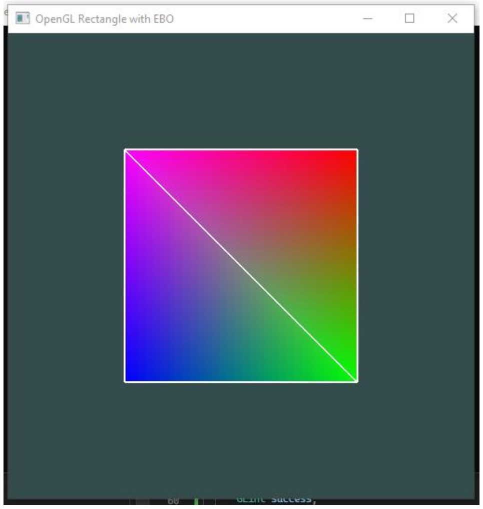
// Shader program initialization

```
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 << "Vertex Shader Compilation Failed\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 << "Fragment Shader Compilation Failed\n" << infoLog << std::endl;
    }

    // Shader 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 << "Shader Program Linking Failed\n" << infoLog << std::endl;
    }

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

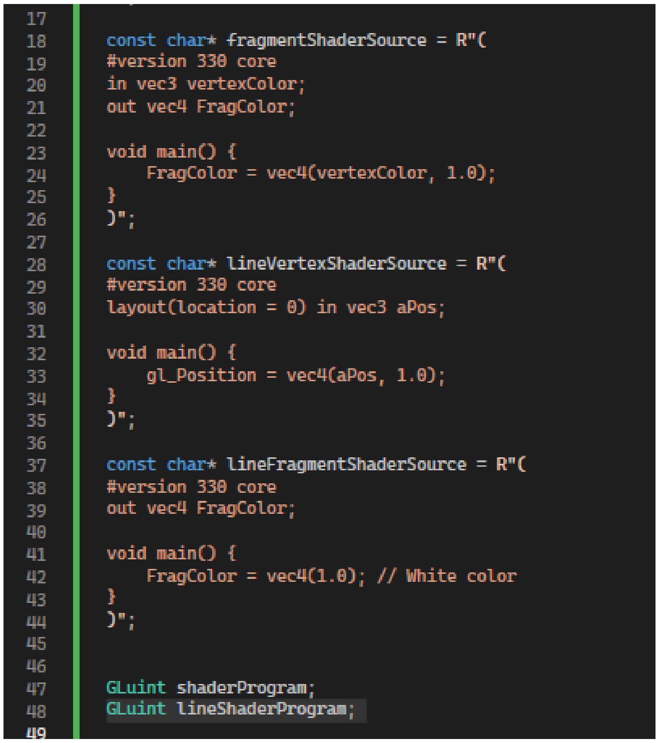
The result:



And I tried to create the outline to show the rectangles.

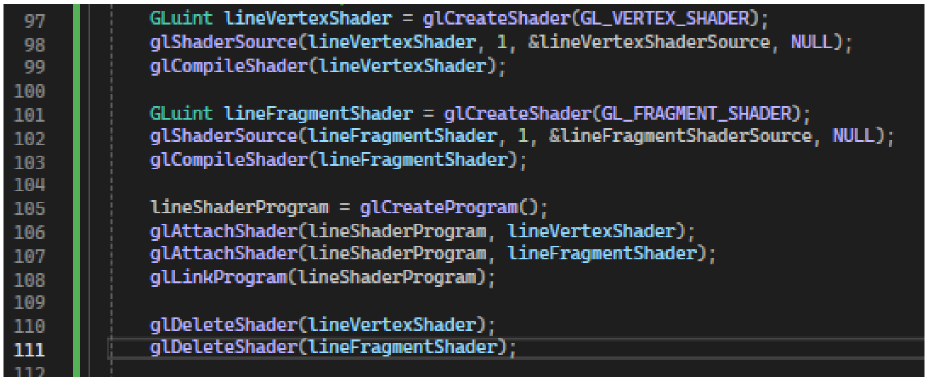


Create the vertex shader and fragment shader for the line. In shader code, we must provide the version of the GLSL being used, it corresponds to the OpenGL version. It takes a 3D position (aPos) as input, and the location = 0 means that this input is bound to attribute location 0 in the vertex data. In the main function, it converts the position into a 4D vector by adding w component with 1.0 (for the homogeneous coordinates for transformations). Also declare the variable for the line shader.



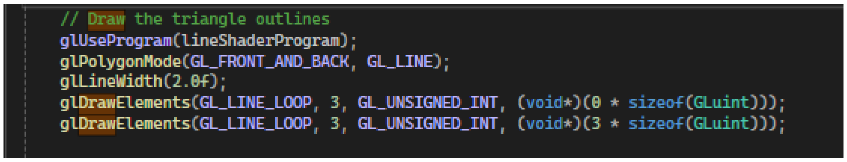
```
const char* fragmentShaderSource = R"(
#version 330 core
in vec3 vertexColor;
out vec4 FragColor;

void main() {
    FragColor = vec4(vertexColor, 1.0);
}
)";

const char* lineVertexShaderSource = R"(
#version 330 core
layout(location = 0) in vec3 aPos;

void main() {
    gl_Position = vec4(aPos, 1.0);
}
)";

const char* lineFragmentShaderSource = R"(
#version 330 core
out vec4 FragColor;

void main() {
    FragColor = vec4(1.0); // White color
}
)";


GLuint shaderProgram;
GLuint lineShaderProgram;
```

Update the *initShader()* function, add the binding of the newly line shader. Then delete it as a standalone shader because it is not needed.



```
    GLuint lineVertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(lineVertexShader, 1, &lineVertexShaderSource, NULL);
    glCompileShader(lineVertexShader);

    GLuint lineFragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(lineFragmentShader, 1, &lineFragmentShaderSource, NULL);
    glCompileShader(lineFragmentShader);

    lineShaderProgram = glCreateProgram();
    glAttachShader(lineShaderProgram, lineVertexShader);
    glAttachShader(lineShaderProgram, lineFragmentShader);
    glLinkProgram(lineShaderProgram);

    glDeleteShader(lineVertexShader);
    glDeleteShader(lineFragmentShader);
```

And the last step is just updating the *display()* function to draw the outline using the shader.



```
    // Draw the triangle outlines
    glUseProgram(lineShaderProgram);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2.0f);
    glDrawElements(GL_LINE_LOOP, 3, GL_UNSIGNED_INT, (void*)(0 * sizeof(GLuint)));
    glDrawElements(GL_LINE_LOOP, 3, GL_UNSIGNED_INT, (void*)(3 * sizeof(GLuint)));
```

Source code:

```
#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);
}
)";

const char* lineVertexShaderSource = R"(
#version 330 core
layout(location = 0) in vec3 aPos;

void main() {
    gl_Position = vec4(aPos, 1.0);
}
)";

const char* lineFragmentShaderSource = R"(
#version 330 core
out vec4 FragColor;

void main() {
    FragColor = vec4(1.0); // White color
}
)";


GLuint shaderProgram;
GLuint lineShaderProgram;

GLuint VAO;
GLuint VBO;
GLuint EBO;

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 << "Vertex Shader Compilation Failed\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 << "Fragment Shader Compilation Failed\n" << infoLog << std::endl;
    }

    // Shader 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 << "Shader Program Linking Failed\n" << infoLog << std::endl;
    }

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    // Line shader compilation
    GLuint lineVertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(lineVertexShader, 1, &lineVertexShaderSource, NULL);
    glCompileShader(lineVertexShader);

    GLuint lineFragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(lineFragmentShader, 1, &lineFragmentShaderSource, NULL);
    glCompileShader(lineFragmentShader);

    lineShaderProgram = glCreateProgram();
    glAttachShader(lineShaderProgram, lineVertexShader);
    glAttachShader(lineShaderProgram, lineFragmentShader);
    glLinkProgram(lineShaderProgram);

    glDeleteShader(lineVertexShader);
    glDeleteShader(lineFragmentShader);

}

void initBuffers() {
    // Rectangle with 4 vertices (x, y, z, r, g, b)
    float vertices[] = {
        // positions         // colors
        0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,  // top right - red
        0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,  // bottom right - green
       -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,  // bottom left - blue
       -0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 1.0f   // top left - purple
    };

    unsigned int indices[] = {
        0, 1, 3,   // first triangle
        1, 2, 3    // second triangle
    };

    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);

    // Bind VAO & VBO, also bind VBO
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    // Bind and fill EBO
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

        // Attributes: 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);

    // Unbind VAO
    glBindVertexArray(0);
}

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

    // Draw the filled rectangle
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);

    // Draw the triangle outlines
    glUseProgram(lineShaderProgram);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2.0f);
    glDrawElements(GL_LINE_LOOP, 3, GL_UNSIGNED_INT, (void*)(0 * sizeof(GLuint)));
    glDrawElements(GL_LINE_LOOP, 3, GL_UNSIGNED_INT, (void*)(3 * sizeof(GLuint)));

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    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("OpenGL Rectangle with EBO");

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

    initShaders();
    initBuffers();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```