OpenGL Triangle Program

1. Initialize OpenGL and Create a Window

1. **Set up a windowing library (e.g., GLFW/GLUT)**
2. **Code snippet for window creation:**

```
#include <GLFW/glfw3.h>

// Initialize GLFW
if (!glfwInit()) {
    std::cerr << "Failed to initialize GLFW" << std::endl;
    return -1;
}

// Create a window
GLFWwindow* window = glfwCreateWindow(800, 600, "Triangle Program",
nullptr, nullptr);
if (!window) {
    std::cerr << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
```

Debugging Tip: Check for errors after each GLFW call using:

```
if (!glfwInit()) {
    std::cerr << "GLFW Initialization Error!" << std::endl;
}
```

---

2. Define Vertex Data

- Vertices of the triangle are in **Normalized Device Coordinates (NDC)**.

```
float vertices[] = {
    -0.5f, -0.5f, 0.0f, // Bottom-left
     0.5f, -0.5f, 0.0f, // Bottom-right
     0.0f,  0.5f, 0.0f  // Top-center
};
```

---

3. Generate and Bind Vertex Buffer Object (VBO)

- Store vertex data in GPU memory.

```
unsigned int VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Notes:

- `GL_STATIC_DRAW` is used because the data doesn't change often.
- Use `glGetError()` after OpenGL calls to detect errors.

---

4. Create and Compile Shaders

Vertex Shader

```
const char* vertexShaderSource = R"(
#version 330 core
layout (location = 0) in vec3 aPos;
void main() {
    gl_Position = vec4(aPos, 1.0);
}
)";
unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, nullptr);
glCompileShader(vertexShader);

// Check for compilation errors
int success;
char infoLog[512];
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
if (!success) {
    glGetShaderInfoLog(vertexShader, 512, nullptr, infoLog);
    std::cerr << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog
<< std::endl;
}
```

Fragment Shader

```
const char* fragmentShaderSource = R"(
#version 330 core
out vec4 FragColor;
void main() {
    FragColor = vec4(1.0, 0.5, 0.2, 1.0); // Orange color
}
)";
unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
```

```
glShaderSource(fragmentShader, 1, &fragmentShaderSource, nullptr);
glCompileShader(fragmentShader);

// Check for errors as above
```

---

5. Link Shaders into a Shader Program

```
unsigned int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

// Check for linking errors
glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
if (!success) {
    glGetProgramInfoLog(shaderProgram, 512, nullptr, infoLog);
    std::cerr << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog <<
std::endl;
}
glUseProgram(shaderProgram);

// Delete shaders after linking
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

---

6. Create and Configure Vertex Array Object (VAO)

- VAOs store vertex attributes and buffer bindings.

```
unsigned int VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

// Bind the VBO to the VAO
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
glEnableVertexAttribArray(0);
```

Note: Ensure the VAO is bound before configuring vertex attributes.

---

7. Render Loop

```
while (!glfwWindowShouldClose(window)) {
    // Clear screen
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw the triangle
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 3);

    // Swap buffers and poll events
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

---

8. Cleanup Resources

```
glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
glDeleteProgram(shaderProgram);
glfwTerminate();
```

---

9. Debugging Tools

1. **OpenGL Error Checking:**

   ```
   GLenum err;
   while ((err = glGetError()) != GL_NO_ERROR) {
       std::cerr << "OpenGL Error: " << err << std::endl;
   }
   ```