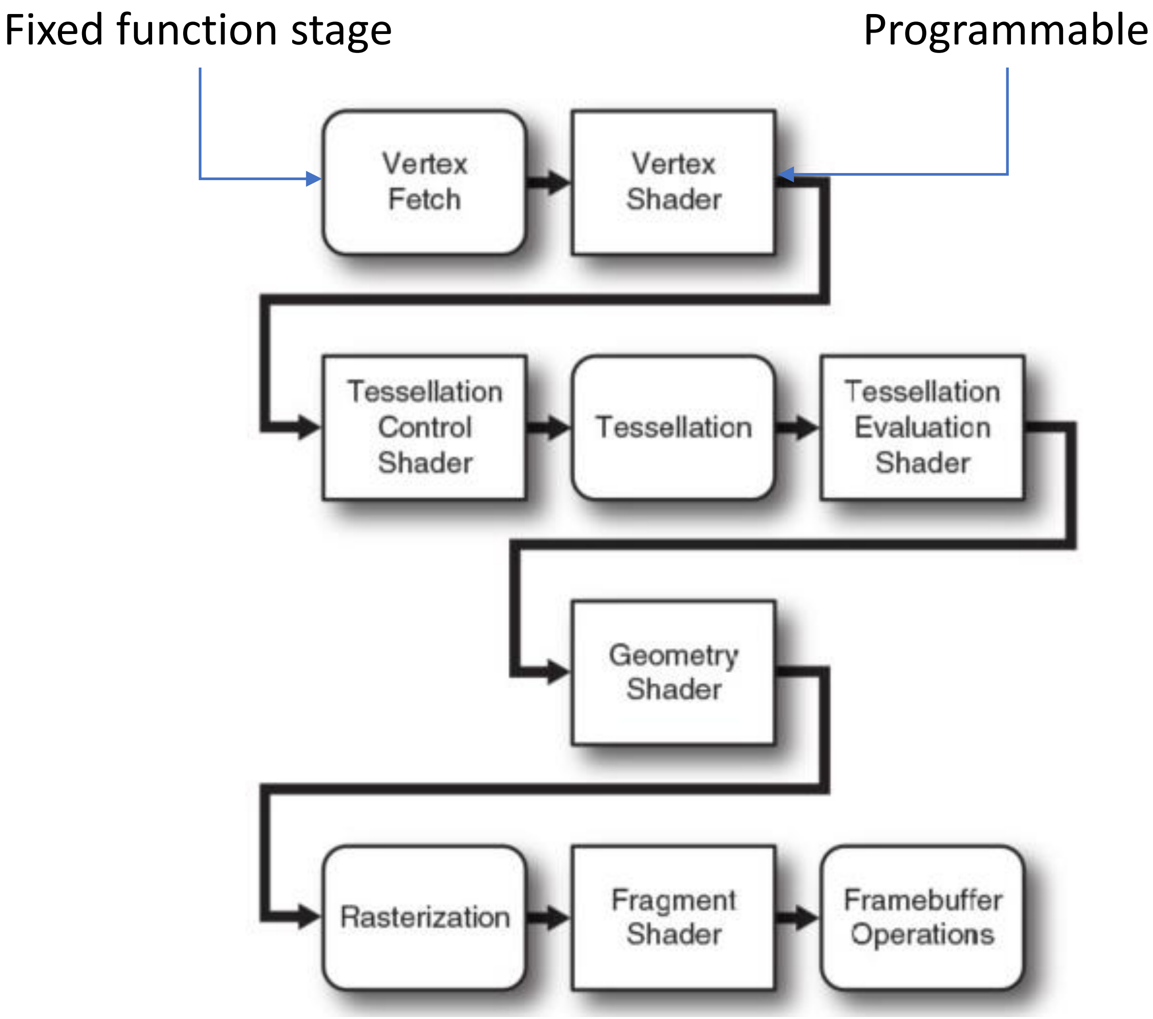
Introduction to Basic OpenGL Pipeline

Dr. Mekides Assefa Abebe and Dr. Fereshteh Mirjalili

Graphics Pipeline

- Buffers and textures
  - Structures designed to store information for rendering
- Front end, primitive assembly: process vertices and pass them to the rasterizer
  - Vertex Shader
  - Tessellation Shader
  - Geometry Shader
- Back end, rasterizes the vector representation of primitives to a pixel representation.
  - Fragment Shader
  - Blending
  - Updating



Fixed function stage

Programmable

Vertex Fetch

Vertex Shader

Tessellation Control Shader

Tessellation

Tessellation Evaluation Shader

Geometry Shader

Rasterization

Fragment Shader

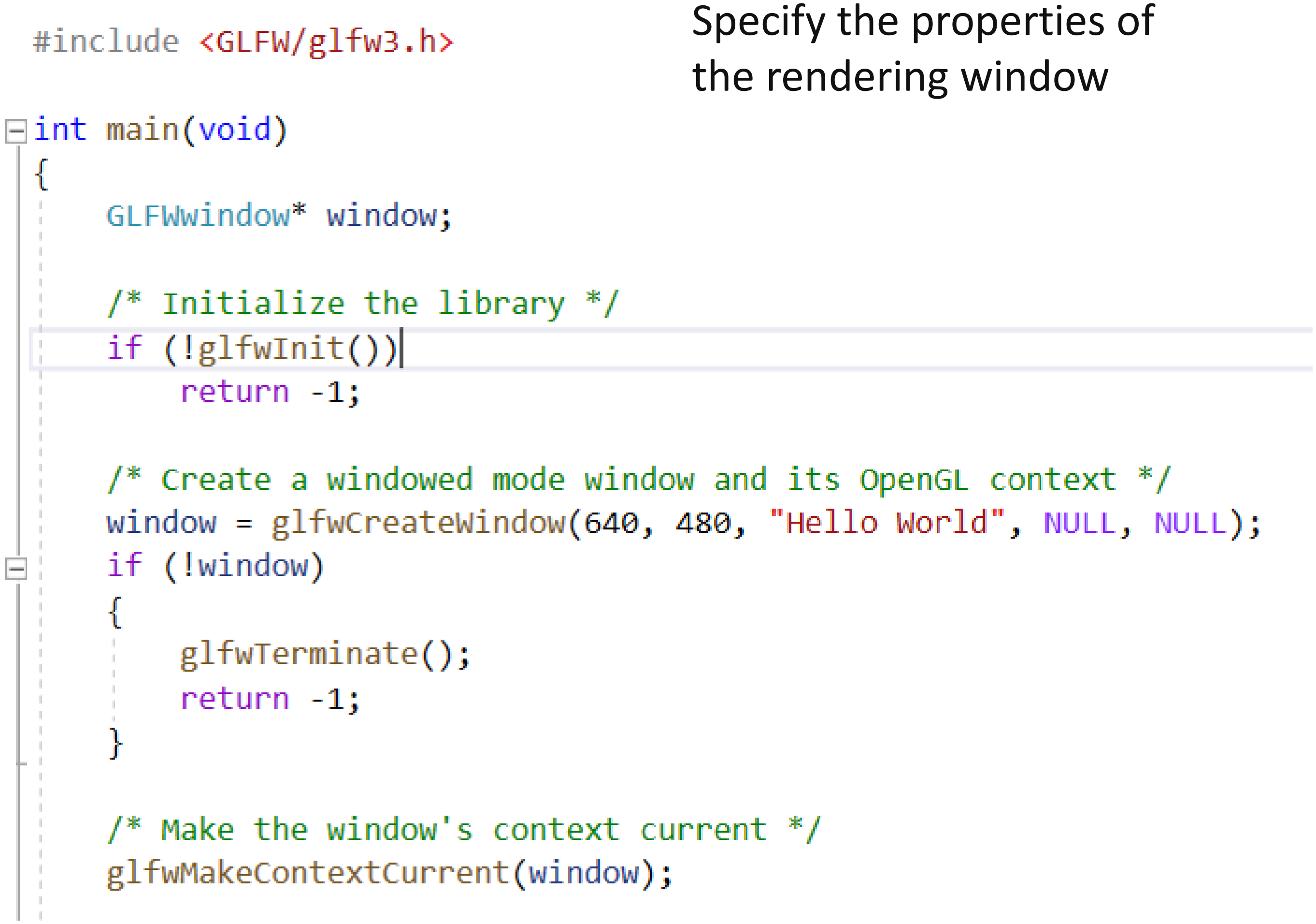
Framebuffer Operations

Graham et al. OpenGL super bible

Libraries

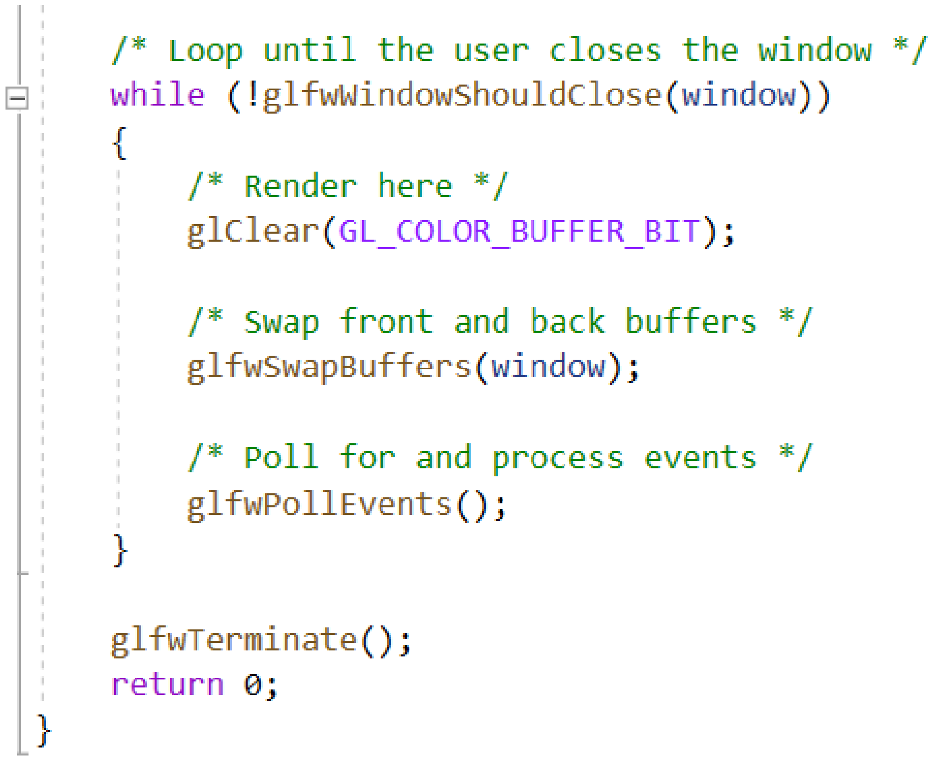
- GLFW: platform independent API for creating windows, context, reading inputs and handling events

Specify the properties of the rendering window



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

int main(void)
{
    GLFWwindow* window;

    /* Initialize the library */
    if (!glfwInit())
        return -1;

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(640, 480, "Hello World", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }

    /* Make the window's context current */
    glfwMakeContextCurrent(window);
```

Initiate the event loop

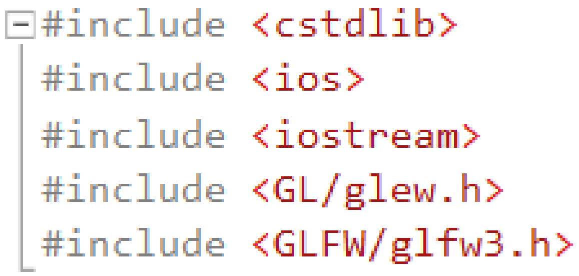


```
    /* Loop until the user closes the window */
    while (!glfwWindowShouldClose(window))
    {
        /* Render here */
        glClear(GL_COLOR_BUFFER_BIT);

        /* Swap front and back buffers */
        glfwSwapBuffers(window);

        /* Poll for and process events */
        glfwPollEvents();
    }

    glfwTerminate();
    return 0;
}
```

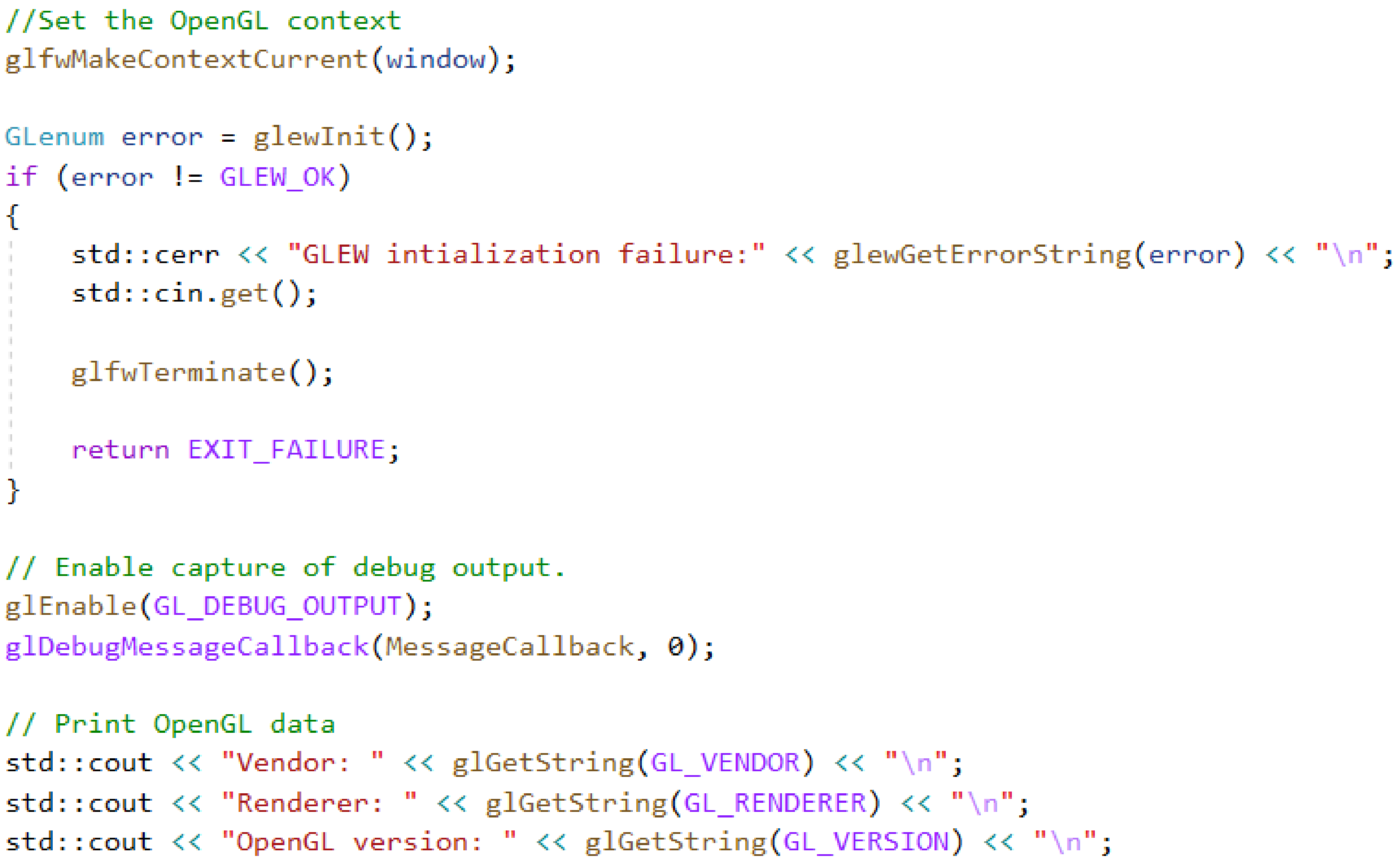
Documentation: https://www.glfw.org/docs/latest/

Libraries

- GLEW: extension loader library
  - Used to declare and load the OpenGL extensions and modern core APIs.
  - Included before GLFW to suppress the legacy OpenGL loading by glfw.
  - Must be initialized after the OpenGL current context is created.
  - After glew initialization: all available OpenGL core and extensions will be available for rendering.



```
#include <cstdlib>
#include <ios>
#include <iostream>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
```

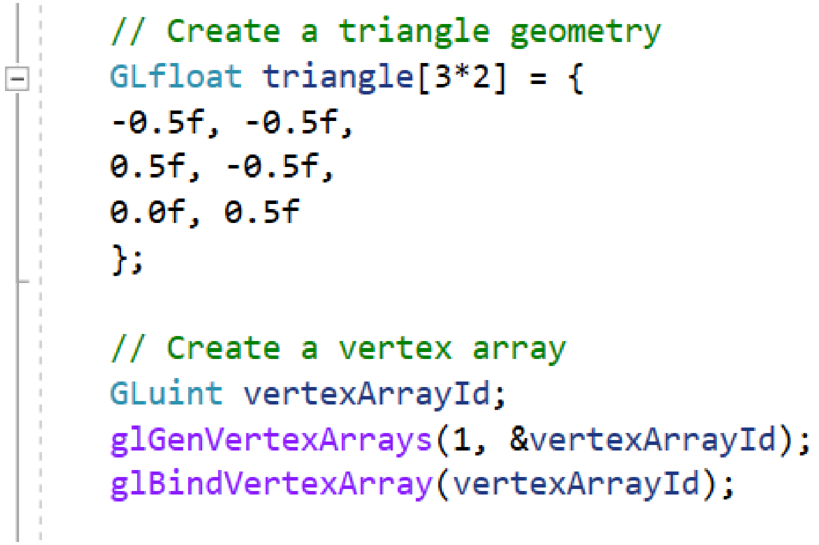


```
//Set the OpenGL context
glfwMakeContextCurrent(window);

GLenum error = glewInit();
if (error != GLEW_OK)
{
    std::cerr << "GLEW intialization failure:" << glewGetErrorString(error) << "\n";
    std::cin.get();

    glfwTerminate();

    return EXIT_FAILURE;
}

// Enable capture of debug output.
glEnable(GL_DEBUG_OUTPUT);
glDebugMessageCallback(MessageCallback, 0);

// Print OpenGL data
std::cout << "Vendor: " << glGetString(GL_VENDOR) << "\n";
std::cout << "Renderer: " << glGetString(GL_RENDERER) << "\n";
std::cout << "OpenGL version: " << glGetString(GL_VERSION) << "\n";
```

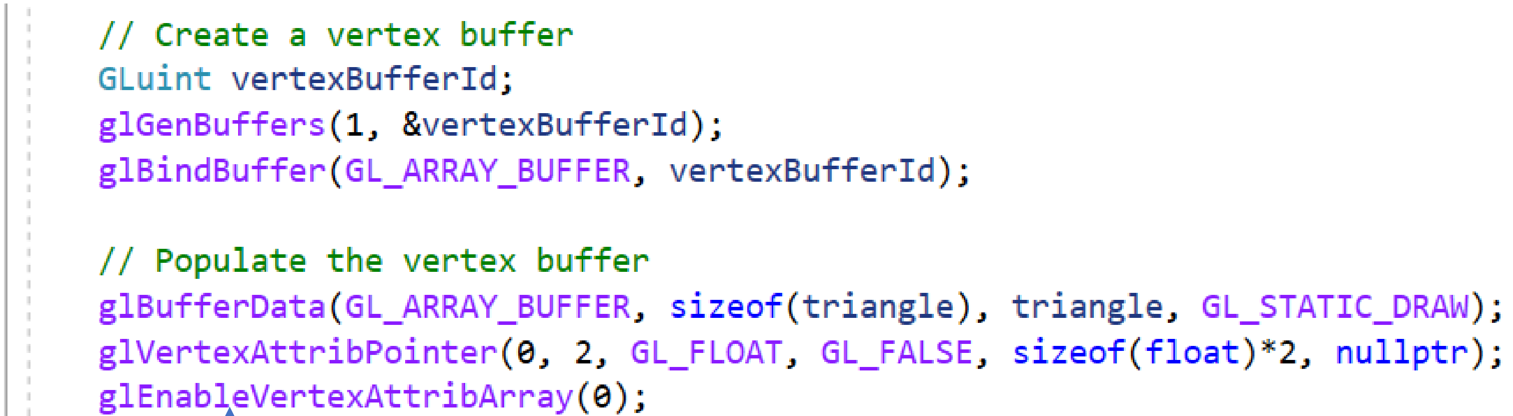
Vertex Buffer Objects (VBO) and Vertex Array Objects (VAO)

- VAO: vertex fetch stage.
  - supply input for the vertex shader

- VBO:
  - upload vertex data to the GPU



```
// Create a triangle geometry
GLfloat triangle[3*2] = {
-0.5f, -0.5f,
0.5f, -0.5f,
0.0f, 0.5f
};

// Create a vertex array
GLuint vertexArrayId;
glGenVertexArrays(1, &vertexArrayId);
glBindVertexArray(vertexArrayId);
```

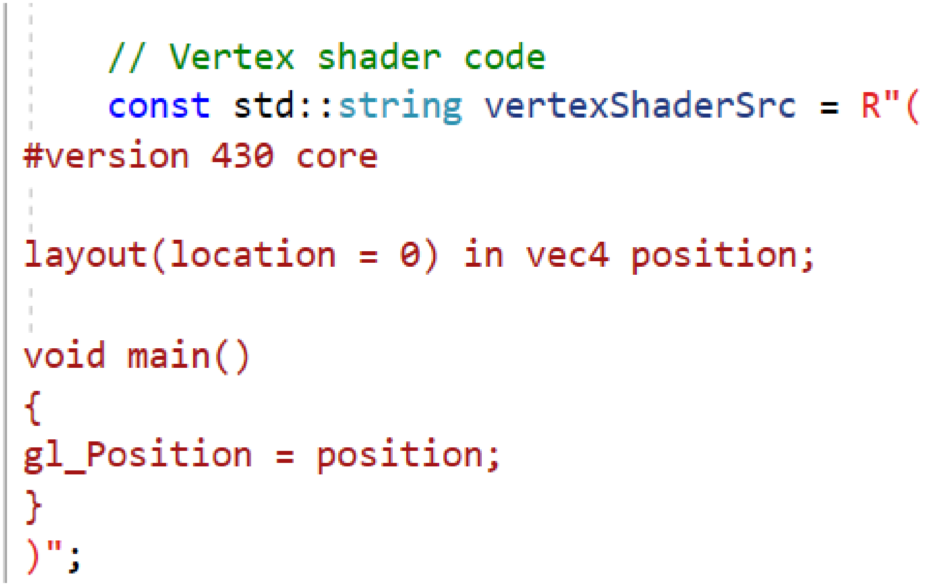


```
// Create a vertex buffer
GLuint vertexBufferId;
glGenBuffers(1, &vertexBufferId);
glBindBuffer(GL_ARRAY_BUFFER, vertexBufferId);

// Populate the vertex buffer
glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, sizeof(float)*2, nullptr);
glEnableVertexAttribArray(0);
```

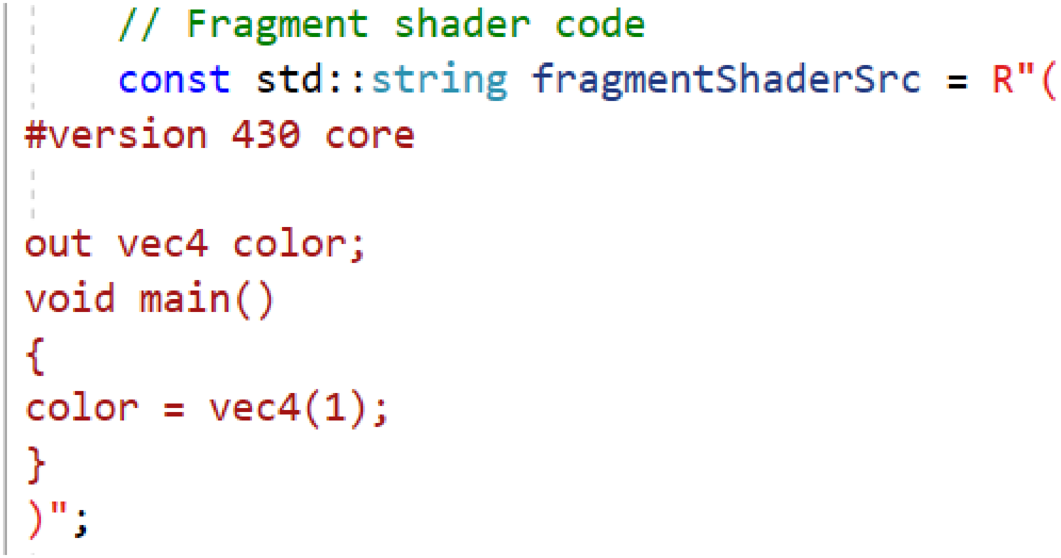
How the data for the vertex shader input is retrieved from the array?

Shaders

- Interpret the data passed through the vertex buffers
- At least vertex and fragment shaders are required to draw something on the screen
- Written in GLSL – OpenGL shading language



```
    // Vertex shader code
    const std::string vertexShaderSrc = R"(
#version 430 core

layout(location = 0) in vec4 position;

void main()
{
gl_Position = position;
}
)";
```



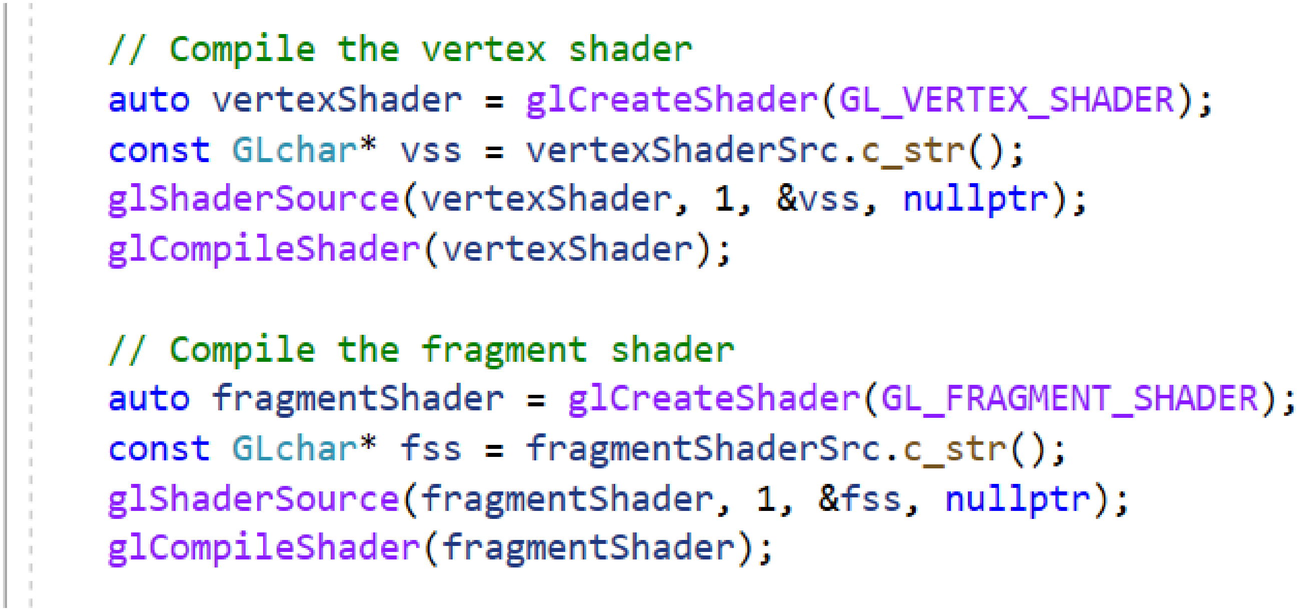
```
    // Fragment shader code
    const std::string fragmentShaderSrc = R"(
#version 430 core

out vec4 color;
void main()
{
color = vec4(1);
}
)";
```

Vertex Shader:
Output the final vertex position in device coordinates and
Output any data the fragment shader requires

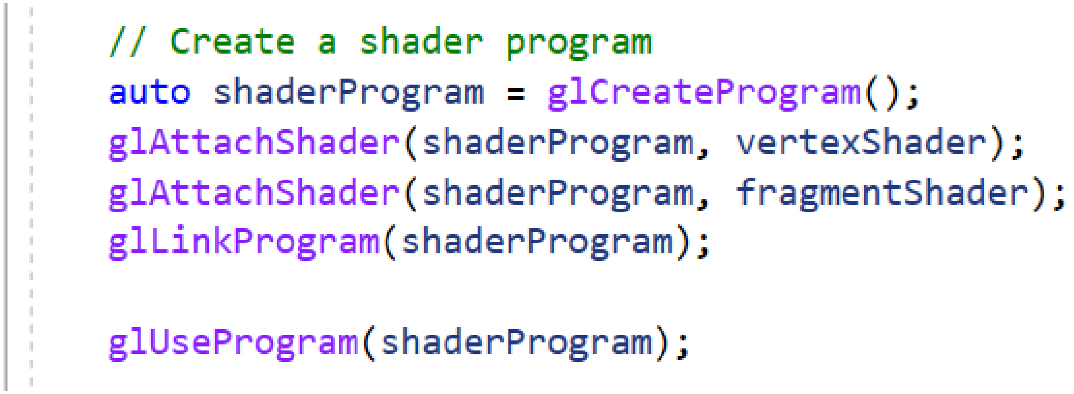
Fragment Shader:
Depends on attributes passed to output without any calculation
Output final color of fragment pixels

Shaders

- Compiling Shaders into code which can be executed by the graphics card
- Program: combines shader objects.
  - Linking creates the connection between vertex and fragment shaders

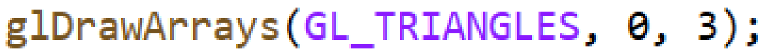


```
// Compile the vertex shader
auto vertexShader = glCreateShader(GL_VERTEX_SHADER);
const GLchar* vss = vertexShaderSrc.c_str();
glShaderSource(vertexShader, 1, &vss, nullptr);
glCompileShader(vertexShader);

// Compile the fragment shader
auto fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
const GLchar* fss = fragmentShaderSrc.c_str();
glShaderSource(fragmentShader, 1, &fss, nullptr);
glCompileShader(fragmentShader);
```



```
// Create a shader program
auto shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

glUseProgram(shaderProgram);
```

Draw primitives in the event loop.



```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Exercise 1

- Use the code from previous lab and modify it to
  - Change the background of the window to green: when the keyboard key ‘g’ is pressed.
  - Change the background back to red: when the ‘R’ key is pressed.
- Commit your final code to your gitlab with in a folder named Lab2_opengl

References

- GLFW documentation: https://www.glfw.org/docs/latest/
- OpenGL functions: https://docs.gl/
- OpenGL super bible book: http://www.openglsuperbible.com/
- Other tutorials: Learn OpenGL https://learnopengl.com/
- YouTube tutorial:
    - the Cherno: https://www.youtube.com/watch?v=W3gAzLwfIP0&list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2&ab_channel=TheCherno