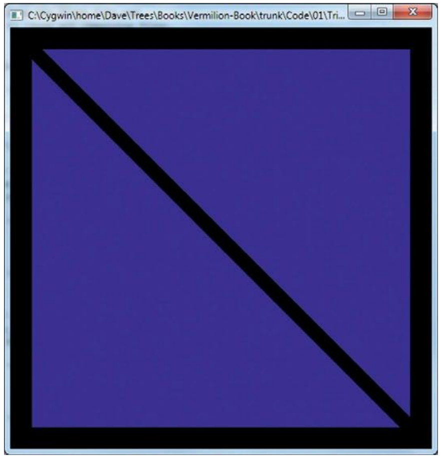
Computer Graphics
- First OpenGL Program

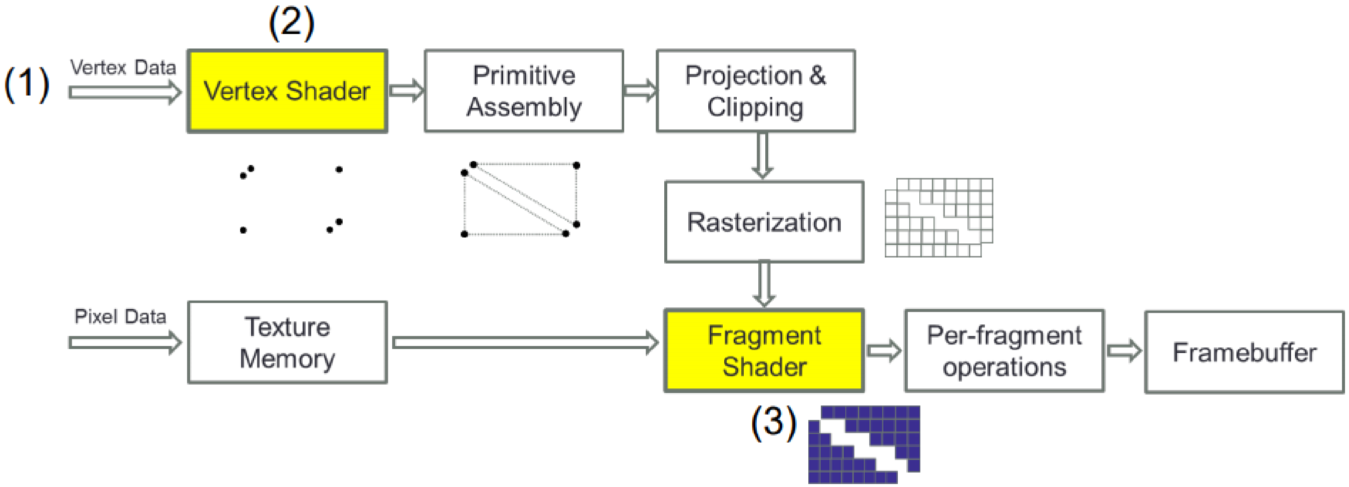
Sung Soo Hwang

First OpenGL Program

- Rendering two triangles

- You will learn:
  1. How to send your vertex data to OpenGL
  2. How to write a pass-through vertex shader
  3. How to write a simple fragment shader for coloring


C:\Cygwin\home\Dave\Trees\Books\Vermilion-Book\trunk\Code\01\Tri...



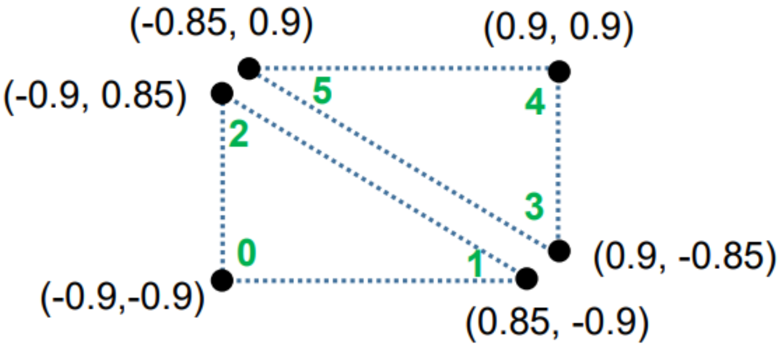
Sending Vertex Data to OpenGL

- Buffer allocation & initialization
  - OpenGL requires that all data be stored in buffer objects managed by the OpenGL server.
  - glBufferData() is most commonly used to allocate new memory space for these objects.

```
GLuint Buffers[1], VertexArrays[1];
glGenBuffers(1, Buffers);
glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

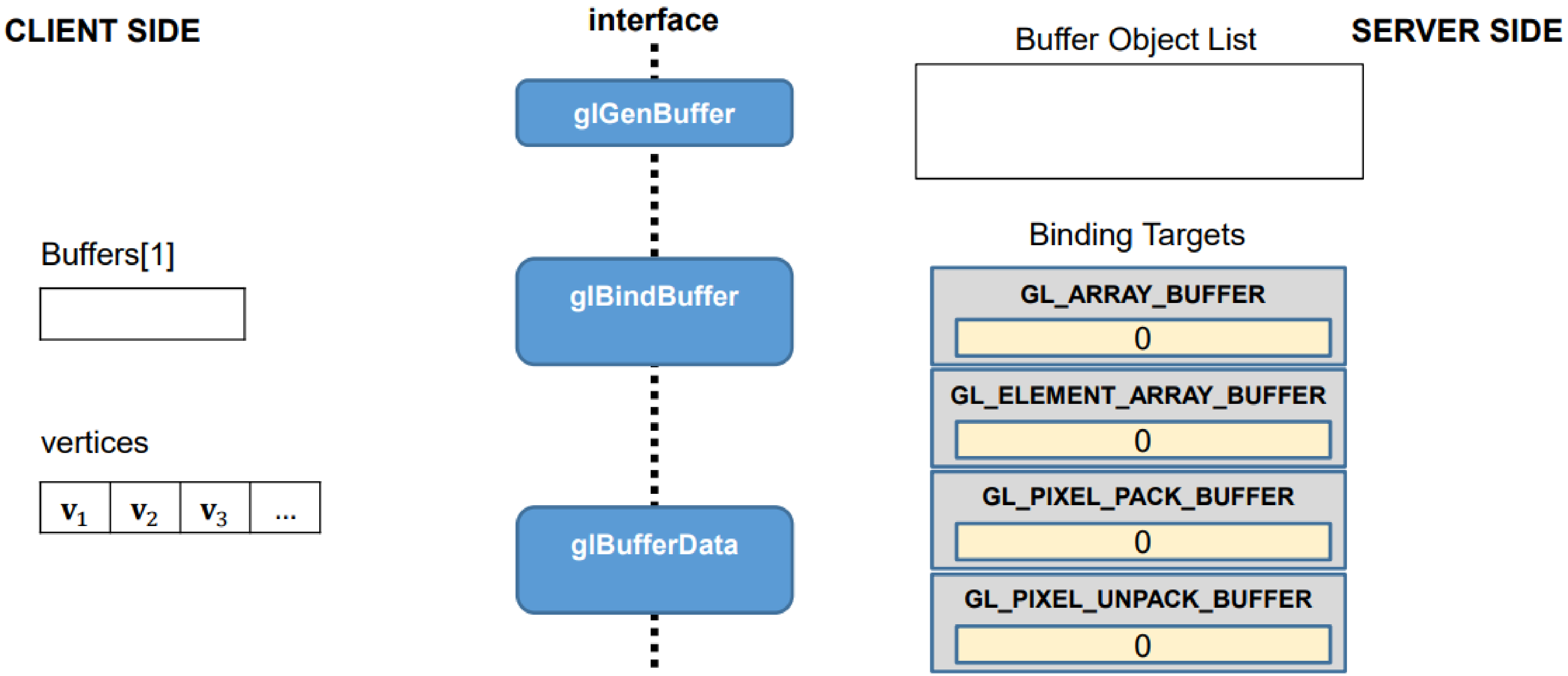
* Example

```
const GLsizei NumVertices = 6;
GLfloat vertices[NumVertices][2] = {
            {-0.90f, -0.90f}, {0.85f, -0.90f},
            {-0.90f, 0.85f}, {0.90f, -0.85f},
            {0.90f, 0.90f}, {-0.85f, 0.90f} };
```



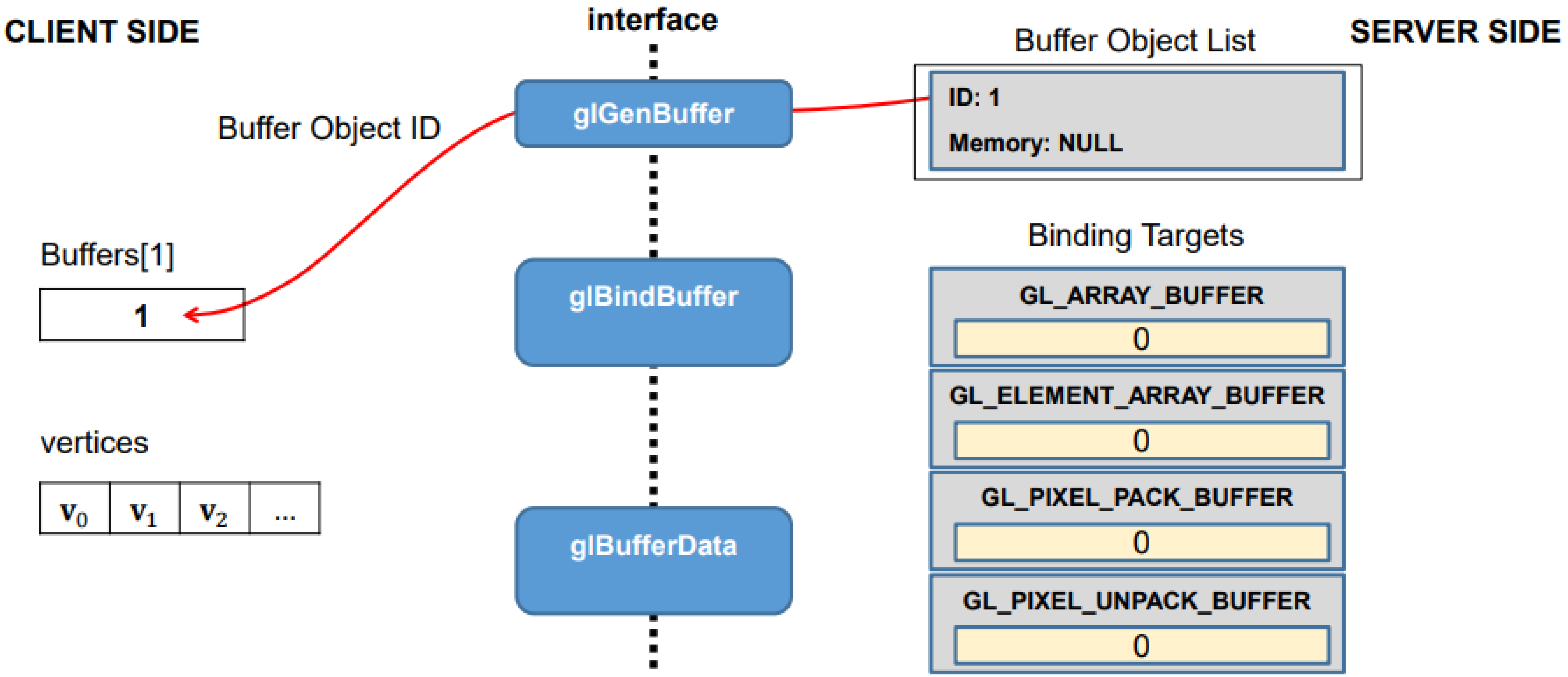
Sending Vertex Data to OpenGL

```
⇨ GLuint Buffers[1], VertexArrays[1];
  glGenBuffers(1, Buffers);
  glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```



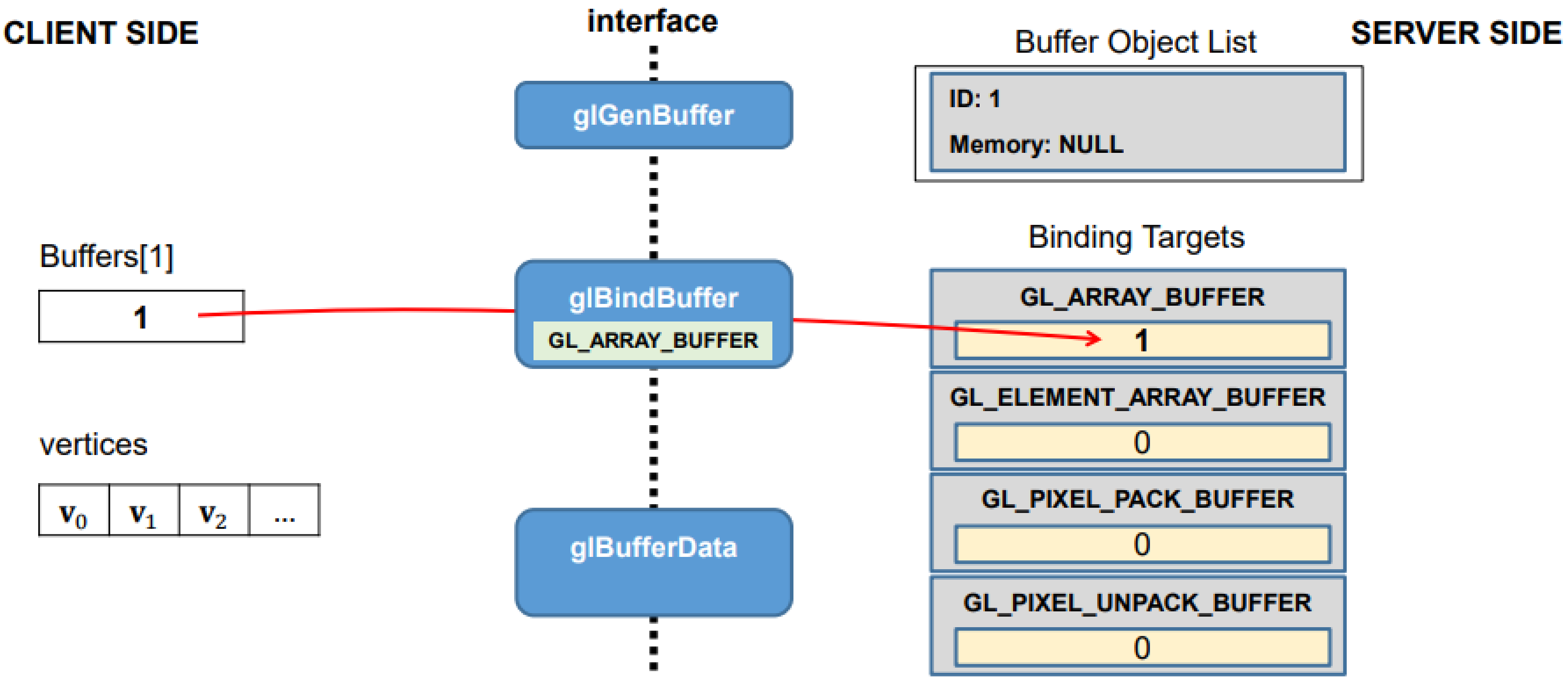
Sending Vertex Data to OpenGL

```
GLuint Buffers[1], VertexArrays[1];
glGenBuffers(1, Buffers);
glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```



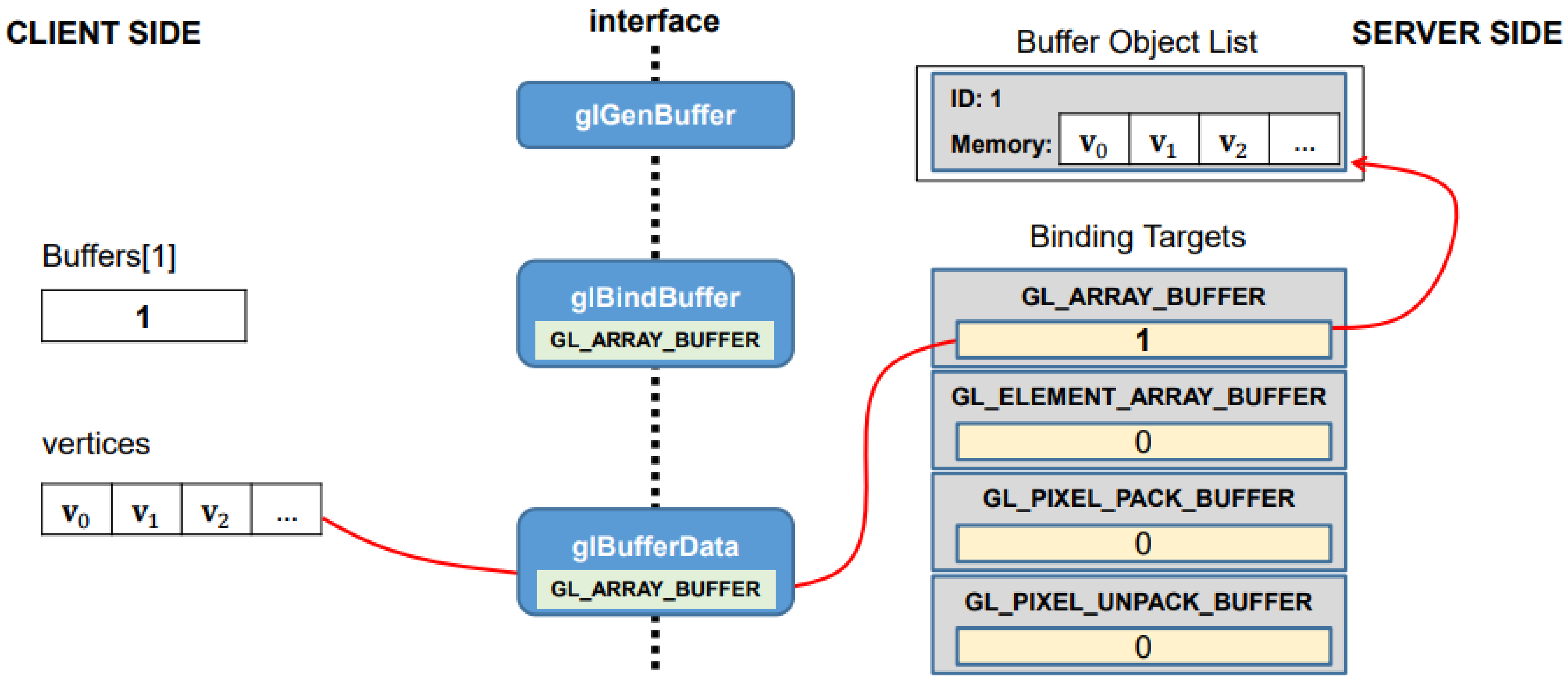
Sending Vertex Data to OpenGL

```
GLuint Buffers[1], VertexArrays[1];
glGenBuffers(1, Buffers);
⇒ glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```



Sending Vertex Data to OpenGL

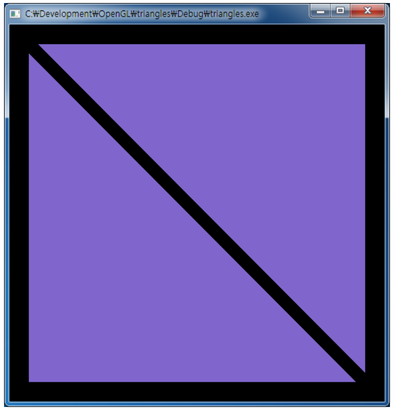
```
GLuint Buffers[1], VertexArrays[1];
glGenBuffers(1, Buffers);
glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
⇨ glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```



Sending Vertex Data to OpenGL

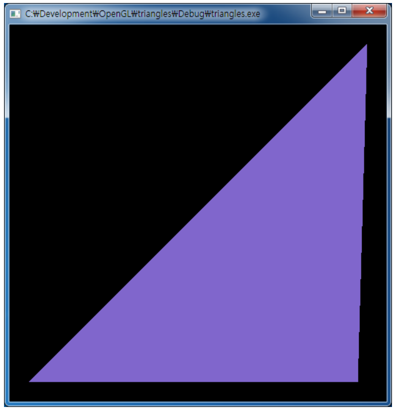
- Request for rendering
  - Once the buffers have been initialized, a request for rendering can be issued by calling one of OpenGL's drawing commands, such as glDrawArrays() or glDrawElements().

```
glDrawArrays(GL_TRIANGLES, 0, 6);
```



Construct and render a sequence of geometric primitives by accessing the buffer data elements in a sequential or an indexed order

```
GLuint indices[3] = { 0, 1, 4 };
glDrawElements(GL_TRIANGLES, 3,
               GL_UNSIGNED_INT, indices);
```



Pass-through Vertex Shader

- Vertex shader
  - Executed to process the data associated with each vertex issued by a drawing command
  - The pass-through vertex shader is one of the simplest vertex shaders that just copies data to pass it through.

```
#version 430                  // GLSL version to use(430 -> ver 4.3)

in vec4 vPosition;            // Declaration of a shader's input attribute(read-only)

void main()                   // Main function
{
        gl_Position = vPosition;
}
```

File name: triangles.vert

vec4 : 4D-vector data type
gl_Position : Built-in 4D vector representing the final processed vertex position

Fragment Shader for Coloring

- Fragment shader
  - Operates on every fragment which is produced by rasterization.

```
#version 430
out vec4 FragColor;
void main()
{
        FragColor = vec4(0.5, 0.4, 0.8, 1.0);
}
```

File name: triangles.frag

Full source code: triangle.cpp

```
#include <stdio.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "LoadShaders.h"
```

Built-in header file which defines utility functions for loading shaders

```
void init();
void display();
```

Initializes OpenGL for drawing triangles

Draws triangles with OpenGL

```
void main(int argc, char** argv)
{
		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_RGBA);
		glutInitWindowSize(512, 512);
		glutCreateWindow(argv[0]);
```

Initialize GLUT to make a window (for more details: https://www.opengl.org/resources/libraries/glut/spec3/node113.html)

```
		GLenum err = glewInit();
		if (err != GLEW_OK) {
				fprintf(stderr, "Error: %s₩n", glewGetErrorString(err));
				exit(EXIT_FAILURE);
		}
```

Initialize GLEW to load OpenGL extensions

```
		init();
		glutDisplayFunc(display);
		glutMainLoop();
}
```

Register a “display” callback function and enter the GLUT event processing loop

Full source code: triangle.cpp

```
#include <stdio.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "LoadShaders.h"

void init();
void display();

void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA);
        glutInitWindowSize(512, 512);
        glutCreateWindow(argv[0]);
        GLenum err = glewInit();
        if (err != GLEW_OK) {
                fprintf(stderr, "Error: %s₩n", glewGetErrorString(err));
                exit(EXIT_FAILURE);
        }
        init();
        glutDisplayFunc(display);
        glutMainLoop();
}
```

Here must be some suitable variation declarations. For example,

```
const GLsizei NumVertices = 6;
GLfloat vertices[NumVertices][2] = {
            {-0.90f, -0.90f}, {0.85f, -0.90f},
            {-0.90f, 0.85f}, {0.90f, -0.85f},
            {0.90f, 0.90f}, {-0.85f, 0.90f} };

GLuint Buffers[1], VertexArrays[1];


...
```

Full source code: triangle.cpp

```
void init()
{
        glGenVertexArrays(1, VertexArrays);
        glBindVertexArray(VertexArrays[0]);

        glGenBuffers(1, Buffers);
        glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

        ShaderInfo shaders[] = {
                {GL_VERTEX_SHADER, "triangles.vert"},
                {GL_FRAGMENT_SHADER, "triangles.frag"},
                {GL_NONE, NULL}
        };

        GLuint program = LoadShaders(shaders);
        glUseProgram(program);

        GLint location = glGetAttribLocation(program, "vPosition");
        glVertexAttribPointer(location, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(location);
}
```

Generate vertex array objects(VAOs) and specify the current active VAO.

* Vertex Array Object(VAO):
An object which contains one or more Vertex Buffer Objects(VBOs)

Full source code: triangle.cpp

```
void init()
{
        glGenVertexArrays(1, VertexArrays);
        glBindVertexArray(VertexArrays[0]);

        glGenBuffers(1, Buffers);
        glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

        ShaderInfo shaders[] = {
                {GL_VERTEX_SHADER, "triangles.vert"},
                {GL_FRAGMENT_SHADER, "triangles.frag"},
                {GL_NONE, NULL}
        };

        GLuint program = LoadShaders(shaders);
        glUseProgram(program);

        GLint location = glGetAttribLocation(program, "vPosition");
        glVertexAttribPointer(location, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(location);
}
```

Compile shaders and produce a program to which the compiled shaders are attached. Then, register the program in OpenGL

Full source code: triangle.cpp

```
void init()
{
        glGenVertexArrays(1, VertexArrays);
        glBindVertexArray(VertexArrays[0]);

        glGenBuffers(1, Buffers);
        glBindBuffer(GL_ARRAY_BUFFER, Buffers[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

        ShaderInfo shaders[] = {
                {GL_VERTEX_SHADER, "triangles.vert"},
                {GL_FRAGMENT_SHADER, "triangles.frag"},
                {GL_NONE, NULL}
        };

        GLuint program = LoadShaders(shaders);
        glUseProgram(program);

        GLint location = glGetAttribLocation(program, "vPosition");
        glVertexAttribPointer(location, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(location);
}
```

Finds the location(or identifier) of a specified vertex attribute

Specifies how to read the buffer data through the attribute

Enable the attribute

Stride, pointer

Full source code: triangle.cpp

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBindVertexArray(VertexArrays[0]);
        glDrawArrays(GL_TRIANGLES, 0, NumVertices);
        glFlush();
}
```

Clear the buffers for color writing

Specifies the current active VAO

Issues a drawing command

Forces the execution of OpenGL commands in finite time