
Computer Graphics

3 - Lab - Sending Data to Shaders

Yoonsang Lee
Hanyang University

Spring 2026

Outline

- Sending Data using **Outs** and **Ins** (Shader → Shader)
- Sending Data by **Specifying Vertex Attributes** (Application → Shader)
- Sending Data using **Uniforms** (Application → Shader)

Sending Data using Outs and Ins (Shader → Shader)

Recall: Vertex Shader

- Input: *Per-vertex* attributes (provided by the application)
 - Position (vertex position)
 - Other attributes such as color, normal, texture coordinates
- Output: *Per-vertex* outputs
 - Position *in clip space* (`gl_Position`)
 - vertex post-processing
 - **Other outputs** such as color, normal, texture coordinates
 - fragment shader
- (We will look at the gray parts later.)

Recall: Fragment Shader

- Input: *Interpolated* **vertex shader outputs**
 - To "**link**" vertex shader output and fragment shader input, **they must have the same type and name.**
 - Fragment shader input is **interpolated** vertex shader output at the fragment location.
- Output: Fragment color
- (We will look at the gray part later.)

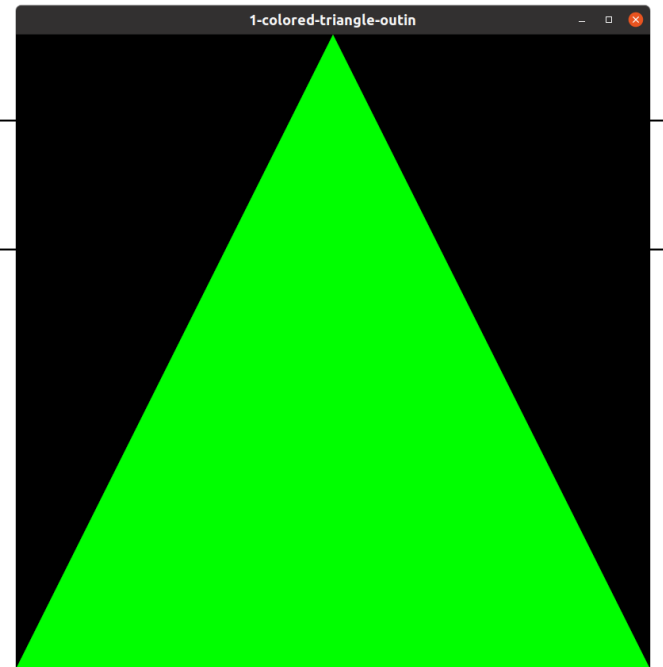
[Code] 1-colored-triangle-outin

Vertex shader

```
#version 330 core
layout (location = 0) in vec3 vin_pos;
out vec4 vout_color;
void main()
{
    gl_Position = vec4(vin_pos.x, vin_pos.y, vin_pos.z, 1.0);
    vout_color = vec4(0,1,0,1);
}
```

Fragment shader

```
#version 330 core
in vec4 vout_color;
out vec4 FragColor;
void main()
{
    FragColor = vout_color;
}
```



Sending Data by Specifying Vertex Attributes (Application → Shader)

Specifying Vertex Attributes

- This is what we have covered in the previous lab.
 - We specified *vertex positions*:

P5. Configure vertex attributes

- Specifying the vertex attribute index
 - If the vertex position attribute index is 0, then:

```
#version 330 core
layout (location = 0) in vec3 vin_pos;

void main()
{
    gl_Position = vec4(vin_pos.x, vin_pos.y, vin_pos.z, 1.0);
}
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
3*glm.sizeof(glm.float32), None);

glEnableVertexAttribArray(0)
```

The index number must match between
the shader and the OpenGL code.

Hanyang University CSE4020, Yoonsang Lee, 2026

- Here, we'll add one more per-vertex attribute – *vertex color*.

Add More Vertex Attributes

- Vertex input data so far (vertex positions only):

```
vertices = glm.array(glm.float32,  
    -1.0, -1.0, 0.0, // left vertex  
    1.0, -1.0, 0.0, // right vertex  
    0.0, 1.0, 0.0 // top vertex  
)
```

- Add per-vertex color:

```
vertices = glm.array(glm.float32,  
    # position          # color  
    -1.0, -1.0, 0.0, 1.0, 0.0, 0.0, # left vertex  
    1.0, -1.0, 0.0, 0.0, 1.0, 0.0, # right vertex  
    0.0, 1.0, 0.0, 0.0, 0.0, 1.0, # top vertex  
)
```

Changes in Vertex Shader

- Let's set vertex attribute indexes as follows:
 - Vertex position: 0
 - Vertex color: 1

```
#version 330 core

layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_color;

out vec4 vout_color;

void main()
{
    gl_Position = vec4(vin_pos.x, vin_pos.y, vin_pos.z, 1.0);
    vout_color = vec4(vin_color, 1); // you can pass a vec3
and a scalar to vec4 constructor
}
```

Changes in Vertex Shader

- The fragment shader is the same.

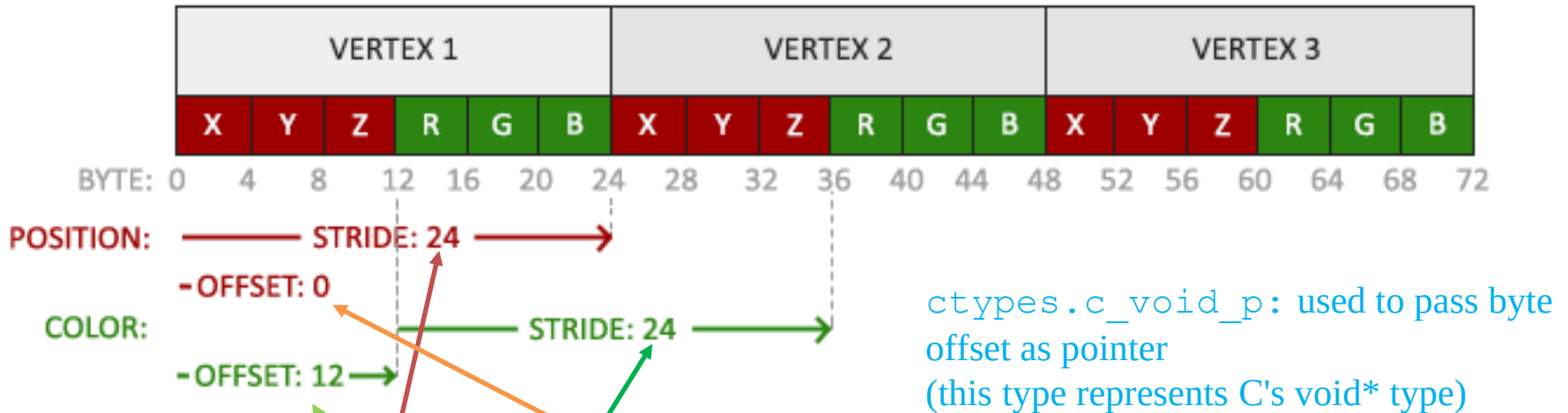
```
#version 330 core

in vec4 vout_color;

out vec4 FragColor;

void main()
{
    FragColor = vout_color;
}
```

Changes in Attributes Configuration



```
# configure vertex positions - index 0
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
6*glm.sizeof(glm.float32), None)
glEnableVertexAttribArray(0)

# configure vertex colors - index 1
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
6*glm.sizeof(glm.float32),
ctypes.c_void_p(3*glm.sizeof(glm.float32)))
glEnableVertexAttribArray(1)
```

[Code] 2-interpolated-triangle

- Vertex input data

```
vertices = glm.array(glm.float32,
    # position      # color
    -1.0, -1.0, 0.0, 1.0, 0.0, 0.0, # left vertex
    1.0, -1.0, 0.0, 0.0, 1.0, 0.0, # right vertex
    0.0, 1.0, 0.0, 0.0, 0.0, 1.0, # top vertex
)
```

- Vertex shader

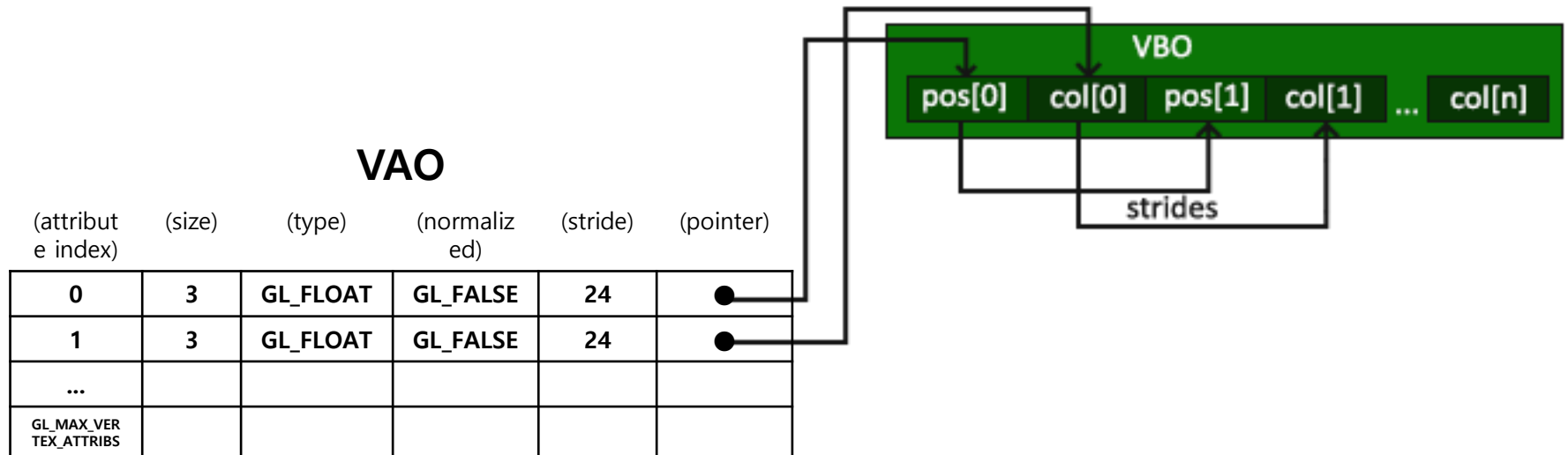
```
#version 330 core
layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_color;
out vec4 vout_color;
void main()
{
    gl_Position = vec4(vin_pos.x, vin_pos.y, vin_pos.z, 1.0);
    vout_color = vec4(vin_color, 1);
}
```

- Vertex attribute configuration

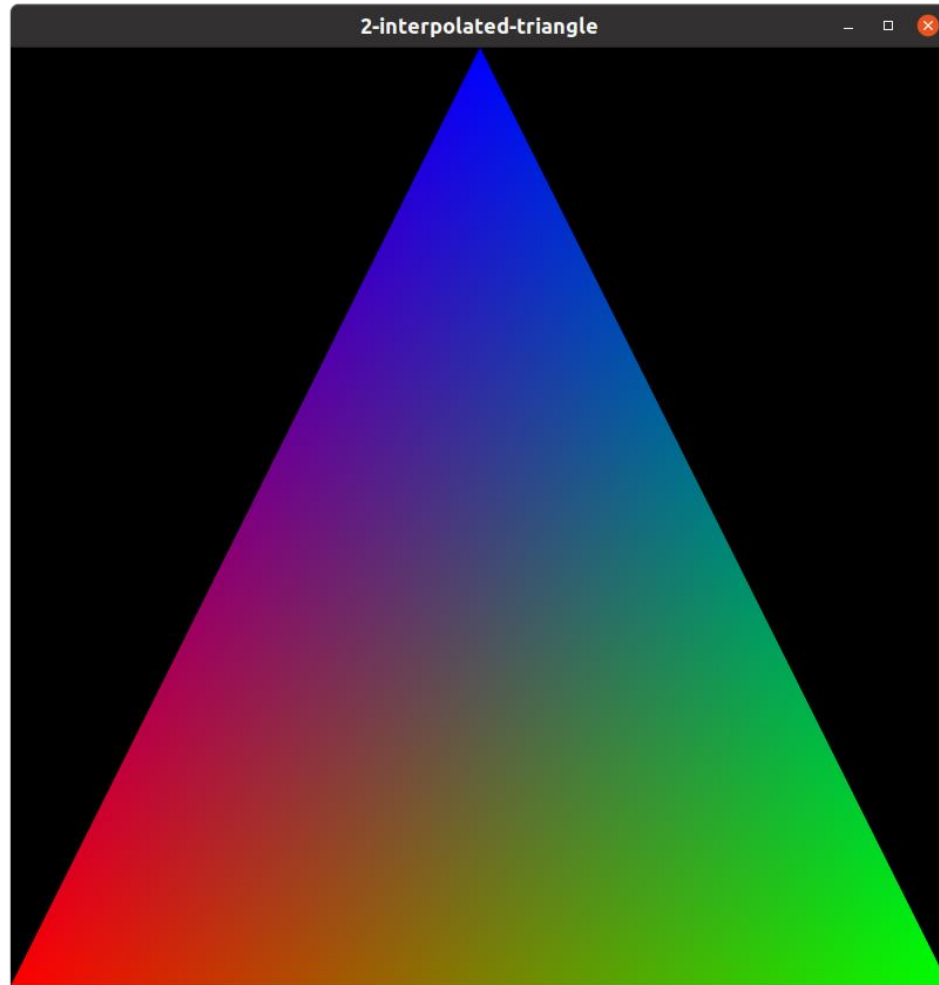
```
# configure vertex positions - index 0
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6*glm.sizeof(glm.float32), None)
glEnableVertexAttribArray(0)

# configure vertex colors - index 1
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*glm.sizeof(glm.float32),
    ctypes.c_void_p(3*glm.sizeof(glm.float32)))
glEnableVertexAttribArray(1)
```

VAO & VBO in this example



[Code] 2-interpolated-triangle

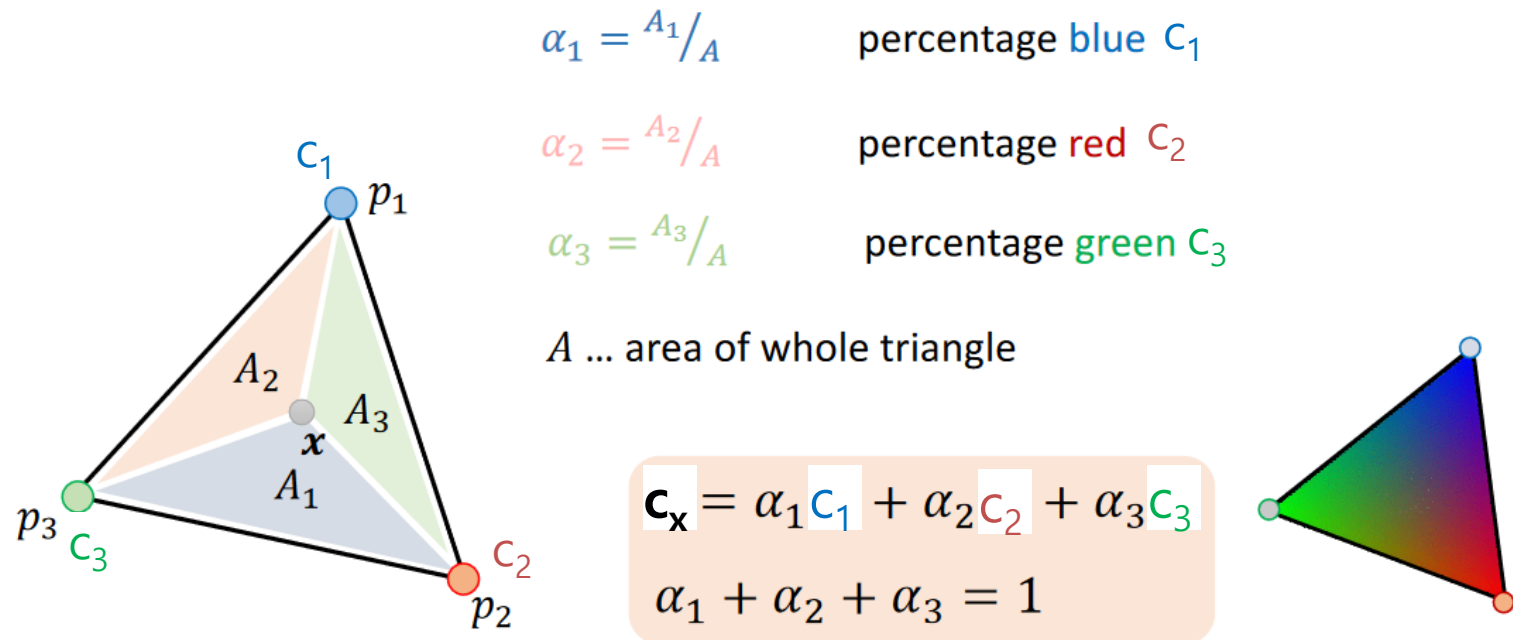


"Interpolated" Fragment Shader Inputs

- Input: *Interpolated* vertex shader outputs
 - To "link" vertex shader output and fragment shader input, they must have the same type and name.
 - Fragment shader input is **interpolated vertex shader output at the fragment location.**
- Output: Fragment color

Fragment Interpolation

- Fragment shader inputs are interpolated based on where the fragment is on the triangle.
- Barycentric interpolation



Quiz 3

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Sending Data using Uniforms (Application → Shader)

Uniform

- *Uniforms* provide the same value to all shader invocations.
 - Unlike vertex attributes, uniforms do not change per vertex/fragment.

Example: setting a constant color

```
#version 330 core
out vec4 FragColor;
uniform vec3 u_color;
void main()
{
    FragColor = vec4(u_color, 1.0);
}
```

- Same value for all fragments/vertices
- Global within a shader program
 - can be accessed from any shader in the same shader program object.
- Keeps value until updated

Setting Uniform Value

- First, we need to find the "location" of the uniform in the shader program.

```
loc_u_color = glGetUniformLocation(shader_program, 'u_color')  
# find uniform's location
```

- `glGetUniformLocation(program, name)`
 - Returns the location of a uniform.
 - `program`: the shader program object
 - `name`: the name of the uniform

Setting Uniform Value

- Once we get the uniform's location, we can update the uniform's value.
- Note that updating uniform require you to first **activate the shader program.**

```
glUseProgram(shader_program)      # updating uniform  
require you to first activate the shader program  
  
glUniform3f(loc_u_color, 0, 0, 1) # set uniform's value
```

glUniform*()

- Specify the value of a uniform variable for the current program object.
- Postfixes
 - f: the function expects a float as its value.
 - i: the function expects an int as its value.
 - ui: the function expects an unsigned int as its value.
 - 3f: the function expects 3 floats as its value.
 - fv: the function expects a float vector/array as its value.
- e.g. `glUniform4f()` takes 4 floats after the uniform's location as arguments.

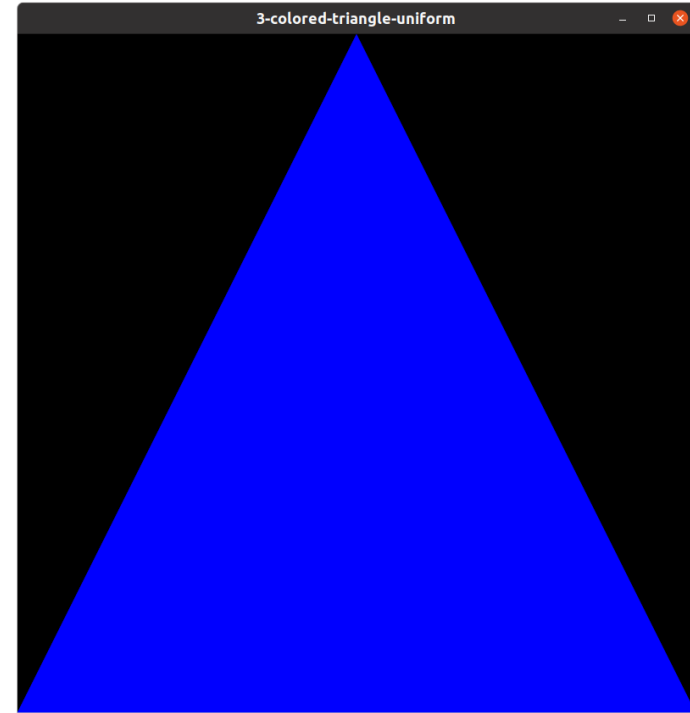
[Code] 3-colored-triangle-uniform

- Fragment shader

```
#version 330 core
out vec4 FragColor;
uniform vec3 u_color;
void main()
{
    FragColor = vec4(u_color, 1.0);
}
```

- Setting Uniform Value

```
loc_u_color = glGetUniformLocation(shader_program,
'u_color') # find uniform's location
glUseProgram(shader_program) # updating uniform
require you to first activate the shader program
glUniform3f(loc_u_color, 0, 0, 1) # set uniform's value
```



For Animation

- For animation, we need to update uniforms every frame.
- Get uniform locations at initialization.
- Update uniforms in the rendering loop.

```
glGetUniformLocation
```

```
while:
```

```
    glUseProgram
```

```
    glUniform*
```

```
    glBindVertexArray(VAO)
```

```
    glDrawArrays
```

[Code] 4-color-changing-triangle

```
...
def main():
    ...
    # load shaders
    shader_program = load_shaders(g_vertex_shader_src,
g_fragment_shader_src)

    # get uniform locations
    loc_u_color = glGetUniformLocation(shader_program,
'u_color')
    ...
    # loop until the user closes the window
    while not glfwWindowShouldClose(window):
        ...
        glUseProgram(shader_program)

        # update uniforms
        t = glfwGetTime()
        blue = (glm.sin(t) + 1) * .5
        glUniform3f(loc_u_color, 0, 0, blue)
        ...
```

* The full source code can be found at <https://github.com/yssl/CSE4020>

Time for Lab Check

- Let's start today's lab check.
- TA will guide you.