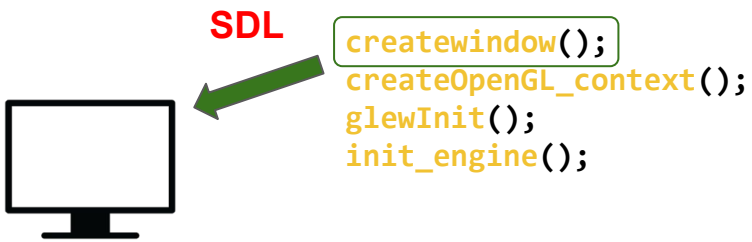
OpenGL Pipeline

Evangelou Iordanis

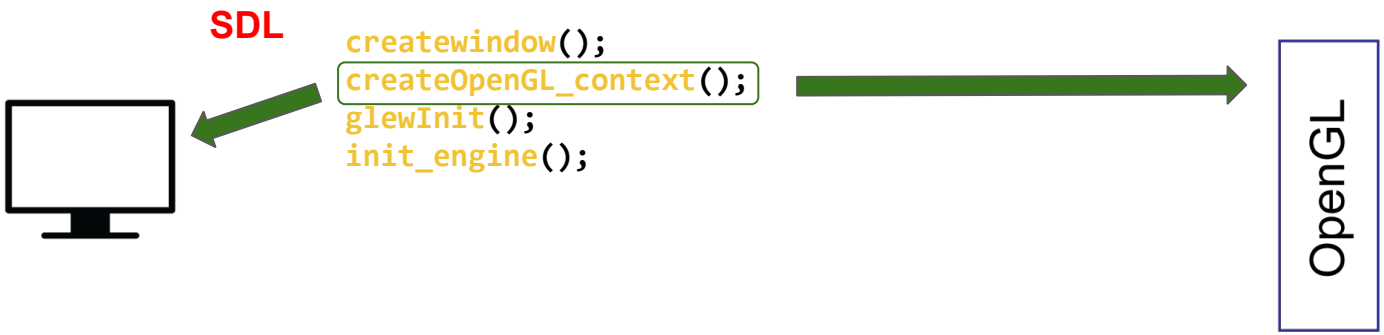
Presentation co-author Anastasios Gkaravelis

Recap

Recap

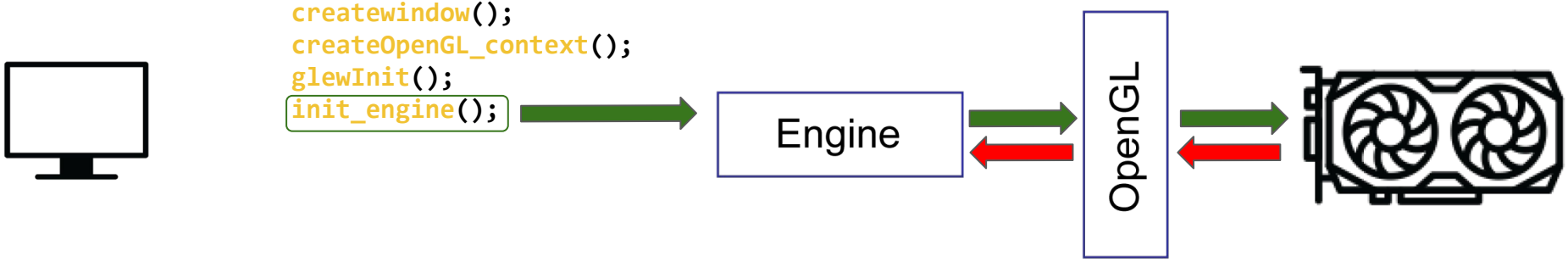

SDL
createwindow();
createOpenGL_context();
glewInit();
init_engine();

Recap


SDL
createwindow();
createOpenGL_context();
glewInit();
init_engine();
OpenGL

Recap

Recap

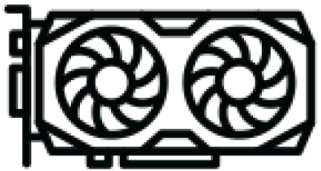

createwindow();
createOpenGL_context();
glewInit();
init_engine();
Engine
OpenGL

Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{



}
```

Engine

OpenGL



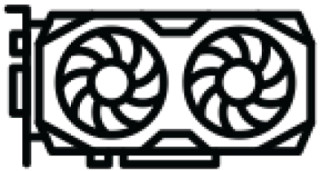
Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();


}
```

SDL

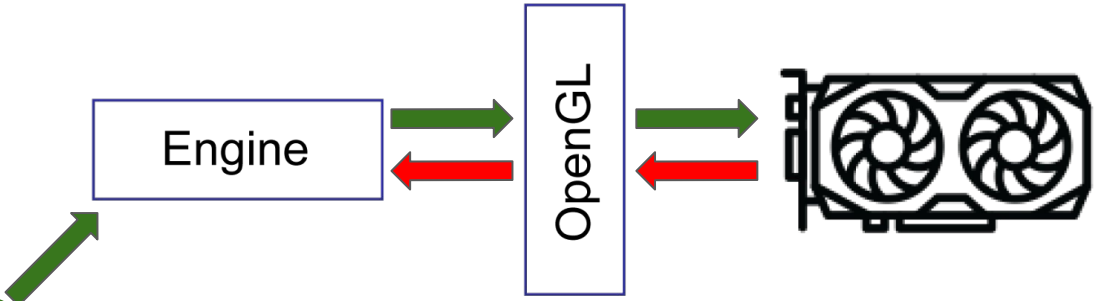
Engine

OpenGL



Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);

}
```

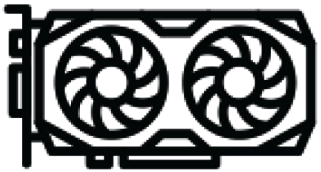


Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

Engine

OpenGL



gl context

Recap

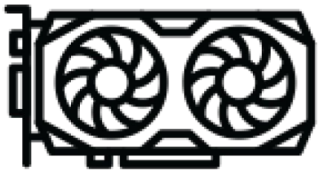
CPU side

GPU side

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

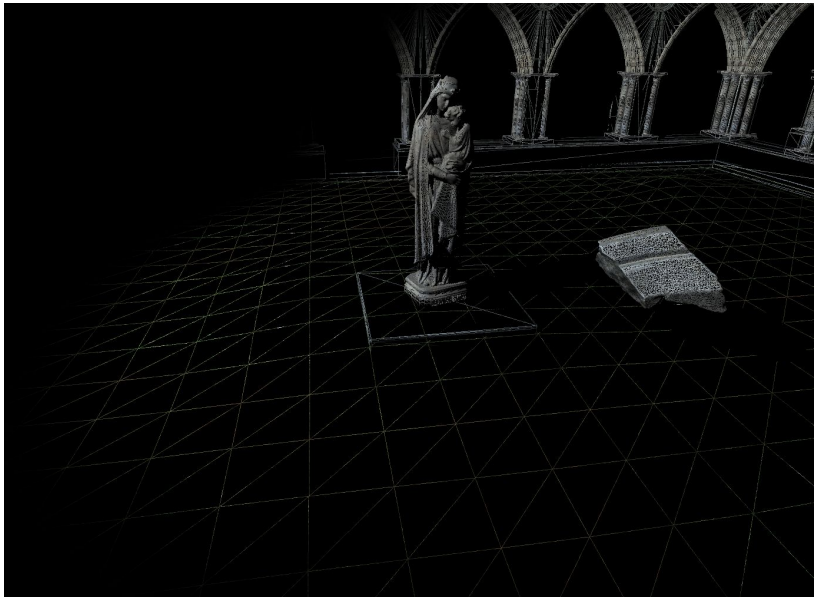
Engine

OpenGL



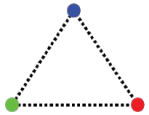
Rasterization pipeline

- Models consist of primitives
  - most commonly triangles



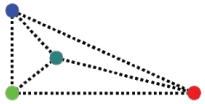
Rasterization pipeline

Input

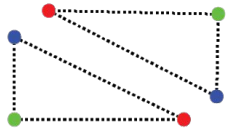


Vertex Shader

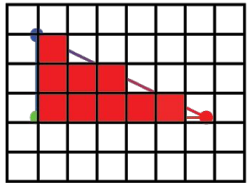
Tessellation Shader



Geometry Shader



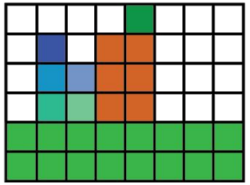
Rasterization



Fragment Shader

Fragment Test

Output Buffer



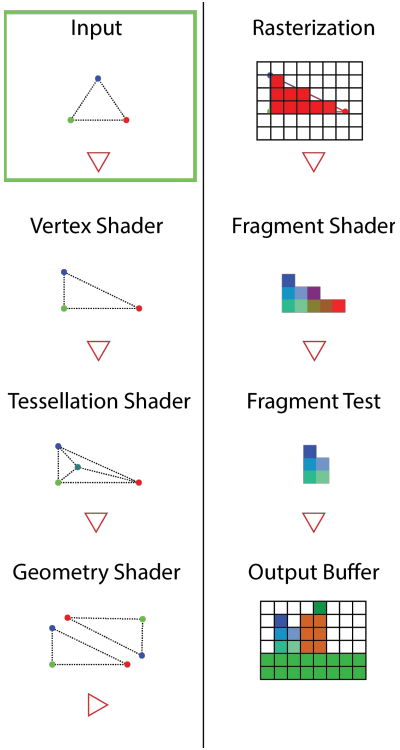
Rasterization pipeline

Input:

- An array of vertices

v0	v1	v2	v3	v4	v6	v7	v8	v9	v10

{ 0.5, 0.5, 0.5 }



Rasterization pipeline

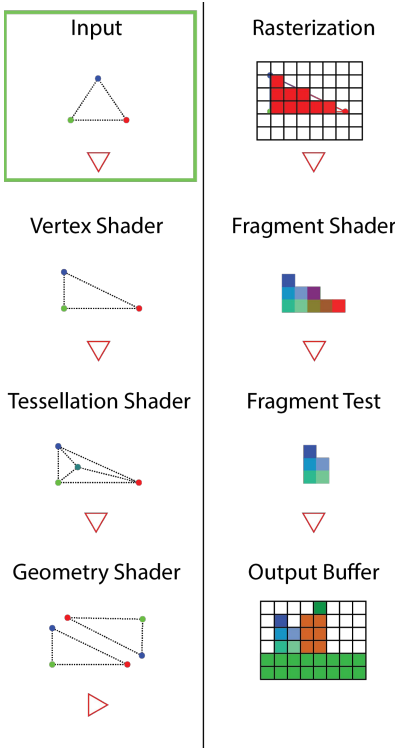
Input:

- An array of vertices

v0	v1	v2	v3	v4	v6	v7	v8	v9	v10

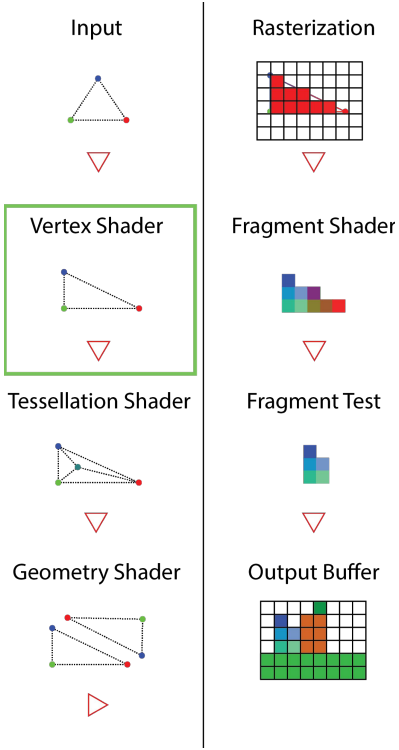
- Allocate GPU memory for vertices with Array Buffers

```
GLfloat data[] = { 0.0, 0.5, 0.0, -0.5, -0.5, 0.0, 0.5, -0.5, 0.0 };
GLuint vbo = 0;

// generate a handle to a buffer that holds vertex data
glGenBuffers(1, &vbo);

// Bind the VBO to get access to the buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo);

// Allocate buffer and fill it with data of size buffer_size in bytes
glBufferData(GL_ARRAY_BUFFER, sizeof(data), &data[0], GL_STATIC_DRAW);

// Unbind VBO
glBindBuffer(GL_ARRAY_BUFFER, 0);
```



Rasterization pipeline

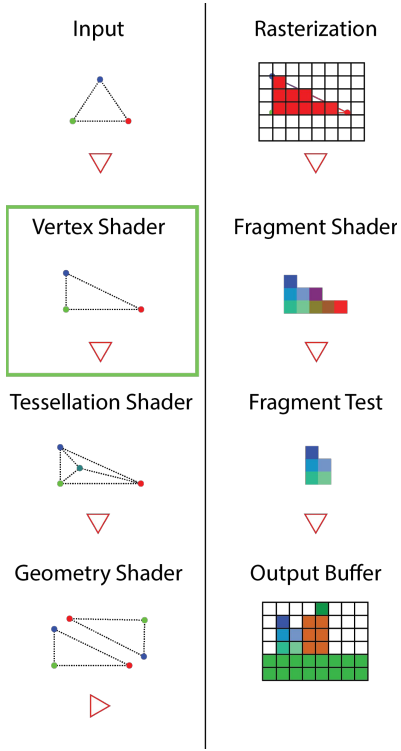
- Processes a stream of vertices



Rasterization pipeline

- Processes a stream of vertices

```
#version 330 core
layout(location = 0) in vec3 coord3d; // input
out vec3 attr_color; // output

void main(void)
{
    attr_color = vec3(1.0);
    gl_Position = vec4(coord3d, 1.0); // output
}
```

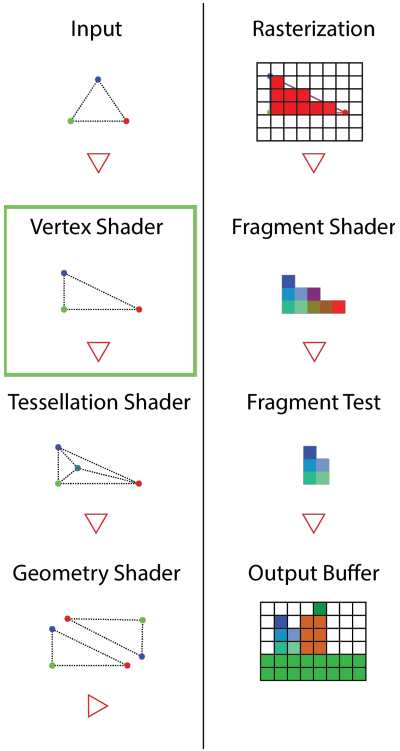


Rasterization pipeline

- Processes a stream of vertices

```
#version 330 core
layout(location = 0) in vec3 coord3d; // input
out vec3 attr_color; // output

void main(void)
{
    attr_color = vec3(1.0);
    gl_Position = vec4(coord3d, 1.0); // output
}
```

```
const char* source = LoadStringFile(fileName);
GLuint vshader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vshader, 1, &source, nullptr);
glCompileShader(vshader);
```

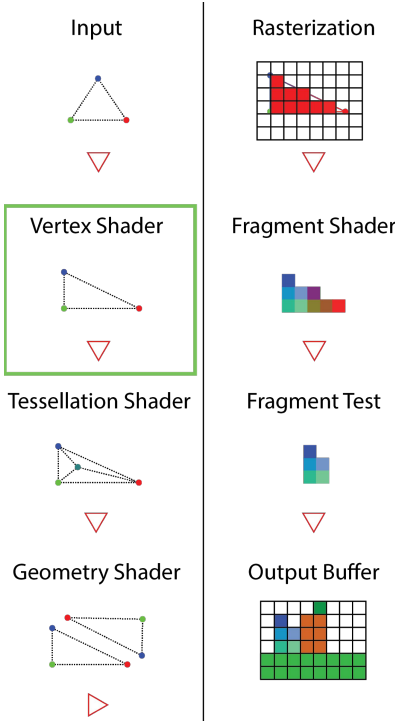


Rasterization pipeline

- Processes a stream of vertices

```
#version 330 core
layout(location = 0) in vec3 coord3d; // input
out vec3 attr_color; // output

void main(void)
{
    attr_color = vec3(1.0);
    gl_Position = vec4(coord3d, 1.0); // output
}
```

```
GLuint vao = 0;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);

glEnableVertexAttribArray(0);

//loc_index, elem_size, type, isNormalized, stride, offset
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

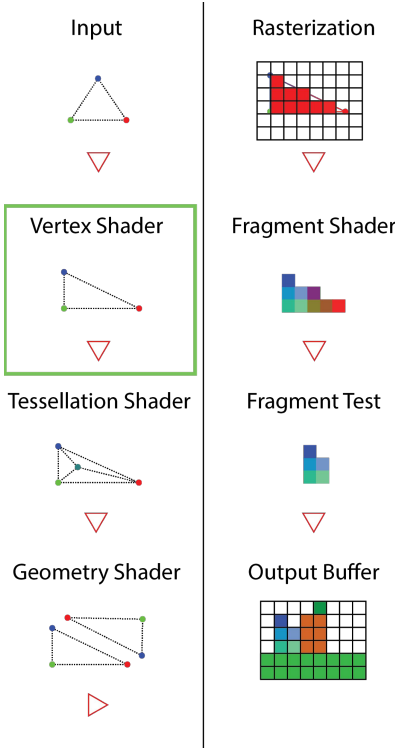


Rasterization pipeline

- Processes a stream of vertices

```
#version 330 core
layout(location = 0) in vec3 coord3d; // input
out vec3 attr_color; // output

void main(void)
{
    attr_color = vec3(1.0);
    gl_Position = vec4(coord3d, 1.0); // output
}
```

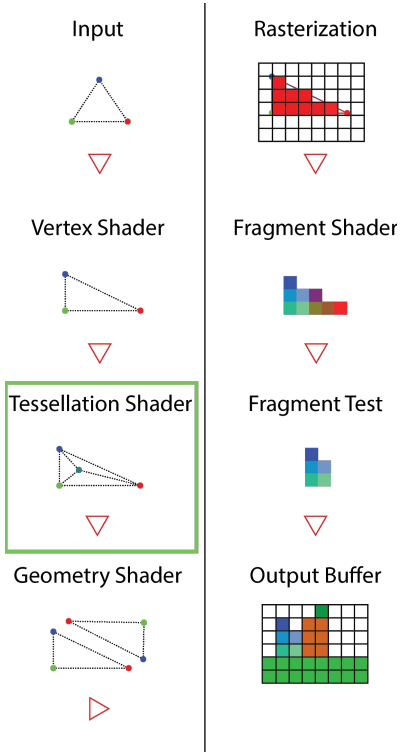
- Apply transformations in vertex level
- Pass attributes to the next stage of the pipeline
    - `gl_Position` (must set)
    - `out variables` (optional)



Rasterization pipeline

Tessellation Shader **(<u>Optional</u>)**

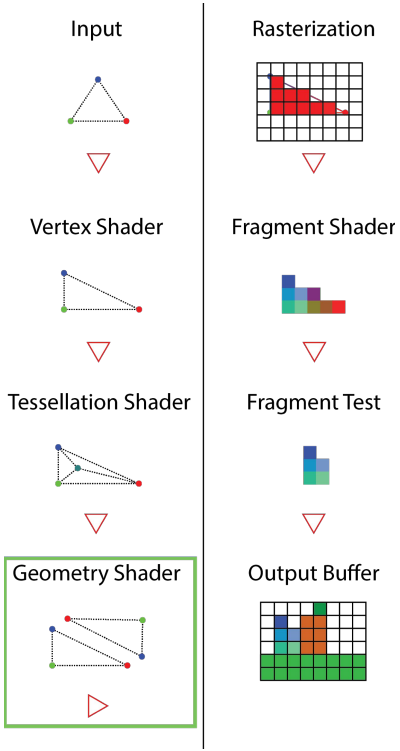
- Receives primitives from previous stage and tessellate them
- Determine the amount of tessellation to apply to a primitive



Rasterization pipeline

Geometry Shader **(<u>Optional</u>)**

- Receives primitives from previous stage
- <u>Generate</u> and output zero or more primitives
  - Not necessarily of the same type

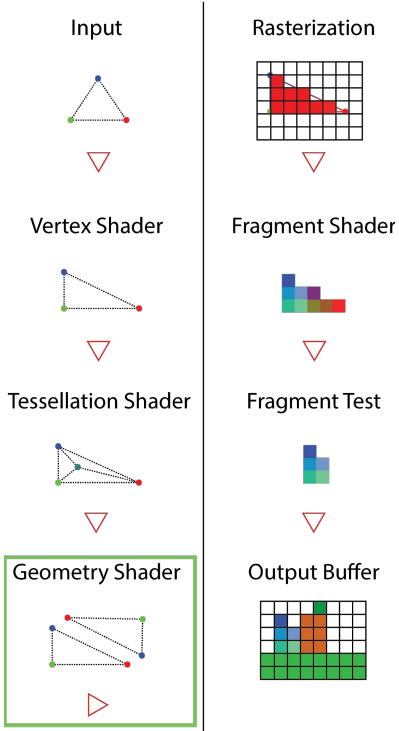


Rasterization pipeline

Geometry Shader (**Optional**)

- Receives primitives from previous stage
- Generate and output zero or more primitives
  - Not necessarily of the same type

```
#version 330 core
layout(points) in; // input
layout(triangle_strip, max_vertices = 3) out; // output

void main(void)
{
    vec3 pos = gl_in[0].gl_Position;
    gl_Position = vec4(pos, 0.0) + vec4(-0.5, -0.5, 0.0, 0.0);
    EmitVertex();
    gl_Position = vec4(pos, 0.0) + vec4(0.5, -0.5, 0.0, 0.0);
    EmitVertex();
    gl_Position = vec4(pos, 0.0) + vec4(-0.5, 0.5, 0.0, 0.0);
    EmitVertex();
    EndPrimitive();
}
```



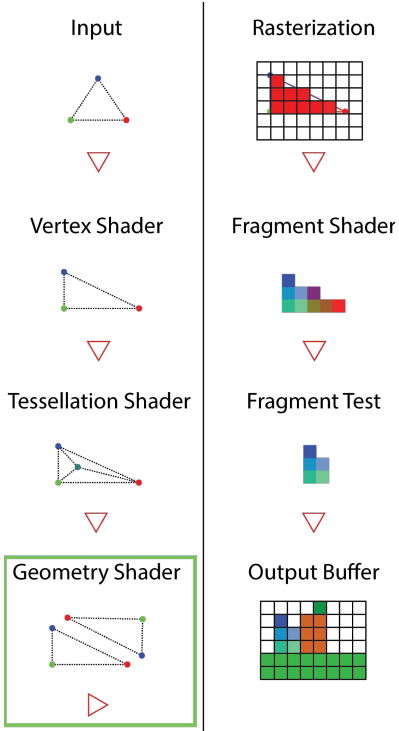
Rasterization pipeline

Geometry Shader (**Optional**)

- Receives primitives from previous stage
- Generate and output zero or more primitives
  - Not necessarily of the same type

```
#version 330 core
layout(points) in; // input
layout(triangle_strip, max_vertices = 3) out; // output

void main(void)
{
    vec3 pos = gl_in[0].gl_Position;
    gl_Position = vec4(pos, 0.0) + vec4(-0.5, -0.5, 0.0, 0.0);
    EmitVertex();
    gl_Position = vec4(pos, 0.0) + vec4(0.5, -0.5, 0.0, 0.0);
    EmitVertex();
    gl_Position = vec4(pos, 0.0) + vec4(-0.5, 0.5, 0.0, 0.0);
    EmitVertex();
    EndPrimitive();
}
```

```
const char* source = LoadStringFile(fileName);
GLuint gshader = glCreateShader(GL_GEOMETRY_SHADER);
glShaderSource(gshader, 1, &source, nullptr);
glCompileShader(gshader);
```

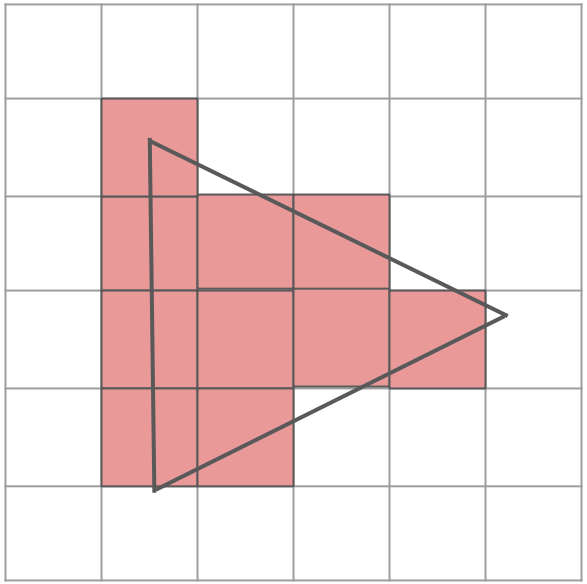


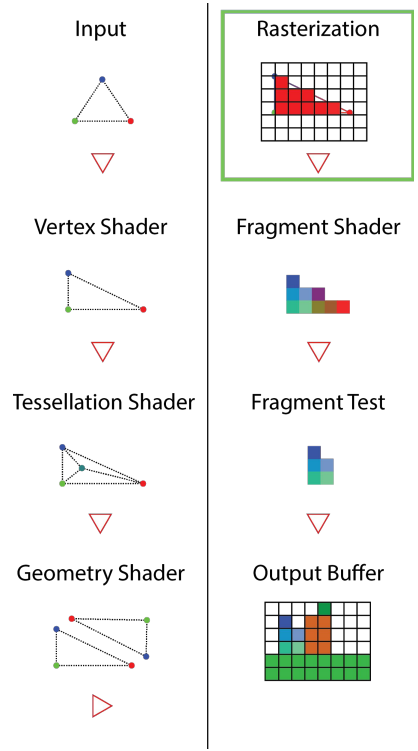
Rasterization pipeline

Rasterization **(non-programmable)**

- Projects primitives onto the image plane
- For each pixel covered, generates a new fragment for the next stage

(6 x 6)

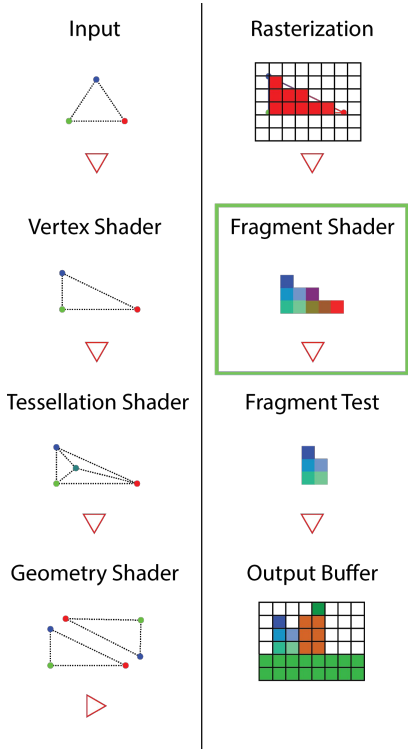




Rasterization pipeline

Fragment shader

- Processes each fragment independently

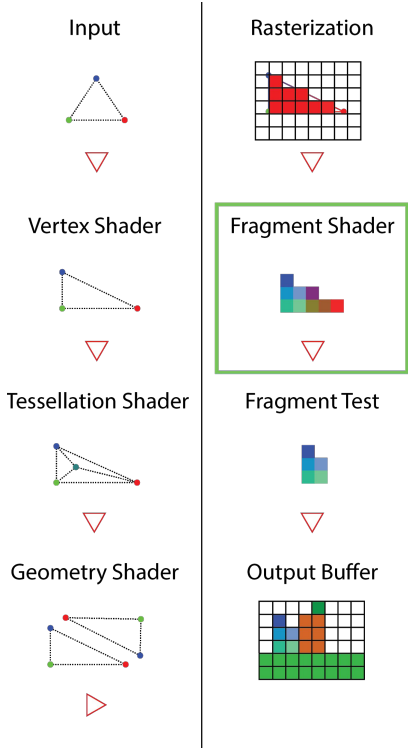


Rasterization pipeline

Fragment shader

- Processes each fragment independently

```
#version 330 core
layout(location = 0) out vec4 out_color; // output
in vec3 attr_color; // input

void main(void)
{
    out_color = vec4(0.1, 0.9, 0.55, 1.0); // output
}
```



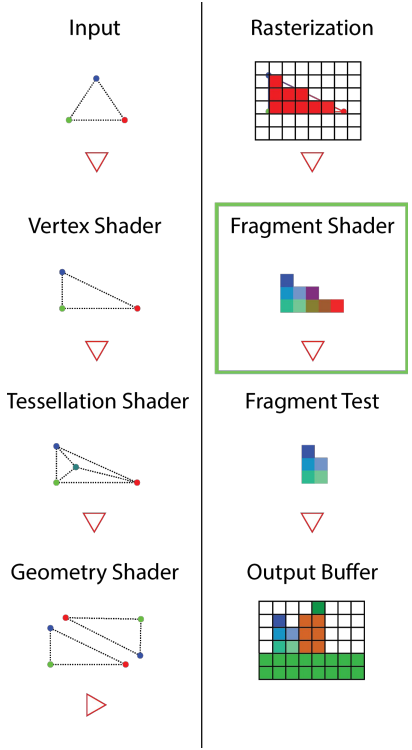
Rasterization pipeline

Fragment shader

- Processes each fragment independently

```
#version 330 core
layout(location = 0) out vec4 out_color; // output
in vec3 attr_color; // input

void main(void)
{
    out_color = vec4(0.1, 0.9, 0.55, 1.0); // output
}
```

```
const char* source = LoadStringFile(fileName);
GLuint fshader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fshader, 1, &source, nullptr);
glCompileShader(fshader);
```



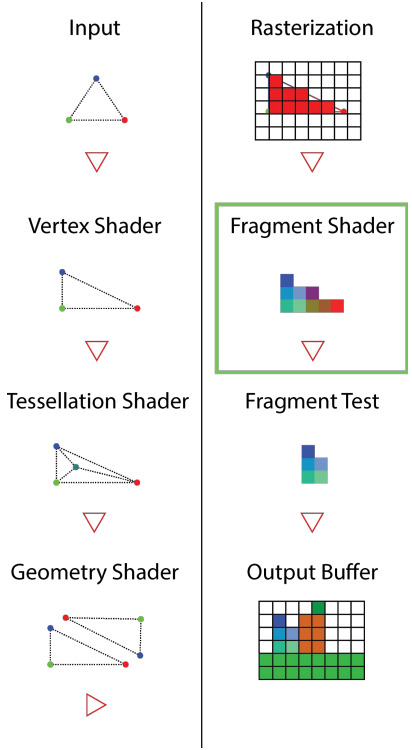
Rasterization pipeline

Fragment shader

- Processes each fragment independently

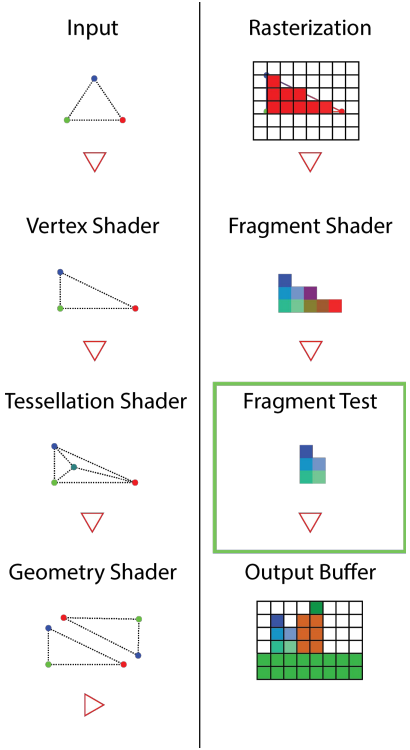
```
#version 330 core
layout(location = 0) out vec4 out_color; // output
in vec3 attr_color; // input

void main(void)
{
    vec2 pixel = gl_FragCoord.xy; // [0, width] x [0, height]
    float depth = gl_FragCoord.z; // [0, 1]
    out_color = vec4(0.1, 0.9, 0.55, 1.0); // output
    // discard;
}
```

- Index the pixel referenced by the current fragment with `gl_FragCoord.xy`
- Index the depth of the fragment with `gl_FragCoord.z`
- Drop the current fragment with the `discard` statement



Rasterization pipeline

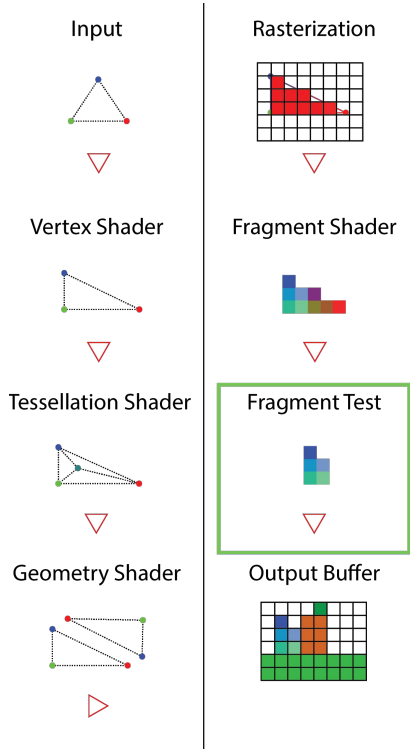
Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**



Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

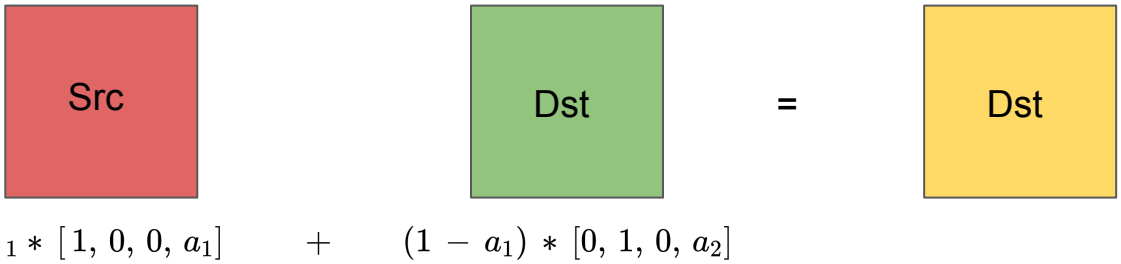
- Blending

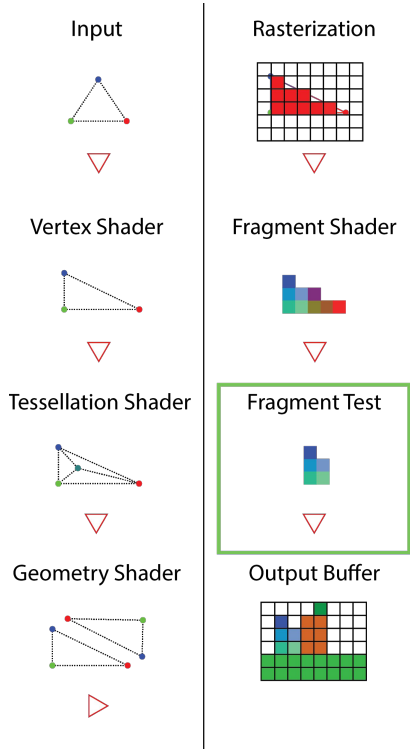


Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending





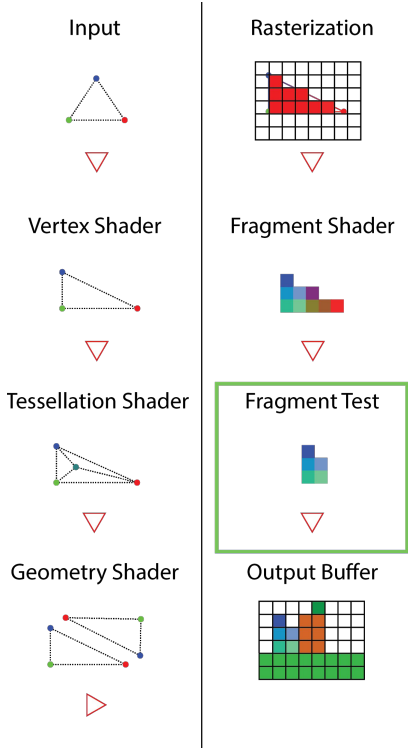
Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending

$$a_1 * [1, 0, 0, a_1] \quad + \quad (1 - a_1) * [0, 1, 0, a_2]$$

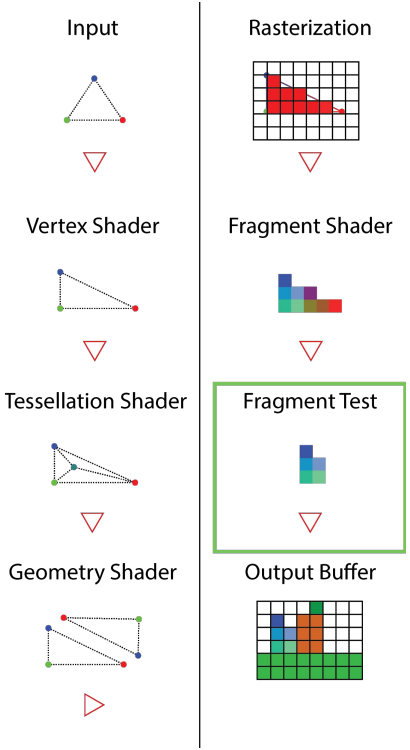
```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glBlendEquation(GL_FUNC_ADD); // +, -, max(), min()

// [...] drawing operations

glDisable(GL_BLEND);
```



Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

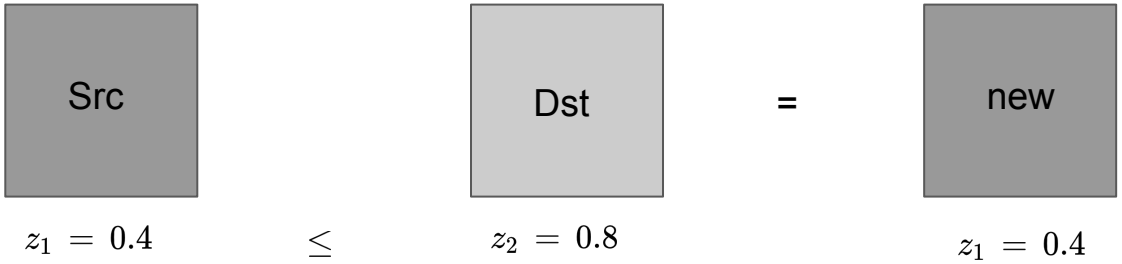
- Blending
- Depth test

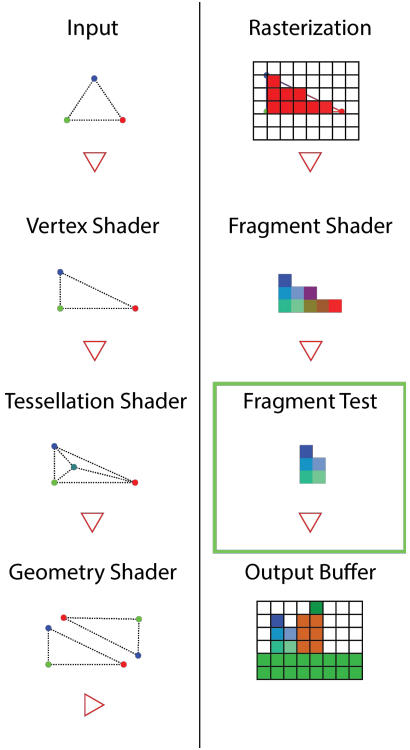


Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test

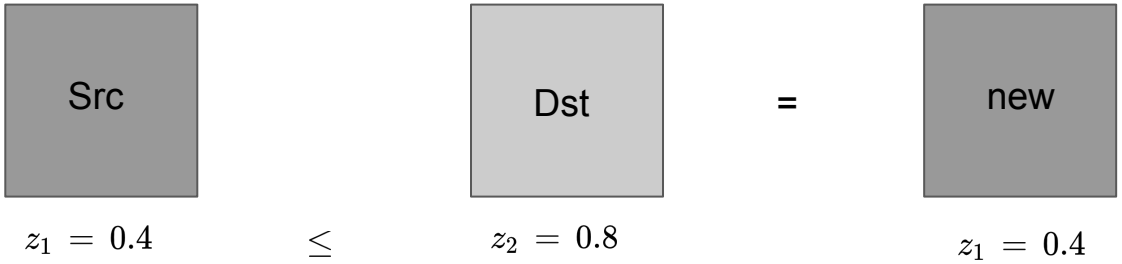




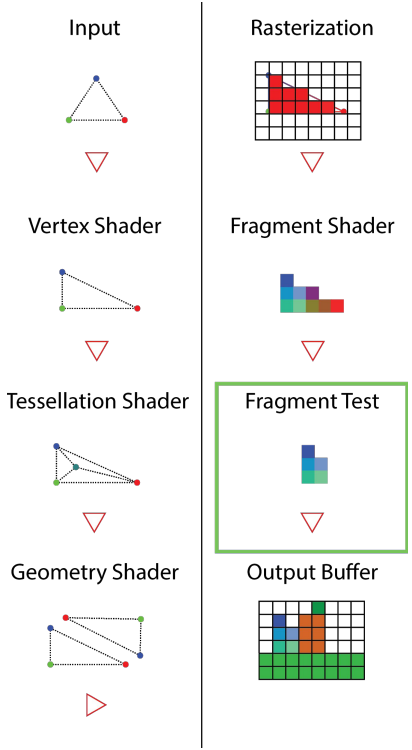
Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test



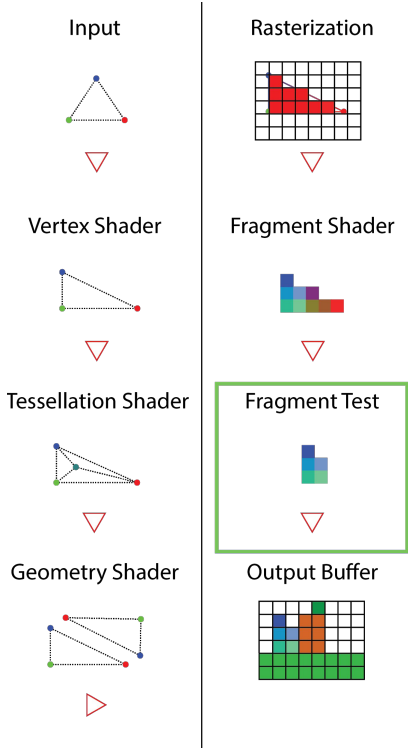

```
glEnable(GL_DEPTH_TEST);
glClear(GL_DEPTH_BUFFER_BIT);
glDepthFunc(GL_LEQUAL); // <, <=, >, >=, ==, always, never

// [...] drawing operations

glDisable(GL_DEPTH_TEST);
```



Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

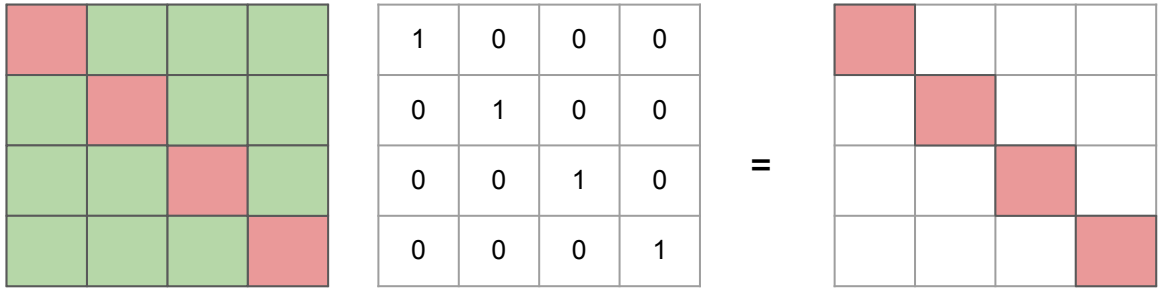
- Blending
- Depth test
- Stencil test

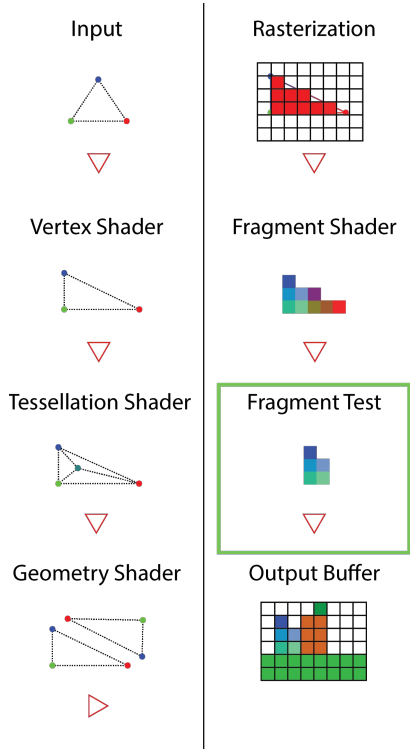


Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test
- Stencil test

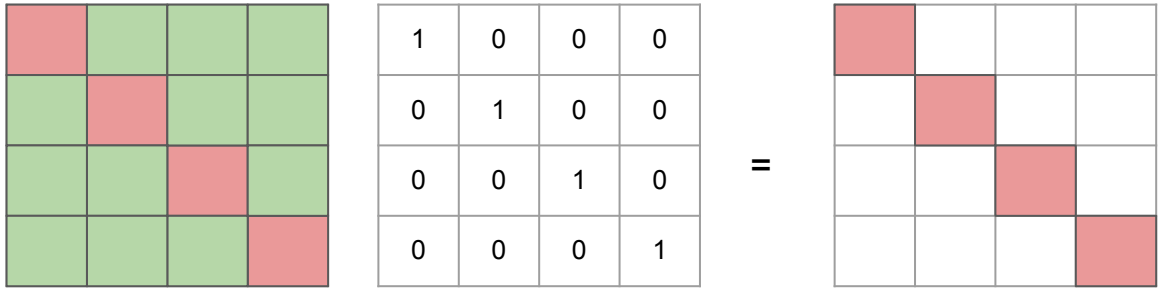




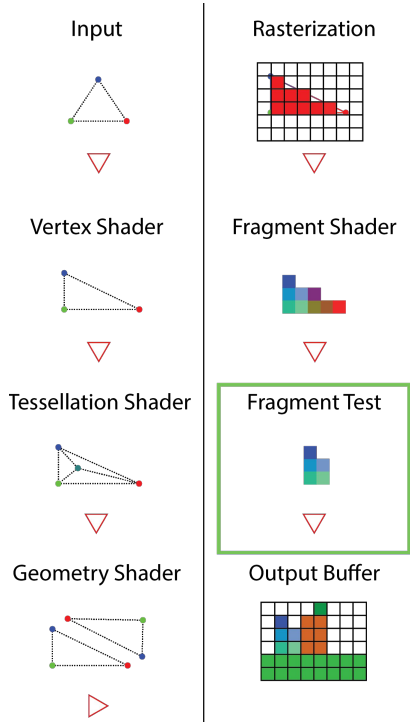
Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test
- Stencil test



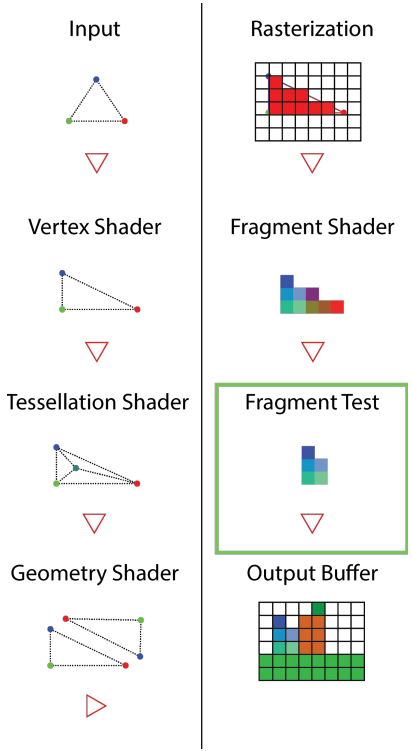

```
glEnable(GL_STENCIL_TEST);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE); // sfail, dfail, pass
glStencilMask(0xFF); // bit to enable write op
glStencilFunc(GL_ALWAYS, 1, 0xFF); // always passes
// [...] drawing operations
glStencilMask(0x00); // bit to disable write op
glStencilFunc(GL_NOTEQAUL, 1, 0xFF); // pass if (ref & mask) != (stencil & mask)
// [...] drawing operations
glDisable(GL_STENCIL_TEST);
```



Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

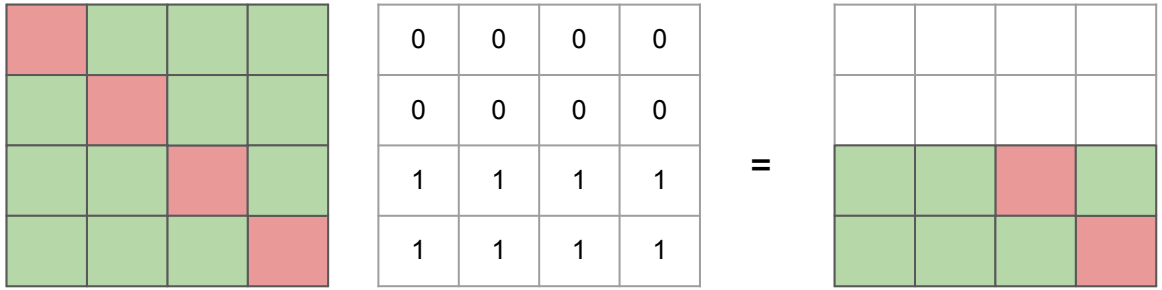
- Blending
- Depth test
- Stencil test
- Scissor test

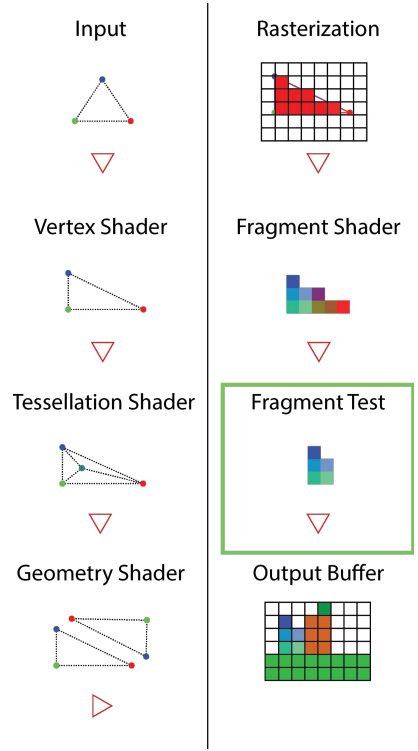


Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test
- Stencil test
- Scissor test

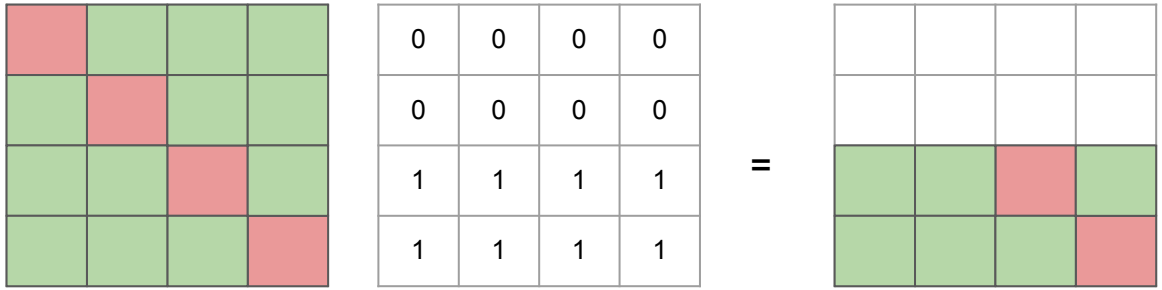




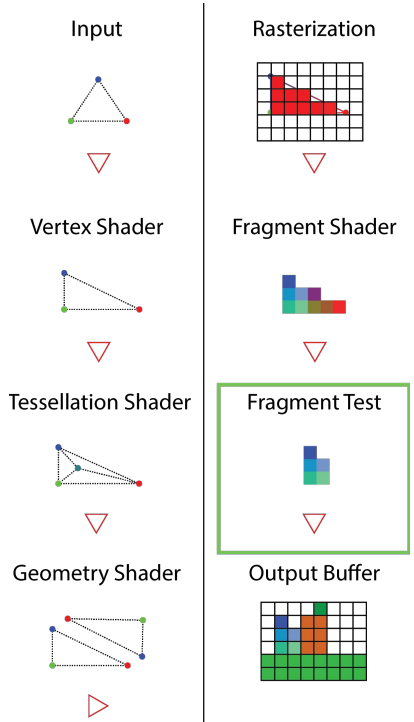
Rasterization pipeline

Per-fragment operations <u>after</u> the fragment shader stage **(non-programmable)**

- Blending
- Depth test
- Stencil test
- Scissor test



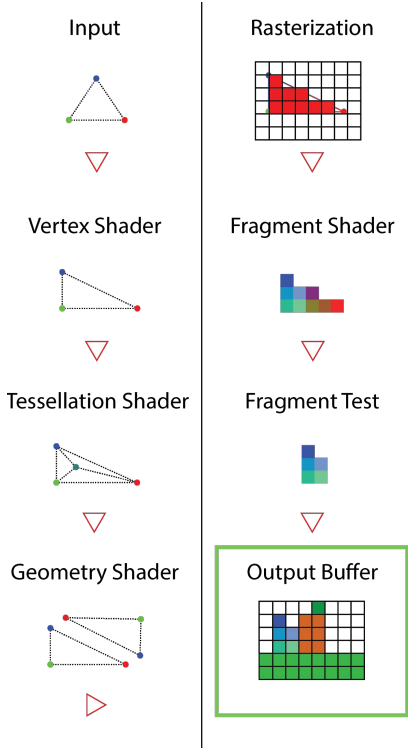

```
glEnable(GL_SCISSOR_TEST);
glScissor(x, y, width, height); // (0, 0, 4, 2)

// [...] drawing operations

glDisable(GL_SCISSOR_TEST);
```



Rasterization pipeline

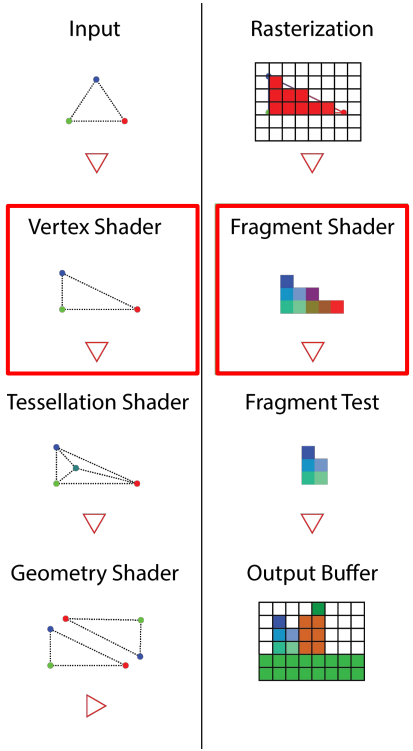
- Write to output buffer

```
glBindFramebuffer(GL_FRAMEBUFFER, 0); // default output buffer
glClear(GL_COLOR_BUFFER_BIT);
glViewport(0, 0, screen_width, screen_height);

// [...] drawing operations
```



Rasterization pipeline

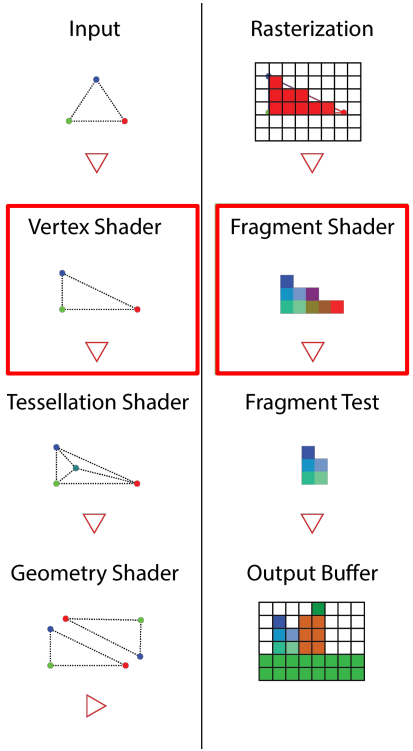
- Load and compile your shaders
  - You need <u>at least</u> a vertex and a fragment shader



Rasterization pipeline

- Load and compile your shaders
  - You need at least a vertex and a fragment shader
- Create a shader program to link shaders

```
GLuint program = glCreateProgram();
GLuint vshader = this->createVertexShader();
GLuint fshader = this->createFragmentShader();
glAttachShader(program, vshader);
glAttachShader(program, fshader);
glLinkProgram(program);
```



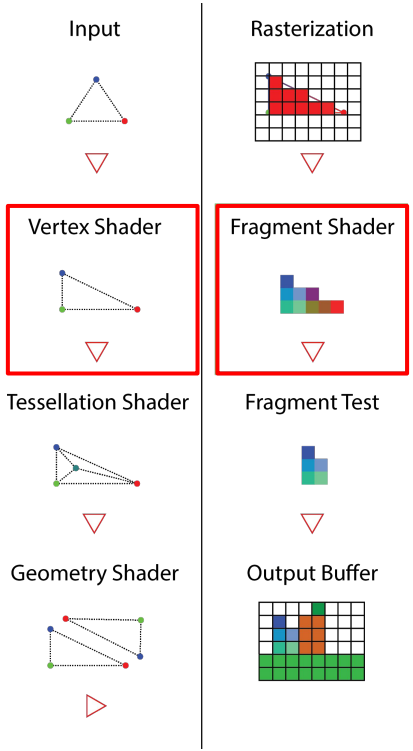
Rasterization pipeline

- Load and compile your shaders
  - You need at least a vertex and a fragment shader
- Create a shader program to link shaders

```
GLuint program = glCreateProgram();
GLuint vshader = this->createVertexShader();
GLuint fshader = this->createFragmentShader();
glAttachShader(program, vshader);
glAttachShader(program, fshader);
glLinkProgram(program);
```

- Retrieve the location of uniform variables

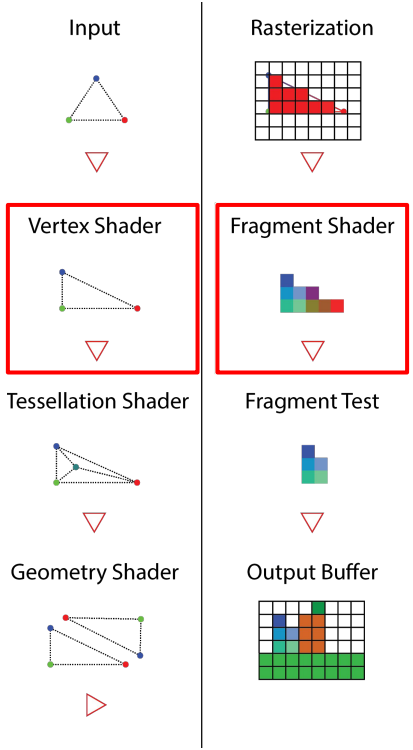
```
std::string uniform_name = “attr_color”;

// Save locally (e.g hash map) the location for each uniform
uniforms[uniform_name] = glGetUniformLocation(program, uniform_name.c_str());
```



Rasterization pipeline

- Load and compile your shaders
  - You need at least a vertex and a fragment shader
- Create a shader program to link shaders
- Retrieve the location of uniform variables
- Launch the rendering loop

```
// [..] bind a framebuffer to draw
// [..] enable all the fragments tests (if any)
glUseProgram(program);
glUniform3f(uniforms[“attr_color”], 0.0, 0.0, 1.0);
glBindVertexArray(vao);

glDrawArrays(GL_TRIANGLES, 0, number_of_vertices);

// [..] disable appropriate operators
glBindVertexArray(0);
glUseProgram(0);
```

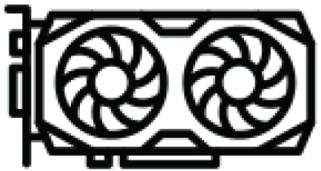


Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

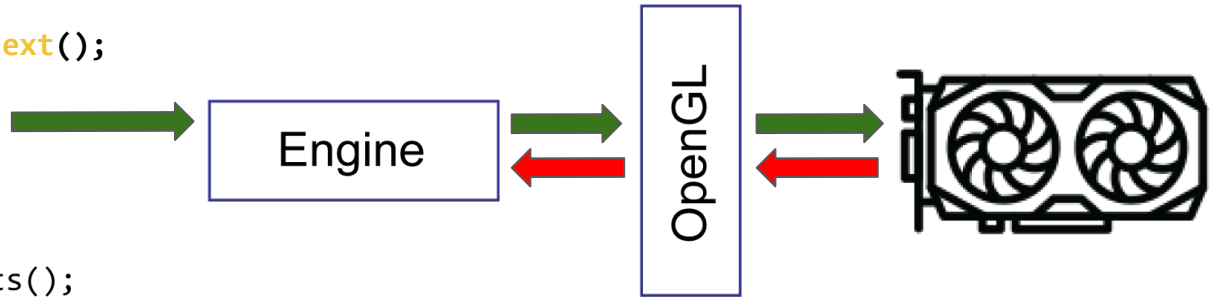
Engine

OpenGL



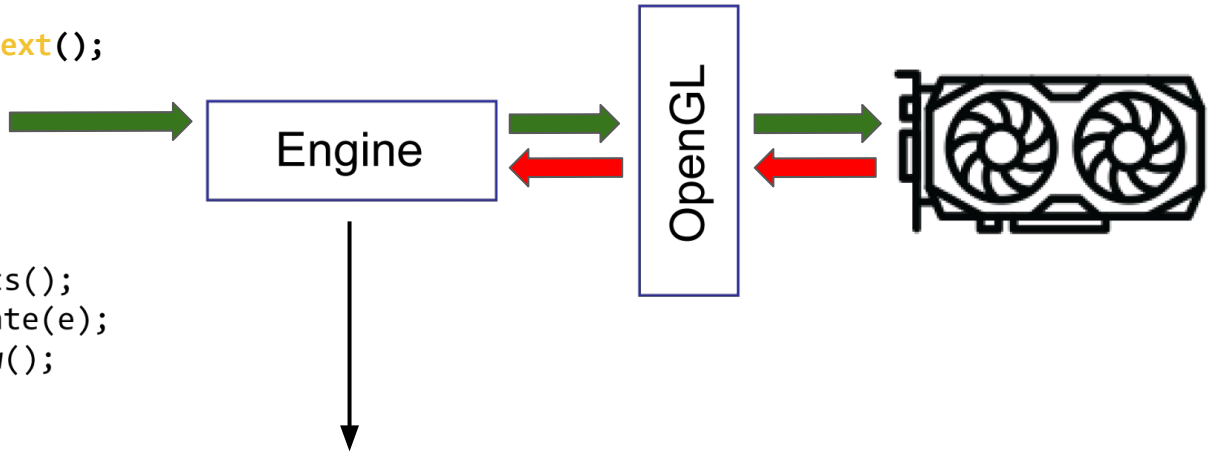
Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```



Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

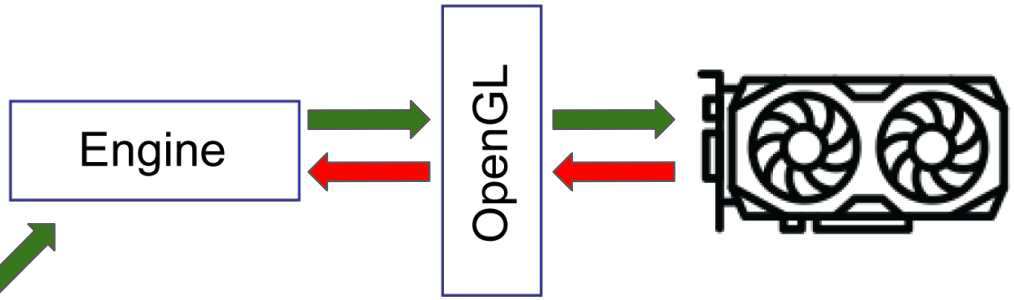


- Load geometry and initialize the openGL buffers
  - Vertex array buffers
  - Vertex array objects/attributes
- Load and compile shaders
- Create and link a program with shaders
- Query and cache uniform locations

Recap

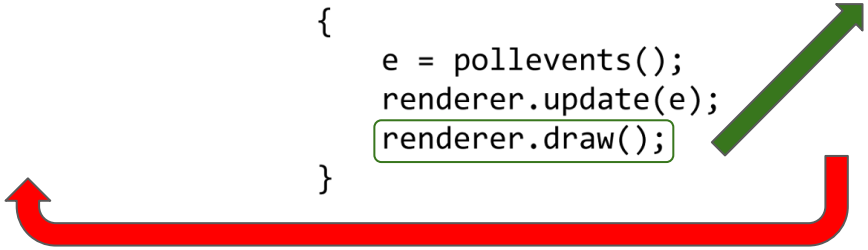
```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

gl context

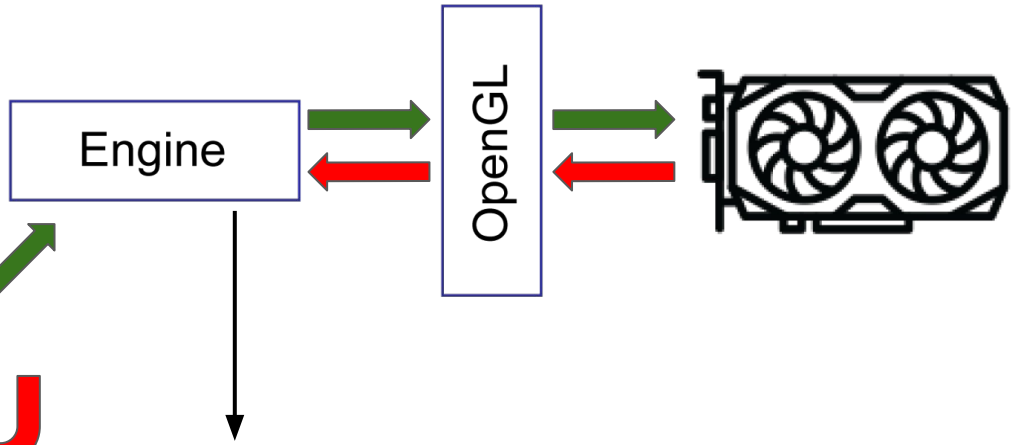


Recap

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```



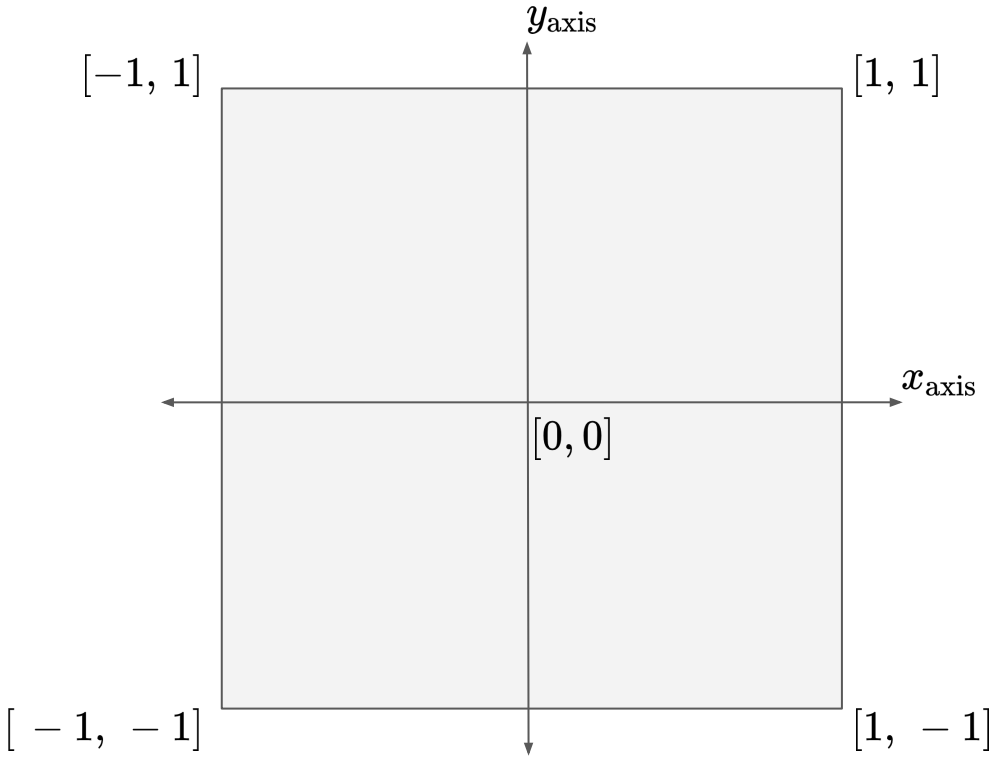
gl context



- Bind framebuffer to draw
- Enable opengl function operations
- Bind your program
  - update uniforms (if any)
- Bind your Vertex array object (for this pipeline)
- Draw the geometry
- Unbind and disable everything

Lab 2

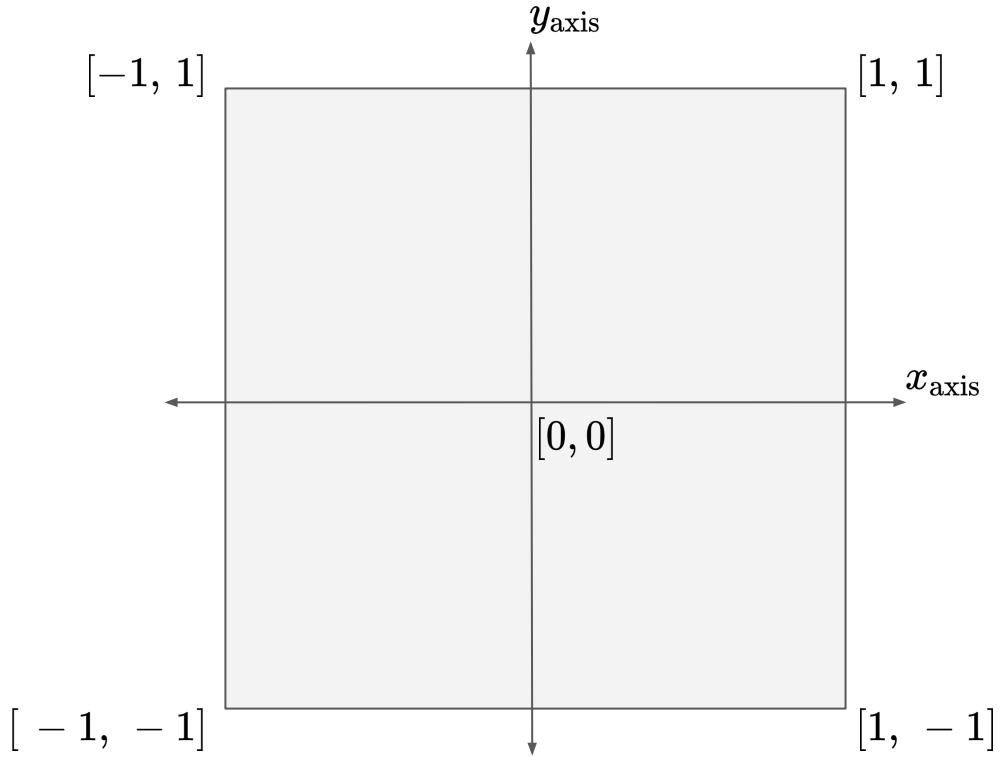
- Normalized device coordinates are from -1.0 to 1.0



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw three points (`GL_POINTS`)

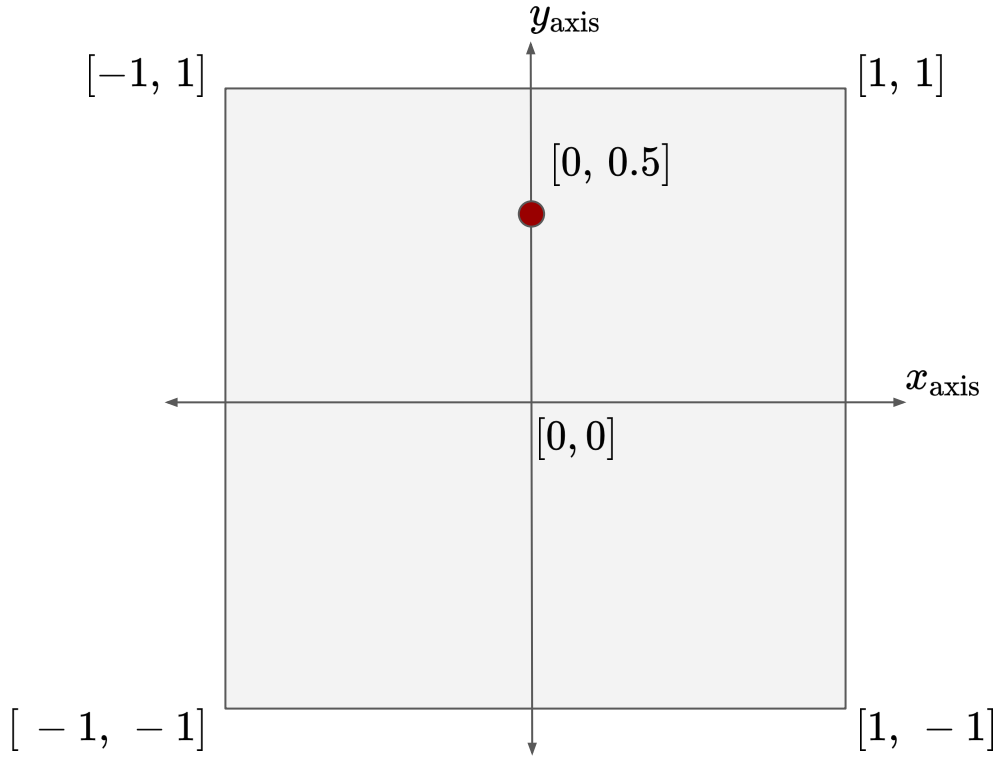
```
float vertices[] = {
  0.0,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArray(GL_POINTS, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw three points (`GL_POINTS`)

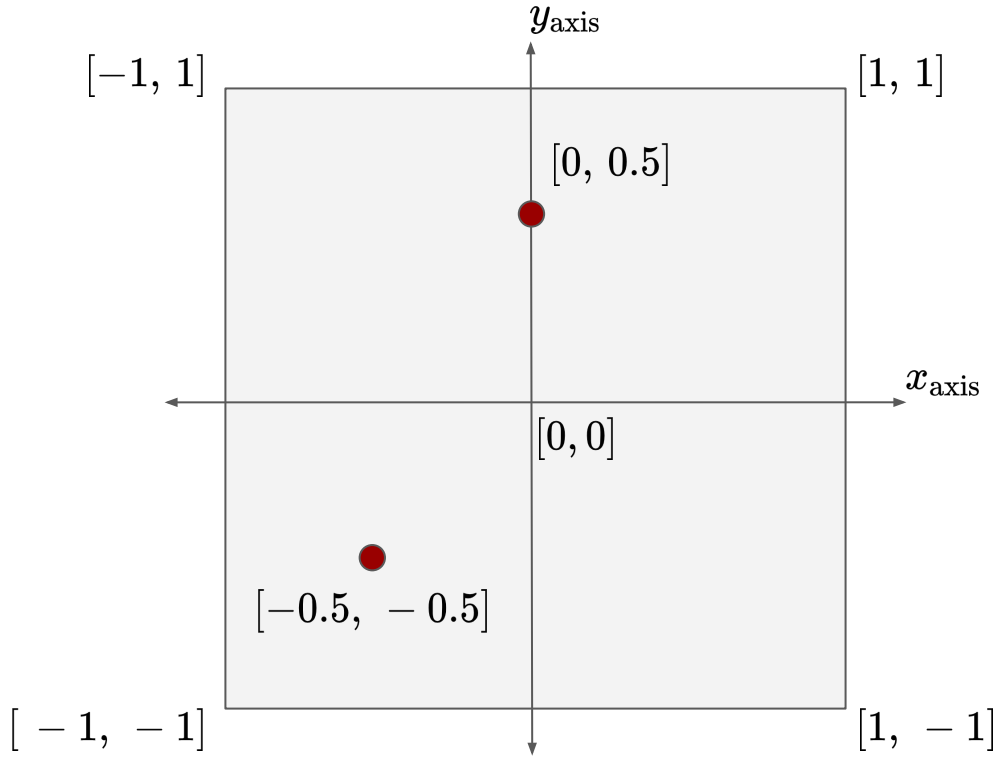
```
float vertices[] = {
  0.0,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_POINTS, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw three points (`GL_POINTS`)

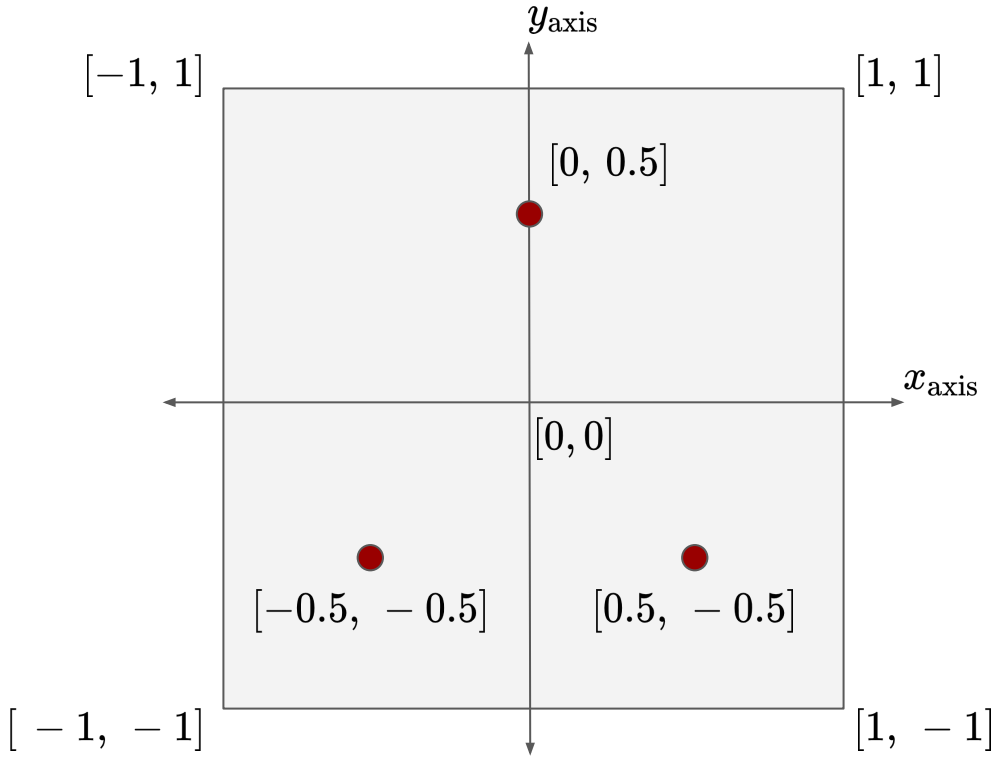
```
float vertices[] = {
  0.0,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_POINTS, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw three points (`GL_POINTS`)

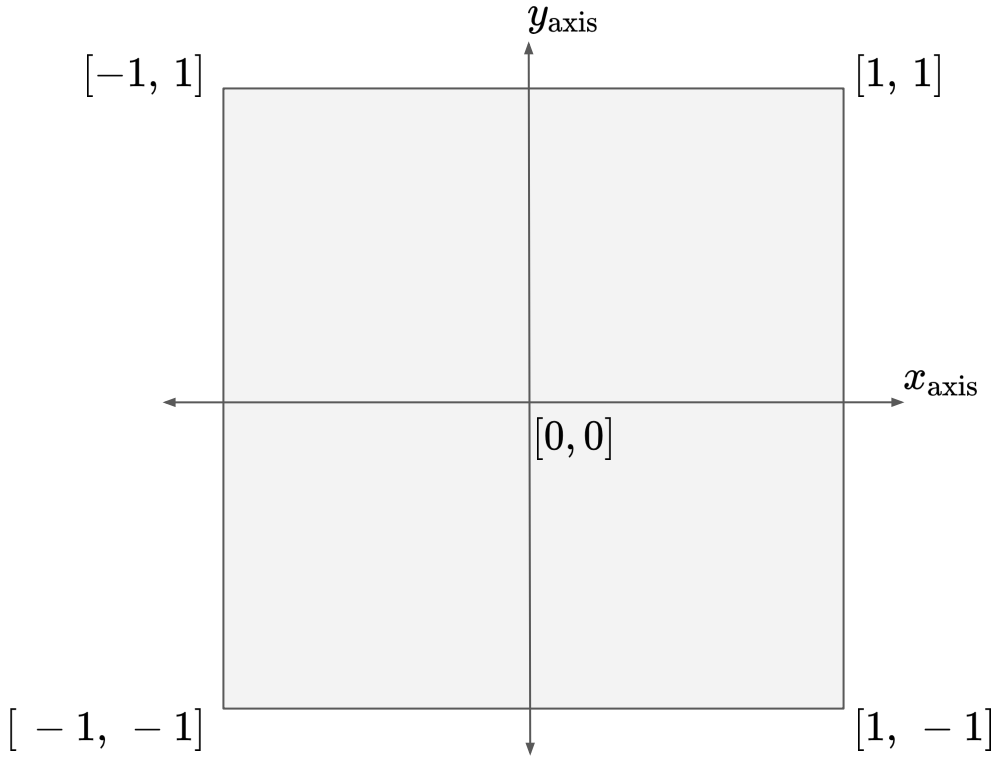
```
float vertices[] = {
  0.0,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_POINTS, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a triangle (`GL_TRIANGLES`)

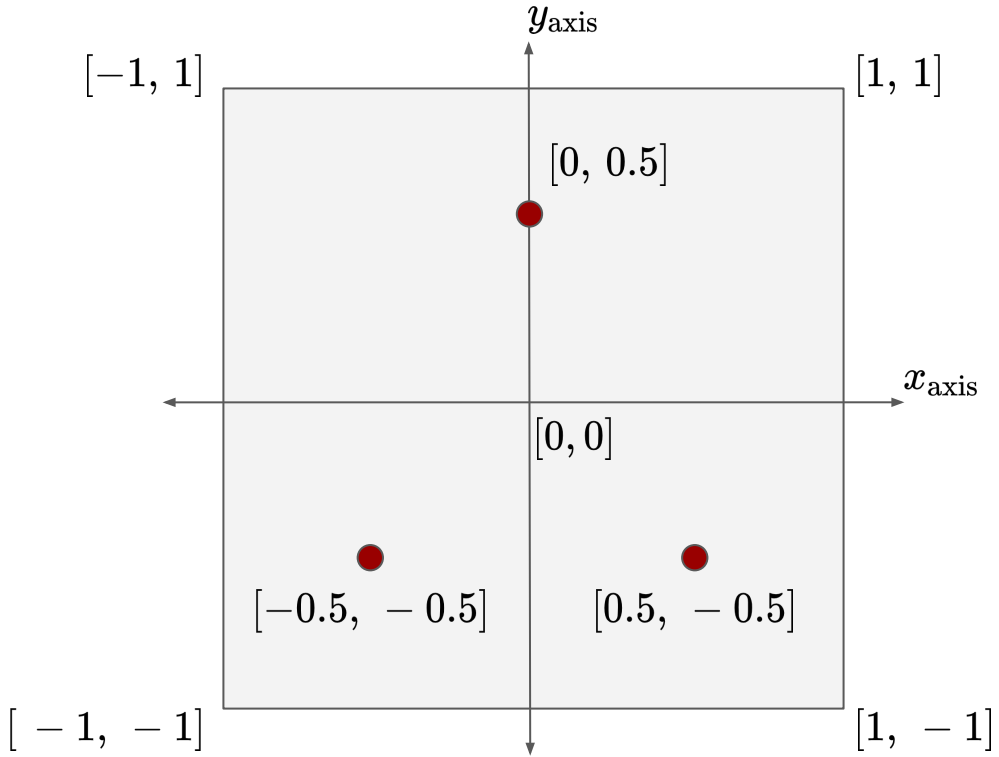
```
float vertices[] = {
  0.0,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a triangle (`GL_TRIANGLES`)

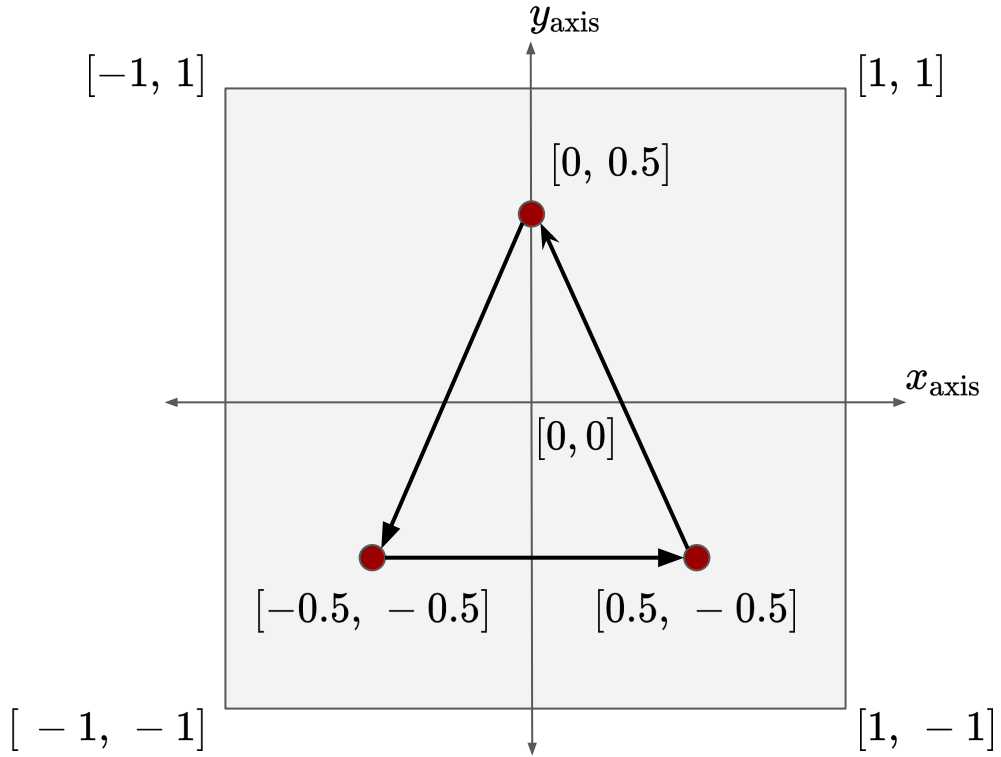
```
float vertices[] = {
 0.0,  0.5, 0.0,
-0.5, -0.5, 0.0,
 0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a triangle (`GL_TRIANGLES`)

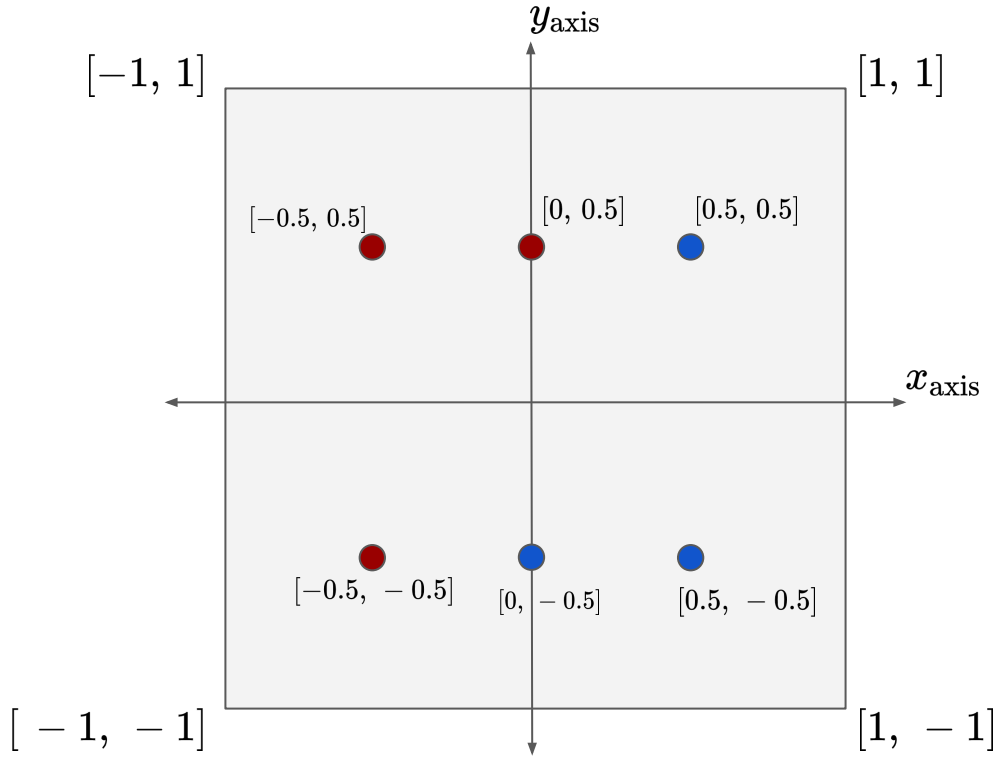
```
float vertices[] = {
 0.0,  0.5, 0.0,
-0.5, -0.5, 0.0,
 0.5, -0.5, 0.0
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices

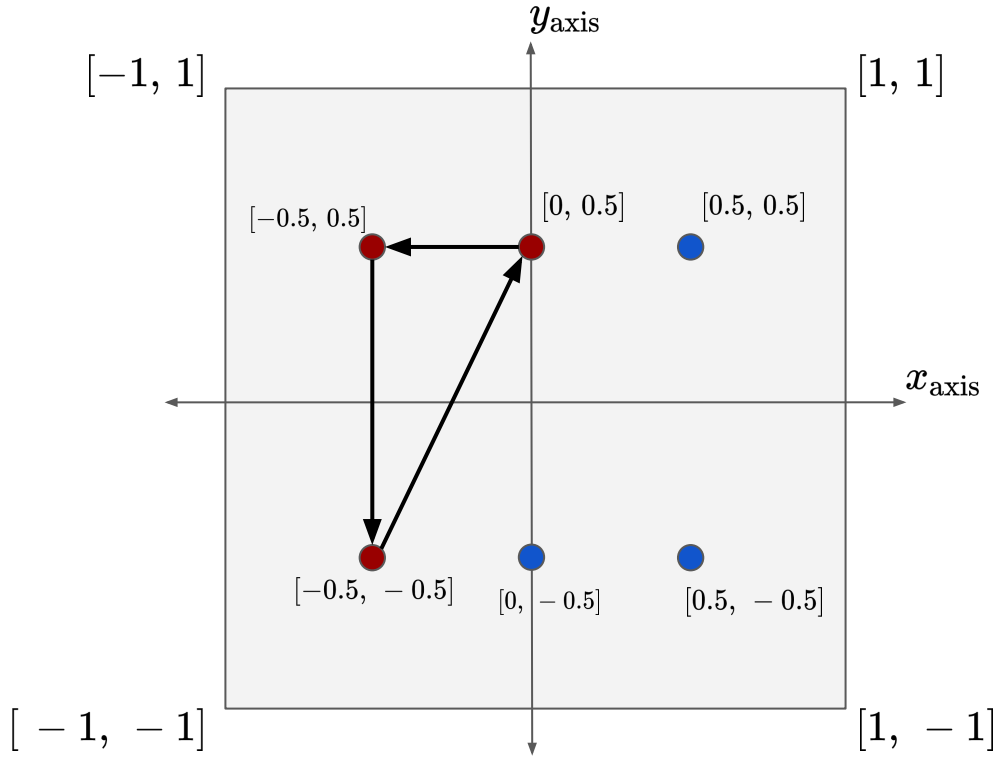
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices

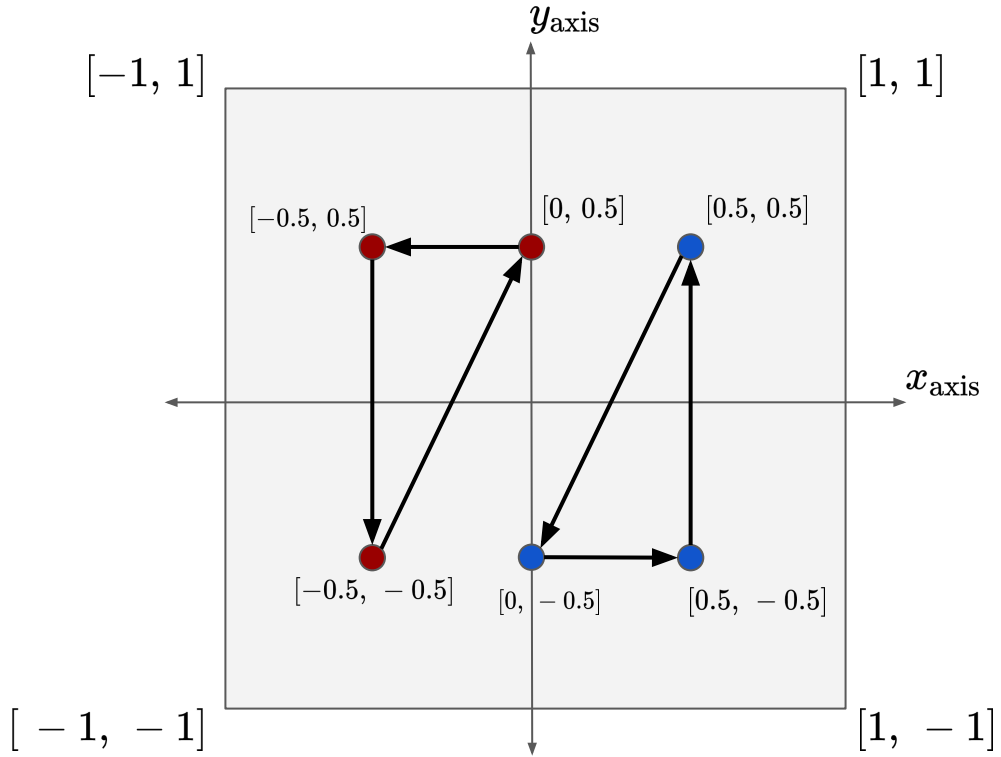
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices

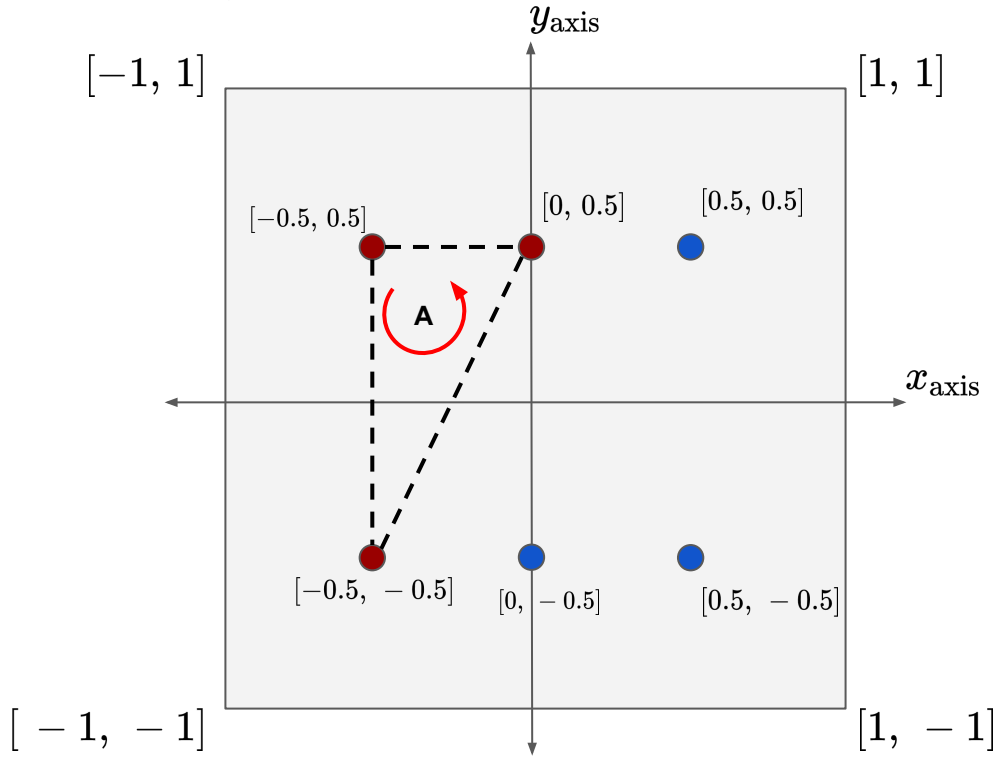
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLES, 0, 3);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices (`GL_TRIANGLE_STRIP`)

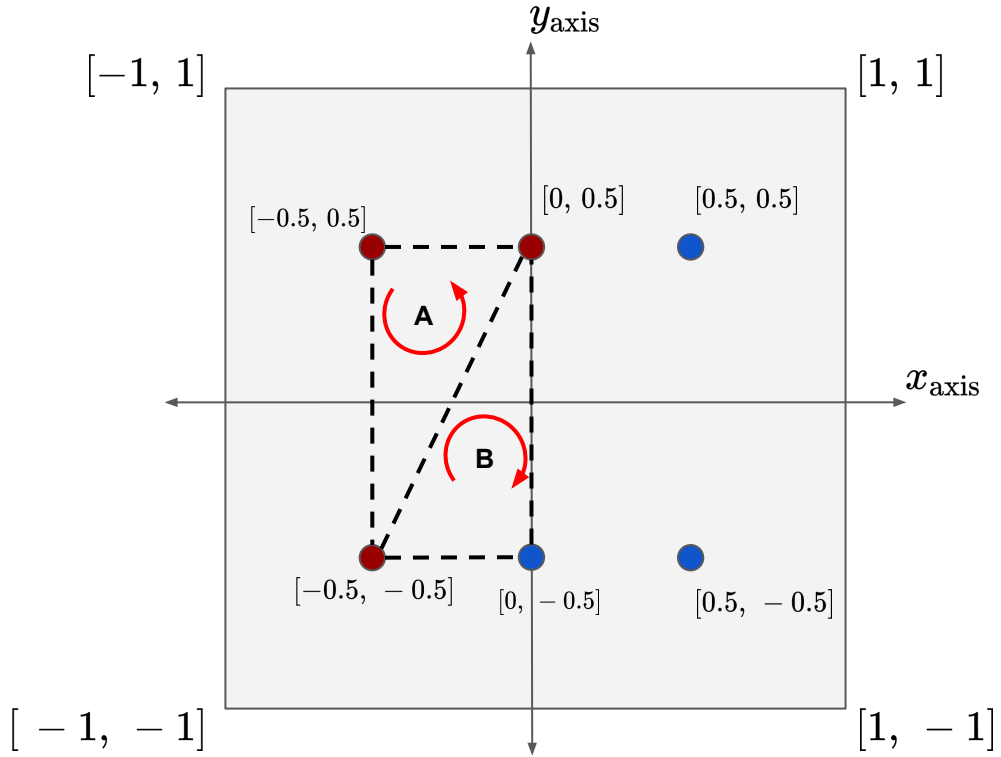
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices (`GL_TRIANGLE_STRIP`)

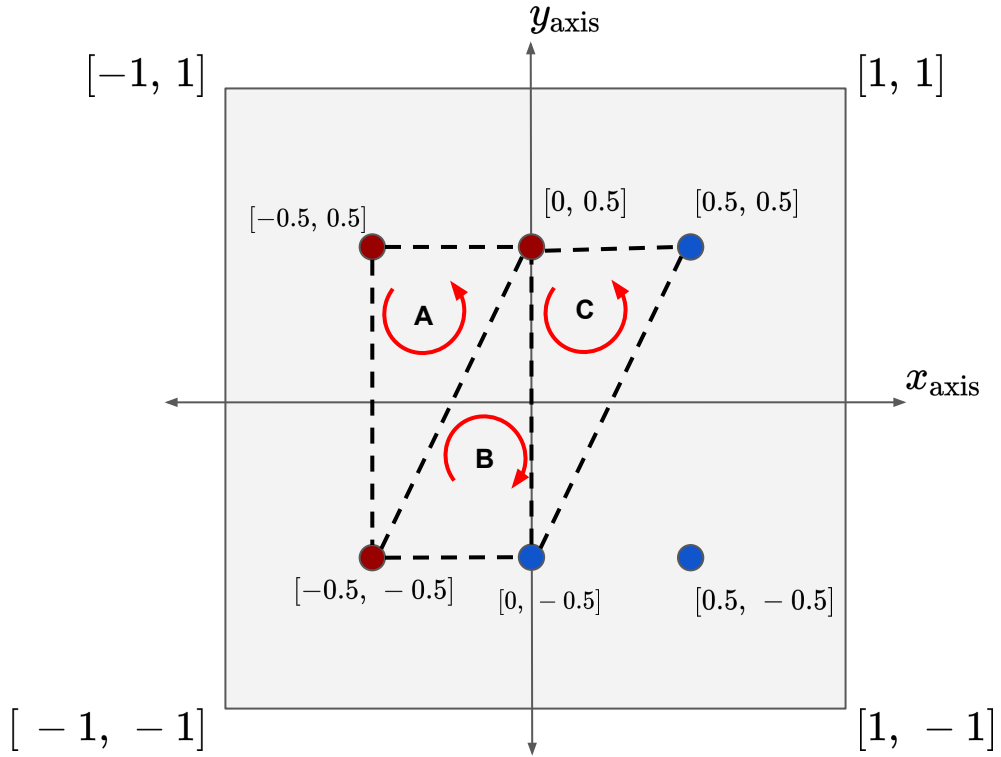
```
float vertices[] = {
 -0.5,  0.5, 0.0,
-0.5, -0.5, 0.0,
 0.0,  0.5, 0.0,
 0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices (`GL_TRIANGLE_STRIP`)

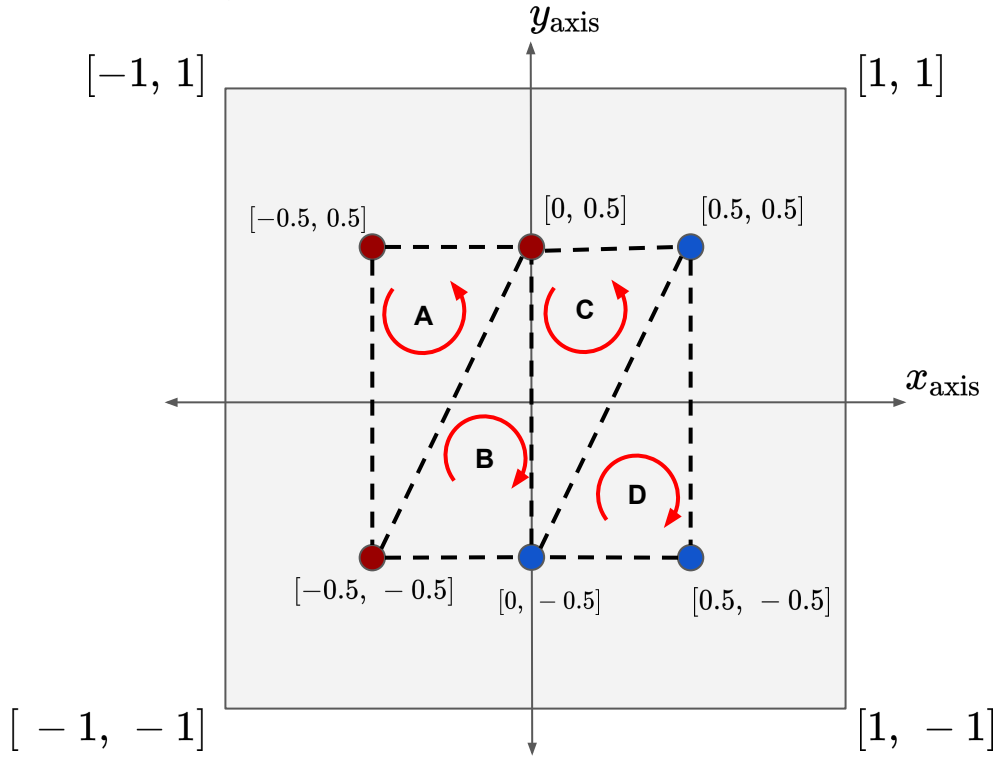
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw a rectangle with 6 vertices (`GL_TRIANGLE_STRIP`)

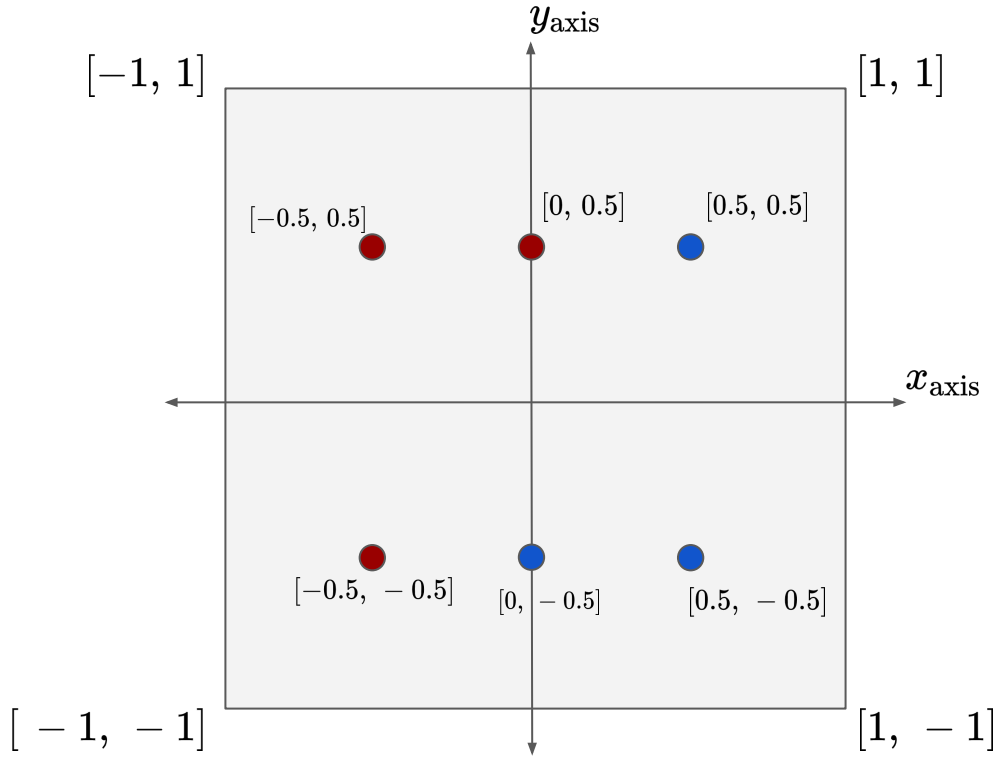
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_TRIANGLE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINES`)

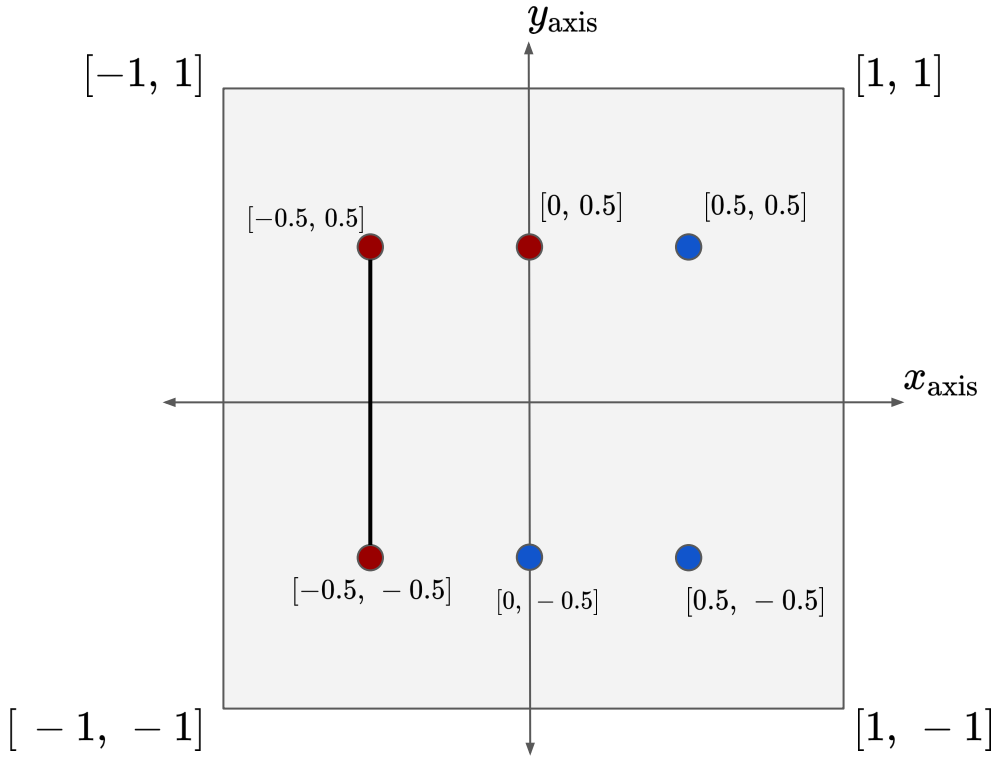
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINES, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINES`)

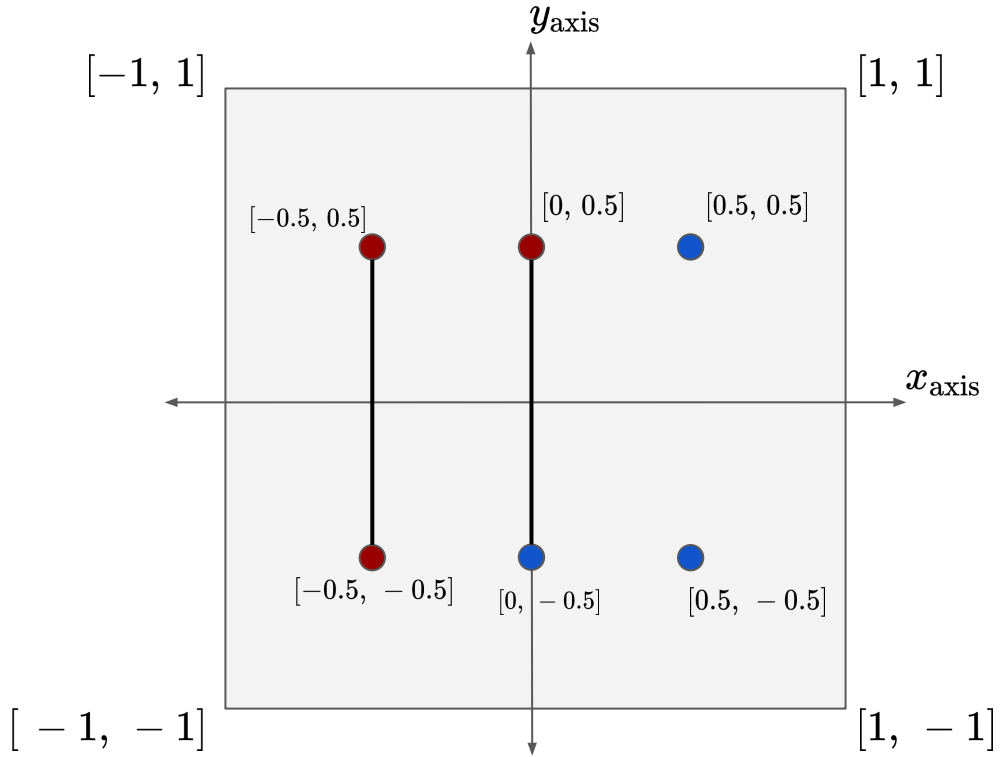
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINES, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINES`)

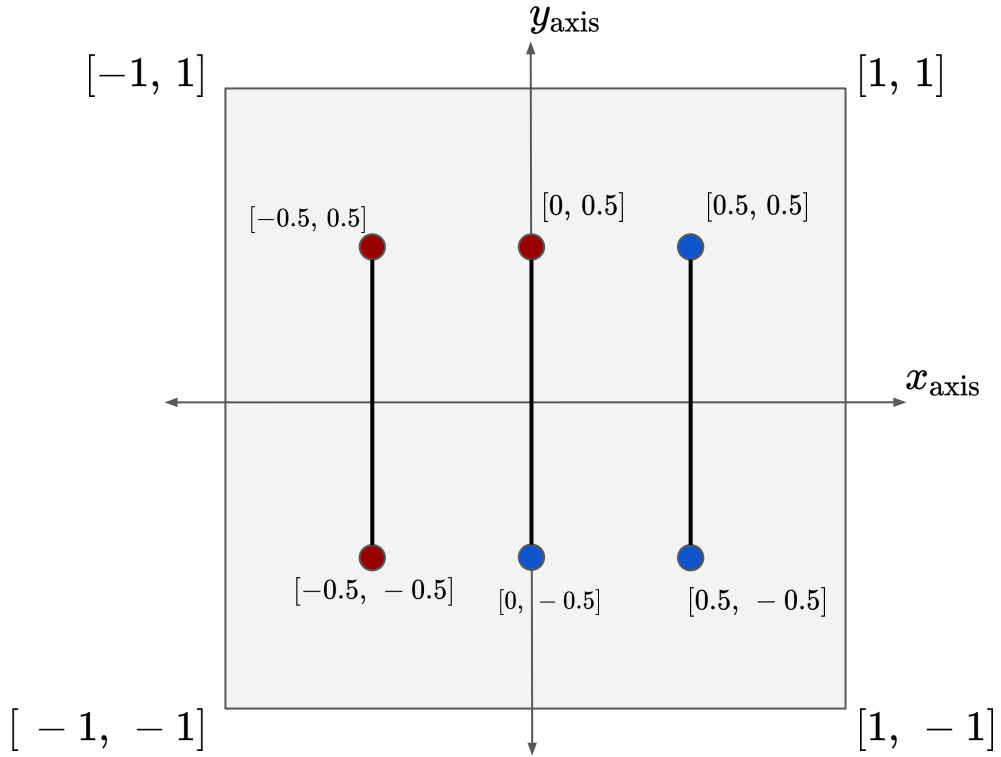
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINES, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINES`)

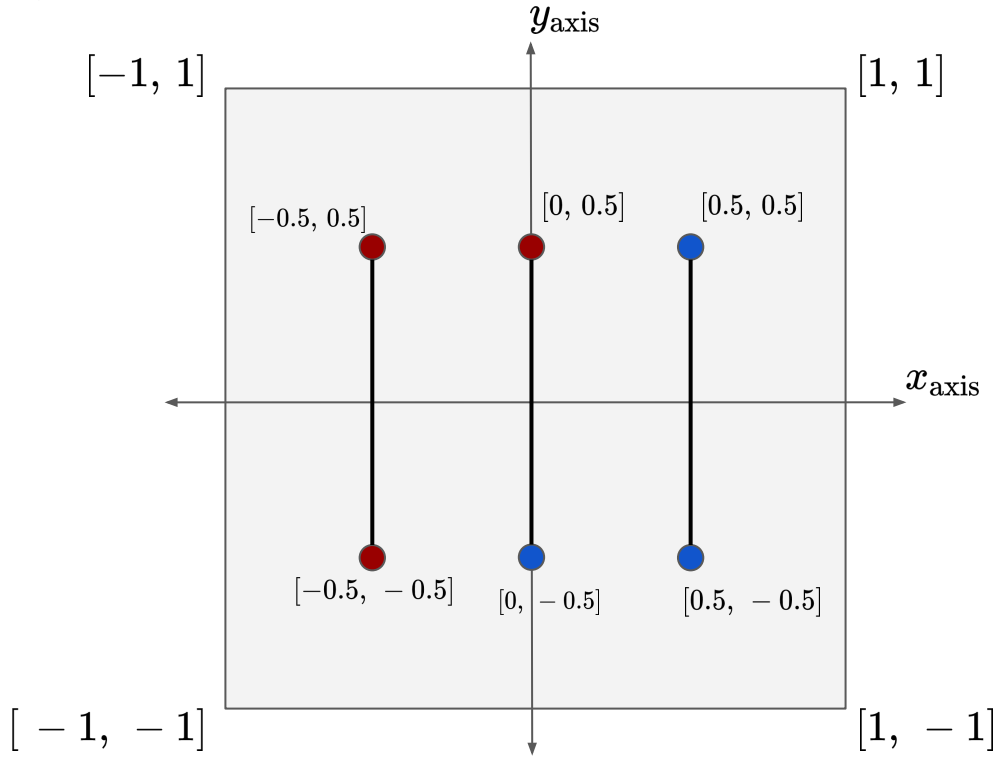
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINES, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINE_STRIP`)

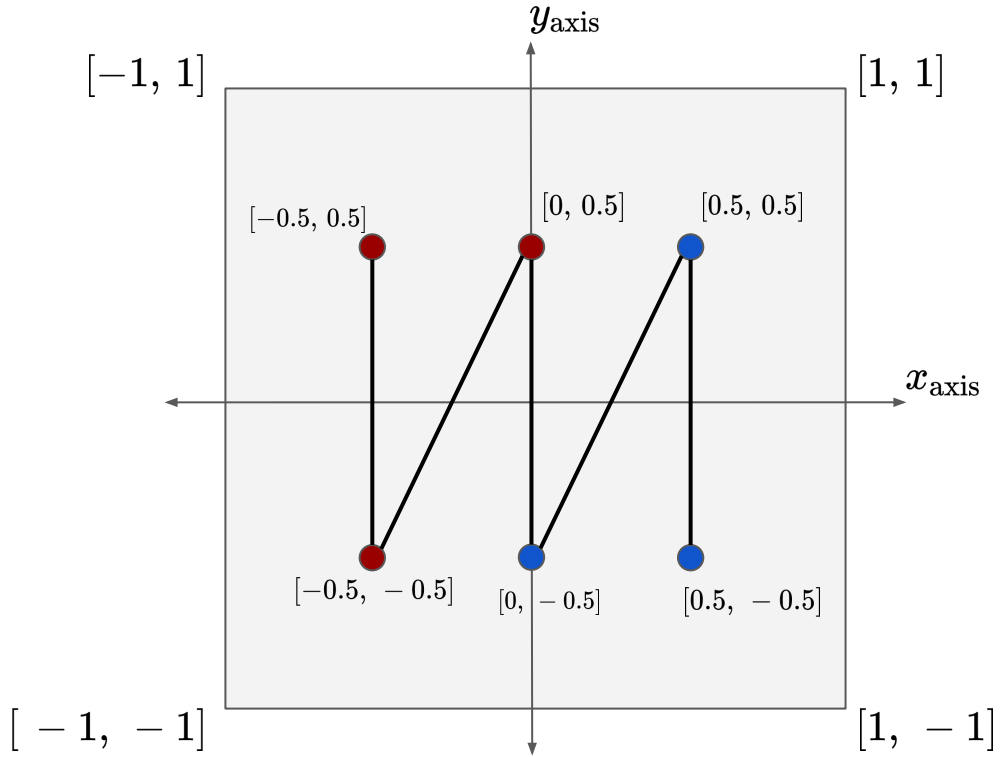
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINE_STRIP`)

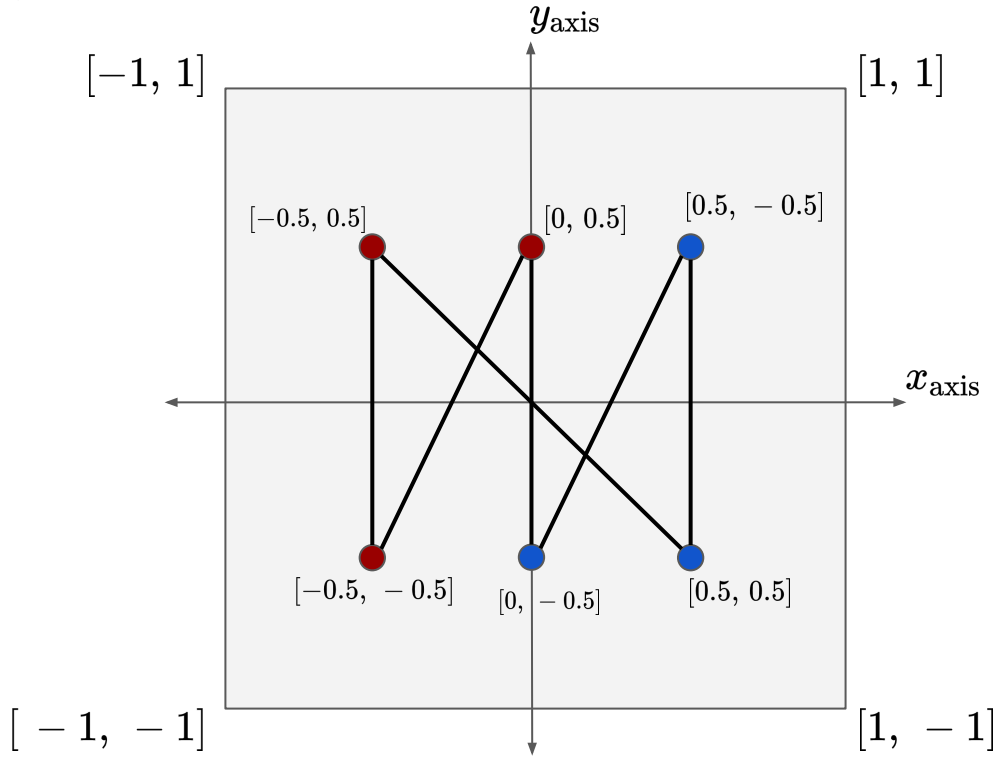
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINE_STRIP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Draw lines with 6 vertices (`GL_LINE_LOOP`)

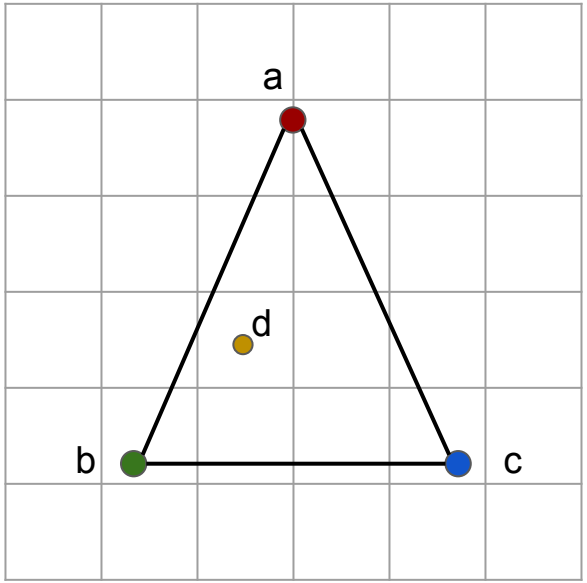
```
float vertices[] = {
 -0.5,  0.5, 0.0,
 -0.5, -0.5, 0.0,
  0.0,  0.5, 0.0,
  0.0, -0.5, 0.0,
  0.5,  0.5, 0.0,
  0.0, -0.5, 0.0,
};

//[..] bind :
   • framebuffer
   • program
   • vao

glDrawArrays(GL_LINE_LOOP, 0, 6);

// [..] unbind everything
```



Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Interpolate triangle

$[-1, 1]$ $[1, 1]$



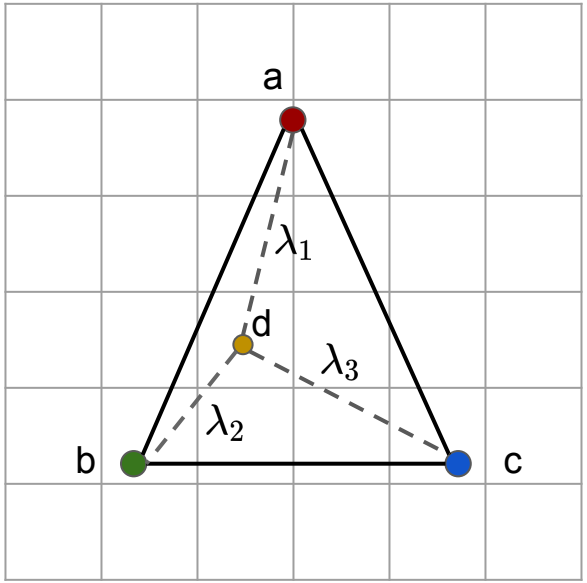
$[-1, -1]$ $[1, -1]$

Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Interpolate triangle

$[-1,\ 1]$ $[1,\ 1]$

$$d = \lambda_1 a + \lambda_2 b + \lambda_3 c \quad \lambda_1, \lambda_2, \lambda_3 \in [0, 1]$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$



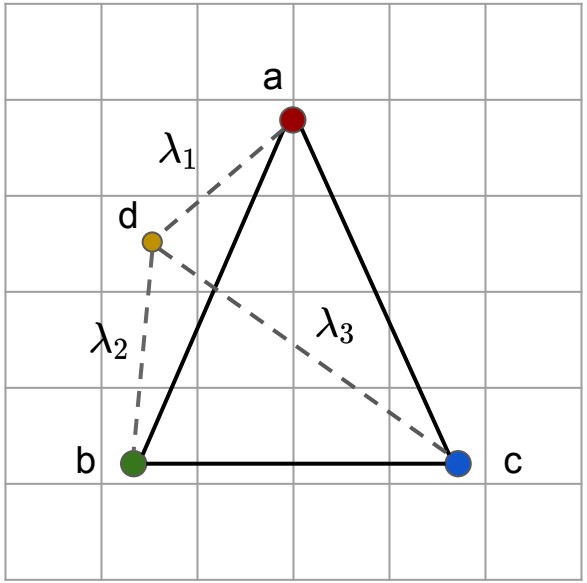
$[-1,\ -1]$ $[1,\ -1]$

Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Interpolate triangle

$[-1,\ 1]$ $[1,\ 1]$

$$d = \lambda_1 a + \lambda_2 b + \lambda_3 c \quad \lambda_1, \lambda_2, \lambda_3 \in [0, 1]$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

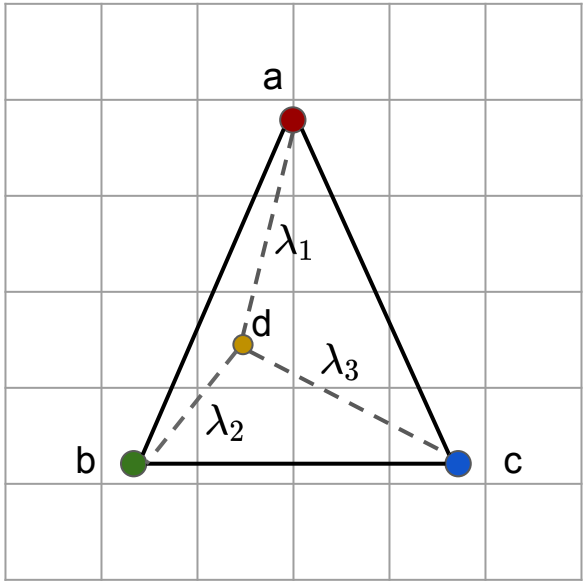


$[-1,\ -1]$ $[1,\ -1]$

Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Interpolate triangle

$[-1,\ 1]$ $[1,\ 1]$



$[-1,\ -1]$ $[1,\ -1]$

$$d = \lambda_1 a + \lambda_2 b + \lambda_3 c \quad \lambda_1, \lambda_2, \lambda_3 \in [0, 1]$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

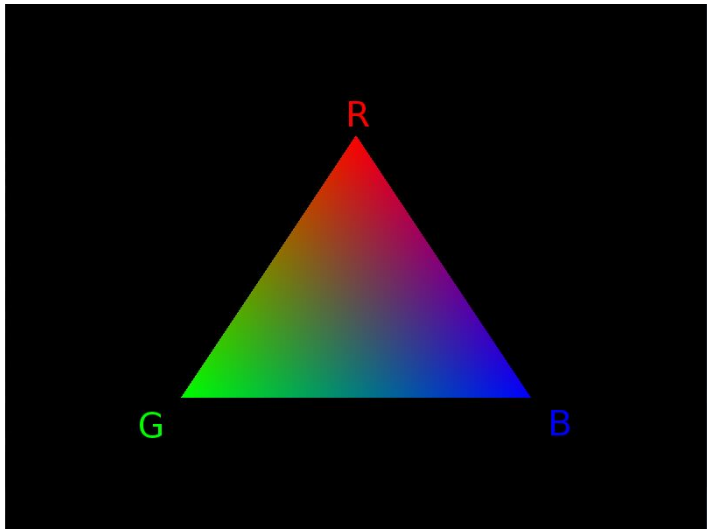
For each pixel sample we evaluate the following:

$$d = \lambda_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \lambda_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Lab 2

- Normalized device coordinates are from -1.0 to 1.0
- Interpolate triangle
- For each pixel sample we evaluate the following:

$$d = \lambda_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \lambda_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



Extra tasks

- Create two single triangles (One red and one green)
- Create more shapes
- Maybe move triangles based on button events
- Apply some post process operations such as blending/scissor/stencil