3D Graphics programming

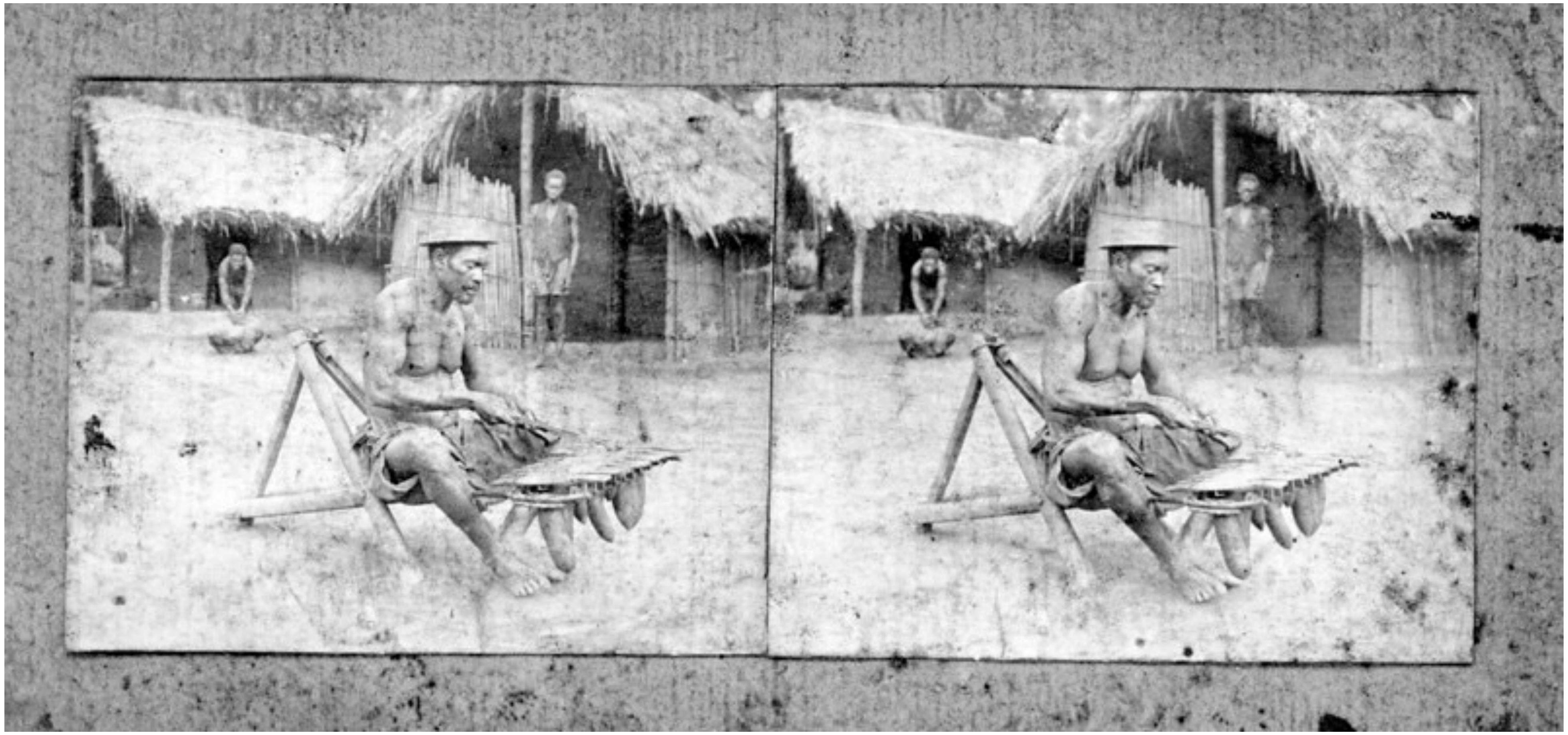
with C++ and OpenGL

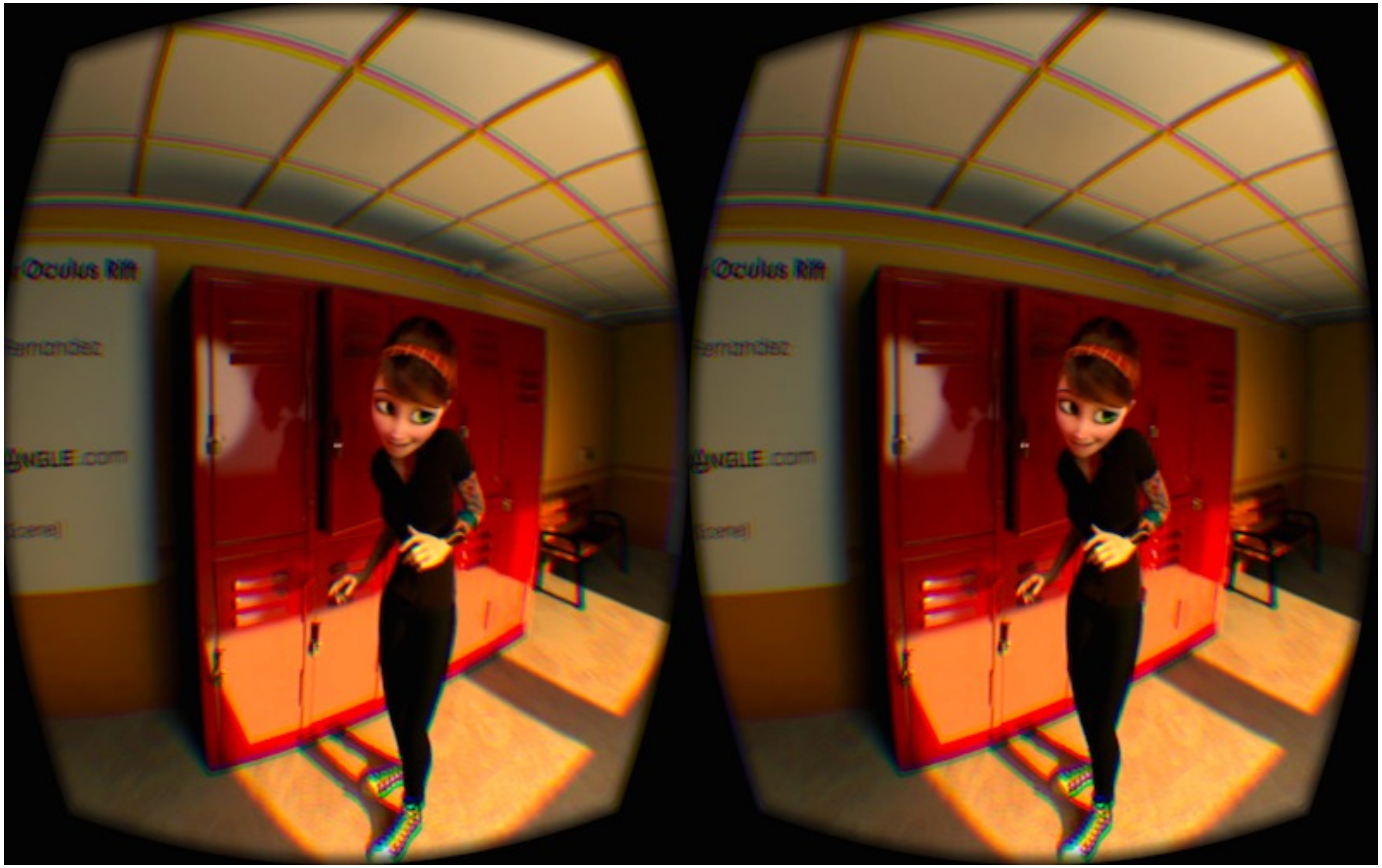
How do we create a 3D impression ?

Depth cues

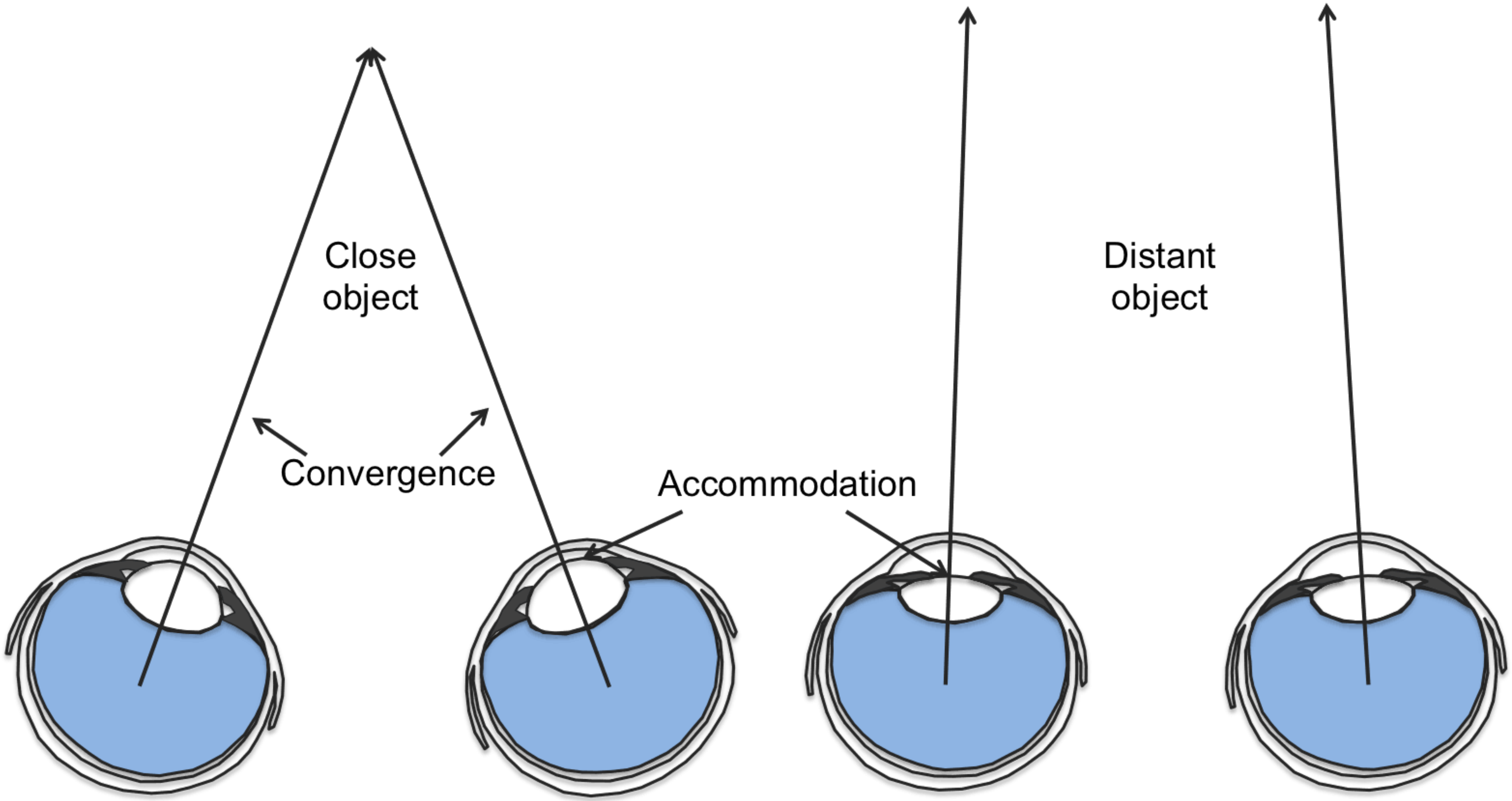
Binocular cues

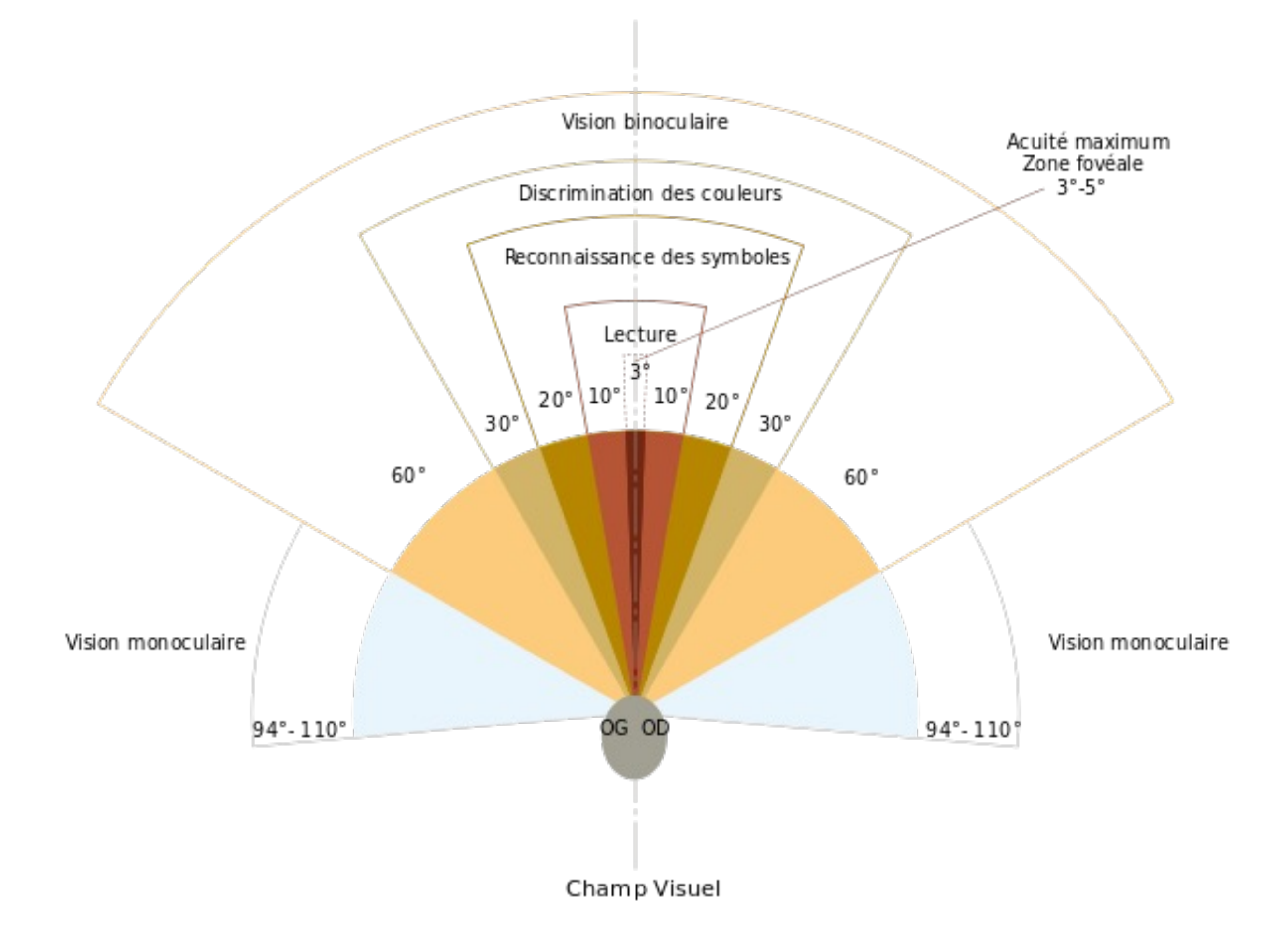
Retinal disparity

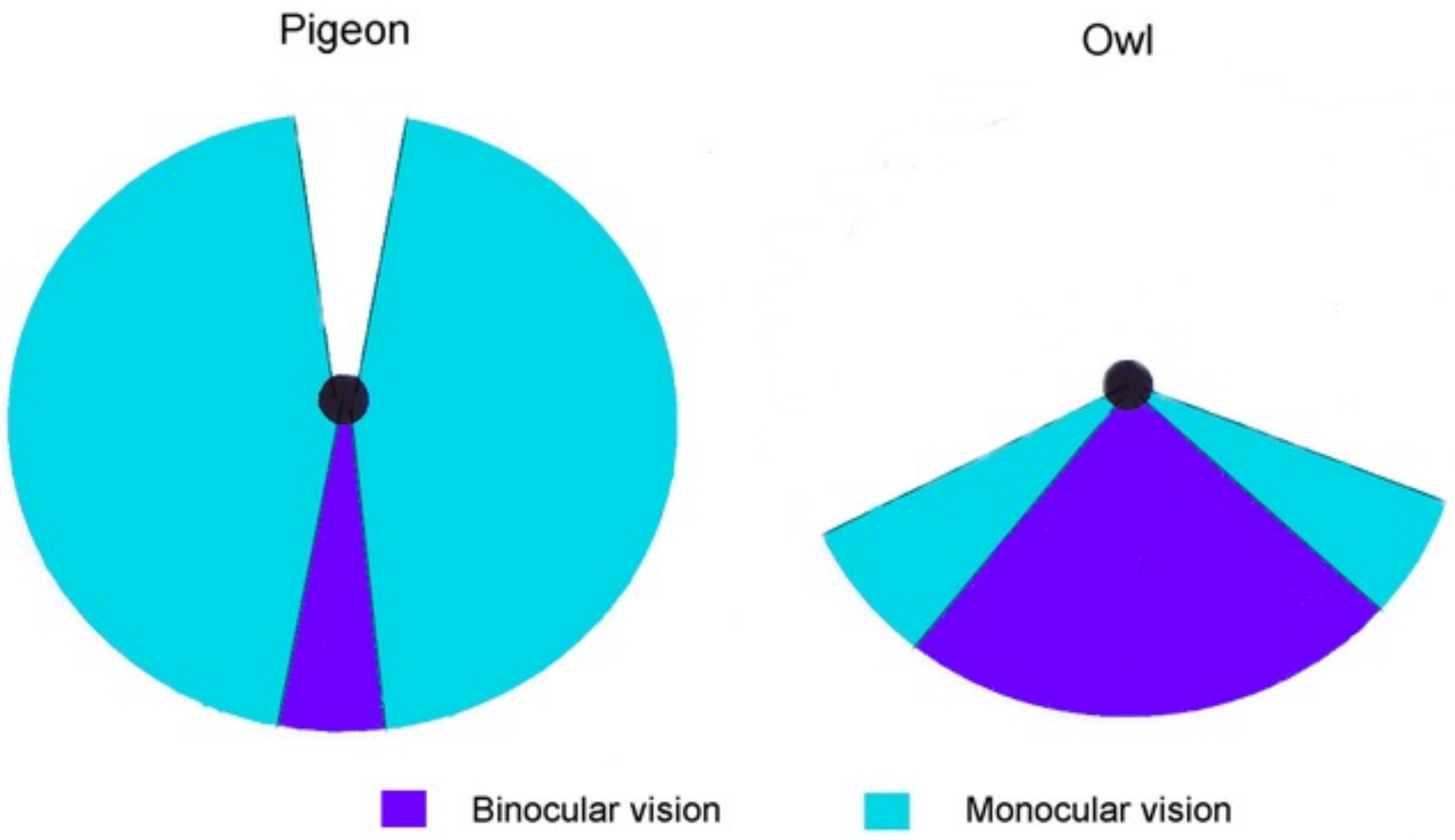



Oculus Rift
Fernandez
NGLE.com
Scene)

Convergence

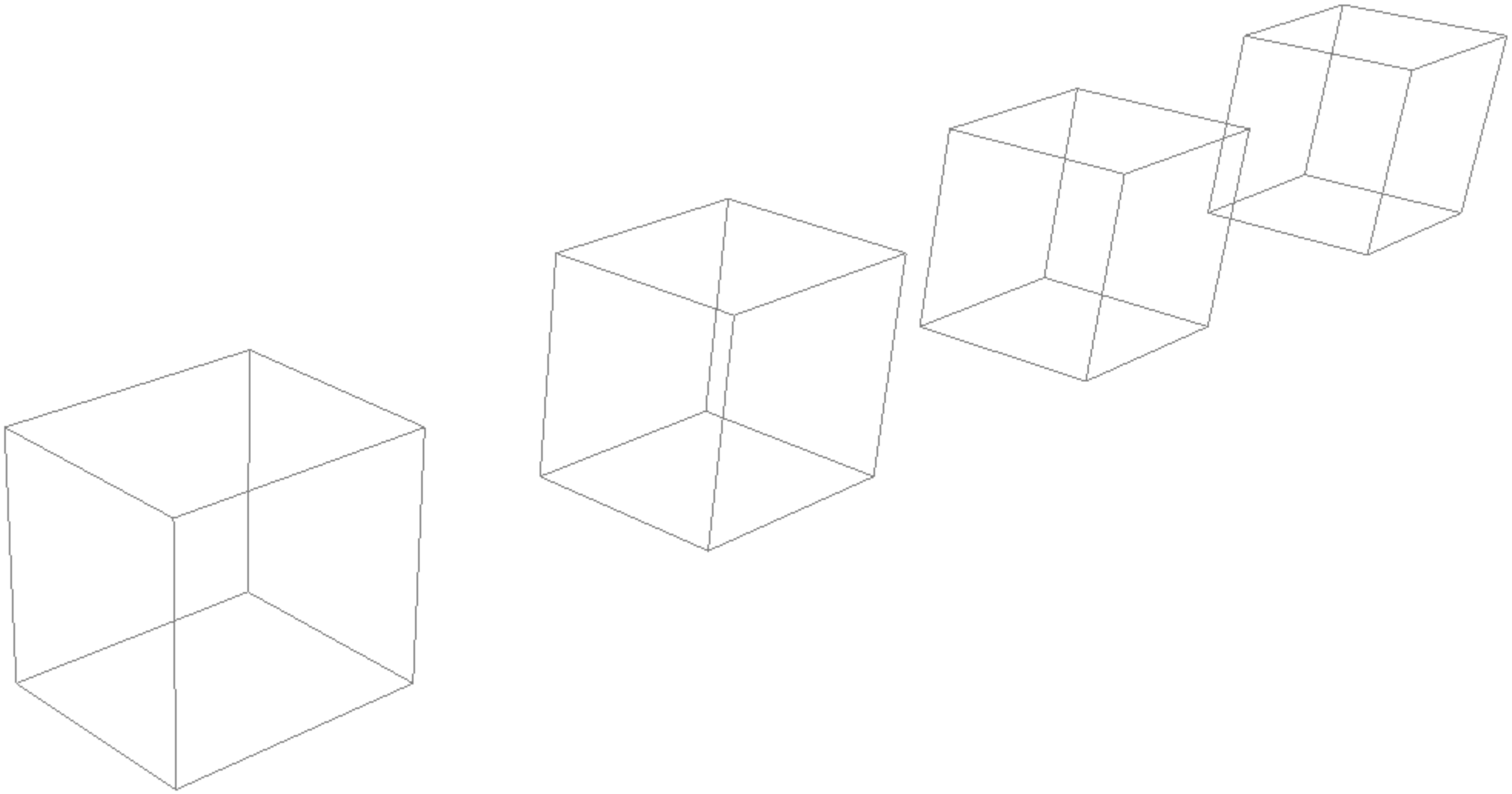



Vision binoculaire
Acuité maximum
Zone fovéale
3°-5°
Discrimination des couleurs
Reconnaissance des symboles
Lecture
3°
20°
10°
10°
20°
30°
30°
60°
60°
Vision monoculaire
Vision monoculaire
94°- 110°
OG
OD
94°- 110°
Champ Visuel

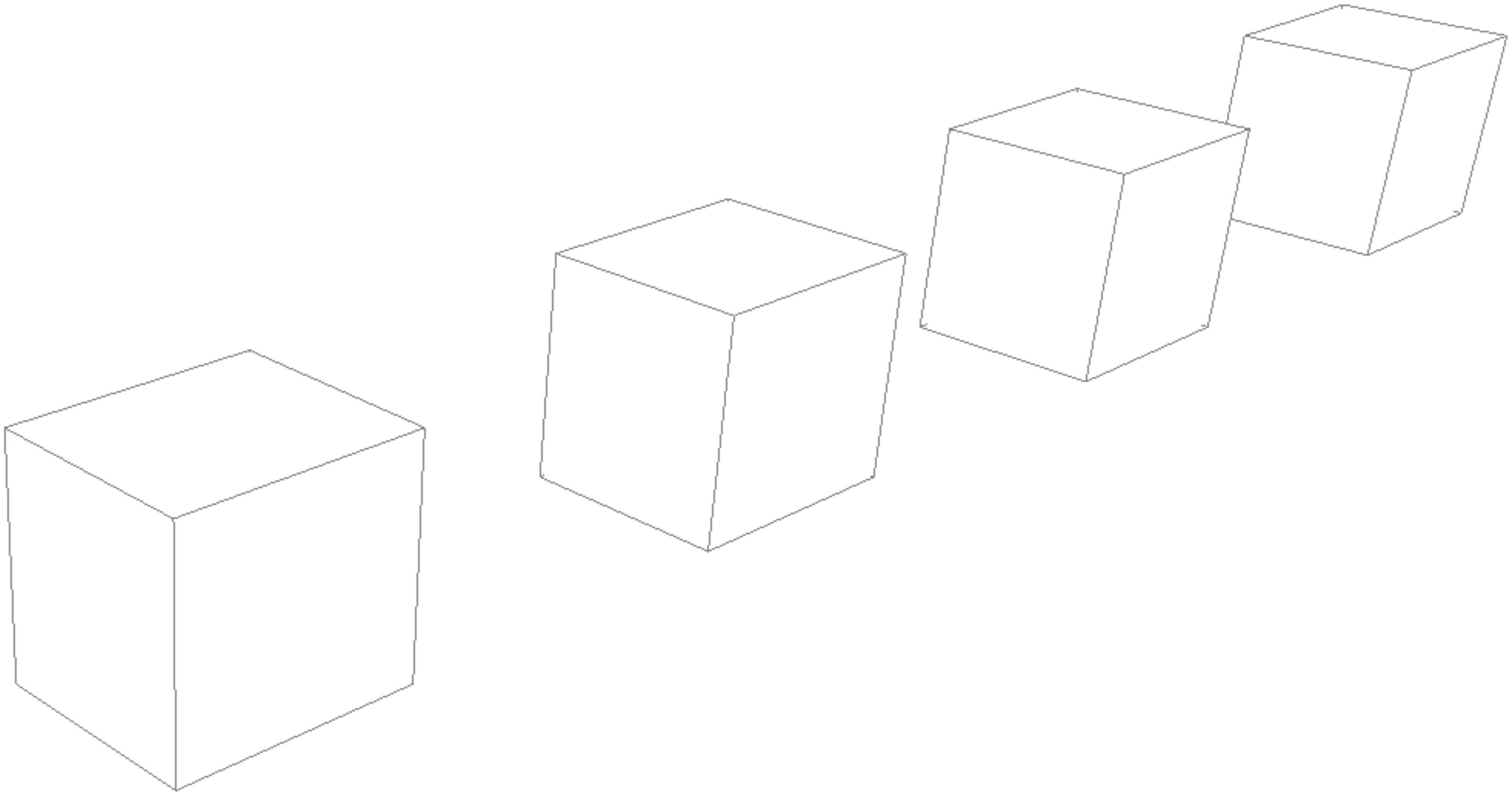

Pigeon
Owl
Binocular vision
Monocular vision

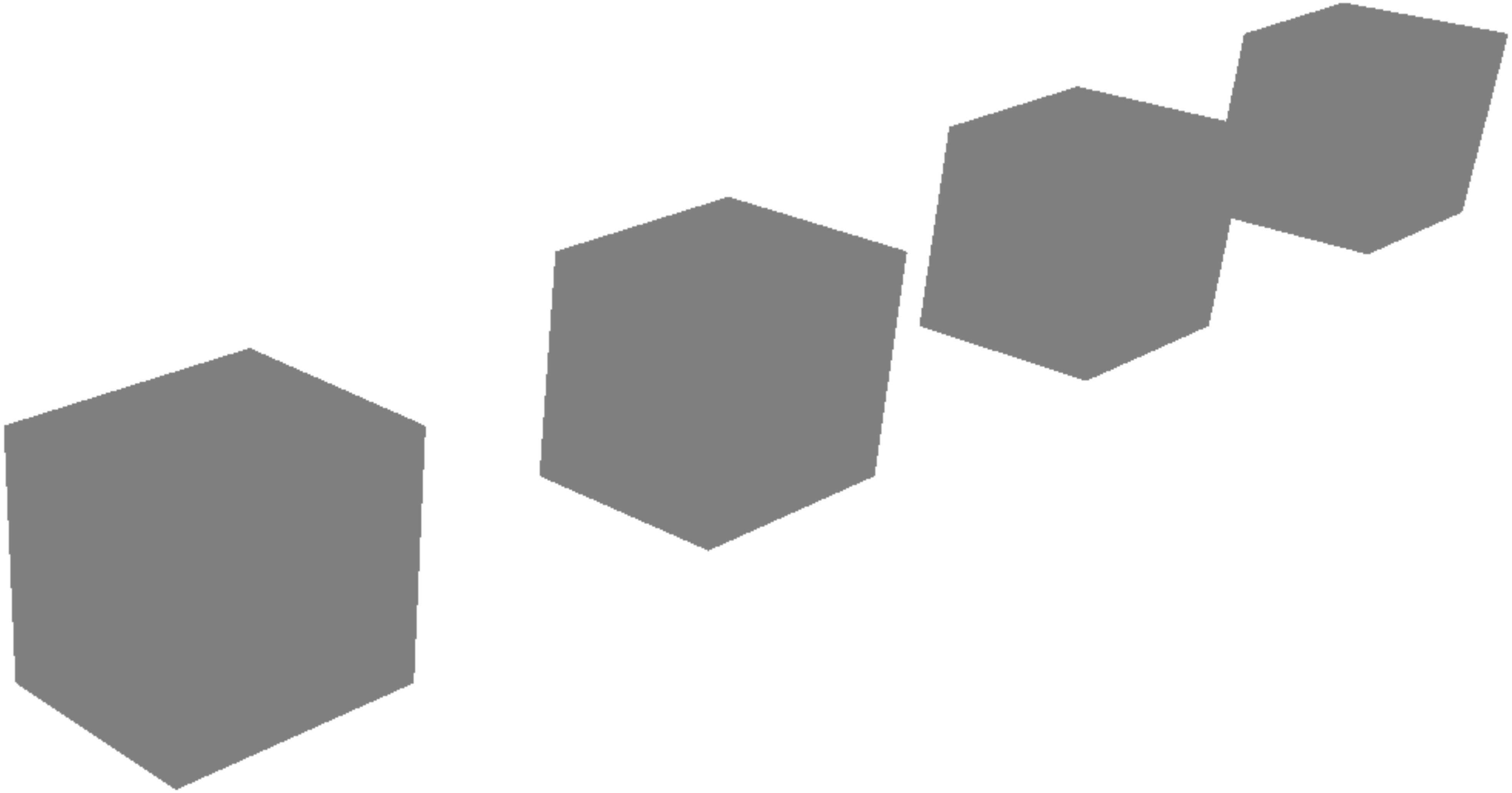
Monocular cues

Perspective

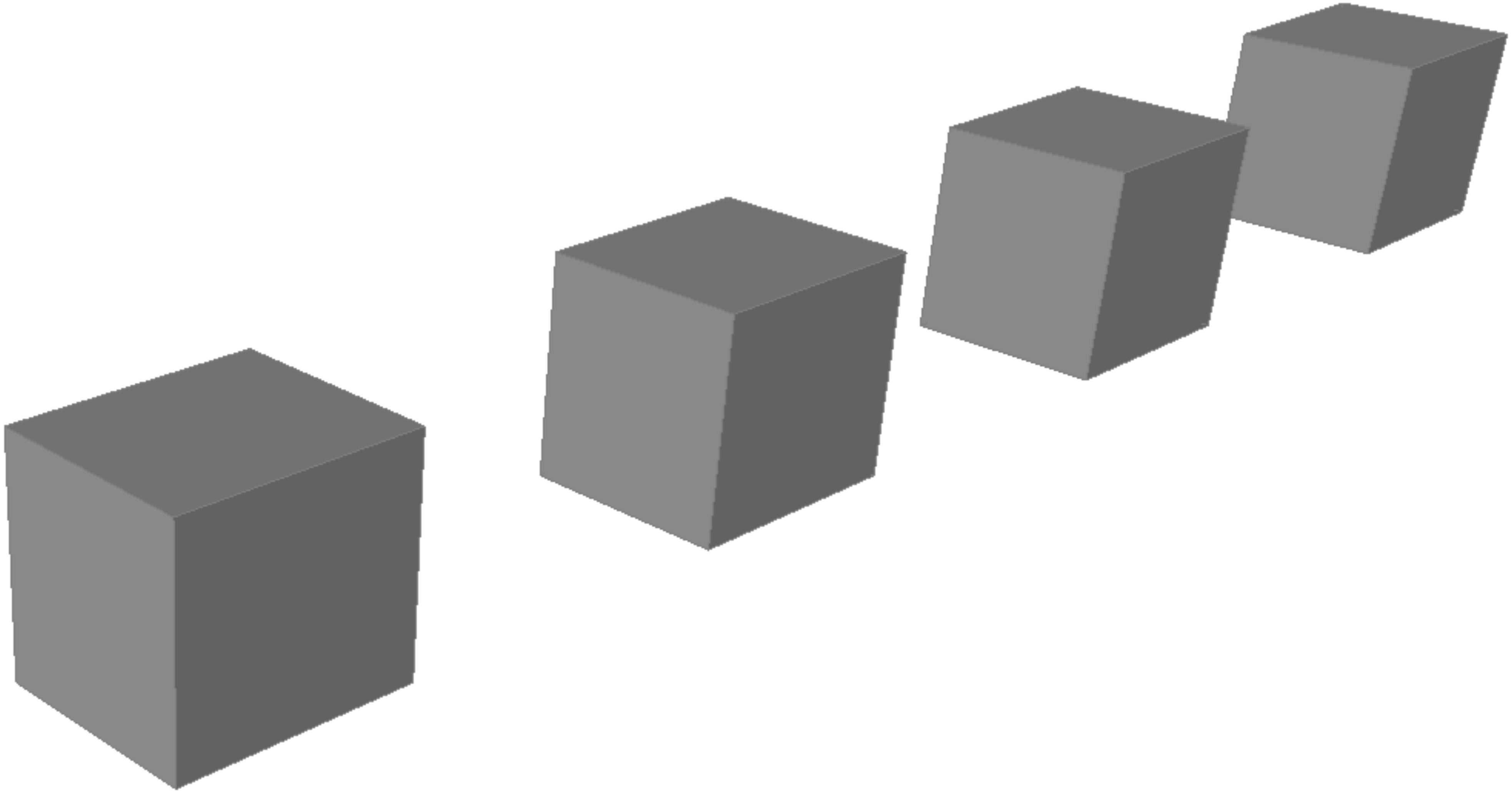


Occlusion





Lightning/shading

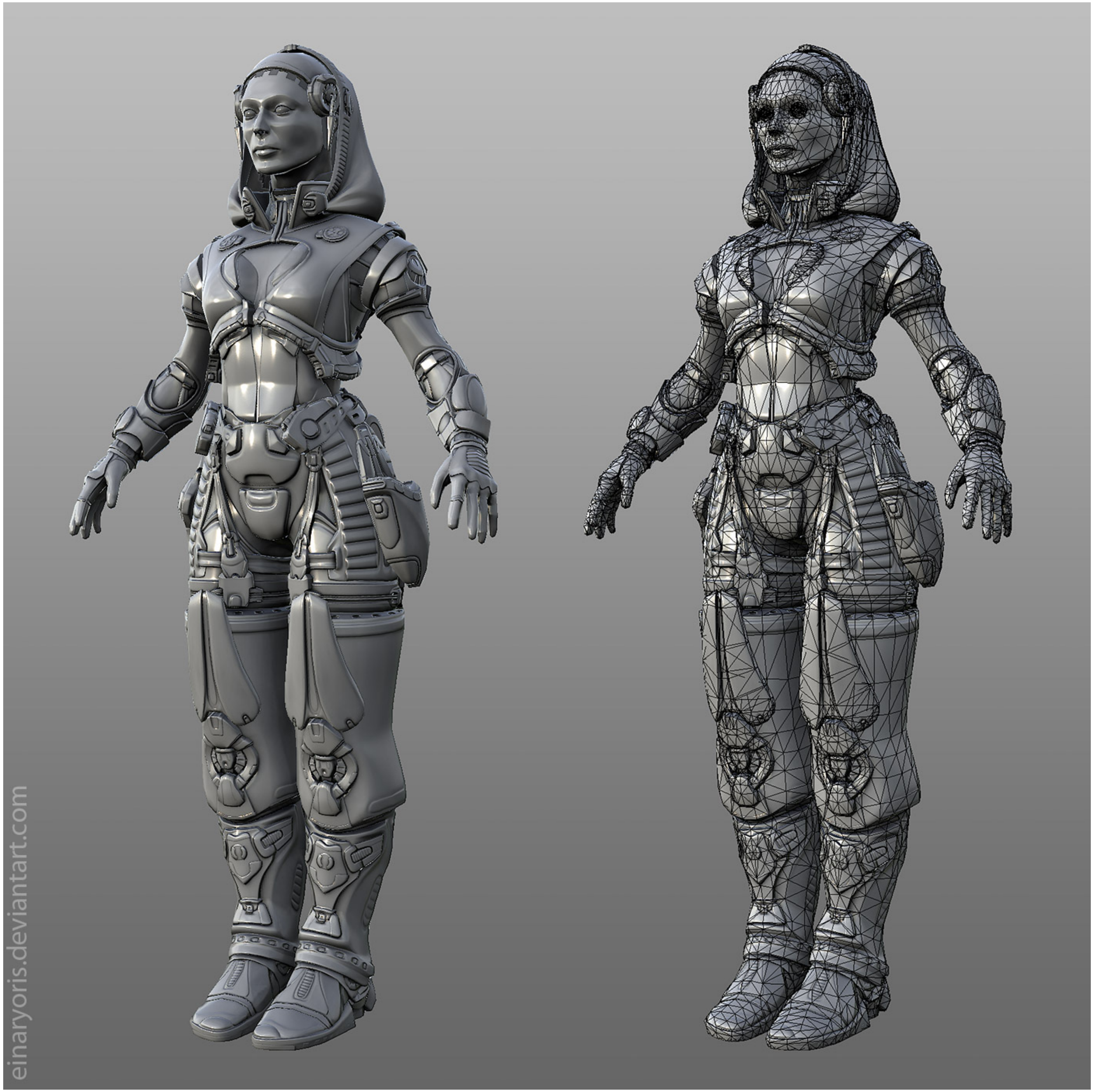


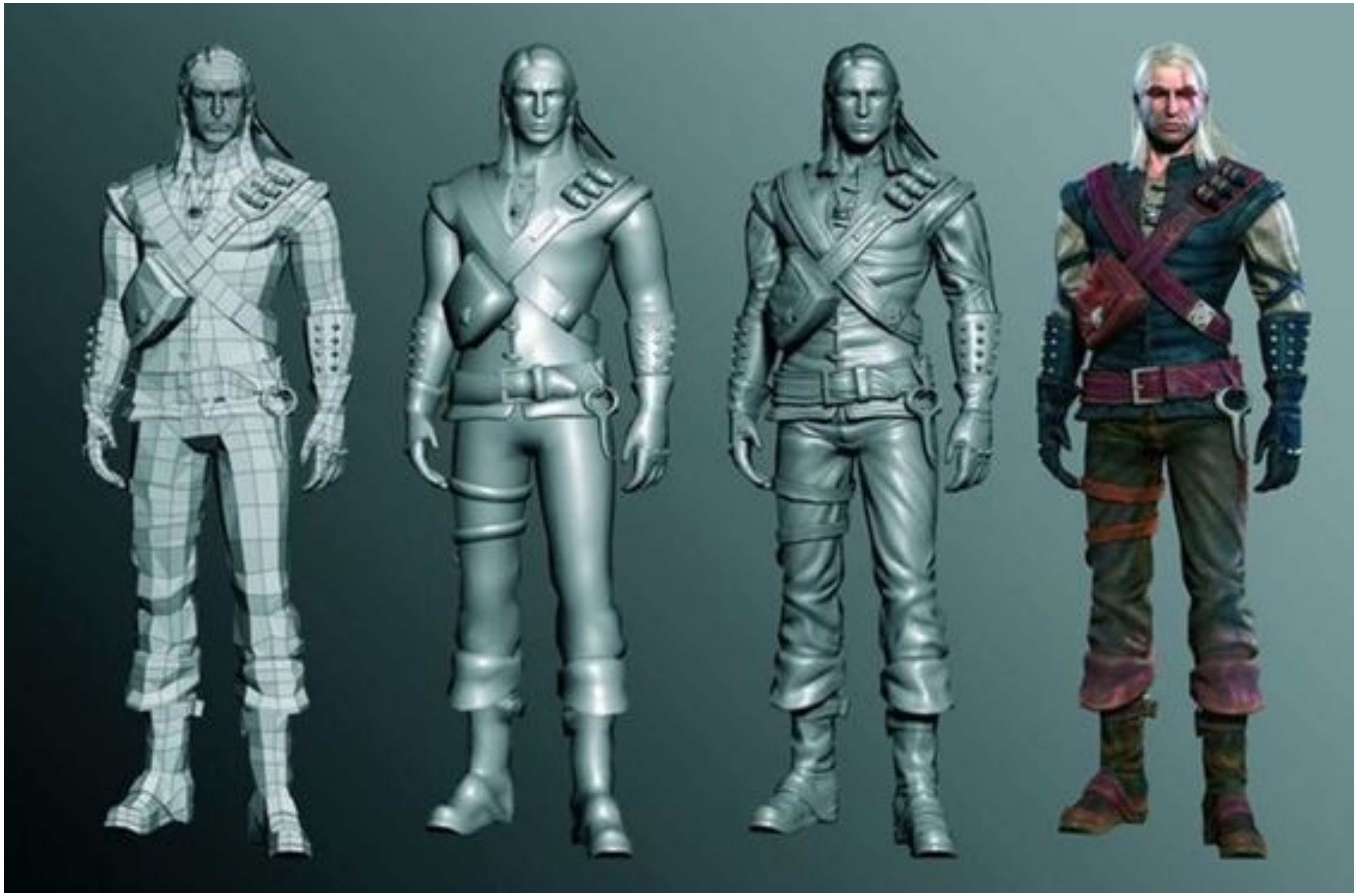
Paralax

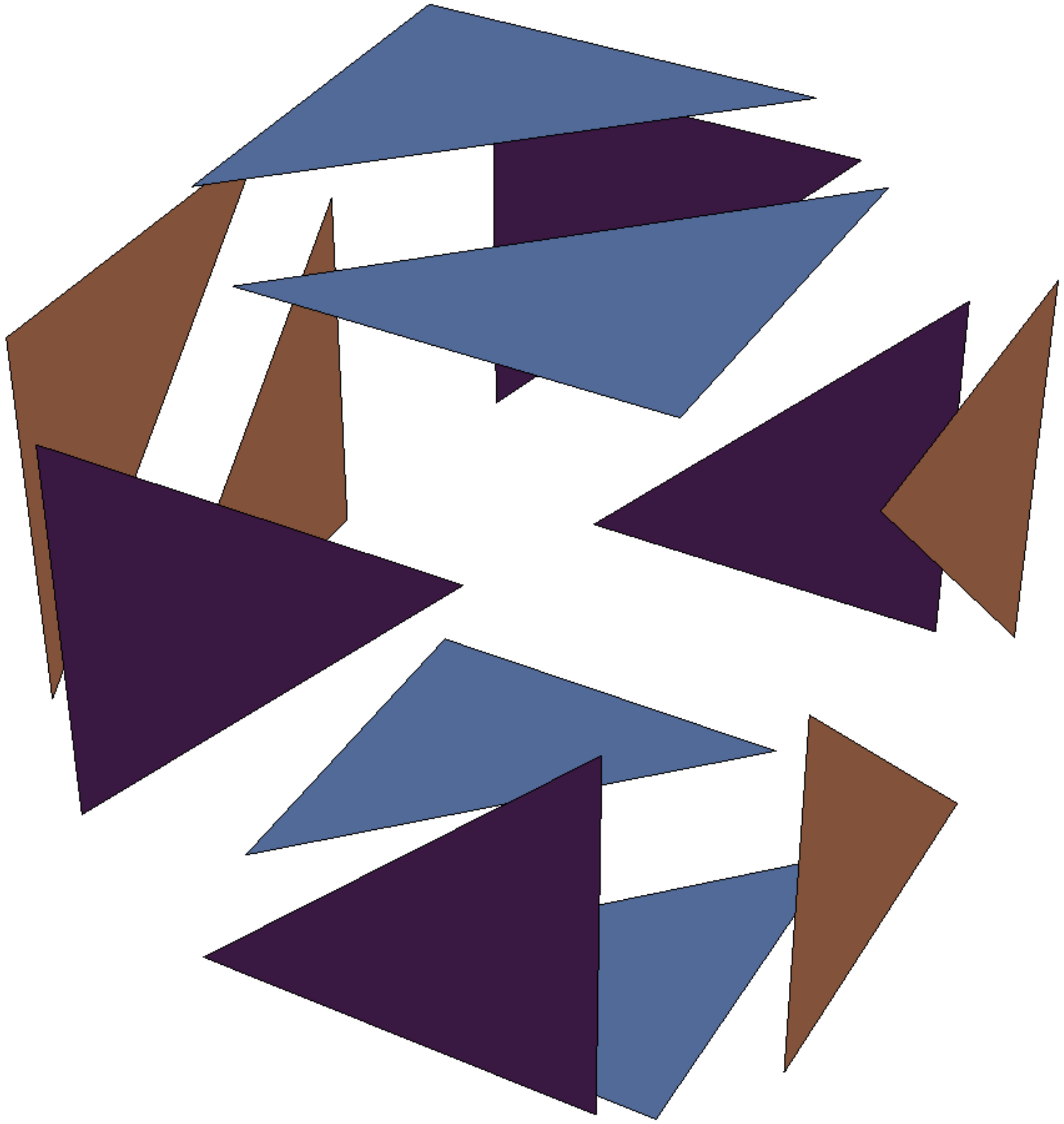
We will be dealing with

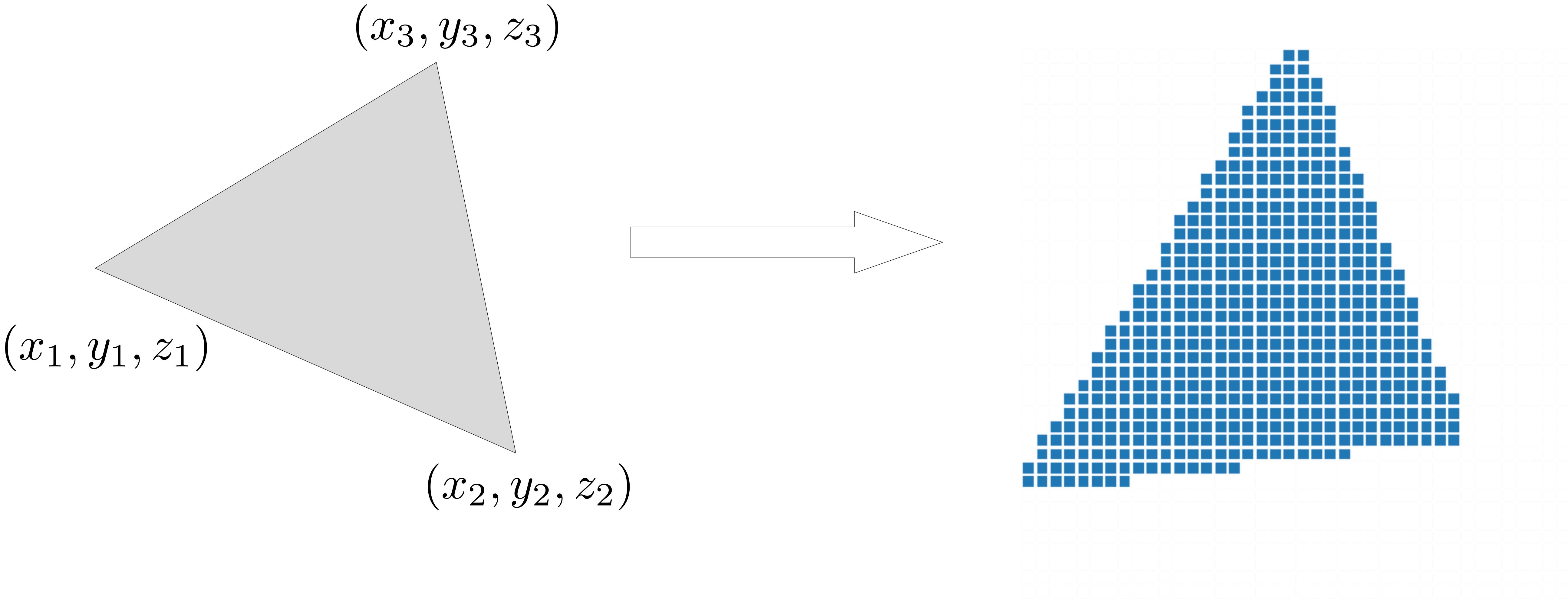
- Perspective
- Occlusion
- Shading

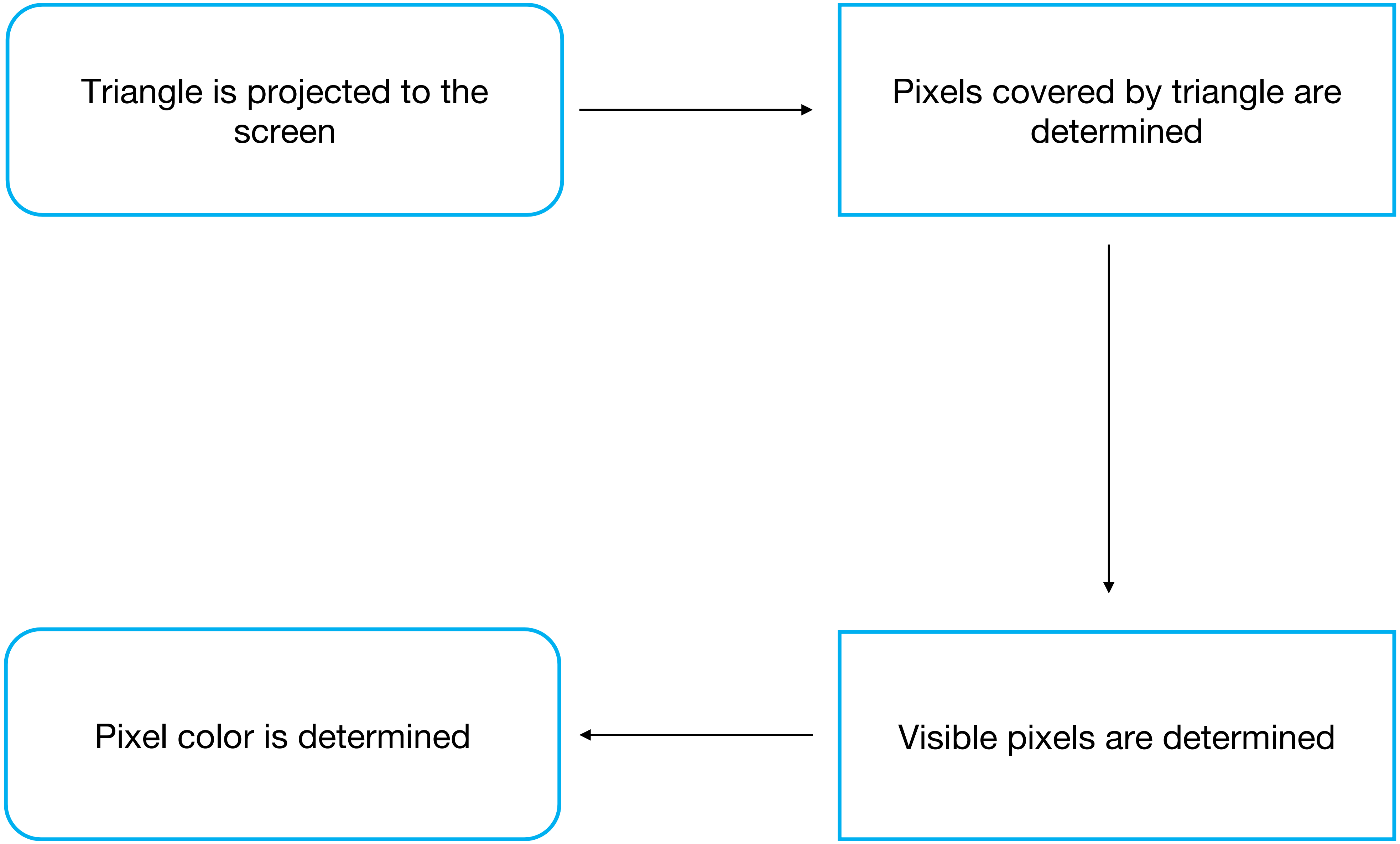
Rendering pipeline








(x_3, y_3, z_3)
(x_1, y_1, z_1)
(x_2, y_2, z_2)



Triangle is projected to the screen

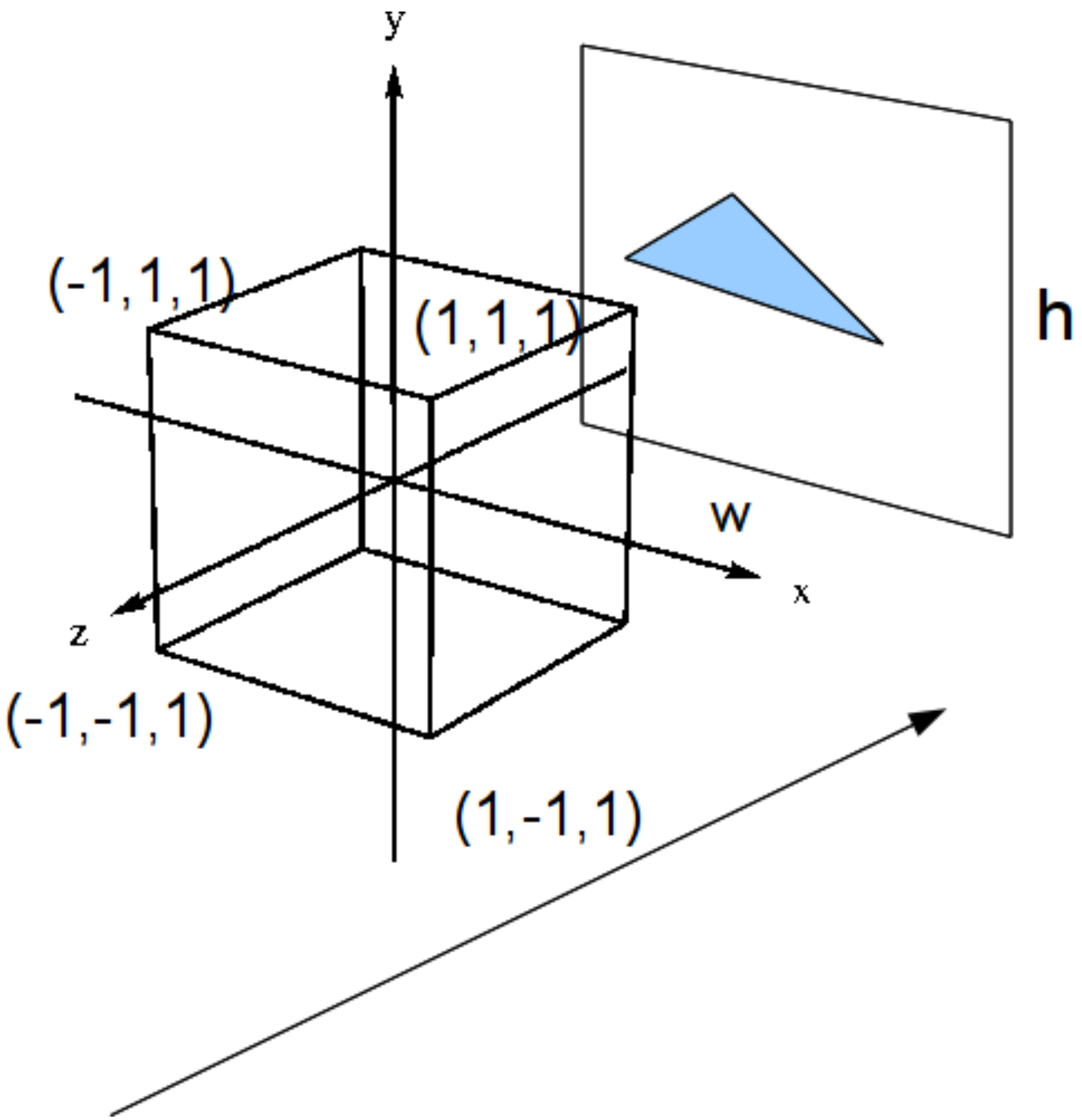
Pixels covered by triangle are determined

Visible pixels are determined

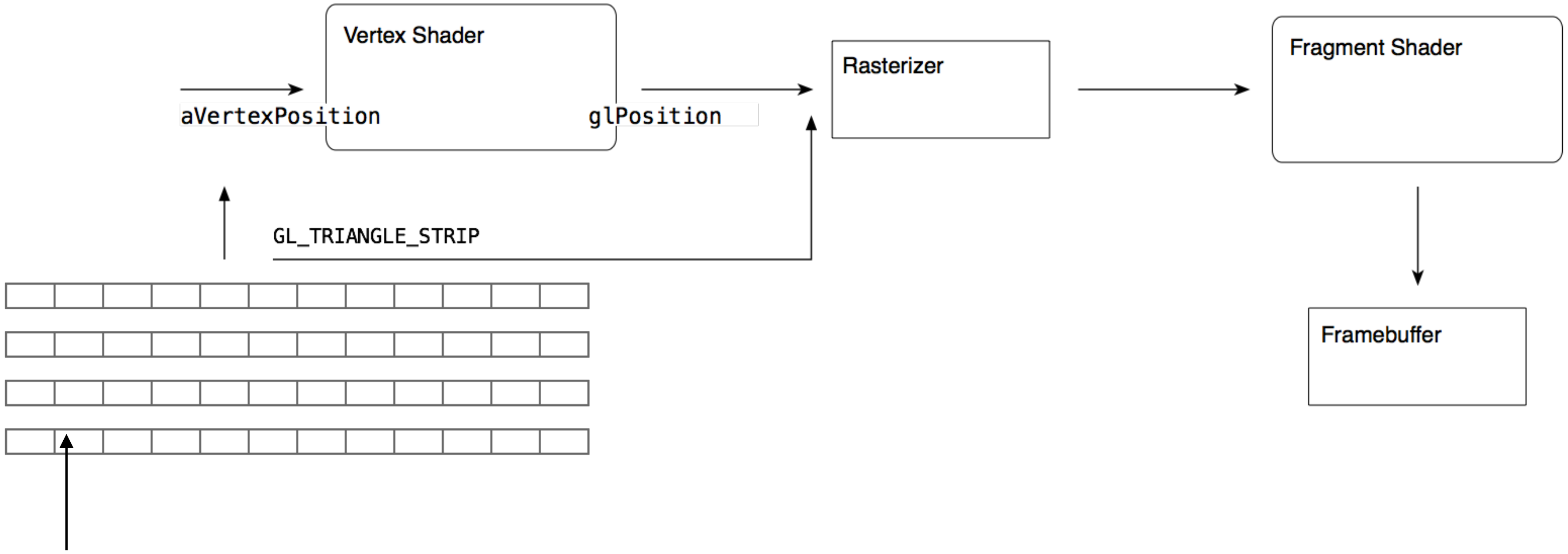
Pixel color is determined

Programable rendering pipeline

OpenGL
TM


y
(-1,1,1)
(1,1,1)
h
w
x
z
(-1,-1,1)
(1,-1,1)

Geometric transformations Rasterisation Shading



Vertex Shader

aVertexPosition

glPosition

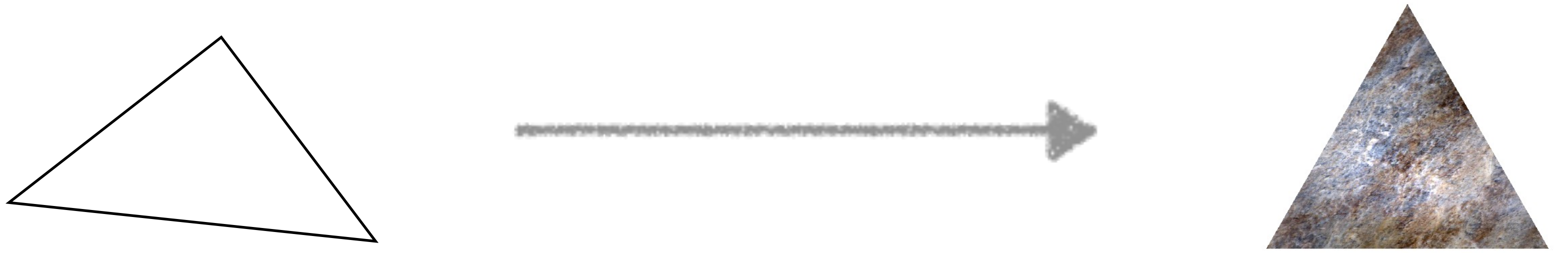
Rasterizer

Fragment Shader

GL_TRIANGLE_STRIP

Framebuffer

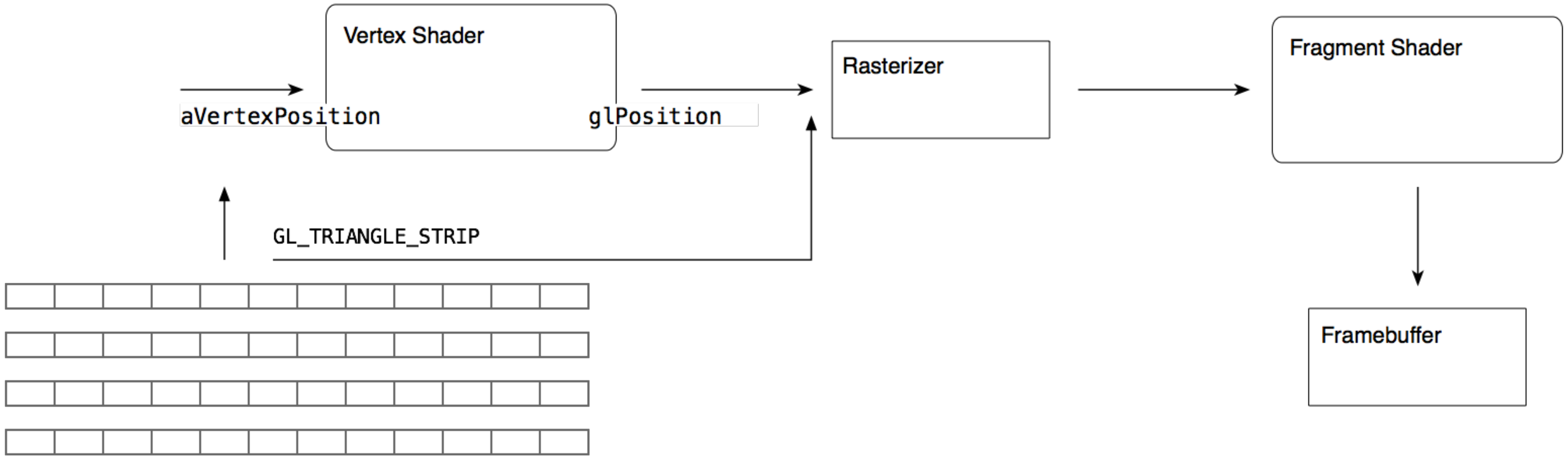
Buffers



Geometric transformations

Rasterisation

Shading



Vertex Shader

aVertexPosition

glPosition

Rasterizer

Fragment Shader

GL_TRIANGLE_STRIP

Framebuffer

Vertex shader program

```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

Clip space coordinates

Normalized device coordinates

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} \longrightarrow \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{pmatrix}$$

gl_Position

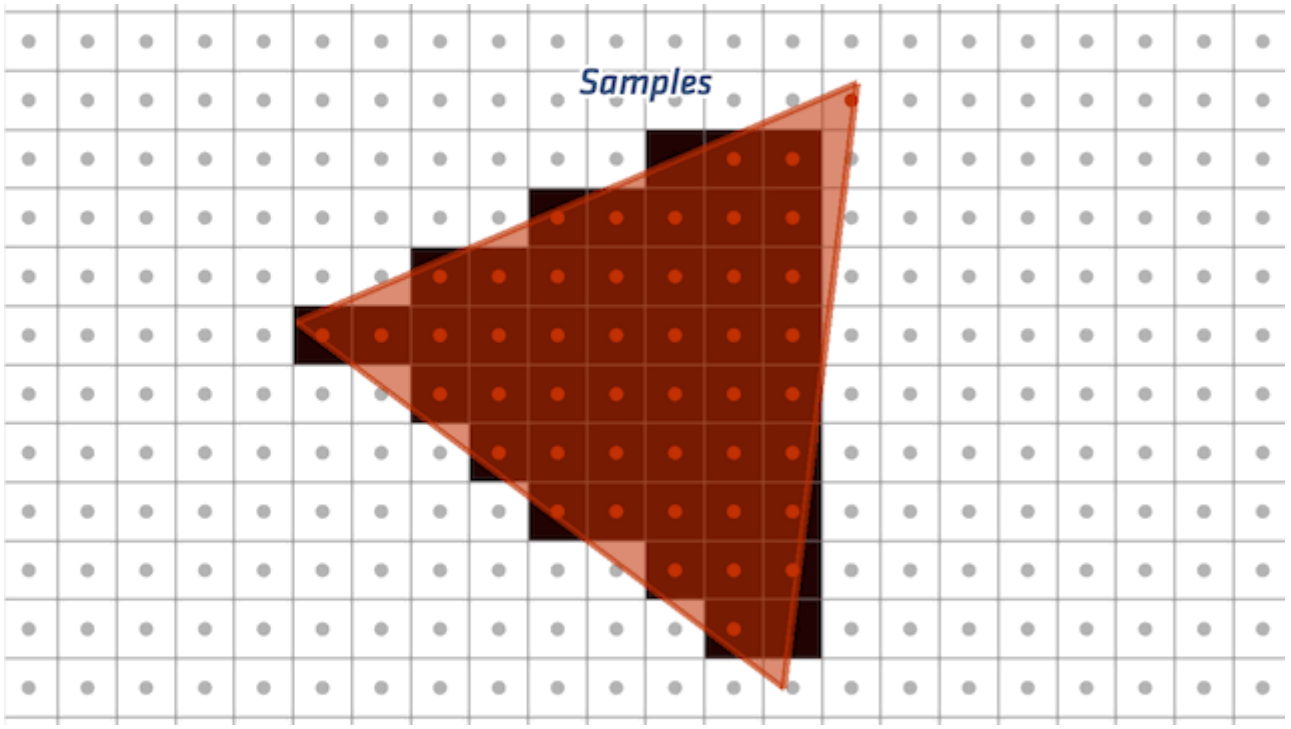
$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}$$

Screen space coordinates

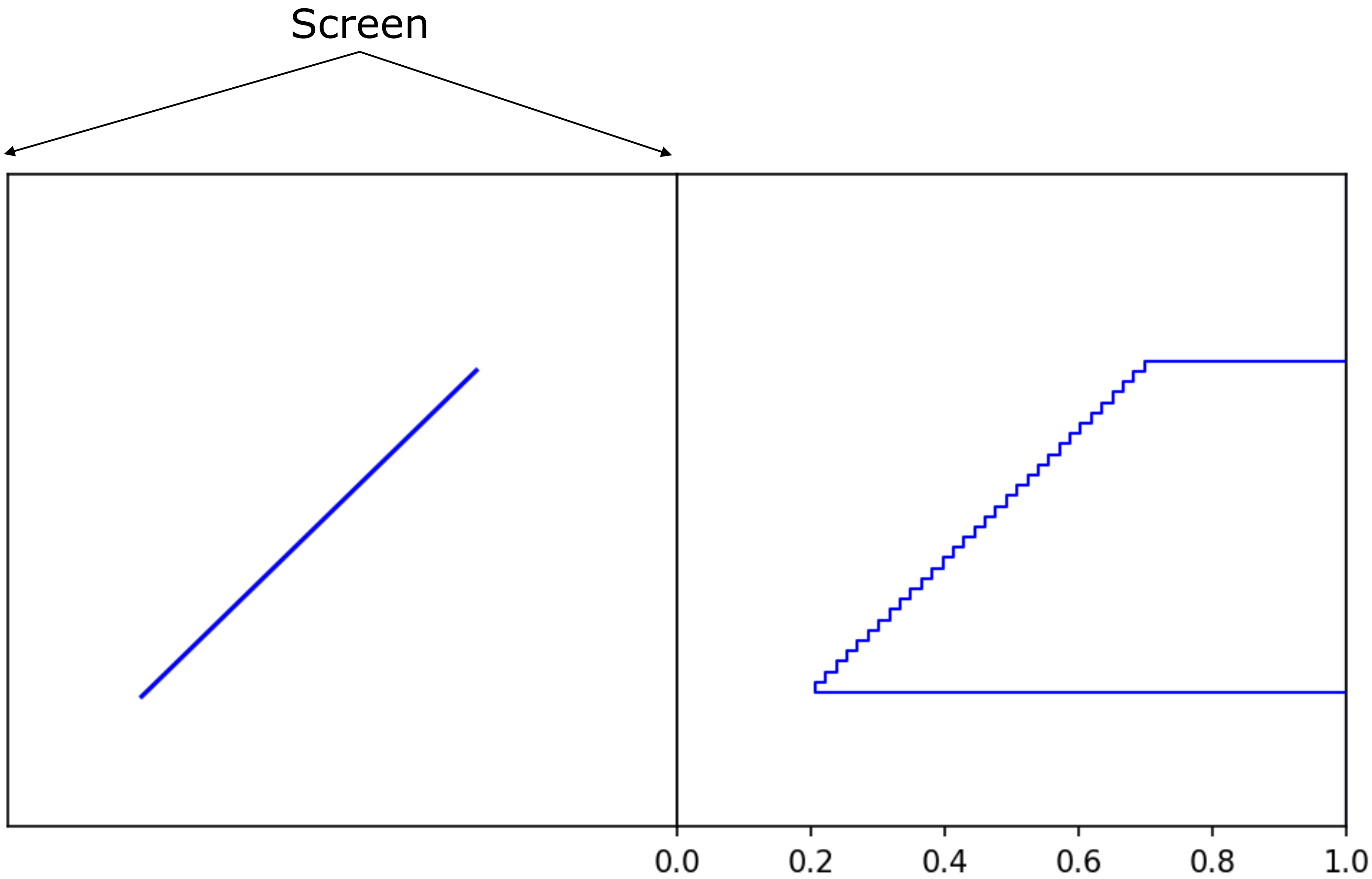
$$\begin{pmatrix} x_s \\ y_s \\ z_s \\ w_s \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(x_c + 1) \times w \\ \frac{1}{2}(y_c + 1) \times h \\ \frac{1}{2}(z_c + 1) \\ \frac{1}{w_c} \end{pmatrix}$$

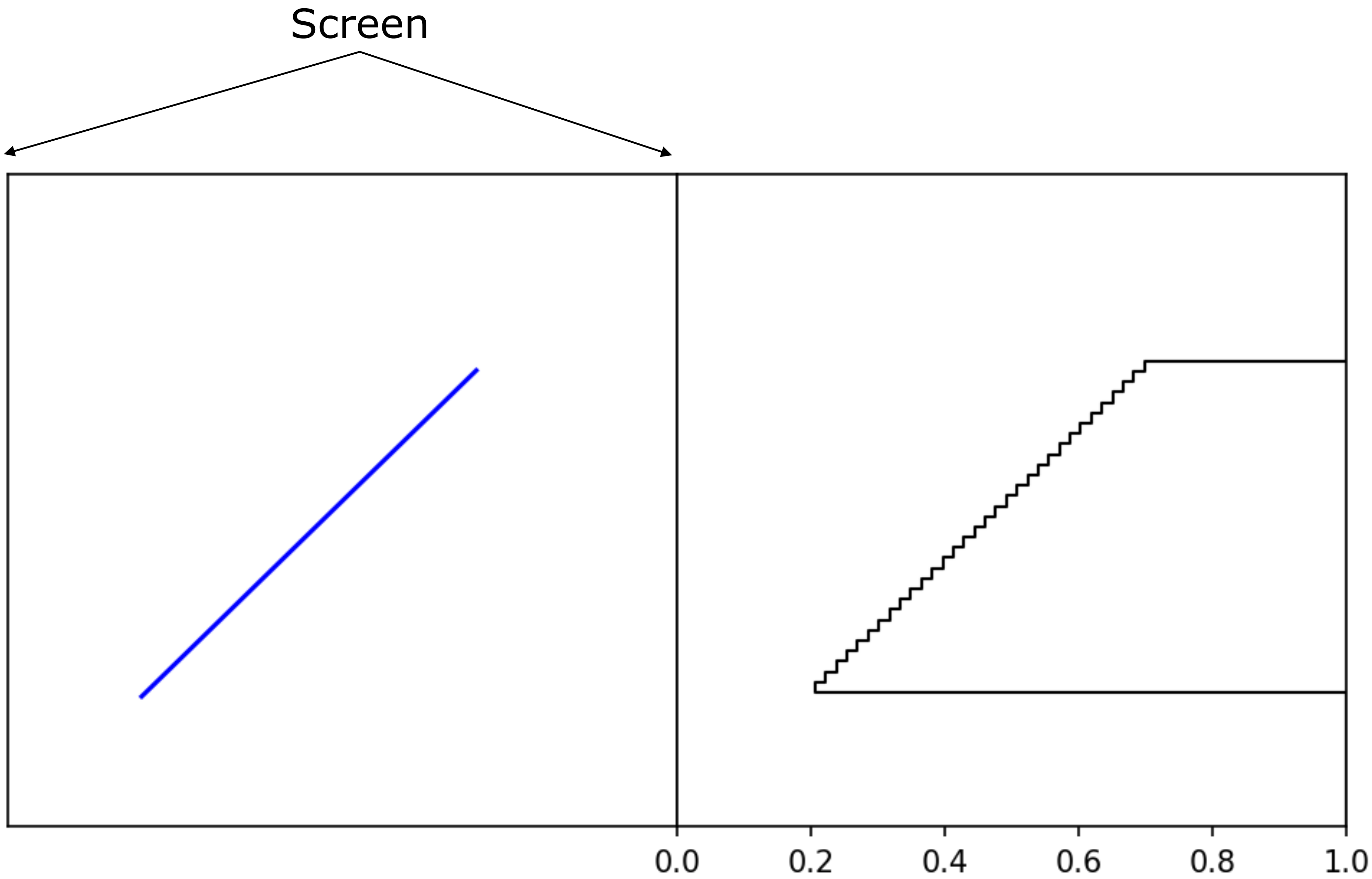
gl_FragCoord

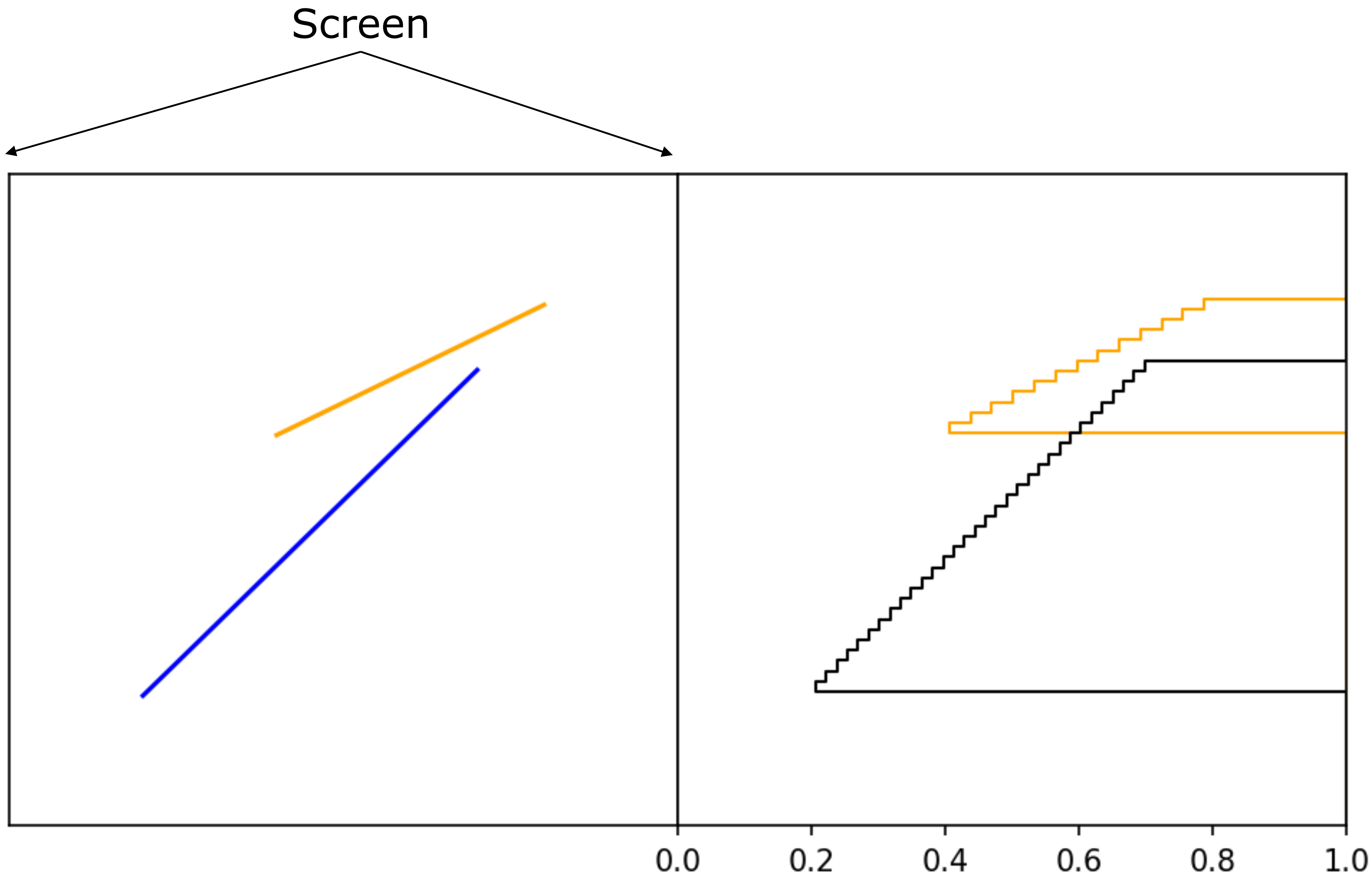
Rasterisation

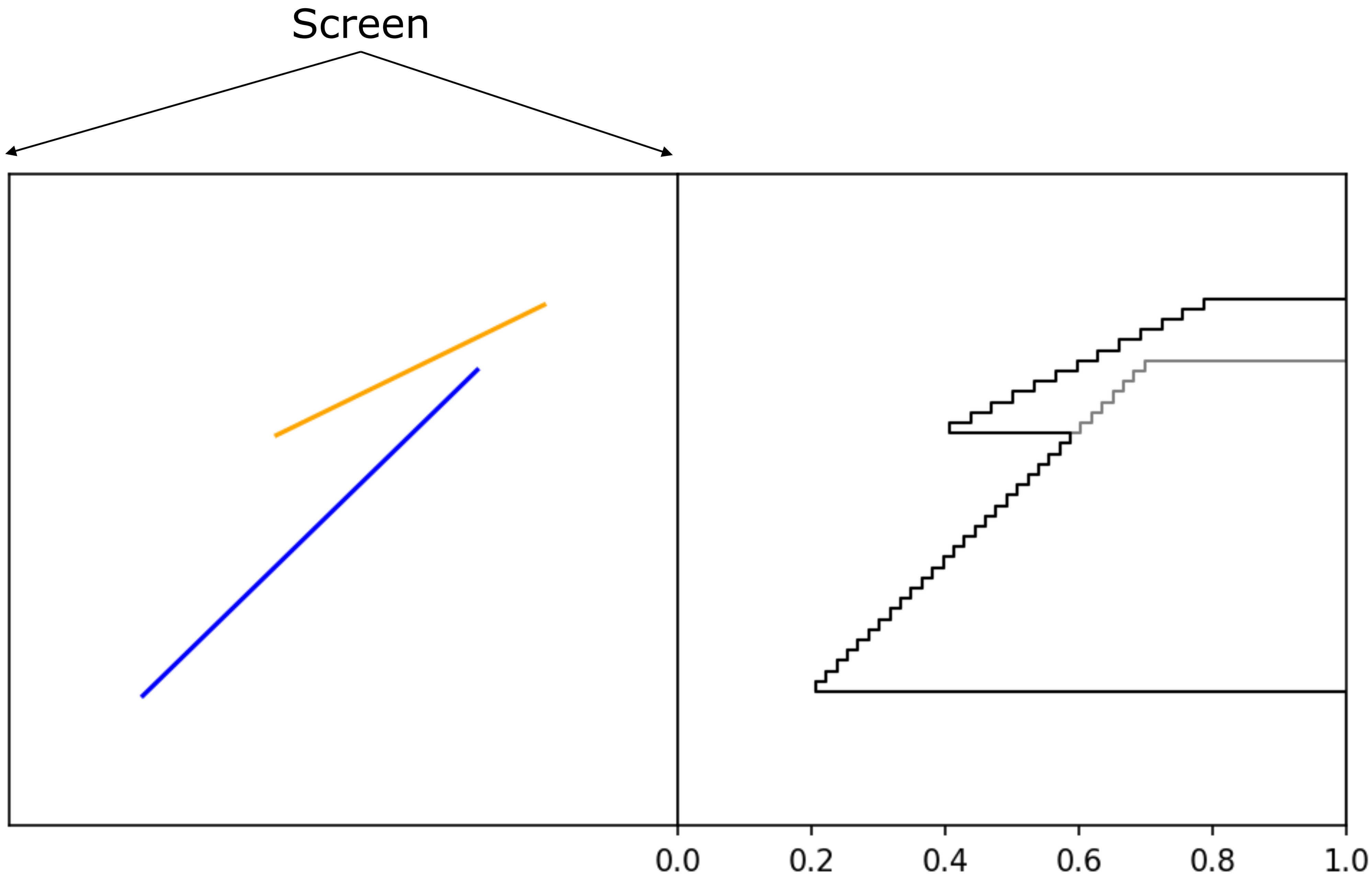


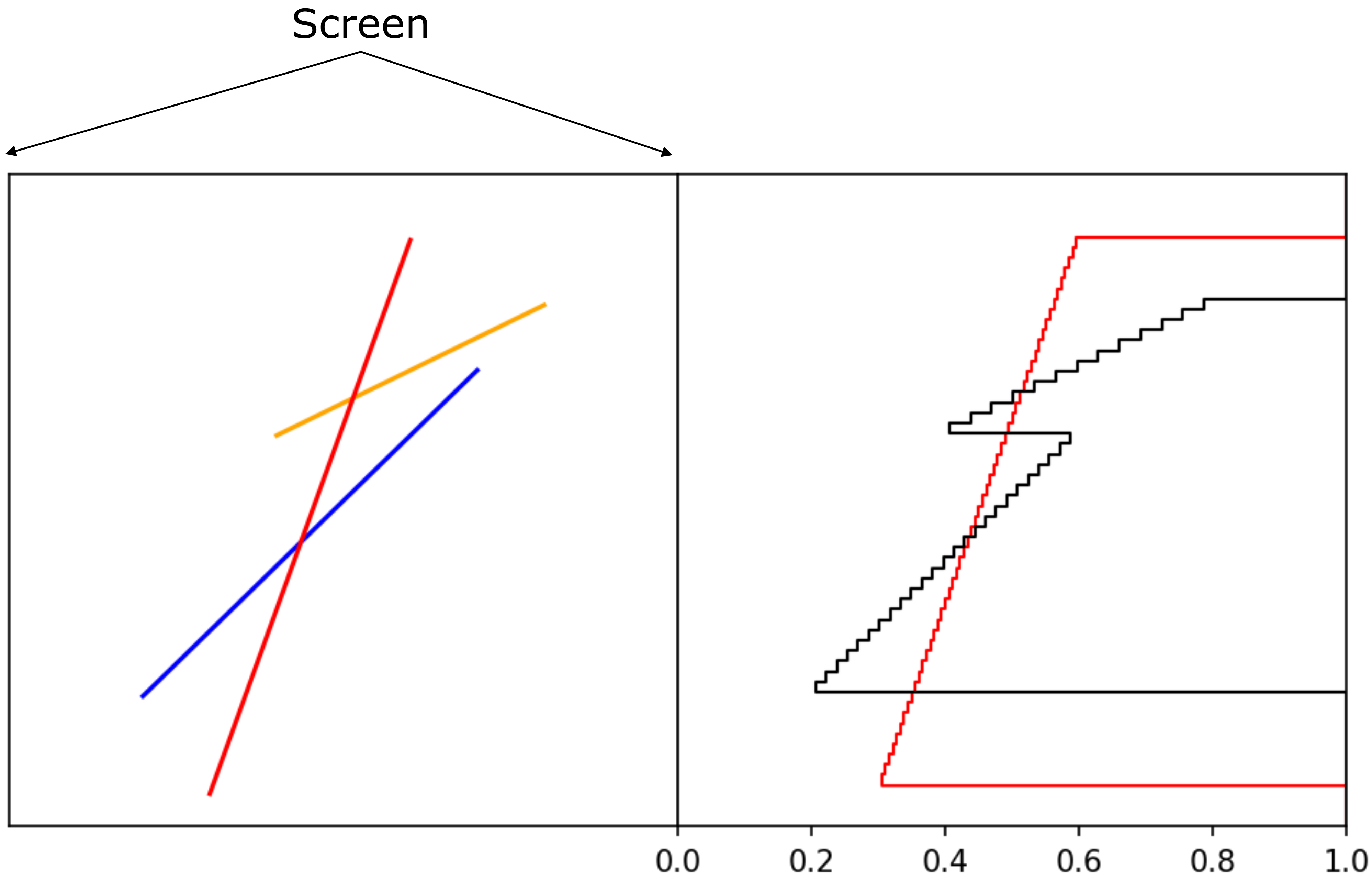
Depth buffer

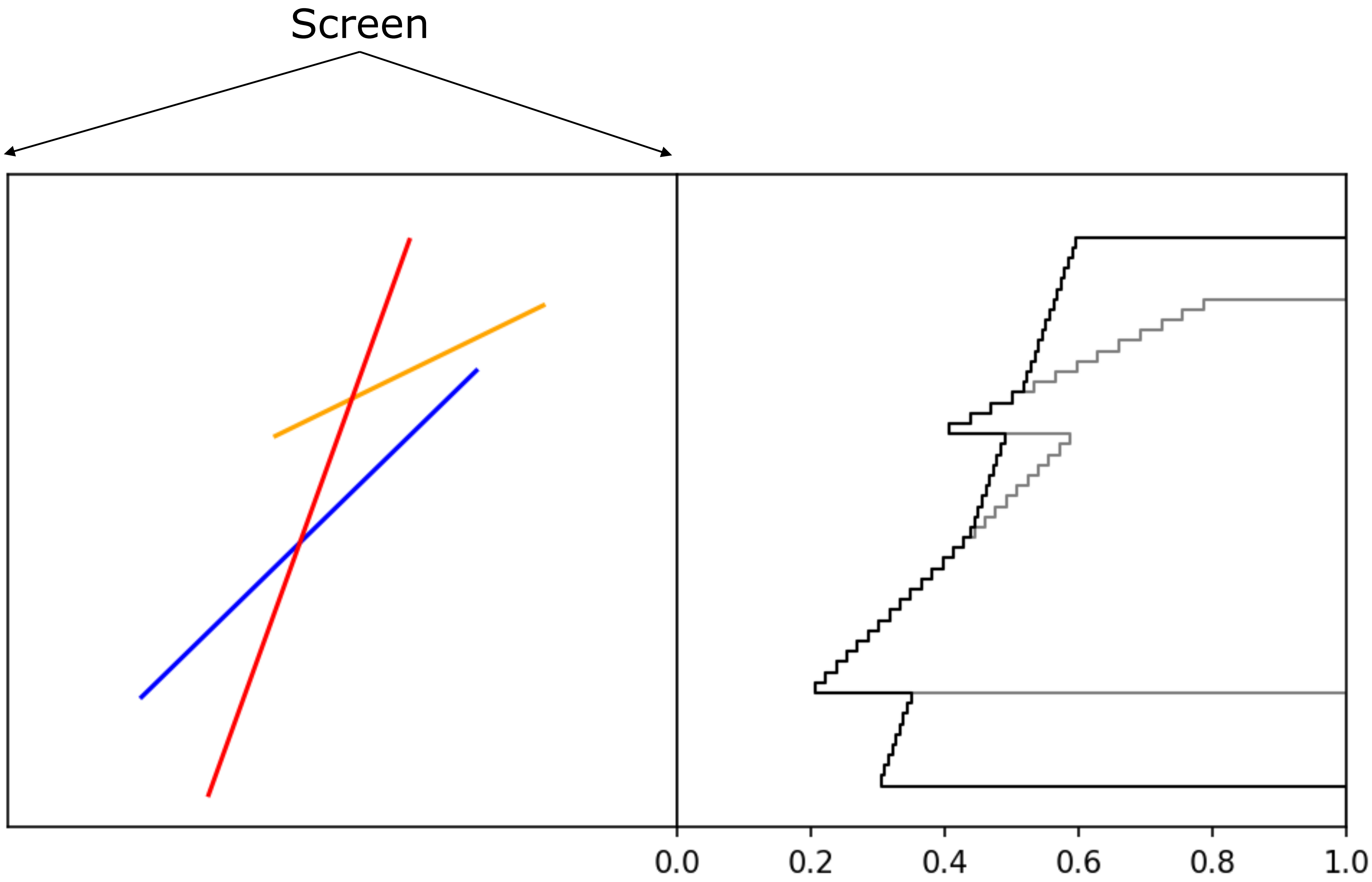

Screen
0.0
0.2
0.4
0.6
0.8
1.0

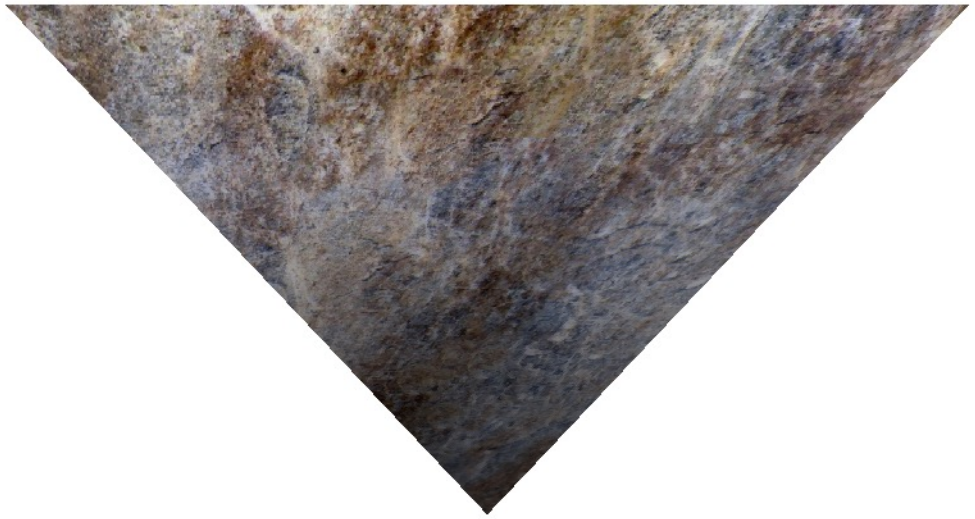

Screen
0.0
0.2
0.4
0.6
0.8
1.0

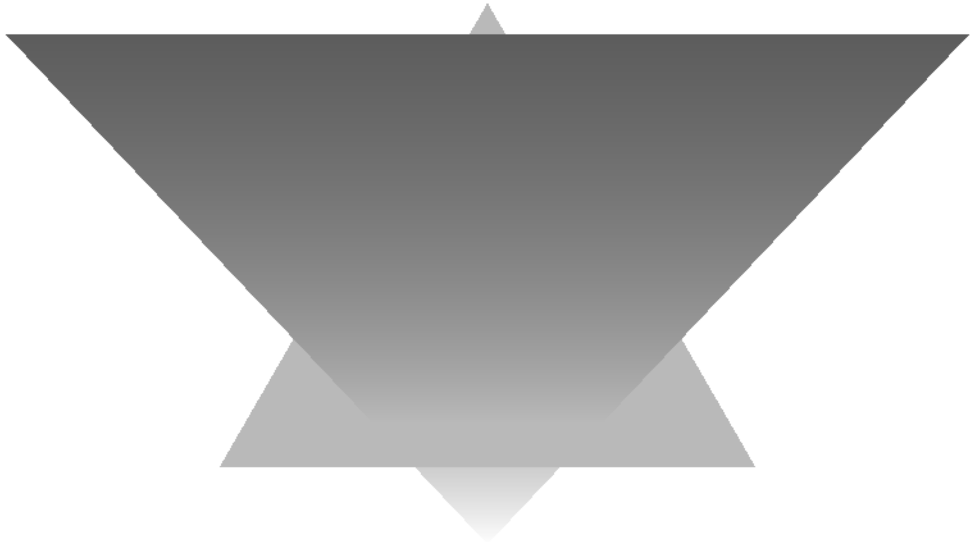

Screen
0.0
0.2
0.4
0.6
0.8
1.0

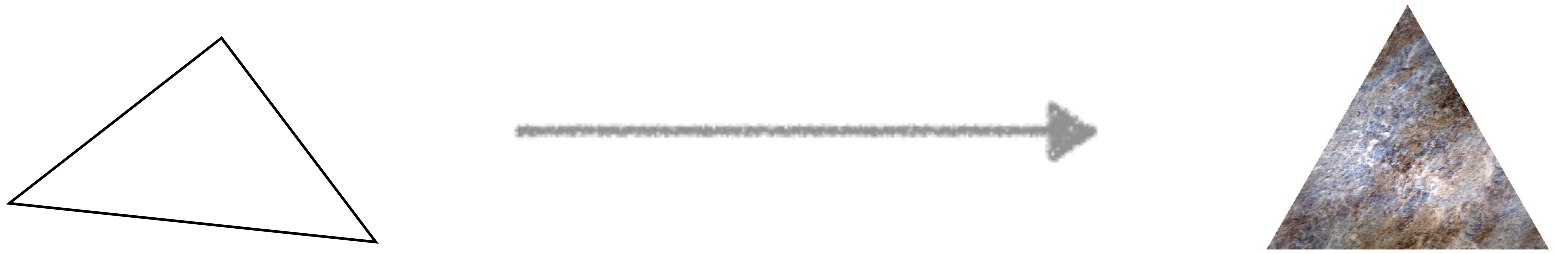

Screen
0.0
0.2
0.4
0.6
0.8
1.0


Screen
0.0
0.2
0.4
0.6
0.8
1.0


Screen
0.0
0.2
0.4
0.6
0.8
1.0



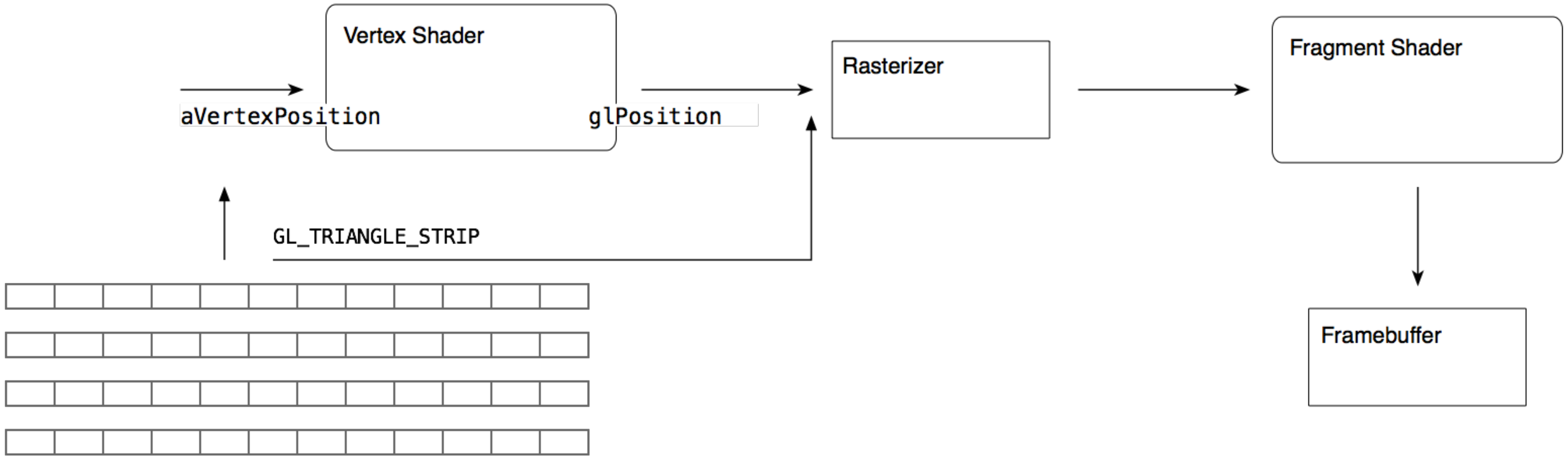




Geometric transformations

Rasterisation

Shading



Vertex Shader

aVertexPosition

glPosition

GL_TRIANGLE_STRIP

Rasterizer

Fragment Shader

Framebuffer

Fragment shader program

```
#version 410

out vec4 vFragColor;

void main() {
   vFragColor = vec4(1.f, 0.f, 0.f, 1.f);
}
```

- Load vertex data into buffers
- Create the program
- Send data to GPU

Nuts and bolts

Vertex Buffers

```
std::vector<Glfloat> vertices {
    -0.5, 0.0, 0.0,
     0.5, 0.0, 0.0,
     0.0, 0.75,0.0
};
```

```
Std::vector<Glfloat> vertices {
    -0.5, 0.0, 0.0,
     0.5, 0.0, 0.0,
     0.0, 0.75,0.0
};

Gluint vbo_handle;
glGenVertexBuffers(1, &vbo_handle);
```

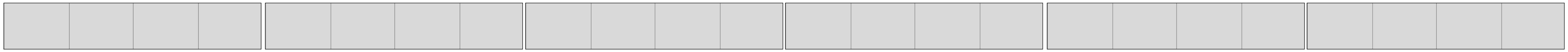
```
Std::vector<Glfloat> vertices {
    -0.5, 0.0, 0.0,
     0.5, 0.0, 0.0,
     0.0, 0.75,0.0
};

Gluint vbo_handle;
glGenVertexBuffers(1, &vbo_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glBufferData(GL_ARRAY_BUFFER, vertices.size()*sizeof(GLfloat),
    vertice.data(), GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER,0);
```

Vertex Object Array

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       3*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```

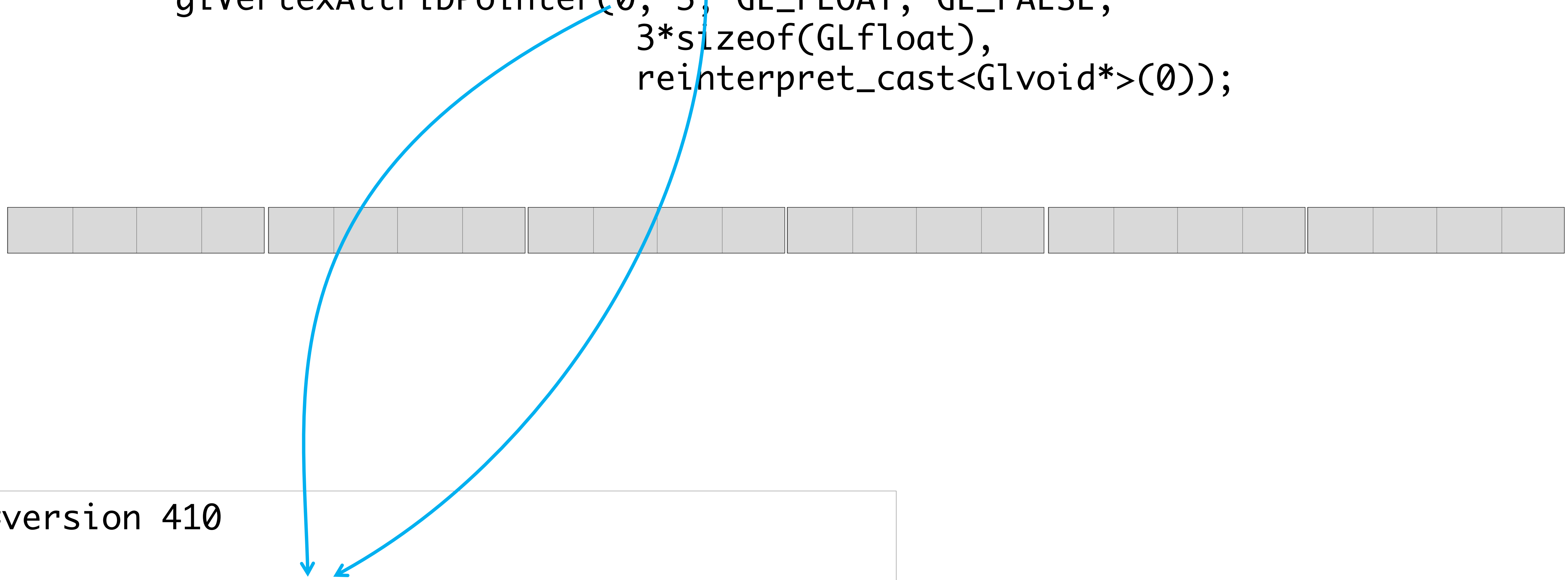
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       3*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```



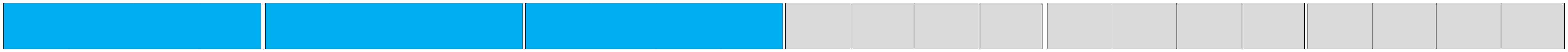
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                      3*sizeof(GLfloat),
                      reinterpret_cast<Glvoid*>(0));
```



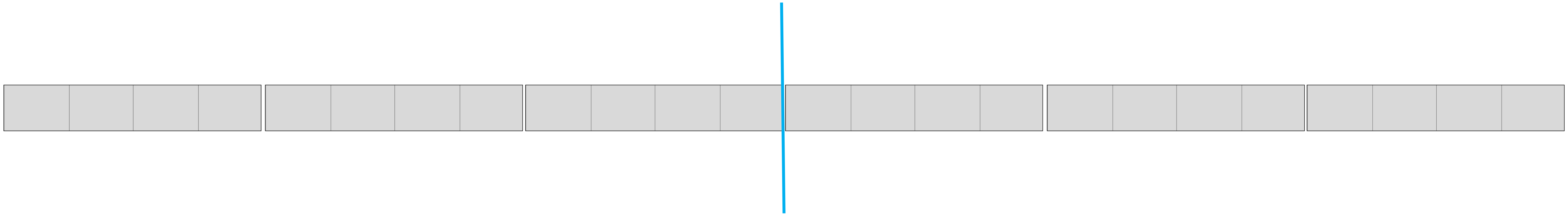
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                         3*sizeof(GLfloat),
                         reinterpret_cast<Glvoid*>(0));
```



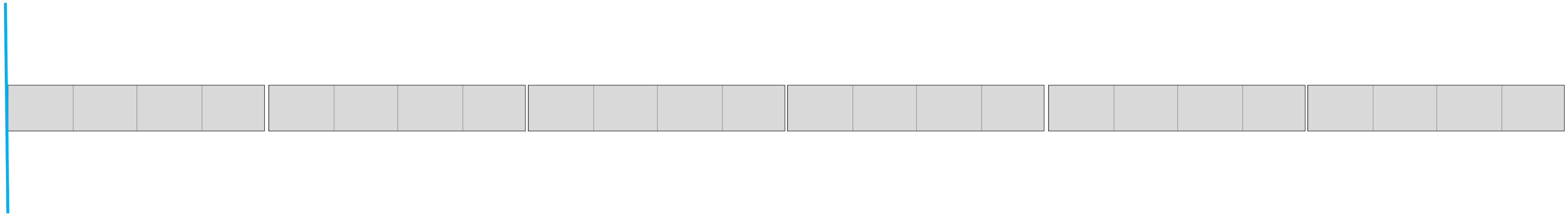
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       3*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```



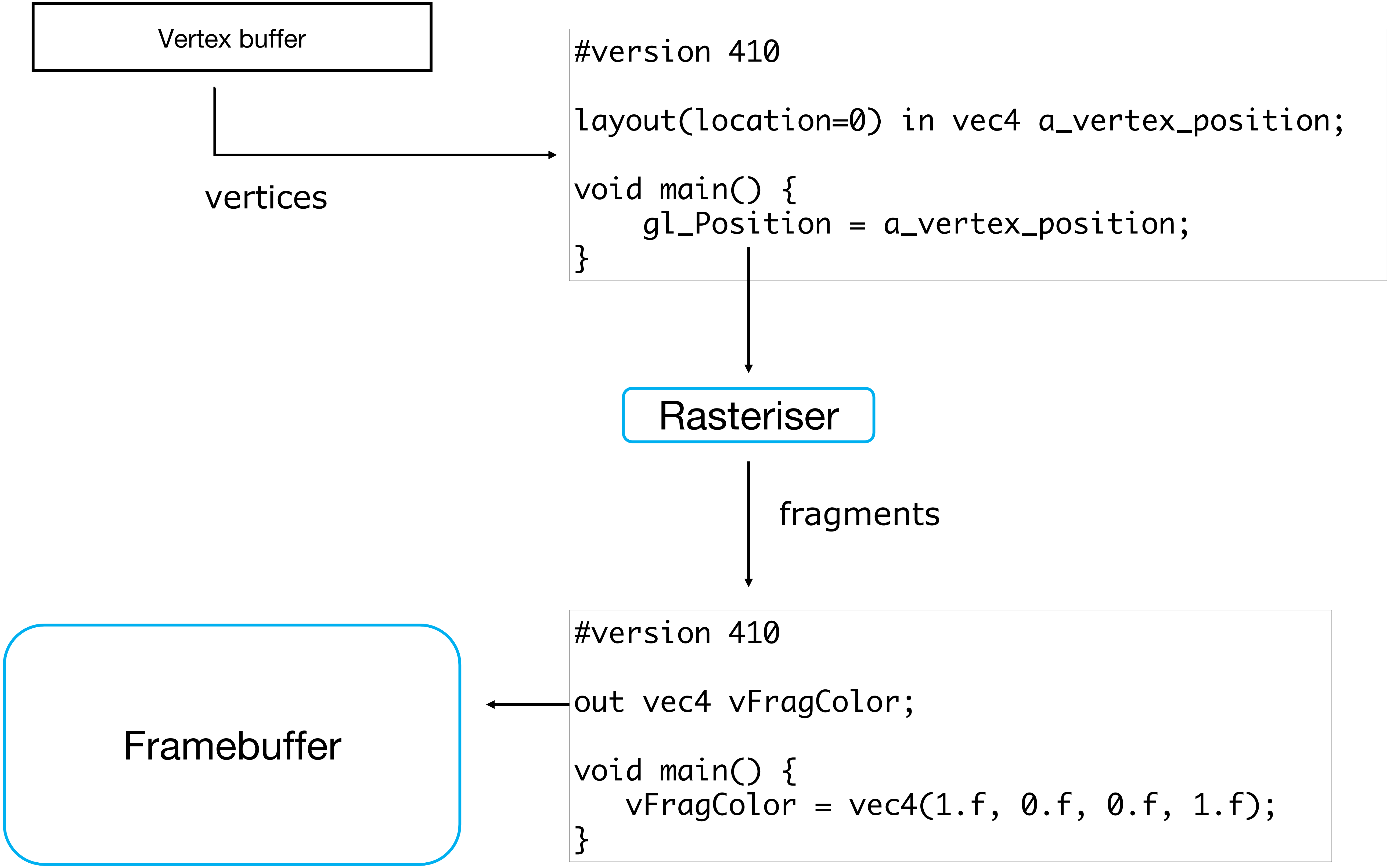
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       3*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```



```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       3*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
glBindBuffer(GL_ARRAY_BUFFER,0u);
glBindVertexArray(0u);
```



Vertex buffer

vertices

```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

Rasteriser

fragments

```
#version 410

out vec4 vFragColor;

void main() {
   vFragColor = vec4(1.f, 0.f, 0.f, 1.f);
}
```

Framebuffer

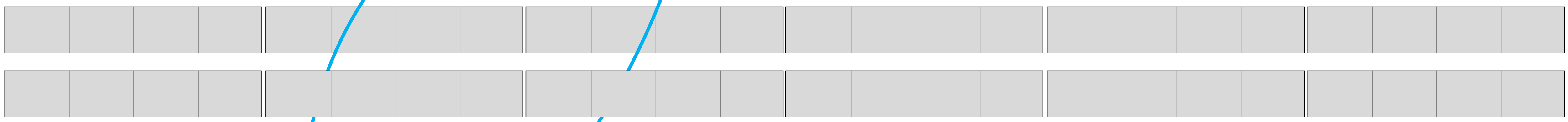
Draw calls

```
glBindVertexArray(vao_handle);
glDrawArrays(GL_TRIANGLE,0,3);
glBindVertexArray(0u);
```

More attributes

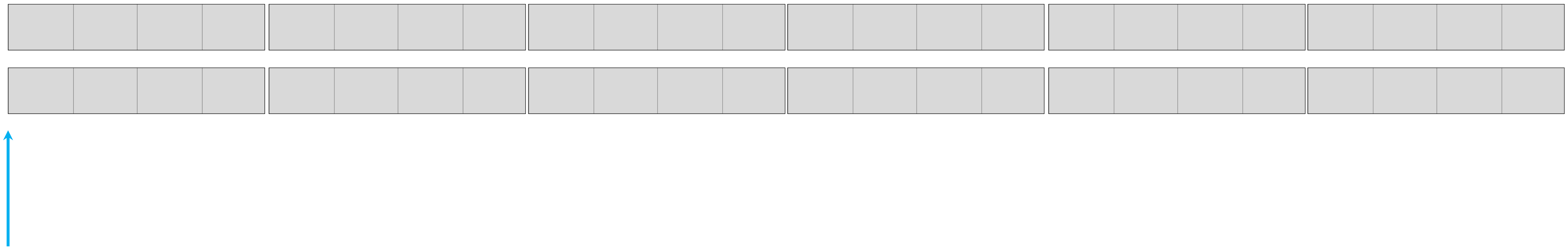
```
Std::vector<Glfloat> vertices {
-0.5, -0.5,  0.0, 1.0, 0.0, 0.0,
 0.5, -0.5,  0.0, 1.0, 0.0, 0.0,
-0.5,  0.5,  0.0, 1.0, 0.0, 0.0,
 0.5, -0.5,  0.0, 0.0, 1.0, 0.0,
 0.5,  0.5,  0.0, 0.0, 1.0, 0.0,
-0.5,  0.5,  0.0  0.0, 1.0, 0.0
};

Gluint vbo_handle;
glGenVertexBuffers(1, &vbo_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glBufferData(GL_ARRAY_BUFFER, vertices.size()*sizeof(GLfloat),
   vertice.data(), GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER,0);
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```



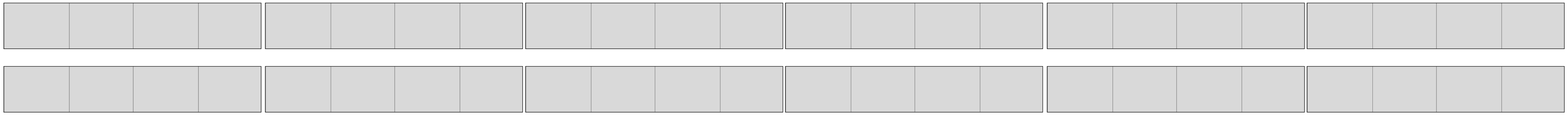
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
```



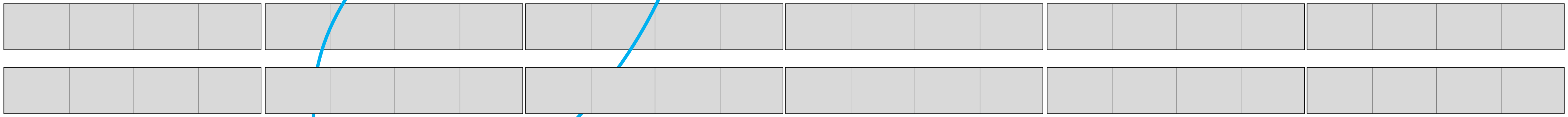
```
#version 410

layout(location=0) in vec4 a_vertex_position;

void main() {
    gl_Position = a_vertex_position;
}
```

```
GLuint vao_handle;
Gluint GenVertexArray(1, &vao_handle);
glBindVertexArray(vao_handle);

glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(3*sizeof(GLfloat)));
```



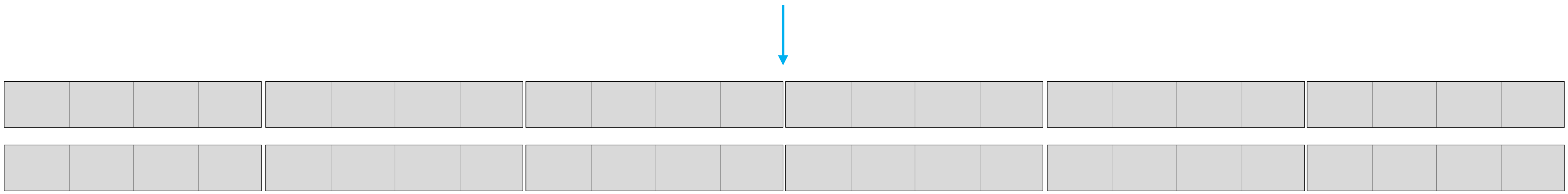
```
#version 410

layout(location=0) in vec4 a_vertex_position;
Layout(location=1) in vec4 a_vertex_color;

out vec4 vertexColor;

void main() {
    vertexColor = a_vertex_color;
    gl_Position = a_vertex_position;
}
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                      6*sizeof(GLfloat),
                      reinterpret_cast<Glvoid*>(0));
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                      6*sizeof(GLfloat),
                      reinterpret_cast<Glvoid*>(3*sizeof(GLfloat)));
```



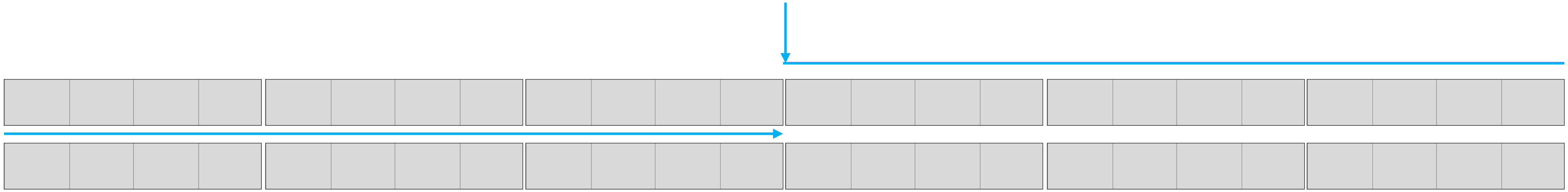
```
#version 410

layout(location=0) in vec4 a_vertex_position;
Layout(location=1) in vec4 a_vertex_color;

out vec4 vertexColor;

void main() {
    vertexColor = a_vertex_color;
    gl_Position = a_vertex_position;
}
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(3*sizeof(GLfloat)));
```

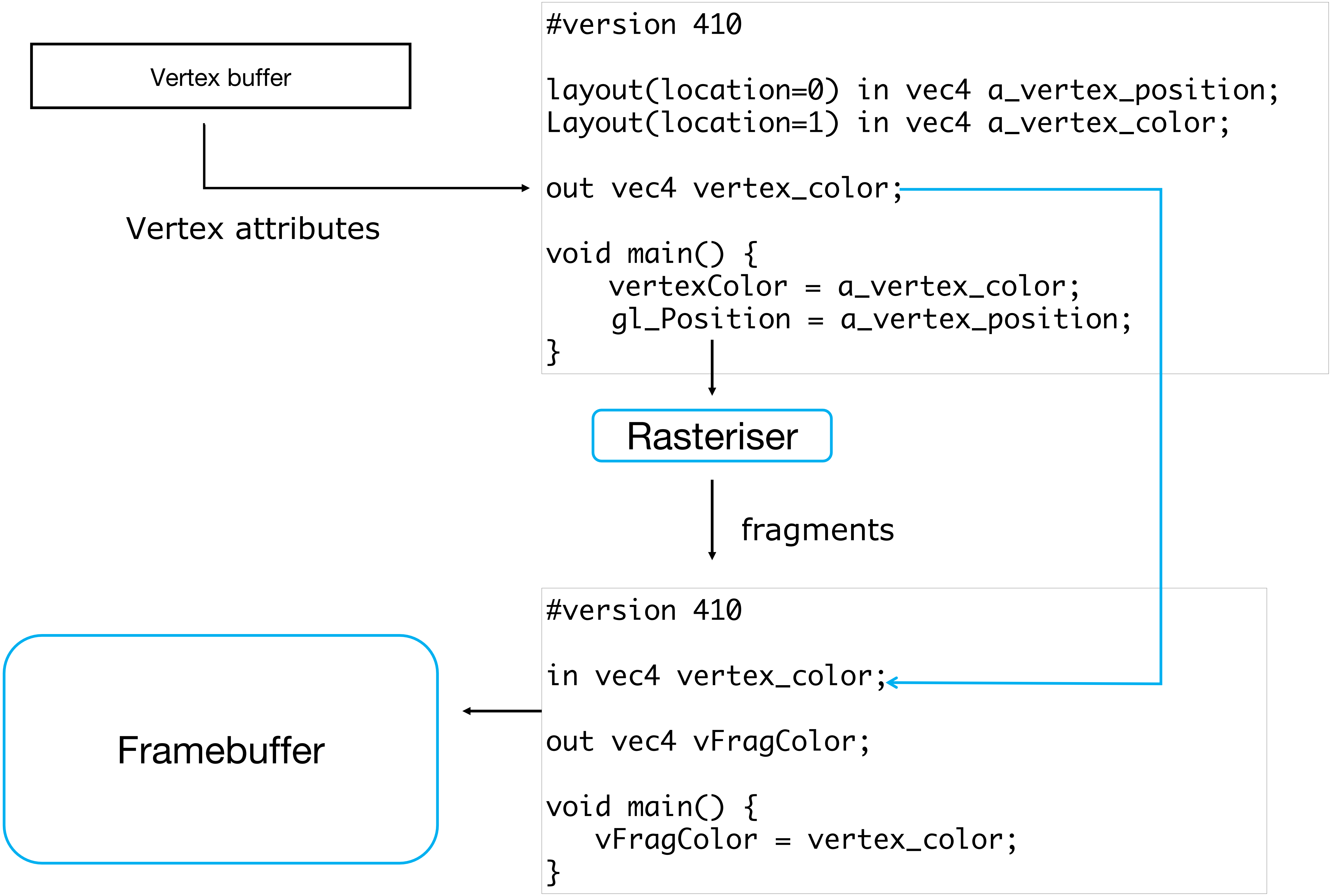


```
#version 410

layout(location=0) in vec4 a_vertex_position;
Layout(location=1) in vec4 a_vertex_color;

out vec4 vertexColor;

void main() {
    vertexColor = a_vertex_color;
    gl_Position = a_vertex_position;
}
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo_handle);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(0));
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                       6*sizeof(GLfloat),
                       reinterpret_cast<Glvoid*>(3*sizeof(GLfloat)));
```



```
#version 410

layout(location=0) in vec4 a_vertex_position;
Layout(location=1) in vec4 a_vertex_color;

out vec4 vertexColor;

void main() {
    vertexColor = a_vertex_color;
    gl_Position = a_vertex_position;
}
```



Vertex buffer

Vertex attributes

```
#version 410

layout(location=0) in vec4 a_vertex_position;
Layout(location=1) in vec4 a_vertex_color;

out vec4 vertex_color;

void main() {
    vertexColor = a_vertex_color;
    gl_Position = a_vertex_position;
}
```

Rasteriser

fragments

```
#version 410

in vec4 vertex_color;

out vec4 vFragColor;

void main() {
   vFragColor = vertex_color;
}
```

Framebuffer

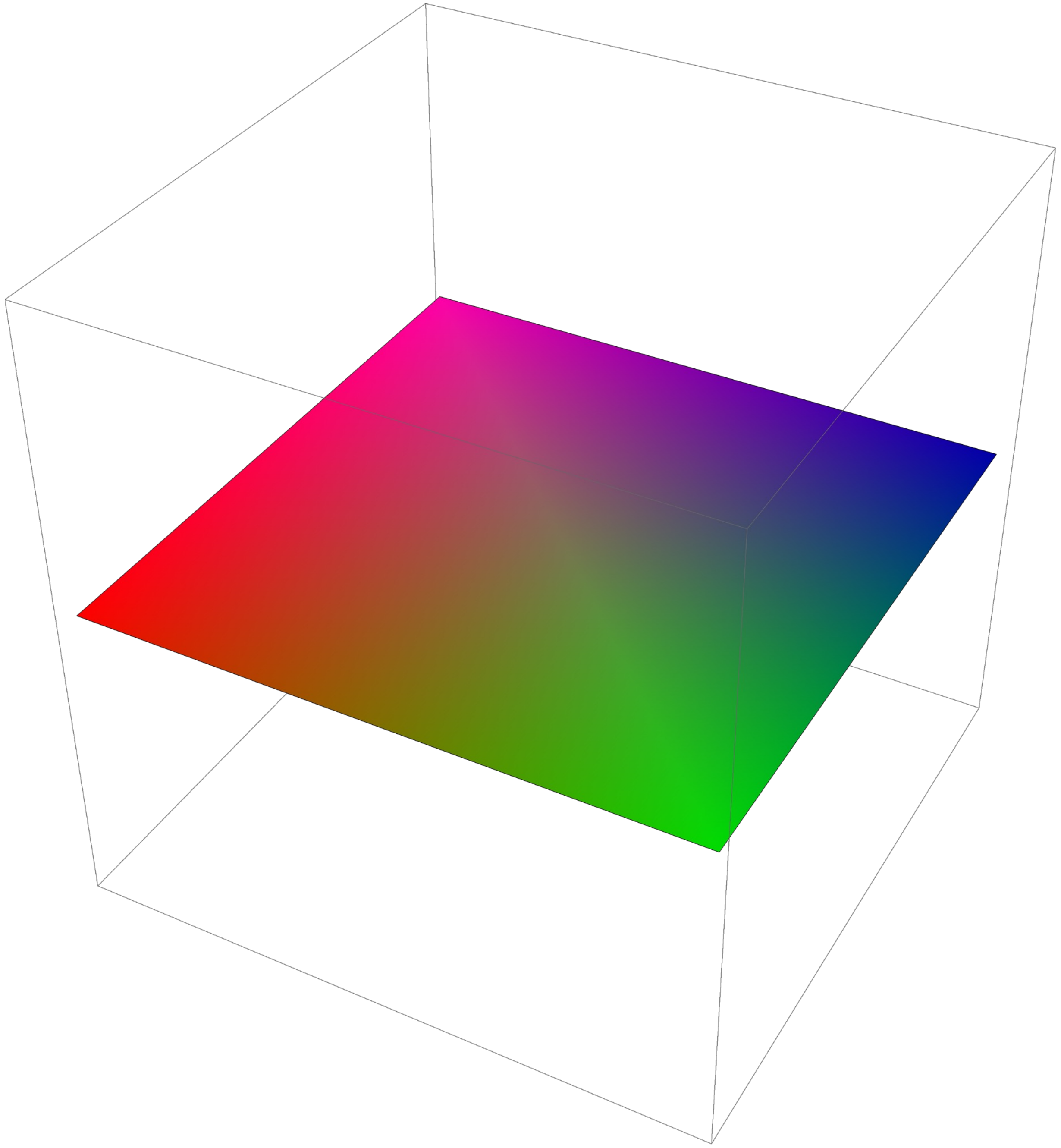
```
glBindVertexArray(vao_handle);
glDrawArrays(GL_TRIANGLE,0,6);
glBinVertexArray(0u);
```

Errors

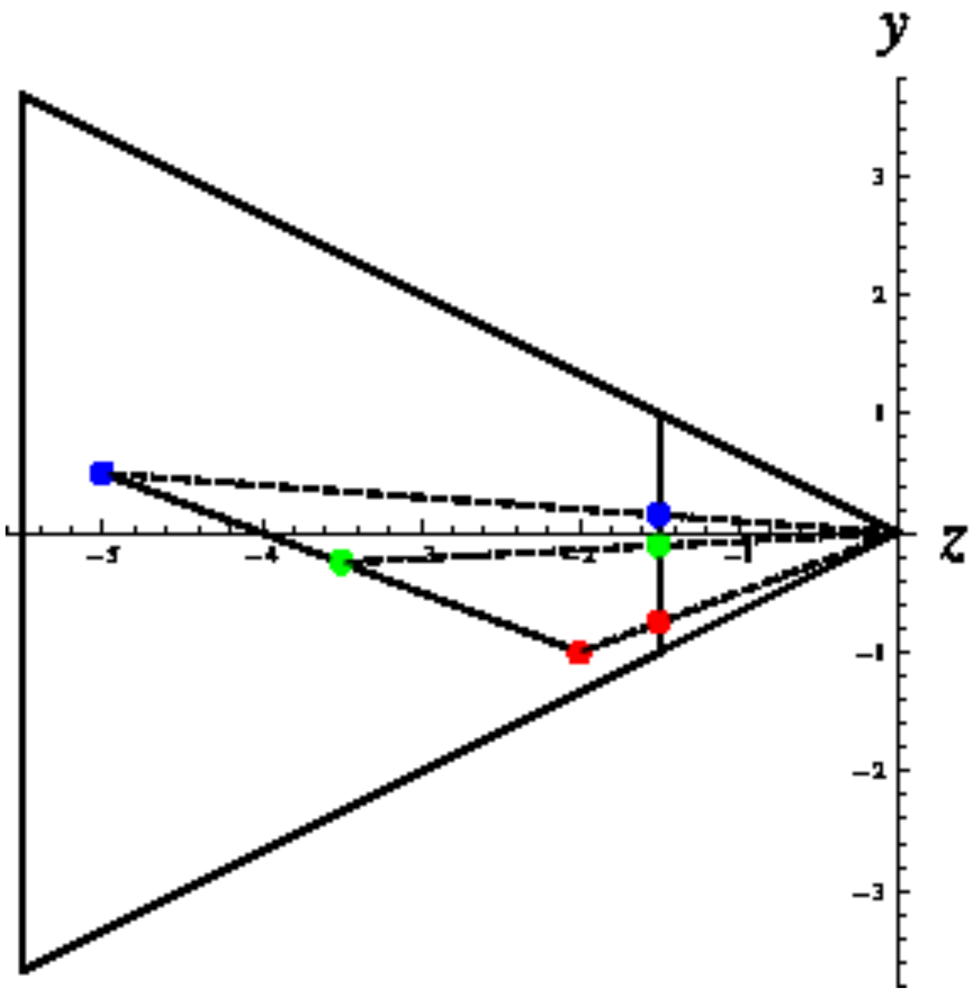
```
glBindBuffer(GL_VERTEX_BUFFER,v_buffer_handle);
auto error = glGetError();
if(error != GL_NO_ERROR) {
    SPDLOG_ERROR(“Encoutered {} error”, error);
}
```

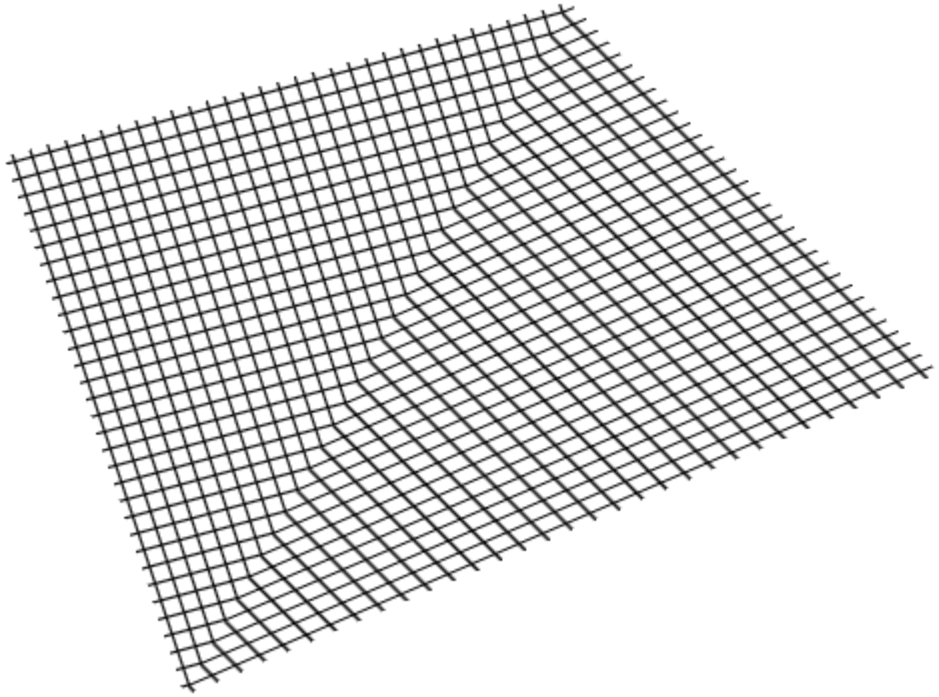
```
OGL_CALL(glBindBuffer(GL_VERTEX_BUFFER,v_buffer_handle));
```

Attribute interpolation

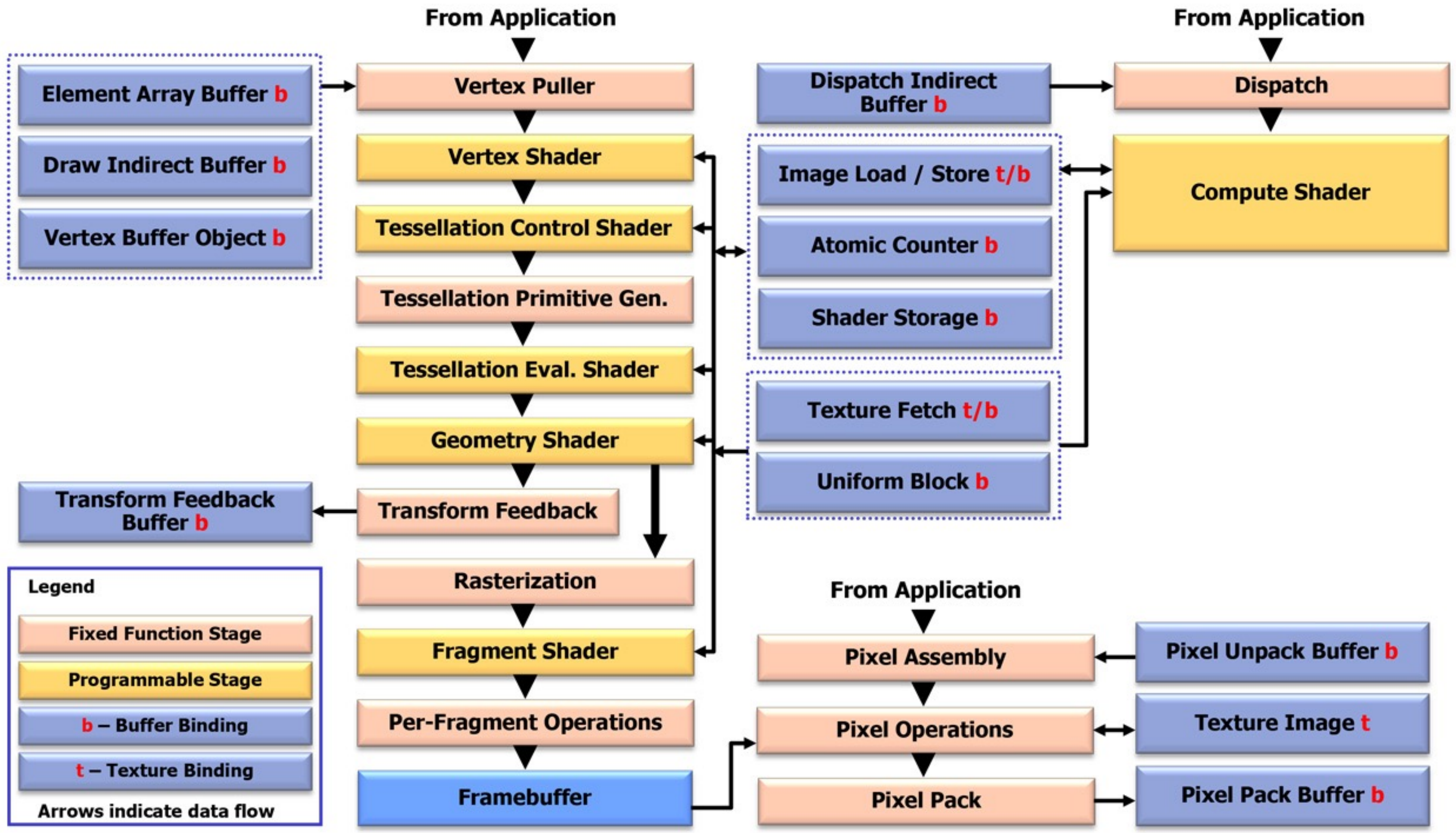


Perspective corrected interpolation


y
3
2
1
−1
−2
−3
z
−5
−4
−3
−2
−1



OpenGL 4.3 with Compute Shaders



From Application

Element Array Buffer b

Draw Indirect Buffer b

Vertex Buffer Object b

Vertex Puller

Vertex Shader

Tessellation Control Shader

Tessellation Primitive Gen.

Tessellation Eval. Shader

Geometry Shader

Transform Feedback Buffer b

Transform Feedback

Rasterization

Fragment Shader

Per-Fragment Operations

Framebuffer

Dispatch Indirect Buffer b

Image Load / Store t/b

Atomic Counter b

Shader Storage b

Texture Fetch t/b

Uniform Block b

From Application

Dispatch

Compute Shader

From Application

Pixel Assembly

Pixel Operations

Pixel Pack

Pixel Unpack Buffer b

Texture Image t

Pixel Pack Buffer b

Legend

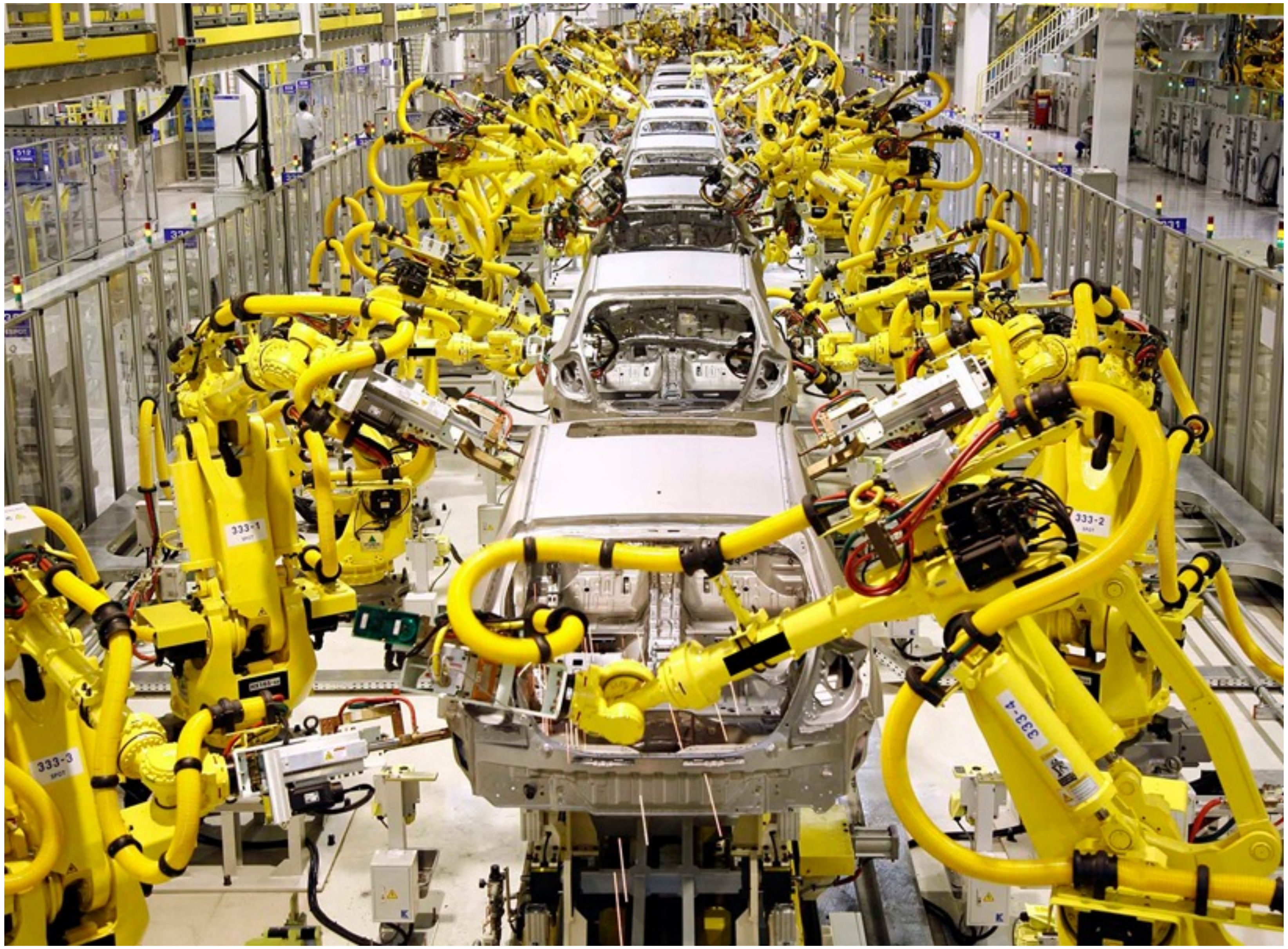
Fixed Function Stage

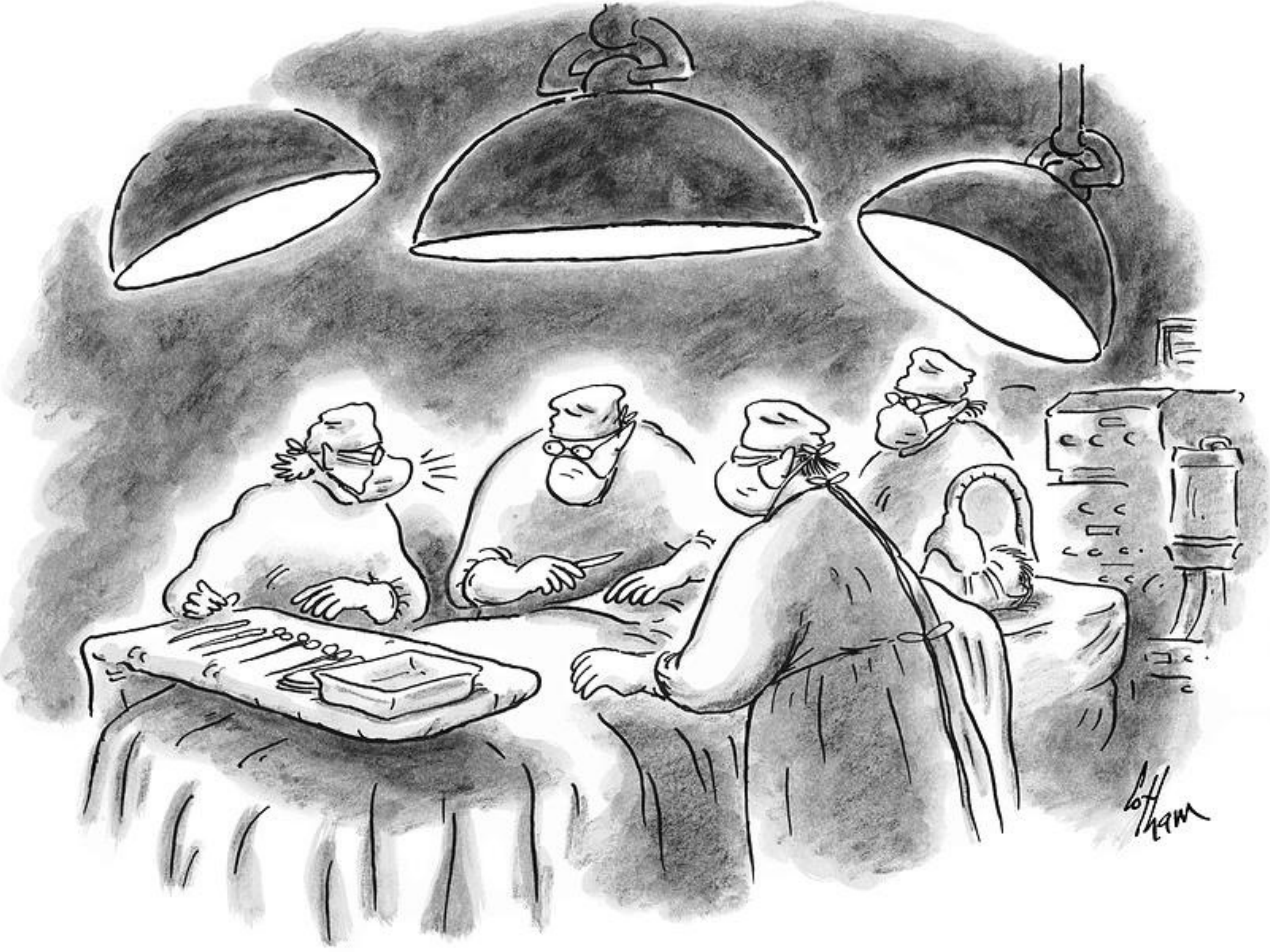
Programmable Stage

b – Buffer Binding

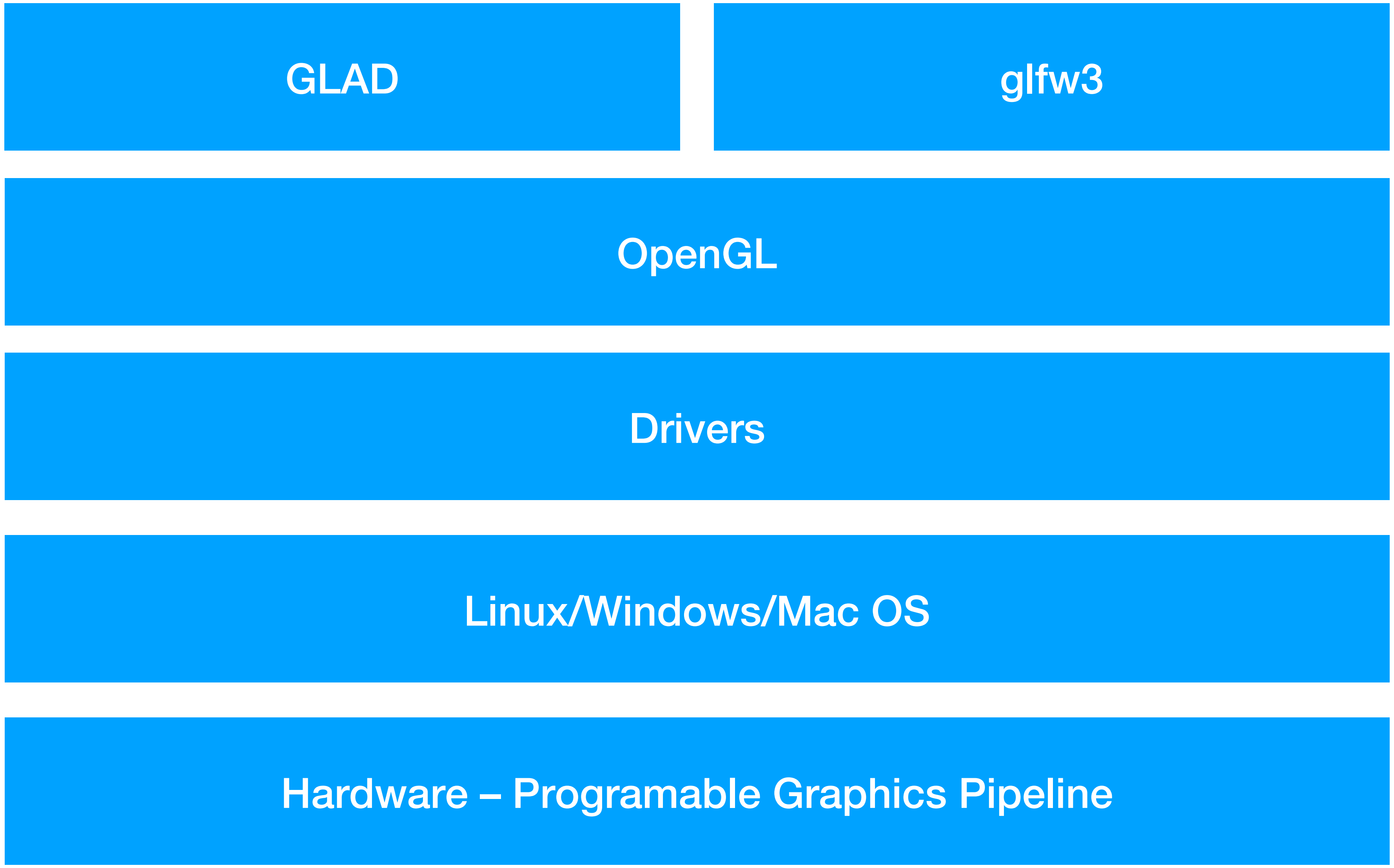
t – Texture Binding

Arrows indicate data flow


333-1
333-2
333-3
333-4



"Spare us the gory details."



GLAD

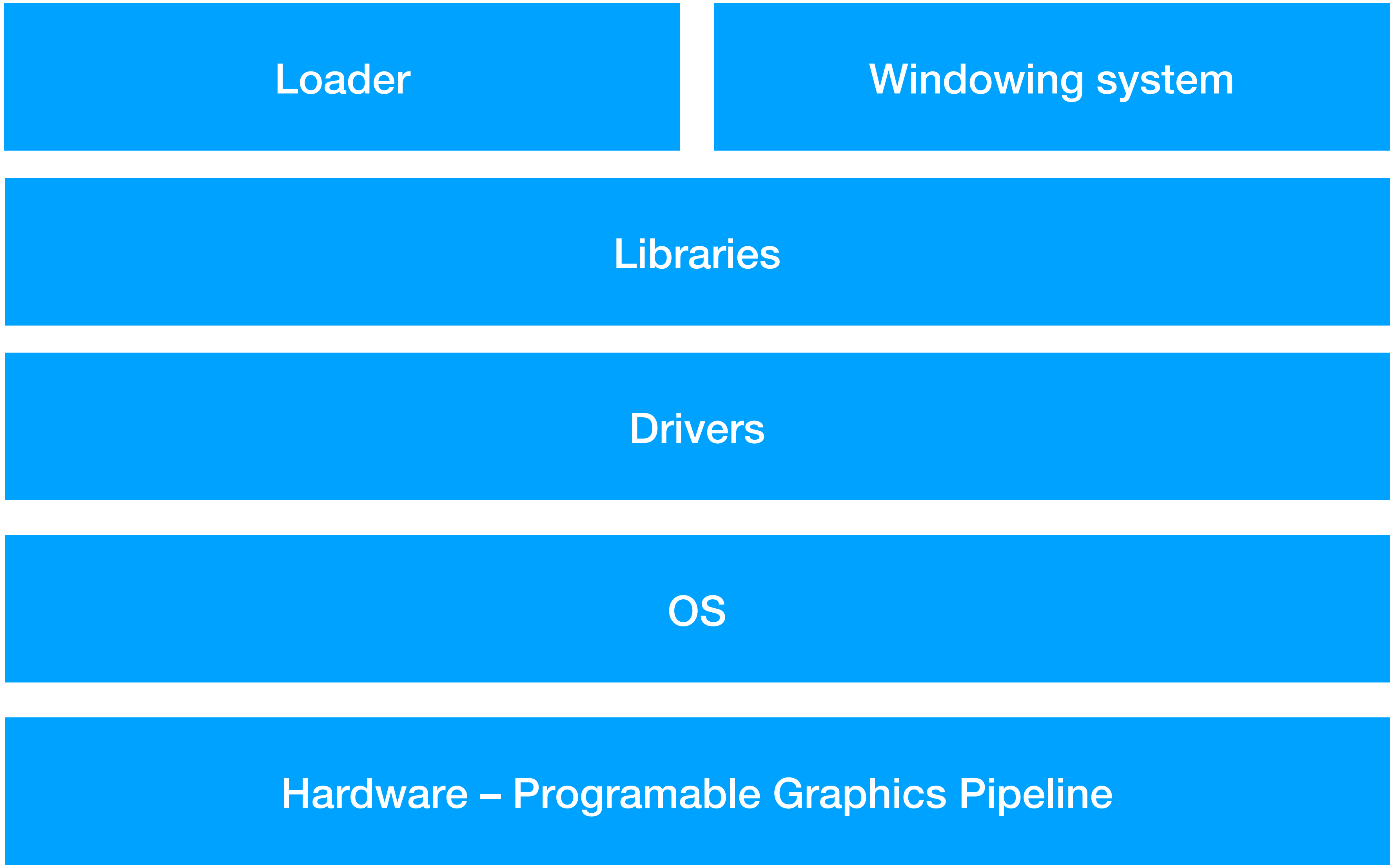
glfw3

OpenGL

Drivers

Linux/Windows/Mac OS

Hardware – Programable Graphics Pipeline



Loader

Windowing system

Libraries

Drivers

OS

Hardware – Programable Graphics Pipeline