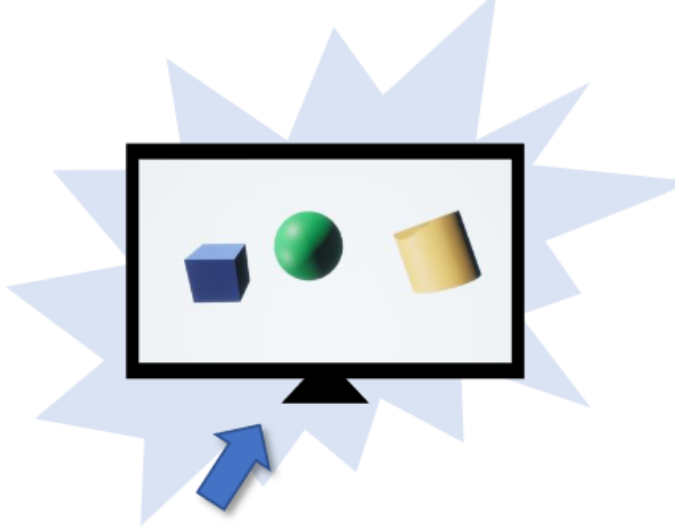
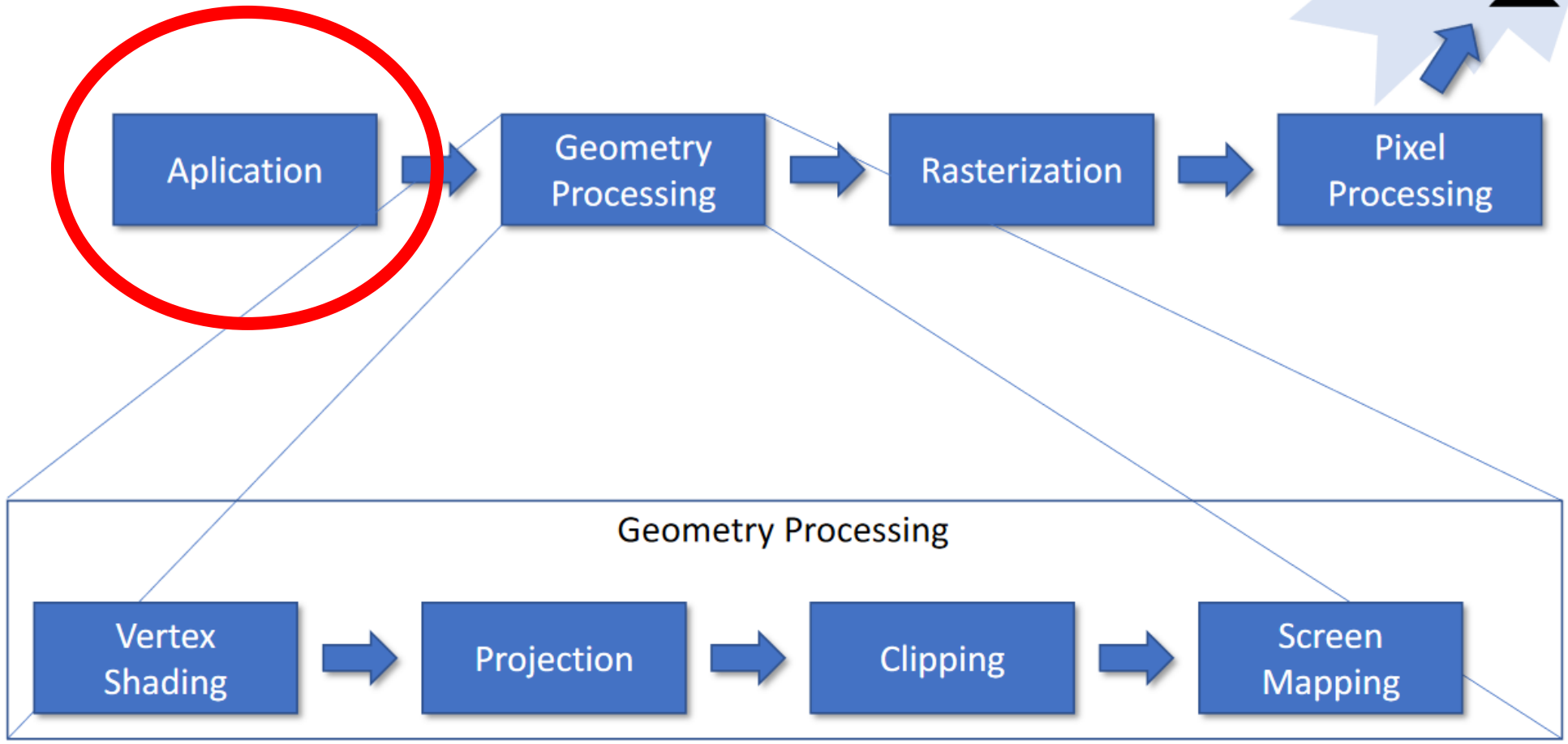
Auxiliar 2- Introducción a OpenGL

Sebastián Olmos H.

CC3501-1: Modelación y Computación Gráfica para Ingenieros

Graphics Rendering Pipeline

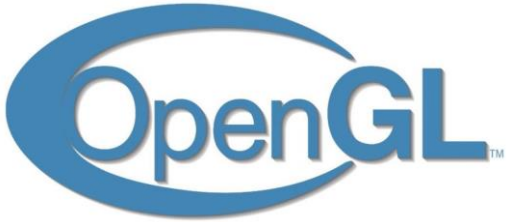




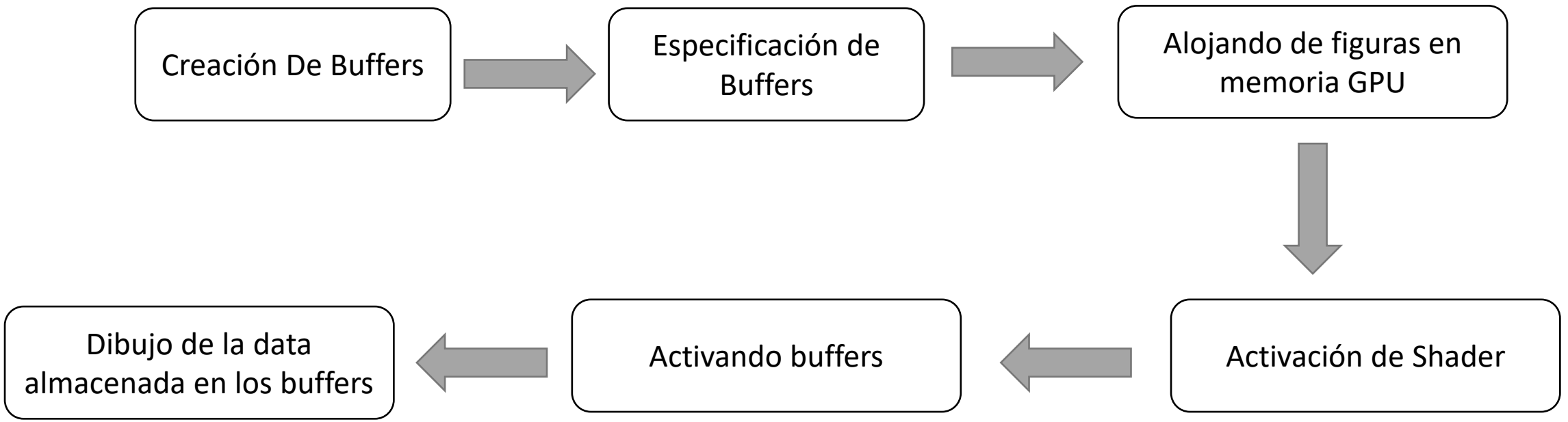
Estructura de nuestras aplicaciones

```
import glfw
from OpenGL.GL import *
import OpenGL.GL.shaders
import numpy as np
```

- ¿Que son OpenGL y GLFW?



OpenGL como máquina de estados



Controlador

- Manejamos el input
- Modificamos valores que se van a ver reflejados en la ejecución de la aplicación

```
class Controller:
        fillPolygon = True


controller = Controller()
def on_key(window, key, scancode, action, mods):
        if action != glfw.PRESS:
                return
        global controller
        if key == glfw.KEY_SPACE:
                controller.fillPolygon = not controller.fillPolygon
        elif key == glfw.KEY_ESCAPE:
                glfw.set_window_should_close(window, True)
        else:
                print('Unknown key')
```

Manejo de ventanas y loop principal

```
if __name__ == "__main
            if not glfw.init():
                        glfw.set_window_should_close(window, True)
            width = 600
            height = 600
            window = glfw.create_window(width, height, "", None, None)
            if not window:
                        glfw.terminate()
                        glfw.set_window_should_close(window, True)
            glfw.make_context_current(window)
            glfw.set_key_callback(window, on_key)
            glClearColor(0.2, 0.2, 0.2, 1.0)
            while not glfw.window_should_close(window):
                        glfw.poll_events()
                        glClear(GL_COLOR_BUFFER_BIT)
                        glfw.swap_buffers(window)
            glfw.terminate()
```

- Se ubica la lógica de la aplicación
- Se esta actualizando constantemente

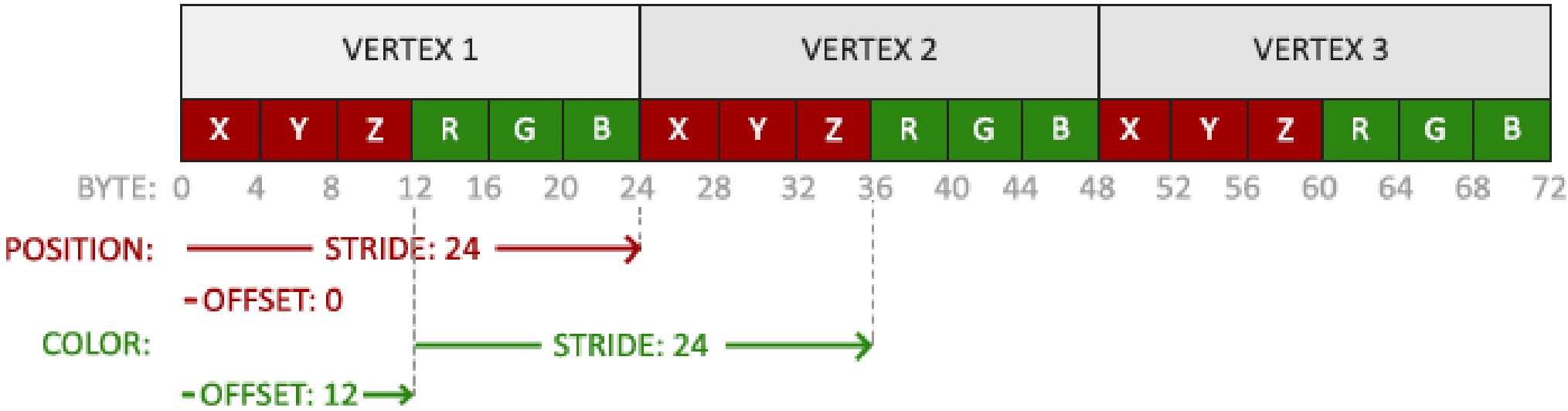
Como se renderizan las figuras?

Generación de buffers

```
# VAO, VBO and EBO and for the shape
vao = glGenVertexArrays(1)
vbo = glGenBuffers(1)
ebo = glGenBuffers(1)
```

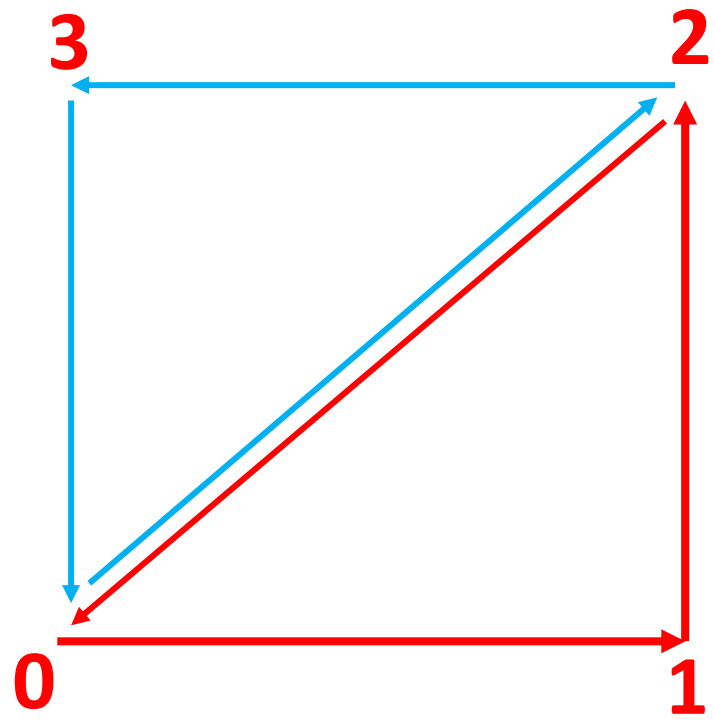
- vao, vbo y ebo son simplemente números enteros que OpenGL utiliza para referenciar memoria reservada en la GPU

Especificando vértices



Especificando vértices (e índices)

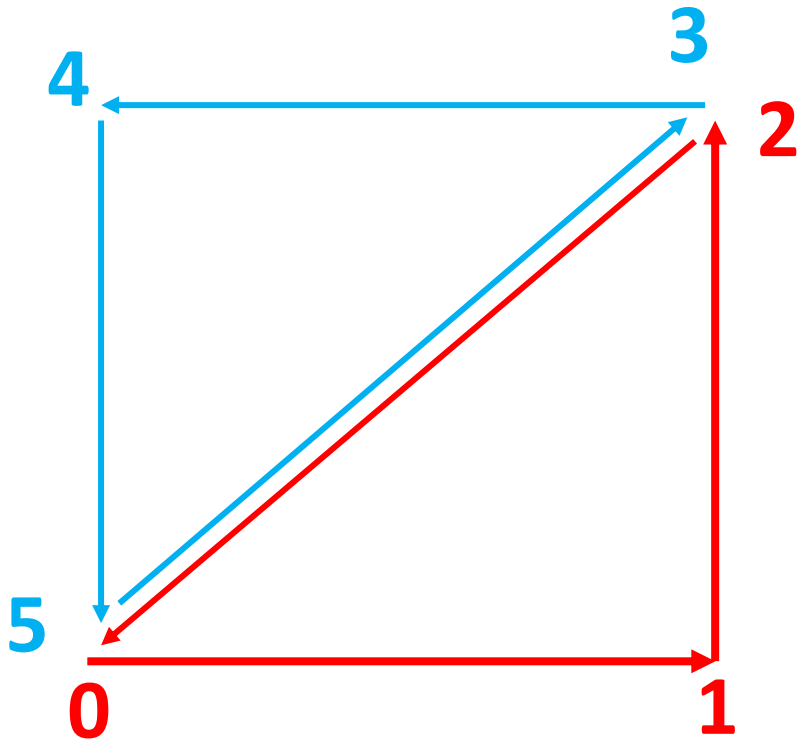
```
vertexData = np.array([
        # positions       colors
        -0.5, -0.5, 0.0, 1.0, 0.0, 0.0,  #0
         0.5, -0.5, 0.0, 0.0, 1.0, 0.0,  #1
         0.5,  0.5, 0.0, 0.0, 0.0, 1.0,  #2
        -0.5,  0.5, 0.0, 1.0, 1.0, 1.0], #3
        dtype = np.float32)
        # Defining connections among vertices
        # We have a triangle every 3 indices specified
        indices = np.array(
                [0, 1, 2,
                2, 3, 0], dtype=np.uint32)
        size = len(indices)
```



Solo si usamos EBO!!

Especificando vértices (sin índices)

```
vertexData = np.array([
        # positions       colors
        -0.5, -0.5, 0.0,  1.0, 0.0, 0.0,  #0
         0.5, -0.5, 0.0,  0.0, 1.0, 0.0,  #1
         0.5,  0.5, 0.0,  0.0, 0.0, 1.0,  #2

         0.5,  0.5, 0.0,  0.0, 0.0, 1.0,  #3
        -0.5,  0.5, 0.0,  1.0, 1.0, 1.0,  #4
        -0.5, -0.5, 0.0,  1.0, 0.0, 0.0], #5
        dtype = np.float32)
```



Enviando la data a memoria GPU

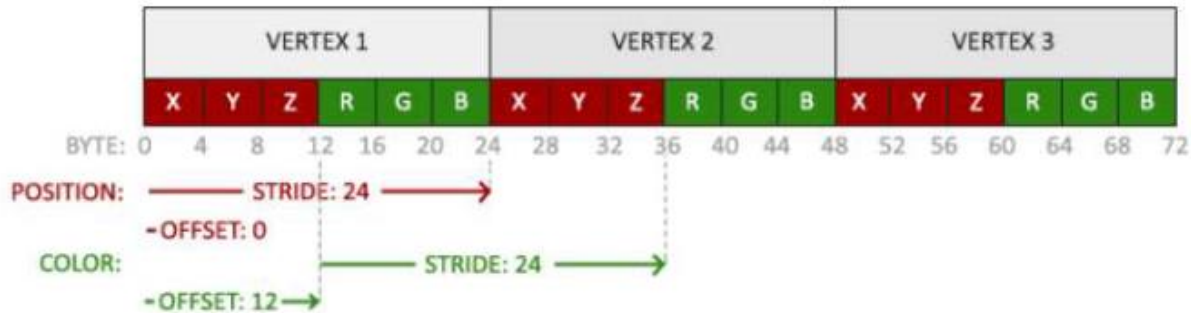
```
# Vertex data must be attached to a Vertex Buffer Object (VBO)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBufferData(GL_ARRAY_BUFFER, len(vertexData) * SIZE_IN_BYTES, vertexData, GL_STATIC_DRAW)

# Connections among vertices are stored in the Elements Buffer Object (EBO)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, len(indices) * SIZE_IN_BYTES, indices, GL_STATIC_DRAW)
```

32 bits en bytes → 4

Que tan frecuentemente cambiará esta información? →

- GL_STATIC_DRAW
- GL_STREAM_DRAW
- GL_DYNAMIC_DRAW

Especificando un VAO

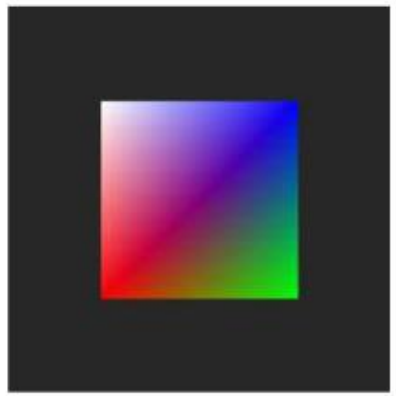



```
# Binding the proper buffers
glBindVertexArray(vao)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo)

# Setting up the location of the attributes position and color from the VBO
# A vertex attribute has 3 integers for the position (each is 4 bytes),
# and 3 numbers to represent the color (each is 4 bytes),
# Henceforth, we have 3*4 + 3*4 = 24 bytes
position = glGetAttribLocation(shaderProgram, "position")
glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(0))
glEnableVertexAttribArray(position)

color = glGetAttribLocation(shaderProgram, "color")
glVertexAttribPointer(color, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(12))
glEnableVertexAttribArray(color)
```

Dibujando con OpenGL

- Utilizando un VBO y un EBO

```
glBindVertexArray(vao)
glDrawElements(GL_TRIANGLES, len(indices), GL_UNSIGNED_INT, None)
```



- Sin usar EBO

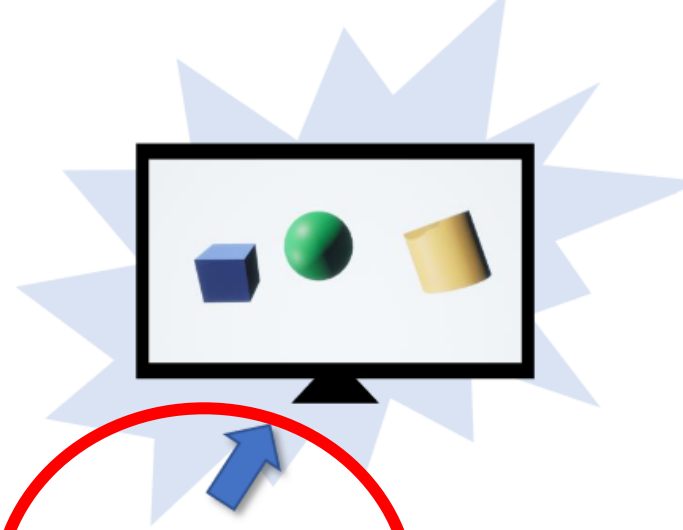
```
glBindVertexArray(vao)
glDrawArrays(GL_TRIANGLES, 0, 3)
```

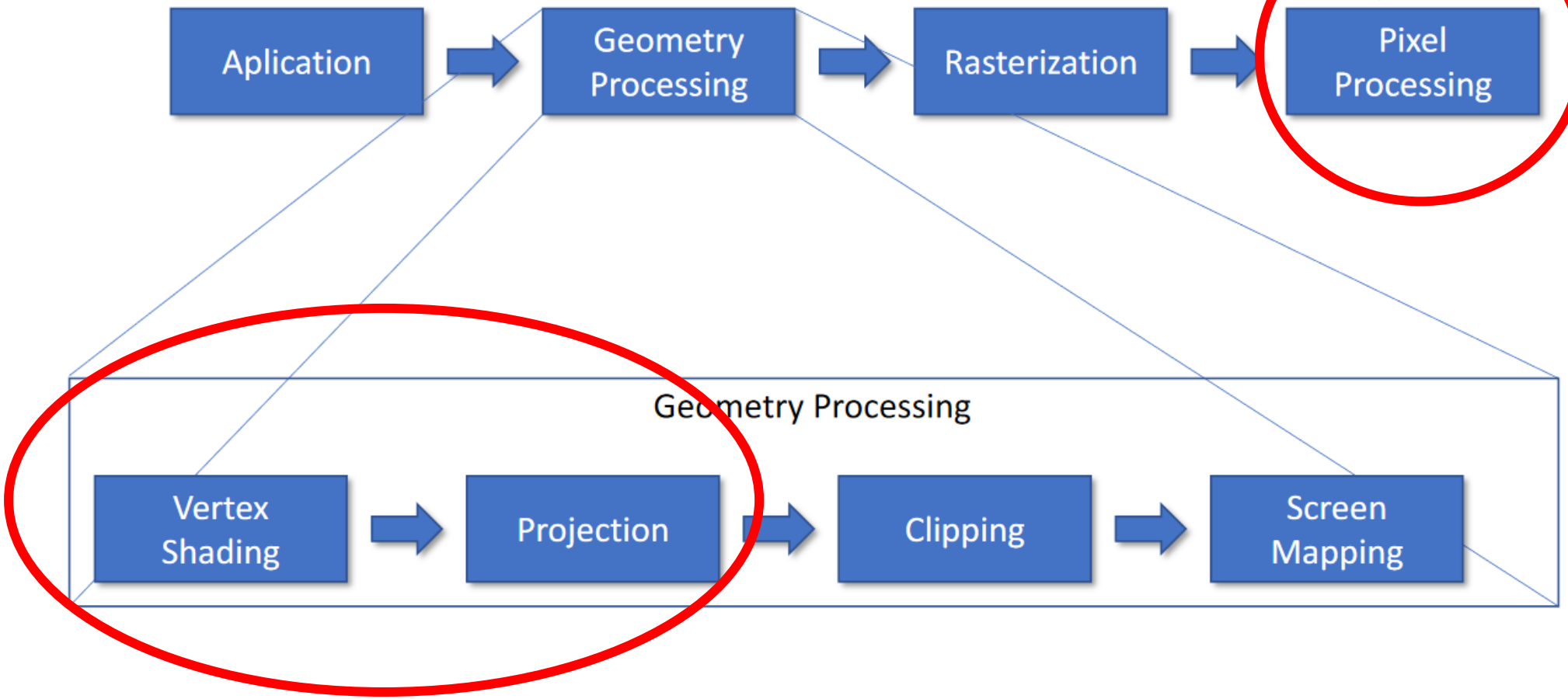
Shaders?

Shaders?

Shaders

Graphics Rendering Pipeline





Vertex Shader

- GPU procesa cada vértice procesando la posición entregada y la información del color

```
vertex_shader = """
#version 130
in vec3 position;
in vec3 color;
out vec3 newColor;
void main()
{
        gl_Position = vec4(position, 1.0f);
        newColor = color;
}
"""
```

Fragment Shader

```
fragment_shader =
""" #version 130
in vec3 newColor;
out vec4 outColor;
void main()
{
        outColor = vec4(newColor, 1.0f);
}
"""
```

- GPU procesa cada fragmento o candidato a pixel, asignándole el color final.

- P1) Copie de los ejemplos el código de ex_quad.py y modifique el programa para que el render se realice sin el ebo.
- Use las clases y funciones provistas en los códigos de la carpeta grafica para refinar su programa

- P2) Cree una función que entregue una figura en GPU del cielo, recibiendo como parámetros la altura inicial y final. Especialice esta función para que reciba como parámetros los colores a interpolar

createSky(y0, yf)

```
vertices = [
# positions    colors
-1.0, y0, 0.0, 0.0, 1.0, 1.0,
 1.0, y0, 0.0, 0.0, 1.0, 1.0,
 1.0, yf, 0.0, 0.8, 1.0, 1.0,
-1.0, yf, 0.0, 0.8, 1.0, 1.0]

indices = [0, 1, 2,
           2, 3, 0]
return Shape(vertices, indices)
```

- P3) Cree una función que dibuje un triangulo simulando una montaña que reciba de parámetro dos colores. Realice otra función que también reciba de parámetros las coordenadas de los vértices
- Junto a la pregunta anterior dibuje un paisaje simple.