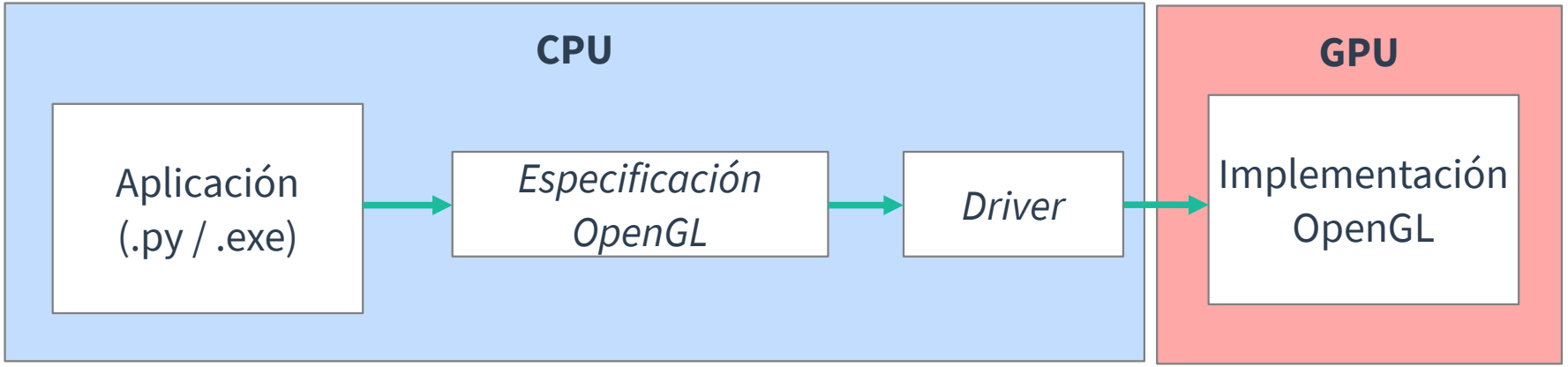
Auxiliar 2

OpenGL y Transformaciones

CC3501 Modelación y Computación Gráfica para Ingenieros
Primavera 2023
Profesor: Iván Sipiran
Auxiliar: Ariel Riveros

OpenGL

API para interactuar con la GPU



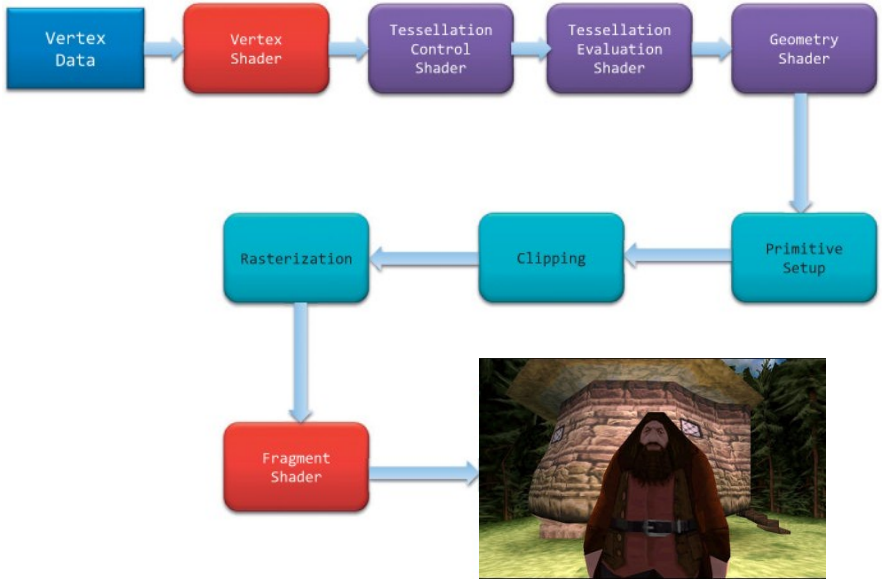
OpenGL

¿Qué era Pyglet de nuevo?

Windowing System

Ventanas
Input de Usuario
Carga de archivos
Wrapping de gráficos

OpenGL

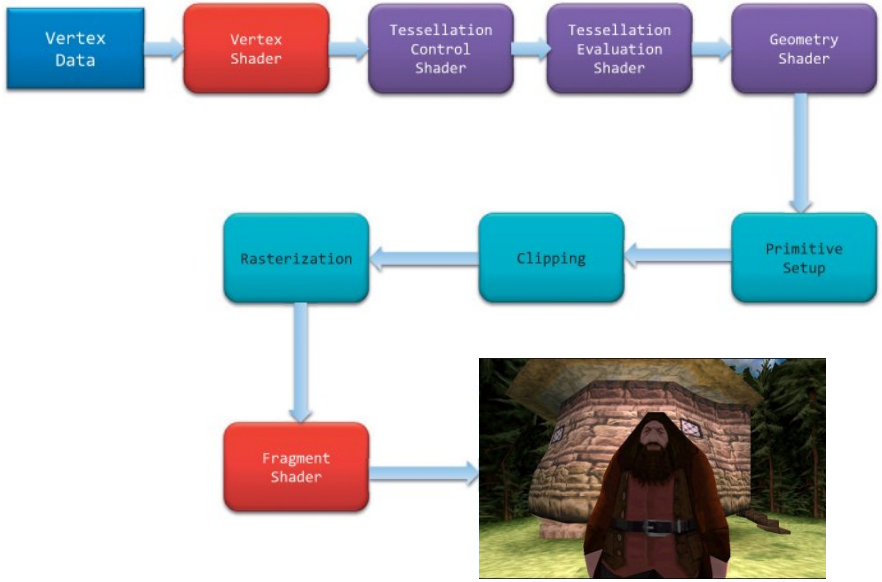


Creación del Pipeline:

En Pyglet

```
# Compilación de shaders
vert_shader = pyglet.graphics.shader.Shader(vertex_source_code, "vertex")
frag_shader = pyglet.graphics.shader.Shader(fragment_source_code, "fragment")
# Creación del pipeline
pipeline = pyglet.graphics.shader.ShaderProgram(vert_shader, frag_shader)
```

OpenGL



Creación del Pipeline:

En OpenGL

```
# Compilación de shaders
vert_shader = glCreateShader(GL_VERTEX_SHADER)  # Creo un vertex shader
glShaderSource(vert_shader, vertex_source)  # Le asigno el código fuente
glCompileShader(vert_shader)    # Compilo el shader

frag_shader = glCreateShader(GL_FRAGMENT_SHADER)    # Creo un fragment shader
glShaderSource(frag_shader, fragment_source)    # Le asigno el código fuente
glCompileShader(frag_shader)    # Compilo el shader

# Creación del pipeline
pipeline = glCreateProgram()    # Creo un programa de shaders
glAttachShader(pipeline, vert_shader)   # Adjunto el vertex shader
glAttachShader(pipeline, frag_shader)   # Adjunto el fragment shader
glLinkProgram(pipeline) # Linkeo el programa
```

OpenGL

Vertex Data:

	Posición (float)			Color (float)		
	x	y	z	r	g	b
0	-0,5	-0,5	0,0	1,0	0,0	0,0
1	0,5	-0,5	0,0	0,0	1,0	0,0
2	0,0	0,5	0,0	0,0	0,0	1,0

OpenGL

	Posición (float)			Color (float)		
	x	y	z	r	g	b
0	-0,5	-0,5	0,0	1,0	0,0	0,0
1	0,5	-0,5	0,0	0,0	1,0	0,0
2	0,0	0,5	0,0	0,0	0,0	1,0

VertexBuffer:

Vértice 0						Vértice 1						Vértice 2					
x	y	z	r	g	b	x	y	z	r	g	b	x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0	0,5	-0,5	0,0	0,0	1,0	0,0	0,0	0,5	0,0	0,0	0,0	1,0

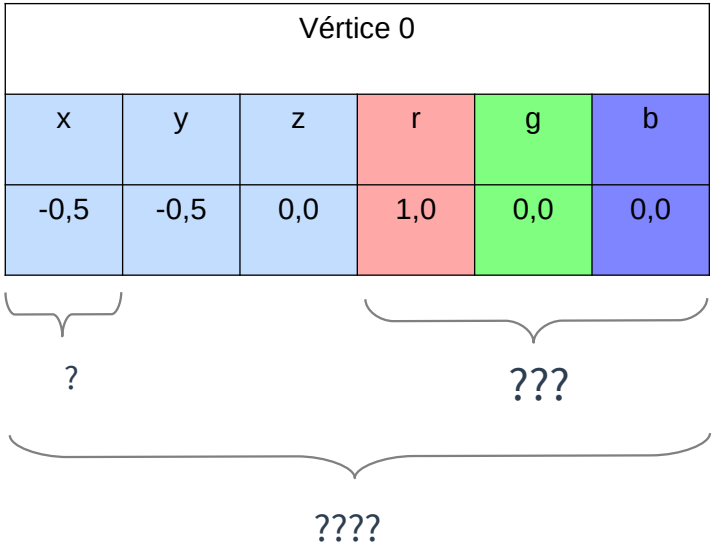
OpenGL

Vértice 0						Vértice 1						Vértice 2					
x	y	z	r	g	b	x	y	z	r	g	b	x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0	0,5	-0,5	0,0	0,0	1,0	0,0	0,0	0,5	0,0	0,0	0,0	1,0

```
# Creación de un (1) buffer
vbo = glGenBuffers(1)
# Seleccionar buffer como un array
glBindBuffer(GL_ARRAY_BUFFER, vbo)
# Transferir datos al buffer
glBufferData(GL_ARRAY_BUFFER, len(vertex_data) * 4, vertex_data, GL_STATIC_DRAW)
```

OpenGL

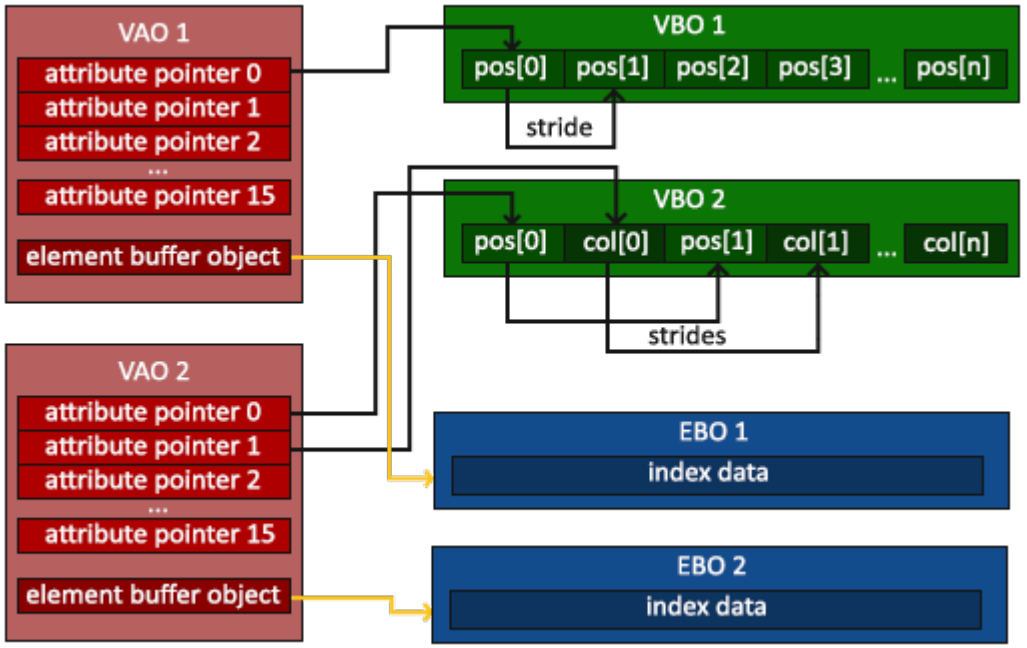
El GPU no sabe qué significa cada valor en el buffer
Hay que describirlo explícitamente



OpenGL

Vertex Array Object

Se usan para describir vértices
No almacenan datos, sólo describen



OpenGL

Vértice 0					
x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0
0	4	8	12	16	20

4 bytes

stride = 24 bytes

```
vao = glGenVertexArrays(1) # Genero un VAO
glBindVertexArray(vao) # Lo selecciono

vbo = glGenBuffers(1)  # Genero un VBO
glBindBuffer(GL_ARRAY_BUFFER, vbo) # Lo selecciono
glBufferData(GL_ARRAY_BUFFER, len(vertex_data) * SIZE_IN_BYTES, vertex_data, GL_STATIC_DRAW)
# 3 floats + 3 floats
stride = 3 * SIZE_IN_BYTES + 3 * SIZE_IN_BYTES

# Posiciones
position = glGetAttribLocation(pipeline, "position")
position_size = 3
glVertexAttribPointer(position, position_size, GL_FLOAT, GL_FALSE, stride, ctypes.c_void_p(0))
glEnableVertexAttribArray(0)

# Colores
color = glGetAttribLocation(pipeline, "color")
color_size = 3
glVertexAttribPointer(color, color_size, GL_FLOAT, GL_FALSE, stride, ctypes.c_void_p(12))
glEnableVertexAttribArray(1)
```

OpenGL

Vértice 0					
x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0
0	4	8	12	16	20

4 bytes

stride = 24 bytes

```
vao = glGenVertexArrays(1) # Genero un VAO
glBindVertexArray(vao) # Lo selecciono

vbo = glGenBuffers(1)  # Genero un VBO
glBindBuffer(GL_ARRAY_BUFFER, vbo) # Lo selecciono
glBufferData(GL_ARRAY_BUFFER, len(vertex_data) * SIZE_IN_BYTES, vertex_data, GL_STATIC_DRAW)
# 3 floats + 3 floats
stride = 3 * SIZE_IN_BYTES + 3 * SIZE_IN_BYTES

# Posiciones
position = glGetAttribLocation(pipeline, "position")
position_size = 3
glVertexAttribPointer(position, position_size, GL_FLOAT, GL_FALSE, stride, ctypes.c_void_p(0))
glEnableVertexAttribArray(0)

# Colores
color = glGetAttribLocation(pipeline, "color")
color_size = 3
glVertexAttribPointer(color, color_size, GL_FLOAT, GL_FALSE, stride, ctypes.c_void_p(12))
glEnableVertexAttribArray(1)
```

OpenGL

Vértice 0					
x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0
0	4	8	12	16	20

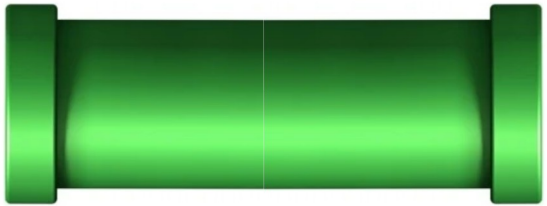
Pyglet hace esto por mi

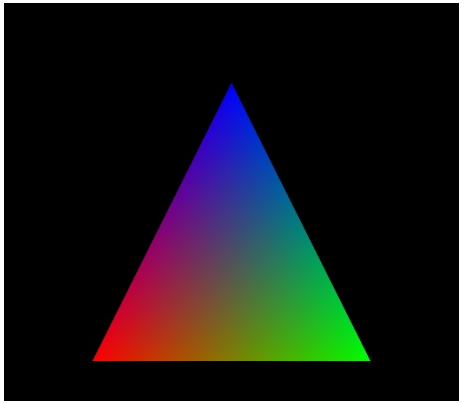
```
if index_data is not None:
    gpu_data = pipeline.vertex_list_indexed(len(position_data) // 3, GL.GL_TRIANGLES, index_data)
else:
    gpu_data = pipeline.vertex_list(len(position_data) // 3, GL.GL_TRIANGLES)

gpu_data.position[:] = position_data
gpu_data.color[:] = color_data
```

OpenGL

Vértice 0						Vértice 1						Vértice 2					
x	y	z	r	g	b	x	y	z	r	g	b	x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0	0,5	-0,5	0,0	0,0	1,0	0,0	0,0	0,5	0,0	0,0	0,0	1,0



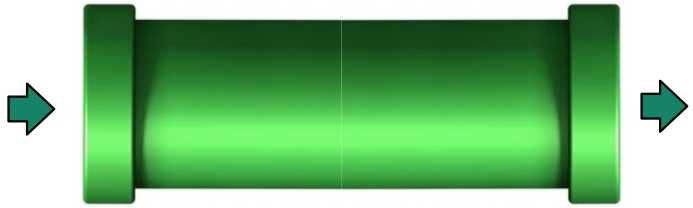


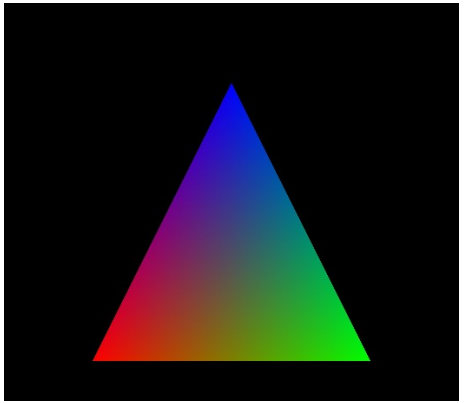
```
glBindVertexArray(vao) # Selecciono el VAO
if index_data is not None:
    glDrawElements(mode, len(vertex_data), GL_UNSIGNED_INT, None)  # Dibujo con índices
else:
    glDrawArrays(mode, 0, len(vertex_data))    # Dibujo sin índices

glBindVertexArray(0)    # Deselecciono el VAO
```

OpenGL

Vértice 0						Vértice 1						Vértice 2					
x	y	z	r	g	b	x	y	z	r	g	b	x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0	0,5	-0,5	0,0	0,0	1,0	0,0	0,0	0,5	0,0	0,0	0,0	1,0



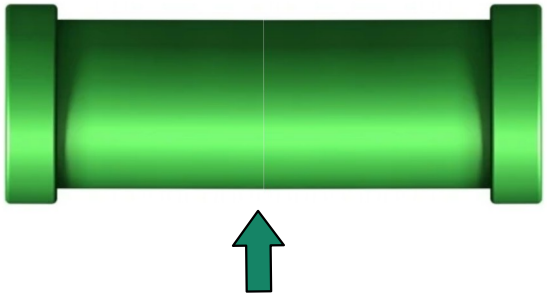


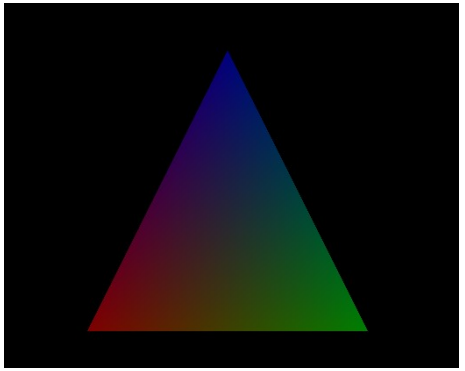
¿Y si quiero modificar algunos valores del triángulo?

OpenGL

Cambiar el buffer es costoso
Se utilizan **Uniforms**

Vértice 0						Vértice 1						Vértice 2					
x	y	z	r	g	b	x	y	z	r	g	b	x	y	z	r	g	b
-0,5	-0,5	0,0	1,0	0,0	0,0	0,5	-0,5	0,0	0,0	1,0	0,0	0,0	0,5	0,0	0,0	0,0	1,0





Intensidad = 0,5

Uniform: Variable global de un Shader

OpenGL

Pueden definirse en el Vertex Shader o en el Fragment Shader

uniform [tipo] [nombre] = [valor por defecto]

```
#version 330

in vec3 position;
in vec3 color;

uniform float u_intensity = 1.0f;

out vec3 fragColor;

void main()
{
    fragColor = color * u_intensity;
    gl_Position = vec4(position, 1.0f);
}

```

OpenGL

En OpenGL hay que ser explícito con los tipos

```
def set_uniform(self, name, value, type):
    location = glGetUniformLocation(self.program, name)

    if location == -1:
        print(f"Warning: Uniform {name} does not exist")
        return

    if type == "matrix":
        glUniformMatrix4fv(location, 1, GL_TRUE, value)
    elif type == "float":
        glUniform1f(location, value)
```

En Pyglet no tanto

```
def set_uniform(self, name, value, type):
    uniform = self[name]
    if uniform is None:
        print(f"Warning: Uniform {name} does not exist")
        return

    if type == "matrix":
        self[name] = np.reshape(value, (16, 1), order="F")
    elif type == "float":
        self[name] = value
```

OpenGL

En OpenGL hay que ser explícito con los tipos

glUniform1f	glUniform3iv
glUniform1fv	glUniform3ui
glUniform1i	glUniform3uiv
glUniform1iv	glUniform4f
glUniform1ui	glUniform4fv
glUniform1uiv	glUniform4i
glUniform2f	glUniform4iv
glUniform2fv	glUniform4ui
glUniform2i	glUniform4uiv
glUniform2iv	glUniformMatrix2fv
glUniform2ui	glUniformMatrix2x3fv
glUniform2uiv	glUniformMatrix2x4fv
glUniform3f	glUniformMatrix3fv
glUniform3fv	glUniformMatrix3x2fv
glUniform3i	glUniformMatrix3x4fv
	glUniformMatrix4fv
	glUniformMatrix4x2fv
	glUniformMatrix4x3fv

En Pyglet no tanto

```
def set_uniform(self, name, value, type):
    uniform = self[name]
    if uniform is None:
        print(f"Warning: Uniform {name} does not exist")
        return

    if type == "matrix":
        self[name] = np.reshape(value, (16, 1), order="F")
    elif type == "float":
        self[name] = value
```

OpenGL

Conceptos importantes

Vertex Buffer (vbo): Arreglo con datos de vértices que se le entrega a la GPU
Element Buffer (ebo): Arreglo de índices asociado a un Vertex Buffer
Vertex Array Object (vao): Descripción de vértices
Uniforms: Variables globales de un shader

OpenGL

Ejercicio 1

Cambiar la intensidad de un triángulo usando uniforms

-¿Qué pasa cuando hay distintas figuras?

- ¿Qué pasa cuando hay distintos pipelines?

Transformaciones

Transformaciones

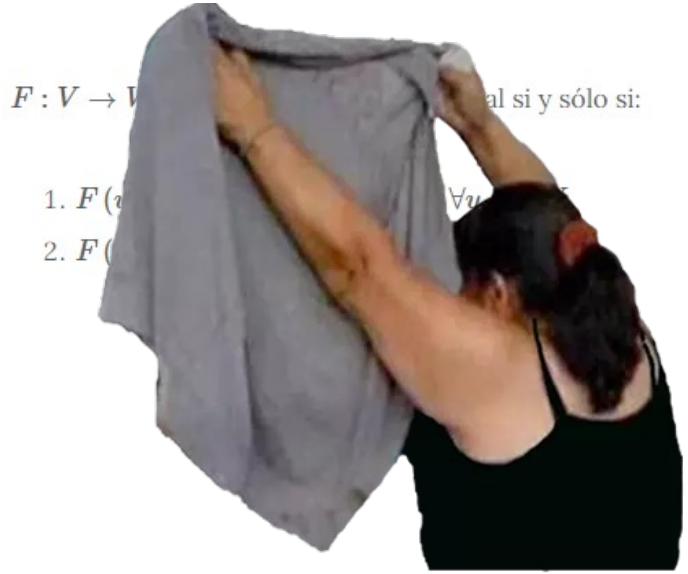
Transformación Lineal

$F : V \to W$ es una transformación lineal si y sólo si:

1. $F(u+v) = F(u) + F(v) \qquad \forall u, v \in V$
2. $F(k.v) = k.F(v) \qquad \forall v \in V, \quad \forall k \in \mathbb{R}$

Transformaciones

Transformación Lineal

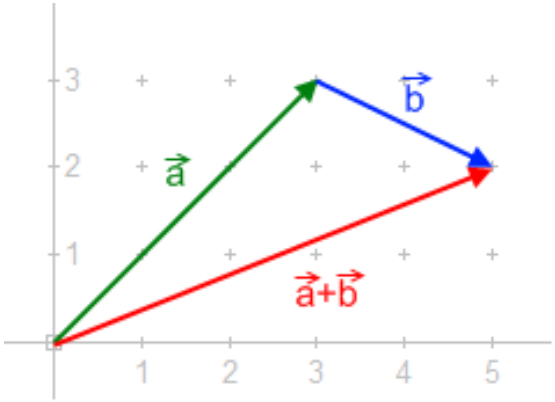


De manera práctica en computación gráfica
qué nos importa:

-Función que modifica un vector
-Se puede encadenar con otras transformaciones
-No es conmutativa, el orden de aplicación importa

Transformaciones

Traslación

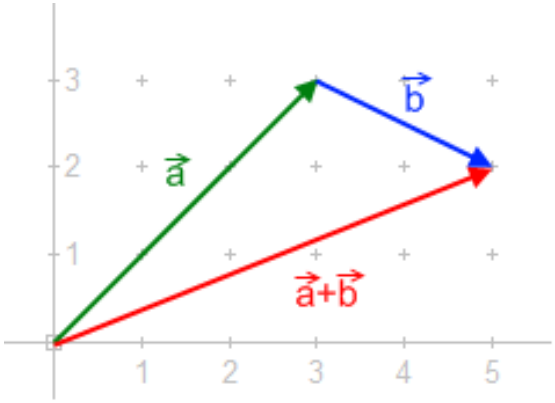


Suma de vectores

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$$

Transformaciones

Traslación



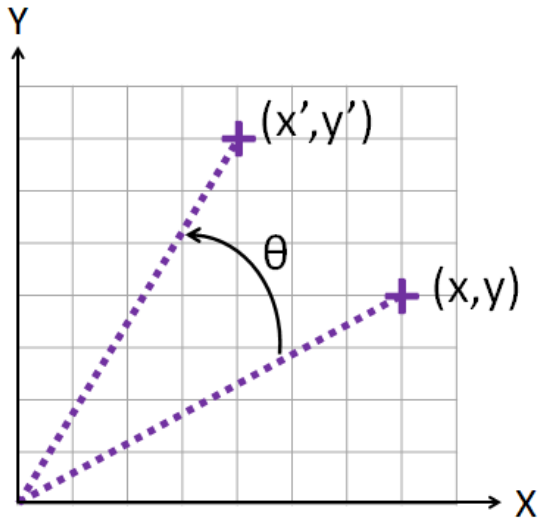
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$$

En coordenadas homogéneas

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Transformaciones

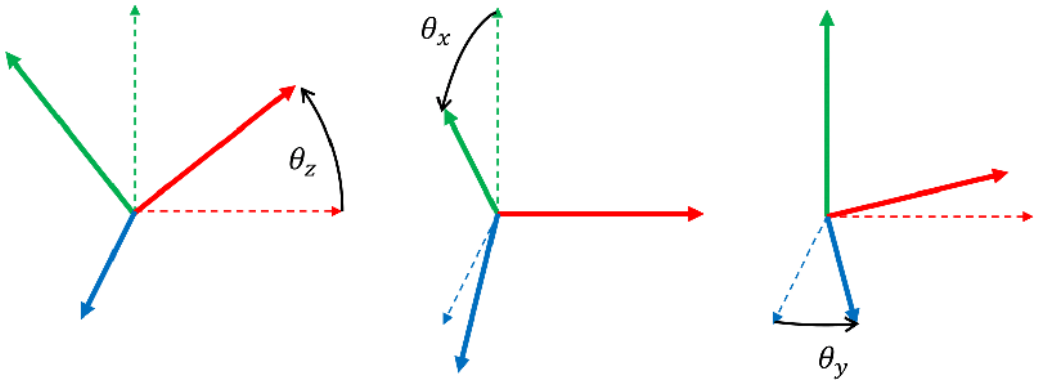
Rotación 2D



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformaciones

Rotación 3D



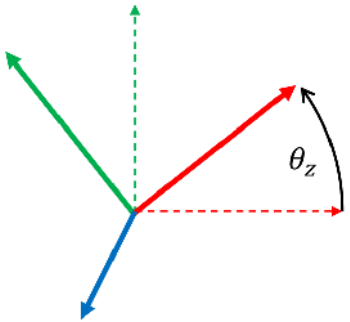
$$\mathbf{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

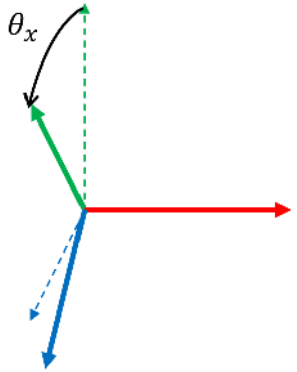
$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

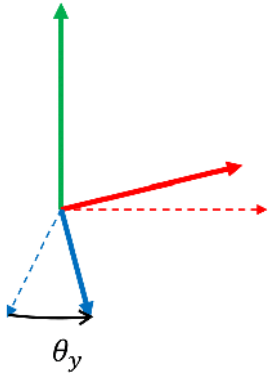
Transformaciones

Rotación 3D





Ángulos Euler



$$R_z(z) = \begin{pmatrix} cos(z) & -sin(z) & 0 & 0 \\ sin(z) & cos(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_x(x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(x) & -sin(x) & 0 \\ 0 & sin(x) & cos(x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_y(y) = \begin{pmatrix} cos(y) & 0 & sin(y) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(y) & 0 & cos(y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$Rot(x, y, z) = R_y(y)R_x(x)R_z(z)$$

Transformaciones

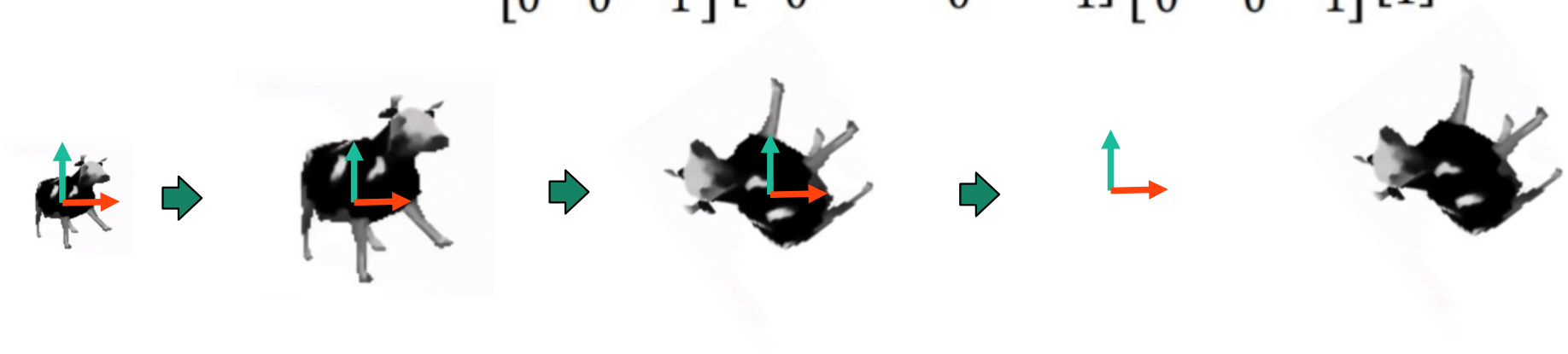
Escala

$$S_v p = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \\ 1 \end{bmatrix}.$$

Transformaciones

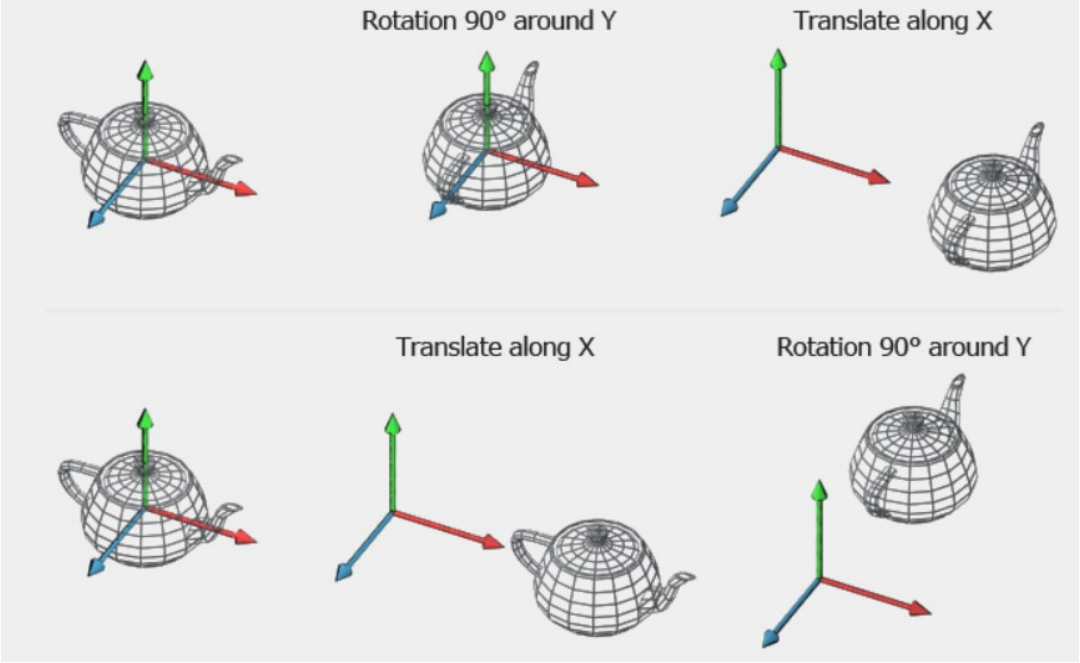
Como las transformaciones son matrices, se pueden encadenar

$$\mathbf{p}' = (\mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S}) \cdot \mathbf{p} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Transformaciones

El orden importa



Transformaciones

¿Cómo se usan en OpenGL?

En el Vertex Shader:

-Se crea un uniform del tipo matriz 4x4

-Cada vértice se multiplica por la matriz de transformación

En la aplicación:

-Se actualiza la posición, rotación y escala de una figura
(hay una librería del curso con las matrices definidas)

-Si hay más de una figura, hay que cambiar el uniform por cada figura

Transformaciones

Ejercicio 2

Usar uniforms para:

-Trasladar un triángulo

-Rotar un triángulo

-Escalar un triángulo

Auxiliar 2

OpenGL y Transformaciones

CC3501 Modelación y Computación Gráfica para Ingenieros
Primavera 2023
Profesor: Iván Sipiran
Auxiliar: Ariel Riveros