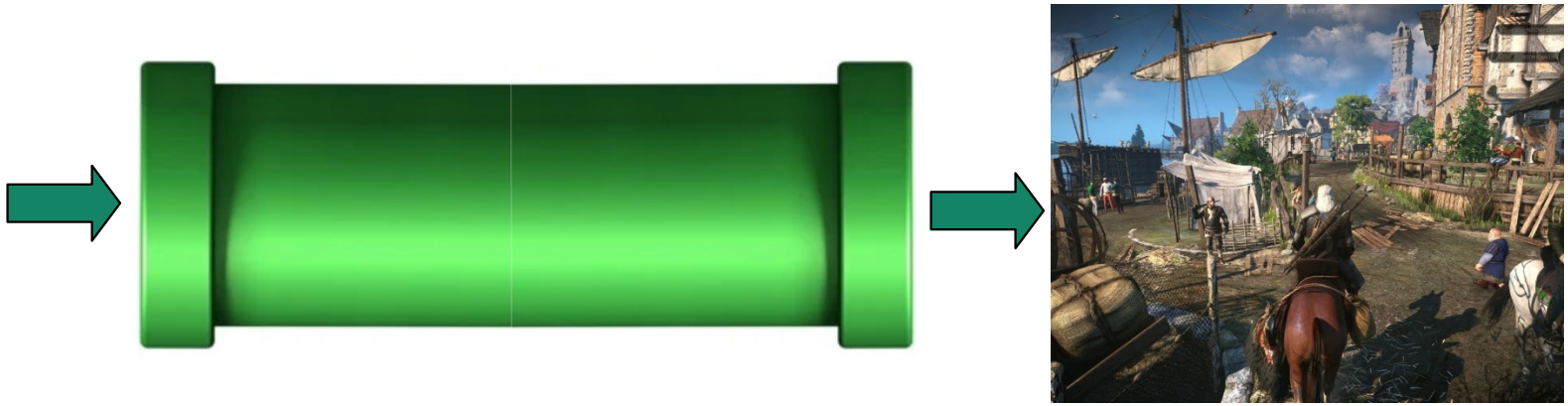
Auxiliar 1

Rendering Pipeline

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

Pipeline gráfico

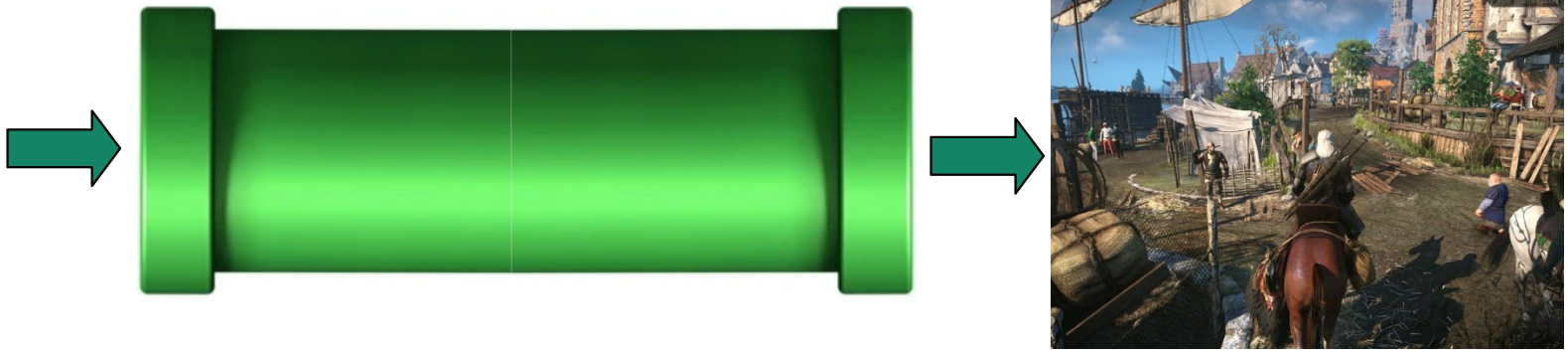
```
v     -5.000000   5.000000    0.000000
v     -5.000000   -5.000000   0.000000
v     5.000000    -5.000000   0.000000
v     5.000000    5.000000    0.000000
vt    -5.000000   5.000000    0.000000
vt    -5.000000   -5.000000   0.000000
vt    5.000000    -5.000000   0.000000
vt    5.000000    5.000000    0.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vp    0.210000    3.590000
vp    0.000000    0.000000
vp    1.000000    0.000000
vp    0.500000    0.500000
```



Pipeline gráfico

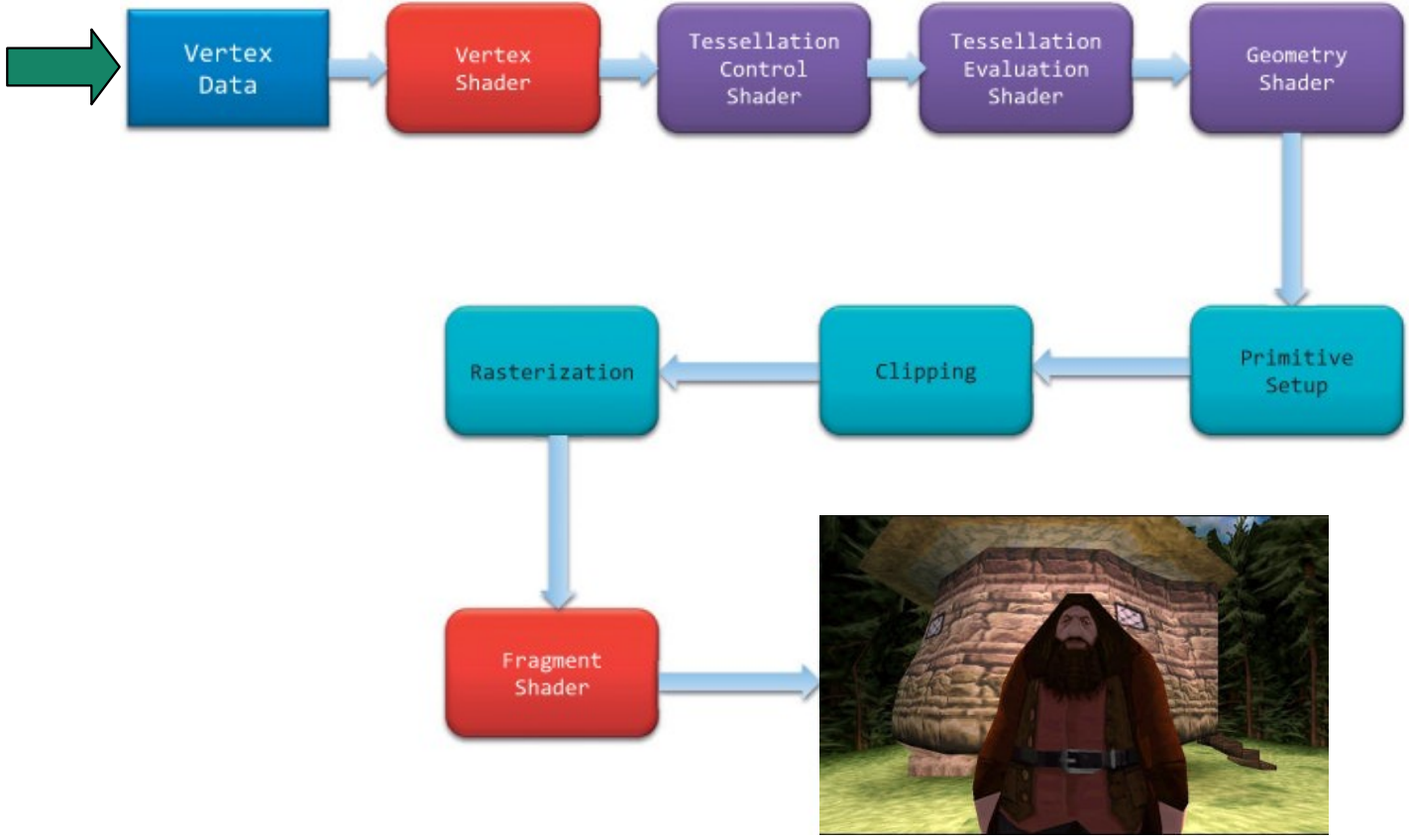
Es un pipeline que al proveerle los recursos necesarios a la GPU (números, archivos, imágenes, etc.) devuelve una imagen que puede ser mostrada en una pantalla

```
v     -5.000000   5.000000    0.000000
v     -5.000000   -5.000000   0.000000
v     5.000000    -5.000000   0.000000
v     5.000000    5.000000    0.000000
vt    -5.000000   5.000000    0.000000
vt    -5.000000   -5.000000   0.000000
vt    5.000000    -5.000000   0.000000
vt    5.000000    5.000000    0.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vn    0.000000    0.000000    1.000000
vp    0.210000    3.590000
vp    0.000000    0.000000
vp    1.000000    0.000000
vp    0.500000    0.500000
```



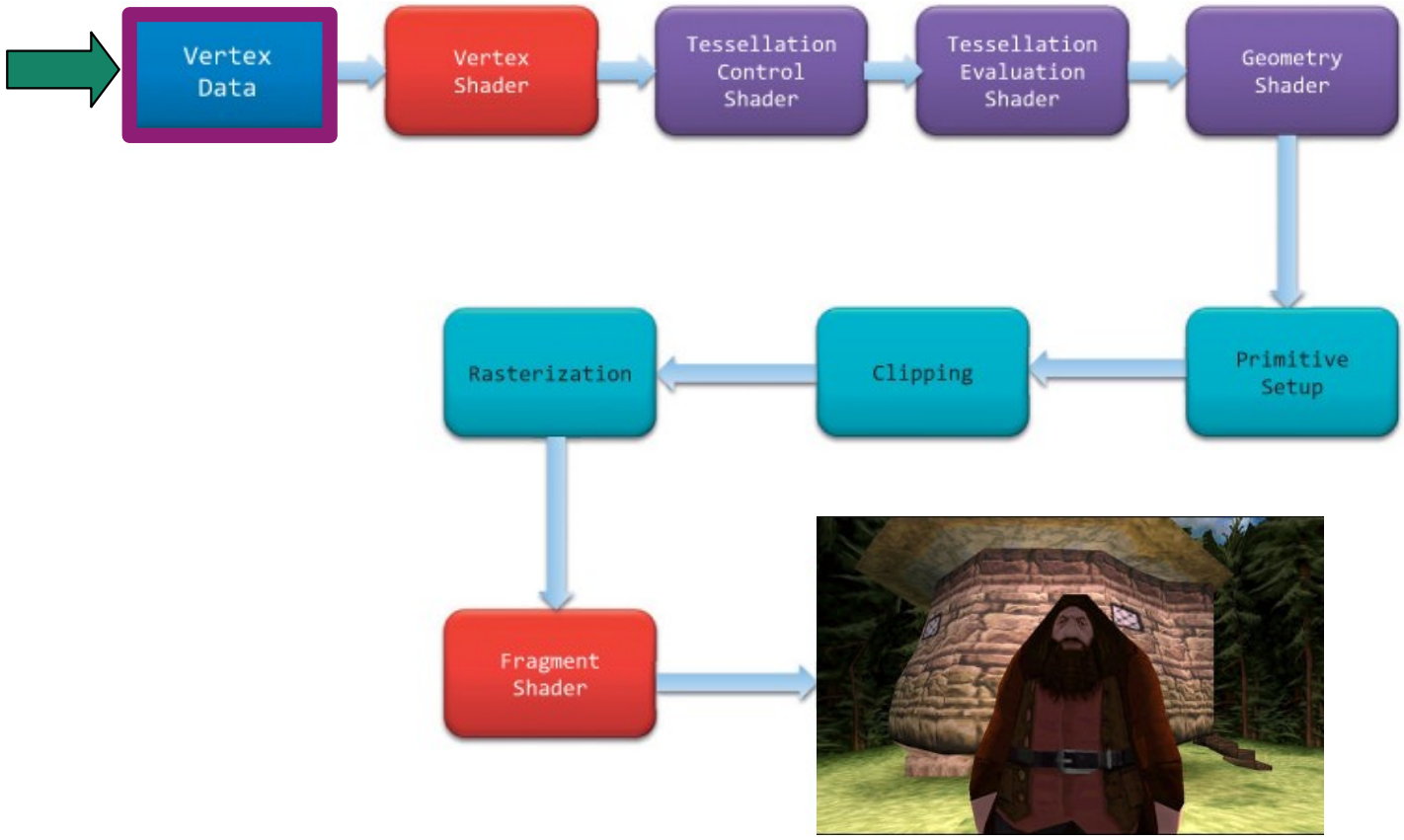
Pipeline gráfico

```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```



Vertex Data

```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```



Vertex Data

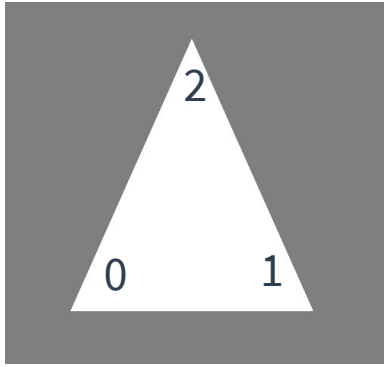
Vértices en CPU

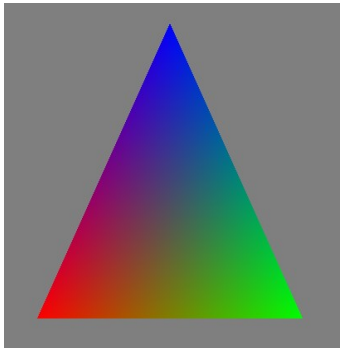
atributo / vértice	Posición (float)		Color (float)			Intensidad
	x	y	r	g	b	(float)
0	-0,5	-0,5	1,0	0,0	0,0	1,0
1	0,5	-0,5	0,0	1,0	0,0	1,0
2	0,0	0,5	0,0	0,0	1,0	1,0

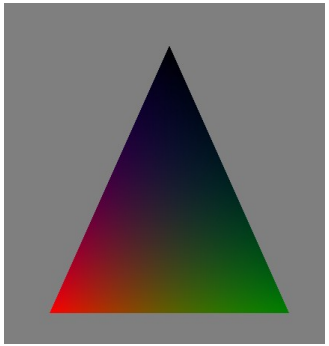
Vertex Data

atributo / vértice	Posición (float)		Color (float)			Intensidad
	x	y	r	g	b	(float)
0	-0,5	-0,5	1,0	0,0	0,0	1,0
1	0,5	-0,5	0,0	1,0	0,0	0,5
2	0,0	0,5	0,0	0,0	1,0	0,0

Cómo se verían
al final del pipeline:

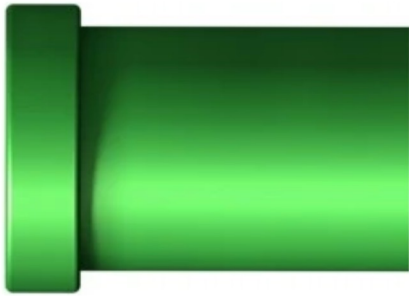






Vertex Data

atributo / vértice	Posición (float)		Color (float)			Intensidad
	x	y	r	g	b	(float)
0	-0,5	-0,5	1,0	0,0	0,0	1,0
1	0,5	-0,5	0,0	1,0	0,0	0,5
2	0,0	0,5	0,0	0,0	1,0	0,0

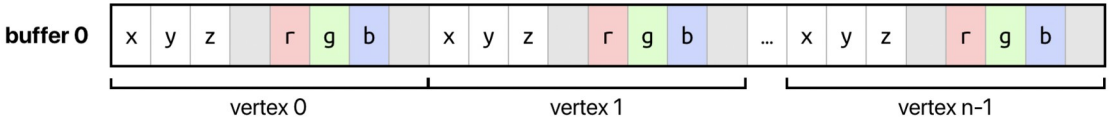


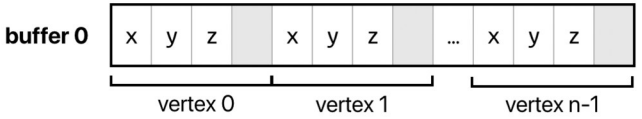
Vértices CPU != Vértices en GPU

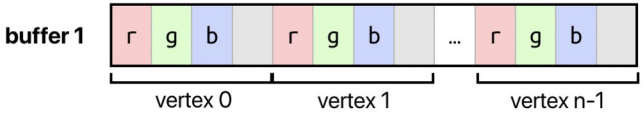
Vertex Data

Vértices en GPU: VertexBuffer

atributo / vértice	Posición (float)		Color (float)			Intensidad
	x	y	r	g	b	(float)
0	-0,5	-0,5	1,0	0,0	0,0	1,0
1	0,5	-0,5	0,0	1,0	0,0	0,5
2	0,0	0,5	0,0	0,0	1,0	0,0

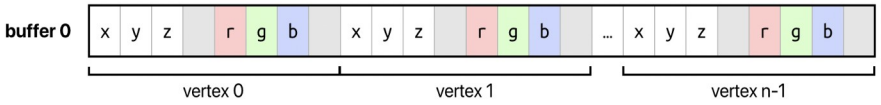


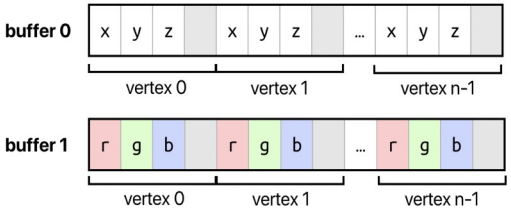




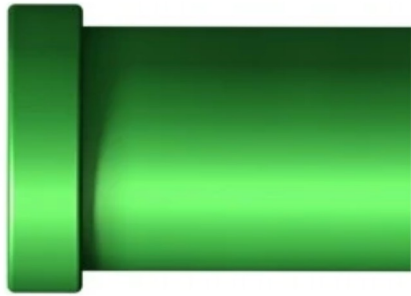
Los buffers sirven para transferir datos en memoria de CPU a la memoria de GPU

Vertex Data

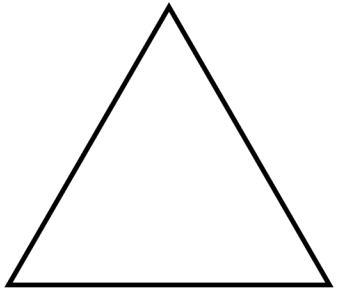

buffer 0
x y z r g b x y z r g b ... x y z r g b
vertex 0
vertex 1
vertex n-1

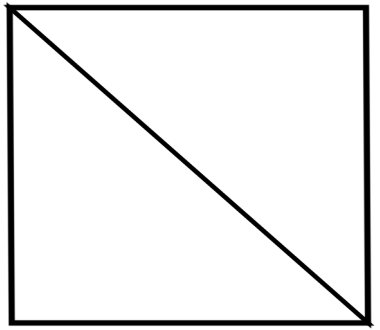

buffer 0
x y z x y z ... x y z
vertex 0
vertex 1
vertex n-1
buffer 1
r g b r g b ... r g b
vertex 0
vertex 1
vertex n-1


SIM, SIM AMIGÃO



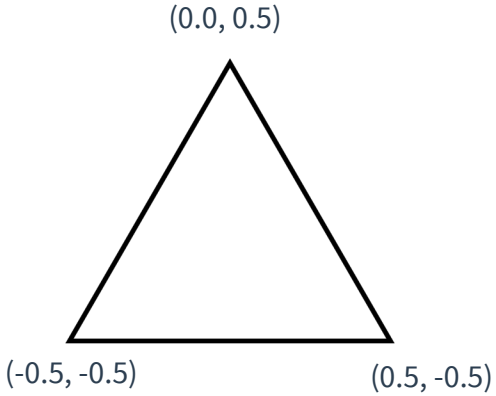
Extra: Index Buffer

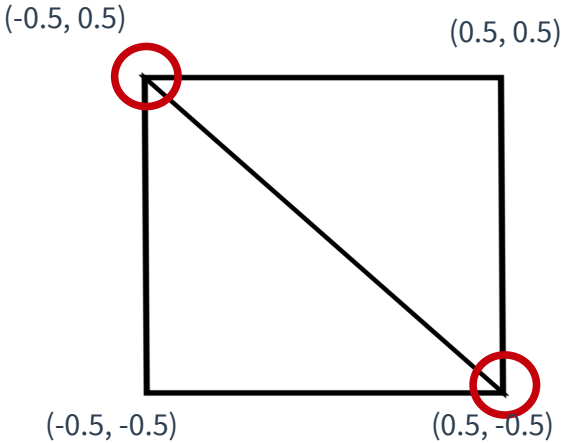




Extra: Index Buffer

Sin índices



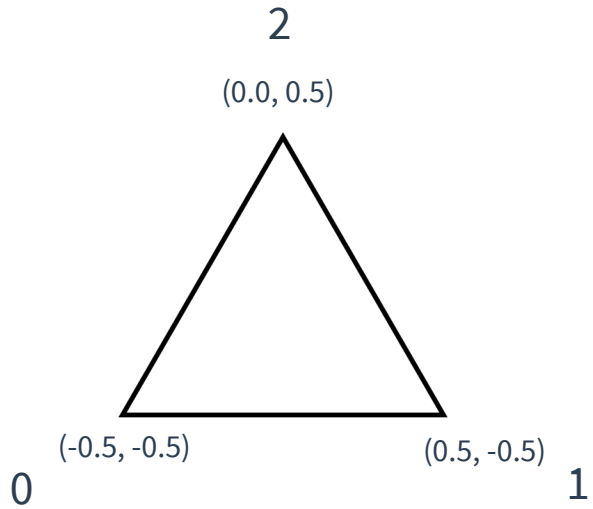


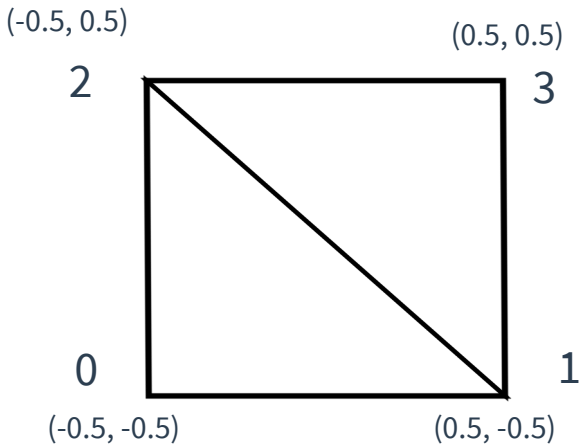
Vertex Buffer = [-0.5, -0.5, 0.5, -0.5, 0.0, 0.5]

Vertex Buffer = [-0.5, -0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5]

Extra: Index Buffer

Con índices





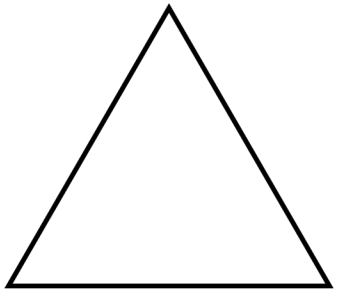
Vertex Buffer = [-0.5, -0.5, 0.5, -0.5, 0.0, 0.5]

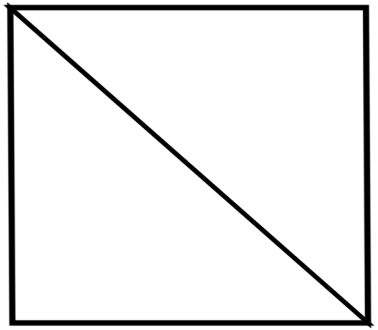
Index Buffer = [0, 1, 2]

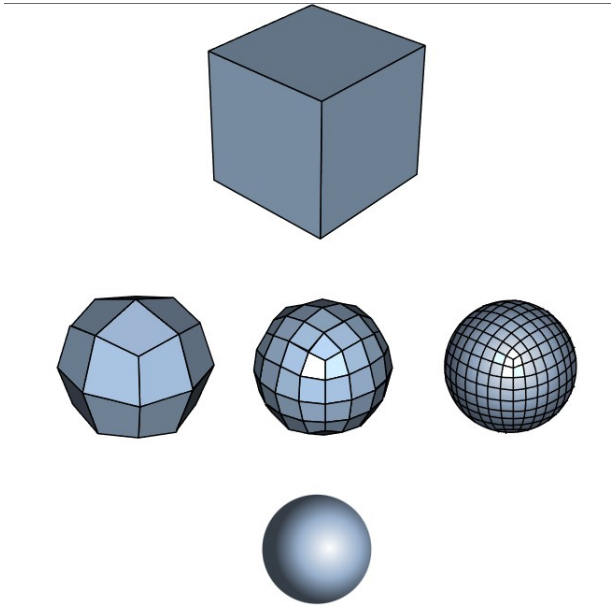
Vertex Buffer = [-0.5, -0.5, 0.5, -0.5, -0.5, -0.5, 0.5, 0.5]

Index Buffer = [0, 1, 2, 1, 3, 2]

Extra: Index Buffer



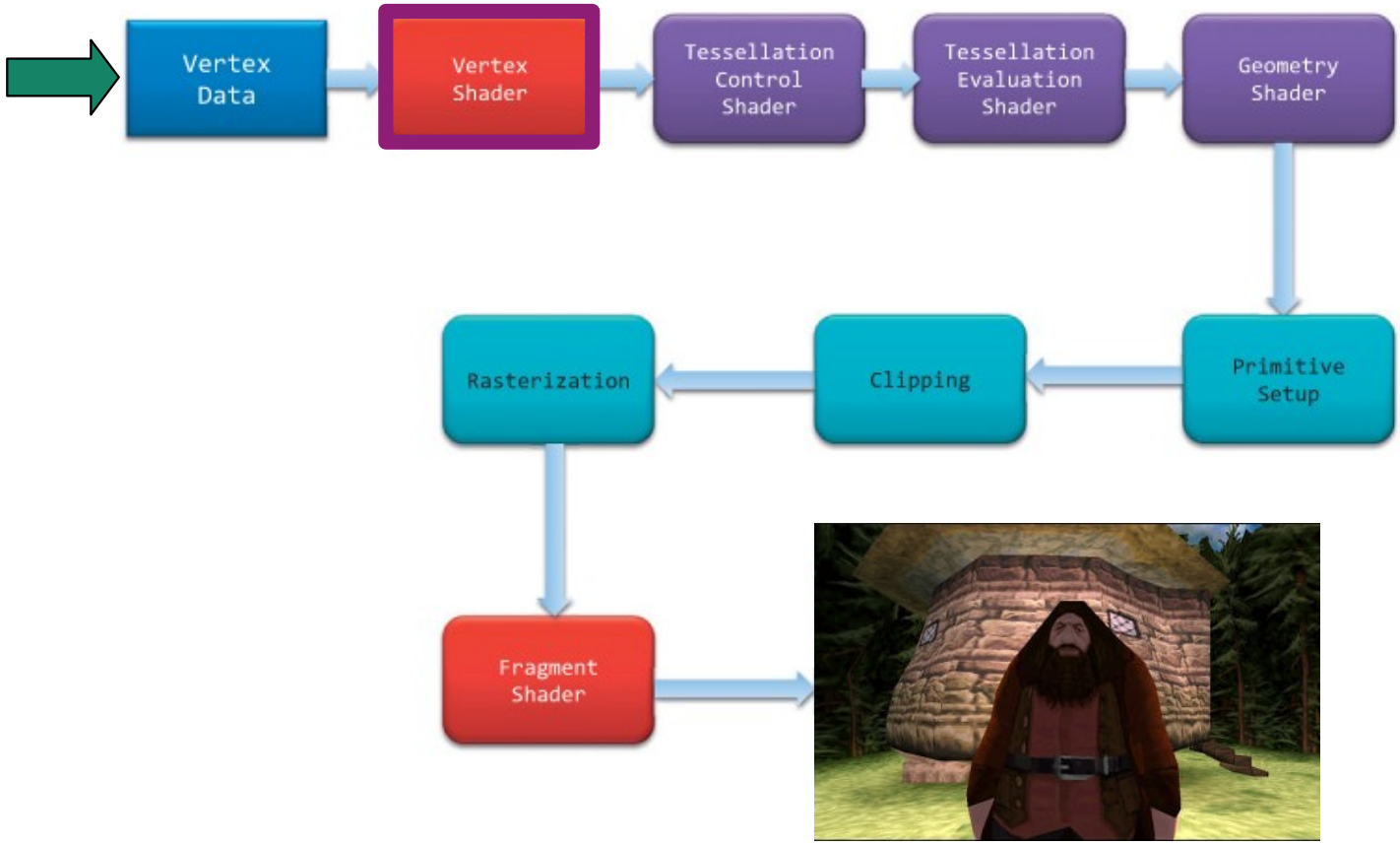




Mientras mayor la definición, mayor posible repetición de vértices

Vertex Shader

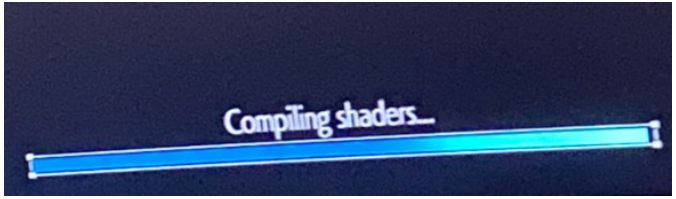
```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```

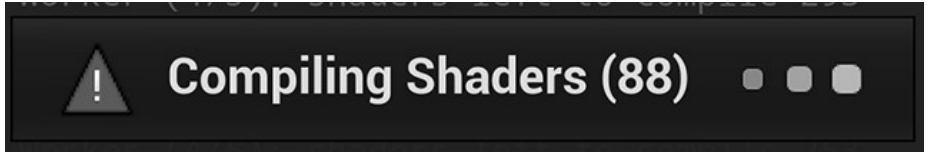


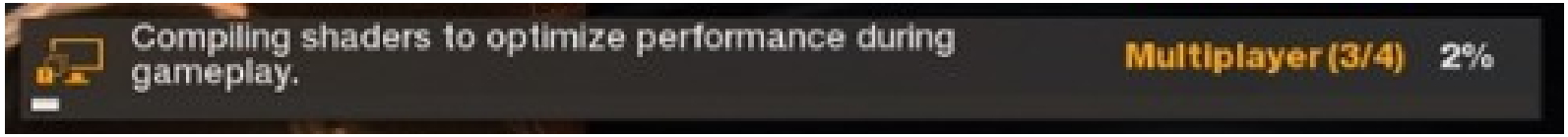
Vertex Shader

Qué es Shader?

Programas que se **ejecutan** en la GPU



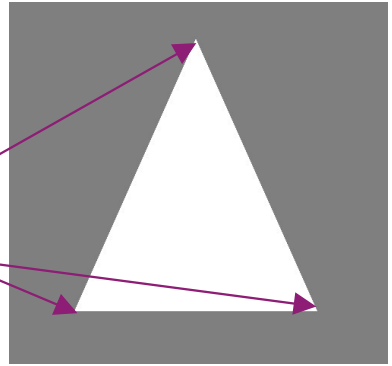




Vertex Shader

Vertex Shader Se ejecuta por cada vértice

atributo / vértice	Posición (float)		Color (float)			Intensidad
	x	y	r	g	b	(float)
0	-0,5	-0,5	1,0	0,0	0,0	1,0
1	0,5	-0,5	0,0	1,0	0,0	0,5
2	0,0	0,5	0,0	0,0	1,0	0,0



Toma como *input un vértice* del vertex buffer
Retorna una posición que el pipeline utiliza para ser trabajada

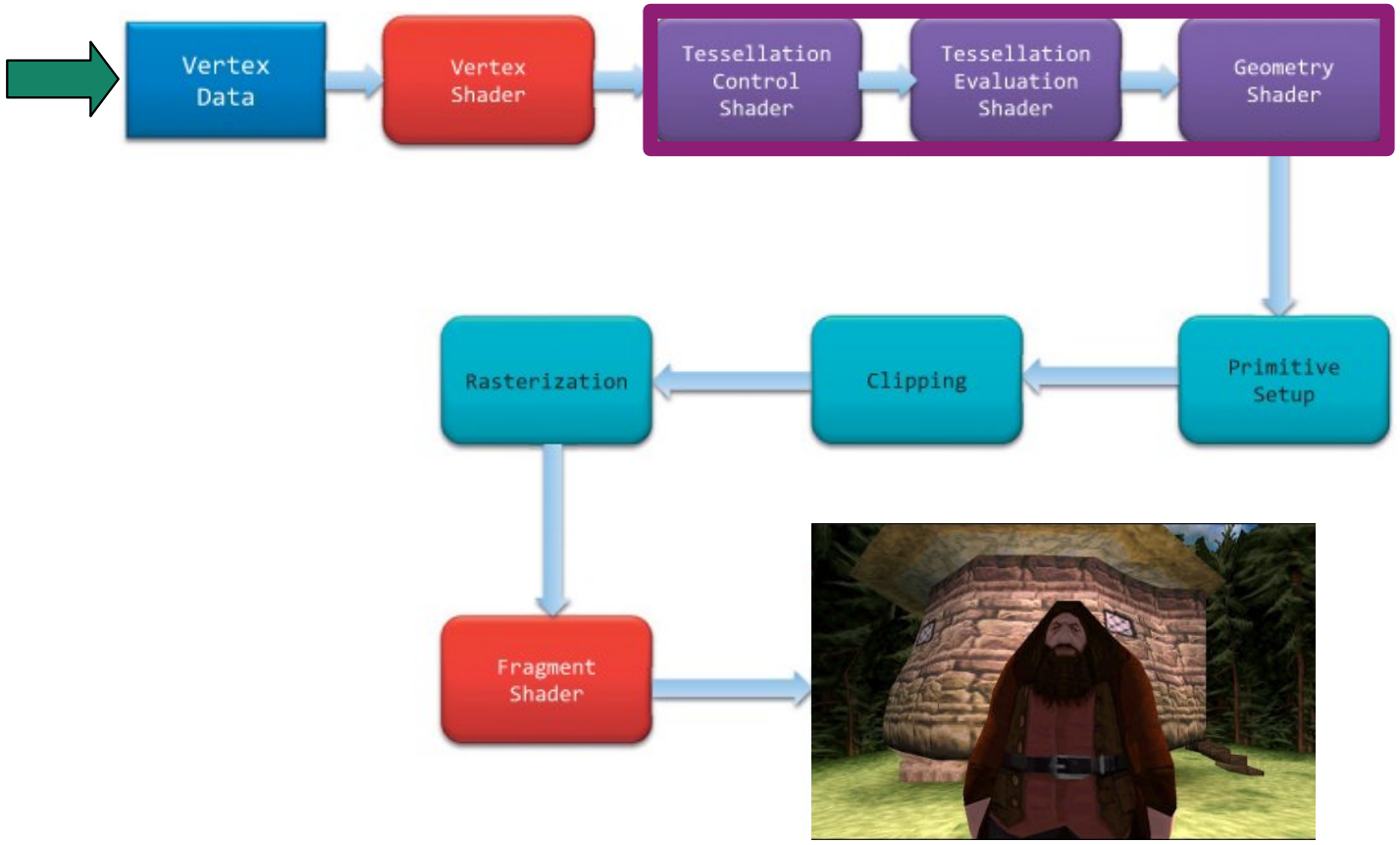
Vertex Shader

Ejemplo de vertex shader en glsl

```
#version 330

in vec2 position;
in vec3 color;
in float intensity;

out vec3 fragColor;
out float fragIntensity;

void main()
{
    fragColor = color;
    fragIntensity = intensity;
    gl_Position = vec4(position, 0.0f, 1.0f);
}
```

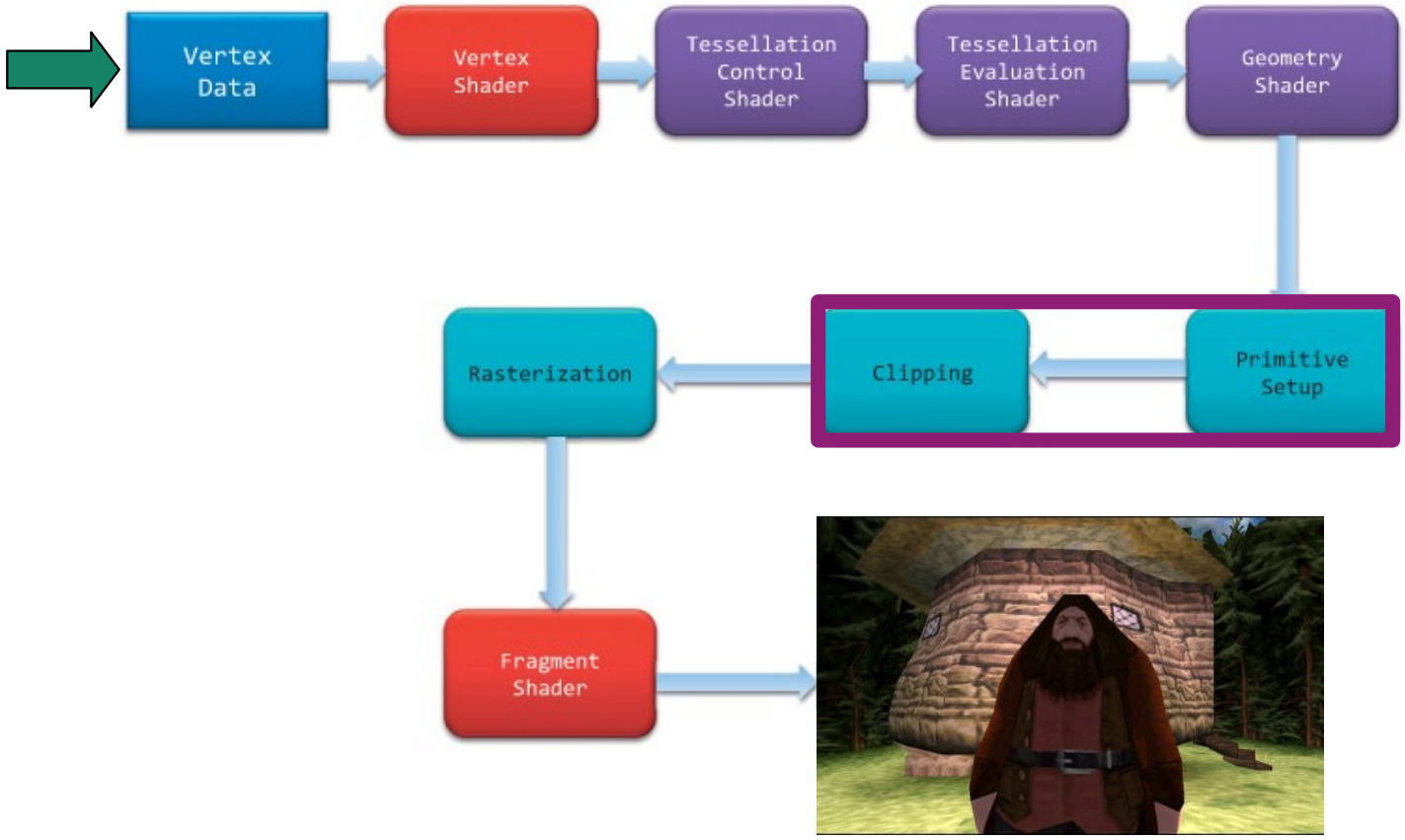
Shaders Configurables

```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```



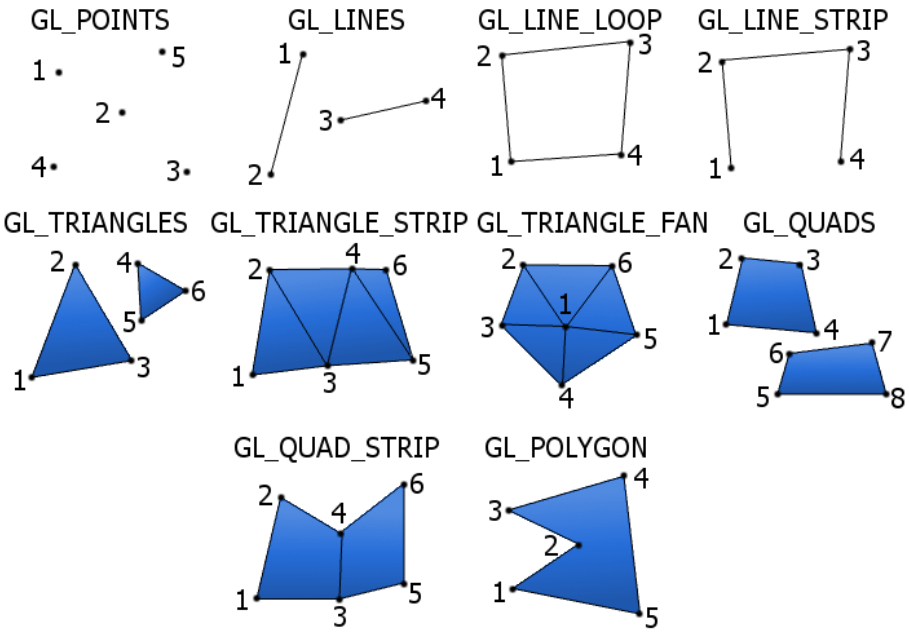
Primitive Setup y Clipping

```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```

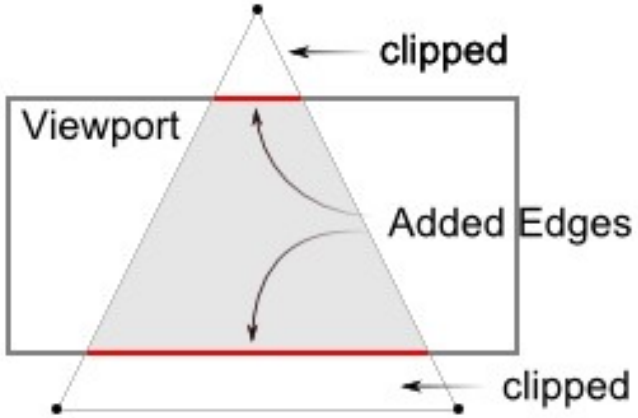


Primitive Setup

Dependiendo del modo de renderizar las primitivas, los vértices forman figuras más complejas



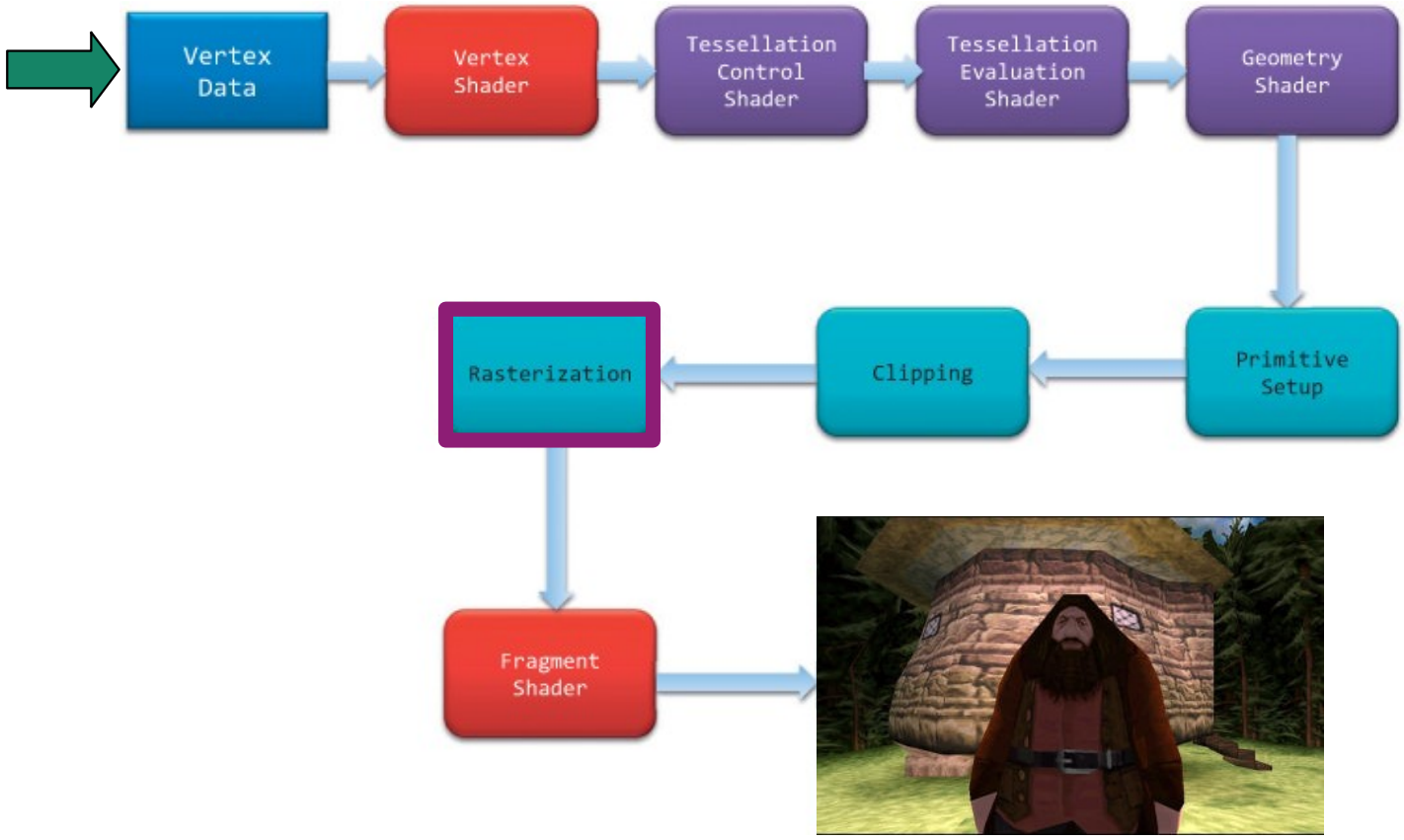
Clipping



Para que en la rasterización no se usen primitivas fueras de vista, éstas se omiten
Y se reajusta la primitiva al viewport

Rasterización

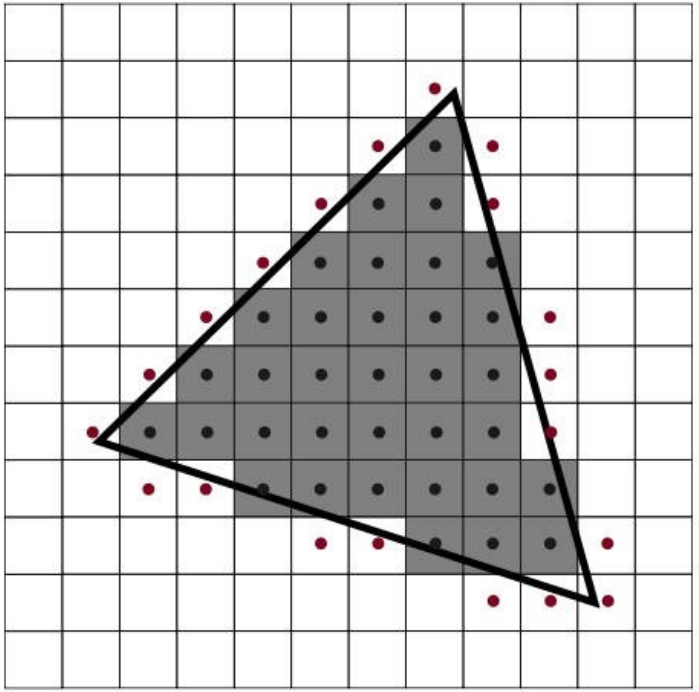
```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```



Rasterización

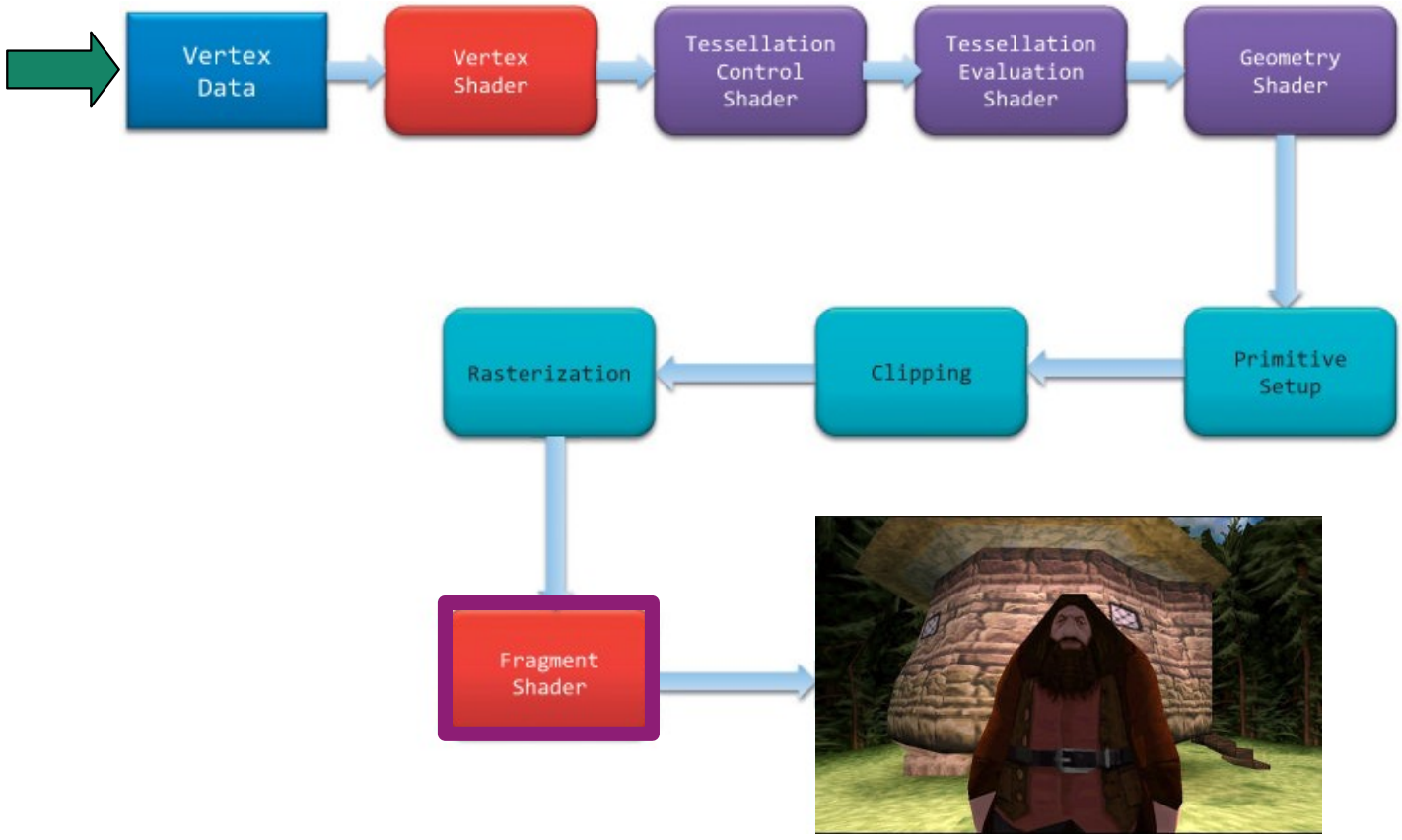
- Asigna un fragmento de la primitiva a un pixel

- Aquí se realiza la interpolación



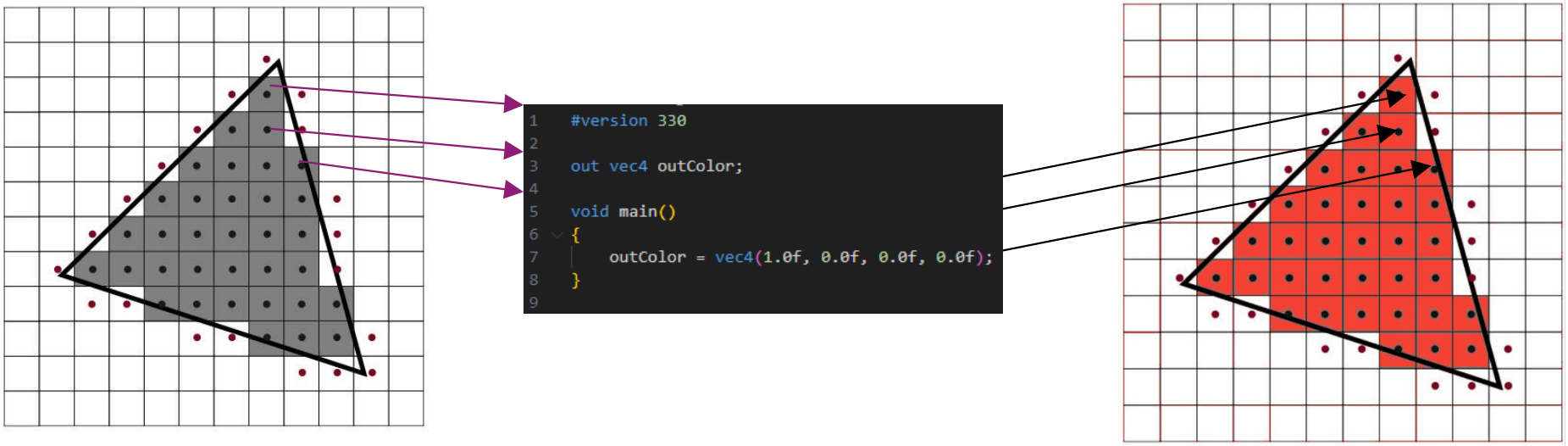
Fragment Shader

```
v       -5.000000   5.000000    0.000000
v       -5.000000   -5.000000   0.000000
v       5.000000    -5.000000   0.000000
v       5.000000    5.000000    0.000000
vt      -5.000000   5.000000    0.000000
vt      -5.000000   -5.000000   0.000000
vt      5.000000    -5.000000   0.000000
vt      5.000000    5.000000    0.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vn      0.000000    0.000000    1.000000
vp      0.210000    3.590000
vp      0.000000    0.000000
vp      1.000000    0.000000
vp      0.500000    0.500000
```



Fragment Shader

Fragment Shader Se ejecuta por cada *fragmento* (*pixel*)



```
#version 330

out vec4 outColor;

void main()
{
    outColor = vec4(1.0f, 0.0f, 0.0f, 0.0f);
}
```

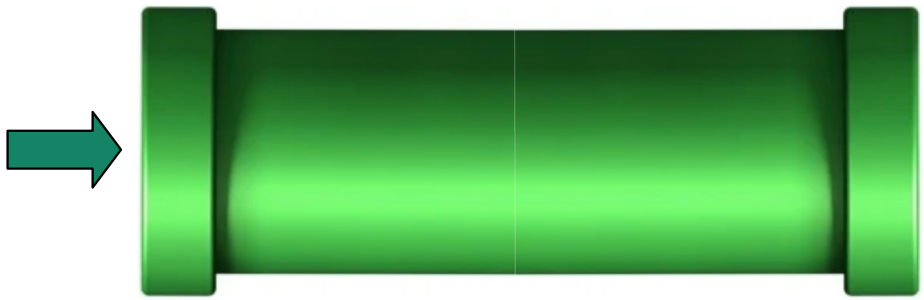
Fragment Shader

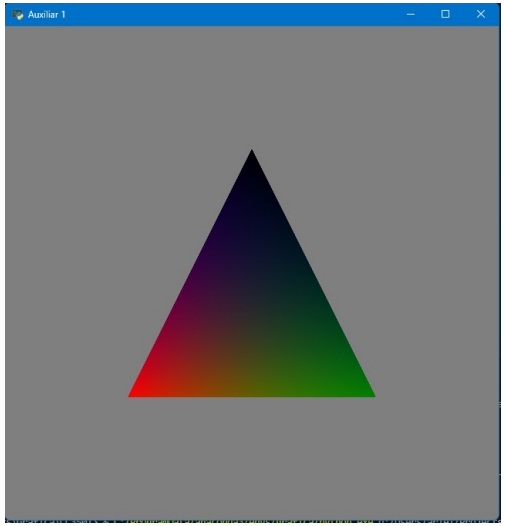
Ejemplo de fragment shader en glsl

```
1   #version 330
2
3   in vec3 fragColor;
4   in float fragIntensity;
5   out vec4 outColor;
6
7   void main()
8   {
9       outColor = fragIntensity * vec4(fragColor, 1.0f);
10  }
```

Ejemplo

```
v       -5.000000    5.000000     0.000000
v       -5.000000    -5.000000    0.000000
v       5.000000     -5.000000    0.000000
```





Auxiliar 1

Rendering Pipeline

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