

Grado de Ingeniería Multimedia  
– Mención en videojuegos



TRABAJO DE FINAL DE GRADO

DESARROLLO DE UN  
MOTOR DE JUEGOS  
*FROM SCRATCH*

ALUMNO

Joel López Romero

PROFESOR PONENTE

Alun Thomas Evans

# ÍNDICE

## 1. Introducción

Estado actual / Problema / Solución

## 2. Proyecto

Objetivos / Resultados (funcionalidades) / Planificación

## 3. Bloque 1: Raycaster



¿Qué es? / Funcionalidades y justificación / Desarrollo del motor

## 4. Bloque 2: 3ngine

## 5. Conclusiones

Resultados / Coste en horas / Líneas de continuación / Experiencia

# 1. INTRODUCCIÓN: SITUACIÓN ACTUAL

## *Desarrollo de videojuegos*

- ✓ Herramientas de creación de assets
- ✓ Motores
  - (de) Físicas
  - Gráficos
- ✓ Editores de videojuegos

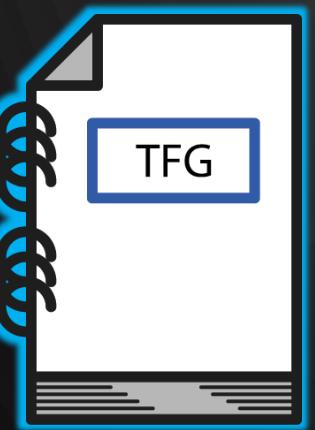


MOTOR DE JUEGOS FROM SCRATCH

# 1. INTRODUCCIÓN: EL PROBLEMA



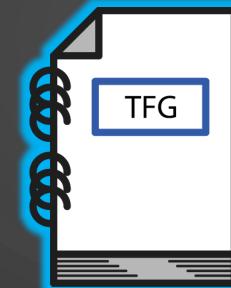
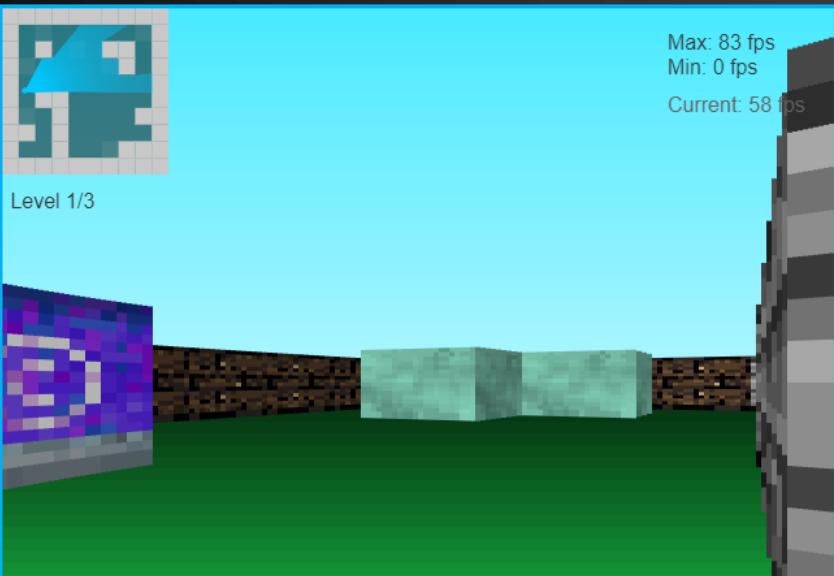
# 1. INTRODUCCIÓN: SOLUCIÓN



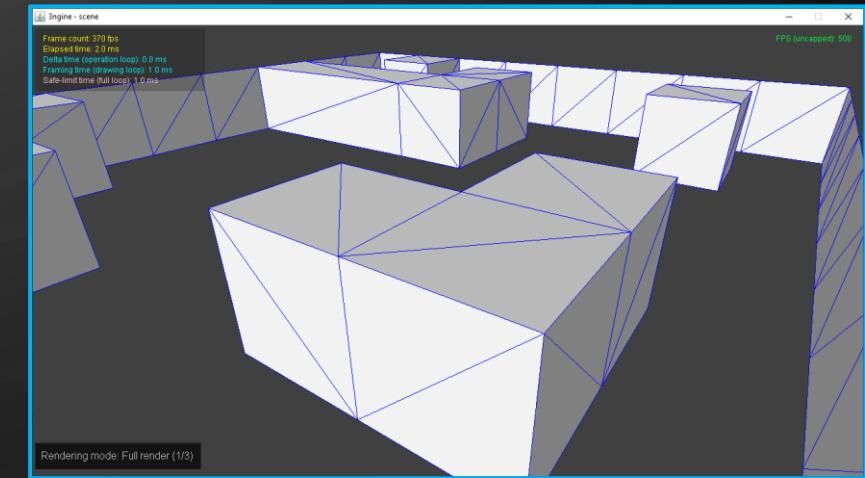
- Conceptos (**mates**)
- Aprender atajos
- Historia
- Salir de la caja
- Desde cero (proyecto inicial vacío)

## 2. PROYECTO

*Raycaster (2D)*

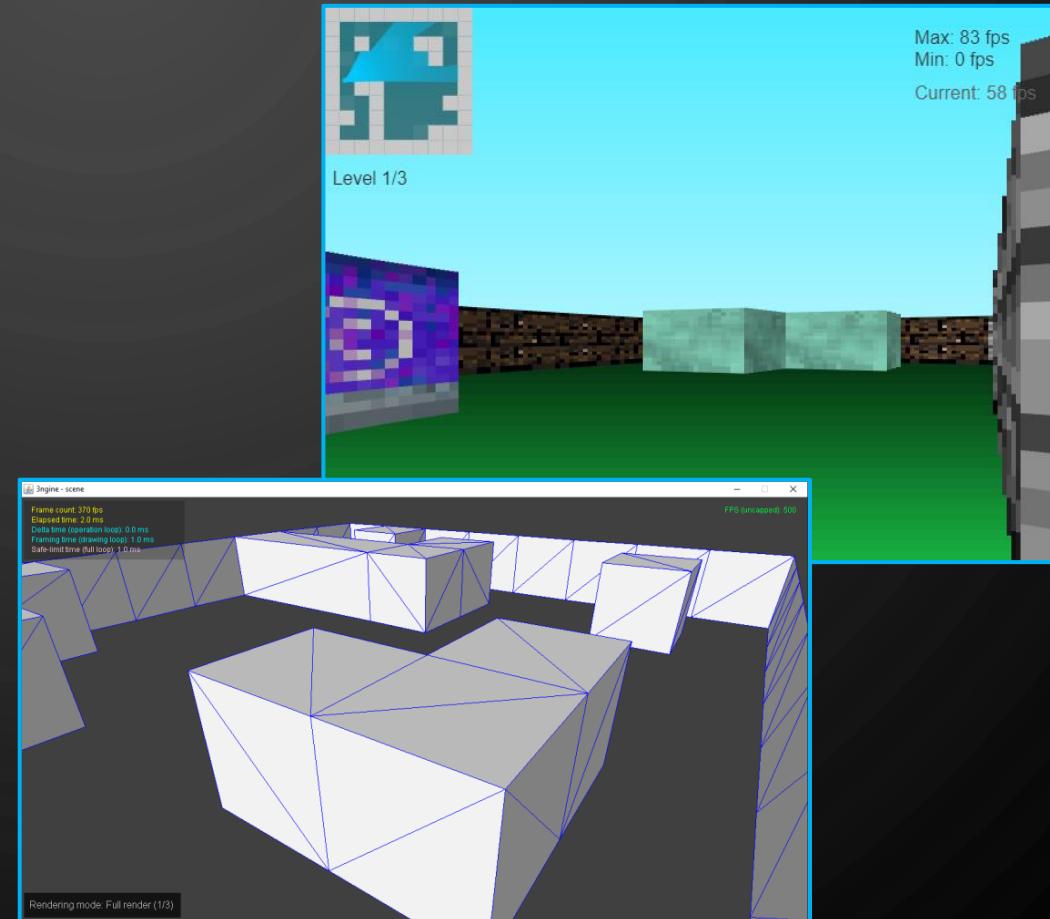


*3ngine (3D)*



## 2. PROYECTO: OBJETIVOS

- ✓ Gestión y lectura de recursos
- ✓ Carga de escena (+ nivel)
- ✓ Cámara (y procesos adicionales)
- ✓ Transformación de entidades
- ✓ Texturizado
- ✓ Colisiones
- ✓ *User input*
- ✓ *Debugging (tiempos y framing)*



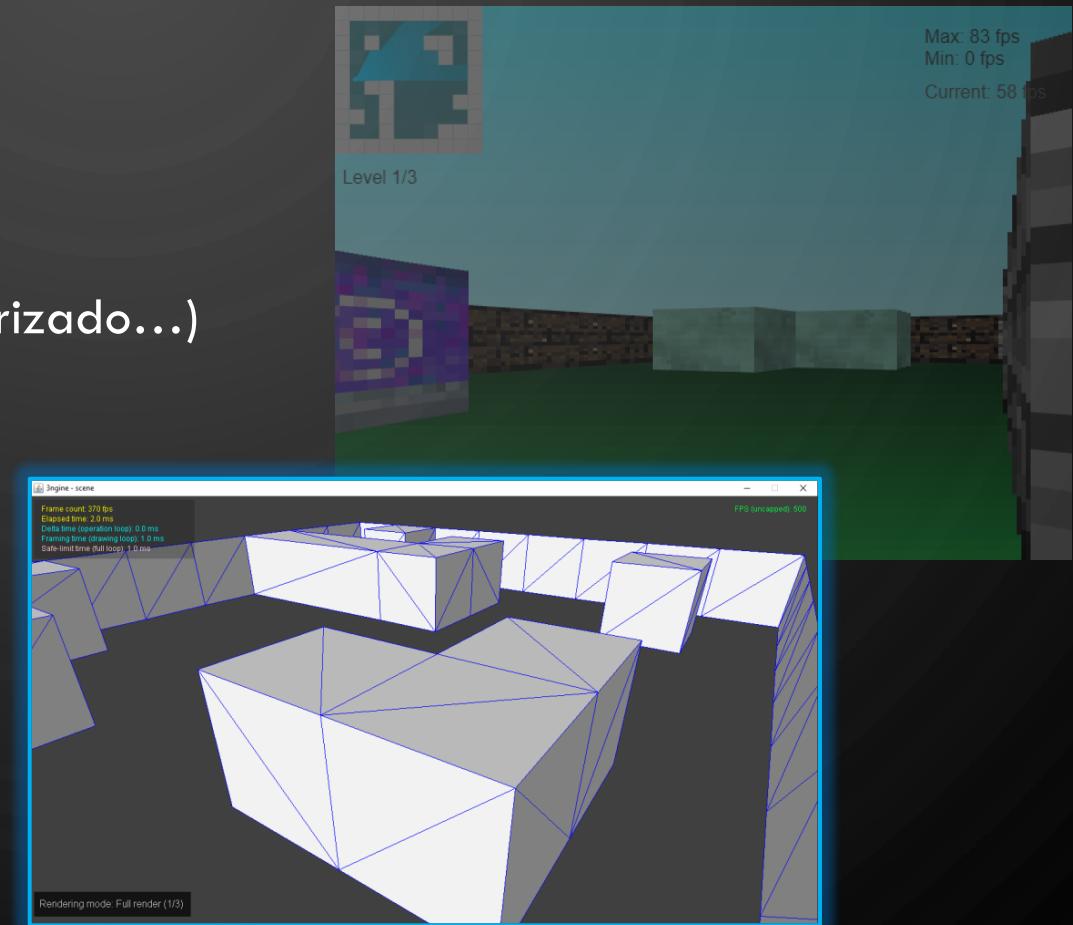
## 2. PROYECTO: OBJETIVOS (RAYCASTER)

- ✓ Gestión y lectura de recursos
- ✓ Carga de escena (+ nivel)
- ✓ Cámara (2D, parámetros variables)
- ✓ Transformación de entidades
- ✓ Texturizado
- ✓ Colisiones
- ✓ *User input*
- ✓ *Debugging (tiempos y framing)*

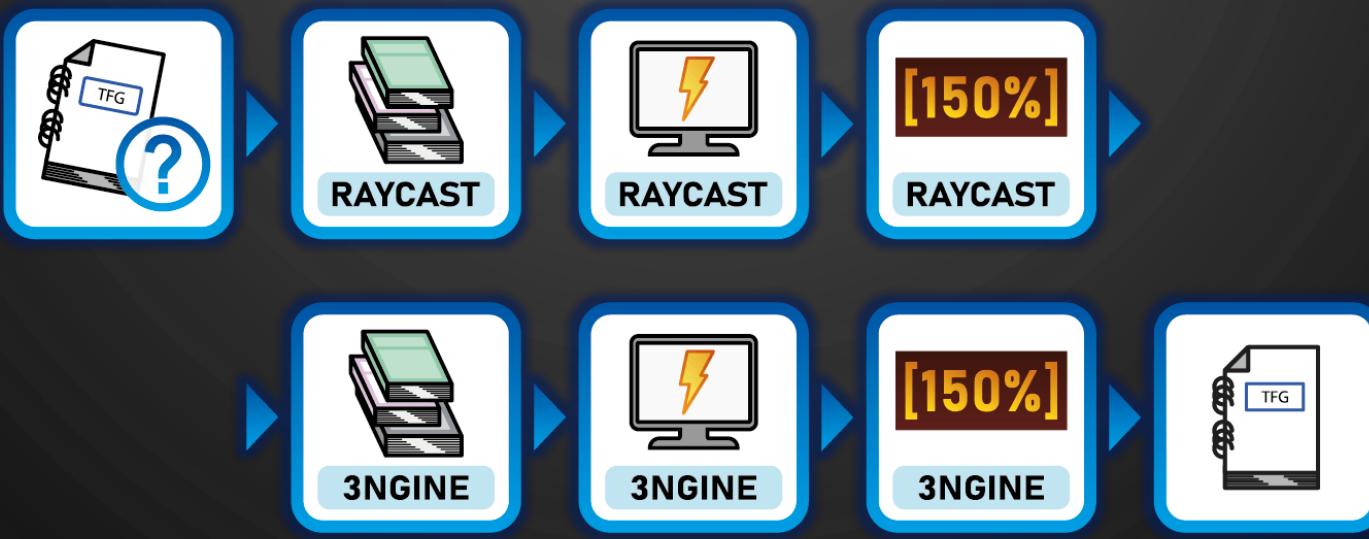


## 2. PROYECTO: OBJETIVOS (3NGINE)

- ✓ Gestión y lectura de recursos
- ✓ Carga de escena (+ nivel)
- ✓ Cámara (ordenación, *clipping*, renderizado...)
- ✓ Transformación de entidades
- ✓ Texturizado
- ✓ Colisiones
- ✓ *User input*
- ✓ *Debugging* (tiempos y *framing*)



## 2. PROYECTO: PLANIFICACIÓN

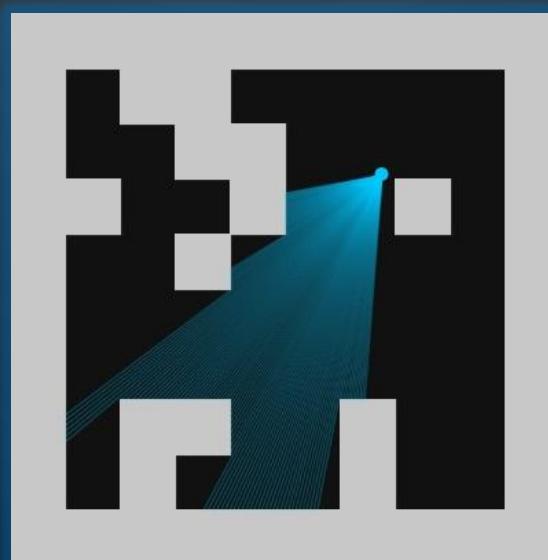


MOTOR DE JUEGOS FROM SCRATCH

10

### 3. BLOQUE 1, RAYCASTER: ¿QUÉ ES?

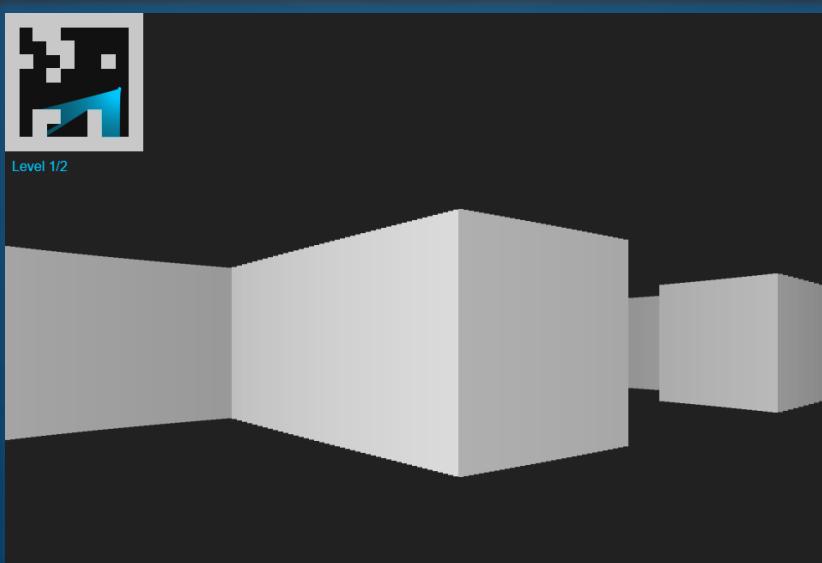
“Método de **renderizado** basado en una **falsa simulación** virtual **3D** mediante información **2D**”



MOTOR DE JUEGOS FROM SCRATCH

### 3. BLOQUE 1, RAYCASTER: ¿QUÉ ES?

“Método de **renderizado** basado en una **falsa simulación** virtual **3D** mediante información **2D**”



MOTOR DE JUEGOS FROM SCRATCH

12

### 3. BLOQUE 1, RAYCASTER: ¿QUÉ ES?

“Método de **renderizado** basado en una **falsa simulación** virtual **3D** mediante información **2D**”



Wolfenstein 3D (1992),  
ID Software

### 3. BLOQUE 1, RAYCASTER: A OBTENER

JavaScript ([entorno web](#))

[Multiplataforma](#)

[Simplicidad](#) (lógica y cálculo)

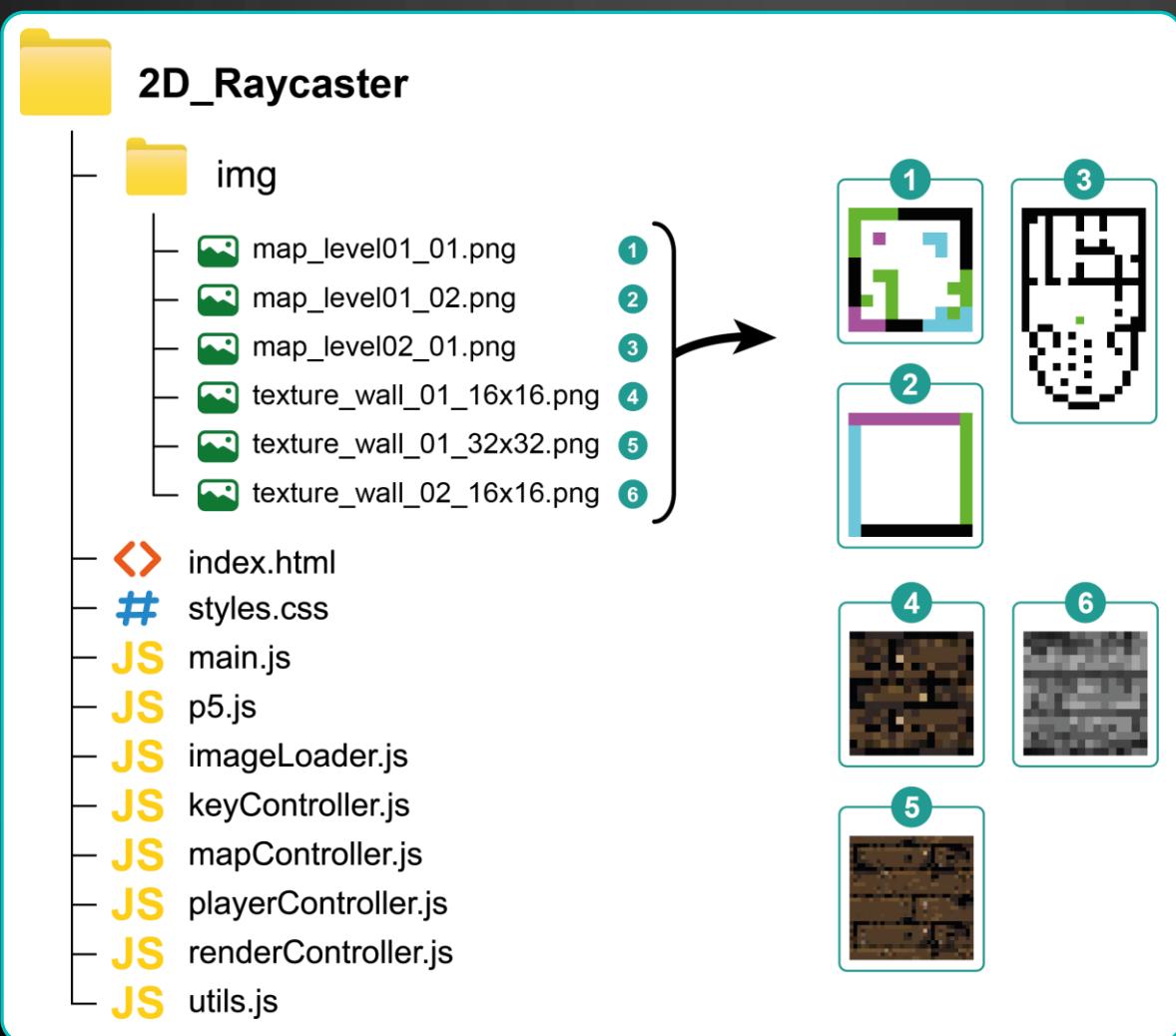
[Resultados rápidos](#)

Renderizado [extremadamente óptimo](#)  
(y completa [adaptabilidad](#))

Aplicación de [trigonometría](#) en el  
cálculo de [colisiones](#) y gráficos  
([preparación para el 3D](#))



# 3. BLOQUE 1, RAYCASTER: DESARROLLO



Proyecto + Mapa

Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

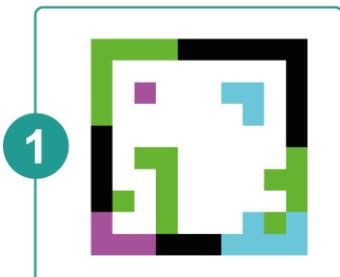
Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO

#### GENERANDO LA ESCENA 01 POR IMÁGENES:

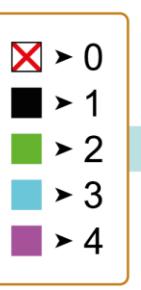
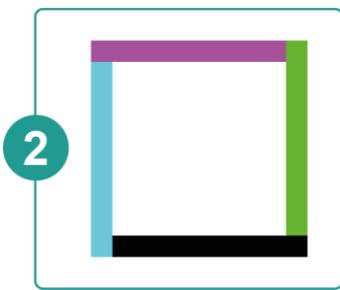
La escena 01 la forman dos niveles:

"map\_level01\_01.png", → 1  
"map\_level01\_02.png", → 2



[2, 2, 2, 2, 1, 1, 1, 1, 1, 1]  
[2, 0, 0, 0, 0, 0, 0, 0, 1]  
[2, 0, 4, 0, 0, 0, 3, 0, 0]  
[2, 0, 0, 0, 0, 0, 3, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 2, 2, 0, 0, 0, 0, 2]  
[1, 0, 0, 2, 0, 0, 0, 0, 2, 2]  
[1, 2, 0, 2, 0, 0, 0, 0, 0, 2]  
[4, 0, 0, 2, 0, 0, 0, 3, 2, 3]  
[4, 4, 4, 1, 1, 1, 3, 3, 3, 3]

Escena 01,  
nivel 01



[4, 4, 4, 4, 4, 4, 4, 4, 4, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 0, 0, 0, 0, 0, 0, 0, 0, 2]  
[3, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Escena 01,  
nivel 02

Proyecto + Mapa

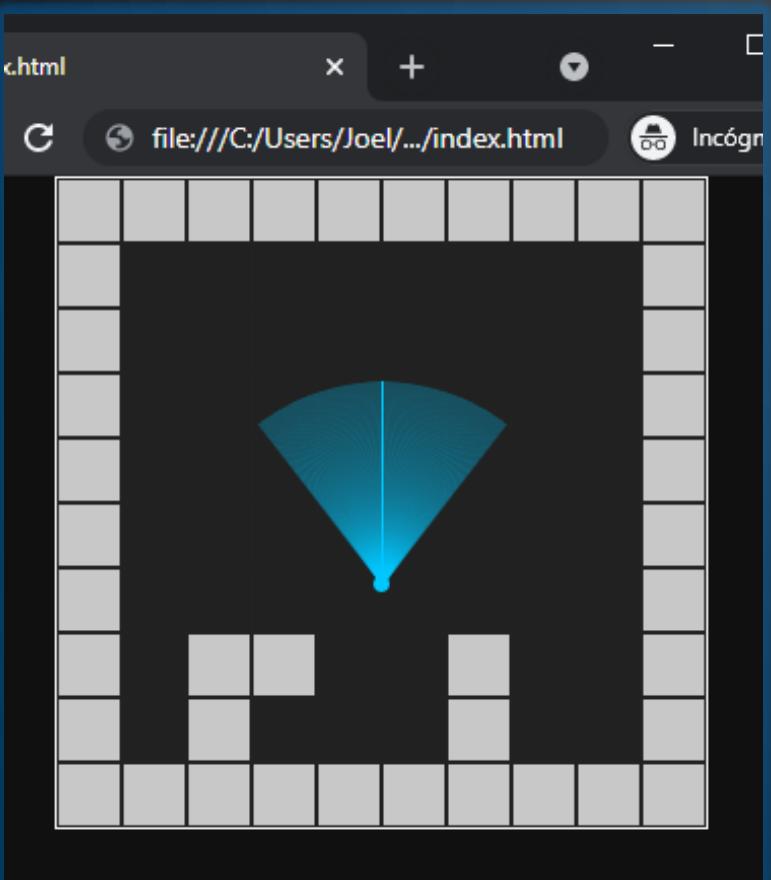
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



PlayerController.js

Player

FOV

xN

Ray

Proyecto + Mapa

Raycasting (Player)

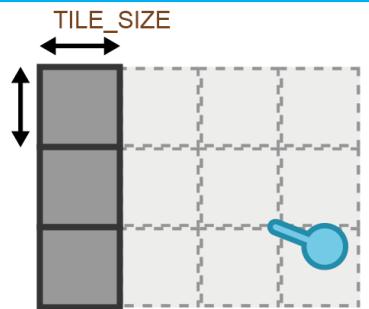
Raycasting (colisiones)

Proyección (vLines)

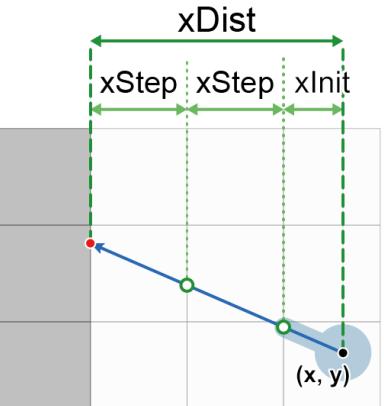
Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO

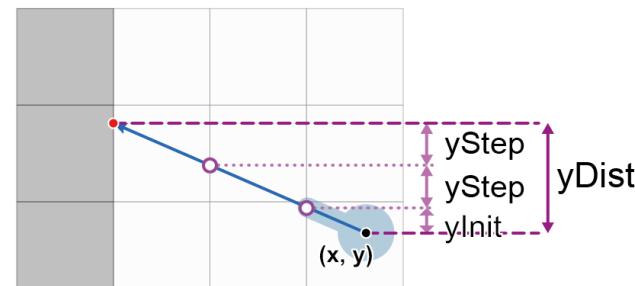
- Colisiones: por separado (**prioridad en X**)
- Distancia del rayo:
  - $xDist = xStep + xInit$
  - $yDist = yStep + yInit$
- **xStep = TILE\_SIZE** (32px)



Render



Distancia en X



Distancia en Y

Proyecto + Mapa

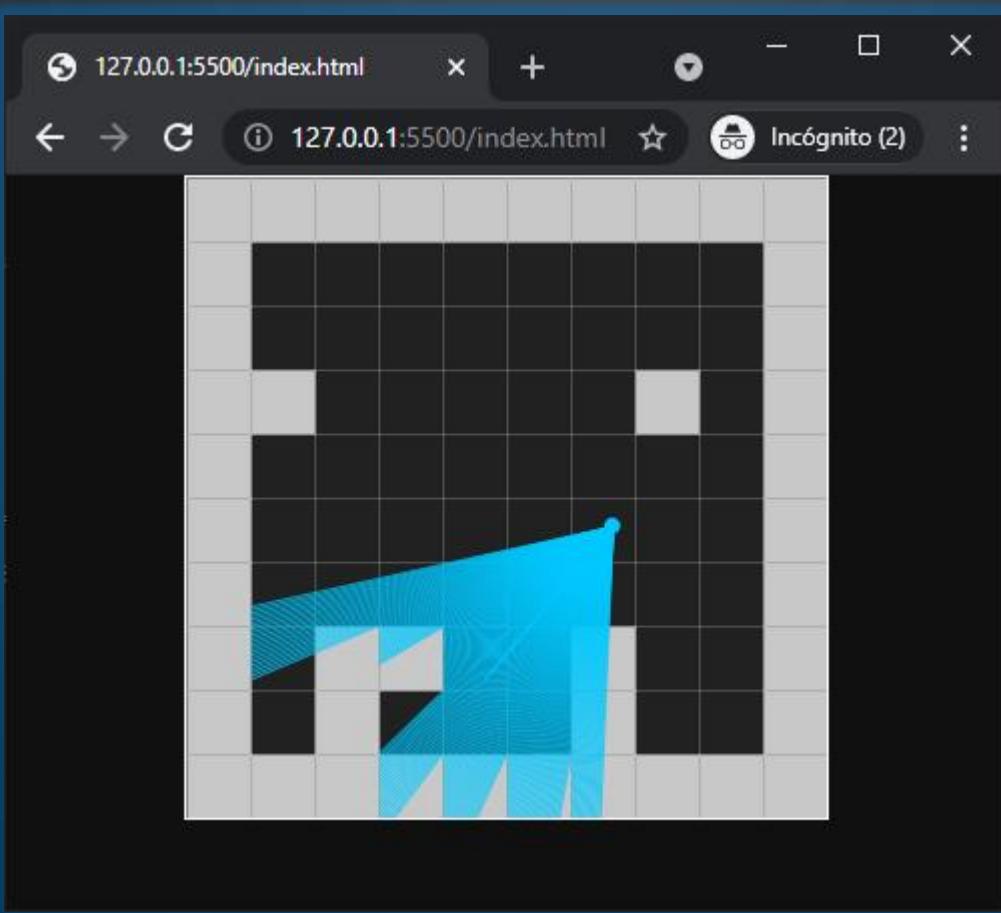
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



Proyecto + Mapa

Raycasting (Player)

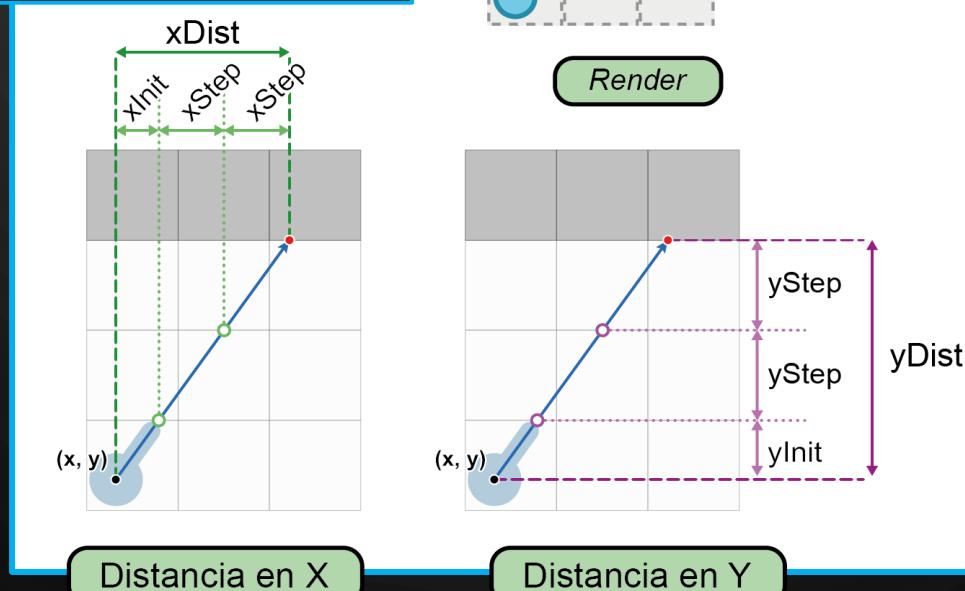
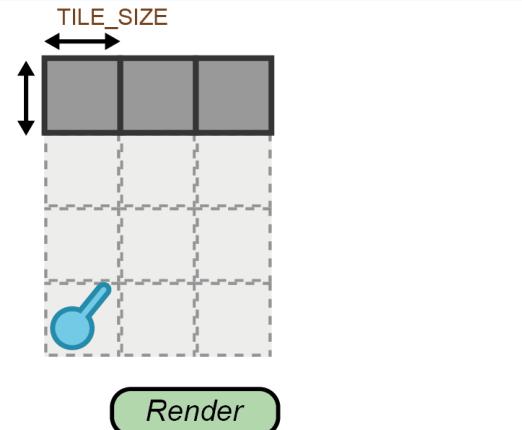
Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO

- Colisiones: por separado (**prioridad en Y**)
- Distancia del rayo:
  - $xDist = xStep + xInit$
  - $yDist = yStep + yInit$
- **yStep = TILE\_SIZE (32px)**



Proyecto + Mapa

Raycasting (Player)

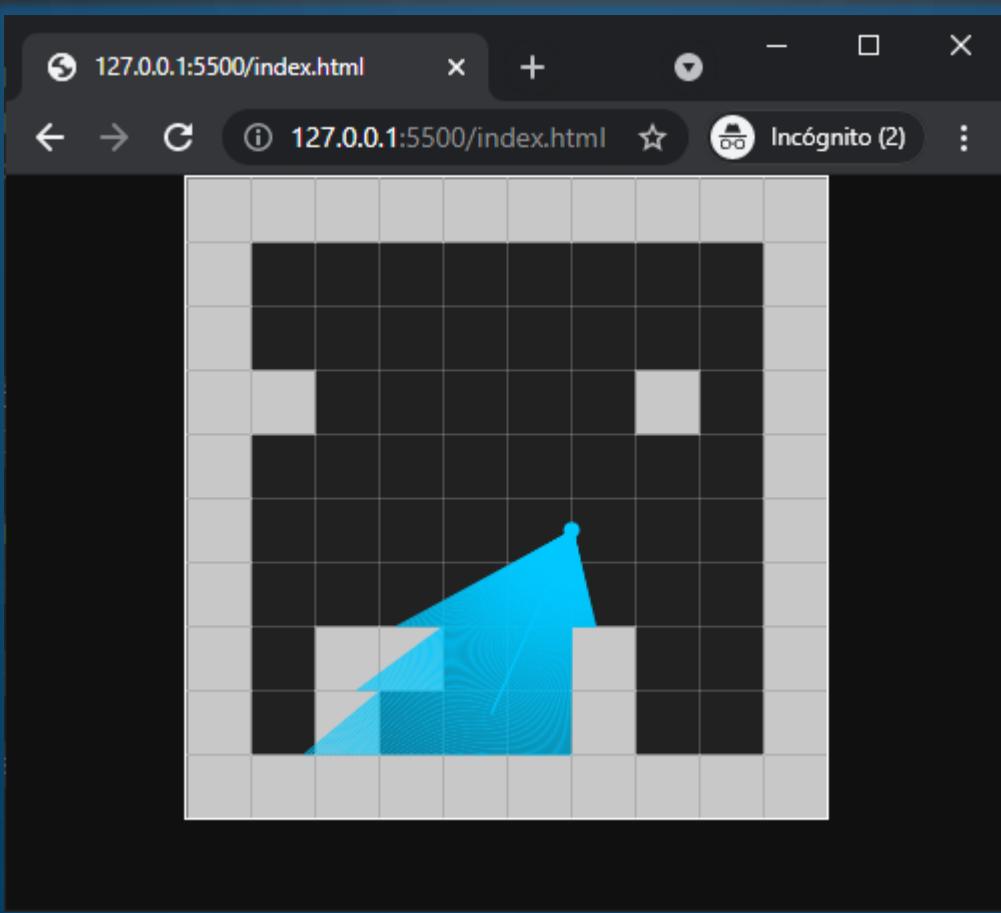
Raycasting (colisiones)

Proyección (vLines)

Texturizado

20

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



Proyecto + Mapa

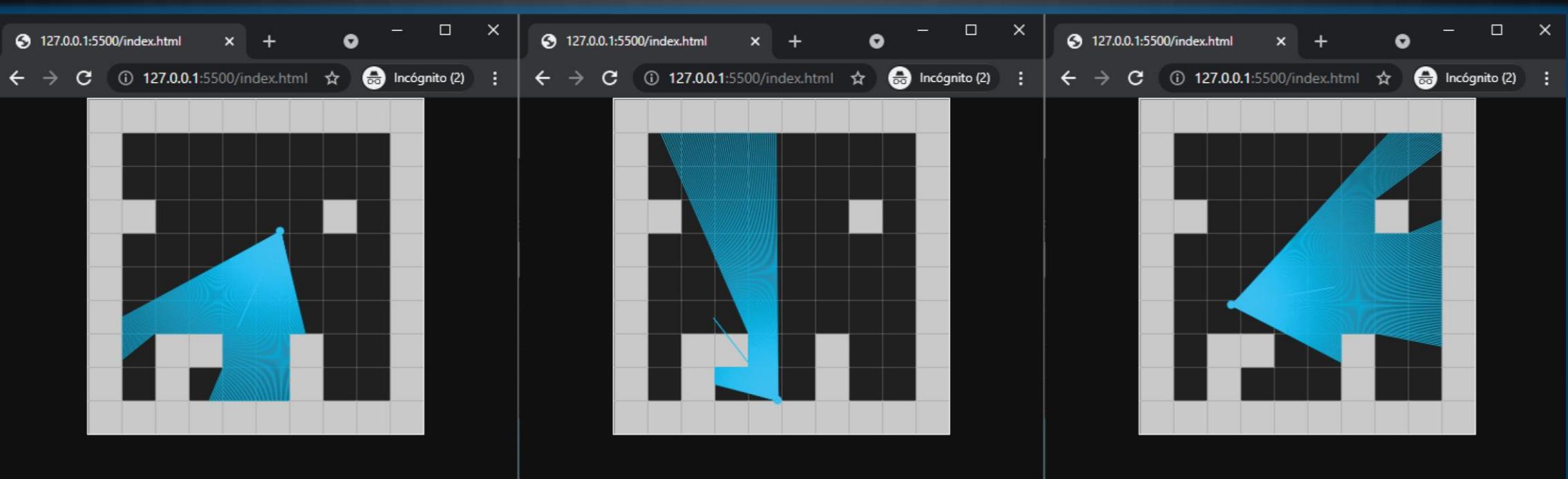
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

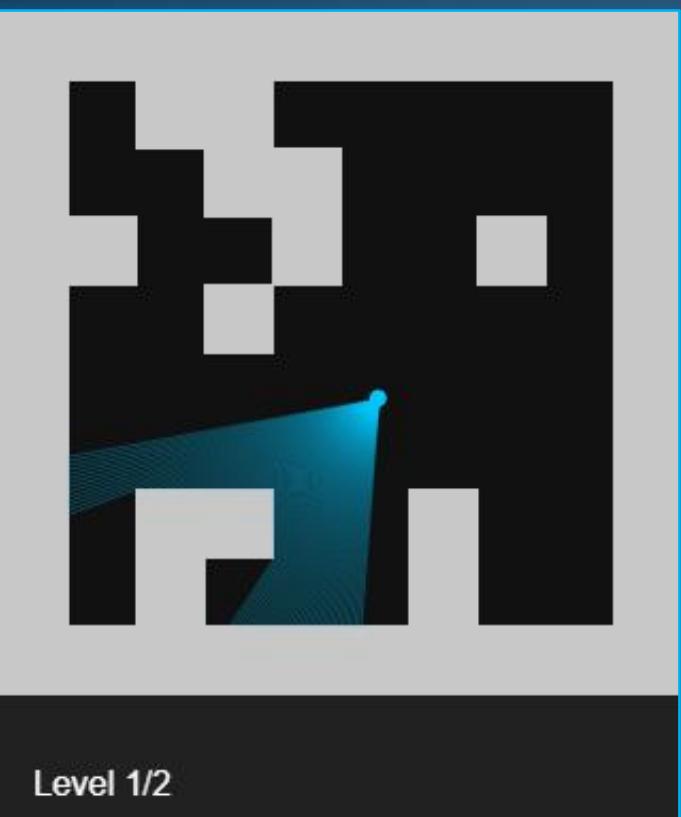
### 3. BLOQUE 1, RAYCASTER: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

22

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



Proyecto + Mapa

Raycasting (*Player*)

Raycasting (colisiones)

Proyección (*vLines*)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



```
calculateVLineHeight(i) {  
    let distToProjectionPlane = VIEWPORT_WIDTH/2 / Math.tan(FOV/2);  
    let height = (TILE_SIZE / objPlayer.fov.rays[i].distance) *  
distToProjectionPlane;  
    return height;  
}
```

Proyecto + Mapa

Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



```
calculateVLineHeight(i) {  
    let distToProjectionPlane = Math.tan(FOV/2);  
    let height = (TI * distToProjectionPlane) / (fov.rays[i].distance) *  
    return height;  
}
```

$$h = \frac{F}{dist}$$

Proyecto + Mapa

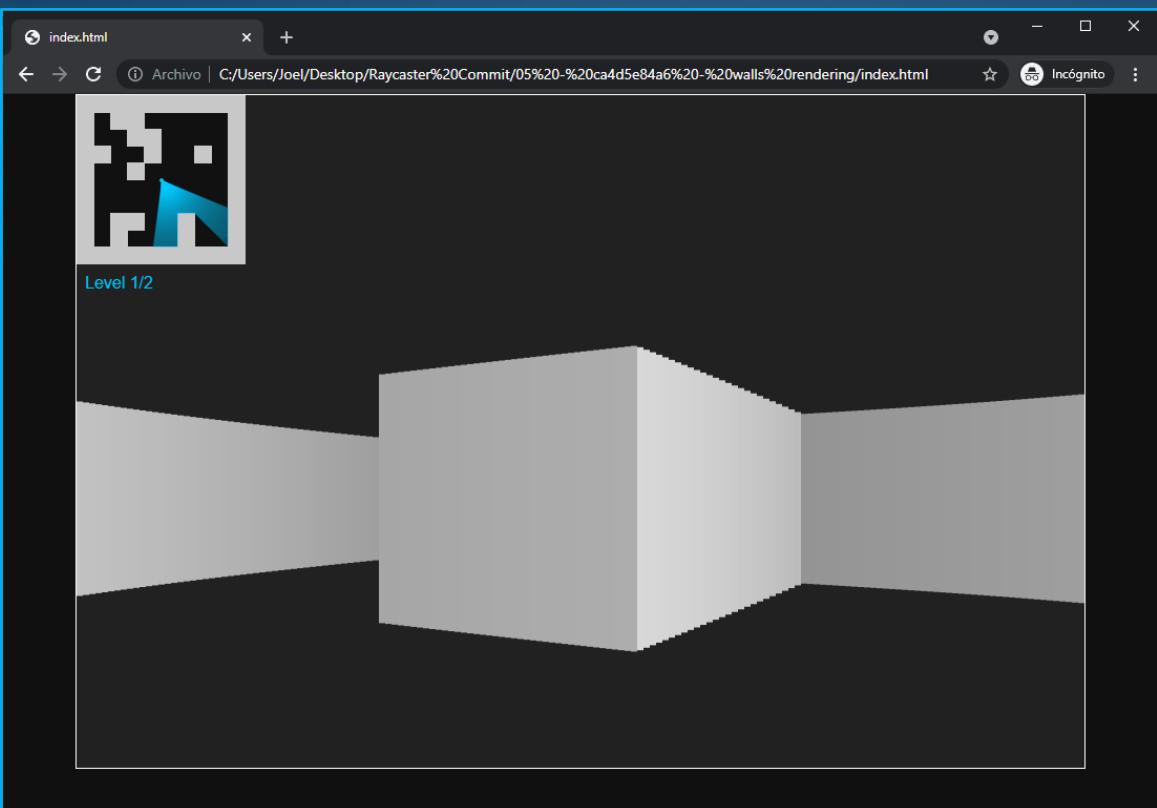
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Mapa

Raycasting (Player)

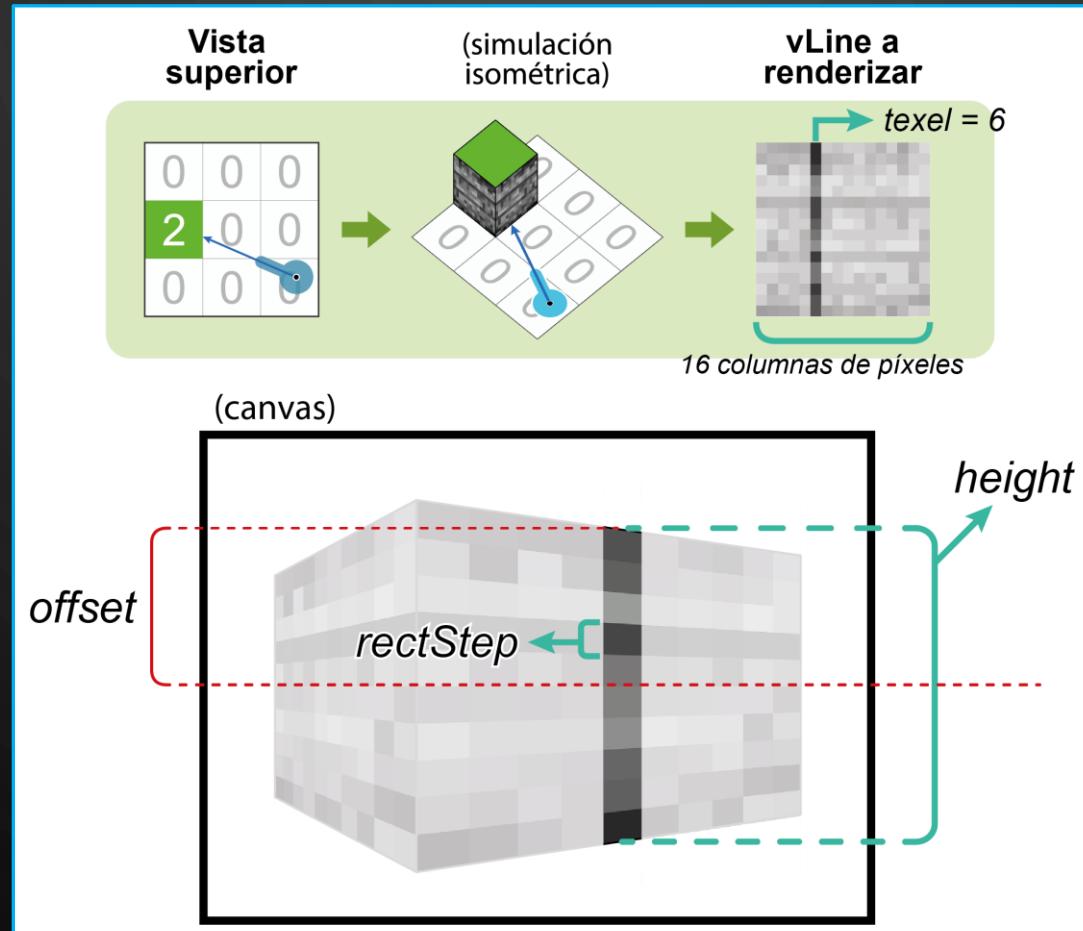
Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO

Calcular la columna de píxeles a aplicar (**Texel**), según la coordenada de colisión del rayo.



Proyecto + Mapa

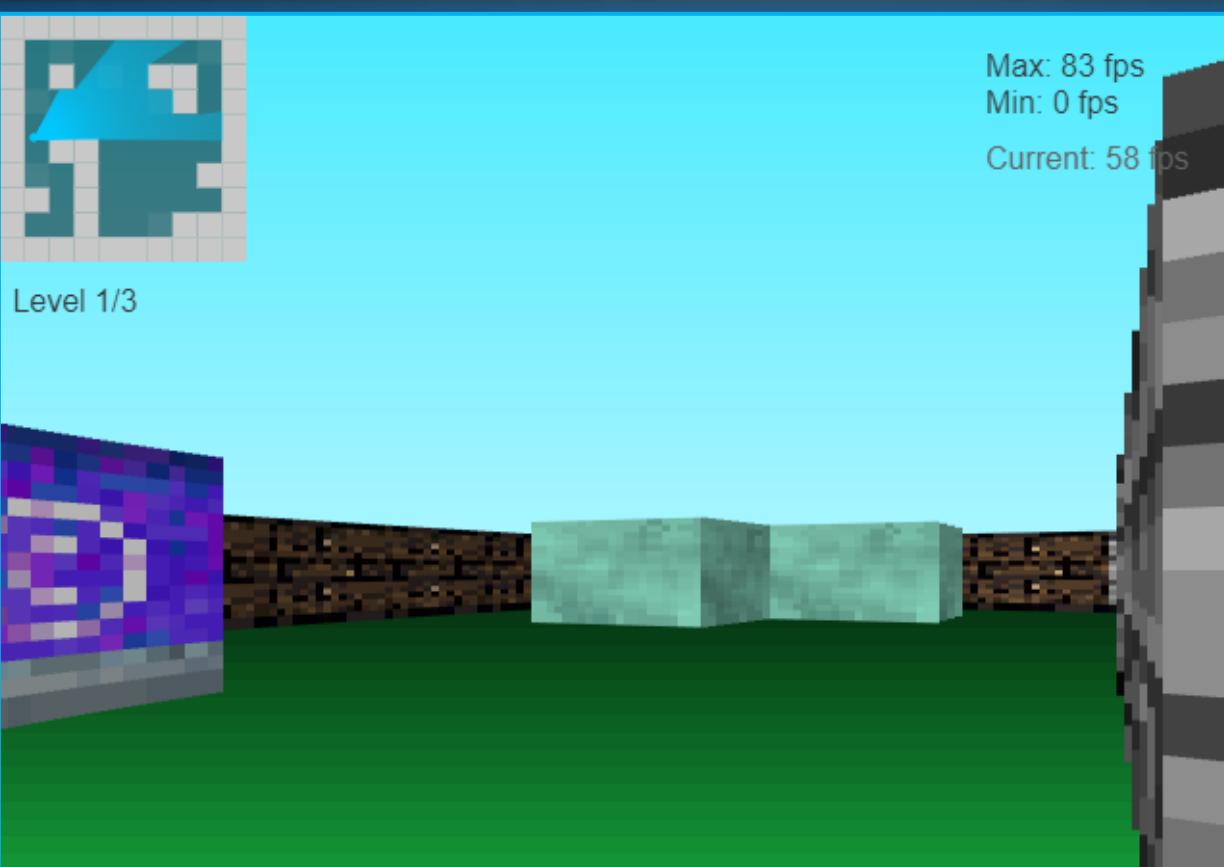
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO



Proyecto + Mapa

Raycasting (Player)

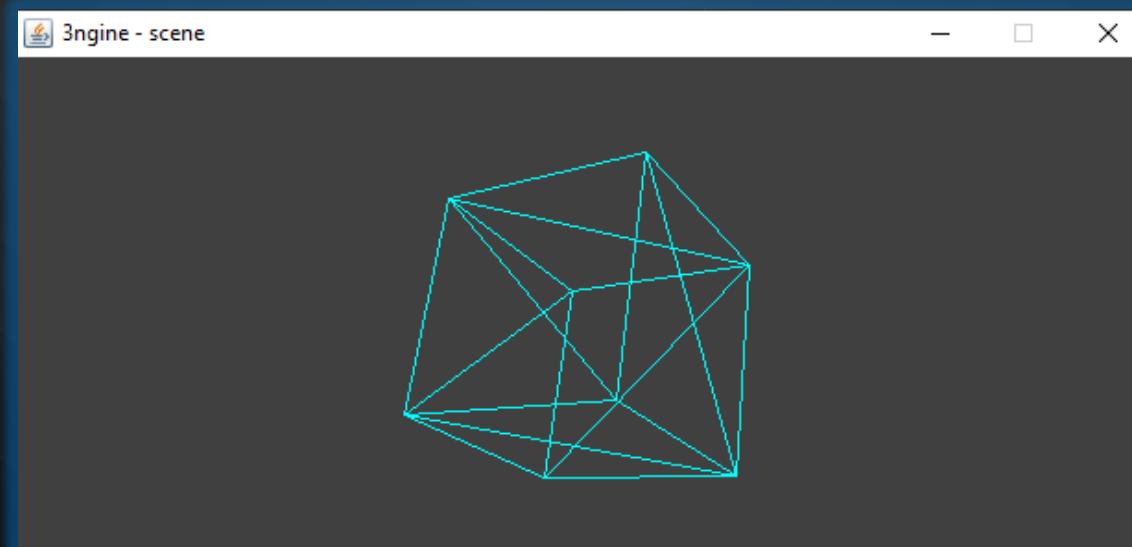
Raycasting (colisiones)

Proyección (vLines)

Texturizado

## 4. BLOQUE 2, 3NGINE: ¿QUÉ ES?

“Motor **transparente**, para la aplicación de **cálculo matemático**,  
el proceso de **renderizado** y la **gestión de entidades**”



MOTOR DE JUEGOS FROM SCRATCH

## 4. BLOQUE 2, 3NGINE: A OBTENER

Java (*desktop*)

Multiplataforma

Estilo de clases: **MVC**

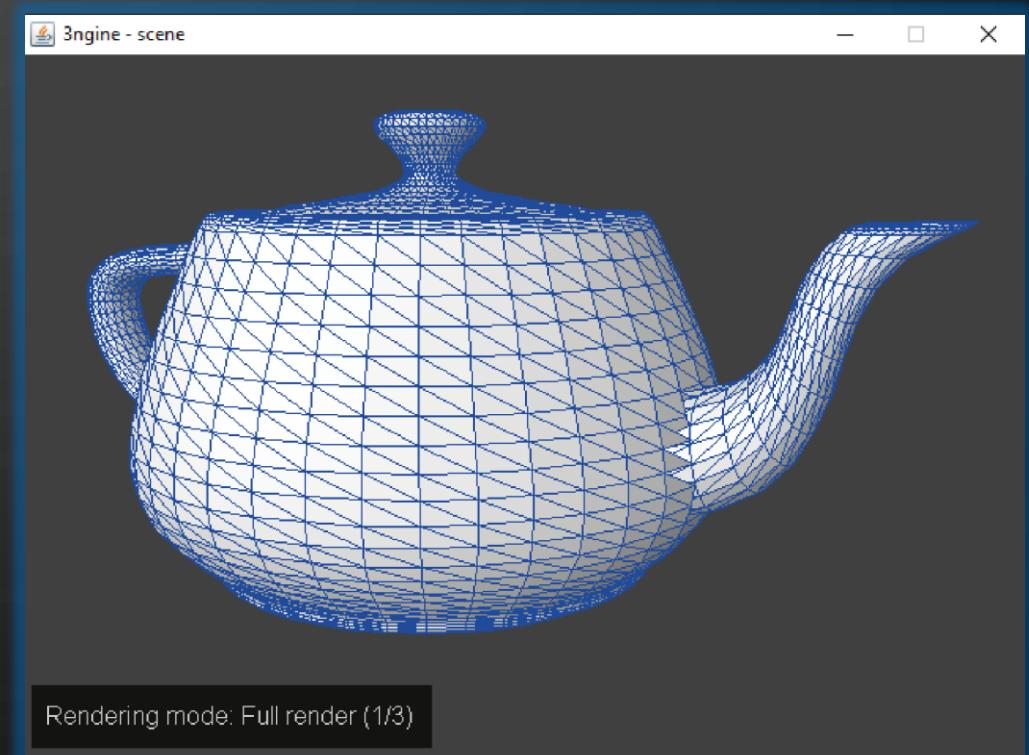
Optimización (matrices)

Lectura de **.OBJ**

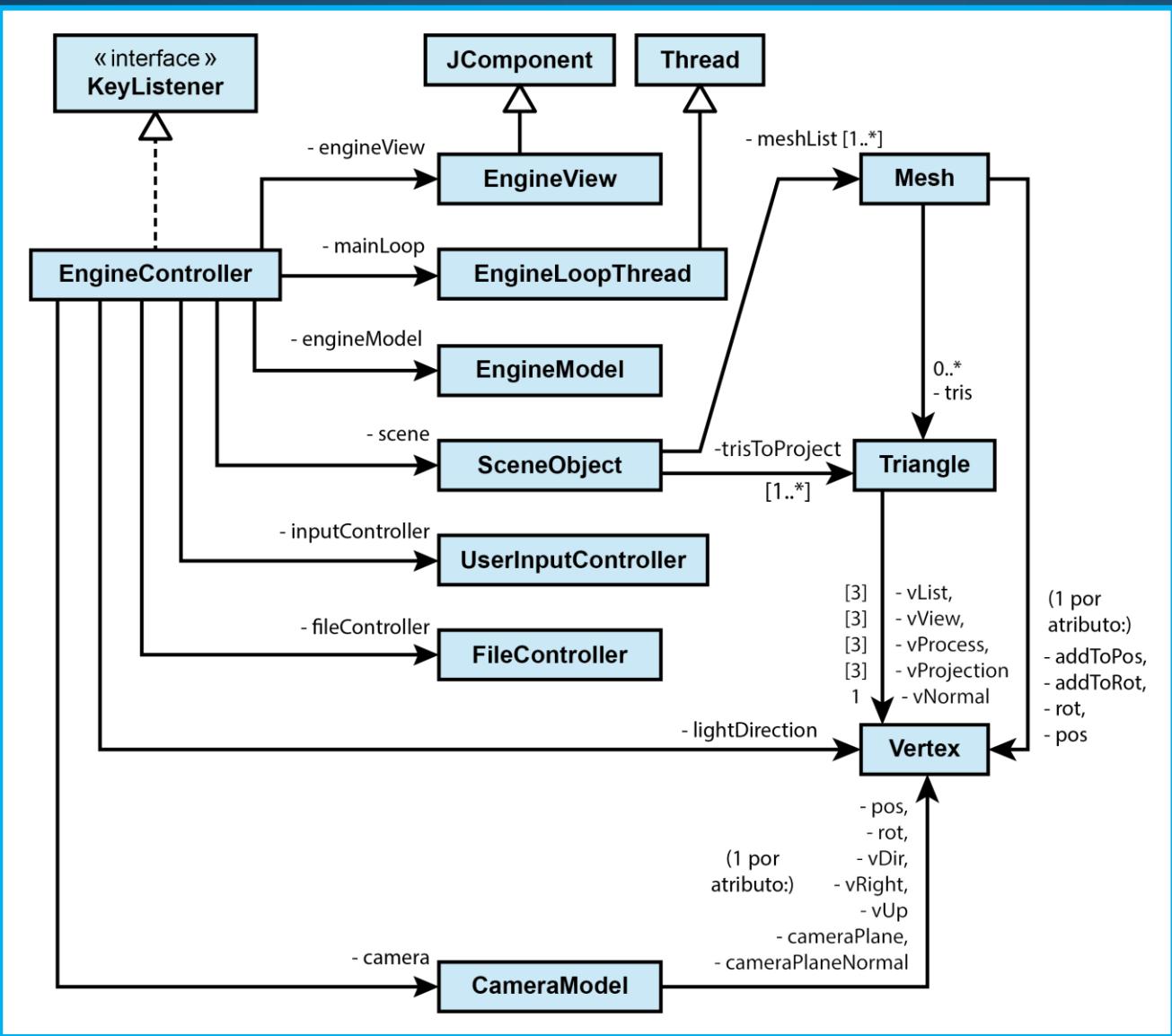
Hilo de ejecución propio

Varios tipos de **renderizado**

Acceso total a **vértices** y **operaciones**  
(sin cajas negras)



## 4. BLOQUE 2, 3NGINE: DESARROLLO



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

- Usuario pulsa “O” para abrir un .OBJ

**FileController:**

```
new JFileChooser();
```

Abre un .OBJ

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

**EngineController:**

```
new SceneObject();
```

Crea una escena

**EngineController:**

```
new EngineLoopThread();
```

Crea el hilo de ejecución

**FPS**  
(desarrollador)

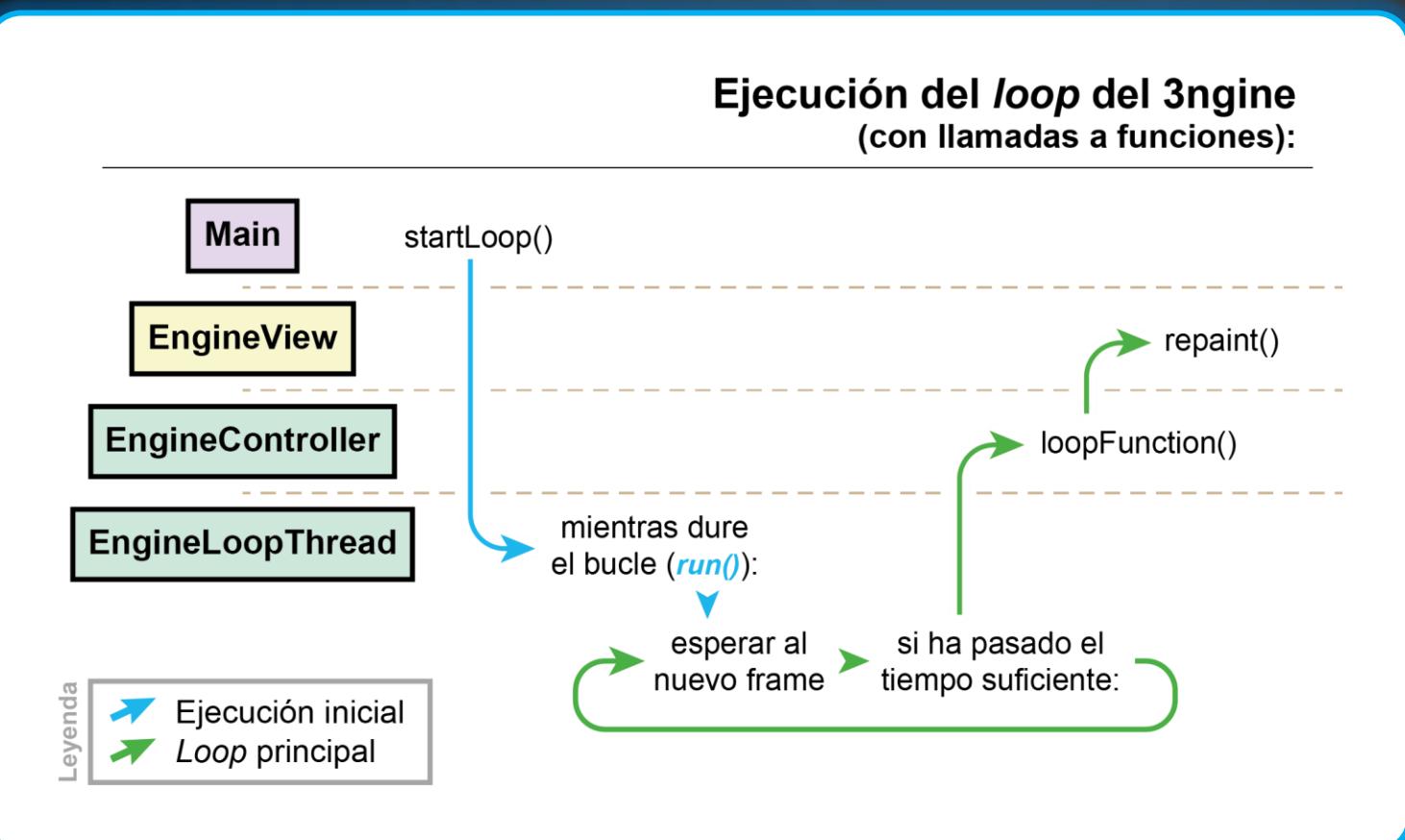
**EngineController:**

```
loopFunction();
```

[Gráficos de la escena]

elapsedTime

## 4. BLOQUE 2, 3NGINE: DESARROLLO



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

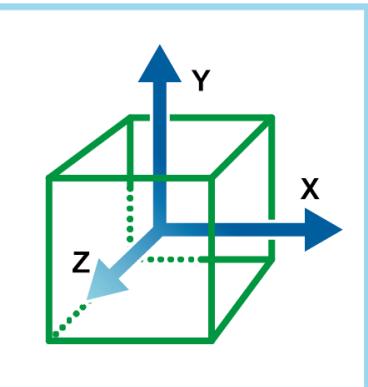
Triángulos: Clipping y sorting

.OBJ y debugging

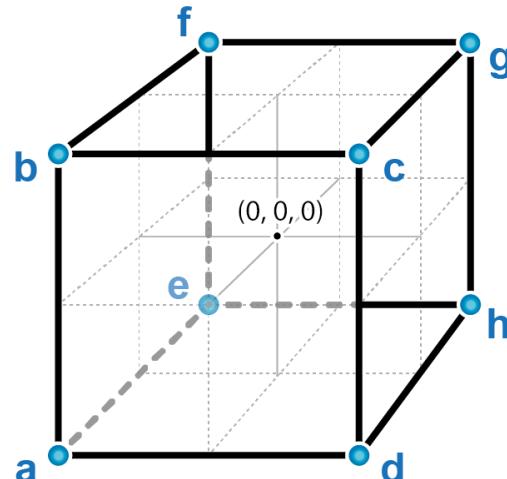
## 4. BLOQUE 2, 3NGINE: DESARROLLO

### Cubo de testeo

Ejes de la escena:



Vértices del cubo:



Coordenadas

- a.  $(-1, -1, 1)$
- b.  $(-1, 1, 1)$
- c.  $(1, 1, 1)$
- d.  $(1, -1, 1)$
- e.  $(-1, -1, -1)$
- f.  $(-1, 1, -1)$
- g.  $(1, 1, -1)$
- h.  $(1, -1, -1)$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### Transformaciones de los vértices

#### Fase 1: Transformación del objeto

- 1 Rotación del objeto en el eje Y (*objRotY*)
- 2 Rotación del objeto en el eje Z (*objRotZ*)
- 3 Rotación del objeto en el eje X (*objRotX*)
- 4 Desplazamiento del objeto (*objTraslation*)

#### Fase 2: Transformación de la cámara

- 5 Desplazamiento de la cámara (*camTraslation*)
- 6 Rotación en el eje Z de la cámara (*camRotZ*)
- 7 Rotación en el eje X de la cámara (*camRotX*)
- 8 Rotación en el eje Y de la cámara (*camRotY*)

#### Fase 3: Proyección (paso de 3D a 2D)

- 9 Proyección ortogonal de la cámara (*camProj*)
- 10 Paso de proyección ortogonal a perspectiva
- 11 Escalar a la ventana (*scaleView*)

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### Transformaciones de los vértices

#### Fase 1: Transformación del objeto

- 1 Rotación del objeto en el eje Y (*objRotY*)
- 2 Rotación del objeto en el eje Z (*objRotZ*)
- 3 Rotación del objeto en el eje X (*objRotX*)
- 4 Desplazamiento del objeto (*objTraslation*)

#### Fase 2: Transformación de la cámara

- 5 Desplazamiento de la cámara (*camTraslation*)
- 6 Rotación en el eje Z de la cámara (*camRotZ*)
- 7 Rotación en el eje X de la cámara (*camRotX*)
- 8 Rotación en el eje Y de la cámara (*camRotY*)

#### Fase 3: Proyección (paso de 3D a 2D)

- 9 Proyección ortogonal de la cámara (*camProj*)
- 10 Paso de proyección ortogonal a perspectiva
- 11 Escalar a la ventana (*scaleView*)

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

```
// VARIABLES NECESARIAS
float α = 0; // ángulo de rotación en el eje X (radianes)
float θ = 0; // ángulo de rotación en el eje Y (radianes)
float φ = 0; // ángulo de rotación en el eje Z (radianes)
float cX = cos(α), sX = sen(α); // seno y coseno de α
float cY = cos(θ), sY = sen(θ); // seno y coseno de θ
float cZ = cos(φ), sZ = sen(φ); // seno y coseno de φ
float dX = 0; // traslación a aplicar en X (píxeles)
float dY = 0; // traslación a aplicar en Y (píxeles)
float dZ = 0; // traslación a aplicar en Z (píxeles)
```

$$\text{Matriz genérica} = \text{Matriz de rotación en Z} \times \text{Matriz de rotación en Y} \times \text{Matriz de rotación en X} \times \text{Matriz de traslación}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} = \begin{bmatrix} cX & -sX & 0 & 0 \\ sX & cX & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cX & 0 & sX & 0 \\ 0 & 1 & 0 & 0 \\ -sX & 0 & cX & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cX & -sX & 0 \\ 0 & sX & cX & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -dX \\ 0 & 1 & 0 & -dY \\ 0 & 0 & 1 & -dZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeо

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

### loopFunction()

#### Transformaciones de los vértices

Fase 1:  
Transformación  
del objeto

- 1 Rotación del objeto en el eje Y (*objRotY*)
- 2 Rotación del objeto en el eje Z (*objRotZ*)
- 3 Rotación del objeto en el eje X (*objRotX*)
- 4 Desplazamiento del objeto (*objTraslation*)

Fase 2:  
Transformación  
de la cámara

- 5 Desplazamiento de la cámara (*camTraslation*)
- 6 Rotación en el eje Z de la cámara (*camRotZ*)
- 7 Rotación en el eje X de la cámara (*camRotX*)
- 8 Rotación en el eje Y de la cámara (*camRotY*)

Fase 3:  
Proyección (paso  
de 3D a 2D)

- 9 Proyección ortogonal de la cámara (*camProj*)
- 10 Paso de proyección ortogonal a perspectiva
- 11 Escalar a la ventana (*scaleView*)

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

loopFunction()

Matriz  
genérica

Traslación  
de cámara

Rotación de la  
cámara en Z

Rotación de la  
cámara en Y

Rotación de la  
cámara en X

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -dX \\ 0 & 1 & 0 & -dY \\ 0 & 0 & 1 & -dZ \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cX & -sX & 0 & 0 \\ sX & cX & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cX & 0 & sX & 0 \\ 0 & 1 & 0 & 0 \\ -sX & 0 & cX & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cX & -sX & 0 \\ 0 & sX & cX & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### Transformaciones de los vértices

Fase 1:  
Transformación  
del objeto

- 1 Rotación del objeto en el eje Y (*objRotY*)
- 2 Rotación del objeto en el eje Z (*objRotZ*)
- 3 Rotación del objeto en el eje X (*objRotX*)
- 4 Desplazamiento del objeto (*objTraslation*)

Fase 2:  
Transformación  
de la cámara

- 5 Desplazamiento de la cámara (*camTraslation*)
- 6 Rotación en el eje Z de la cámara (*camRotZ*)
- 7 Rotación en el eje X de la cámara (*camRotX*)
- 8 Rotación en el eje Y de la cámara (*camRotY*)

Fase 3:  
Proyección (paso  
de 3D a 2D)

- 9 Proyección ortogonal de la cámara (*camProj*)
- 10 Paso de proyección ortogonal a perspectiva
- 11 Escalar a la ventana (*scaleView*)

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### 1 Coordenadas ortogonales del vértice (matriz de proyección)

```
// VARIABLES NECESARIAS
float sX = 1.28f; // ancho del sensor de la cámara
float sY = 0.72f; // altura del sensor de la cámara
float fL = 1f;    // distancia focal de la cámara
float vX = fL * width / 2 * sX; // factor de conversión en X
float vY = fL * height / 2 * sY; // factor de conversión en Y
```

Coordenadas  
ortogonales  
del vértice

$$\begin{bmatrix} \text{Vértice a} \\ \text{proyectar} \end{bmatrix} = \begin{bmatrix} \text{X} \\ \text{Y} \\ \text{Z} \\ 1 \end{bmatrix} \times \begin{bmatrix} \text{Matriz de proyección} \\ \text{vX} & 0 & 0 & 0 \\ 0 & \text{vY} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

loopFunction()

### 2) Coordenadas del vértice con perspectiva

```
// Cálculo de la perspectiva (xP e yP)
float xP = x / z;
float yP = y / z;
// Crear otro Vertex con los nuevos valores
Vertex vProjection = new Vertex(xP, yP, 0f);
```

$$\begin{bmatrix} \mathbf{xP} \\ \mathbf{yP} \\ 0 \\ 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

### loopFunction()

#### 3 Escalado final del vértice a las dimensiones de la ventana

```
// VARIABLES NECESARIAS
float width = 1280f; // ancho de la ventana (píxeles)
float height = 720f; // altura de la ventana (píxeles)
```

Vértice  
final, con  
escalado

$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_P \\ y_P \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & \text{width}/2 \\ 0 & -1 & 0 & \text{height}/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

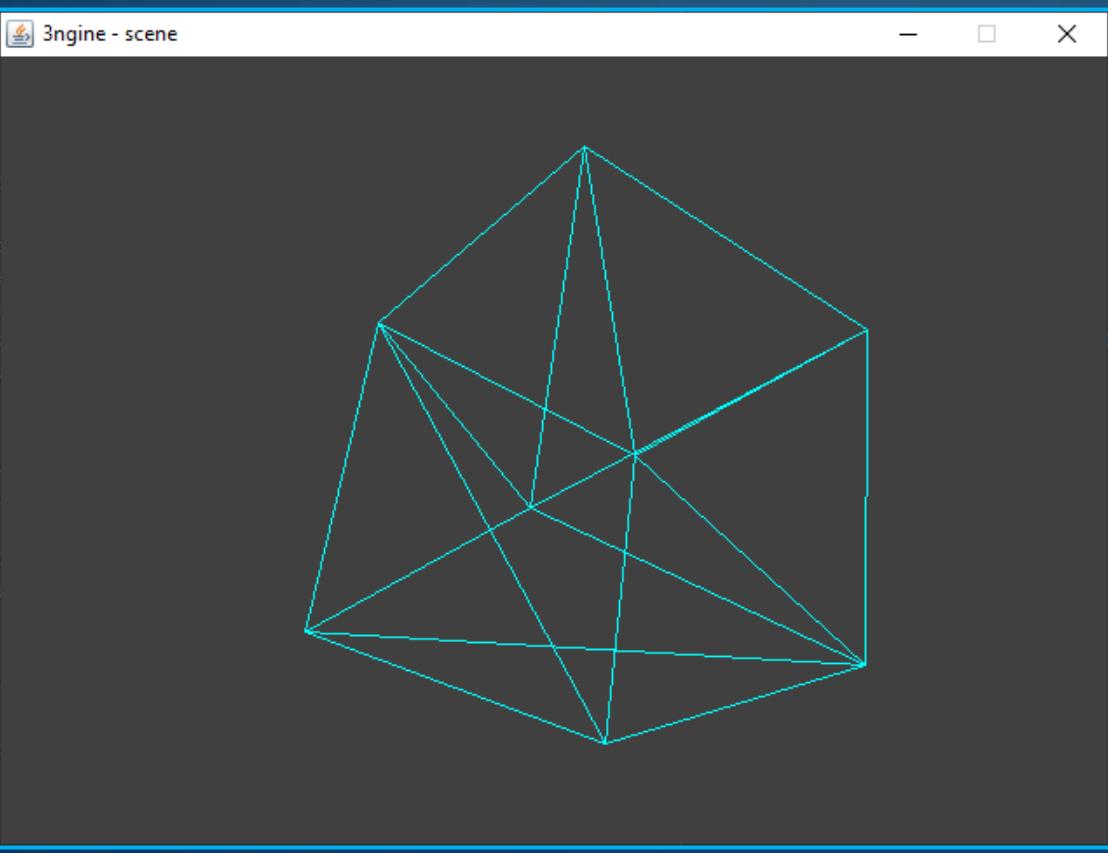
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

Vertex

Dirección de la luz

FLAT-SHADING



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

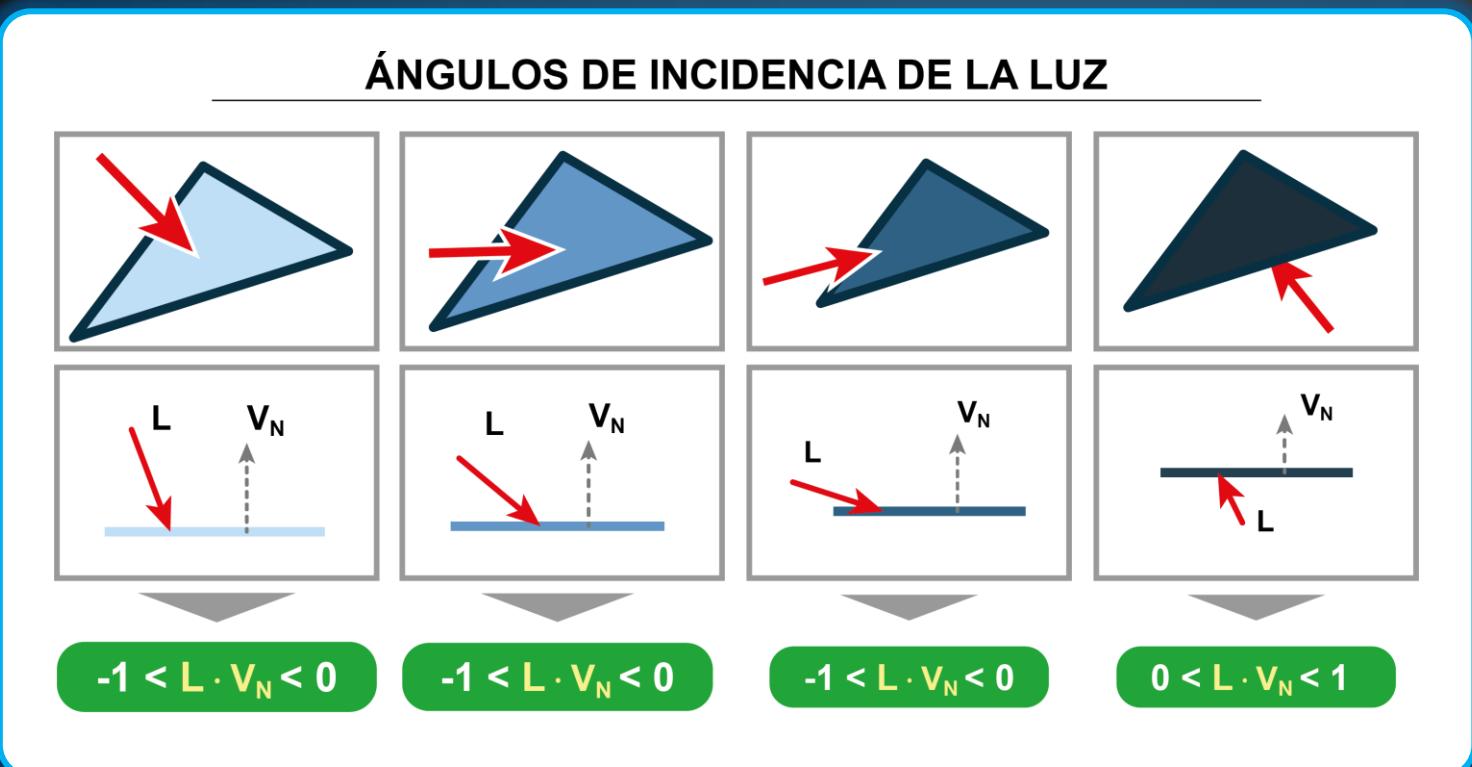
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

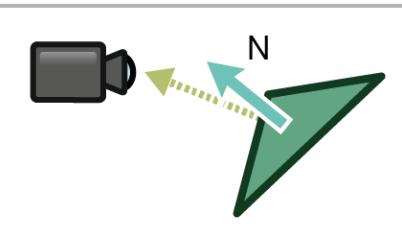
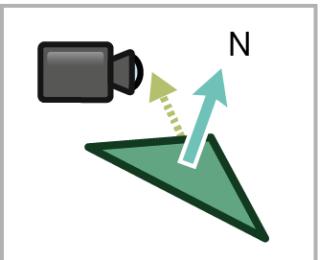
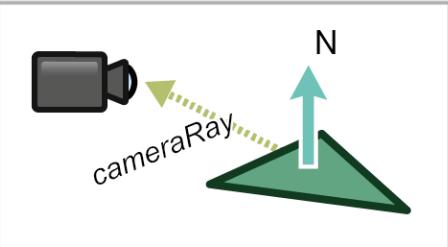
Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

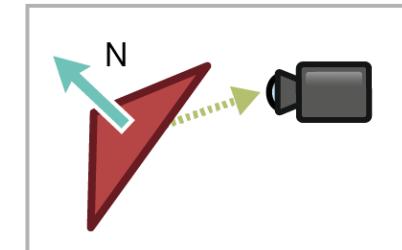
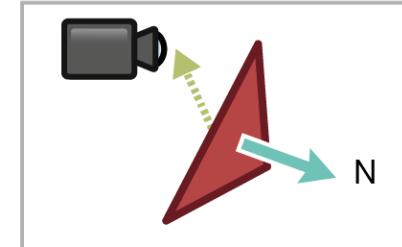
\*Optimización\*

### Triángulo visible



$$N \cdot \text{cameraRay} \geq 0$$

### Triángulo invisible



$$N \cdot \text{cameraRay} < 0$$

checkIfFacingCamera()

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

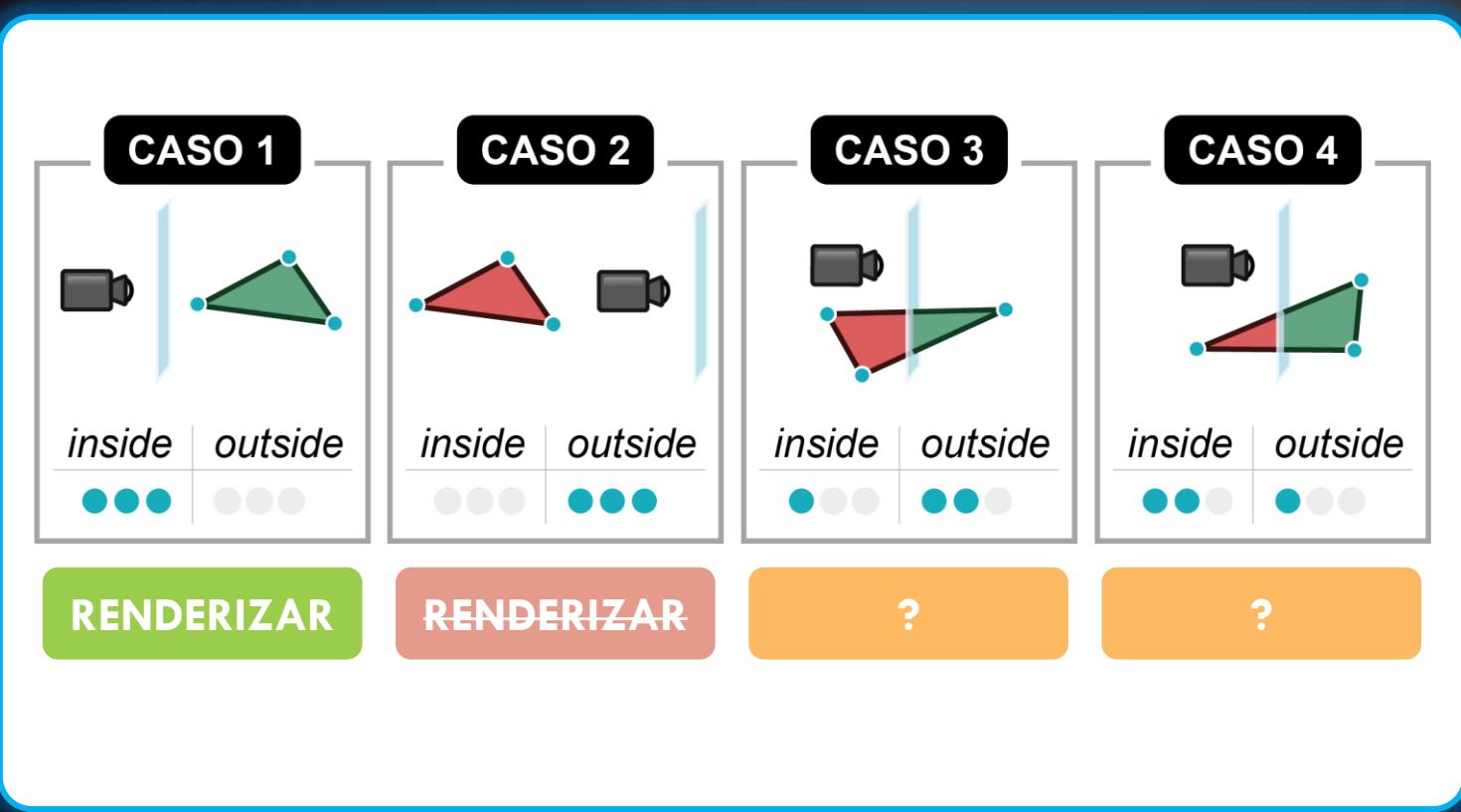
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



RENDERIZAR

RENDERIZAR

?

?

MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

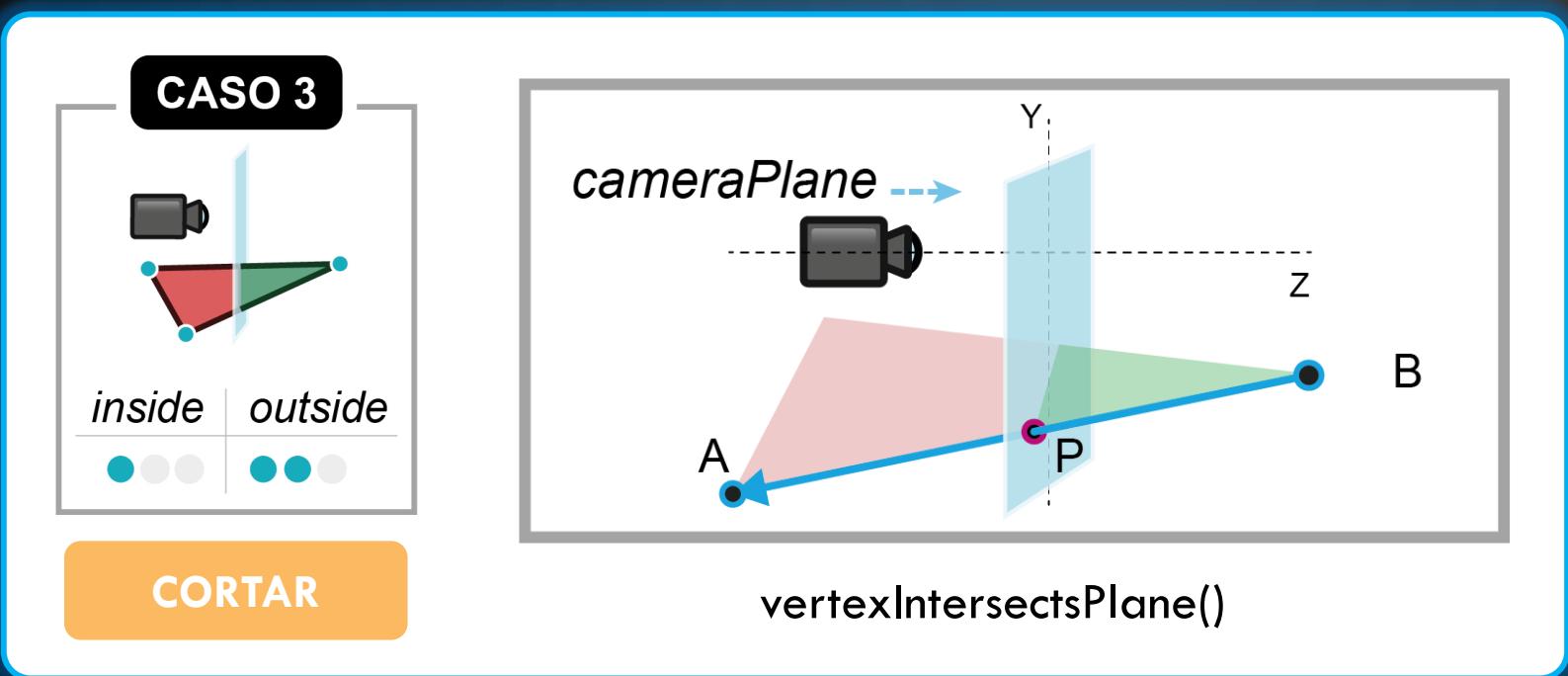
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO



(páginas 115 y 116 de la guía)

MOTOR DE JUEGOS FROM SCRATCH

## Proyecto + Exec. thread

## Cubo de testeo

## Vértices: transformación

## Luz direccional

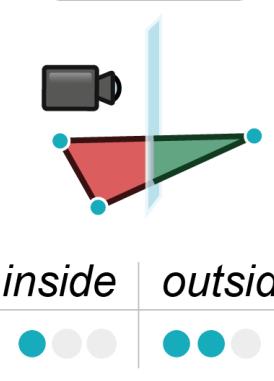
## Triángulos: Camera-facing

## Triángulos: Clipping y sorting

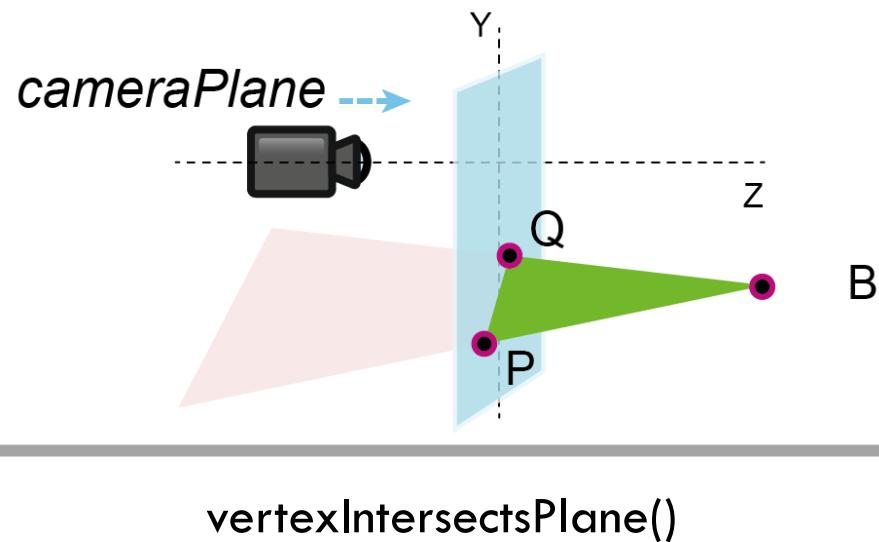
## .OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

CASO 3



CORTAR



(páginas 115 y 116 de la guía)

MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

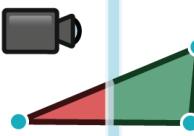
Triángulos: Clipping y sorting

.OBJ y debugging

50

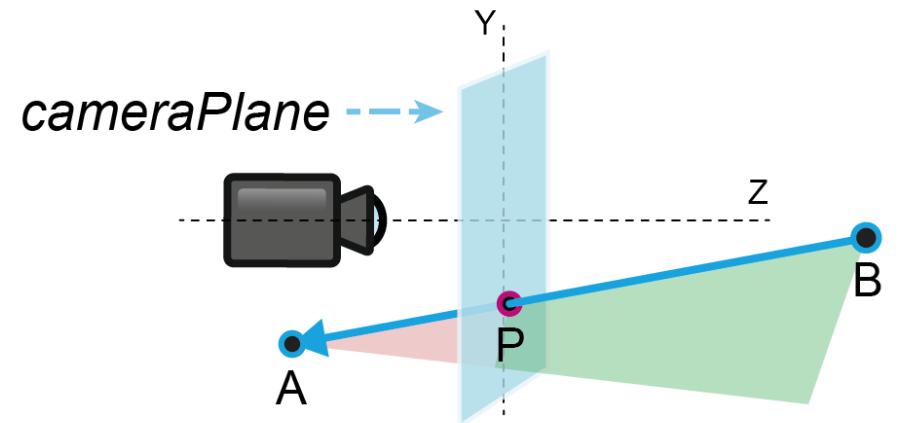
## 4. BLOQUE 2, 3NGINE: DESARROLLO

CASO 4



inside outside

CORTAR



(páginas 115 y 116 de la guía)

MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

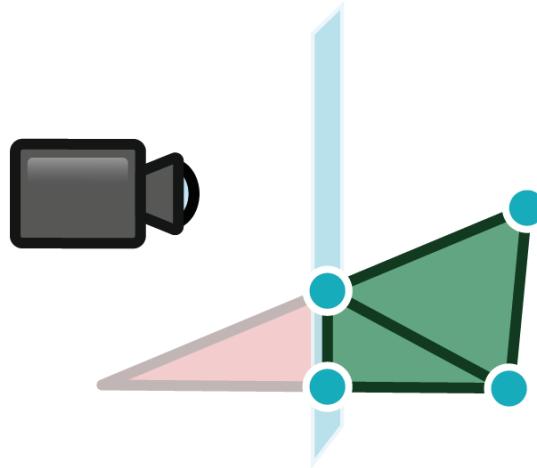
CASO 4



inside outside



CORTAR



(páginas 115 y 116 de la guía)

MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

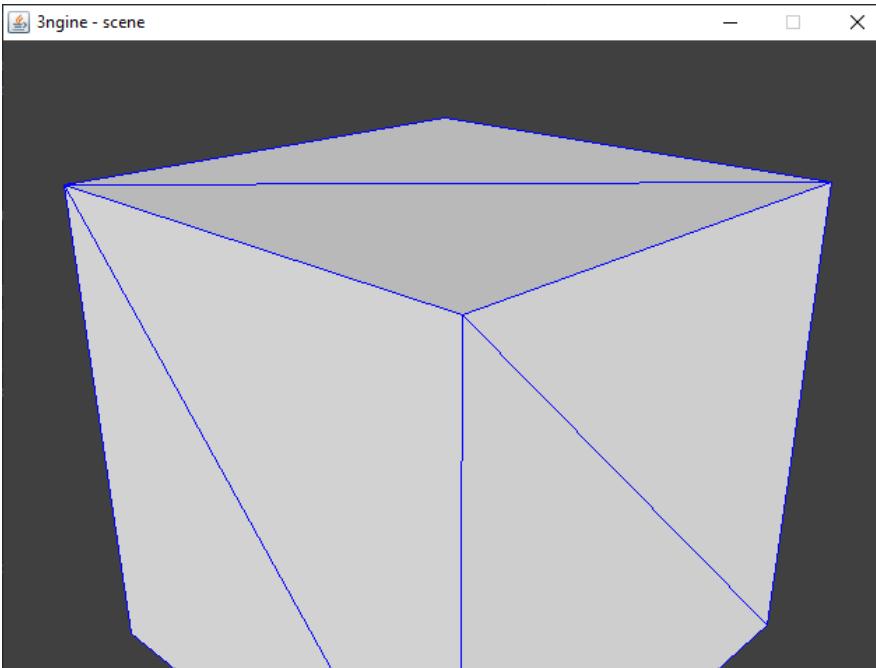
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



`cameraPlane = (0, 0, -0.1)`

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

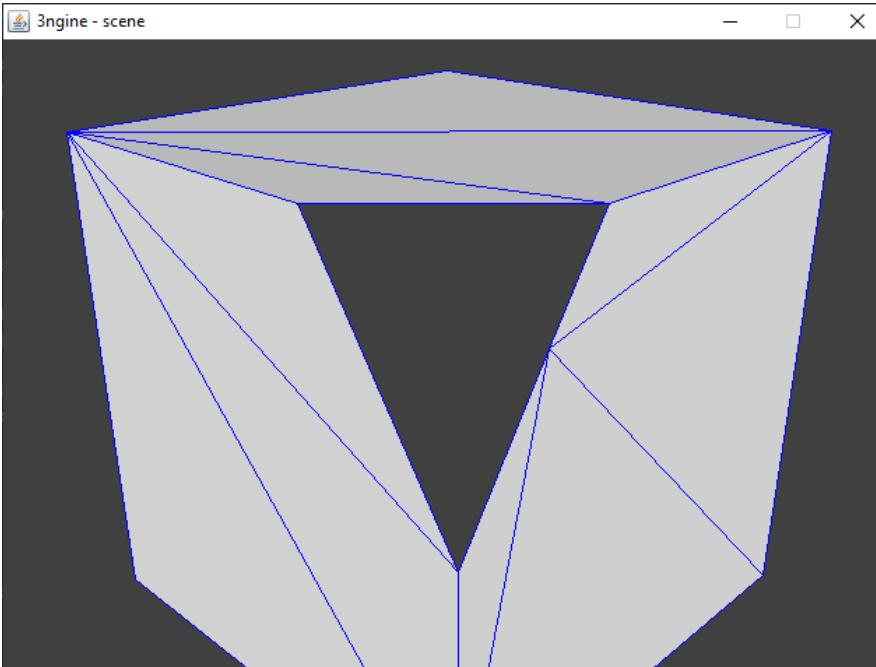
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



`cameraPlane = (0, 0, -3)`

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

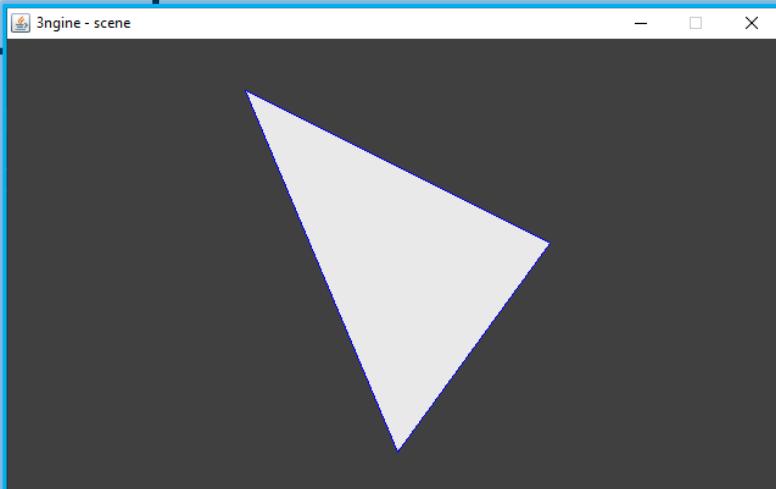
## 4. BLOQUE 2, 3NGINE: DESARROLLO

**simple\_triangle.obj**

```
# Triángulo de prueba
v -1.000000 1.000000 -1.000000
v 0.000000 -1.000000 1.000000
v 1.000000 0.000000 -1.000000
f 1 2 3
```

**etiquetas:**

```
# >> comentario
v >> vértice
f >> triángulos
```



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

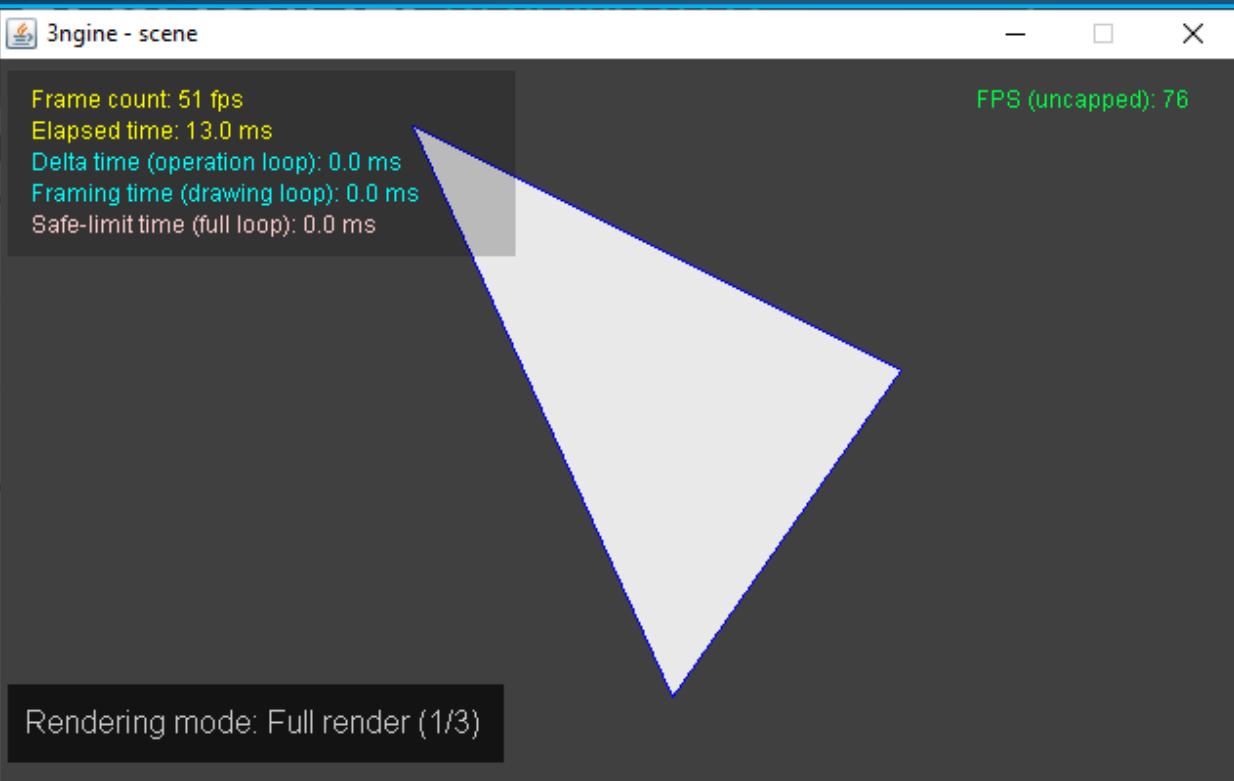
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

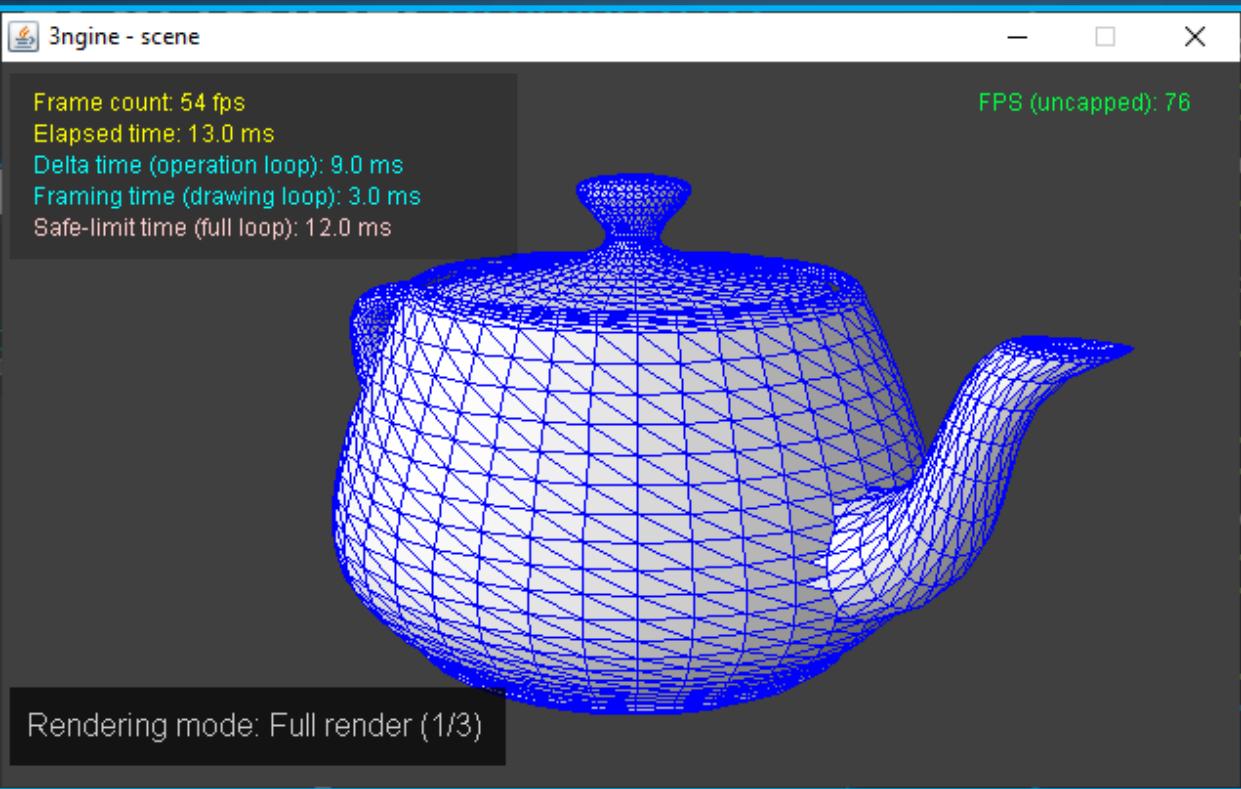
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

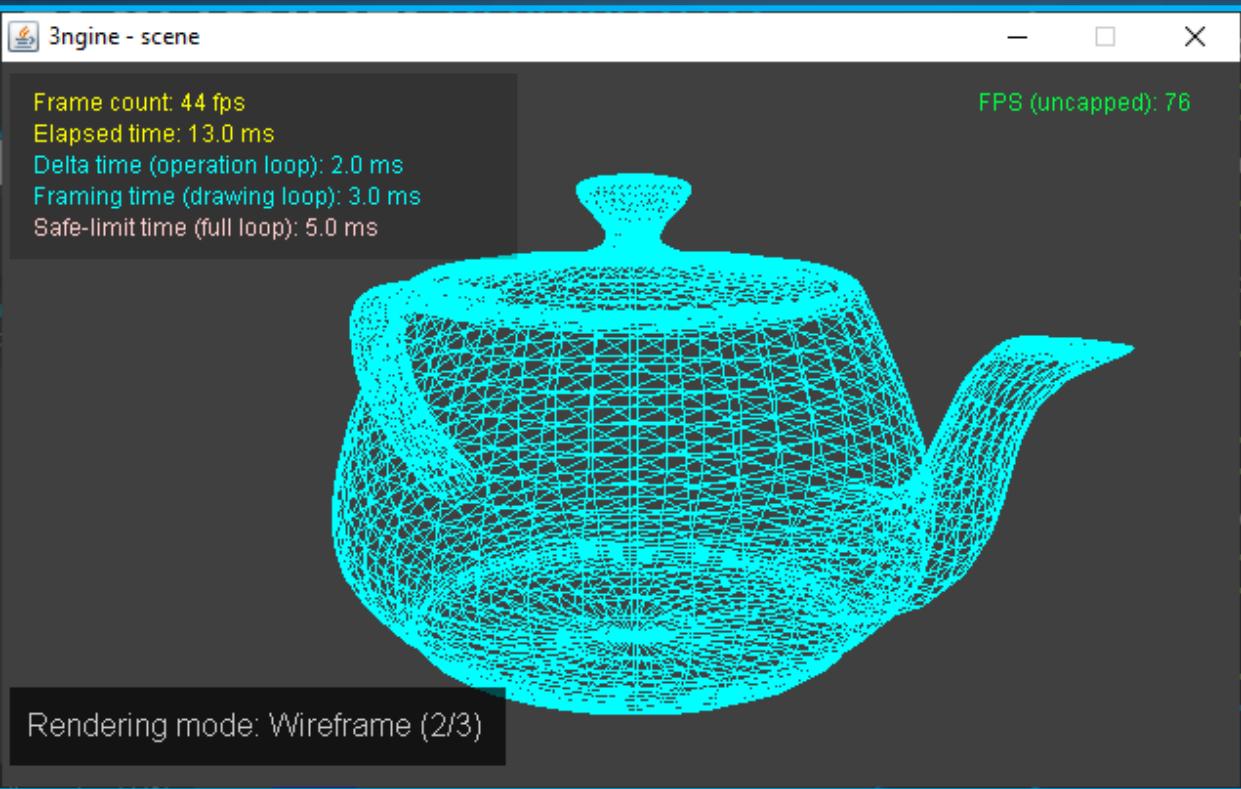
Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging

## 5. CONCLUSIONES: RESULTADOS

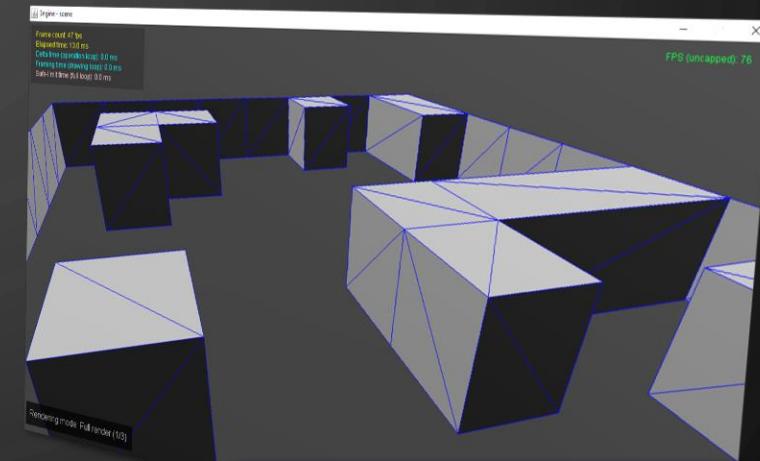


**Raycaster (web)**

100% adaptable

Simulación 3D *low-spec*

¡Listo para la acción!



**Motor 3D (desktop)**

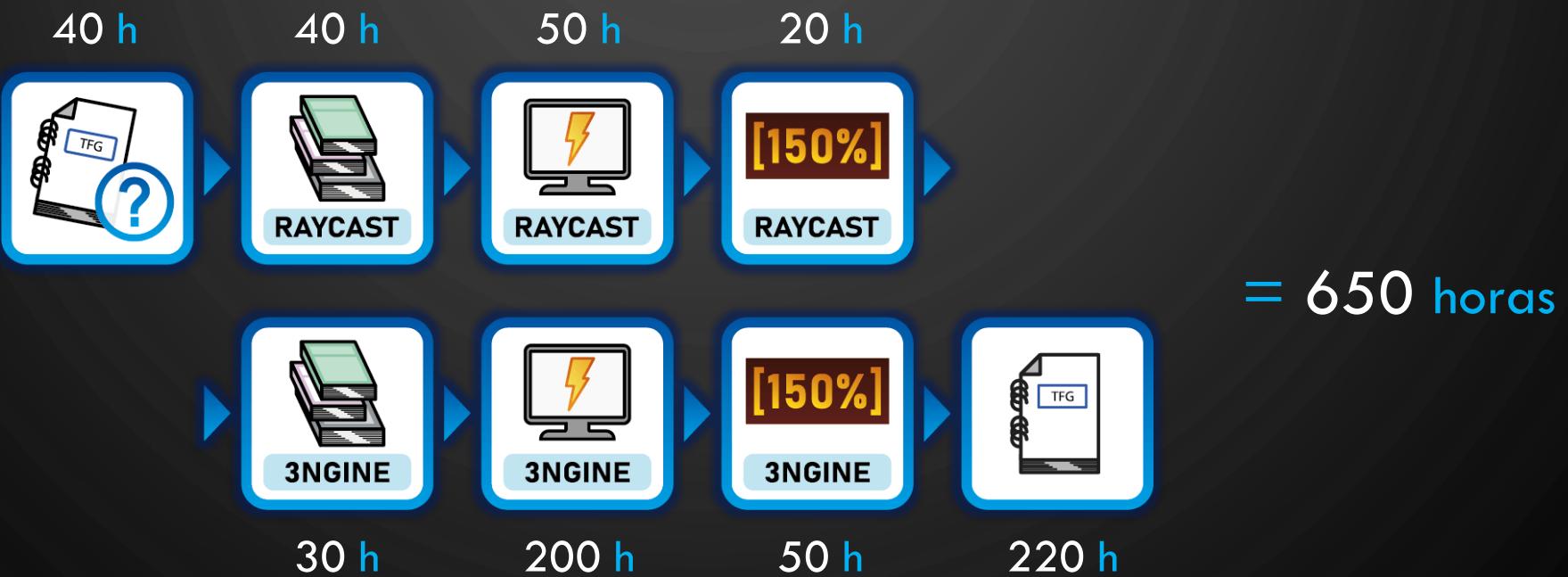
Herramientas de análisis

Lectura de .obj

Completamente de cero



## 5. CONCLUSIONES: COSTE EN HORAS



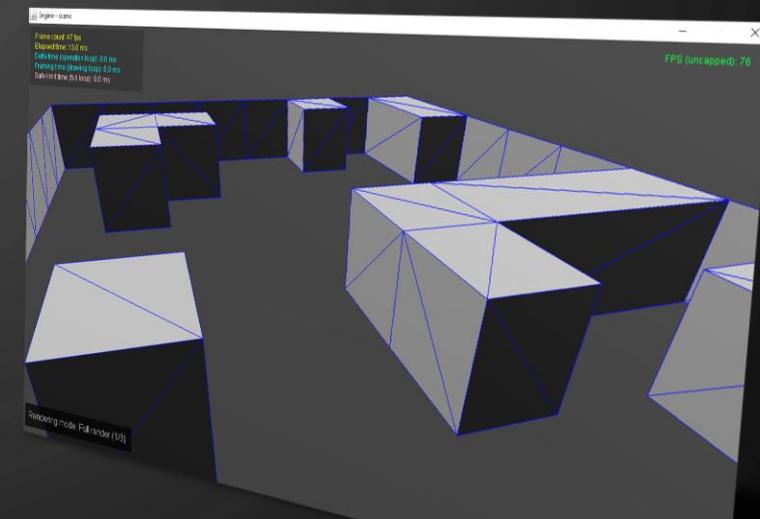
MOTOR DE JUEGOS FROM SCRATCH

60

## 5. CONCLUSIONES: LÍNEAS DE CONTINUACIÓN

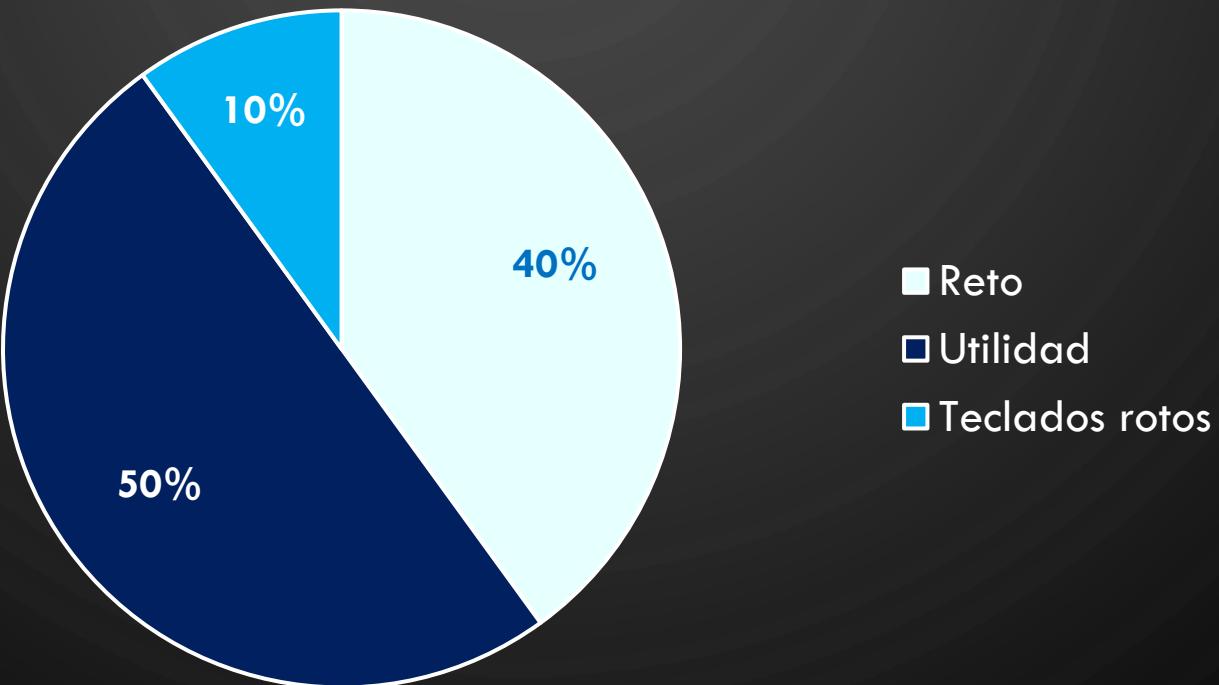


- Muros de distinta altura
- Mapeado cuadricular
- Transformaciones en el entorno



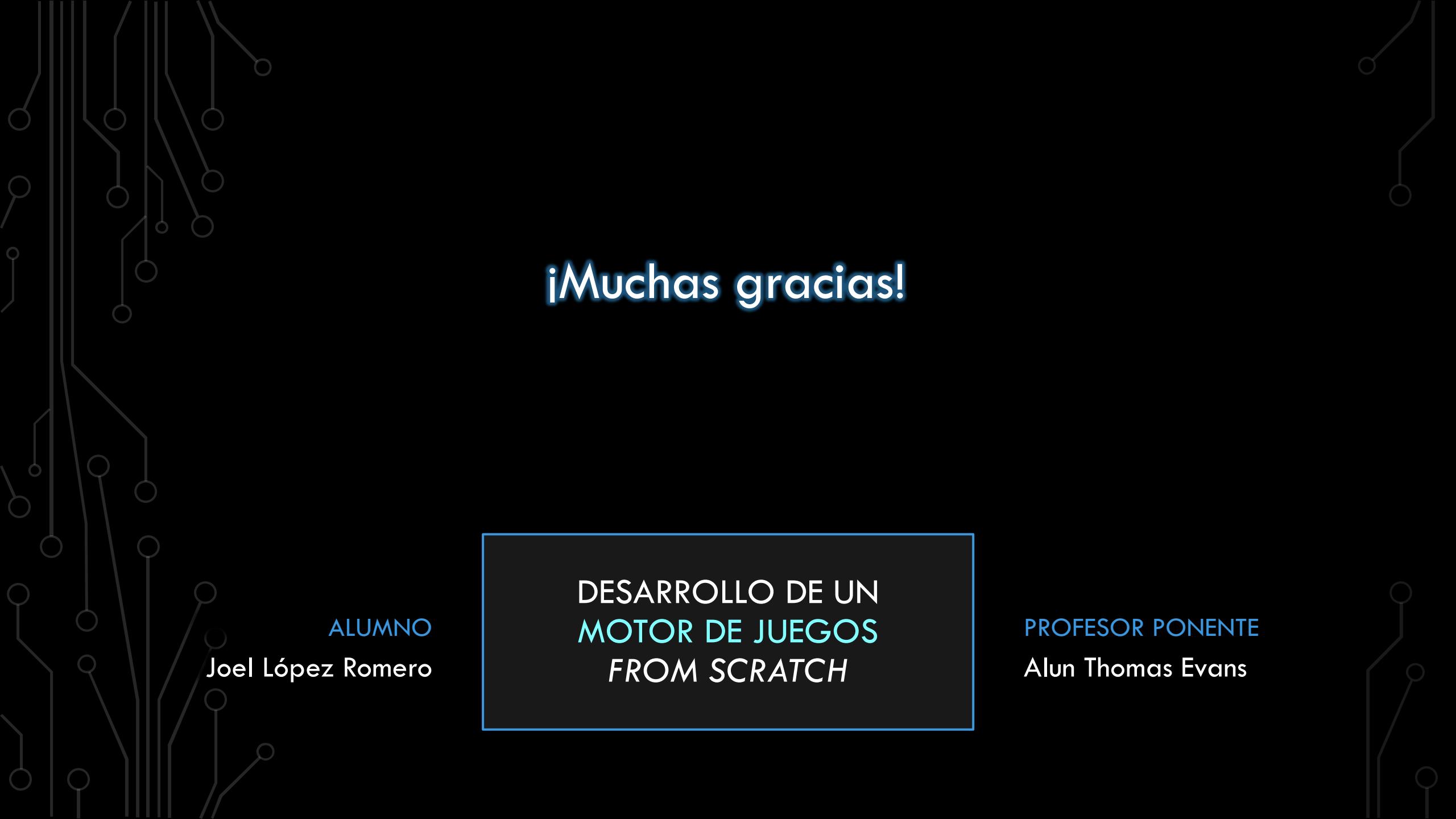
- Texturas
- Más tipos de *shading*/luces
- Partículas / postprocesado

## 5. CONCLUSIONES: EXPERIENCIA



MOTOR DE JUEGOS FROM SCRATCH

62



# ¡Muchas gracias!

ALUMNO

Joel López Romero

DESARROLLO DE UN  
MOTOR DE JUEGOS  
*FROM SCRATCH*

PROFESOR PONENTE

Alun Thomas Evans

## 6. ANEXOS

MOTOR DE JUEGOS FROM SCRATCH

64

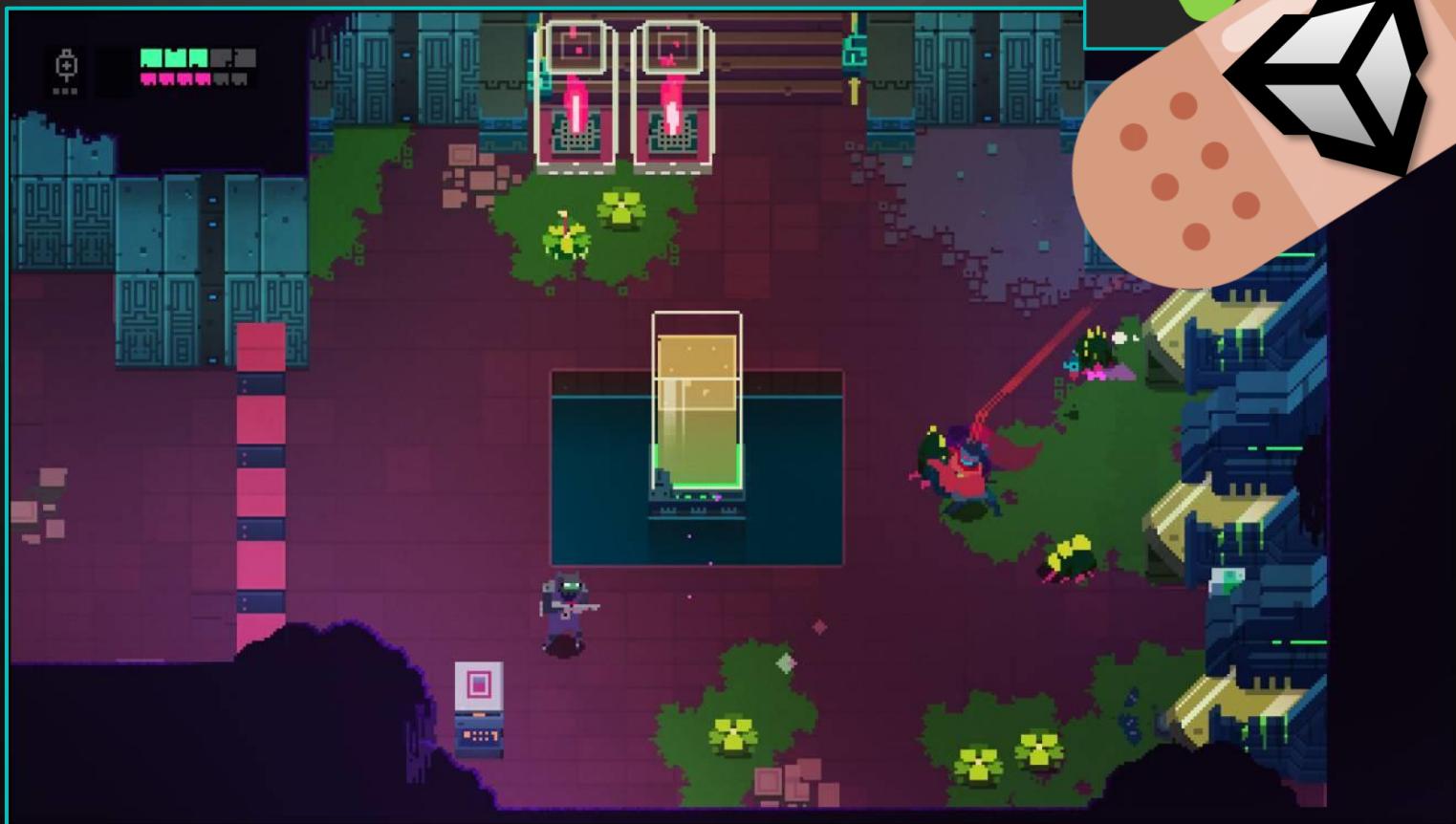
## 6. ANEXOS: GRADO MULTIMEDIA (+ VIDEOJUEGOS)

- ✓ Programación 1 ([CodeLite](#), [C](#))
- ✓ Bases de Datos + DPO ([Java](#))
- ✓ Gráficos 1 y 2 ([Visual Studio](#), [C++](#))
- ✓ Programación de videojuegos ([Visual Studio](#), [C](#))
- ✓ Producción de videojuegos ([Unity](#), [C#](#))
- ✓ IRV ([Unreal Engine](#))
- ✓ Proyectos web ([JavaScript](#))

MOTOR DE JUEGOS FROM SCRATCH

## 6. ANEXOS

(Game Maker Studio)



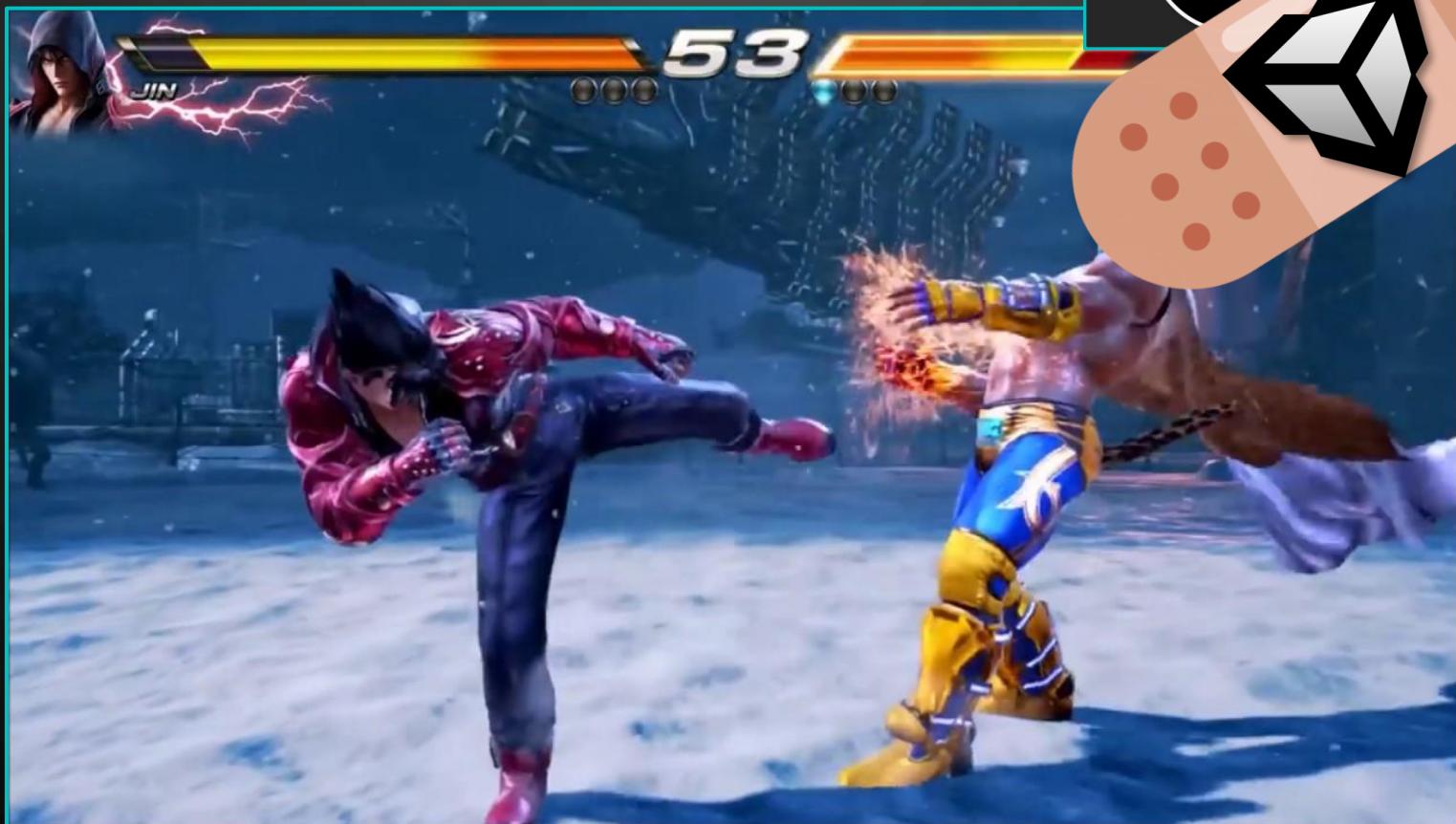
## 6. ANEXOS

CryEngine 3



## 6. ANEXOS

(Unreal Engine 4)

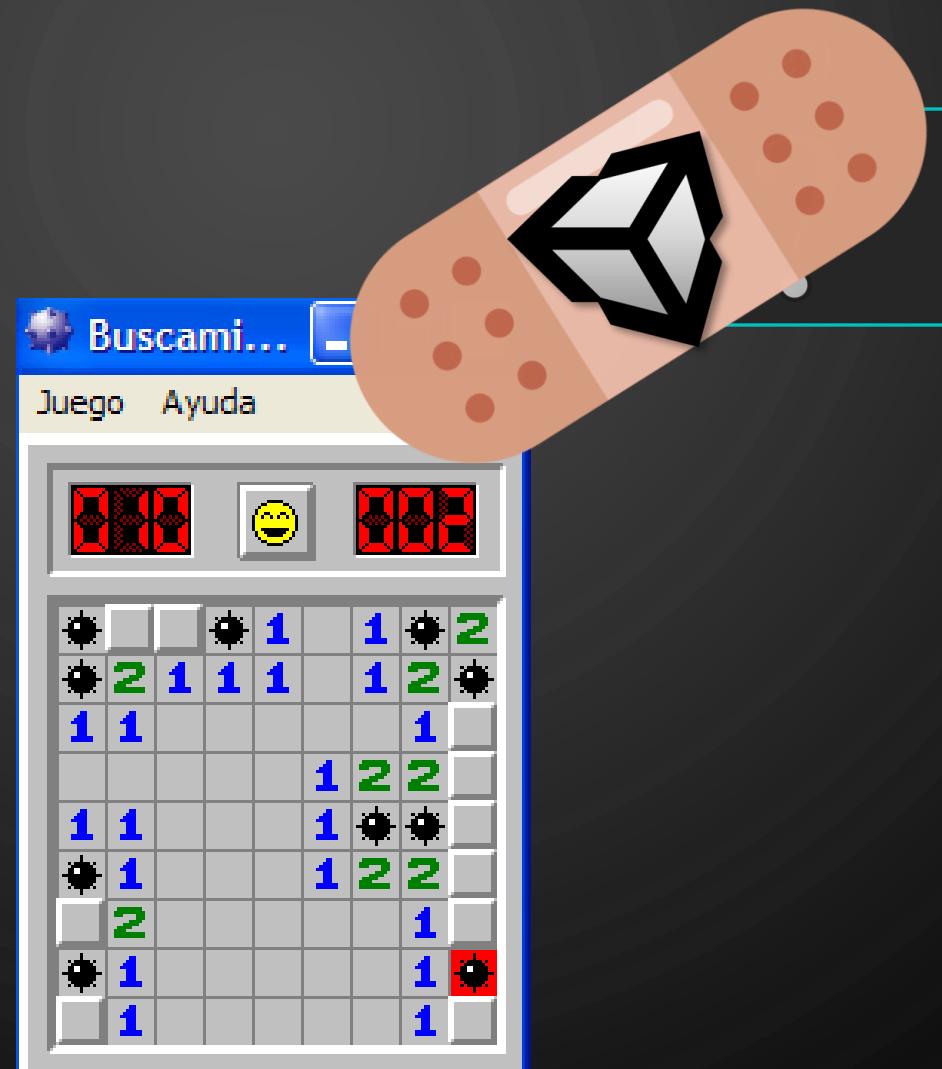


68

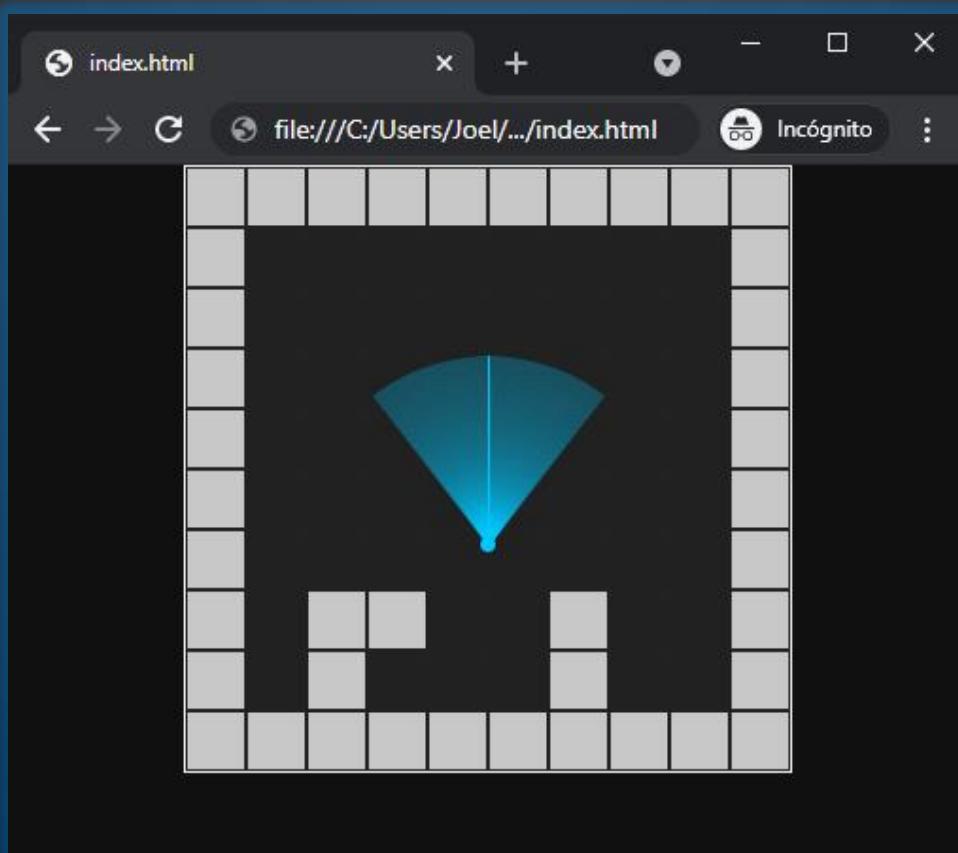
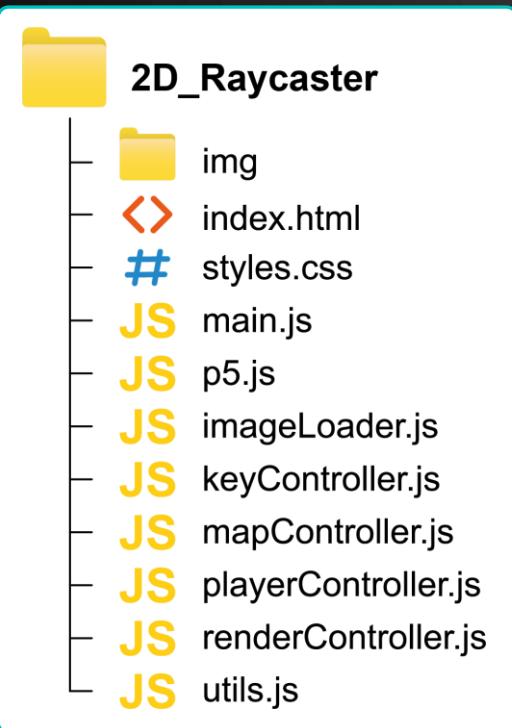
## 6. ANEXOS



## 6. ANEXOS



### 3. BLOQUE 1, RAYCASTER: DESARROLLO (ESTADO INICIAL)



MOTOR DE JUEGOS FROM SCRATCH

Proyecto + Mapa

Raycasting (Player)

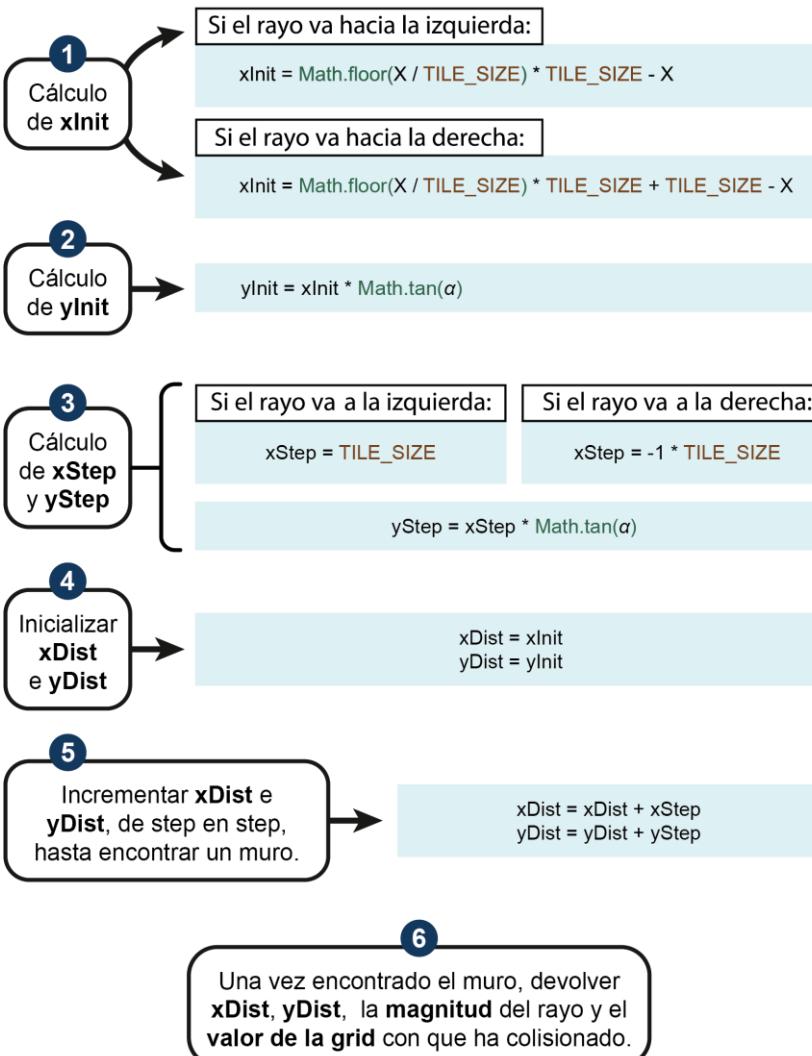
Raycasting (colisiones)

Proyección (vLines)

Texturizado

# 3. BLOQUE 1, RAYCASTER: DESARROLLO (COLISIONES EN X)

## PROCESO DE CÁLCULO DE LAS COLISIONES HORIZONTALES:



Proyecto + Mapa

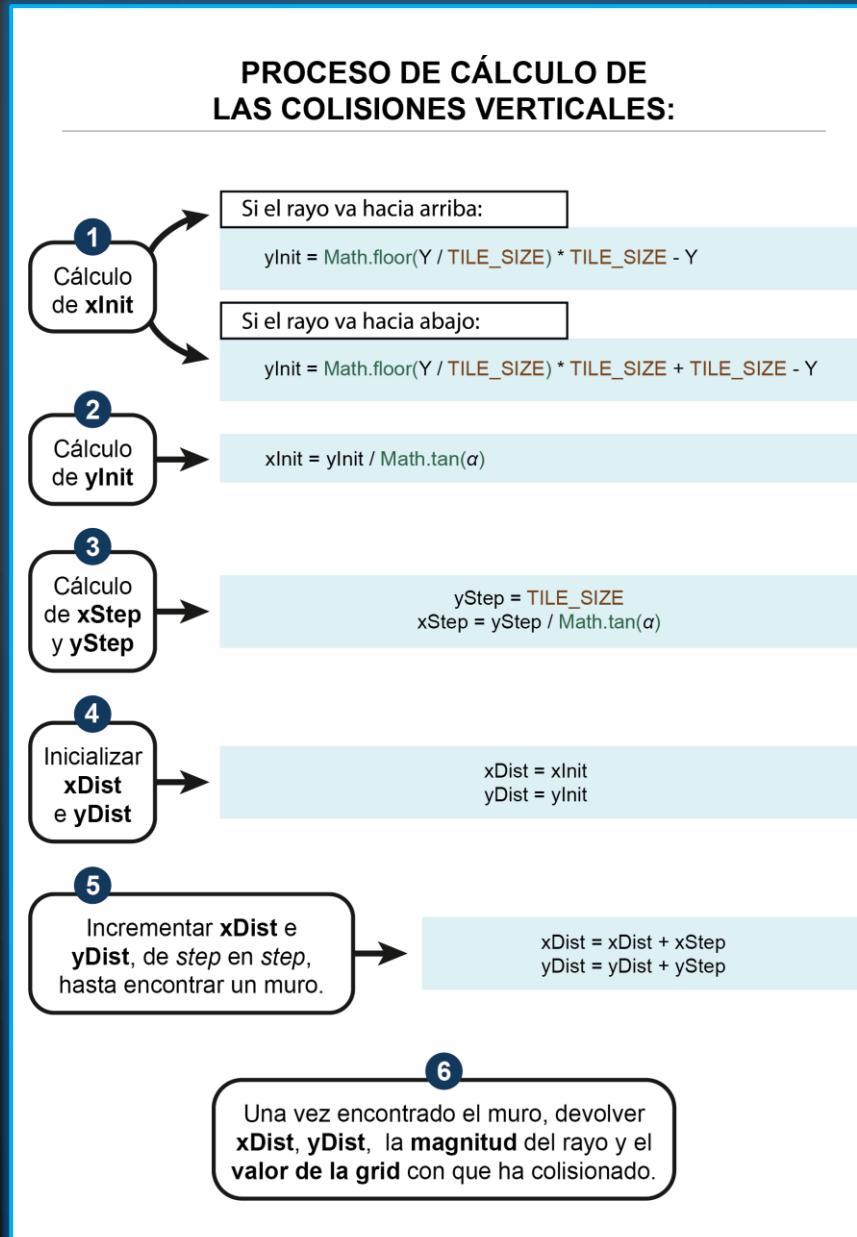
Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO (COLISIONES EN Y)



Proyecto + Mapa

Raycasting (Player)

Raycasting (colisiones)

Proyección (vLines)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO (TEXELLING, 1)

**PASO 1** Calcular píxel de colisión del rayo (respecto a `TILE_SIZE`)

The diagram shows a 4x4 grid of tiles. A player character is at position `(player.x, player.y)`. A ray is cast from the player with a direction vector `ray.dY` (vertical component) and `ray.dX` (horizontal component). The ray intersects a green-tiled wall. The collision point is calculated relative to the `TILE_SIZE`. A red dot marks the collision point on the wall.

**¡Recordar!** `dX` y `dY` son longitudes del rayo, no coordenadas.

Si la colisión es horizontal:  
(como en el ejemplo)

`TILE_SIZE` (= 32 px)  $\approx 29$  px

Si la colisión fuera vertical:

`TILE_SIZE` (= 32 px)  $\approx 14$  px

Coordenada (en píxeles) en que se encuentra la colisión.

Proyecto + Mapa

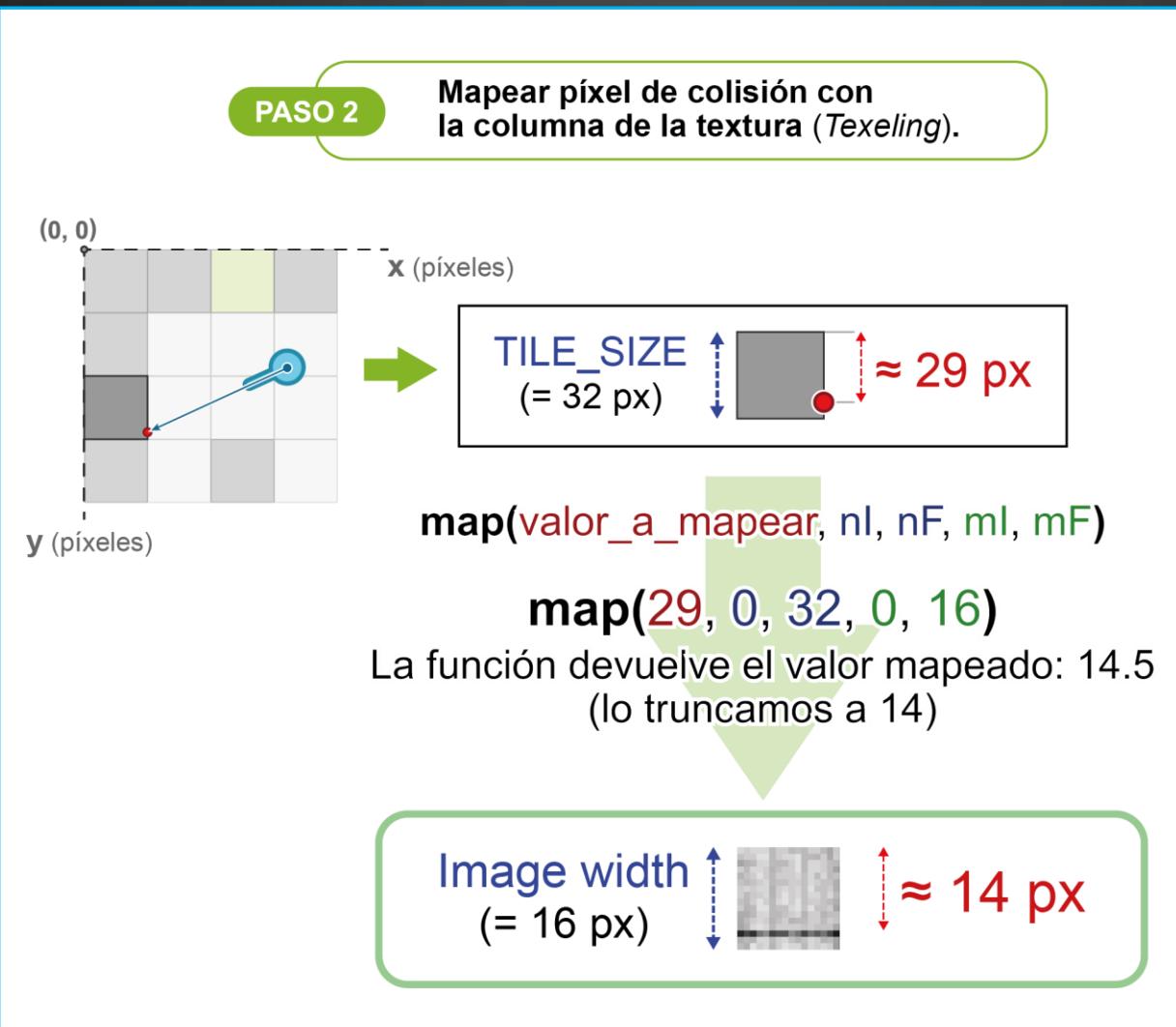
Raycasting (Player)

Raycasting (colisiones)

Proyección (`vLines`)

Texturizado

### 3. BLOQUE 1, RAYCASTER: DESARROLLO (TEXELLING, 2)



Proyecto + Mapa

Raycasting (Player)

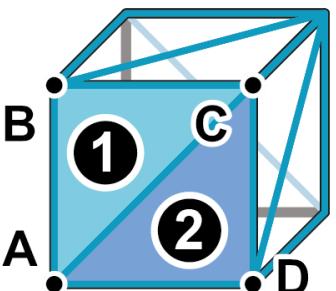
Raycasting (colisiones)

Proyección (vLines)

Texturizado

## 4. BLOQUE 2, 3NGINE: DESARROLLO

### Cubo de testeo



- A.  $(-1, -1, 1)$
- B.  $(-1, 1, 1)$
- C.  $(1, 1, 1)$
- D.  $(1, -1, 1)$

1  
new Triangle(  
    new Vertex(-1, -1, 1),  
    new Vertex(-1, 1, 1),  
    new Vertex( 1, 1, 1)  
);  
→ A  
→ B  
→ C

2  
new Triangle(  
    new Vertex(-1, -1, 1),  
    new Vertex( 1, 1, 1),  
    new Vertex( 1, -1, 1)  
);  
→ A  
→ C  
→ D

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### Transformaciones de los vértices

Fase 1:  
Transformación  
del objeto

- 1 Rotación del objeto en el eje Y (*objRotY*)
- 2 Rotación del objeto en el eje Z (*objRotZ*)
- 3 Rotación del objeto en el eje X (*objRotX*)
- 4 Desplazamiento del objeto (*objTraslation*)

Fase 2:  
Transformación  
de la cámara

- 5 Desplazamiento de la cámara (*camTraslation*)
- 6 Rotación en el eje Z de la cámara (*camRotZ*)
- 7 Rotación en el eje X de la cámara (*camRotX*)
- 8 Rotación en el eje Y de la cámara (*camRotY*)

Fase 3:  
Proyección (paso  
de 3D a 2D)

- 9 Proyección ortogonal de la cámara (*camProj*)
- 10 Paso de proyección ortogonal a perspectiva
- 11 Escalar a la ventana (*scaleView*)

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

# 4. BLOQUE 2, 3NGINE: DESARROLLO

## loopFunction()

### 1 Coordenadas ortogonales del vértice (matriz de proyección)

```
// VARIABLES NECESARIAS
float sX = 1.28f; // ancho del sensor de la cámara
float sY = 0.72f; // altura del sensor de la cámara
float fL = 1f; // distancia focal de la cámara
float vX = fL * width / 2 * sX; // factor de conversión en X
float vY = fL * height / 2 * sY; // factor de conversión en Y
```

$$\begin{bmatrix} \text{Coordenadas ortogonales del vértice} & \rightarrow & \begin{matrix} \text{Vértice a proyectar} \\ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \end{matrix} & = & \begin{matrix} \text{Matriz de proyección} \\ \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix} \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

loopFunction()

### 2) Coordenadas del vértice con perspectiva

```
// Cálculo de la perspectiva (xP e yP)
float xP = x / z;
float yP = y / z;
// Crear otro Vertex con los nuevos valores
Vertex vProjection = new Vertex(xP, yP, 0f);
```

$$\begin{bmatrix} \mathbf{xP} \\ \mathbf{yP} \\ 0 \\ 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

### loopFunction()

#### 3) Escalado final del vértice a las dimensiones de la ventana

```
// VARIABLES NECESARIAS
float width = 1280f; // ancho de la ventana (píxeles)
float height = 720f; // altura de la ventana (píxeles)
```

Vértice  
final, con  
escalado

Vértice con  
perspectiva

Matriz de  
escalado

$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_P \\ y_P \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & \text{width}/2 \\ 0 & -1 & 0 & \text{height}/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

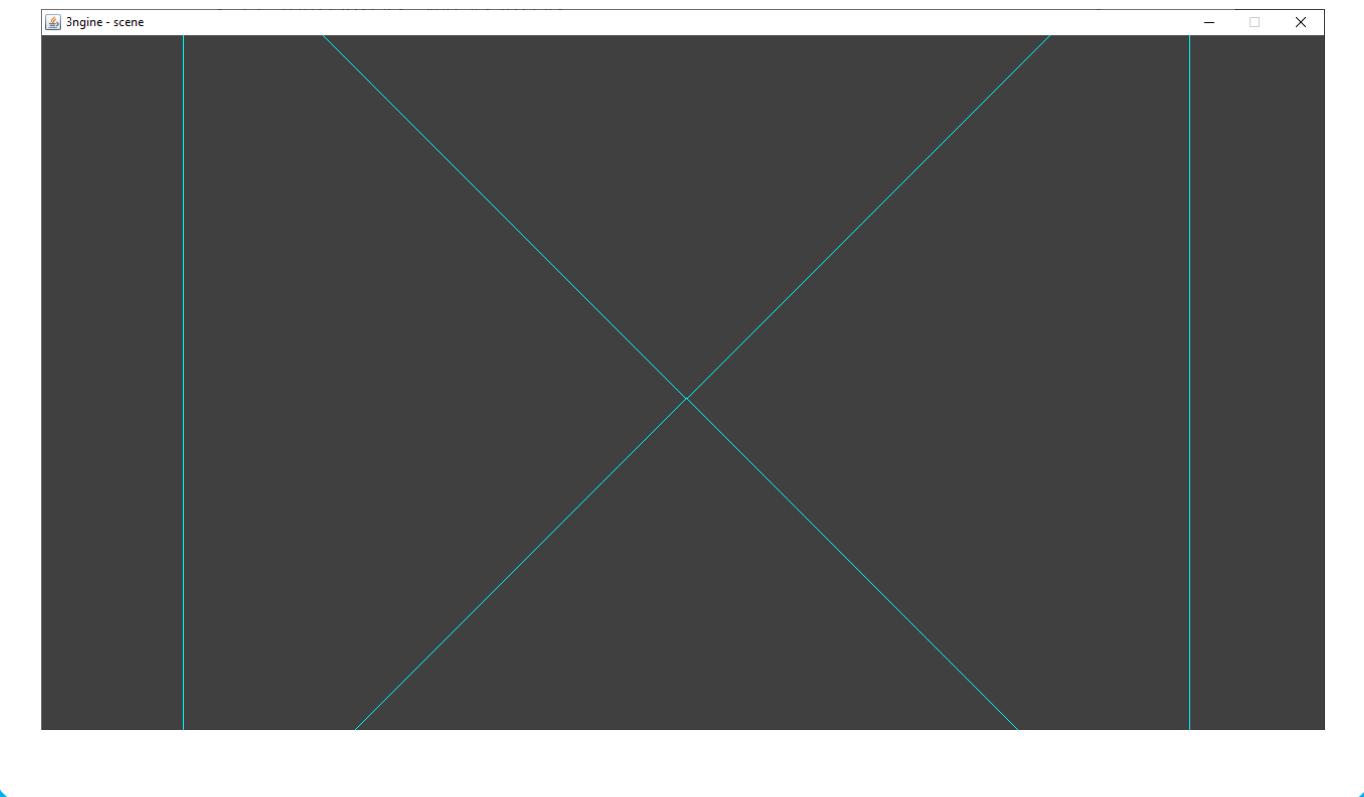
Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO

**loopFunction()**



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

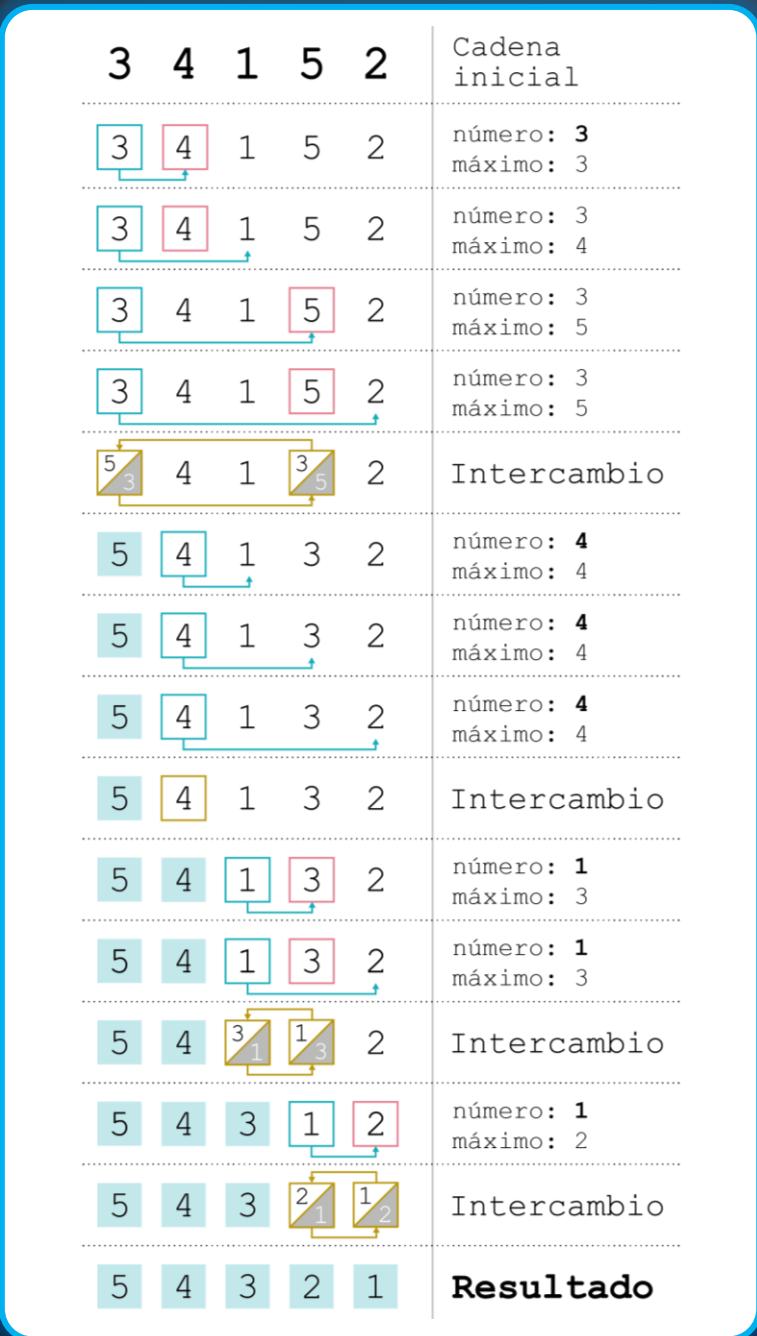
Luz direccional

Triángulos: Clipping y sorting

Triángulos: Camera-facing

.OBJ y debugging

## 4. BLOQUE 2, 3NGINE: DESARROLLO



Proyecto + Exec. thread

Cubo de testeo

Vértices: transformación

Luz direccional

Triángulos: Camera-facing

Triángulos: Clipping y sorting

.OBJ y debugging