



Cub3D

Mi primer RayCaster con la minilibX

Resumen: Este proyecto está inspirado en el juego epónimo mundialmente conocido, considerado como el primer FPS jamás realizado. Le permitirá explorar la técnica del ray-casting. Su objetivo será crear una vista dinámica dentro de un laberinto, en el que tendrá que encontrar la salida.

Índice general

I.	Preámbulo	2
II.	Objetivos	3
III.	Reglas comunes	4
IV.	Parte Obligatoria - Cub3D	5
V.	Parte extra	10
VI.	Ejemplos	12



Capítulo I

Preámbulo

Desarrollado por ID Software y los ultrafamosos John Carmack y John Romero, publicado en 1992 por Apogee Software, Wolfenstein 3D es el primer “First Person Shooter” de la historia del videojuego.



Figura I.1: John Romero (izquierda) y John Carmack (derecha) posando para la posteridad

Wolfenstein 3D es el predecesor de juegos como Doom (Id Software, 1993), Doom II (Id Software, 1994), Duke Nukem 3D (3D Realm, 1996) y Quake (Id Software, 1996), que son otras de las piedras angulares del mundo de los videojuegos.

Y ahora, le toca a Ud. revivir la Historia...

Capítulo II

Objetivos

Los objetivos de este proyecto son parecidos a los de los demás proyectos: Rigor, utilización del C, utilización de algoritmos básicos, búsqueda de información, etc.

Como es un proyecto de diseño gráfico, Cub3D le va a permitir practicar sus dotes de diseñador: ventanas, colores, eventos, formas, etc.

En definitiva, Cub3D es un área de juego extraordinaria para explorar aplicaciones prácticas de las matemáticas sin necesidad que comprender los detalles.

Gracias a la información disponible en Internet, va a utilizar las matemáticas como una herramienta para crear algoritmos elegantes y eficaces.



Le recomendamos que pruebe el juego original antes de empezar:
<http://users.atw.hu/wolf3d/>

Capítulo III

Reglas comunes

- Su proyecto debe estar programado respetando la Norma. Si tiene archivos o funciones extras, entrarán dentro de la verificación de la norma y, como haya algún error de norma, tendrá un 0 en el proyecto.
- Sus funciones no pueden pararse de forma inesperada (segmentation fault, bus error, double free, etc.) salvo en el caso de un comportamiento indefinido. Si esto ocurre, se considerará que su proyecto no es funcional y tendrá un 0 en el proyecto.
- Cualquier memoria reservada en el montón (heap) tendrá que ser liberada cuando sea necesario. No se tolerará ninguna fuga de memoria.
- Si el proyecto lo requiere, tendrá que entregar un Makefile que compilará sus códigos fuente para crear la salida solicitada, utilizando los flags `-Wall`, `-Wextra` y `-Werror`. Su Makefile no debe hacer relink.
- Si el proyecto requiere un Makefile, su Makefile debe incluir al menos las reglas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los extras, debe incluir en su Makefile una regla `bonus` que añadirá los headers, bibliotecas o funciones que no estén permitidos en la parte principal del proyecto. Los extras deben estar dentro de un archivo `_bonus.{c/h}`. Las evaluaciones de la parte obligatoria y de la parte extra se hacen por separado.
- Si el proyecto autoriza su `libft`, debe copiar sus códigos fuente y su Makefile asociado en un directorio `libft`, dentro de la raíz. El Makefile de su proyecto debe compilar la biblioteca con la ayuda de su Makefile y después compilar el proyecto.
- Le recomendamos que cree programas de prueba para su proyecto, aunque ese trabajo **no será ni entregado ni evaluado**. Esto le dará la oportunidad de probar fácilmente su trabajo al igual que el de sus compañeros.
- Deberá entregar su trabajo en el git que se le ha asignado. Solo se evaluará el trabajo que se suba al git. Si Deepthought debe corregir su trabajo, lo hará al final de las evaluaciones por sus pares. Si surge un error durante la evaluación Deepthought, esta última se parará.

Capítulo IV

Parte Obligatoria - Cub3D

Nombre del programa	cub3D
Ficheros de entrega	Todos los archivos
Makefile	all, clean, fclean, re, bonus
Argumentos	Un map con el formato *.cub
Funciones externas autorizadas	<ul style="list-style-type: none">• open, close, read, write, malloc, free, perror, strerror, exit• Todas las funciones de la lib math (-lm man man 3 math)• Todas las funciones de la MinilibX
Libft autorizada	Sí
Descripción	Tiene que crear una representación gráfica en 3D "realista" de lo que podríamos ver dentro de un laberinto con una vista en primera persona. Tiene que crear esta representación utilizando los principios del ray-casting mencionados anteriormente.

Los requisitos son los siguientes:

- Tiene que utilizar la `minilibX`. Puede usar la versión disponible en su OS o la generada a partir del código fuente. Si trabaja con el código fuente, tendrá que seguir las mismas reglas que con la `libft`.
- La gestión de las ventanas debe ser perfecta: gestión de la minimización, pasar de una ventana a otra, etc.

- Debe mostrar texturas distintas (tiene para elegir) en función de si los muros se encuentran en la cara norte, sur, este u oeste.

- Su programa tiene que ser capaz de mostrar un objeto (sprite) en lugar de un muro.
- Su programa tiene que ser capaz de tener colores distintos para el suelo y el techo.
- Por si algún día a **deepthought** le interesa evaluar su proyecto, cuando el segundo argumento sea "**--save**" su programa tiene que poder guardar la primera imagen renderizada en formato **bmp**.
- Si no hay segundo argumento, el programa tendrá que mostrar la imagen en una ventana respetando las siguientes reglas:
 - Las teclas de dirección del teclado de la izquierda y de la derecha deben permitir que se pueda rotar la cámara (mirar a la izquierda y a la derecha)
 - Las teclas W, A, S y D deben permitir que se desplace la cámara (desplazamiento del personaje)
 - Al pulsar la tecla **ESC** se debe cerrar la ventana y salir del programa de forma limpia.
 - Al hacer clic sobre el aspa roja de la ventana se debe cerrar la ventana y salir del programa de forma limpia.
 - Si el tamaño de la ventana pedida en el map es más grande que el de la pantalla, se tendrá que limitar el tamaño de la ventana al de la pantalla.
 - Se recomienda encarecidamente el uso de **imágenes** de la **minilibX**.
- Su programa debe recibir como primer argumento un archivo de descripción de la escena con la extensión **.cub**.
 - El map debe estar compuesto únicamente por estos 4 caracteres: **0** para los espacios vacíos, **1** para los muros, **2** para un objeto, **N**, **S**, **E** o **W** que representan la posición inicial del jugador y su orientación.

Este map sencillo debe ser válido:

```
111111  
100101  
102001  
1100N1  
111111
```

- El map debe estar encerrado/rodeado por muros, si no el programa tendrá que devolver un error.
- Aparte de la descripción del map, se puede separar cada tipo de elemento con una o varias líneas vacías.
- La descripción del mapa debe ir siempre al final del archivo; el resto de elementos pueden ir colocados en cualquier orden.
- La información relativa a cada elemento puede ir separada por uno o varios espacios.

- Excepto del mapa, la información de cada elemento puede estar separado por uno o más espacio(s).
- El mapa debe ser parseado como aparece en el fichero. Espacios son elementos validos del mapa, que significan que no hay nada. Depende de usted gestionarlo. Tiene que poder parsear todo tipo de mapa, de momento que el mapa respete las reglas.
- En cada elemento, el primer carácter será su identificador (uno o dos caracteres), seguido de la información específica del elemento respetando un orden preciso como:
 - Resolución:

```
R 1920 1080
```

 - ◊ identificador: **R**
 - ◊ tamaño de renderizado eje x
 - ◊ tamaño de renderizado eje y
 - textura norte:

```
NO ./path_to_the_north_texture
```

 - ◊ identificador: **NO**
 - ◊ camino hacia la textura norte
 - textura sur:

```
SO ./path_to_the_south_texture
```

 - ◊ identificador: **SO**
 - ◊ camino hacia la textura sur
 - textura oeste:

```
WE ./path_to_the_west_texture
```

 - ◊ identificador: **WE**
 - ◊ camino hacia la textura oeste
 - textura este:

```
EA ./path_to_the_east_texture
```

 - ◊ identificador: **EA**
 - ◊ camino hacia la textura este
 - textura del sprite:

```
S ./path_to_the_sprite_texture
```

 - ◊ identificador: **S**
 - ◊ camino hacia la textura sprite
 - Color del suelo:

```
F 220,100,0
```

 - ◊ identificador: **F**
 - ◊ colores R,G,B rango [0,255]: **0, 255, 255**

- Color del techo:

```
C 225,30,0
```

- ◊ identificador: **C**
- ◊ colores R,G,B rango [0,255]: **0, 255, 255**

- Ejemplo minimalista de escena de la parte obligatoria .cub:

```
R 1920 1080
NO ./path_to_the_north_texture
SO ./path_to_the_south_texture
WE ./path_to_the_west_texture
EA ./path_to_the_east_texture

S ./path_to_the_sprite_texture
F 220,100,0
C 225,30,0

1111111111111111111111111111
1000000000110000000000001
1011000001110000002000001
100100000000000000000000001
11111111011000001110000000000001
1000000000110000011101111111111
1111011111111011100000010001
1111011111111011101010010001
11000000110101011100000010001
10002000000000001100000010001
100000000000000001101010010001
1100000111010101111011110N0111
11110111 1110101 101111010001
11111111 1111111 111111111111
```

- Si surge cualquier problema de configuración en el archivo, el programa tendrá que devolver “Error\n” seguido de un mensaje explícito que Ud. elegirá.

Capítulo V

Parte extra



Solo se evaluarán los ejercicios extra si la parte obligatoria está perfecta. Con perfecta, nos referimos a que esté todo completo y que en ningún caso se cuelgue, incluso cuando se cometan errores graves como un uso incorrecto u otras cosas. Si no dispone de todos los puntos de la parte obligatoria, se ignorará la parte extra.

Lista de ejercicios extra:

- Colisión contra los muros
- Una skybox
- Textura sobre el suelo o el techo.
- Un HUD.
- Posibilidad de mirar hacia arriba y hacia abajo.
- Saltar o agacharse.
- Un efecto de sombra basado en la distancia.
- Una barra de vida.
- Más objetos en el laberinto.
- Colisión con los objetos.
- Ganancia/pérdida de puntos de vida al toparse con objetos/trampas.
- Puertas que pueden estar abiertas/cerradas.
- Puertas secretas.
- Animaciones (disparos o sprites animados).
- Varios niveles.
- Sonido y música.
- ¡Armas y malos con los que pelear!



Para obtener todos los puntos de bonificación, debe validar al menos 14 ejercicios, así que elija con cuidado pero sin perder demasiado tiempo.



En esta parte extra puede utilizar otras funciones, siempre y cuando pueda justificar su uso en la evaluación. También puede modificar el archivo de la escena para que se adapte a sus necesidades. ¡Sea listo!

Capítulo VI

Ejemplos



Figura VI.1: Wolfeinstein3D Juego original utilizando el ray-casting.

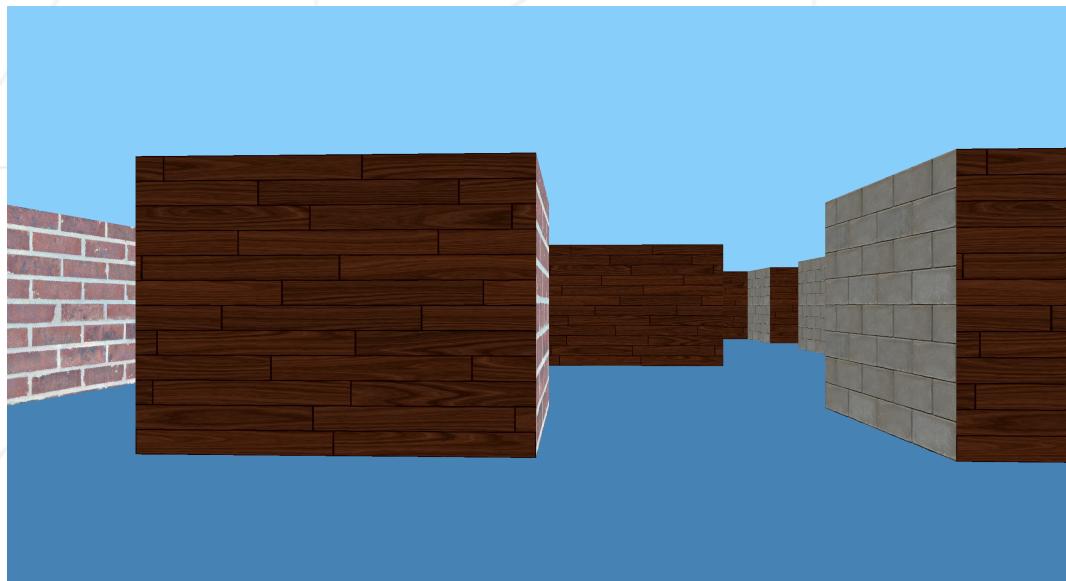


Figura VI.2: Ejemplo de lo que se espera de Ud. en la parte obligatoria.



Figura VI.3: Ejemplo de tarea extra con un minimap, texturas de suelo y techo y un famoso erizo animado.



Figura VI.4: Otro ejemplo de tarea extra con un HUD, una barra de vida, un efecto de sombra y un arma que puede disparar

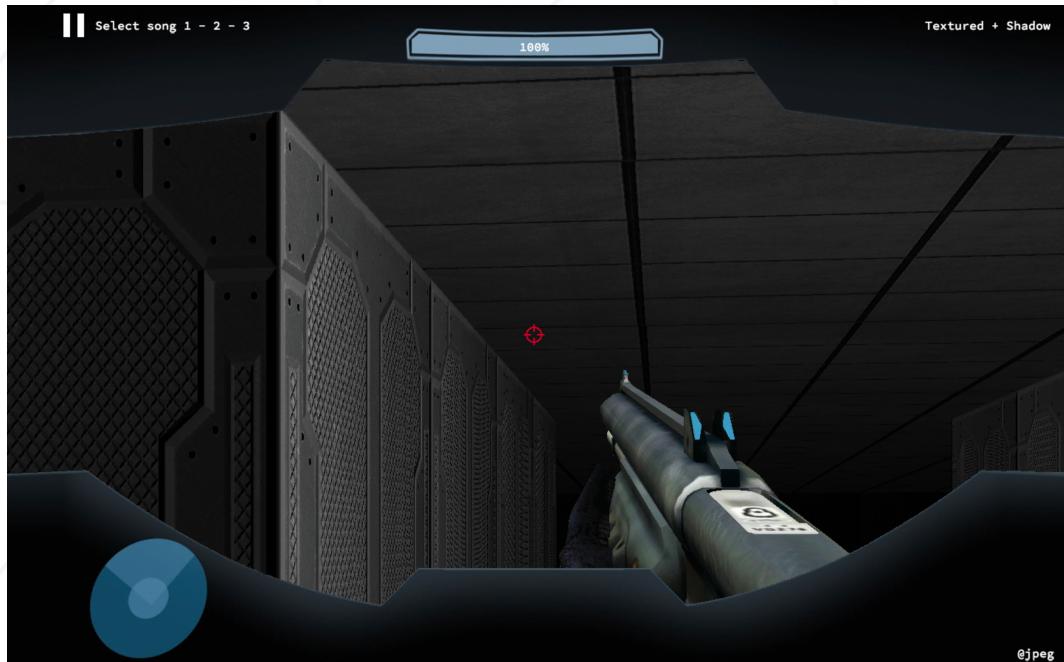


Figura VI.5: Otro ejemplo de la parte extra con un arma de su elección y el jugador que mira hacia el techo