



## cub3D

Mi primer RayCaster con miniLibX

*Resumen: Este proyecto está inspirado en el reconocido mundialmente Wolfenstein 3D, el primer FPS jamás creado. Te permitirá explorar el ray-casting. Tu misión es crear una vista dinámica dentro de un laberinto, en el que tendrás que abrirte paso.*



# Índice general

I.	Avance	2
II.	Objetivos	3
III.	Instrucciones generales	4
IV.	Parte obligatoria - cub3D	5
V.	Parte bonus	10
VI.	Ejemplos	11



# Capítulo I

## Avance

Desarrollado por **Id Software**, por los reconocidos John Carmack y John Romero, publicado en 1992 por **Apogee Software**, **Wolfenstein 3D** es el primer “First Person Shooter” real en la historia de los videojuegos.



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

**Wolfenstein 3D** es el antecesor de juegos como **Doom** (**Id Software**, 1993), **Doom II** (**Id Software**, 1994), **Duke Nukem 3D** (**3D Realm**, 1996) y **Quake** (**Id Software**, 1996), considerados hitos eternos en el mundo de los videojuegos.

Ahora, es tu momento de hacer Historia...

# Capítulo II

## Objetivos

El objetivo de este proyecto es similar a todos los del primer año: Rigot, uso de C, uso de algoritmos básicos, búsqueda de información, etc.

Como un proyecto de diseño gráfico, cub3D te permitirá mejorar tus habilidades en las siguientes áreas: ventanas, colores, eventos, llenar figuras, etc.

Para terminar cub3D es un ejercicio destacable para explorar las aplicaciones prácticas de las matemáticas sin tener que entrar en detalles.

Con la ayuda de los numerosos recursos disponibles en internet, usarás las matemáticas como una herramienta para crear elegantes y eficientes algoritmos.



Te recomendamos probar la experiencia original antes de empezar con el proyecto:

<http://users.atw.hu/wolf3d/>

# Capítulo III

## Instrucciones generales

- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un 0 si hay algún error de norma dentro.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni tener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un 0 durante la evaluación.
- Toda la memoria alocada en heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el subject lo requiere, deberás entregar un **Makefile** que compilará tus archivos fuente al output requerido con las flags **-Wall**, **-Werror** y **-Wextra**, por supuesto tu **Makefile** no debe hacer relink.
- Tu **Makefile** debe contener al menos las normas **\$(NAME)**, **all**, **clean**, **fclean** y **re**.
- Para entregar los bonus de tu proyecto, deberás incluir una regla **bonus** en tu **Makefile**, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos **\_bonus.{c/h}**. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la **libft**, deberás copiar su fuente y sus **Makefile** asociados en un directorio **libft** con su correspondiente **Makefile**. El **Makefile** de tu proyecto debe compilar primero la librería utilizando su **Makefile**, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo a tu repositorio **Git** asignado. Solo el trabajo de tu repositorio **Git** será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus compañeros. Si se encuentra un error durante la evaluación de Deepthought, la evaluación terminará.

# Capítulo IV

## Parte obligatoria - cub3D

Nombre de programa	cub3D
Archivos a entregar	Todos tus archivos
Makefile	all, clean, fclean, re, bonus
Argumentos	un mapa en formato *.cub
Funciones autorizadas	<ul style="list-style-type: none"><li>• open, close, read, write, printf, malloc, free, perror, strerror, exit</li><li>• Todas las funciones de la librería de matemáticas (-lm man man 3 math)</li><li>• Todas las funciones de la miniLibX</li></ul>
Se permite usar libft	Sí
Descripción	Debes crear una representación gráfica 3D "realista" del interior de un laberinto desde una perspectiva de primera persona. Deberás crear esta representación utilizando los principios del Ray-Casting como se menciona previamente.

Los principios son los siguientes:

- Debes utilizar la `miniLibX`. Ya sea en la versión disponible en el sistema operativo, o en su fuente. Si decides trabajar con la fuente, deberás seguir las mismas reglas que con tu `libft`, escritas en la parte de **Instrucciones Generales**.
- La gestión de tu ventana debe permanecer impecable: cambiar de ventana, minimizar, etc.

- Muestra diferentes texturas de muros (la elección es tuya) que varíen dependiendo del lado al que mire la pared (norte, sur, este u oeste).

- Tu programa debe ser capaz de poner los colores del techo y del suelo a dos diferentes.
- El programa debe mostrar la imagen en una ventana respetando las siguientes normas:
  - Las flechas izquierda y derecha del teclado deben permitirte mirar hacia la izquierda o hacia la derecha en el laberinto.
  - Las teclas W, A, S y D deben permitirte mover el punto de vista a través del laberinto.
  - Pulsar ESC debe cerrar la ventana y cerrar el programa limpiamente.
  - Hacer clic en la cruz roja de la ventana debe cerrarla y terminar el programa limpiamente.
  - El uso de imágenes de la `miniLibX` se recomienda encarecidamente.

- Tu programa debe aceptar como primer argumento un archivo con la extensión `.cub` que contenga la descripción de la escena.

- El mapa se compondrá exclusivamente de estos 6 caracteres: **0** para un espacio vacío, **1** para un muro, y **N**, **S**, **E** y **W** para la posición inicial del jugador y su orientación inicial.

Este es un mapa válido sencillo:

```
111111  
100101  
101001  
1100N1  
111111
```

- El mapa debe estar cerrado/rodeado de muros, si no el programa debe devolver un error.
- Excepto por el contenido del mapa, cada tipo de elemento puede estar separado por una o más líneas vacías.
- Excepto por el contenido del mapa, que debe estar siempre al final, cada tipo de elemento puede estar establecido en cualquier orden en el archivo.
- Excepto por el mapa, cada tipo de dato tras un elemento puede estar separado por uno o más espacios.
- El mapa debe procesarse tal y como aparece en el archivo. Los espacios forman una parte válida del mapa y es cuestión tuya cómo utilizarlos. Debes poder procesar cualquier tipo de mapa, siempre y cuando respete las reglas de mapas.

- El primer dato de cada elemento (excepto el mapa) es el identificador (compuesto por uno o dos caracteres), seguido de toda la información específica para cada uno de ellos en un orden estricto, es decir:
  - Textura norte:

```
NO ./ruta_a_la_textura_norte
```

    - ◊ identificador: **NO**
    - ◊ ruta a la textura norte
  - Textura sur:

```
NO ./ruta_a_la_textura_sur
```

    - ◊ identificador: **SO**
    - ◊ ruta a la textura sur
  - Textura este:

```
NO ./ruta_a_la_textura_este
```

    - ◊ identificador: **EA**
    - ◊ ruta a la textura este
  - Textura oeste:

```
NO ./ruta_a_la_textura_oeste
```

    - ◊ identificador: **WE**
    - ◊ ruta a la textura oeste
  - Color de suelo:

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

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

- Color de techo:

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

- ◊ identificador: C

- ◊ Colores R,G,B en rango [0,255]: **0, 255, 255**

- Ejemplo de la parte obligatoria con un mapa .cub minimalista:

```
NO ./ruta_a_la_textura_norte
SO ./ruta_a_la_textura_sur
WE ./ruta_a_la_textura_oeste
EA ./ruta_a_la_textura_este

F 220,100,0
C 225,30,0

      111111111111111111111111111111
      10000000001100000000000001
      10110000011100000000000001
      100100000000000000000000001
111111111011000001110000000000001
10000000001100000111011111111111
1111011111111011100000010001
11110111111110111010010001
11000000110101011100000010001
100000000000000001100000010001
100000000000000001101010010001
11000001110101011111011110N0111
11110111 1110101 101111010001
11111111 1111111 111111111111
```

- En caso de que algún tipo de error de configuración en el archivo, el programa debe salir correctamente y devolver ".Error\n" seguido de un mensaje explícito de tu elección.

# Capítulo V

## Parte bonus



Los bonus solo serán evaluados si y solo si la parte obligatoria está PERFECTA. Por PERFECTA queremos naturalmente decir que esté completa, que no falle incluso en casos de errores tontos o mal uso. Por lo tanto, en caso de que tu parte obligatoria no obtenga TODOS los puntos durante la evaluación, los bonus serán completamente IGNORADOS.

Lista de bonus:

- Colisiones de muros.
- Un sistema de minimapa.
- Puertas que se puedan abrir y cerrar.
- Sprites animados.
- Rotar el punto de vista con el ratón.



Tendrás luego la oportunidad de crear juegos mejores, no pierdas mucho tiempo



Tienes permitido utilizar otras funciones para completar la parte bonus siempre y cuando su uso esté justificado durante tu evaluación. Tienes permitido modificar el formato del archivo esperado para adecuarse a tus necesidades, así como añadir símbolos al mapa para completar la parte extra. Sé inteligente.

# Capítulo VI

## Ejemplos



Figura VI.1: Juego Wolfenstein3D con RayCasting original.

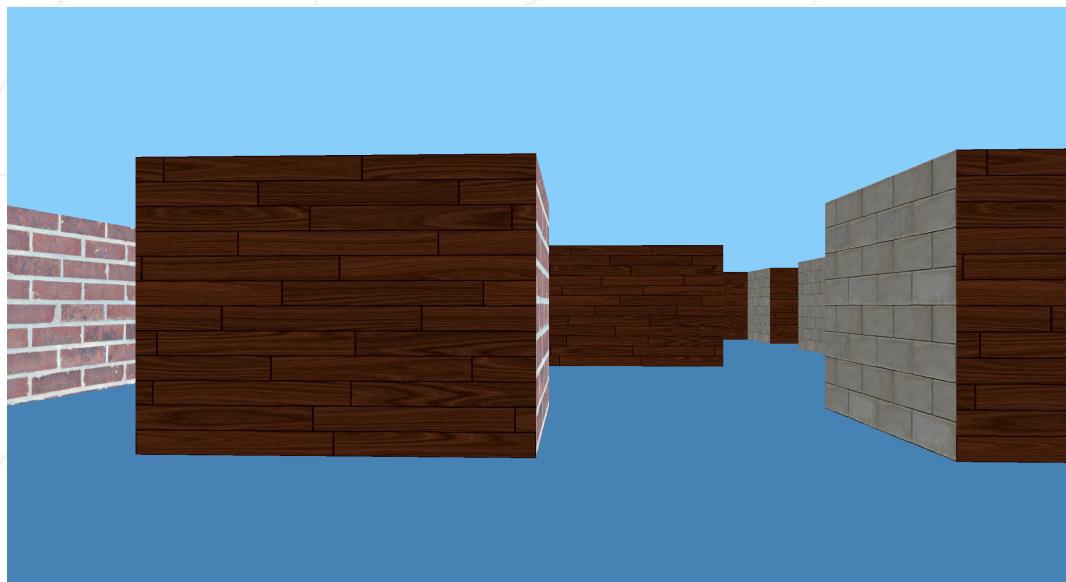


Figura VI.2: Ejemplo de cómo se puede ver tu juego con la parte obligatoria.



Figura VI.3: Ejemplo de la parte bonus con un minimapa, texturas de suelo y techo y el sprite de un famoso erizo animado.