



# cub3D

My first RayCaster with miniLibX

*Summary: This project is inspired by the world-famous eponymous 90's game, which was the first FPS ever. It will enable you to explore ray-casting. Your goal will be to make a dynamic view inside a maze, in which you'll have to find your way.*



# Contents

I	Foreword	2
II	Goals	3
III	Common Instructions	4
IV	Mandatory part - cub3D	5
V	Bonus part	9
VI	Examples	11



# Chapter I

## Foreword

Developed by **Id Software** by the über famous John Carmack and John Romero, published in 1992 by **Apogee Software**, **Wolfenstein 3D** is the first true “First Person Shooter” in the history of video games.



Figure I.1: John Romero (left) and John Carmack (right) posing for posterity.

**Wolfenstein 3D** is the ancestor of games like **Doom** (**Id Software**, 1993), **Doom II** (**Id Software**, 1994), **Duke Nukem 3D** (**3D Realm**, 1996) and **Quake** (**Id Software**, 1996), that are additional eternal milestones in the world of video games.

Now, it's your turn to relive History...

# Chapter II

## Goals

This project's objectives are similar to all this first year's objectives: Rigor, use of C, use of basic algorithms, information research etc.

As a graphic design project, **cub3D** will enable you to improve your skills in these areas: windows, colors, events, fill shapes, etc.

To conclude **cub3D** is a remarkable playground to explore the playful practical applications of mathematics without having to understand the specifics.

With the help of the numerous documents available on the internet, you will use mathematics as a tool to create elegant and efficient algorithms.



We recommend that you test the original game before starting this project:

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

# Chapter III

## Common Instructions

- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a 0 if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, and your `Makefile` must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your `Makefile`, which will add all the various headers, libraries or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated `Makefile`. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

# Chapter IV

## Mandatory part - cub3D

<b>Program name</b>	cub3D
<b>Turn in files</b>	All your files
<b>Makefile</b>	all, clean, fclean, re, bonus
<b>Arguments</b>	a map in format *.cub
<b>External functs.</b>	<ul style="list-style-type: none"><li>• open, close, read, write, printf, malloc, free, perror, strerror, exit</li><li>• All functions of the math library (-lm man man 3 math)</li><li>• All functions of the MinilibX</li></ul>
<b>Libft authorized</b>	Yes
<b>Description</b>	You must create a "realistic" 3D graphical representation of the inside of a maze from a first person perspective. You have to create this representation using the Ray-Casting principles mentioned earlier.

The constraints are as follows:

- You **must** use the `miniLibX`. Either the version that is available on the operating system, or from its sources. If you choose to work with the sources, you will need to apply the same rules for your `libft` as those written above in **Common Instructions** part.
- The management of your window must remain smooth: changing to another window, minimizing, etc.
- Display different wall textures (the choice is yours) that vary depending on which side the wall is facing (North, South, East, West).

- Your program must be able to display an item (sprite) instead of a wall.
- Your program must be able to set the floor and ceiling colors to two different ones.
- In case the Deepthought has eyes one day to evaluate your project, your program must save the first rendered image in `bmp` format when its second argument is `--save`.
- If no second argument is supplied, the program displays the image in a window and respects the following rules:
  - The left and right arrow keys of the keyboard must allow you to look left and right in the maze.
  - The W, A, S and D keys must allow you to move the point of view through the maze.
  - Pressing `ESC` must close the window and quit the program cleanly.
  - Clicking on the red cross on the window's frame must close the window and quit the program cleanly.
  - If the declared screen size in the map is greater than the display resolution, the window size will be set depending to the current display resolution.
  - The use of `images` of the `minilibX` is strongly recommended.
- Your program must take as a first argument a scene description file with the `.cub` extension.
  - The map must be composed of only 4 possible characters: **0** for an empty space, **1** for a wall, **2** for an item and **N,S,E** or **W** for the player's start position and spawning orientation.  
This is a simple valid map:

```
111111
100101
102001
1100N1
111111
```
  - The map must be closed/surrounded by walls, if not the program must return an error.
  - Except for the map content, each type of element can be separated by one or more empty line(s).
  - Except for the map content which always has to be the last, each type of element can be set in any order in the file.
  - Except for the map, each type of information from an element can be separated by one or more space(s).
  - The map must be parsed as it looks like in the file. Spaces are a valid part of the map, and is up to you to handle. You must be able to parse any kind of map, as long as it respects the maps rules.

- o Each element (except the map) firsts information is the type identifier (composed by one or two character(s)), followed by all specific informations for each object in a strict order such as :

- \* Resolution:

```
R 1920 1080
```

- identifier: **R**
- x render size
- y render size

- \* North texture:

```
NO ./path_to_the_north_texture
```

- identifier: **NO**
- path to the north texture

- \* South texture:

```
SO ./path_to_the_south_texture
```

- identifier: **SO**
- path to the south texture

- \* West texture:

```
WE ./path_to_the_west_texture
```

- identifier: **WE**
- path to the west texture

- \* East texture:

```
EA ./path_to_the_east_texture
```

- identifier: **EA**
- path to the east texture

- \* Sprite texture:

```
S ./path_to_the_sprite_texture
```

- identifier: **S**
- path to the sprite texture

- \* Floor color:

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

- identifier: **F**
- R,G,B colors in range [0,255]: **0, 255, 255**

- \* Ceiling color:

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

- identifier: **C**
  - R,G,B colors in range [0,255]: **0, 255, 255**

- Example of the mandatory part with a minimalist .cub scene:

```
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
10000000000110000000000001
1011000001110000002000001
10010000000000000000000000001
11111111011000001110000000000001
10000000001100000111011111111111
1111011111111011100000010001
1111011111111011101010010001
11000000110101011100000010001
10002000000000001100000010001
10000000000000001101010010001
110000011101010111101110N0111
11110111 1110101 101111010001
11111111 1111111 111111111111
```

- If any misconfiguration of any kind is encountered in the file, the program must exit properly and return "Error\n" followed by an explicit error message of your choice.

# Chapter V

## Bonus part



Bonuses will be evaluated only if your mandatory part is **PERFECT**. By **PERFECT** we naturally mean that it needs to be complete, that it cannot fail, even in cases of nasty mistakes like wrong uses etc. Basically, it means that if your mandatory part does not obtain **ALL** the points during the grading, your bonuses will be entirely **IGNORED**.

Bonus list:

- Wall collisions.
- A skybox.
- Floor and/or ceiling texture.
- An HUD.
- Ability to look up and down.
- Jump or crouch.
- A distance related shadow effect.
- Life bar.
- More items in the maze.
- Object collisions.
- Earning points and/or losing life by picking up objects/traps.
- Doors which can open and close.
- Secret doors.
- Animations of a gun shot or animated sprite.
- Several levels.
- Sounds and music.
- Rotate the point of view with the mouse.

- Weapons and bad guys to fight!



To earn all bonus points you need to validate at least 14 of them, so choose wisely but be careful to not waste your time!



You are allowed to use other functions to complete the bonus part as long as their use is justified during your evaluation. You are also allowed to modify the expected scene file format to fit your needs.  
Be smart!

# Chapter VI

## Examples



Figure VI.1: Wolfenstein3D original game usinfg RayCasting.

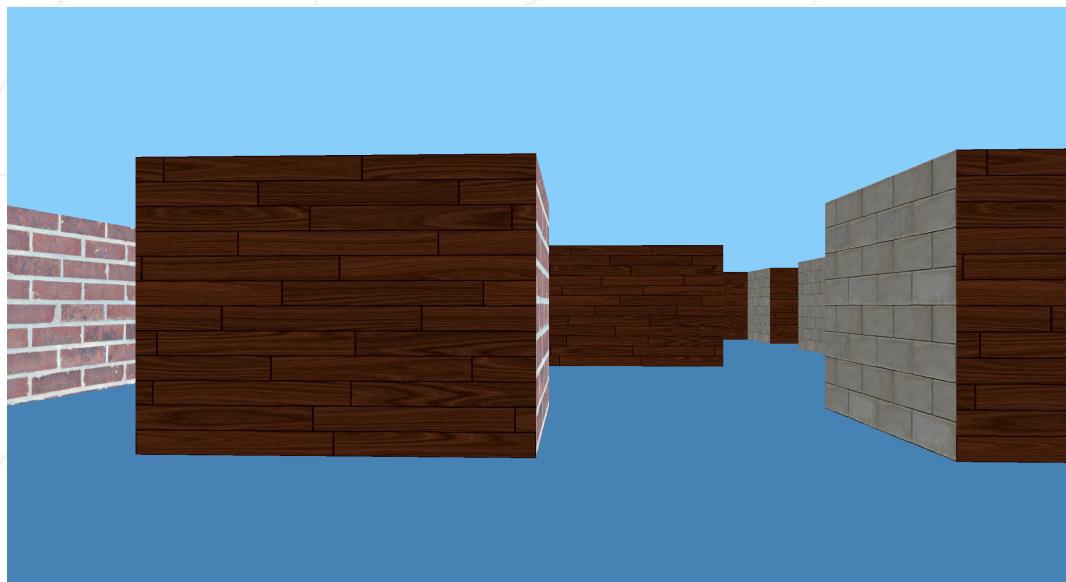


Figure VI.2: Example of what your project could look like as per the mandatory part.



Figure VI.3: Example of the bonus part with a minimap, floor and ceiling textures and an animated famous hedgehog sprite.



Figure VI.4: Another example of the bonus part with an HUD, a life bar, sounds and music, a shadow effect and a gun which can shoot.



Figure VI.5: Another example of the bonus part with another weapon choice and with the player looking at the ceiling.