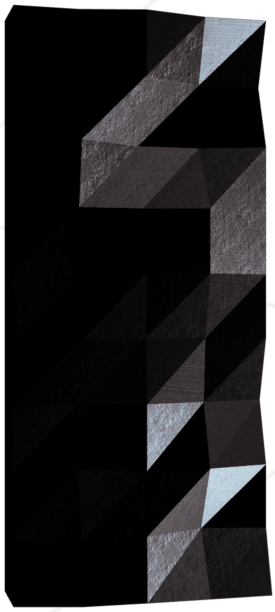

Wolf3D

Call Apogee Say Aardwolf

Pedago pedago@42.fr

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

Chapter I

Foreword

Figure I.1: `Wolfenstein 3D`'s title screen by `Id Software`

`Wolfenstein 3D` is a seminal game in the pantheon of early video games. 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.

Embodying the American spy Willian "B.J." Blazkowicz, the player attacks and decimates the Nazi army (and another army of zombies while he is at it) with the support of destructive weapons one more lethal than the next. At the peak of the Story, William finally faces off an Adolf Hitler, equipped with no less than four Gatling guns and a robotic protective armor... A true wonder of storytelling, violence and historical details.

`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...

Figure I.2: John Romero (left) et John Carmack (right) posing for posterity.

Chapter II

Introduction

Don't let `Wolfenstein 3D`'s basic graphics and violent gameplay fool you! It is in fact a highly complex and advanced game. There is a reason why John Carmack is considered one of the world's best coders.

In this project you must follow in the footsteps of this brillant programmer and write your own version of `Wolfenstein 3D`... Hold on, we won't ask you to re-write not the ENTIRE `Wolfenstein 3D`. One has to be be realistic here. Your mission will be to write, using the ray casting technique, a 3D representation of a maze in which a player can find his/her way.

Once you have achieved this goal, textures, weapons, enemies, secrets, sounds, music... The limit is your imagination.

We recommand you to test the original game before starting:
http://3d.wolfenstein.com

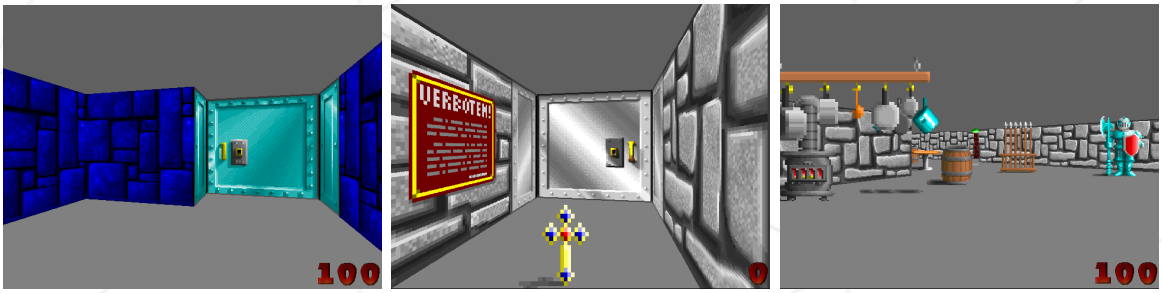

Figure II.1: raycasting used for the graphic representation of the project `Wolf3D`.

Chapter III

Objectives

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

As a graphic design project, `Wolf3D` will enable you to solidify your skills in this area: windows, colors, events, fill shapes etc.

To conclude `Wolf3D` is a remarkable playground to explore the playful practical applications of mathematics without having to understand the specifics. John Carmack is undoubtedly a genius. He was one of the first programmers to think about these applications and remains famous for them today. With the help of the numerous documents available on the internet, you will use mathematics as a tool to create elegant and efficient algorithms.

Imagine yourself in 1991. You are working for a SME in Mesquite TX; you love to watch gory movies and listen to Heavy Metal music; GPU and hardware acceleration do not yet exist; and you will become a Video-Game Rock Star in less than 2 years. In two words, HAVE FUN!

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The executable file must be named `wolf3d`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- If you are clever, you will use your library for your `wolf3d`. Submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- You cannot use global variables.
- Your project must be written in accordance with the Norm. Only norminette is authoritative.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You'll have to submit a file called `author` containing your username followed by a '\n' at the root of your repository.

```
$>cat -e author
xlogin$
```

- You can use `MacOS` native `MinilibX` library already installed on the imacs, or you can use MinilibX from its sources that you'll need to integrate similarly to libft. Last option, you can use additional graphic libraries (`X11`, `SDL`, etc...). If the library you are using is not installed on the imacs, you will have to submit the sources of this library in your repository, and it will have to be automatically compiled, without doing anything more than compiling your project, exactly like `MinilibX` or like your `libft`. No matter which graphic library you can only use its basic drawings functions similar to `MinilibX`: Open a window, lit a pixel and manage events.

- Within the mandatory part, you are allowed to use only the following libc functions:
    - `open`
    - `read`
    - `write`
    - `close`
    - `malloc`
    - `free`
    - `perror`
    - `strerror`
    - `exit`
    - All functions of the math library (`-lm` man `man 3 math`)
    - All functions of the `MinilibX` or their equivalent in another graphic library.
- You are allowed to use other functions or other librairies to complete the bonus part as long as their use is justified during your defense. Be smart!
- You can ask your questions on the forum, on slack...

Chapter V

Mandatory part

The fun part of this project is the bonuses. But before you can start hunting Nazis, you need to ace totally and completely the mandatory part.

You must create a 3D graphically "realistic" representation that we could have from inside a maze in a subjective view. You need to create this representation using the Ray-Casting principles mentioned earlier.

- You can choose the size and the shape of your labyrinth, but it has to be a file outside of your sources. Your evaluator should have the possibility to modify it without recompiling the whole project.
- The management of your window must remain smooth: passing over of another window, minimization, etc.
- 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.
- The arrows on the keyboard must allow you to move in real time in the maze, like in the original game.
- Display different wall textures (the choice is yours) that vary depending on which compass point the wall is facing (North, South, East, West).

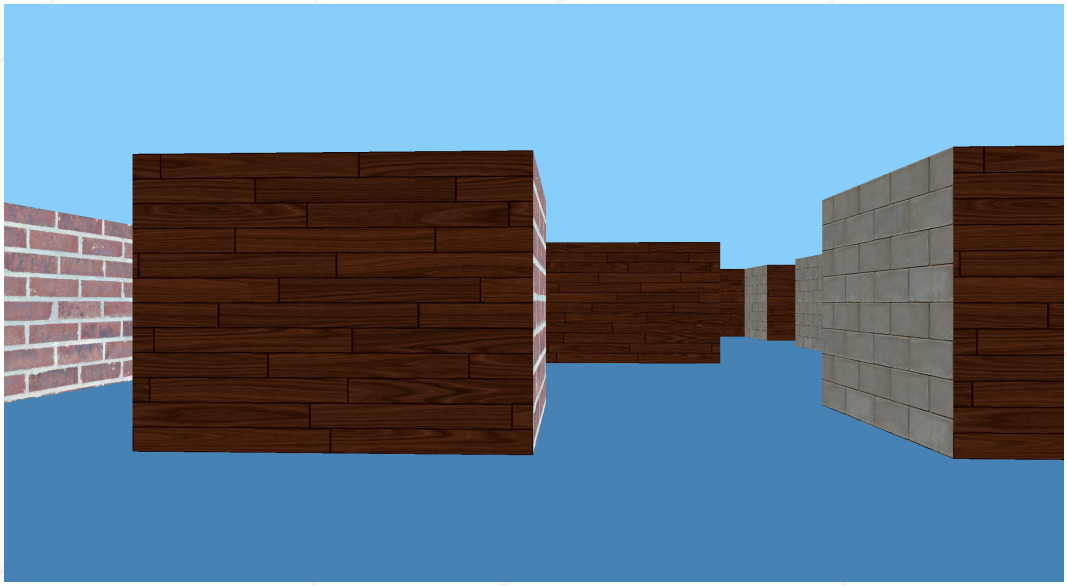

Figure V.1: Example of what your project could look like as per the mandatory part.

Chapter VI

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.

- Wall collisions.
- A skybox.
- Floor and/or ceiling texture.
- Items in the maze.
- Object collisions.
- Earning points by picking up objects.
- Doors can open and close.
- Bad guys to fight.
- Secret doors.
- Animations.
- Several levels.
- Sounds and music.
- A `Doom` Engine (that would be so sexy!).
- A `Duke Nukem 3D` engine (that would be super duper sexy!).

Chapter VII

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all and don't forget your author file!