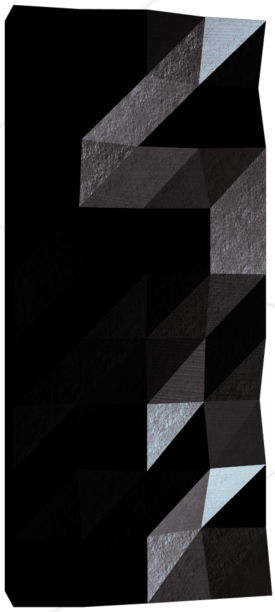

DooM-Nukem' 3D

Ultra cub3D

Summary: Congratulations, you completed the wolf/cub3D project! But beware, you are now in a more complex universe made of pieces of various shapes, scattered with decorations and furniture, filled with enemies.

Version: 3.1

Contents

Chapter I

Foreword

Following `Wolfenstein 3D`, John Carmack and John Romero continued their common adventure. Once realizing some additional levels for `Wolf3D` special editions, they started wotking on the sequel. Bigger, better, stronger. Towards the end of 1993, they released `Doom`, another first-person shoot'em up. The immersion is even more intense, the atmosphere is even more engaging, and Doom immediately became, and remained, a reference in video games.

In the UAC's (Union Aerospace Corporation) premises on Mars, you play an ex-Marine who has to clean the base of a whole bunch of monsters coming straight from hell. These appeared because of radioactive experiments that created a breach in space-time. Each monster uglier than the one before, up to the huge final boss, you're going to need all the weapons available to defeat them. Ultra violent gameplay, tempo set by the Heavy Metal soundtrack, even today you can't call yourself a gamer if you've never played it.

As the the ID Software adventures were unfolding, another universe was built in another video game company: Apogee Software. As early as 1991, in Los Angeles invaded by aliens, `Duke Nukem` is fighting to take them down. At the time, it's only a Mario-style 2D platform. After a swift incursion in space for the second scenario in 1993, the 1996 third version is a 3D, first person game. With more visual effects than `Doom`, a cartoon-like design and lots of sex, violence, cult dialogues and off-beat humor, this third opus is a dazzling success. Released under the Apogee Software's 3D Realms brand, `Duke Nukem 3D` becomes Doom's rightful heir in the realm of pseudo 3D games, before the emergence of GPUs and hardware acceleration to create full 3D.

Once again, it's time for you to relive History...

Chapter II

Introduction

We will ask you to create a mix of `Doom` and `Duke Nukem 3D`. Many basic functional elements are identical. However, they were not implemented in the same way by the designers of the two games. It's up to you to find the one that you prefer and that will make it possible to implement both the requested features and the ones you can't wait to invent.

You'll still be using the raycasting technique, but you will need a lot more work than for your `Wolf3D`. The bonus part then is now a core element of the project.

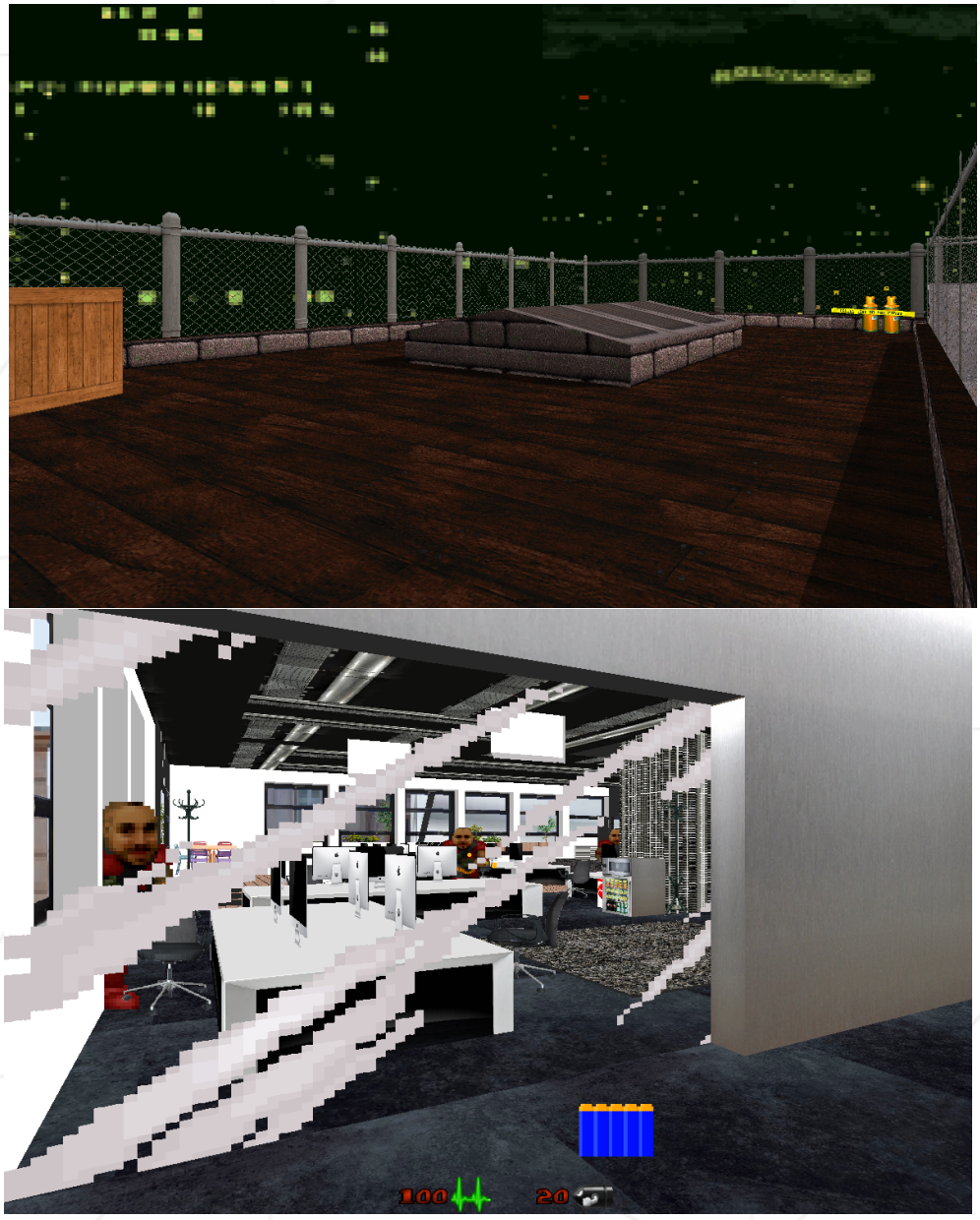

Figure II.1: Use of raycasting for the graphic rendering of the `Doom-Nukem` project.

Chapter III

Objectives

The goals of this project include the same goals as all the other projects: rigor, `C`, commonly used algorithms, information search, literature analysis, etc.

But as a graphic programming project, `Doom-Nukem` will also allow you to strengthen your knowledge in this particular field: windows, images, events, filling in forms, etc. But you should be familiar with all this by now...

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organize and name your files as you see fit, but you must follow the following rules.
- The executable file must be named `doom-nukem`.
- 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 `libft` for your `Doom-Nukem`. Add your `libft` folder, 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.
- You have to handle errors carefully. In no way can your program quit unexpectedly (segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You can use any library you want, however you remain limited to use them for:
  - Basic drawing functions
  - Window management
  - Light a pixel
  - Manage events
  - Sound and music handling.
- In this project, you are allowed to use all the libC (man 2) syscalls, as well as `malloc`, `free`, `perror`, `strerror`, `exit`, all the math lib functions (`-lm`), and all the `MinilibX` functions - or their equivalent in another graphic library. You'll have to recode a png or tga (or anything else you need) reader.
- You are allowed to use other functions or even other librairies to complete the bonus part as long as you can justify their use during your evaluation. Be smart!

Chapter V

Mandatory part

All the fun of this project lies in its gameplay features. But before you get to count your points as you defeat the monsters and aliens - and all the XP that gets you, you still have to work a bit. Luckily some of the work has already been done and is your `wolf3d/cub3d` (please bear in mind that even if it's mandatory, this part will not earn you any point).

You must create the "realistic" 3D graphic representation that you would have from a subjective view within your universe. You must use the Ray-Casting method for this representation. No hardware acceleration and no 3D libraries are allowed.

`Wolf3d` features that you must include:

- You can move through the universe in real-time via the keyboard arrows, just like in the two original games. 360-degree rotation, forward and back move.
- Pressing the `ESC` key must close the window and exit the program properly.
- Clicking the red cross on the window border must close the window and exit the program properly.
- There are textures on the walls.

New graphic features:

- You can look up and down.
- The areas you go through don't have a fixed shape, for example the rooms can have any number of walls in all possible directions.
- The floor and ceiling have adjustable heights, which means different areas will not be at the same height - and you'll have to manage that smoothly.
- The floor and ceiling may not be horizontal but inclined planes.
- The floors and ceilings have textures.
- There can be a sky instead of a ceiling.
- Walls can be partially transparent, unveiling the room behind.

- Decorative elements can be placed on the walls over the main texture (paintings, posters...).
- Objects or characters can be placed in the world as sprites that always face the player.
- Objects or characters can be placed in the world as multiple sprites displayed according to the orientation between the object and the player (you'll see the object or character from behind and from the side as you walk around it).
- Each room or area has a light control tool, affecting both the walls and the objects within.
- Text messages can be displayed during the game over the rest of the graphics.
- There must be a HUD (life, currency, ammunition, carried artifacts...) By that we mean several elements arranged on the player's view and not a simple one-block banner.

About the gameplay:

- The view follows the mouse smoothly, making it possible to spin around (a whole 360) and to look up and down.
- Strafing is possible.
- Interactions with walls and steps of a reasonable height should be managed properly.
- The player can run, jump, fall, bend down, and get up.
- The player can fly or swim (same difference...).
- The objects present may or may not block the passage, in proportion to their visual representation.
- The items around can be picked up - or not, and supply an inventory.
- The player can interact with the elements of the rooms (walls, objects...), whether by simple proximity (walking on a dangerous area) or because of a voluntary action (pressing a button).
- Actions or series of actions can result from interactions, from objects being picked up, or even in an independent way. They can be timed.
- Actions can alter all the elements of the game, all the shapes, all the properties (the shape of a room, the height of a ceiling, the texture of a wall or an object, the location of the specific texture on a door that indicates its "handle", etc).
- The 2 previous points mean that your game must contain animations, various devices like door opening systems, keys, and ways to use them, elevators, and secret passages.
- Specific characters and/or objects can have their actions, reactions and interactions in the universe.

- To mimic the `Doom` and `Duke Nukem` universes, projectiles can be fired and interact with the background, the objects, the characters, even the player itself.
- There is a story with a mission, a goal to reach, a target (save the earth, get rich,... even something very simple, check out Doom).
- There is a beginning and an end for each level.
- There are sound effects.
- There's music.

Other considerations:

- A level editor is mandatory. It can be integrated with the main executable and activated by a command line option, or it can be a separate binary. It can be used to completely define a level: the areas and the differences in height between them, the textures, the actions and interactions, etc.
- Just like in the original games, you must pack in a single file all the elements you're going to need for the game: the level's design, textures and gameplay elements. One file per level is accepted, but it must be self-sufficient. The `doom-nukem` binary and the chosen level file must be self-sufficient.

Chapter VI

Bonus part

Expected bonuses :

- A menu to choose the level or some difficulty options.
- A genuine aesthetic research, an atmosphere, neat and detailed sceneries.
- A complex and researched story and scenario.
- Many useless interactions and animations that contribute to the immersion into the world (phone booth, vending machines, public toilets, projectile stains on the walls, etc).
- You can play in network with other players with a lobby, team/solo modes, capture the flag mode, etc.
- Multithreaded rendering using pthread.
- PLenty of other things we haven't even imagined yet!

`The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.`

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.