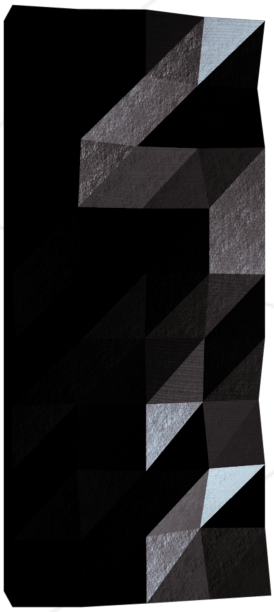

Wolf3D

Call Apogee Say Aardwolf

Pedago Team pedago@42.fr

Résumé: Inspiré du célèbre jeu éponyme des années 90 qui fut le premier First Person Shooter, ce projet vous permet d'aborder la technique du ray-casting. Votre but sera de représenter une vue dynamique à l'intérieur d'un labyrinthe dans lequel on peut se déplacer.

Table des matières

Chapitre I

Préambule

FIGURE I.1 – Écran titre de `Wolfenstein 3D` par `Id Software`

`Wolfenstein 3D` est un jeu embématique du panthéon des jeux vidéos. Développé chez `Id Software` par les célèbrissimes John Carmack et John Romero, et publié en 1992 par `Apogee Software`, `Wolfenstein 3D` est le premier véritable "First Person Shooter" de l'histoire du jeu vidéo.

Incarnant l'espion Américain William "B.J." Blazkowicz, le joureur s'attaque et décime l'armée Nazi (et une armée de mutants morts vivants, tant qu'à y être) à l'aide d'armes plus desctructrices les unes que les autres. Chapitre culminant de l'Histoire, William affronte finalement un Adolf Hitler équipé d'une armure de combat robotique et de quatre Gatlings... Un véritable bijou d'écriture, de violence et de justesse historique.

`Wolfenstein 3D` est l'ancêtre de jeux tels que `Doom` (`Id Software`, 1993), `Doom II` (`Id Software`, 1994), `Duke Nukem 3D` (`3D Realm`, 1996) et `Quake` (`Id Software`, 1996), qui marquèrent à leur tour le monde du jeu vidéo pour toujours.

A votre tour de revivre l'Histoire...

FIGURE I.2 – John Romero (gauche) et John Carmack (droite) posant pour la postérité.

Chapitre II

Introduction

`Wolfenstein 3D` est un jeu plus complexe et plus abouti que ses graphismes simples et son gameplay violent ne le laissent supposer. John Carmack n'est pas un des meilleurs programmeurs du monde pour rien.

Dans ce projet, vous devez marcher sur les traces de ce programmeur de genie et écrire votre propre impémentation de `Wolfenstein 3D`... Bon peut-être pas TOUT `Wolfenstein 3D`, soyons réalistes, mais au moins la représentation en 3D par raycasting d'un labyrinthe dans lequel le joueur peut se déplacer.

Une fois cet objectif atteint, textures, armes, ennemis, secrets, sons, musiques, ... La limite est votre imagination.

Nous vous recommandons la lectures de ces quelques articles pour en apprendre d'avantage au sujet du raycasting :

- http://fr.wikipedia.org/wiki/Raycasting
- http://projet-moteur-3d.e-monsite.com/pages/raycasting/raycasting.html
- http://zupi.free.fr/PTuto/index.php?ch=ptuto&p=ray
- https://developer.mozilla.org/samples/raycaster/RayCaster.html

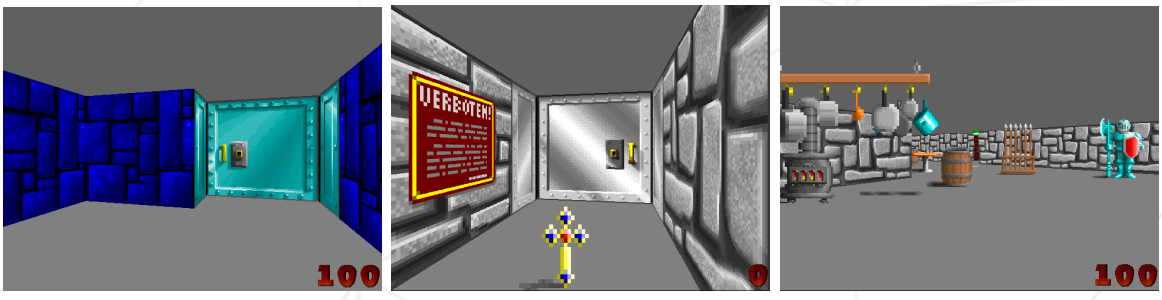

FIGURE II.1 – Utilisation du raycasting pour le rendu graphique du projet `Wolf3D`.

Chapitre III

Objectifs

Les objectifs de ce projet comprennent toujours les objectifs communs à tous les premiers projets ayant lieu avant le stage : rigueur, pratique du `C`, pratique d'algos classiques, recherche d'infos, exploitation de documentations, etc.

Étant un projet de programmation graphique, `Wolf3D` vous permettra également de consolider vos acquis specifiques à cette discipline : les fenêtres, les couleurs, les events, remplir des formes, etc.

Pour conclure, `Wolf3d` est un formidable terrain de jeu pour explorer les applications vidéo-ludiques des mathématiques sans pour autant les comprendre en détails. John Carmack est un genie pour avoir été l'un des premiers à y penser et reste célèbre pour ses implémentations, mais à l'aide des très nombreuses documentations que vous trouverez sur internet, les mathématiques ne seront pour vous qu'un outil pour créer des algorithmes élégants et pourquoi pas, efficaces.

Imaginez-vous que nous sommes en 1991, que vous travaillez pour une jeune entreprise de Mesquite au Texas, que vous aimez les films gores, le Heavy Metal, que les GPU et l'acceleration matérielle n'existent pas, et que vous allez devenir une rock star de la programmation de jeux videos dans moins de deux ans. Bref, amusez-vous !

Chapitre IV

Consignes générales

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et de nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées içi.
- L'exécutable doit s'appeller `wolf3d`.
- Vous devez rendre un `Makefile`. Ce `Makefile` doit compiler le projet, et doit contenir les règles habituelles. Il ne doit recompiler et relinker le programme qu'en cas de nécessité.
- Si vous êtes malin et que vous utilisez votre biliothèque `libft` pour votre `wolf3d`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépôt de rendu. Votre `Makefile` devra compiler la bibliothèque, en appelant son `Makefile`, puis compiler votre projet.
- Les variables globales sont interdites.
- Votre projet doit être en `C` et à la Norme. La norminette fait foi.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (segmentation fault, bus error, double free, etc...).
- Votre programme ne doit pas avoir de fuites mémoire.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Vous pouvez utiliser la `MinilibX` native `MacOS` qui est sur les dumps, ou alors, vous pouvez utiliser la `MinilibX` à partir de ses sources qui seront à intégrer de la même manière que celles de la libft. Dernière possibilité, vous pouvez aussi utiliser d'autres bibliothèques d'affichage (`X11`, `SDL`, etc...). Si la bibliothèque que vous utilisez n'est pas installée sur les machines de cluster, vous devrez fournir les sources de cette bibliotèque dans votre rendu, et elle devra se compiler automatiquement exactement comme la `MinilibX` ou votre `libft`, sans autre action que de compiler votre rendu. Quelle que soit la bibliothèque d'affichage, vous ne devez utiliser que ses fonctions de dessin basiques similaires à celles de la `MinilibX` : ouvrir une

fenêtre, allumer un pixel et gérer les évènements.

- Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes de la libc :
  - `open`
  - `read`
  - `write`
  - `close`
  - `malloc`
  - `free`
  - `perror`
  - `strerror`
  - `exit`
  - Toutes les fonctions de la lib math (`-lm` et `man 3 math`)
  - Toutes les fonctions de la `MinilibX` ou leurs équivalents dans une autre bibliothèque d'affichage.
- Vous avez l'autorisation d'utiliser d'autres fonctions, voire d'autres bibliothèques, dans le cadre de vos bonus à condition que leur utilisation soit dûment justifiée lors de votre correction. Soyez malins.
- Vous pouvez poser vos questions sur le forum, sur slack, ...

Chapitre V

Partie obligatoire

Tout le fun de ce projet réside dans ses bonus. Mais avant de vous lancer dans la chasse aux Nazis, vous devez d'abords réussir complètement et parfaitement sa partie obligatoire.

Vous devez créer la représentation graphique "réaliste" en 3D que l'on pourrait avoir à l'intérieur d'un labyrinthe en vue subjective. Vous devez réaliser cette représentation en utilisant le principe du Ray-Casting mentionné plus tôt.

- Vous avez le choix de la taille et de la forme du labyrinthe.
- La gestion de votre fenêtre d'affichage doit rester cohérente : passage d'une autre fenêtre dessus, minimisation, ...
- Appuyer sur la touche `ESC` doit fermer la fenêtre et quitter le programme proprement.
- Cliquer sur la croix rouge sur la bordure de la fenêtre doit fermer la fenêtre et quitter le programme proprement.
- Les flèches du clavier doivent permettre de se déplacer en temps réel dans le labyrinthe, comme dans le jeu d'origine.
- La couleur des murs doit varier suivant l'orientation (nord, sud, est, ouest). L'implémentation de certains bonus de ce sujet peut éclipser cette consigne (textures sur les murs par exemple).

FIGURE V.1 – Rendu type tel que demandé dans la partie obligatoire.

Chapitre VI

Partie Bonus

Les bonus ne seront évalués que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc. Concrètement, cela signifie que si votre partie obligatoire n'obtient pas TOUS les points à la notation, vos bonus seront intégralement IGNORÉS.

- On ne rentre pas dans les murs
- Des textures sur les murs
- Une texture de ciel (une skybox)
- Une texture de sol et/ou de plafond
- Des objets dans le labyrinthe
- On ne rentre pas dans certains objets
- Des objets que l'on ramasse pour faire des points
- Des portes qui s'ouvrent et se ferment
- Des méchants à combattre
- Des passages secrets
- Des animations
- Plein de niveaux
- Son et musique
- Un moteur `Doom` (ça c'est très sexy !)
- Un moteur `Duke Nukem 3D` (ça c'est très très sexy)

Chapitre VII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépôt `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué.

Bon courage à tous et n'oubliez pas votre fichier auteur !