



Cub3D

Mon premier RayCaster avec la minilibX

Résumé: Ce projet est inspiré du jeu Wolfenstein3D, considéré comme le premier FPS jamais développé. Il vous permettra d'explorer la technique du ray-casting. Votre objectif est de faire une vue dynamique au sein d'un labyrinthe, dans lequel vous devrez trouver votre chemin.



Version: 9

Table des matières

I	Préambule	2
II	Objectifs	3
III	Règles communes	4
IV	Partie Obligatoire - Cub3D	5
V	Bonus part	10
VI	Exemples	11



Chapitre I

Préambule

Développé par ID Software et les ultra-célèbres John Carmack et John Romero, publiée en 1992 par Apogee Software, Wolfenstein 3D est le premier "First Person Shooter" dans l'histoire du jeu-vidéo.



FIGURE I.1 – John Romero (gauche) et John Carmack (droite) posant pour la postérité

Wolfenstein 3D est l'ancêtre des jeux tels que Doom (Id Software, 1993), Doom II (Id Software, 1994), Duke Nukem 3D (3D Realm, 1996) et Quake (Id Software, 1996), qui sont des pierres angulaires additionnelles dans le monde du jeu vidéo.

Et maintenant, c'est votre tour de revivre l'Histoire...

Chapitre II

Objectifs

Les objectifs de ce projet sont similaires à tous les autres projets et leurs objectifs : Rigueur, utilisation du C, utilisation d'algorithmes basiques, recherche d'informations, etc.

Comme c'est un projet de design graphique, Cub3D vous permettra de travailler vos talents de designer : fenêtres, couleurs, événements, formes, etc.

En conclusion, Cub3D est une aire de jeu remarquable pour explorer les applications pratiques des mathématiques sans avoir à en comprendre les spécificités.

Avec l'aide des nombreux documents disponibles sur internet, vous utiliserez les mathématiques en temps qu'outil de création d'algorithmes élégants et efficaces.



Nous vous recommandons de tester le jeu original avant de commencer :
<http://users.atw.hu/wolf3d/>

Chapitre III

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront incluses dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier différent : `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier `libft` contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre IV

Partie Obligatoire - Cub3D

Nom du programme	cub3D
Fichiers de rendu	Tous les fichiers
Makefile	all, clean, fclean, re, bonus
Arguments	Une map dans le format *.cub
Fonctions externes autorisées	<ul style="list-style-type: none">• open, close, read, write, printf, malloc, free, perror, strerror, exit• Toutes les fonctions de la lib math (-lm man man 3 math)• Toutes les fonctions de la MinilibX
Libft autorisée	Yes
Description	Vous devez créer une représentation graphique 3D "réaliste" que nous pourrions avoir au sein d'un labyrinthe en utilisant une vue subjective. Vous devez créer cette représentation en utilisant les principes du ray-casting mentionnés plus tôt.

Les contraintes sont les suivantes :

- Vous devez utiliser la `minilibX`. Soit dans la version disponible sur votre OS, ou depuis ses sources. Si vous décidez de travailler avec les sources, les mêmes règles que la `libft` s'appliquent comme ceux écrits ci-dessus dans la partie **Common Instructions**.
- La gestion des fenêtres doit être parfaite : gestion de la minimalisation, du passage d'une autre fenêtre, etc
- Vous devez afficher des textures différentes (vous avez le choix) selon si les murs sont face nord, sud, est, ouest.

- Votre programme doit être capable d'avoir des couleurs différentes pour le sol et le plafond
- Le programme affiche l'image dans une fenêtre et respecte les règles suivantes :
 - Les touches flèches du gauche et droite du clavier doivent permettre de faire une rotation de la caméra (regarder à gauche et à droite)
 - Les touches W, A, S et D doivent permettre de déplacer la caméra (déplacement du personnage)
 - Appuyer sur la touche ESC doit fermer la fenêtre et quitter le programme proprement
 - Cliquer sur la croix rouge de la fenêtre doit fermer la fenêtre et quitter le programme proprement
 - L'utilisation d'`images` de la `minilibX` est fortement recommandée.
- Votre programme doit prendre en premier argument un fichier de description de scène avec pour extension `.cub`
 - La map doit être composée d'uniquement ces 6 caractères : **0** pour les espaces vides, **1** pour les murs, et **N,S,E** ou **W** qui représentent la position de départ du joueur et son orientation.
Cette simple map doit être valide :

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

- La map doit être fermée/entourée de murs, sinon le programme doit renvoyer une erreur.
- Mis à part la description de la map, chaque type d'élément peut être séparée par une ou plusieurs lignes vides.
- La description de la carte sera toujours en dernier dans le fichier, le reste des éléments peut être dans n'importe quel ordre.
- Sauf pour la map elle-même, les informations de chaque élément peuvent être séparées par un ou plusieurs espace(s).
- La carte doit être parsée en accord avec ce qui est présenté dans le fichier. Les espaces sont une partie valable de la carte, c'est à vous de les gérer correctement. Vous devez pouvoir parser n'importe quelle sorte de carte, tant qu'elle respecte les règles de carte.

- Pour chaque élément, le premier caractère est l'identifiant (un ou deux caractères), suivi de toutes les informations spécifiques à l'élément dans un ordre strict tel que :

- texture nord :

```
NO ./path_to_the_north_texture
```

- identifiant : **NO**

- chemin vers la texture nord

- South texture :

```
SO ./path_to_the_south_texture
```

- identifiant : **SO**

- chemin vers la texture sud

- West texture :

```
WE ./path_to_the_west_texture
```

- identifiant : **WE**

- chemin vers la texture ouest

- East texture :

```
EA ./path_to_the_east_texture
```

- identifiant : **EA**

- chemin vers la texture est

- Couleur du sol :

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

- identifiant : **F**

- couleurs R,G,B range [0,255] : **0, 255, 255**

- Couleur du plafond :

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

- identifiant : **C**
- couleurs R,G,B range [0,255] : **0, 255, 255**

- Exemple minimaliste de scène de la partie obligatoire .cub :

```
NO ./path_to_the_north_texture
SO ./path_to_the_south_texture
WE ./path_to_the_west_texture
EA ./path_to_the_east_texture

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

111111111111111111111111
100000000011000000000001
101100000111000000000001
100100000000000000000001
111111110110000011100000000001
100000000011000001110111111111
11110111111101100000010001
111101111111011101010010001
11000000110101011100000010001
100000000000000001100000010001
100000000000000001101010010001
1100000111010101111011110N0111
11110111 1110101 101111010001
11111111 11111111 111111111111
```

- Si un problème de configuration de n'importe quel type est rencontré dans le fichier, le programme doit quitter et renvoyer "Error\n" suivi d'un message d'erreur explicite de votre choix.

Chapitre V

Bonus part



Les bonus ne seront évalués que si votre partie obligatoire est PARFAITE. Par PARFAITE, nous voulons naturellement dire que tout est complet, que ça n'échoue dans aucun cas de figure, même en cas d'erreur terrible comme un mauvais usage ou autre. Si vous n'avez pas tous les points de la partie obligatoire, votre partie bonus sera totalement ignorée.

Liste des bonus :

- Collision contre les murs
- Un système de minicarte.
- Porte qui peuvent être ouvertes/fermées.
- Animations (des sprites animés).
- Faites pivoter le point de vue avec la souris.



Vous pourrez créer de meilleurs jeux plus tard, ne perdez pas trop de temps !



Dans le cadre des bonus, vous êtes autorisé à utiliser d'autres fonctions ou à ajouter des symboles sur la carte pour compléter la partie bonus tant que leur utilisation est justifiée lors de votre évaluation. Vous êtes également autorisé à modifier le format de fichier de scène attendu pour l'adapter à vos besoins. Soyez intelligent !

Chapitre VI

Exemples



FIGURE VI.1 – Wolfenstein3D Jeu original utilisant le ray-casting.

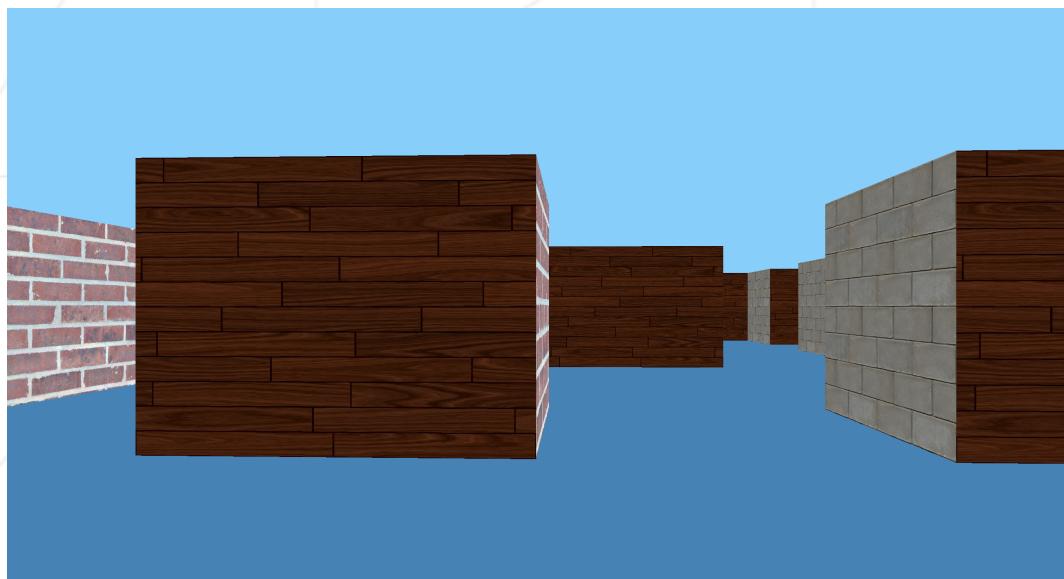


FIGURE VI.2 – Exemple de ce que la partie obligatoire attend de vous.



FIGURE VI.3 – Exemple de bonus avec une minimap, des textures sol et plafond, et un célèbre hérisson animé.



FIGURE VI.4 – Un autre exemple de bonus avec un HUD, une barre de vie, un effet d'ombre et une arme qui peut tirer

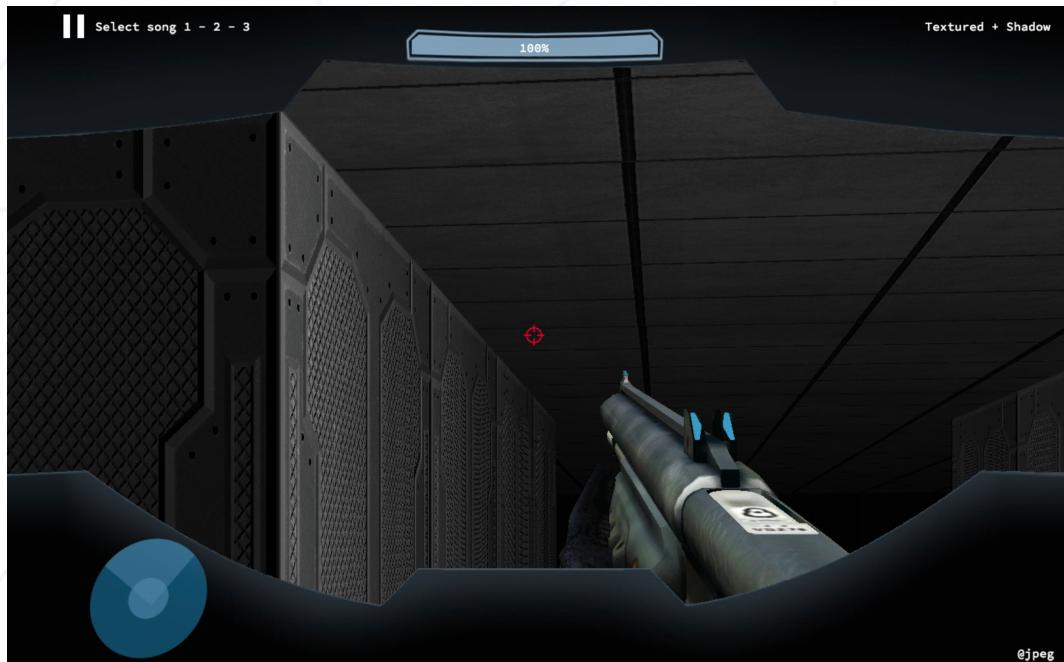


FIGURE VI.5 – Autre exemple de partie bonus avec une arme de votre choix et le joueur qui regarde le plafond