COLORED MEMORIES

Nelly DUPUYDENUS
Loona GAILLARD
IMAC 2022
ANNEE 2020-2021

INTRODUCTION

Dans le cadre de nos études en école d'ingénieur IMAC (Image Multimédia Audiovisuel et Communication), nous avons été amenées à réaliser un environnement de jeu vidéo dans lequel l'utilisateur peut se déplacer. L'objectif de ce projet est de nous familiariser avec la programmation orientée objet, d'explorer et utiliser les fonctions de dessin OpenGL 3+ ainsi que de mettre en application des principes d'architecture logicielle.
Le projet s'est déroulé en plusieurs étapes ; nous avons d'abord dû nous approprier le projet et ses conditions, afin de bien comprendre toutes les demandes.
Puis nous avons ensuite décidé d'un synopsis et d'une ambiance que nous voulions donner à notre environnement.
Puis est venue la question de l'architecture de notre programme.

Nous verrons donc au cours de cette synthèse les points suivants: conception et réalisation de notre environnement, du game design aux fonctionnalités implémentées. Nous reviendrons sur l'organisation de notre projet, qui a été perturbée dûe aux conditions de travail dans lesquelles nous avons évolué au cours de ce projet. quelles sont les difficultés que nous avons pu rencontrer.

SOMMAIRE

CAHIER DES CHARGES

Synthèse d'image :

Ce que l'on a :	Ce que l'on n'a pas :
Utilisation des VertexBufferObject, desVertexArrayObject et des Index Buffer Object. Utilisation des shaders. Une lumière directionnelle + une spotlight. Vous proposerez à minima un modèle d'illumination de type Blinn-Phong avec tex- ture.	Un fichier texte qui décrit chaque meshes.

Développement :

Ce que l'on a :	Ce que l'on a pas :
Compilation avec CMake classes classes et fonctions template usage d'outils de la STL messages d'erreurs pour traiter les erreurs utilisateurs (fichiers inexistants, en- trées invalides, etc.). espaces de nommage fonctions lambdas	héritage polymorphisme exceptions pour traiter les erreurs systèmes (version d'opengl incompatibles, plus de ram, etc.)

Fonctionnalités

Milestone 0 (Charger une scène 3D)

- Charger l'ensemble des modèles 3D (positions, UVs, normals...) et positionner ces éléments dans la scène.
- Charger les matériaux (paramètres des interactions lumières-matières des modèles 3D : coef de diffusion, albedo, et textures...).
- Charger les lumières (au moins une lumière directionnelle).

Milestone 1 (Explorer la scène)

- Explorer la scène en vue à la première personne : Se déplacer avec le clavier et regarder autour de soi avec la souris.

Milestone 2 (Interagir avec la scène)

- Animer un objet ou changer l'aspect de la scène en jouant avec les matériaux et les lumières en cliquant sur un objet ou en allant dans un endroit spécifique
- Utiliser des portails pour passer d'un monde à un autre / d'une scène à une autre

Milestone 3 (Jouer)

- Résoudre un(e) énigme/puzzle ou retrouver une série d'objets cachés
- Trouver une sortie ou une destination spécifique

Fonctionnalités additionnelles

- Une bonne expérience immersive possède une bonne ambiance sonore, utilisez SDL_mixer pour jouer une musique et créer des effets sonores
- Utiliser la librairie Assimp ou le format de fichier glTF pour charger des objets 3D
- Ajouter un narrateur (textuel ou auditif)
- Gérer les collisions

PRESENTATION DU JEU

I. SYNOPSIS

Vous êtes le seul capable de vous ramener à la vie. Dans d'étranges univers colorés composés d'objets de la vie quotidienne, vous devez retrouver le portail qui vous permettra de retrouver les vôtres. Vous traversez des objets ayant appartenu à votre vie tel un fantôme, si vous vous cognez à quelque chose il s'agit du portail...!!!

II. FONCTIONNALITÉS

1. Démarrage du jeu :

Pour démarrer le jeu il vous suffit d'ouvrir votre terminal puis de vous rendre dans le dossier Template-build du jeu grâce à la commande **cd Bureau/.../Template-build.** une fois dans le dossier, il vous suffit de taper **cmake ../Template**, tout une configuration se fait. Ensuite, entrez **make** en ligne de commande. Le fichier crée un exécutable sous le nom de **Colored_Memories** dans le dossier src.
Pour exécuter ce fichier, tapez **./src/Colored_Memories.**
Une fenêtre devrait alors apparaître. Laissez la scène se charger entièrement.

```
loona@loona-VirtualBox:~$ cd Bureau/Colored_Memories/Template-build
loona@loona-VirtualBox:~/Bureau/Colored_Memories/Template-build$ cmake ../Template-build
```

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/loona/Bureau/Colored_Memories/Template-build
loona@loona-VirtualBox:~/Bureau/Colored_Memories/Template-build$ make
[ 52%] Built target glimac
[100%] Built target Colored_Memories
loona@loona-VirtualBox:~/Bureau/Colored_Memories/Template-build$ ./src/Colored_Memories
```

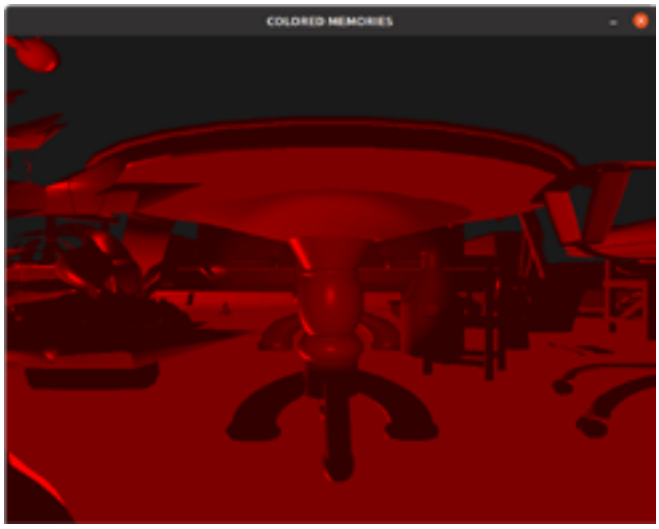
2. Les différents mondes

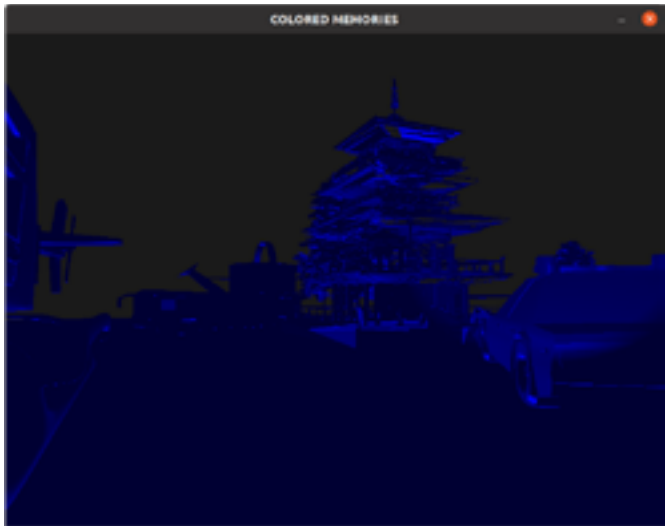
Après avoir lancé la partie, le joueur atterrit dans la première scène, il s'agit d'une scène simple composée de 4 cubes et de lumière blanche, souvent synonyme de la mort.

Dès son arrivée dans la scène un audio explicatif se lance. Cet audio est là pour lui expliquer le synopsis et comment se déplacer dans la scène. Les déplacements sont gérés grâce aux touches **Z** (en avant), **S** (en arrière), **Q** (à gauche) et **D** (à droite). Le joueur peut également effectuer une rotation autour de lui grâce au clic droit de sa souris accompagné d'un léger déplacement de sa souris dans le sens où il veut s'orienter.
Pour passer d'une scène à l'autre, le joueur doit appuyer sur **entrée**.

Le deuxième monde est un monde rouge, souvent représentatif du danger, il plonge le spectateur dans une ambiance assez angoissante.

La musique en arrière-plan ajoute la sensation de danger et de quête à l'univers graphique.





Le troisième monde est un monde bleu, couleur de quelque chose de meilleur et qui fait souvent rêver comme le ciel.

La musique est différente et elle donne une dimension encore plus dramatique à la scène, on sent qu'on est proche de la fin.

Une fois que le joueur a réussi à trouver tous les portails qui le ramènent à la vie, il se retrouve dans la scène que celle de l'introduction.
Une voix off explique qu'il a gagné le jeu.

Pour quitter le jeu, le joueur doit appuyer sur entrée. La fenêtre du jeu se ferme alors.

3. Synthèse d'image

Création et importation model 3D :

Afin de créer notre univers, nous avons utilisé ***Assimp***. Cette bibliothèque open source peut gérer de nombreux formats d'objet 3D et les intégrer à une scène OpenGL.

Pour créer nos modèles 3D, nous avons utilisé le logiciel ***AutoDesk Maya 3D***.
Après avoir créer nos objets, scènes dans Maya et de leur avoir appliqué une texture,, nous exportons le fichier sous format .obj. En exportant sous ce format, Maya nous crée un fichier .mbl qui permettra ensuite à Assimp de pouvoir lire et afficher correctement nous objets. Nous plaçons ensuite ces fichiers dans le dossier ***assets/models/nom_dossier*** ce qui nous permet d'avoir une arborescence correctement organisée. Pour pouvoir afficher correctement la texture Assimp a besoin de l'image utilisée dans Maya, c'est pour quoi des fichiers images se trouvent également dans ces dossiers.

Nous voulions que notre portail ait une forme neutre et basique, c'est pourquoi nous avons décidé de choisir un cube. Pour créer ce cube nous utilisons un ***tableau de vertices*** qui définit chaque sommets de notre cube. Au lieu d'appliquer une texture à notre cube nous lui appliquons une couleur directement.
Pour créer ce cube nous avons créé une structure ***TresorProgram*** (dans tresor.hpp) dans laquelle il y a la déclaration de la position de ce dernier ainsi que les vbo et vao qui seront utilisés pour sa construction. Il y a ensuite des fonctions qui permettent l'initialisation du cube, soit le bindage des vao et vbo (fonction ***initilisationTresor***). La fonction ***DrawTresor*** permet d'afficher le cube dans la boucle de rendu.

Chargement lumières :

Concernant les lumières, nous avons créé des shaders qui gèrent plusieurs lumières en même temps. En effet, nous voulions que le joueur ait l'impression d'avoir une lampe torche lors de ces déplacements. Mais la scène avait également besoin d'une lumière directionnelle. Nous utilisons 4 shaders en tout ; une paire de shaders qui prend en compte la texture des objets importés via Assimp, il s'agit des shaders ***multipleLights.vs.glsl*** et de ***lightShader.fs.glsl***. L'autre paire (***tresorShader.fs.glsl*** et ***tresorShader.vs.glsl***) prend en compte la couleur du cube portail. La structure des shaders reste la même mais au lieu de travailler sur une texture on travaille sur la couleur du cube.
Les lumières et shaders sont chargées grâce aux structures ***ObjetProgram*** (dans structures.hpp) et ***TresorProgram*** (dans tresor.hpp). Dans ces fichiers nous faisons appel à la class ***Program*** qui se situe dans ***l'espace de nommage glimac***.

Explorer la scène

Pour se déplacer dans la scène le joueur doit utiliser son clavier :
Z : en avant.
S : en arrière
Q : à gauche
D : à droite
Il peut également tourner autour de lui grâce au clic droit de sa souris.
Tout cela est géré grâce à la classe ***SDLWindowManager*** dans l'espace de nommage glimac ; notamment avec les fonctions ***isKeyPressed()***, ***isMouseButtonPressed()***.

Interagir avec la scène

Le joueur interagit avec la scène lorsqu'il rentre en collision avec le cube. Cette action fait office de portail et permet au joueur de se téléporter de la scène rouge à la scène bleue. Lors de cette collision un bruit de choc est émis et l'écran devient noir.

Jouer

Notre jeu propose une énigme à résoudre ; il faut retrouver l'objet qui ramènera le joueur à la vie parmi tout un tas d'objets de la vie quotidienne qui se ressemblent les uns les autres.

Fonctionnalités additionnelles

Notre jeu fonctionne grâce à l'univers coloré que nous lui avons donné. Mais il n'y a pas d'image sans son. C'est pourquoi nous avons décidé d'utiliser ***SDL_Mixer***. Cette fonctionnalité permet de plonger définitivement le joueur dans notre univers et de donner une autre dimension au jeu. En effet, en plus des sentiments de tension et danger ressenti par l'ambiance visuelle, le joueur ressent également cette pression par l'ouïe. Nous avons donc choisi des musiques épiques pour immerger le joueur de la meilleure des manières.

Pour permettre au joueur de se plonger dans notre univers, nous avons trouvé nécessaire de faire commencer le jeu dans une scène neutre où les explications et directives concernant le jeu lui sont indiquées par voie orale. Nous nous sommes donc enregistrés afin de proposer un texte personnalisé.

Notre portail est activé lorsque le joueur le percute. Pour effectuer cela, il a fallu gérer les collision avec le cube. Nous avons donc créé la fonction ***CheckCollisions*** (dans le fichiers collisions.hpp) qui renvoie un ***booléen***. Cette fonction prend en paramètre les coordonnées actuelles du joueur (float x, y, z) ainsi que le cube pour pouvoir récupérer sa position. Si la fonction détecte une collision alors elle renvoie true sinon elle renvoie false.
C'est ce booléen que nous avons utilisé dans la boucle de rendu pour effectuer nos changements de scènes ainsi que pour jouer le son de choc.

III. Développement

1. Nos classes :

Game

Notre classe Game permet de gérer tout ce qui concerne le jeu et de lancer les différentes étapes du jeu. En effet, elle est constituée de plusieurs attributs qui sont des booléens et qui, selon leur valeur, lancent différents moments du jeu grâce à plusieurs conditions gérées avec des if dans le main.cpp. Le constructeur du jeu permet d'initialiser le jeu avec les valeurs rentrées au début du main.cpp. La classe est ensuite constituée d'accesseurs (getter) et de mutateurs (setter) permettant respectivement de récupérer les valeurs de chaque booléen et de les modifier.

Scene

Notre classe scène nous permet de gérer nos scene.obj, ainsi que la musique et le bruitage. Le constructeur de la scène appelle la ***fonction loadScene*** qui est une fonction d'initialisation de la scène. Elle remplit les ***vector objets, musiques et bruitages***, correspondant à la scène qui est demandée (méthode ***push_back()***), grâce à l'appel des constructeurs de la classe ***Mode***l et ***SDL_Mixer***. Lors de cette initialisation nous ***gérons les erreur*** liées à la création des ***vector Mix_Chunk***. S'il y a une erreur dans l'importation de model alors la classe Model nous en averti c'est pourquoi il n'y a pas de gestion d'erreur concernant cela dans la fonction loadScene. Si la fonction loadScene n'a pas pu fonctionner alors ***l'erreur est affichée dans le terminal***.

La ***fonction DrawMyScene*** permet d'afficher les models chargés. Pour chaque objet on appelle la méthode Draw de la classe Model (dans model.hpp).
De la même manière, la ***fonction mixMyScene*** permet d'enclencher la fonction qui lit les fichiers audio.

2. Fonctions lambdas

Pour faciliter le traitement concernant les lumières, nous avons créé des fichiers propres à cette tâche ; ***lumières.cpp*** et ***lumières.hpp***. Dans ces fichiers vous pouvez retrouver les prototypes des fonctions ainsi que ces dernières. Elles permettent simplement de gérer l'envoi des données aux shaders correspondants au programme utilisé.

Nous avons également fait une fonction rattachée à aucune classe pour pouvoir gérer les collisions

CONCLUSION

A travers la réalisation de ce projet, nous avons été confrontées à des difficultés d'ampleur plus ou moins différentes. Tout d'abord, nous pouvons noter les difficultés liées à notre aspect novice dans l'utilisation des shaders et d'OpenGL en général qui ralentissaient notre avancée.

Ensuite, il y a eu des problèmes de matériel qui ont été plutôt handicapants. La machine virtuelle de Loona était très lente, c'est-à-dire que tous les déplacements de la souris sont très lents et souvent peu précis. L'écriture était également compliquée, en effet la machine prend en compte le temps où la touche est appuyée et écrit en autant de fois la lettre demandée que de temps d'appui, et la sensibilité est très élevée. La machine a également du mal à suivre lors de l'ouverture d'application, ce qui l'a obligé à souvent quitter la virtual box de manière forcée. Réinstaller plusieurs fois une machine virtuelle en essayant de changer la configuration n'y a rien changé. Cela la ralentissait beaucoup pour coder et se déplacer dans les scènes 3D générées était très compliqué.

Et enfin, il y a également eu les problèmes concernant la santé mentale qui ont beaucoup affecté la mise en œuvre du projet. La situation sanitaire actuelle pèse beaucoup sur le moral des étudiants en général et elle vient s'ajouter à des problèmes déjà présents. Nelly a notamment souffert de multiples crises d'angoisses intensifiant un problème de concentration déjà présent. Loona n'a pas non plus échappé à cela.

Malgré ces problèmes, nous nous sommes accrochées et nous sommes fières de notre rendu de projet final. Il est loin d'être parfait mais il remplit (presque) tous les critères. Ce projet nous a appris à avoir une meilleure architecture entre nos fichiers. De plus, la découverte de la programmation en 3D était plutôt satisfaisante et elle nous a réconciliées avec l'utilisation d'OpenGL, car l'expérience de l'année passée n'avait pas laissé de très bons souvenirs (encore dûs aux conditions de travail exceptionnelles que nous avions pu avoir).

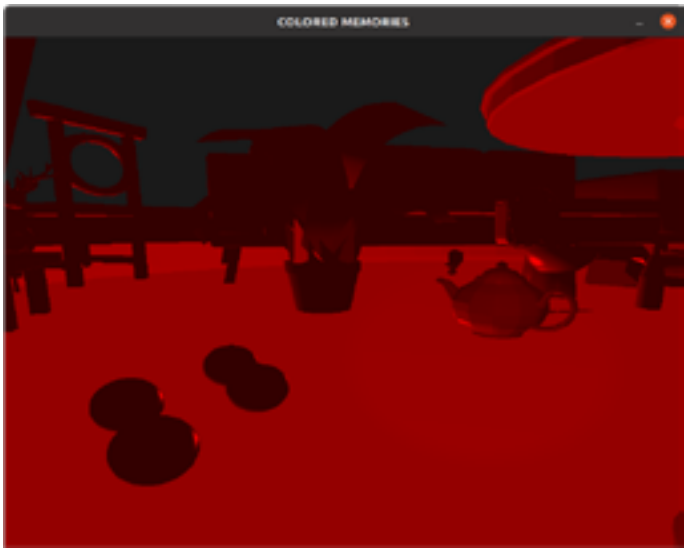
REMERCIEMENTS

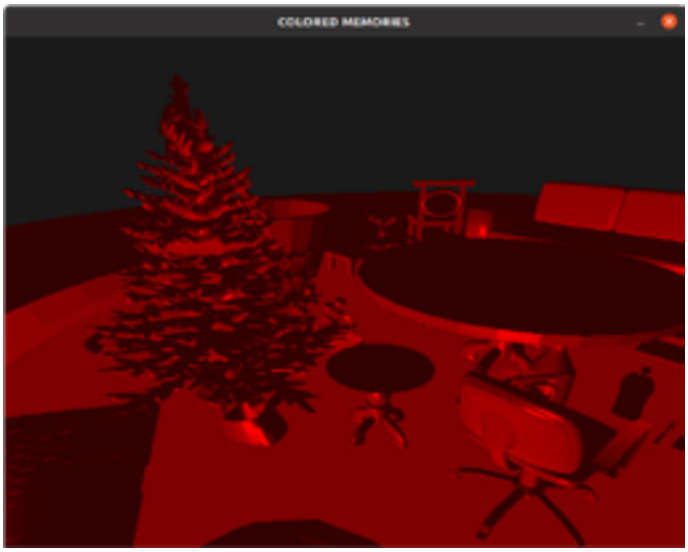
Nous tenons à remercier l'équipe pédagogique présente sur ce projet pour son enseignement et son aide apportée tout au long de ce module malgré les conditions difficiles et exceptionnelles que nous traversons.

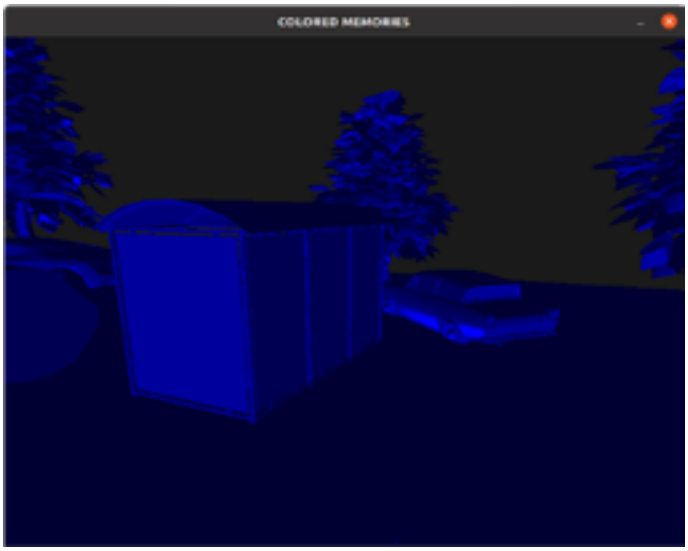
Crédits :
Musique Catalyst, auteur : Alexander Nakarada
(https://www.youtube.com/watch?v=DB5JuUbYbv0)
Musique Courage and Willpower, auteur : Keys of Moon Music
(https://youtu.be/AkwkvBhsFvI)

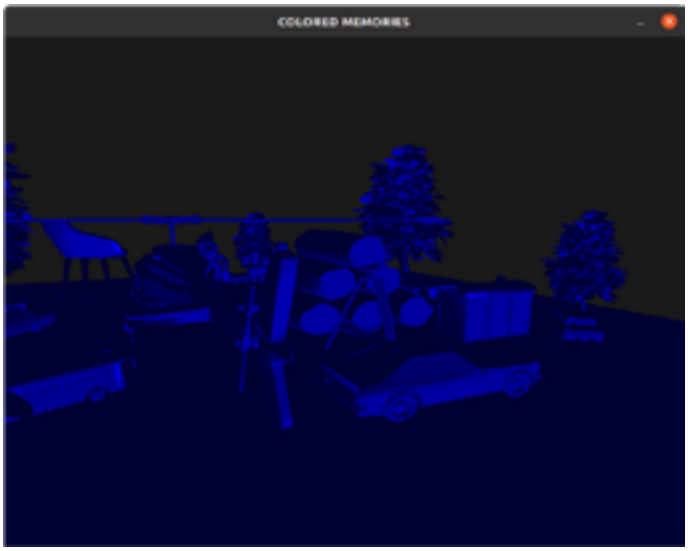
Nelly DUPUYDENUS
Loona GAILLARD
IMAC 2022
ANNEE 2020-2021

ANNEXES


COLORED MEMORIES


COLORED MEMORIES


COLORED MEMORIES


COLORED MEMORIES

COLORED MEMORIES