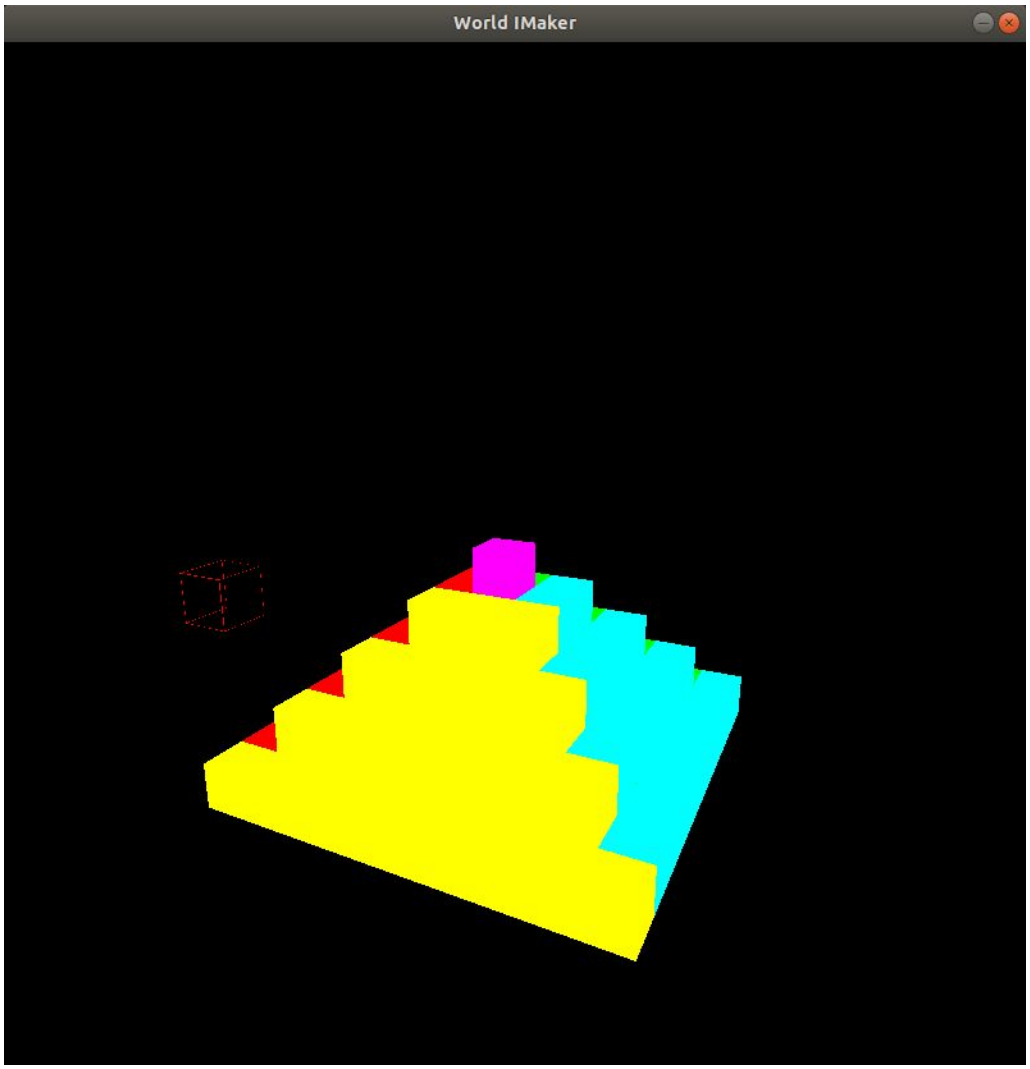
RAPPORT DE PROJET
WORLD IMAKER



CHIKAR Louisa - LESBATS Lucie

IMAC2

INTRODUCTION

World Imaker est un outil d'édition et de visualisation d'une scène 3D construite à base de cubes réalisé par CHIKAR Louisa et LESBATS Lucie. Grâce à notre programme, l'utilisateur peut créer une scène en ajoutant ou supprimant des cubes, en les déplaçant et en les faisant changer de couleur, mais aussi en extrudant ou en creusant la scène. Son oeuvre peut ensuite être sauvegardée pour être chargée plus tard. Nous avons également implémenté des fonctions mathématiques permettant à l'utilisateur de générer une scène en fonction d'un fichier de configuration.

Ce projet ayant pour but de d'appliquer les notions vues en cours de C++, mathématiques pour l'informatique et synthèse d'images, nous nous sommes servi des cours et des TP comme base de travail.

FONCTIONNEMENT DE L'APPLICATION

Le programme se divise en deux interfaces: la fenêtre SDL et le terminal. Pour lancer l'application il suffit d'exécuter la commande *./src/worldimaker*.

Ensuite, l'utilisateur a le choix entre plusieurs démarrages:

- Soit il charge une scène sauvegardée en entrant le chemin du fichier, la scène s'affiche alors;
- soit il génère une scène en entrant le chemin du fichier de configuration, la scène s'affiche alors;
- soit il initialise une scène à zéro, la scène s'affiche avec une base de trois cubes de hauteur.

À partir de là, l'utilisateur peut modifier la scène à sa guise ou la sauvegarder pour pouvoir la charger lorsqu'il reviendra sur l'application.

SCHÉMA DE SÉQUENCE

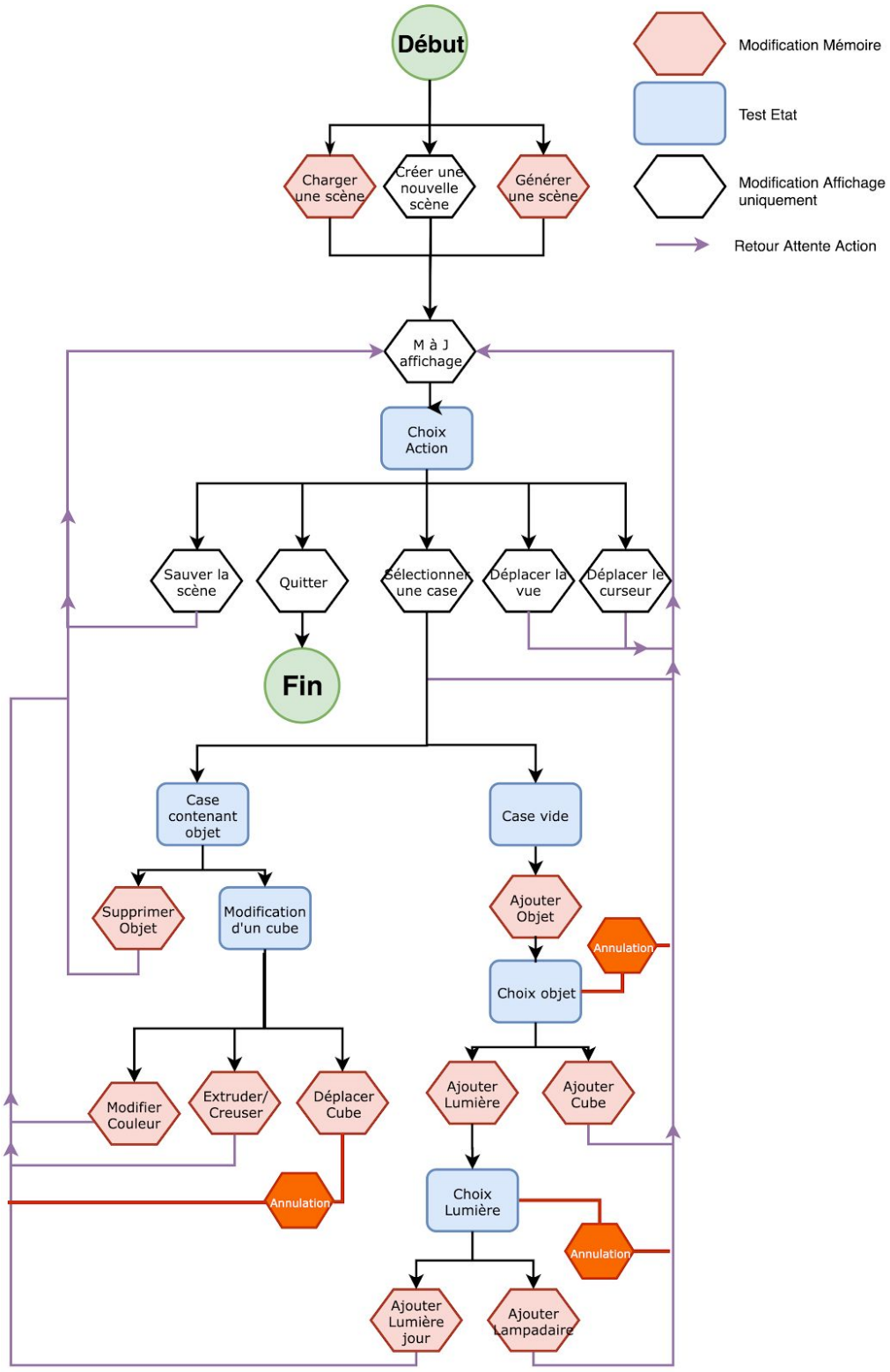
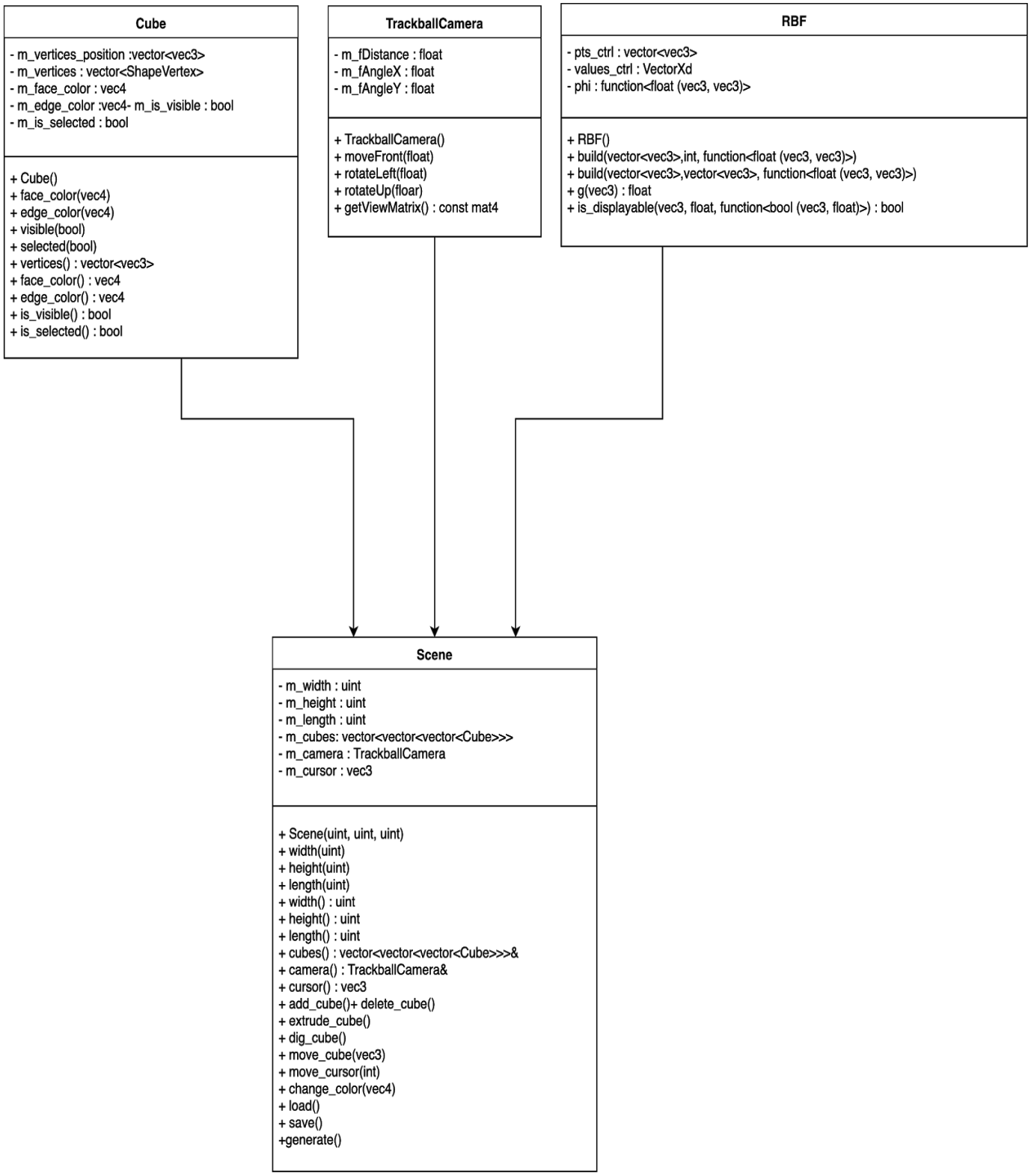


DIAGRAMME DE CLASSES



FONCTIONNALITÉS

FONCTIONNALITÉ	IMPLÉMENTATION INTÉGRALE	IMPLÉMENTATION PARTIELLE
Affichage d'une scène avec des cubes	✓	
Visualisation de la scène grâce à une caméra	✓	
Edition des cubes	✓	
Sculpture du terrain	✓	
Génération procédurale	✓	
Ajout de lumières		❌ : lumières fonctionnelles mais pas élégantes
Sauvegarde/Chargement d'une scène (fonc. additionnelle)	✓	
Déplacement d'un cube (fonc. additionnelle)	✓	
Suppression de la scène (fonc. additionnelle)	✓	

Affichage d'une scène avec des cubes

- Utilisation de la caméra de la classe *Scene* pour afficher ses données

Dans la classe *Scene*, on retrouve tout ce dont on a besoin pour créer et afficher une scène 3D. Elle contient notamment un tableau de cubes ayant comme état "affiché", "sélectionné" ou "invisible", et une caméra qui se contrôle à la souris.

En fonction de l'état d'un cube, nous affectons des variables uniformes transmises aux shaders. Si un cube est affiché, la couleur de ses faces et de ses arêtes est celle choisie par l'utilisateur (vert par défaut) et sa transparence est minimale (alpha = 1), sinon sa transparence est maximale (alpha = 0). S'il est sélectionné ses arêtes sont rouges (indépendamment de la couleur des faces et de la visibilité du cube).

Pour montrer une sélection sur un cube invisible, on utilise la fonction *glPolygonMode* en mode ligne pour n'afficher que les arêtes des polygones. Toutefois, un cube est en réalité composé de plusieurs triangles et cette fonction affiche les arêtes de ces triangles. Il faut donc spécifier dans le fragment shader qu'il ne faut dessiner que les arêtes des triangles correspondant à celles du cube pour ne pas avoir de diagonales qui s'affichent.

Visualisation de la scène grâce à une caméra

L'utilisateur peut tourner autour du centre de la scène et ainsi visualiser ce qu'il a construit grâce à une caméra Trackball. Pour cela, il lui suffit de cliquer et glisser sa souris sur le côté ou vers le haut. Il peut se rapprocher et s'éloigner du centre de la scène grâce à la molette de sa souris ou grâce à son trackpad.

Nous avons commencé par implémenter une caméra Free Fly car nous pensions qu'elle serait plus appropriée à notre application. Cependant nous nous sommes rapidement rendu compte que les déplacements étaient très peu intuitifs du fait de l'absence de bras ou de mains représentant le personnage que l'utilisateur aurait endossé. Nous avons donc décidé de revenir à une caméra peut être plus simple mais de loin plus intuitive et plus pratique pour voir la scène sous tous ses angles et pour pouvoir se déplacer très facilement autour de son oeuvre.

Edition des cubes

- Sélection et modification des cubes

L'utilisateur utilise les flèches directionnelles ainsi que les touches ***P*** et ***M*** pour déplacer le curseur dans la scène. La sélection est visible de par les contours affichés en rouge. Une fois un cube sélectionné, l'utilisateur à le choix entre plusieurs couleurs associées à un numéro chacune : ***5*** rouge, ***6*** jaune, *7* cyan, ***8*** vert et ***9*** magenta.

Sculpture du terrain

- Extrusion ou creusement

L'utilisateur peut ajouter (touche ***1***), supprimer (touche ***2***), extruder (touche ***3***) ou creuser (touche ***4***) un cube.

Pour extruder, il suffit d'ajouter un cube en haut du cube sélectionné, s'il y a déjà un cube on ajoute au-dessus de celui-ci et ainsi de suite jusqu'à ce que la place soit libre pour ajouter un cube. S'il y a des cubes jusqu'à la hauteur maximale de la scène, l'utilisateur ne peut pas extruder.

Pour creuser, il faut supprimer le cube étant le plus au-dessus du cube sélectionné. S'il n'y en a pas c'est le cube lui-même qui est supprimé.

Génération procédurale

- Utilisation des RBF pour calculer une valeur interpolée qui détermine l'affichage ou non d'un cube
- Classe RBF générant la fonction calculant la valeur interpolée d'un cube selon une RBF, des points de contrôle et une valeur pivot.

La génération d'une scène se fait en plusieurs étapes:

1. Création des points de contrôle (choisis par l'utilisateur ou aléatoirement grâce à la fonction *shuffle*)
2. Affectation de valeurs aléatoires aux points de contrôle grâce à une fonction aléatoire de *distribution normale*
3. Choix de la fonction radiale de type lambda fonction

4. Choix de la valeur pivot : comme nous utilisons des valeurs aléatoires, nous avons trouvé plus efficace d'utiliser comme pivot la moyenne de toutes les valeurs, l'affichage se fait donc si la valeur interpolée d'un cube est "assez proche" de la valeur pivot
5. Création de la fonction d'interpolation : la valeur interpolée est calculée selon la formule

$$g(p) = \sum_{i=0}^{nb\ points\ de\ controle\ -1} \Omega_i \times \Phi(p, pointControle_i)$$

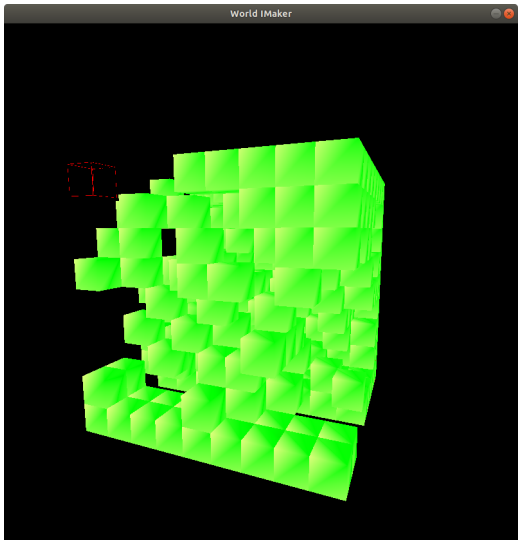
Avec

- Ω_i calculé grâce à la fonction *Eigen::PartialPivLu.solve*; nous avons choisi la méthode LU car la matrice *phi_mat* est symétrique de diagonale 1
- $\Phi(p, pointControle_i)$ la fonction radiale choisie appliquée à la distance cartésienne entre le point du cube p et le i^e point de contrôle.

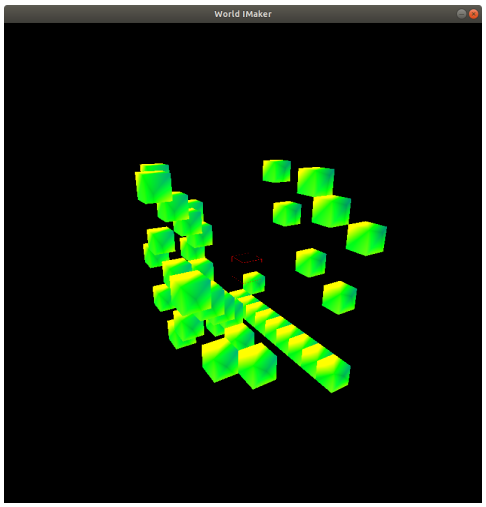
6. Choix de la condition d'affichage grâce à un prédicat de type lambda fonction.

Dans la classe Scène, la méthode *generate* met à jour l'affichage de ses cubes en fonction de la distance de leur valeur interpolée et de la valeur pivot.

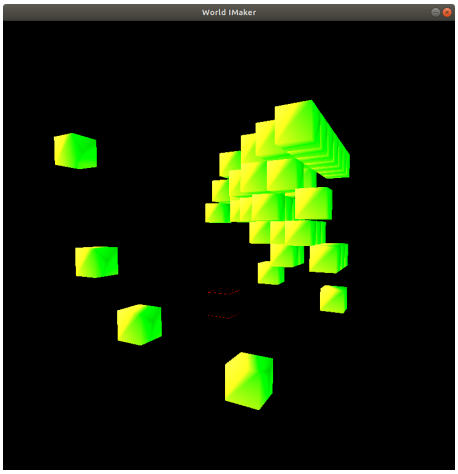
Quelques exemples de scènes générées



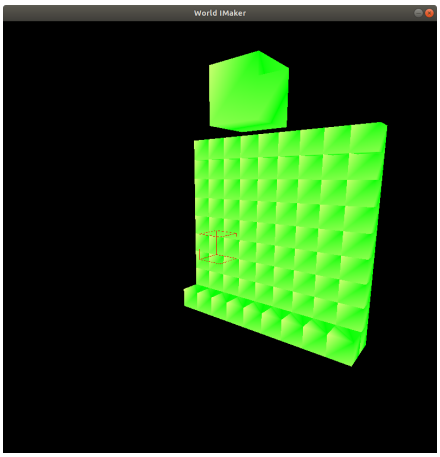
$\Phi(x,y) = \sqrt{1 + (\varepsilon \times |d(x,y)|^2)^2}$, $\varepsilon = 0,01$, points de contrôle au centre de la scène



$\Phi(x,y) = \frac{1}{1+(\varepsilon \times |d(x,y)|^2)^2}$, $\varepsilon = 0,01$, points de contrôle aux sommets de la scène



$\Phi(x,y) = \varepsilon \times |d(x,y)|^2$, $\varepsilon = 0,01$, points de contrôle aux sommets de la scène



$\Phi(x,y) = \sqrt{1 + (\varepsilon \times |d(x,y)|^2)^2},\ \varepsilon = 0,01$, points de contrôle au centre de la scène

Dans tous les cas, la condition d'affichage est que la valeur interpolée du cube doit être à ε près de la moyenne des valeurs de tous les cubes.

Les fonctions racines ont tendances à créer des blocs sur les bords tandis que les rationnelles sont plus éparses vers les points de contrôle. Les fonctions linéaires et exponentielles quant à elles produisent des résultats très différents d'un lancement à l'autre, certainement à cause des valeurs aléatoires.

Ajout des lumières

- Lumière directionnelle : l'utilisateur peut choisir d'ajouter une source de lumière directionnelle représentant le soleil en appuyant sur la touche ***S*** de son clavier.
- Lumière ponctuelle : l'utilisateur peut ajouter une source de lumière ponctuelle représentant un lampadaire en appuyant sur la touche ***L*** de son clavier. La source de lumière est placée sur le bloc sur lequel est située la sélection et supprime automatiquement le dernier lampadaire posé (s'il y en avait déjà un de posé).

Pour ces deux types de lumières, des variables uniformes sont envoyées au shader. C'est dans celui ci que la lumière est calculée grâce au modèle de Blinn-Phong, comme vu en TD.

Cette fonctionnalité n'est que partielle car la lumière ponctuelle ne se pose pas ponctuellement, au contraire de ce que son nom indique. De même, la lumière directionnelle est appliquée à chaque cube et pas au monde en lui même. Nous nous sommes rendu compte trop tard que c'était parce qu'elles étaient dans le shader appliqué

à chaque cube et pas à la scène. Chaque lumière posée est donc appliquée à chaque cube et pas globalement, d'où ce rendu très éloigné de ce que l'on imaginait.

Ces erreurs n'ont pas été résolues par manque de temps et pourraient être améliorées pour une version corrigée du projet.

Sauvegarde/Chargement d'une scène

- Utilisation des fonctions d'entrée/sortie

Nous avons créé un parseur permettant de lire un fichier de sauvegarde décrit ainsi :

lg 1 : **w** (largeur de la scène)

lg 2 : **h** (hauteur de la scène)

lg 3 : **d** (profondeur de la scène)

lg 4 : **X** (nombre de cubes affichés)

lg 5 : **x y z C** (abscisse ordonnée côte couleur)

lg 6 : **x y z C** (abscisse ordonnée côte couleur)

…

On réinitialise ensuite la scène avec les bonnes dimension en utilisant la méthode *resize* puis on met à jour la liste des cubes à afficher ainsi que leur couleur.

Déplacement d'un cube

Le déplacement d'un cube se fait en plusieurs étapes:

1. L'utilisateur sélectionne le cube à déplacer et appuie sur ***G***
2. Si le cube existe, il reste sélectionné pendant tout le déplacement et le curseur devient bleu pour montrer la nouvelle position possible.
3. Lorsque la destination est choisie, l'utilisateur appuie sur la touche ***Espace*** pour confirmer le déplacement. Le cube d'origine devient alors invisible et celui de destination s'affiche avec les couleurs de ce dernier.
4. L'utilisateur a le choix à tout moment d'annuler le déplacement en appuyant sur la touche ***Échap***, le curseur revient alors sur le cube d'origine qui n'a pas bougé.

Suppression de la scène

En appuyant sur ***D***, l'utilisateur peut supprimer tous les cubes de la scène. Pour cela, tous les cubes sont rendus invisibles et le curseur est replacé au centre de la scène.

DIFFICULTÉS RENCONTRÉES

L'analyse des fonctions radiales a été problématique car les fonctions ont une partie aléatoire qui empêche d'en déduire un résultat. De plus, nous voulions créer deux sortes d'initialisation pour les points de contrôle : une avec la liste des points fournie et l'autre avec leur nombre pour pouvoir choisir ces points de façon aléatoire par la suite. Par manque de temps, nous avons donc eu recours à la duplication de code pour gérer les deux initialisations.

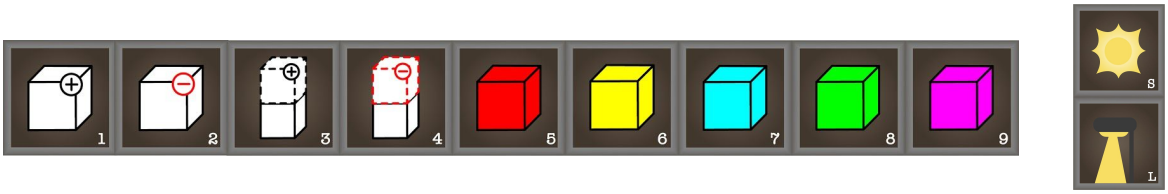
Les contours des cubes se sont toujours affichés étrangement en fonction de la position de la caméra. Selon le point de vue, les contours ne s'affichent que partiellement voire pas du tout et nous avons eu du mal à trouver comment faire apparaître les contours du curseur au premier plan constamment.

Quant à la lumière, nous nous sommes rendu compte trop tard qu'elle s'appliquait localement sur chaque cube au lieu d'éclairer la scène entière. Pour y remédier, nous aurions pu utiliser un deuxième jeu de shaders qui ne se serait occupé que de l'éclairage pour qu'il soit dissocié des cubes. D'autre part, les lumières ont été compliquées à gérer notamment à cause des normales des sommets du cube dont ne nous trouvions pas les valeurs. Le fait de n'avoir aucun exemple d'une lumière correcte n'a pas aidé non plus à nous rendre compte si ce que nous codions était bon ou non.

PISTES DE CONTINUATION

Nous avions prévu de faire une interface plus intuitive qui se placerait uniquement dans la fenêtre de visualisation (et non pas entre la fenêtre et le terminal comme c'est le cas maintenant) en utilisant les outils ImGui. Nous avions aussi pour idée de placer les bandeaux que vous pouvez trouver ci dessous en bas et à droite de l'écran comme aide pour l'utilisateur, mais nous avons rencontré des difficultés à gérer à la fois des objets 3D et ces textures que nous voulions appliquer sur des rectangles 2D, nous

avons donc laissé cette fonctionnalité moins indispensable pour plus tard.



Nous voulions également abstraire plus le code notamment pour avoir une classe dont la tâche serait de gérer les fonctions d'OpenGL. Nous avons essayé avec la classe *Interface* mais la gestion des buffers et arrays nous a posé problème donc nous avons laissé ces fonctions dans le main.

Nous aurions enfin aimé résoudre les différents problèmes liés aux lumières et permettre l'ajout de plusieurs lampadaire pour une scène un peu plus complète.

EXEMPLES DE COMMANDES

La commande servant à lancer l'exécution de notre application est *./src/worldimaker*. Voici quelques exemples d'expériences utilisateur.

- Toto souhaite arriver sur une fenêtre vierge pour créer une scène de toutes pièces. Il entre pour cela la commande *./src/worldimaker*, puis la question "Do you want to load a scene ?" lui est posée. Il répond "n" à cette question, puis également à la suivante qui est "Do you want to generate a scene ?". Il entre ensuite les dimensions de la scène qu'il souhaite créer et une scène avec une base de 3 cubes sur le sol du monde comme demandé.
- Cette fois ci, Toto voudrait charger une scène qu'il a créée et enregistrée précédemment. Pour cela, il répondra "y" à la première question posée dans le terminal et entrera le chemin du fichier sauvegardé. Par exemple, il lui suffira d'écrire "sqrt_centered_compact" pour ouvrir la scène du même nom. Il pourra ouvrir ainsi toutes les scènes enregistrées dans le dossier *code/saving*.
- Toto manque d'inspiration, il aimerait avoir quelques idées. Il va pour cela répondre "y" à la question "Do you want to generate a scene ?". Il pourra tester différentes RBF, avec par exemple les commandes "config1_exp" ou "config5_lin_max". Il peut retrouver toutes ces possibilités dans le dossier *code/config*.

RETOURS INDIVIDUELS

Ce projet était très satisfaisant à réaliser. Le sujet était clair et suffisamment détaillé pour nous donner une ligne directive. Les outils des bibliothèques STL et glm ont rendu l'implémentation très facile. Avec un peu plus de temps, nous aurions assurément poursuivi le développement pour améliorer notre programme.

Louisa CHIKAR

J'ai bien aimé travailler sur ce projet. Les consignes étaient claires et le sujet très précis, ce qui nous a permis de comprendre rapidement ce que nous avions à réaliser. Je trouve intéressant de travailler avec OpenGL et SDL car nous avons vite des résultats visuels qui nous permettent de savoir où on en est et ce qu'il nous reste à faire. J'aurais aimé avoir plus de temps pour travailler sur ce projet car avec les autres projets et les partiels à travailler en parallèle j'ai été un peu frustrée de ne pas avoir pu travailler plus dessus pour le mener à bien. Dans tous les cas c'est un projet que j'ai aimé développer et qui m'a permis d'en apprendre plus en synthèse d'images plus particulièrement.

Lucie LESBATS

CONCLUSION

Pour conclure, nous sommes assez satisfaites de ce que nous avons produit. Notre projet n'est pas complet mais nous avons réussi à avoir un résultat fonctionnel et qui remplit la majorité des consignes. Nous avons toutes deux beaucoup appris grâce à ce projet et progressé en algorithmique et en synthèse d'images.

LIEN REPO GIT

https://github.com/Lucie98765/World_IMaker