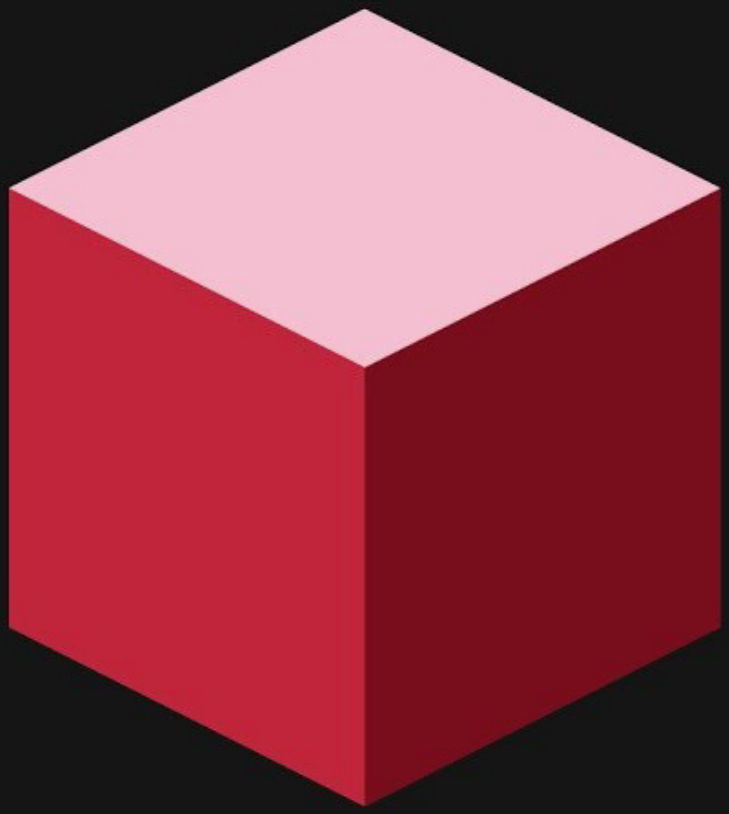
Rapport projet de C/openGL

Création d'un IMAKERWORLD



Roxane Vallée
Sarah Veysset

IMAC 2 - 2019/2020

Introduction

Dans le cadre de nos études en **école d'ingénieur IMAC** (Image, Multimédia, Audiovisuel, Communication), nous avons été amenées à **réaliser** un **éditeur-visualiseur de terrain** et de **scène en 3D.**.

Le but est d'**apprendre à gérer un projet en équipe**, de la reformulation du cahier des charges jusqu'à sa réalisation complète en **respectant les exigences** du commanditaire. Vous verrez dans ce projet une application directe des compétences acquises au cours de notre cursus scolaire notamment en ce qui concerne les **langages C++** et **OpenGL.**

Le projet s'est déroulé en trois étapes :

- La **compréhension du sujet** qui consiste à étudier le cahier des charges fonctionnel et à le reformuler.

- La **conception et la réalisation** du projet : nous évoquerons dans cette partie le **choix des outils et des langages utilisés** pour la réalisation du site et du développement proprement dit (le codage). Nous parlerons aussi des **résultats obtenus, des problèmes rencontrés** lors de la conception et enfin nous proposerons quelques idées **d'amélioration du logiciel.**

- Le **planning** : celui-ci nous a permis de mesurer l'avancement de nos travaux et l'atteinte de nos objectifs.

Présentation de l'application

Brief du projet

Toto aimerait bien créer un joli **monde en 3D** dans lequel il pourrait **naviguer**, afin de pouvoir le présenter lors de son entretien de recrutement pour une boîte de dessin-animés. Malheureusement, il ne possède **aucune compétence** pour le réaliser. C'est pourquoi en tant que très bon ami, nous nous proposons de **l'aider** dans cette tâche.

Synopsis

Toto nous a ensuite expliqué tout le **projet en détail** :

L'application que souhaite Toto consiste donc en un **éditeur de terrains 3D,** où le paysage est représenté uniquement par des **cubes**. La scène est donc un **énorme pavé** représentant le **monde constitué de WxLxH cubes** unitaires dont certains sont "vides" et d'autres sont des cubes de **différentes couleurs**. L'utilisateur devra non seulement pouvoir **se déplacer** dans la scène, mais également **la modifier.**

Travail accompli

	Fait	Pas fait	Fait mais ne marche pas
Réalisation d'une structure ordonnée (partie algorithmique)	X		
Makefile (partie algorithmique)	X		
Affichage d'une scène avec des cubes de couleurs unies	X		
Création d'une caméra (partie infographie)	X		
Création d'un curseur visible (partie algorithimie & infographie)	X		
Création d'un bouton pour modifier la couleur des cubes (partie infographie)	X		
Ajouter un bloc	X		
Supprimer un bloc	X		
Ajouter des cubes du haut de la colonne	X		
Supprimer des cubes du haut de la colonne	X		
Système de génération procédurale	X		

Travail accompli

	Fait	Pas fait
Lumière directionnelle	X	
Point de lumière	X	
Amélioration de la séléction		X
Outils de painting		X
Sculpting ++		X
Sauvegarde / chargement de la scène	X	
Chargement de modèles 3D		X
Niveau de discrétisation		X
Blocs texturés	X	
Bouton pour supprimer tous les cubes	X	

Fonctionnement du logiciel

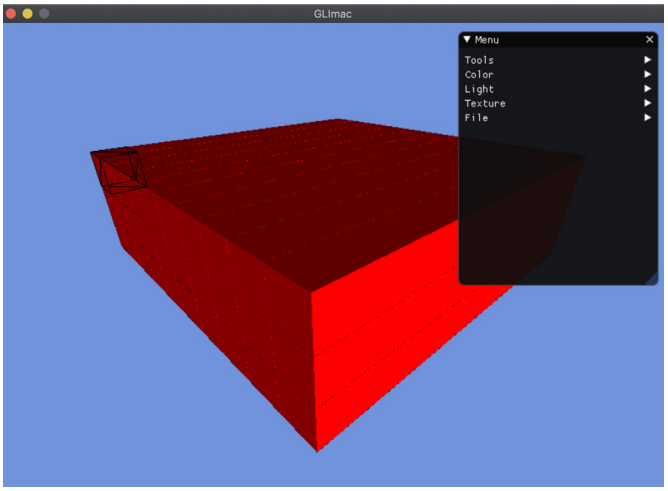
Description détaillée des fonctionnalités

Démarrage de l'éditeur

Pour démarrer le l'éditeur de terrain, l'utilisateur doit **ouvrir un terminal** depuis le dossier «build». Il entre ensuite la **commande « cmake ../Imac_World/. »** puis **« make »** et enfin le chemin : **« ./src/src_main ».**

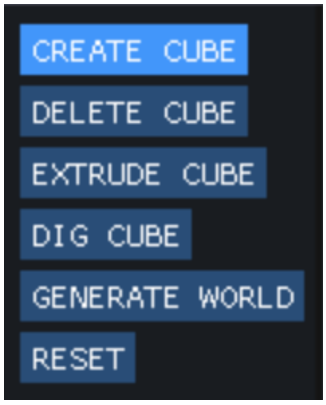
Menu principal

Lorsque l'éditeur se lance, la première image qui apparaît est **une couche de 3 cubes** de la même couleur. Sur la droite, se trouve le **menu principal** avec les **différents outils** permettant de **modifier la scène.**



Utilisation des outils

Dans le sous-menu **«Tools»** on retrouve les outils suivants :



- **CREATE CUBE :** en appuyant sur ce bouton, un **cube est créé**.
- **DELETE CUBE :** en appuyant sur ce bouton, un **cube est supprimé**.
- **EXTRUDE CUBE :** en appuyant sur ce bouton, un **cube est créé en haut de la «colonne».**
- **DIG CUBE :** en appuyant sur ce bouton, un **cube est supprimé en haut de la «colonne».**
- **GENERATE WORLD :** en appuyant sur ce bouton, **un monde déjà défini est créé.**
- **RESET :** en appuyant sur ce bouton, **tous les cubes sont supprimés.**

Dans le sous-menu **«Color»** on retrouve un **Color Edit** permettant de **modifier la couleur** d'un cube.

Dans le sous-menu **«Light»** on retrouve les outils suivants :

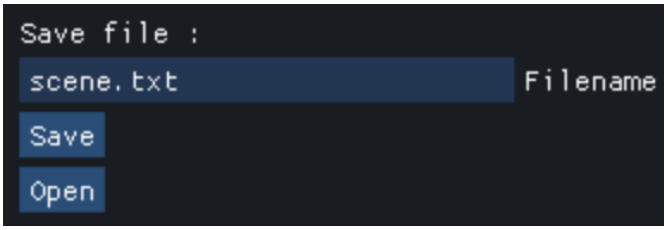
- **DAY :** en appuyant sur ce bouton, la **lumière directionnelle est activée.**
- **NIGHT :** en appuyant sur ce bouton, le **point de lumière est activé.**

Fonctionnement du logiciel

Dans le sous-menu **«Texture»** on retrouve les outils suivants :

- FLOWER : en appuyant sur ce bouton, le **cube devient texturé.**
- COLOR : en appuyant sur ce bouton, le **cube devient coloré.**

Dans le sous-menu **«File»** on retrouve les outils suivants :



- SAVE : en appuyant sur ce bouton, **la scène est sauvegardé.**
- OPEN : en appuyant sur ce bouton, **la scène choisi est ouverte.**

Déplacement dans la scène

Les touches **«zqsd»** permettent de se déplacer dans la scène respectivement **en avant, à gauche, en arrière, à droite.** Les touches **«wx»** permettent de se déplacer respectivement **en haut et en bas.**

La **souris** permet de déplacer la **vue de la caméra.**

Déplacement du curseur

Les **flèches directionnelles** permettent de **déplacer le curseur** dans l'espace **à gauche, à droite, en haut et en bas.** Les touhes du clavier **«lm»** permettent de déplacer le curseur sur l'axe z, respectivement **en avant et en arrière.**

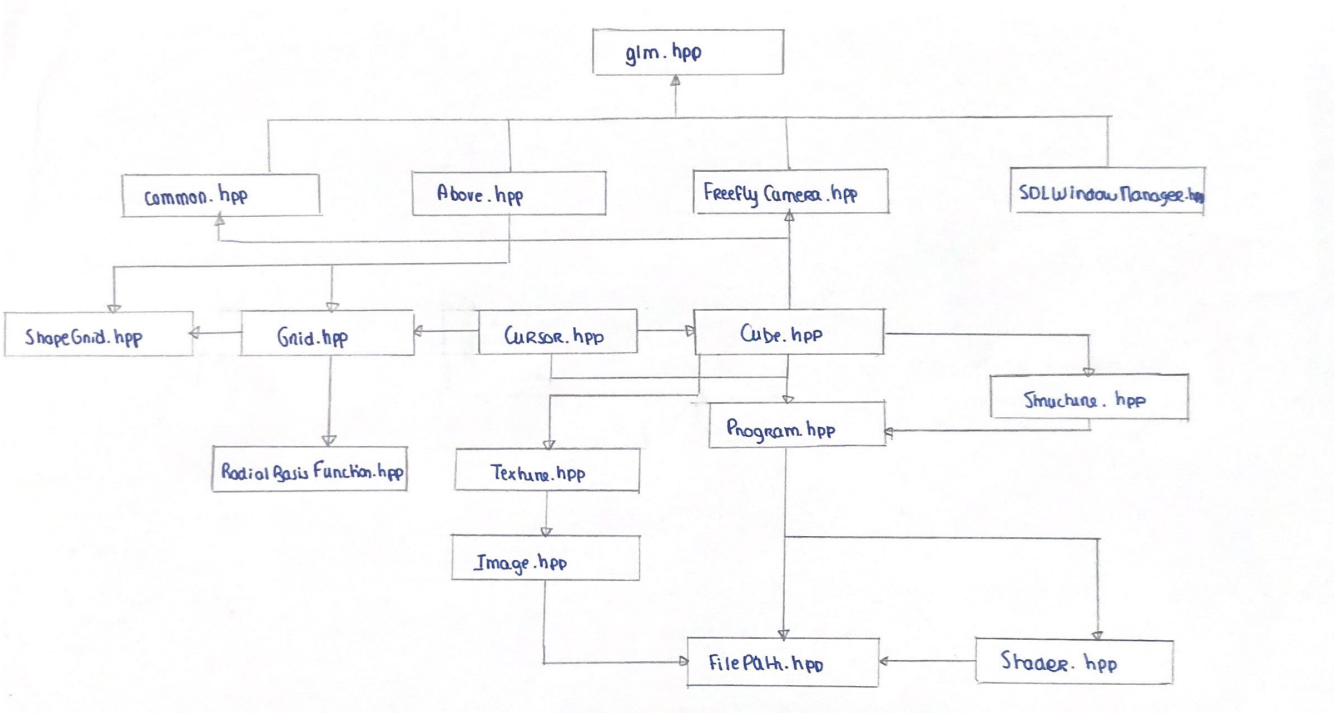
Architecture de l'application

Découpage de notre application

Pour **découper** au mieux notre application, nous avons décidé de **répartir notre code** en plusieurs fichiers .cpp et .hpp. Chaque fichier **constitue un élément ou une fonctionnalité** avec les méthodes qui lui sont attribuées.

Par **exemple,** dans notre **fichier Grid** (.cpp et .hpp) nous avons la **création de cube,** la **suppression de cube**, et les **différentes actions** concernant les cases de notre grille. D'autres fichiers comme **ShapeGrid** permettent de **créer la structure** de notre code et comment nous allons **ranger les différentes informations**. Nous avons également des fichiers comme **Cube**, qui gèrent toute la **partie Interface et OpenGL.**

Voici un petit **schéma** pour mieux comprendre :



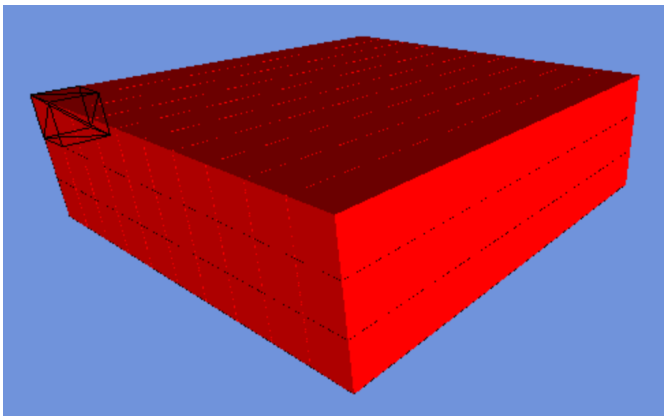
Détail des fonctionnalitées

Pour ce projet, nous avons implémenté **différentes fonctionnalitées** que nous allons vous décrire

Affichage d'une scène de cubes

Composition d'une grille

Afin de pouvoir placer et créer différents cubes dans notre scène, nous avons **décomposé notre scène en une grille** (Grid.hpp). Dans cette grille, chaque **case peut être soit vide soit contenir un cube**. Si notre case est remplie, elle sera insérée dans un vecteur contenant tous nos cubes.



Un cube possède plusieurs informations :
- **ses coordonnées**
- **son type (si il possède une texture ou non)**
- **sa couleur**

A l'initialisation de notre scène, notre grille sera composée d'une **couche de 3 cubes,** représentant le sol de notre monde.

Dessin d'un cube

Pour représenter les cubes dans notre scène, nous avons décidé de créer **le même cube à une position unique** et de lui appliquer une **translation en fonction des coordonnées de notre grille.**

Le dessin du cube se fait en plusieurs parties :
- nous définissons les **différents sommets de notre cube**
- à partir des **sommets nous dessinons chaque face**
- le tout est **stocké dans un vecteur**
- nous l'affichons ensuite dans notre scène

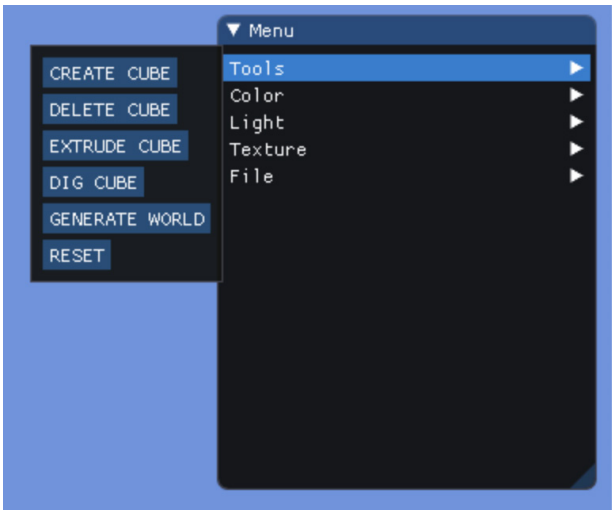
Caméra

Nous avons privilégié une caméra Freefly plutôt qu'une caméra Trackball car nous voulions nous **déplacer librement** dans la scène.

Détail des fonctionnalitées

Edition des cubes

Création des boutons



Pour créer un menu élégant et intiuitif nous avons décidé d'utiliser la **librairie ImGui.** Elle permet d'afficher des fenêtres simplement. Ainsi, pour chaque **bouton** cliqué une fonction est appelé **générant le changement voulu.**

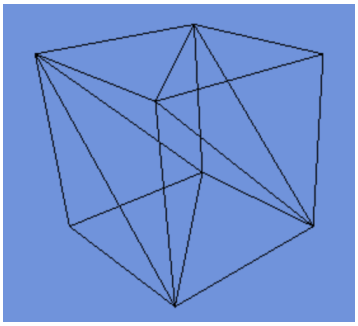
Sélection de cubes et création curseur

Afin de pouvoir se déplacer dans notre scène, nous **créons un curseur**. Le curseur possède des **coordonnées pour se déplacer dans notre grille** et **vérifier** pour chaque **case si elle est vide ou non.**

Pour **sélectionner** un cube, nous **vérifions** à partir des coordonnées de notre curseur, **si la case est déjà remplie par un cube**. Une fois cette condition vérifiée nous pouvons **changer la couleur** de notre objet ou sa **texture.**

Afin de visualiser la sélection, nous créons un cube avec **uniquement les arrêtes visibles** à la position du curseur. Nous nous déplaçons le curseur à partir du clavier de notre ordinateur.

Dessin du curseur



Notre curseur est un cube passé en **mode «wireframe».** C'est pourquoi nous avons créé un **héritage de notre classe Cube.** Ainsi, cela nous permet d'utiliser les méthodes et les attributs de notre cube sans avoir à tout recréer. Nous avons simplement réalisé une **surcharge permettant d'ajouter la position** de notre curseur en paramètres.

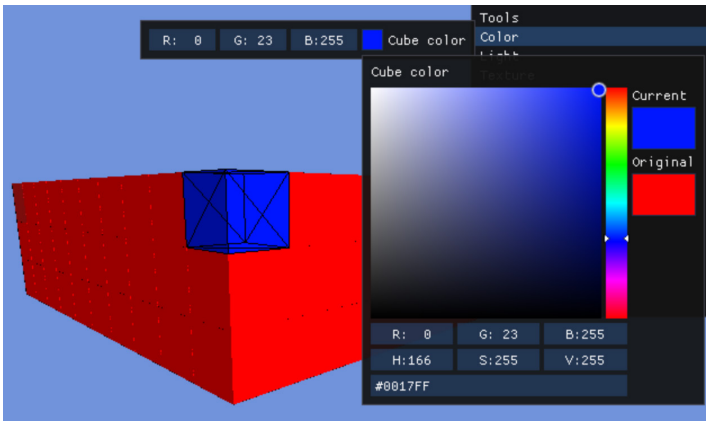
Détail des fonctionnalitées

Le passage en mode «wireframe» s'effectue grâce aux lignes suivantes :

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glDisable(GL_DEPTH_TEST);
    glDrawArrays(GL_TRIANGLES, 0, getVertexCount());
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

Modification de la couleur

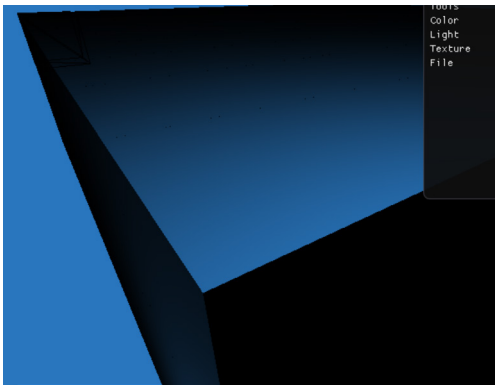
Dans notre interface, nous pouvons choisir une couleur parmi un **nuancier**. Nous récupérons les données dans imgui afin de modifier la couleur de **notre case sélectionnée.**



Création des lumières

Nous avons décidé de placer le calcul de la lumière directionnelle et de notre point de lumière dans **le même fragment shader**. Ainsi c'est soit l'un soit l'autre qui est calculé en fonction du bouton cliqué («DAY» ou «NIGHT»).

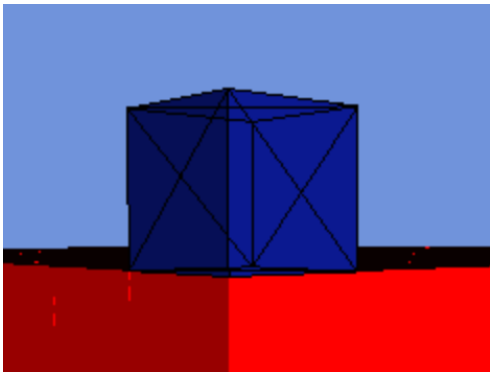
Ces deux lumières sont dirigées en fonction de la **caméra.** Ainsi, le point de lumière agit comme une **lampe torche** sur notre scène.



Détail des fonctionnalitées

Sculpture du terrain

Ajout/suppression cube

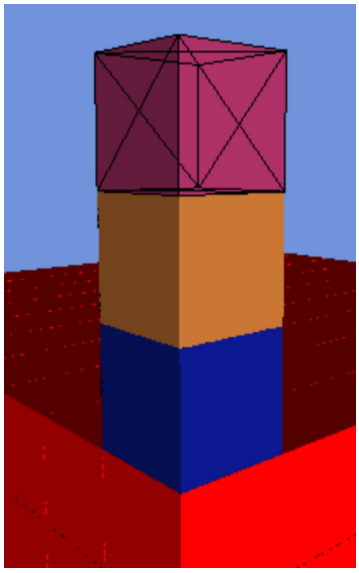


Nous pouvons **ajouter ou supprimer un cube** à partir de la **position du curseur.** Pour ajouter un cube, nous vérifions que la **case est vide** (sans cube) et nous créons un **cube à partir des coordonnées du curseur.** Le **vecteur** de notre grille **augmentera alors d'une case** et **stockera les informations** de notre nouveau cube.

Nous avons utilisé **deux types de créations de cubes** : soit on connaît **le type et la couleur** soit ils seront automatiquement mis par défaut. La **création de cubes** à partir du type et de la couleur nous **servira** pour **charger une sauvegarde.**

Pour **supprimer**, le principe est le même. Nous sélectionnons un cube, après vérification que la case n'est pas vide, et supprimons le cube en supprimant la case correspondant aux coordonnées du vecteur.

Extrude et Dig



Pour **extrude ou dig** au niveau du curseur, nous nous positionnons toujours sur le cube le plus haut de la colonne. Nous **vérifions** en amont que le **curseur se trouve sur une case contenant un cube.**
Ainsi, pour extrude, nous **rajoutons** un cube **en haut de la colonne** de notre curseur, et nous supprimons le cube en haut de la colonne pour dig.
Nous **mettons le vecteur de notre grille à jour** pour chaque changement effectué au niveau des cubes.

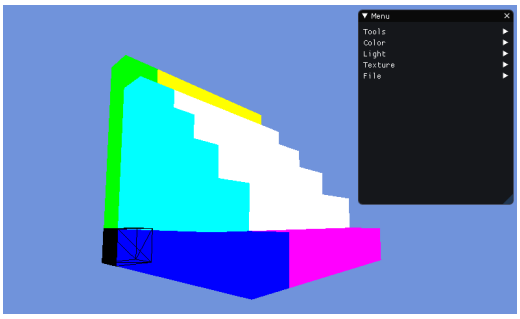
Détail des fonctionnalitées

Radial Basis Functions

Afin de générer une scène nous utilisons les **« radial basis functions »**.
Pour cela nous avons utilisé plusieurs fonctions radiales :

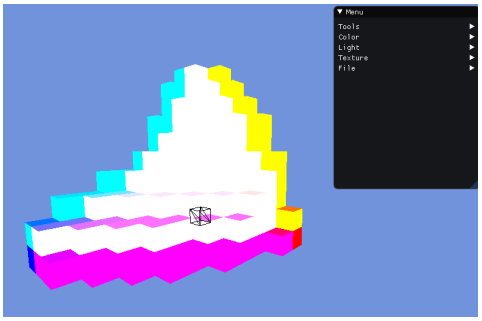
- la fonction **Gaussian** :

$$\varphi(r) = e^{-(\varepsilon r)^2}$$



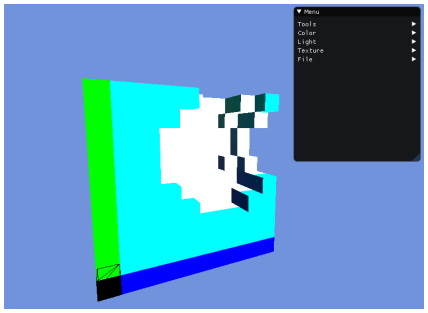
- la fonction **Multiquadratic** :

$$\varphi(r) = \sqrt{1 + (\varepsilon r)^2}$$



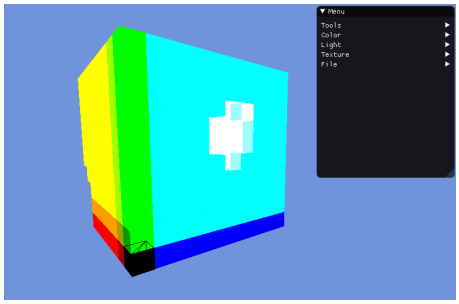
- la fonction **Inverse quadratic** :

$$\varphi(r) = \frac{1}{1 + (\varepsilon r)^2}$$



- la fonction **Inverse multiquadratic** :

$$\varphi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$



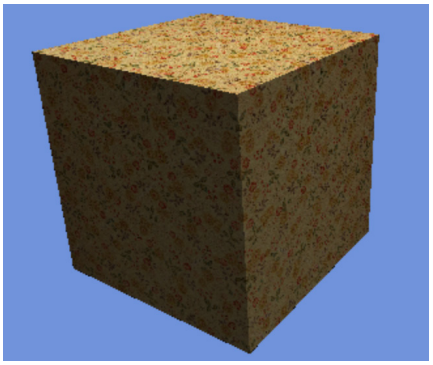
Détail des fonctionnalitées

Nous **positionnons** dans notre scène **certaines valeurs à des cases choisies** (nous en avons sélectionné 3). Si la **valeur est supérieur à 0**, il faut **dessiner un cube**, sinon, la **case reste vide.**
A partir de **ces points de contrôle**, nous allons pouvoir **définir une valeur pour chaque case** en **passant par les fonctions radiales.**

Mais avant de pouvoir définir la valeur de chaque case de notre scène, nous devons **trouver la valeur omega**. Omega est le résultat de la multiplication de la Matrice Inverse M (M, matrice créée à partir des fonctions radiales) et du vecteur contenant la valeur des points de contrôle.

Détail des fonctionnalitées additionnelles

Modification de la texture



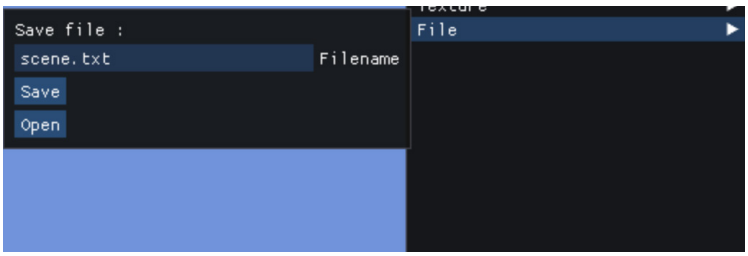
Pour **créer des cubes texturés**, nous avons utilisé une **fonction template.** C'est pourquoi notre **fonction** drawCube() se situe **dans le fichier cube.hpp**. En effet, notre fonction **a besoin du programme de shaders** correspondant, soit le coloré soit le texturé. C'est donc cette solution qui a été privilégié.

Sauvegarde / lecture scène

Nous avons rajouté la fonctionnalité de **charger et sauvegarder une scène** à partir d'un fichier.

Pour enregistrer, nous **choisissons le nom de notre sauvegarde** dans notre interface et nous créons un fichier à son nom. Dans ce fichier nous **écrivons les informations de chaque case contenant un cube**, c'est-à-dire les **valeurs de notre vecteur dans notre Grid.** Ainsi le fichier comprendra les **coordonnées des cubes**, le **type** et la **couleur**.

Dans le sens inverse, pour **lire une sauvegarde**, nous devons **écrire son nom** dans notre interface (fenêtre imgui). Après **vérification de l'existence de ce fichier**, nous pouvons **lire ses données** et les **stocker dans notre vecteur Grid** qui comprend tous les cubes de notre grille. Nous utilisons la fonction **création cube** prenant en compte la **couleur** et le **type**.



Nos difficultés et solutions

1. Dessin du cube

Le premier problème était de **dessiner un cube**. Il était assez long et compliqué de rentrer à la main chaque sommet de notre cube. Notre code se retrouvait avec plusieurs lignes illisibles.

Solution : nous avons **dessiné le cube face par face**, en rangeant chaque face dans une fonction différente et les reliant dans le constructeur de cube.

2. L'algorithme de la génération Procédurale

Le deuxième problème était de **rédiger l'algorithme pour les radial basis functions.** Nous avons eu des soucis pour **relier nos fonctions des radial basis functions avec notre grille** comprenant notre cube.

Solution : Comme solution, nous avons rédigé chaque étape dans des **fonctions différentes** pour une **meilleure lisibilité** de notre code. Nous avons également créé une fonction dans notre fichier Grid qui permet de faire le lien entre nos radial basis functions et la création de cubes. Enfin, nous avons **vérifier la suppression de notre scène actuelle** avant de générer une nouvelle scène.

3. Problème pour la sauvegarde et la lecture fichier

Lors de la rédaction de l'algorithme pour sauvegarder les scènes, nous avons créer un nouveau fichier .cpp contenant nos fonctions. Malheureusement nous nous sommes rendus compte que notre algorithme utilisait directement des fonctions présentes dans notre fichier Grid. Le **code ne compilait pas** si il se trouvait en dehors de ce fichier.

Solution : Nous avons donc décidé de **déplacer cette fonction** sauvegarde et lecture directement dans notre classe grille afin d'avoir un accès direct aux attributs et méthodes. Également, ces fonctions lectures et sauvegarde ont subi des changements avec l'ajout de la texture et sa sauvegarde dans le fichier.

Pour aller plus loin

Les améliorations

Du côté des améliorations, elles sont **quasiment infinies.** Nous pouvons **améliorer la sélection**, **réaliser différents outils de painting** ou **de scultping ++, charger des modèles 3D** ou encore **ajouter du son.**

Notre conclusion individuelle

Roxane : Avant de commencer le projet, avec Sarah, nous avons essayé de **terminer les Tps de synthèse d'images** afin d'avoir tous les éléments nécessaires pour coder. **J'appréhendais** un peu **le temps de travail**, nous avons donc décidé de commencer le projet un peu en avance et de **se répartir les tâches** pour travailler chacune de notre côté pendant les vacances de Noël. C'était assez **compliqué de travailler à distance**, heureusement, nous avons codé les fonctionnalités de base ensemble, avant les vacances. J'ai beaucoup apprécié de voir le résultat de la fenêtre au fur et à mesure que nous rajoutons des fonctionnalités. J'ai bien aimé **coder la structure de base** pour gérer notre scène mais j'ai eu plus de mal à coder et comprendre la génération procédurale. Nous avons eu quelques problèmes mais la plupart des difficultés ont été résolues facilement, nous permettant de continuer d'avancer dans notre code.

Sarah : Comme l'a dit Roxane, il a fallu **terminer les TPs de synthèse d'image** afin de débuter le projet concrètement ce qui a eu pour conséquence de **retarder le lancement du projet.** Hormis les très nombresues difficultées pour ma part, j'ai pris **plaisir à coder** et à **apprendre de nouvelles choses**. Personnellement, je regrette le **manque d'élèments graphique** à créer comme nous avions pu avoir pour le projet de l'an passé (IMAC TOWER DEFENSE).

Conclusion générale

Malgré les difficultés que nous avons rencontrées nous sommes **satisfaites du résultat.** Nous avons essayé de **remplir toutes les contraintes du projet** tout en allant plus loin en **proposant d'autres fonctionnalités**.

Il nous a aussi permis de **mettre en application toutes les connaissances apprises** au cours de cette première année au sein de l'IMAC.

Remerciements

Nous tenons particulièrement à **remercier nos professeurs** qui nous ont permis de **mener à bien ce projet** de part **leur disponibilité** et **leurs précieux conseils.**