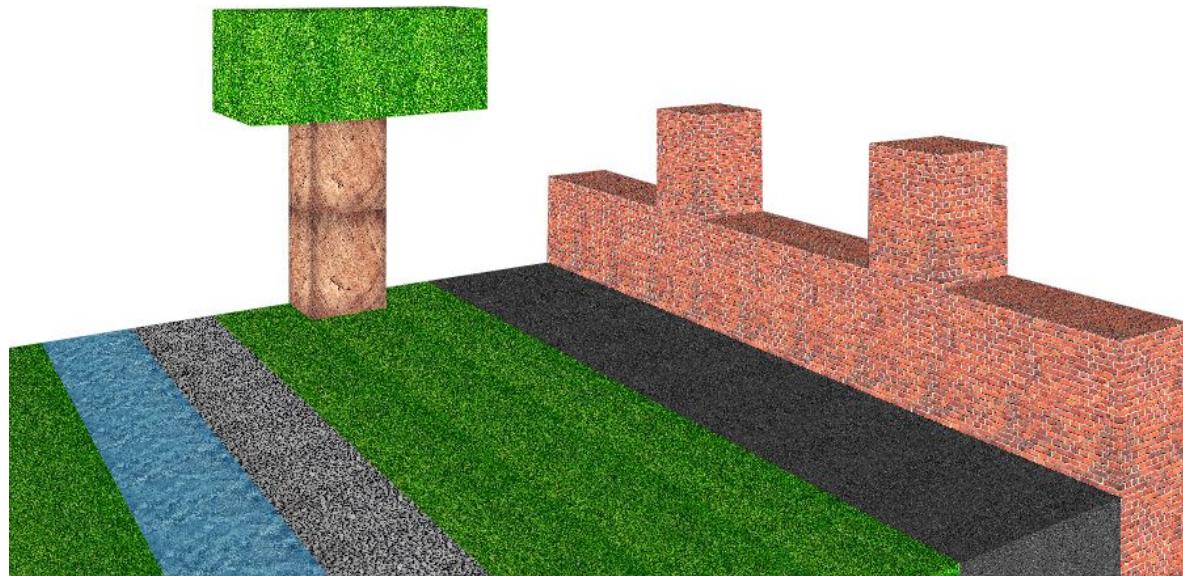


World IMaker

MANSION Amélia & SGRO' Manon



Sommaire

Présentation du projet	3
Diagramme de classes	4
Choix d'architecture logicielle	5
Fonctionnement de l'application	7
Compilation	7
Commandes du jeu	7
Organisation des fichiers - Arborescence	9
Synthèse (tableau récapitulatif)	10
Fonctionnalités implémentées	12
Affichage de la scène avec des cubes	12
Edition des cubes & Sculpture du terrain	12
Génération procédurale	12
Observations	12
Ajout de lumières	18
Sauvegarde / chargement de la scène	18
Chargement de modèles 3D	18
Blocs texturés	19
Résultats obtenus	20
Génération procédurale	20
Ajout de lumières	21
Sauvegarde/Chargement de fichiers	22
Fonctions d'édition des blocs	23
Chargement de modèles 3D	24
Difficultés rencontrées	26
Note personnelle	27
MANSION Amélia	27
SGRO' Manon	27

Présentation du projet

Ce projet consiste en la réalisation d'un éditeur-visualiseur de terrain et de scène en 3D. Un certain nombre de fonctionnalités décrites dans le sujet du projet ont été implémentées, afin de permettre à l'utilisateur de créer un monde à partir de blocs cubiques et de naviguer dedans.

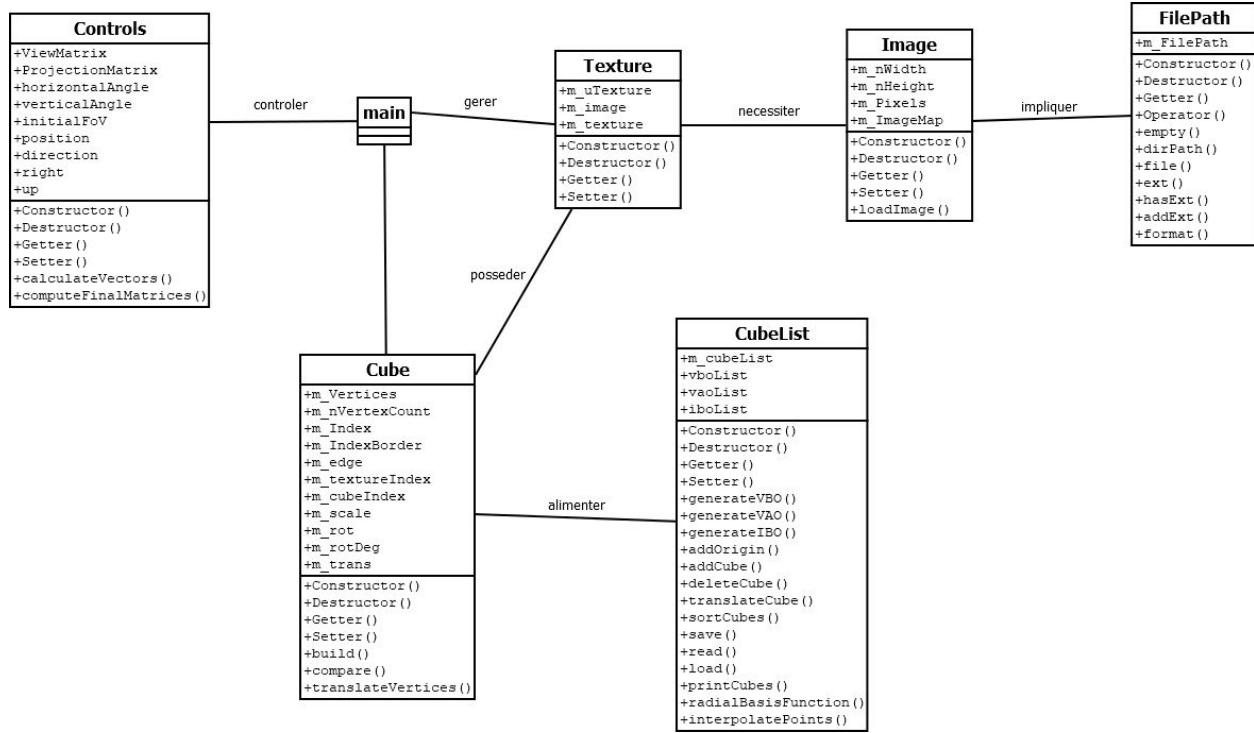
Celui-ci a été réalisé dans le cadre de la deuxième année à l'IMAC dans le cadre des cours :

- Programmation Orientée Objet (C++)
- Synthèse d'images (OpenGL)
- Mathématiques

Ce devoir a été réalisé par le binôme composé de :

- MANSION Amélia
- SGRO' Manon

Diagramme de classes



Il est important de préciser que ce diagramme est l'aboutissement de nombreuses réflexions et a été amené à évoluer. En effet, la structure générale des classes est identique à celle définie en amont du projet. Cependant, lors de la phase de réalisation, nous avons pris soin d'ajouter des méthodes et attributs à ce diagramme, afin de pouvoir obtenir une vue globale de la structure du projet à tout moment.

Choix d'architecture logicielle

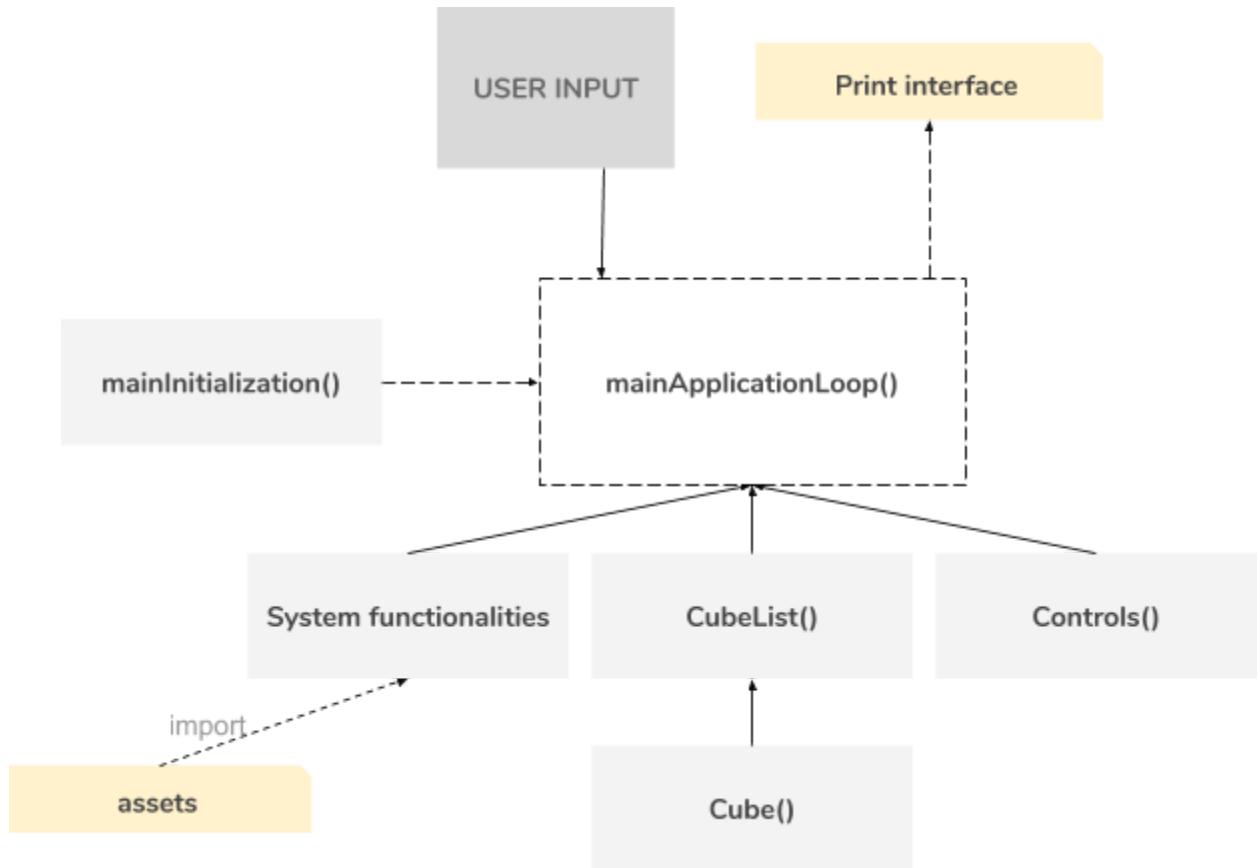
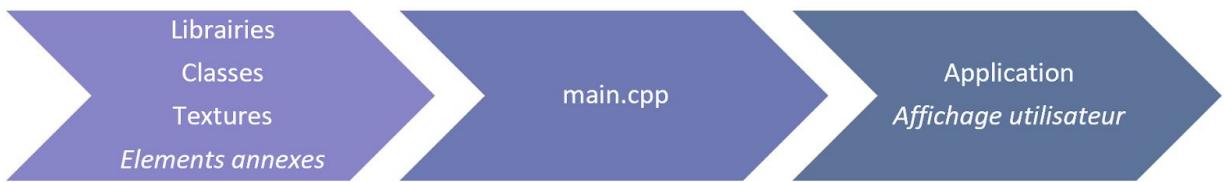
Lorsque nous avons débuté ce projet, nous avons travaillé à partir d'exemples vus en cours. Afin de concevoir l'architecture logicielle de notre projet final, deux possibilités s'offraient donc à nous : utiliser la structure déjà existante ou en créer une nouvelle. Par soucis d'efficacité, nous avons fait le choix d'utiliser le modèle qui nous était fournie durant les TD de Synthèse d'Images. Cette organisation de fichiers et de classes convenait bien à la réalisation de notre projet puisqu'elle nous permettait de la prendre en main rapidement, d'ajouter des libraires ou éléments plus ou moins facilement, et qu'elle comprenait déjà un certain nombre de fichiers utiles à la gestion des fenêtres et des scènes 3D.

Nous avons construit notre projet sur la base de cette structure, en répondant aux spécificités demandées :

- Le fichier principal gère l'initialisation du projet, ainsi que la gestion des événements et de l'affichage de la scène.
- Notre première réflexion a été de déléguer les fonctions de gestion des cubes à une classe *Cube* qui gérerait les informations nécessaire à l'affichage et à l'édition d'un cube.
- Nous avons ensuite pensé à centraliser la gestion de l'ensemble des cubes à une classe *CubeList*, qui gérerait le stockage et le parcours des cubes, afin que notre fichier principal ne s'occupe plus que d'appeler cette liste de cube lorsque l'on souhaiterait l'afficher et la modifier.
- Nous avons également séparé la gestion de la caméra qui gère le calcul des matrices à partir des données envoyées par l'utilisateur.

Au fur et à mesure du projet, nous avons ajouté et modifié des classes afin de déléguer des fonctionnalités auxquelles nous n'avions pas pensé au départ, mais cette structure nous a permis de clarifier les principales fonctionnalités que nous devions mettre en place.

Schéma simplifié de l'architecture :



Fonctionnement de l'application

Compilation

Afin de pouvoir créer et lancer un exécutable de l'application World Imaker, il suffit de se placer dans le dossier build, puis de taper les lignes de commandes suivantes :

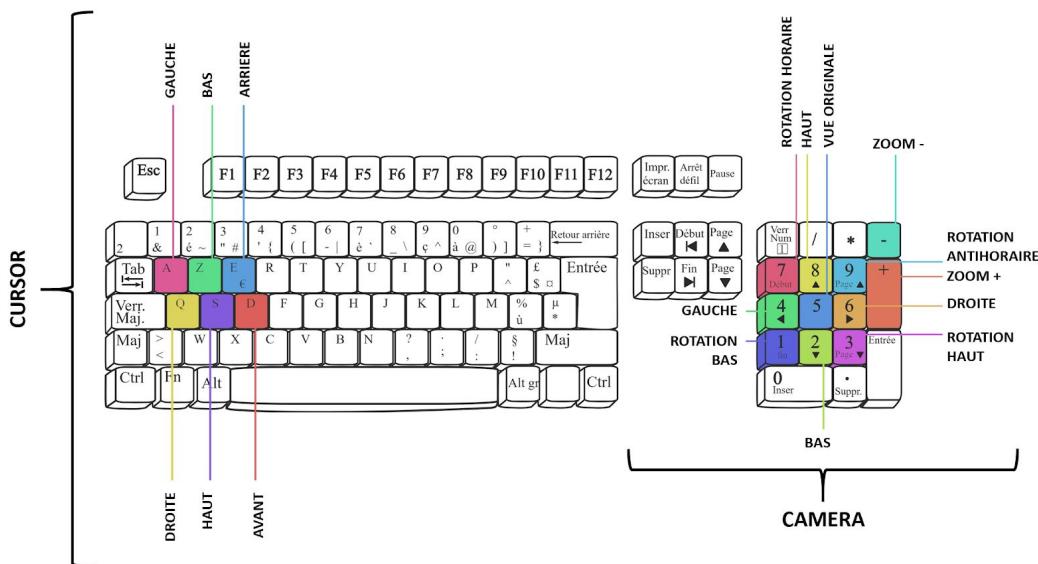
```
cmake ..
```

```
make
```

```
./bin/World_Imaker
```

Les étapes d'installation et de compilation sont expliquées plus en détail dans le fichier README du projet.

Commandes du jeu



Contrôles caméra

Utiliser le clavier numérique pour déplacer la caméra :

+/- : Zoom

8/2 : Haut/Bas

4/6 : Gauche/Droite

1/3 : Rotation Bas/Haut

7/9 : Rotation Gauche/Droite

5 : Réinitialisation sur l'origine

Contrôles curseur

Utiliser le menu CURSOR ou le clavier pour déplacer le curseur dans la scène :

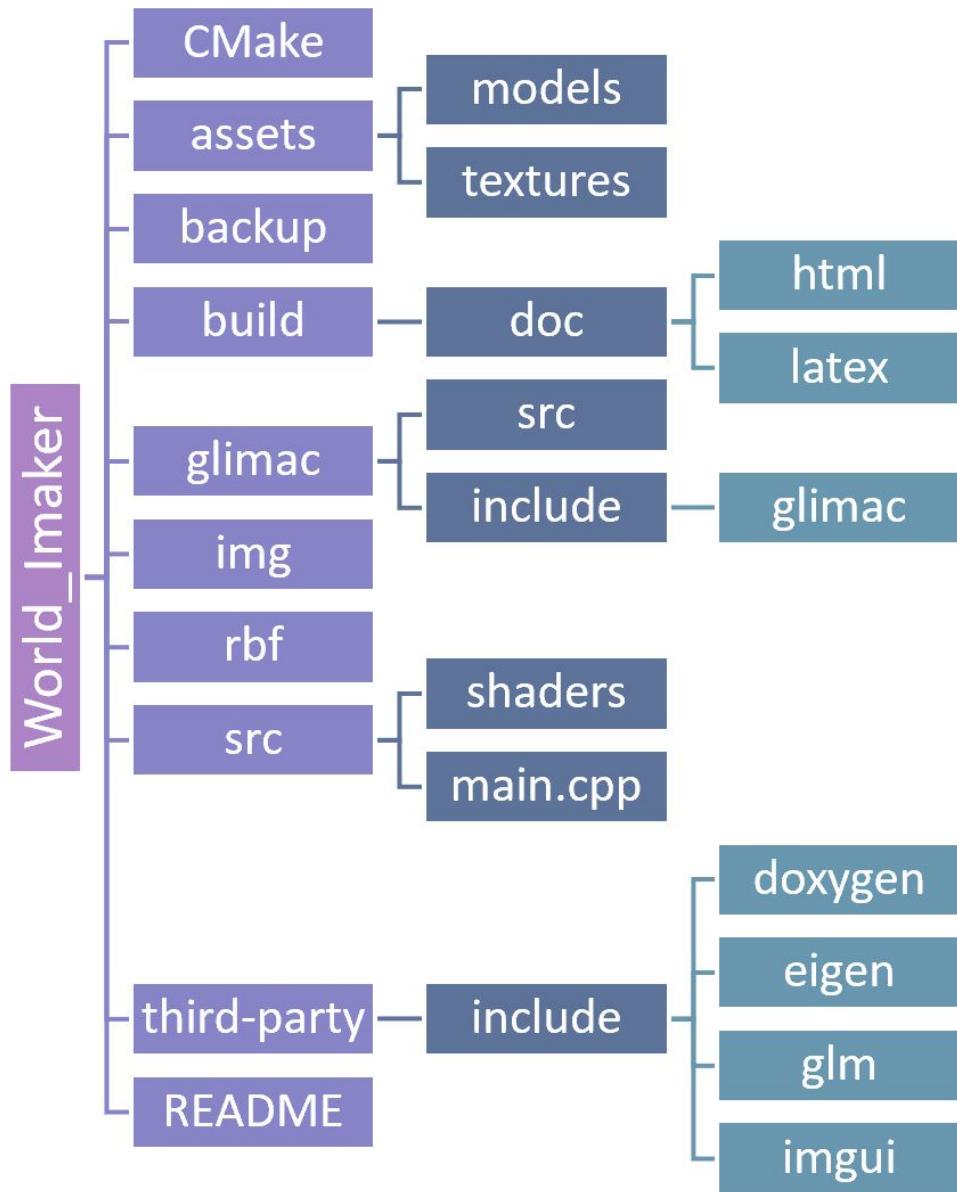
A/Q : Gauche/Droite

Z/S : Haut/Bas

E/D : Avant/Arrière

Organisation des fichiers - Arborescence

Voici l'arborescence des divers fichiers permettant le bon fonctionnement de l'application :



Synthèse (tableau récapitulatif)

Fonctionnalités	Implémentée	Implémentée partiellement	Non réalisée
Fonctionnalités requises			
Affichage de la scène avec des cubes			
Edition des cubes			
Sculpture du terrain			
Génération procédurale			
Ajout de lumières			
Fonctionnalités additionnelles			
Amélioration de la sélection			
Outils de painting			
Sculpting ++			
Sauvegarde / chargement de la scène			
Chargement de modèles 3D			
Niveau de discréétisation			
Blocs texturés			

Fonctionnalités implémentées partiellement :

- Chargement de modèles 3D
 - Il est possible de charger un modèle 3D à partir de son fichier .obj et .dds
 - Il est impossible de déplacer, ajouter ou supprimer un modèle 3D à exécution
 - L'implémentation de modèles 3D n'a pas été placé dans une fonction dédiée, cela se fait directement dans le main (il serait nécessaire de corriger ce point).

Fonctionnalités implémentées

Affichage de la scène avec des cubes

L'affichage de la scène est géré par le biais du *main* se trouvant dans le fichier *main.cpp* ; celui-ci va ainsi gérer la création et l'affichage des cubes. Les cubes sont créés et affichés par les classes *Cube* et *CubeList*. Par ailleurs, les fonctions d'affichage passent par l'utilisation des vertex et des fragment shaders.

Edition des cubes & Sculpture du terrain

A l'aide des classes *Cube* et *CubeList*, il est possible de modifier les cubes présents sur la scène ainsi que la scène en général. Par exemple, il est possible de positionner des cubes dans la scène, de les supprimer ou encore de modifier leurs textures. Par ailleurs, la gestion des évènements se fait via une interface graphique générée grâce à *ImGui*.

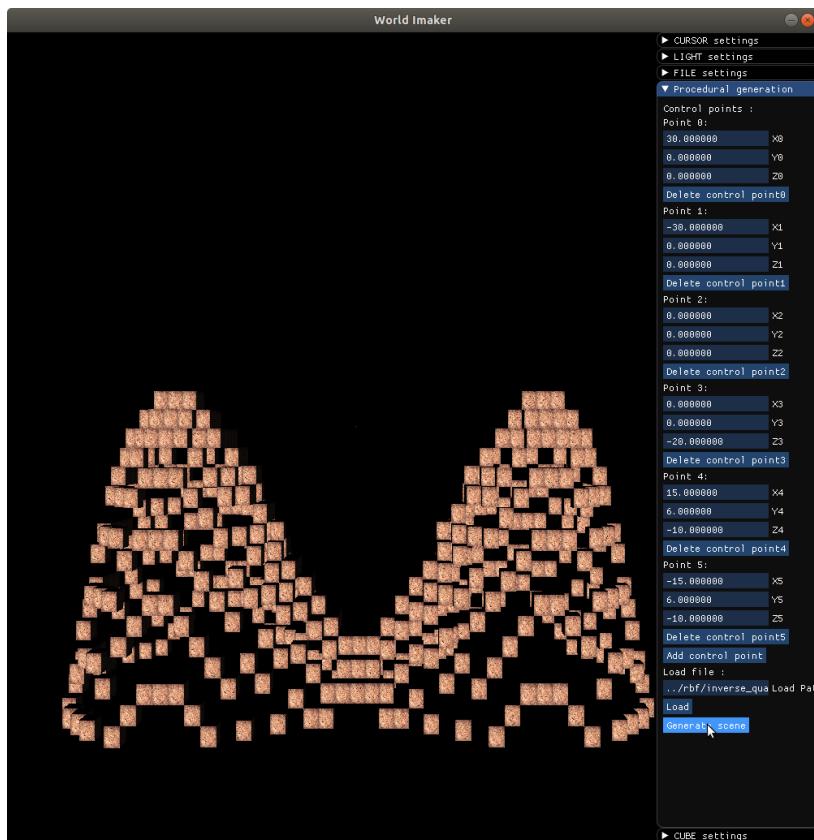
Génération procédurale

Afin d'automatiser la création de scènes et de terrains, il est possible de générer des cubes à partir de points de contrôle grâce à des *radial basis function*. Une fois les points de contrôle choisis, les cubes supplémentaires sont interpolés afin de créer un nouveau terrain. Cette fonctionnalité (calcul des poids pour des points de contrôle donnés et calcul des nouveaux cubes en conséquence) est gérée par la classe *CubeList* et utilise les méthodes de la bibliothèque *Eigen*. Un menu permet à la fois de choisir des points de contrôle manuellement et de charger des points de contrôle depuis un fichier créé préalablement. Les fichiers contiennent également la *radial basis function* et la valeur d'*epsilon* choisie.

Observations

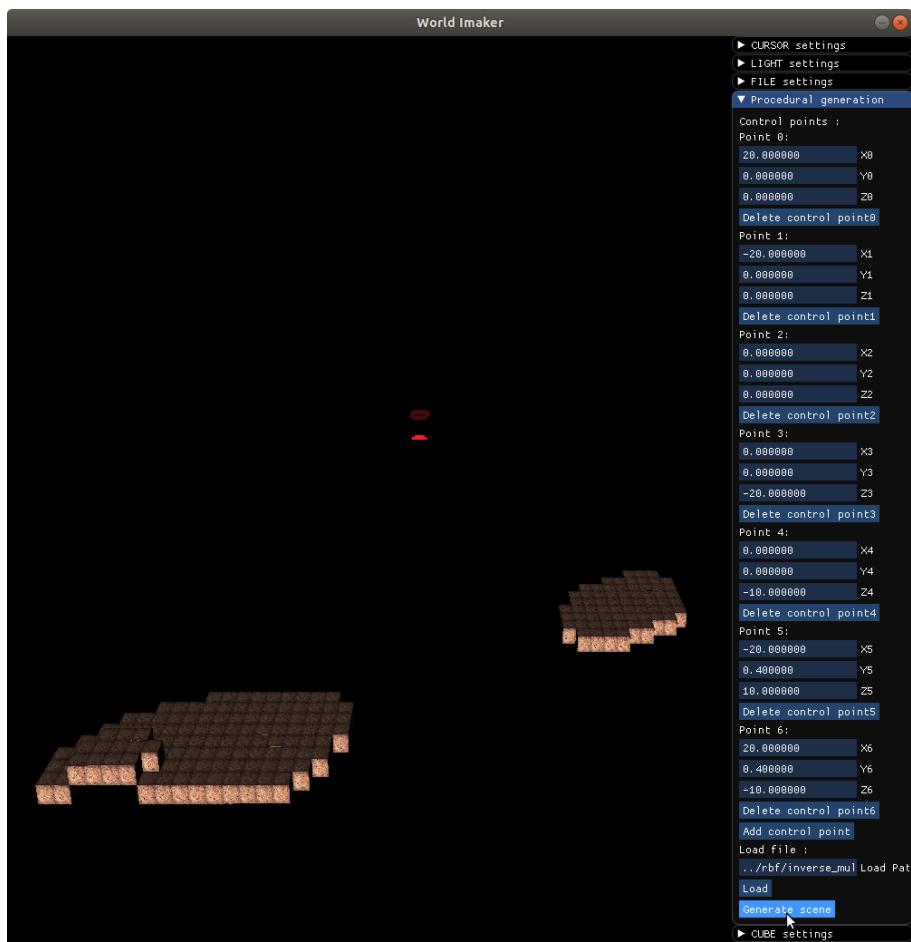
Selon les *radial basis functions* utilisées, les terrains générés sont différents. Au fil de nos expérimentations, nous avons pu observer que certaines fonctions étaient plus efficaces que d'autres pour générer des terrains divers.

Les fonctions utilisant des exponentielles et des fonctions inverses sont plus aptes à créer des courbes importantes. Faire varier *epsilon* permet de modifier (entre autre) l'arrondi des sommets.



$$M(x_i, x_j) = \frac{1}{1+(0.15*distance)^2}$$

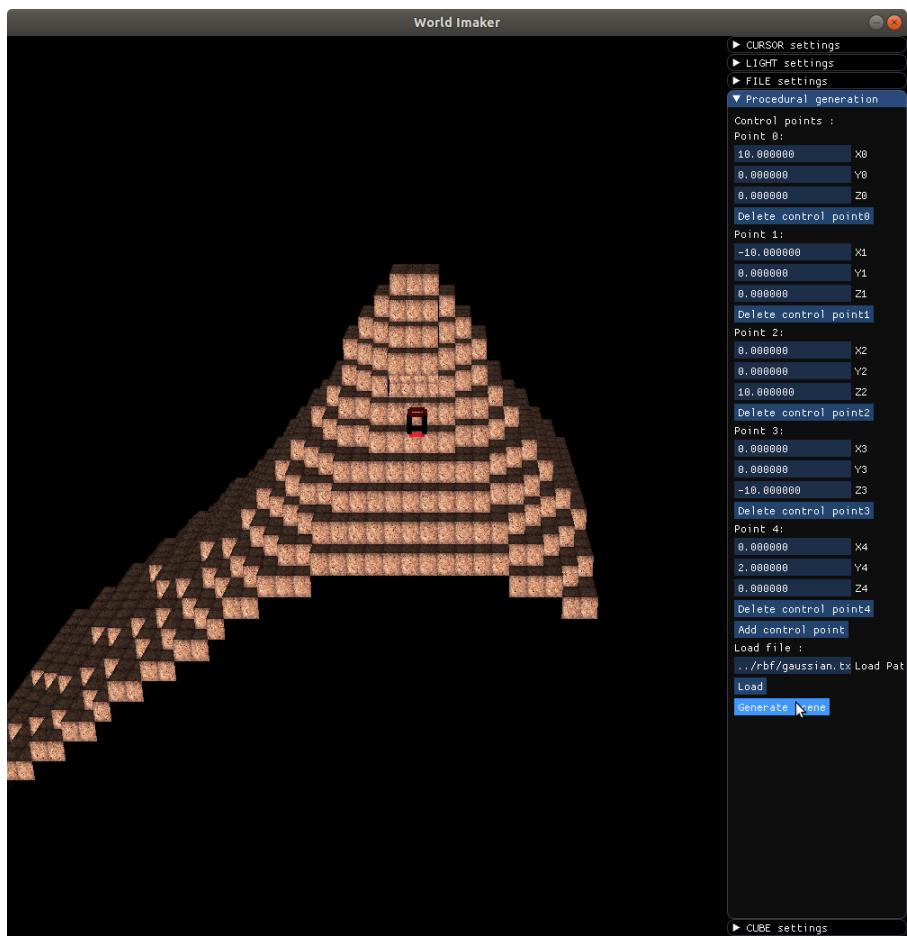
(./rbf/inverse_quadratic_test.txt)



$$M(x_i, x_j) = \frac{-1}{\sqrt{1+distance^2}}$$

(..../rbf/inverse_multiquadric_test.txt)

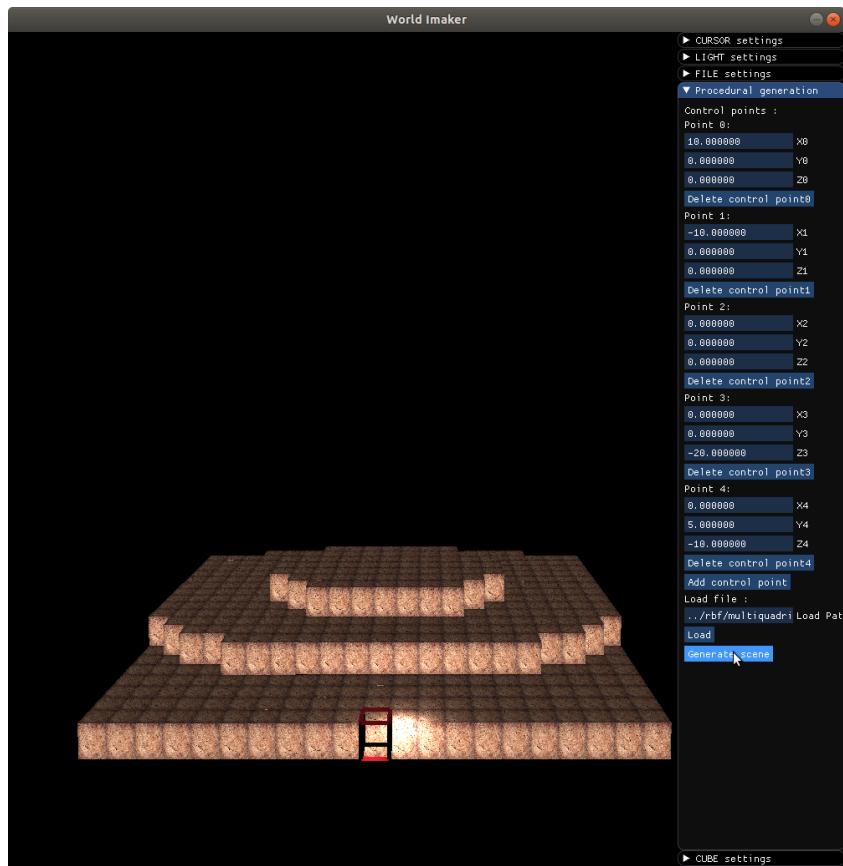
La génération des points étant limitée en hauteur, seul les sommets des pics apparaissent ici.



$$M(x_i, x_j) = -e^{-(5*distance)^2}$$

(./rbf/gaussian.txt)

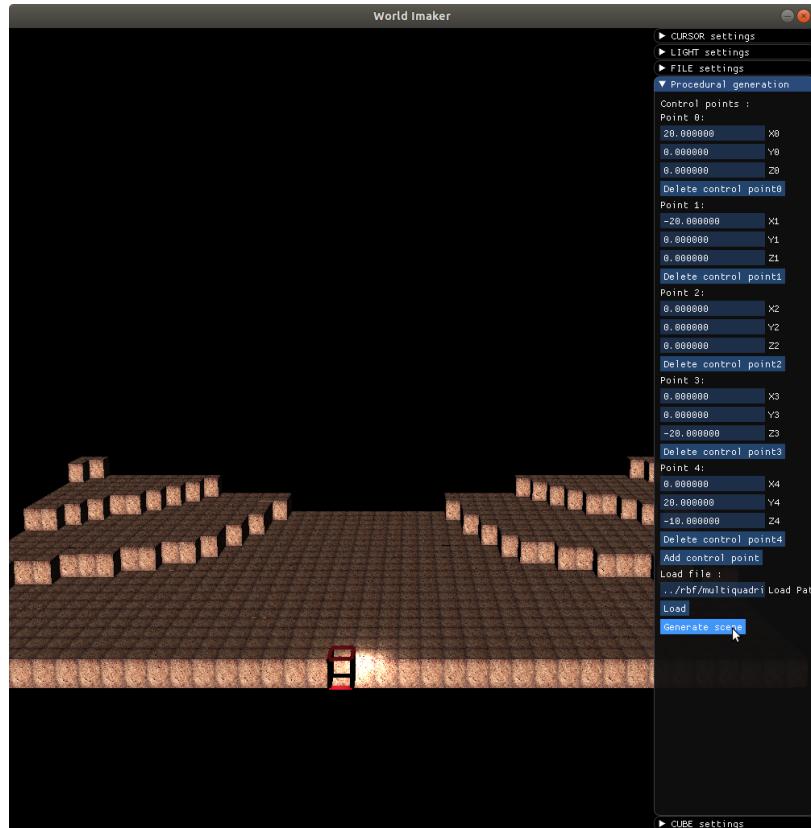
Les terrains créés à partir des fonctions *multiquadric* sont plus plats que les autres.



$$M(x_i, x_j) = \sqrt{1 + (distance)^2}$$

([..\\rbf\\multiquadric.txt](#))

Sur des terrains trop étroits, avec des distances trop petites, il est difficile de gérer les courbes et les aplats. Un point de contrôle placé trop loin des autres génère un creux plutôt qu'un pic.



En partant du même exemple, si l'on augmente la hauteur du point de contrôle central, la fonction s'évase et des pics commencent à apparaître aux bords du terrain, tandis que le centre reste creux.

([..../rbf/multiquadric.txt](#))

Ajout de lumières

Pour ce projet, nous avons fait le choix d'implémenter un modèle d'éclairage simple permettant de gérer des lumières ponctuelles et directionnelles, sans ombres. Pour ce faire, nous avons mis en place le modèle de Blinn-Phong. Ici, c'est le fragment shader qui se charge des calculs d'éclairage à l'aide des diverses fonctions. Par la suite, nous renvoyons les matrices en leur appliquant les transformations de lumière.

Sauvegarde / chargement de la scène

Au cours du projet, nous avons implémenté une fonctionnalité de sauvegarde et de chargement de scène simple. Elle permet de lire et d'écrire dans un fichier texte à l'emplacement choisi. La mise en forme de ce fichier est définie par défaut et n'est pas contrôlée par l'utilisateur. Les fonctions permettant la lecture et l'écriture des fichiers sont définies dans la classe *CubeList* qui gère l'ensemble des cubes de la scène. La gestion des emplacements, des chargements et des sauvegardes s'effectue grâce à une interface *ImGui*. Sont sauvegardés dans les fichiers les index, les positions, les textures des cubes ainsi que la position, l'intensité des lumières. A la lecture de ces fichiers, la scène est réinitialisée à l'aide de la classe *CubeList* puis rafraîchie grâce aux informations fournies par le fichier.

Chargement de modèles 3D

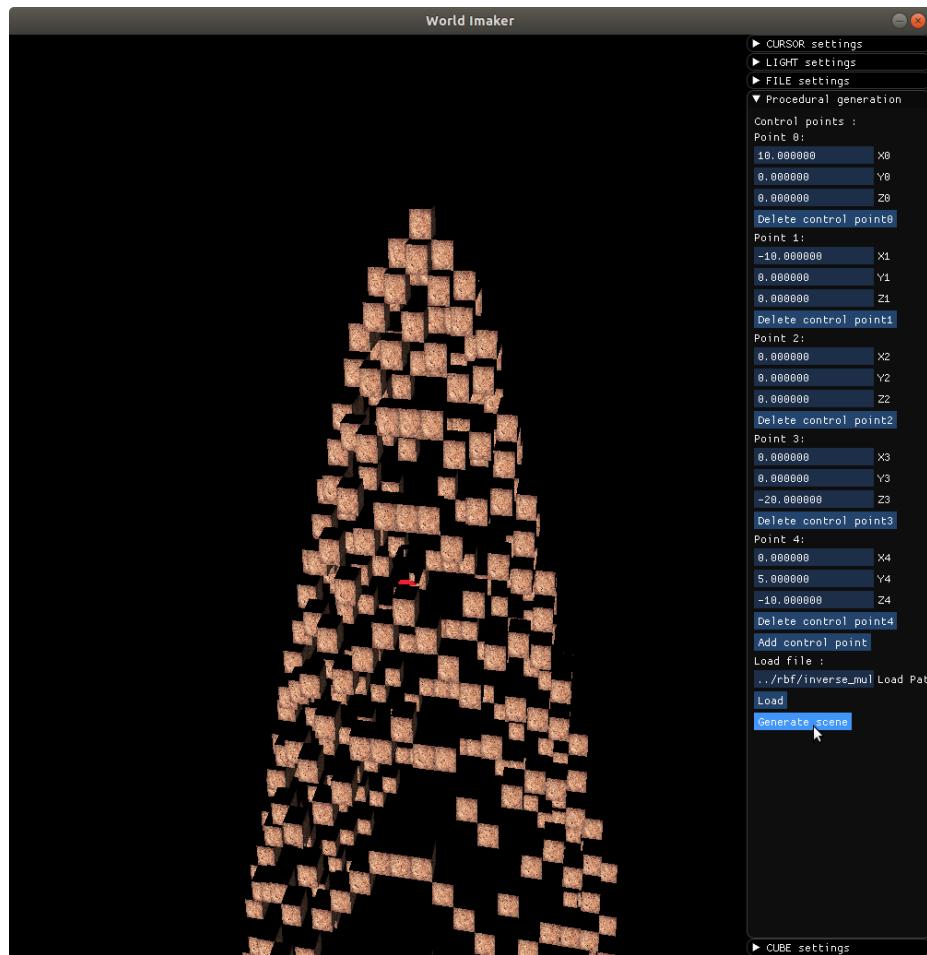
Pour cette fonctionnalité, nous avons souhaité pouvoir mettre en place des modèles 3D dans la scène qui seraient chargés à partir d'un fichier. Ne parvenant pas à utiliser la librairie *assimp*, nous avons opté pour une fonction permettant de charger des modèles 3D. Dans notre cas, le format de fichier utilisé est le format OBJ. Les modèles 3D ne doivent posséder qu'une seule coordonnée UV et une normale par sommet. L'implémentation de cette fonctionnalité n'est que partielle car sa gestion s'effectue dans le *main*. L'édition, la modification, le déplacement et la suppression de ces objets à runtime ne sont pour le moment pas possibles.

Blocs texturés

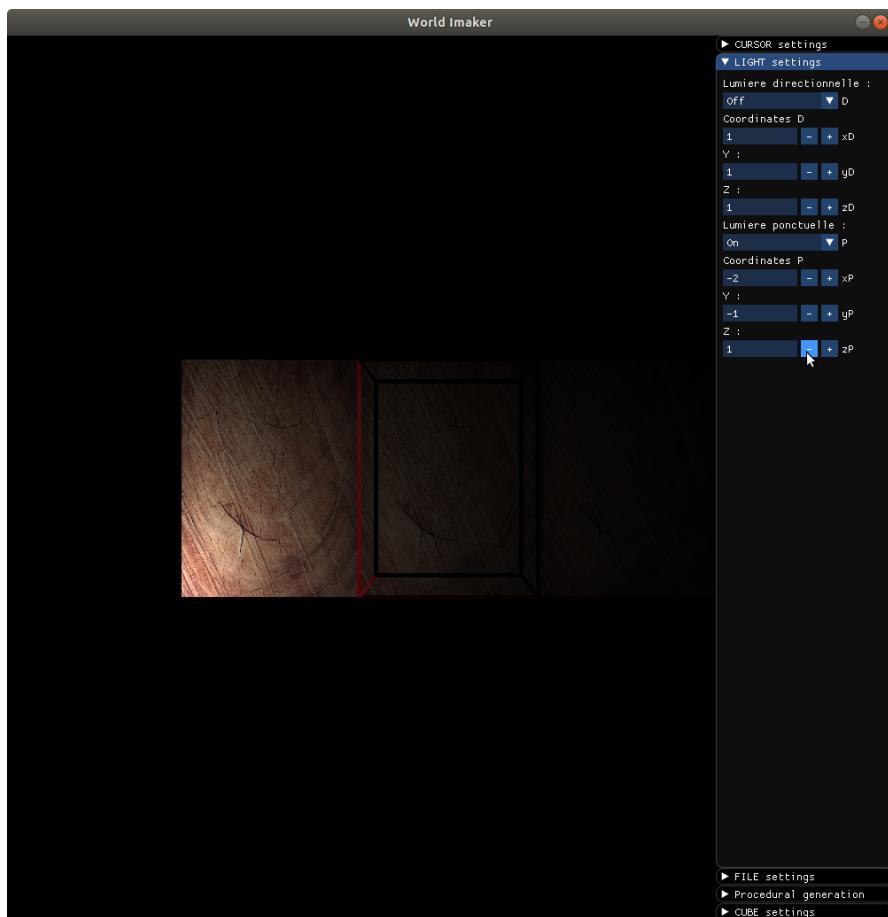
Afin d'appliquer une texture sur nos blocs, nous avons dû mettre en place des fonctions d'accès aux images représentant les textures. Par la suite, il était nécessaire de modifier nos shaders. Par exemple, le fragment shader a besoin des coordonnées UV de la texture. De plus, le multi-texturing permet de dessiner des cubes de différentes textures dans une même scène.

Résultats obtenus

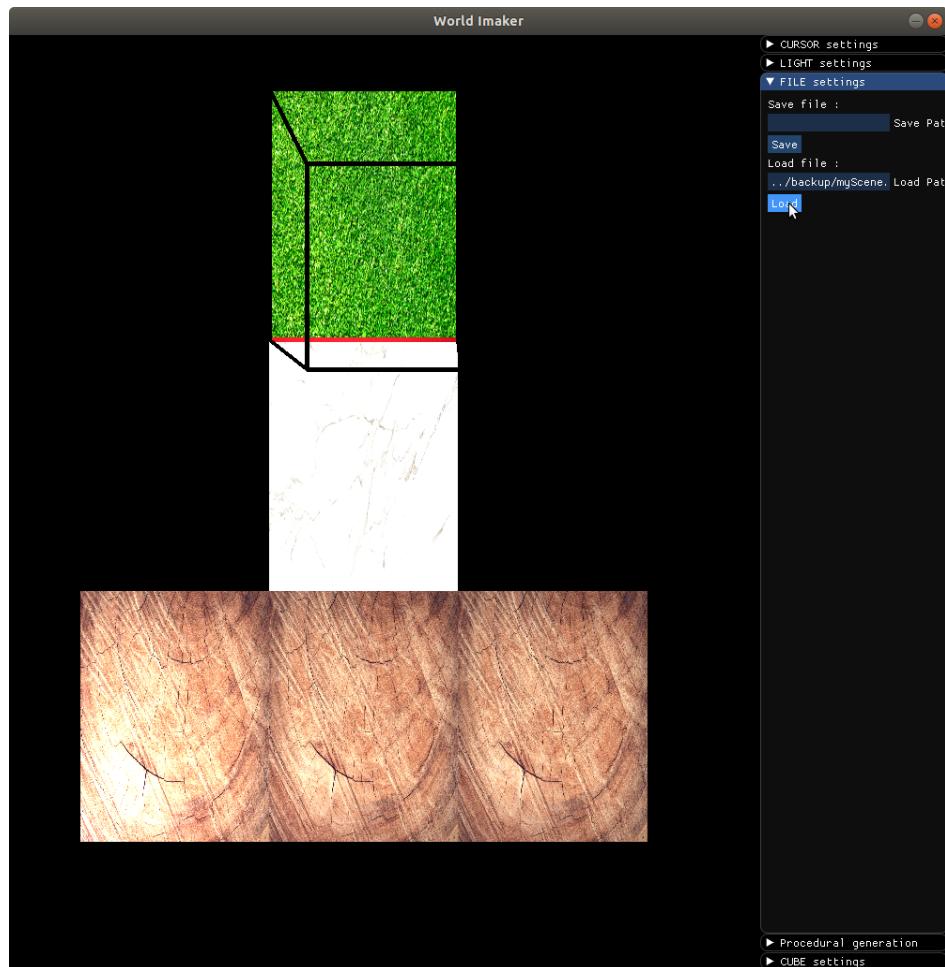
Génération procédurale



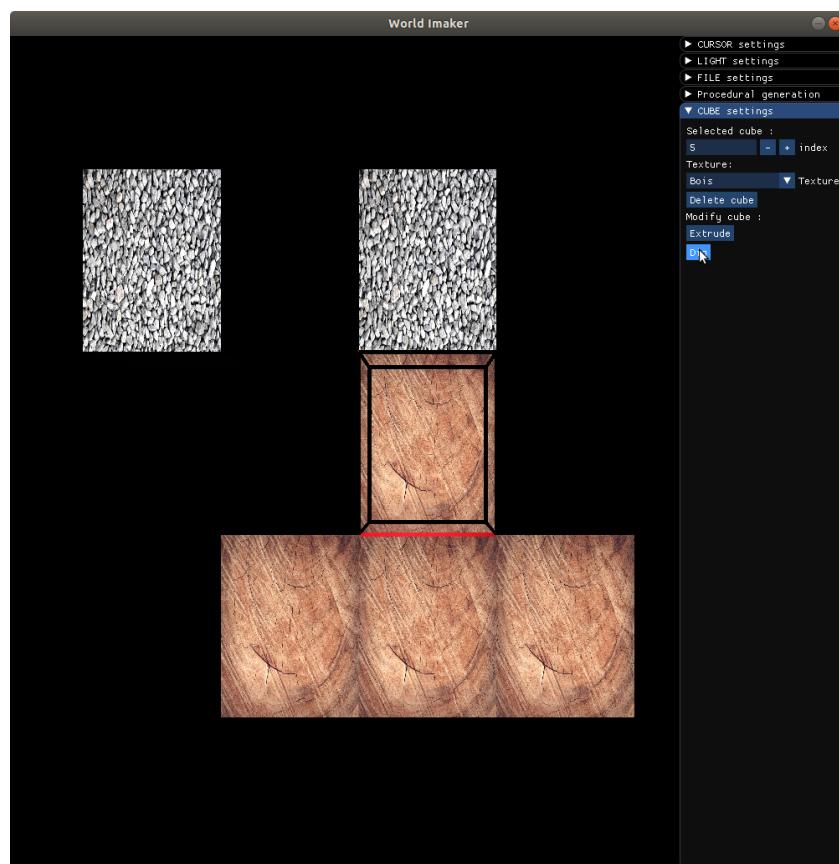
Ajout de lumières



Sauvegarde/Chargement de fichiers

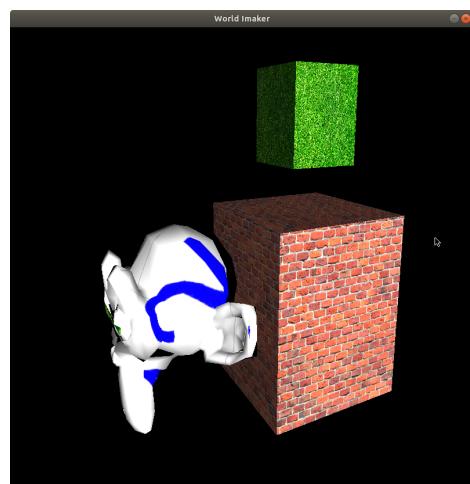
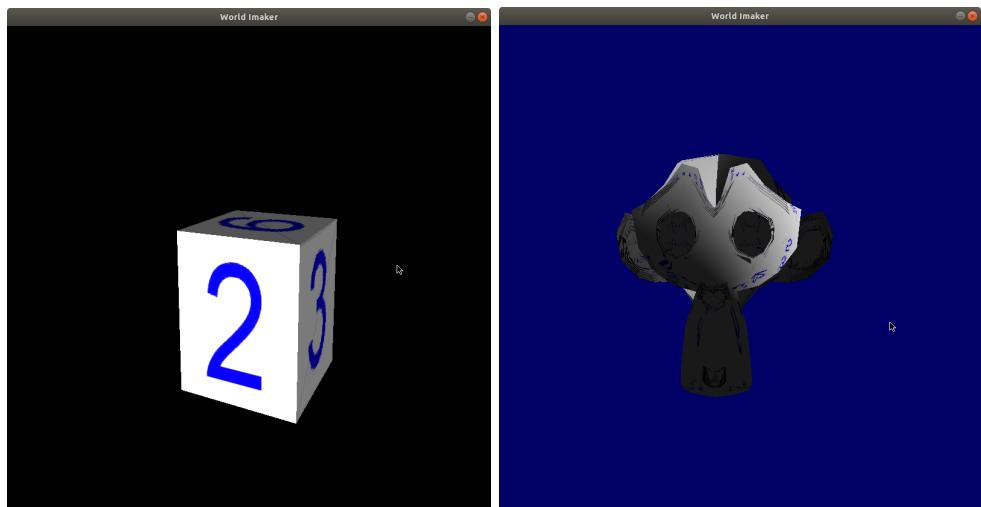


Fonctions d'édition des blocs

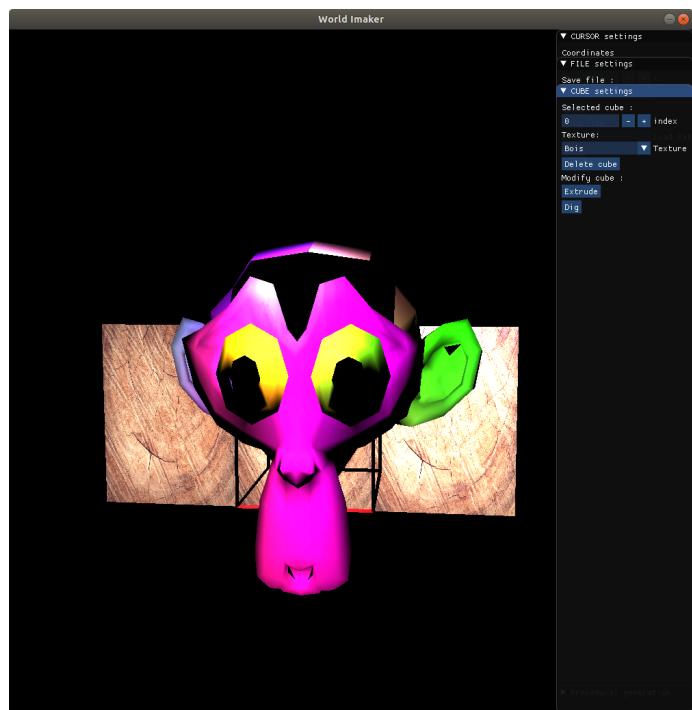


Chargement de modèles 3D

Exemples lors des premiers tests



Exemple disponible à partir de l'exécutable World_Imaker_2



Difficultés rencontrées

Durant la réalisation de ce projet, nous avons fait face à diverses difficultés.

La première fut la mise en place du Git. En effet, nous avions trop peu l'habitude de cet outil de travail. C'est pourquoi nous avons eu quelques difficultés à mettre en place les branches et les merge correctement. De ce fait, nous avons perdu beaucoup de temps à corriger certains conflits inutilement. Cependant, en avançant sur ce projet, nous avons trouvé une organisation des tâches qui nous convenait, et nous sommes désormais à l'aise avec cet outil.

Par ailleurs, nous avons également eu des difficultés lors de la mise en place des librairies, principalement celle de *ImGui*. Ce fut l'une des premières librairies que nous souhaitions mettre en place, mais il nous a été difficile de trouver de la documentation à son sujet. De plus, l'importation de ces librairies via nos fichiers cmake existants nous a pris beaucoup de temps, en raison de notre manque d'expérience dans la manipulation de cet outil.

Nous avons rencontré des problèmes liés à l'utilisation des ordinateurs mis à disposition à l'ESIPE. En effet, Linux n'étant pas forcément installé sur l'ensemble de nos machines personnelles, nous avons été amenées à utiliser les ordinateurs de l'école régulièrement. Cela nous a obligé non seulement à travailler sur des horaires restreintes, mais également à gérer des problèmes d'espace mémoire redondants et de logiciels limités. De ce fait, nous avons dû mettre en place une organisation efficace sur une période de temps réduite, afin de terminer le projet selon les spécifications demandées.

Enfin, il est important de souligner qu'au vu du temps imparti pour réalisation de ce projet, ainsi que des diverses difficultés que nous avons rencontrées, énumérées ci-dessus, nous n'avons pas pu mettre en place un code aussi harmonieux que nous l'aurions voulu. En effet, la mise en place de "bonnes pratiques" telles que la gestion totale des erreurs, une documentation complète et des fonctions optimales, n'ont été que partiellement implémentées.

Note personnelle

MANSION Amélia

Ce projet m'a permis de mettre en pratique de nouvelles connaissances récemment acquises durant les cours de Programmation et Algorithmie, Synthèse d'Images et Mathématiques. Il fut dense et ambitieux, mais il m'a permis de réellement comprendre des subtilités qui n'étaient que théoriques. Durant ce projet, nous avons pu mettre en place une cohésion d'équipe avec Manon afin d'être le plus productives possible. De plus, nous pouvions compter sur nos enseignants et camarades pour nous aider à débuguer notre code.

SGRO' Manon

Grâce à ce projet, j'ai pu réellement mettre en pratique et comprendre ce que nous avions récemment appris durant les cours de Programmation, Synthèse d'Images et Mathématiques. Je suis satisfaite des nouvelles compétences que j'ai acquises malgré le manque de connaissances que j'avais au départ et les difficultés auxquelles nous avons dû faire face. Nous avons trouvé une bonne cohésion d'équipe avec Amélia, et nous avons reçu une aide précieuse de la part de nos enseignants et de nos camarades. Grâce à cela, nous avons pu atteindre la plupart des objectifs que nous nous étions fixés avec plus d'efficacité et moins de tension que lors des projets similaires de l'année précédente.