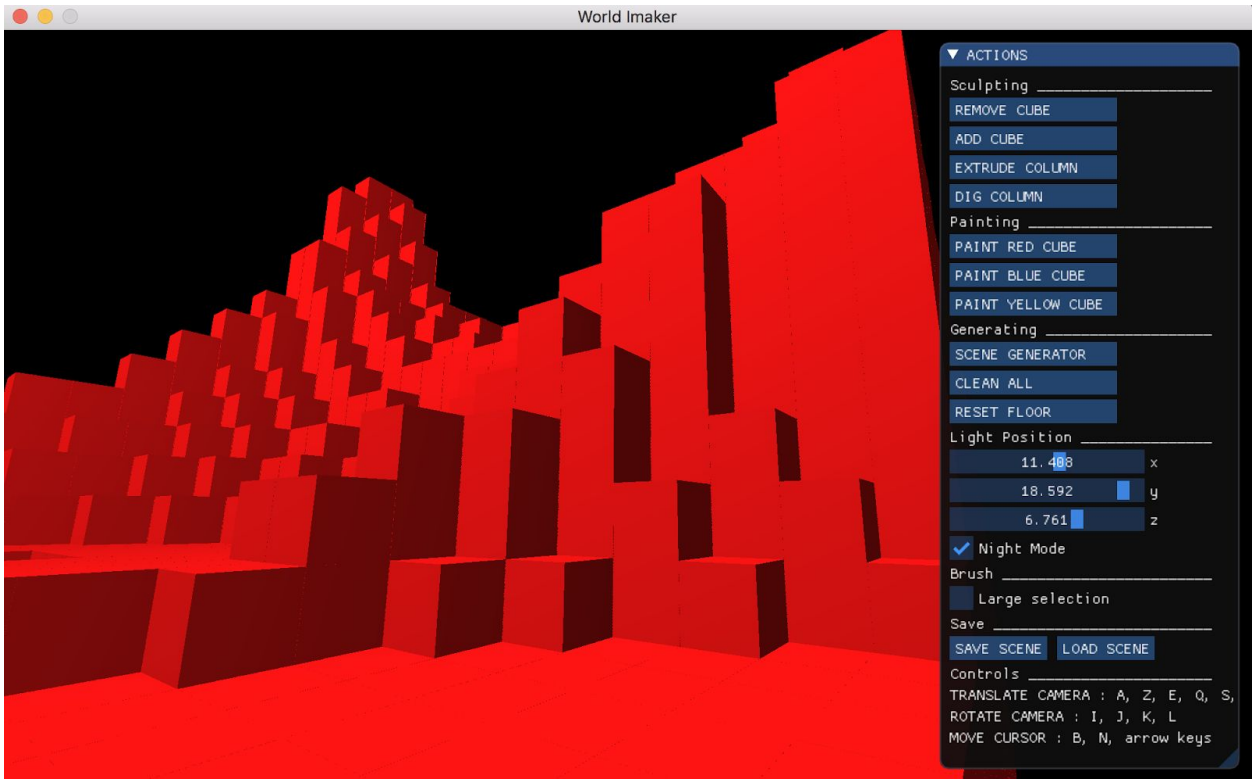
Johan BOYER & Elisa CIAVALDINI

WORLD IMAKER

CODER UN ÉDITEUR DE TERRAIN



SOMMAIRE

I) DESCRIPTIF GÉNÉRAL

Fonctionnalités requises	Détails	Implémentée	Commentaire
Etat initial	-Affichage d'une couche de trois cubes d'épaisseur	Complètement	
	-Utilisation d'une Freefly Camera pour le déplacement dans la scène	Complètement	Au clavier
Edition des cubes	-Manipuler un curseur	Complètement	Au clavier
	-Visualiser ce curseur	Complètement	Affichage en Wireframe
	-Changer le type du cube sélectionné	Complètement	Par l'interface, via bouton
Sculpture du terrain	-Outils de base (ADD, REMOVE, EXTRUDE, DIG)	Complètement	Au clavier ou par l'interface via boutons
Génération procédurale	-Générer une scène en utilisant les RBF	Complètement	Par l'interface, via bouton
Ajout de lumières	-Changer de mode en jeu	Complètement	Par l'interface, via une checkbox
	-Utiliser une lumière directionnelle	Complètement	Day Mode: Checkbox off
	-Utiliser des lumières ponctuelles	Partiellement	Night Mode: Checkbox on. 1 lumière, non matérialisée
	-Positionner ces objets en jeu (Direction ou position)	Partiellement	Par l'interface, via sliders

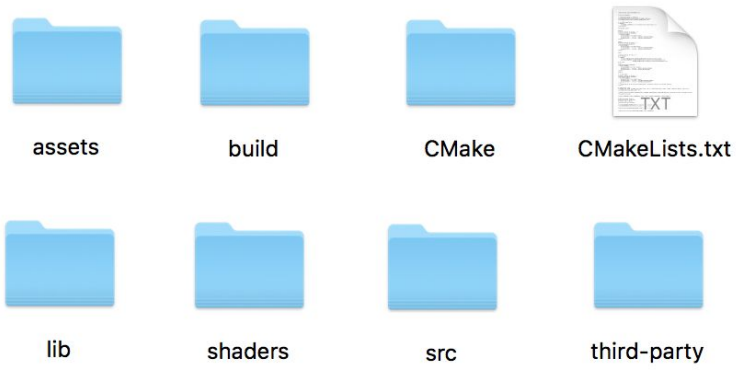
Fonctionnalités additionnelles	Détails	Implémentée	Commentaire
Amélioration de la sélection	-Sélection grâce à la souris	Non	
Outils de painting	-Peindre la zone sous la souris	Partiellement	La zone peinte se trouve autour du curseur, via l'outil Brush
	-Régler le diamètre d'application	Complètement	Par l'interface, via sliders
	-Presser un bouton pour activer ce mode	Complètement	Par l'interface, via checkbox
Sculpture du terrain améliorée	-Amélioration des fonctions EXTRUDE / DIG	Non	
Sculpture du terrain améliorée 2 *	-Amélioration des fonctions ADD / REMOVE:	Complètement	Le groupe de cubes créés/supprimés se trouve autour du curseur, via l'outil Brush
	-Régler le diamètre d'application	Complètement	Par l'interface, via sliders
	-Presser un bouton pour activer ce mode	Complètement	Par l'interface, via checkbox
Sauvegarde / Chargement de la scène	-Ecrire un fichier contenant les informations nécessaires à décrire une scène	Complètement	
	-Lire un fichier contenant les informations nécessaires à décrire une scène	Complètement	
	-Utiliser l'interface pour sauvegarder et charger une scène	Partiellement	Par l'interface, via bouton Chemins statiques

Fonctionnalités additionnelles	Détails	Implémentée	Commentaire
Chargements de modèles 3D	-Importer des modèles extérieurs	Non	
Niveau de discrétisation	-Ajuster la taille des cubes en jeu	Non	
Blocs texturés	-Appliquer un set de textures aux cubes	Non	
Outils utiles *	-Supprimer tous les cubes de la scène d'un clic	Complètement	Par l'interface, via bouton
	-Réinitialiser le sol de l'état initial	Complètement	Par l'interface, via bouton

**Les fonctionnalités marquées d'un astérisque ne sont pas indiquées dans le sujet, elles peuvent correspondre à la partie "vos idées".*

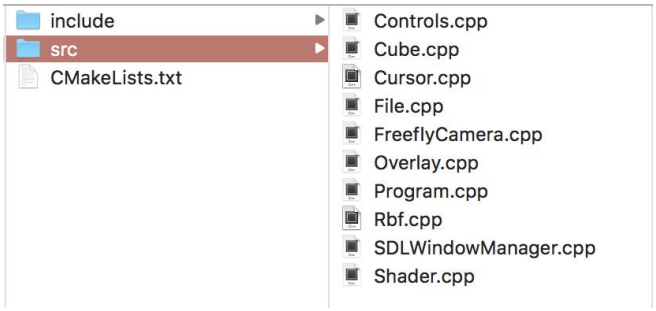
II) ARCHITECTURE

Organisation :

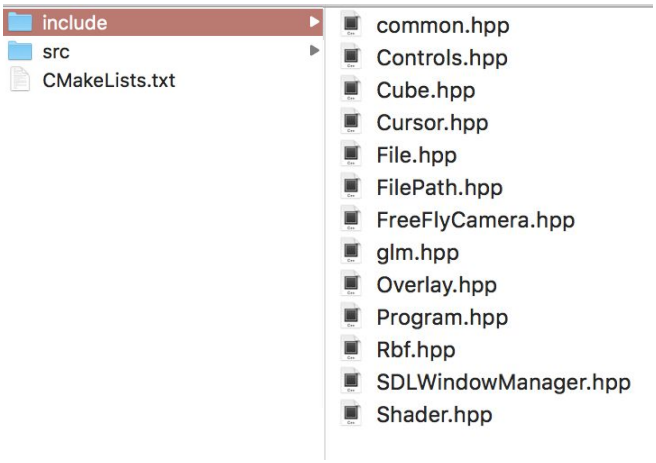


- assets contient la liste.txt des points de contrôle nécessaires pour les RBF

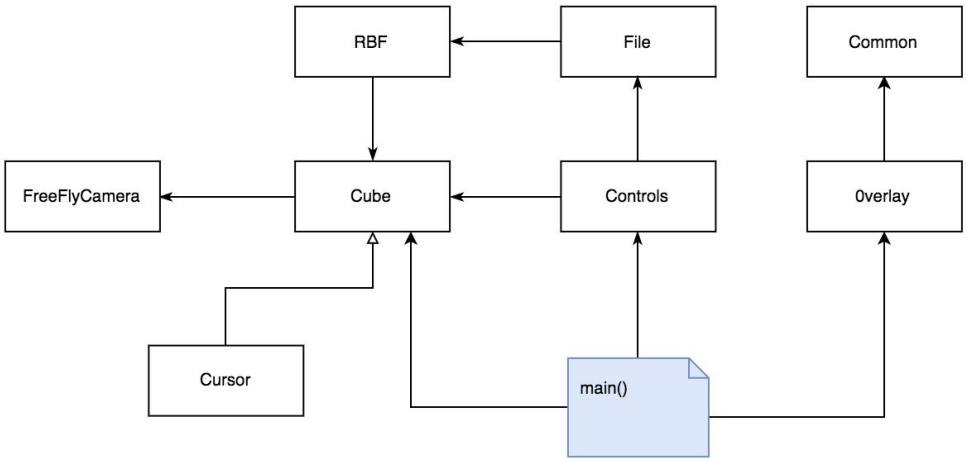
- lib contient les librairies utilisées : glimac (les fichiers sources et header du programme) et Imgui.



fichiers .cpp du programme



fichiers .hpp du programme



échanges entre les fichiers du programme

- Common.h : contient les variables globales consexpr du programme, ainsi que la structure ShaderProgram nécessaire pour le chargement des shaders.

- Overlay.h : contient toutes les fonctions relatives à la librairie Imgui et à l'interface utilisateur.

- Controls.h : contient les fonctions permettant de gérer les actions sur la scène et les cubes (sculptCube(), paintCube, MoveCamera() …)

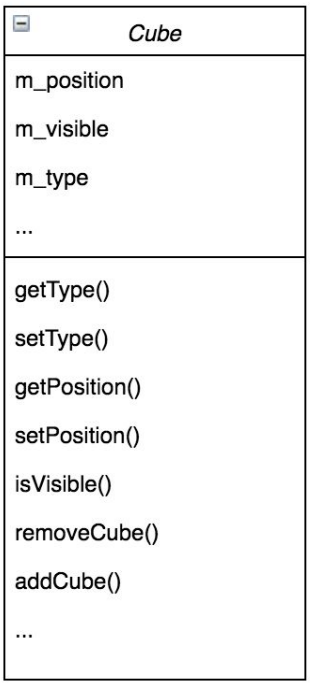
III) GUIDE UTILISATEUR

Mouvement Caméra	**Mouvement curseur**
Rotation gauche : **L** Rotation droite : **J** Déplacement latéral gauche : **Q** Déplacement latéral droit : **D** Avancer : **Z** Reculer : **S** Regarder vers le haut : **E** Regarder vers le bas : **A**	Droite : **Flèches directionnelles** Gauche : **Flèches directionnelles** Haut : **Flèches directionnelles** Bas : **Flèches directionnelles** Avancer : **N** Reculer : **B**

IV) FONCTIONNALITÉS

A) REQUISES

a) Affichage d'une scène avec des cubes

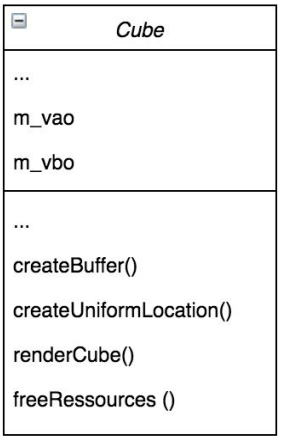


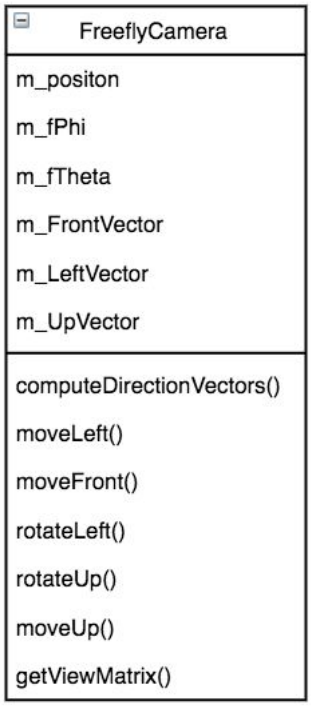
- *Génération de cubes*

Nous avons implémenté une classe "Cube".
Chaque cube est caractérisé par sa position, sa visibilité et son type.
Nous utilisons ces objets dans une liste de cubes qui remplit l'espace.
L'espace est donc rempli de cubes, et nous jouons ensuite sur leur visibilité et leur type pour les afficher ou non et choisir leur couleur.

Au lancement, les 3 premières couches de cubes sont rendues visible afin de matérialiser un sol de base.

La partie opengl gérant les buffers et le dessin des cubes est également déléguée dans cette classe "Cube". Les différentes méthodes vont permettre de créer les buffers(*createbuffer()*), les variables uniformes (*createUniformLocation()*), de dessiner le cube(*renderCube()* et de libérer l'espace (*freeRessources()*)





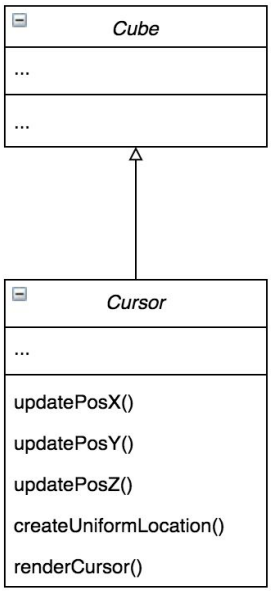
- *Caméra*

Nous avons implémenté une classe "FreeFlyCamera".
Cette classe permet de gérer la caméra qui va pouvoir se déplacer dans la scène grâce aux touches du clavier.

Les événement relatifs à la gestion de la caméra sont gérés dans les fichiers "Control.hpp / Control.cpp".
Les évènement appellent des méthodes de la classe FreeflyCamera.
Ces différentes méthodes permettent les mouvements de translation (*moveLeft()*,*moveFront()* et *moveUp()*) et des mouvements de rotation (*RotateLeft()* et *RotateUp()*).

b) Edition des cubes

- *sélection des cubes*



Pour la sélection des cubes, nous avons créé une classe fille “Cursor” qui hérite de Cube.
Cela permet au curseur d’avoir les mêmes caractéristiques qu’un Cube, ce qui facilite et simplifie son utilisation.
Le curseur possède donc la même taille d’un cube et va pouvoir se déplacer facilement de cube en cube.
L’utilisateur peut déplacer le curseur à l’aide des flèches directionnelles et des touches “N” et “B”. Lors d’une interaction de ce type l’utilisateur déclenche un événement qui appelle une fonction permettant au curseur de se déplacer. Ces fonctions sont des méthodes implémentées à la classe Curseur qui vont changer la position du cube Curseur.

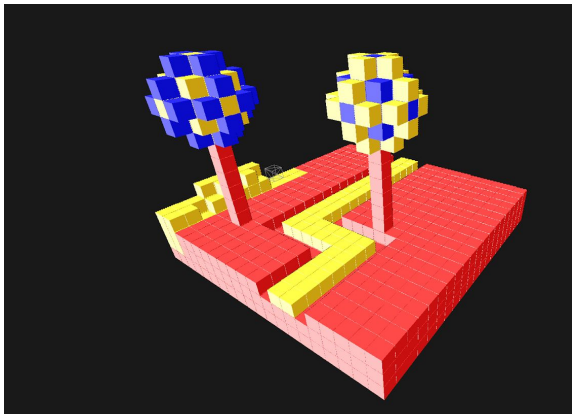
Lorsque le curseur se trouve sur un cube on fait correspondre sa position avec le cube correspondant présent dans la liste de cube grâce à une formule.

- *affichage du curseur*

Le curseur est affiché de la même manière qu’un cube. Cependant on utilise la fonction *glPolygonMode()* pour pouvoir lui donner un aspect de wireframe, le différenciant ainsi des autres cubes, et permettant de dessiner les contours du cube sélectionné. Pour afficher le curseur en permanence on utilise *glDisable(GL_DEPTH_TEST)* et *glEnable(GL_DEPTH_TEST)*.

- *édition des couleurs des cubes*

Pour gérer la couleur des cubes et du curseur on utilise des shaders. La couleur d’un cube (visible ou non) est défini par son attribut m_type. On a défini 3 couleurs pour les valeurs 1=rouge, 2=bleu et 3=jaune.
Lors du changement de type, le shader va se charger de changer la couleur du cube grâce à la variable uniforme envoyée “uCubeType”.



c) Sculpture du terrain

Les méthodes *addCube()* et *removeCube()* modifient la visibilité du cube, s'il est visible alors il est affiché. Cette condition est testée dans la méthode dédiée au dessin.
Dans la fonction *sculptCubes()* ces méthodes sont appelées pour implémenter les quatre fonctionnalitées de sculpture du terrain.

Pour add et remove : En partant des coordonnées du curseur on détermine l'indice du cube correspondant, comme vu plus haut, on peut alors ajouter ou non le cube en le rendant visible ou non.

*Index du cube dans la liste de cube = cursorPosition.z*volume+cursorPosition.x+cursorPosition.y*volume*volume*

Pour extrude et dig : Toujours à partir des coordonnées du curseur on détermine l'indice du cube le plus bas de la colonne (grâce à une autre formule) pour extruder ou creuser.

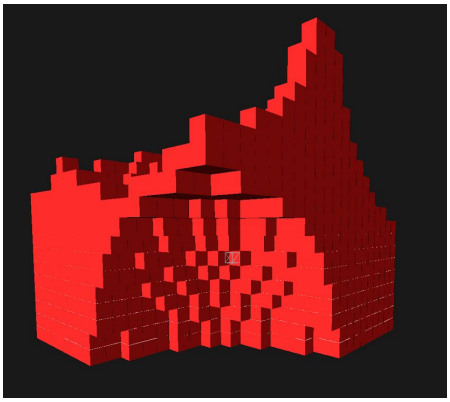
*Index du cube le plus bas de la colonne dans la liste de cube = cursorPosition.z*volume+cursorPosition.x;*

d) Génération procédurale

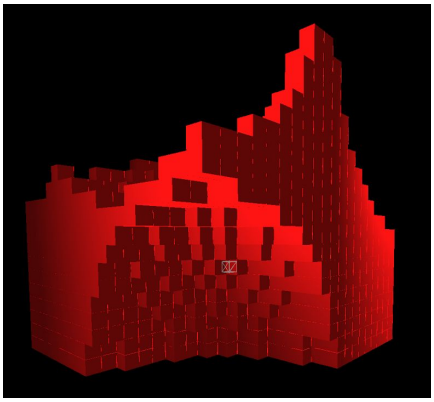
Il est possible de générer une scène à partir d'un fichier .txt contenant les points de contrôle et les coefficients en utilisant les radial basis functions, nous avons décidé de garder la fonction Gaussienne. (voir partie RBF)

e) Ajout de lumières

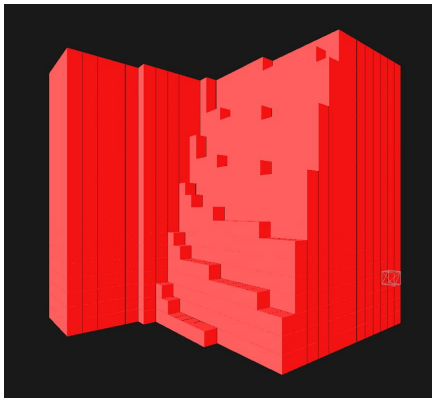
Le mode jour/nuit est réglable depuis l'interface à l'aide d'une checkbox.
La direction initiale et le type de lumière sont défini par un vec4 qui est envoyé en variable uniforme au shader.
En mode jour, la lumière directionnelle s'active, des sliders sont disponibles pour gérer la diffusion de la lumière sur les axes x,y et z.
En mode nuit, on bascule sur une lumière ponctuelle non matérialisée que l'utilisateur peut positionner sur les axes x,y et z.



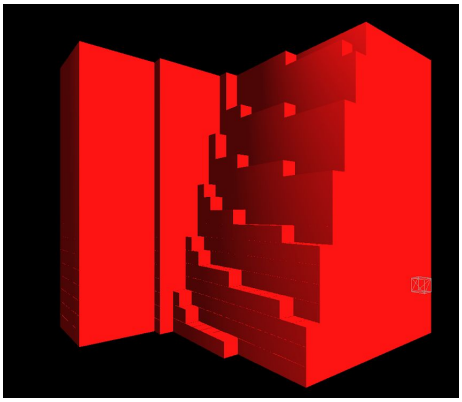
scène jour



scène nuit



scène jour

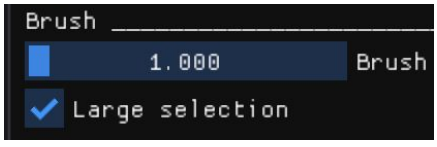


scène nuit

B) ADDITIONNELLES

a) Outils Brush

En travaillant sur les RBF nous sommes parvenus à créer une zone d'action autour d'un point sélectionné. Nous avons donc utilisé cet outil comme "Brush" pour améliorer la sélection sans utiliser le curseur de la souris.
La taille de cette brush est réglable via l'interface et on peut l'activer en un clic.
La brush teste tous les cubes de la zone d'édition, calcule leur distance et la compare aux conditions implémentées. Les cubes dans la zone sont alors modifiés.



b) Outils de painting

Avec cette brush, il est alors facile de peindre un groupe de cubes en une seule fois.
L' action effectuée après sélection est un *setType()* qui va modifier le type des cubes dans la zone de sélection et ainsi leur couleur.

c) Outils de sculpting

De même, toujours avec cet outil Brush, nous avons pu améliorer l'efficacité des fonctions d'ajout et de suppression de cubes. Il est possible d'ajouter et de supprimer plusieurs cubes en un clic. Les actions effectuées après sélection sont *addCube()* ou *removeCube()* et vont permettre de modifier la visibilité des cubes dans la zone de sélection.

d) Sauvegarde / Chargement de la scène

Les fichiers "File.cpp/File.hpp" regroupent les fonctions permettant d'écrire et de lire des fichiers .txt contenant les informations nécessaires à la construction d'une scène.

Ces deux fonctionnalités sont accessibles via l'interface de l'application, les chemins sont entrés en dur et donc non modifiables.

e) Nos idées

Afin de faciliter l'expérience utilisateur nous avons implémenté deux nouvelles fonctionnalités de génération.

La première permet de rendre invisible tous les cubes de la scène. Il est ainsi possible de créer une scène sans sol par défaut.

La seconde complète la première car elle permet de générer le sol par défaut. Elle ne supprime pas la scène en cours de création, elle ajoute seulement le sol de départ.

V) RADIAL BASIS FUNCTIONS

A) MATHÉMATIQUES

Dans un premier temps nous avons effectué les calculs sur papier en prenant un exemple d'une matrice 3x3. Cela nous a permis de comprendre ce qu'est une RBF et comment les utiliser pour interpoler des valeurs dans un champ de points.

Une fois les mathématiques assimilées, nous avons défini les étapes à suivre dans le code pour résoudre notre système de la forme AW=B:

- Créer la matrice A
- Calculer les valeurs et remplir la matrice
- Créer le vecteur B
- Résoudre le système

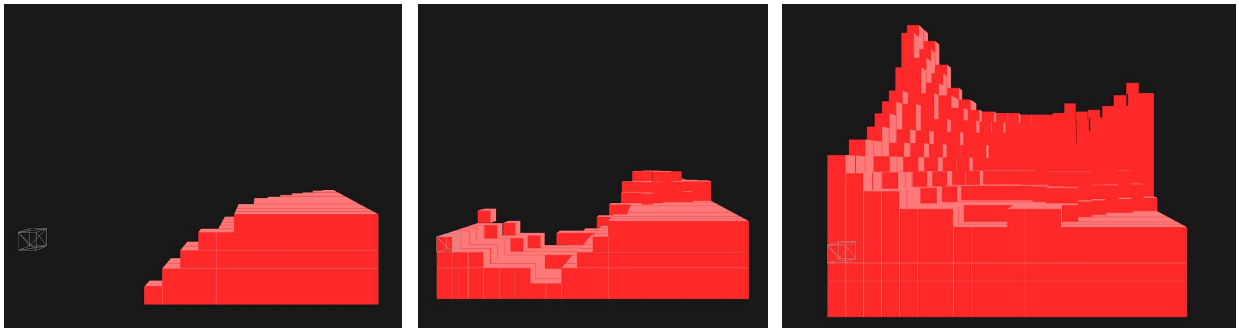
Nous avons remarqué que la matrice A est symétrique. Il nous suffit donc de calculer la matrice triangulaire stricte de l'ajouter à sa transposée et d'enfin ajouter la matrice identité multipliée par le coefficient A(0,0).

B) TESTS

a) <u>Influence d'epsilon</u>

Nous avons commencé par utiliser la fonction Gaussienne. Nous avons effectué plusieurs tests afin de fixer une valeur satisfaisante pour epsilon.

Les résultats ci-dessous illustrent l'évolution du relief en fonction de epsilon à points de contrôles fixés.

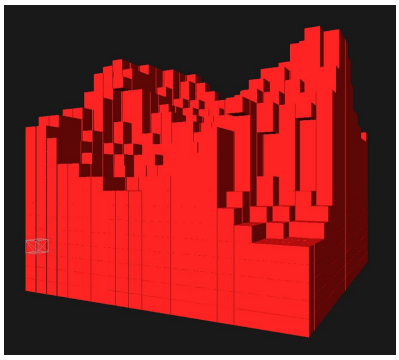


ε=0.1 *ε=0.5* *ε=1.0*

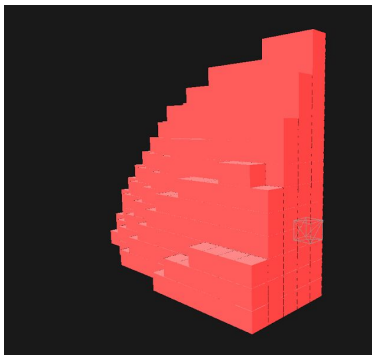
Nous avons donc préféré un epsilon valant 1 pour créer un relief conséquent.

b) Différentes fonctions

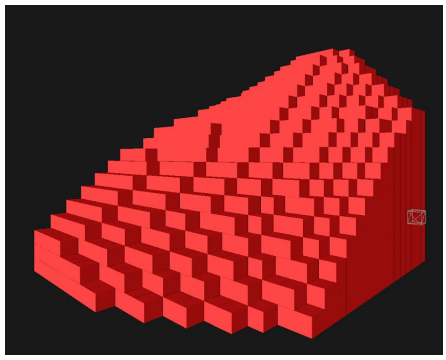
Nous avons implémenté plusieurs autres RBF, nous les avons testés afin de voir laquelle était plus maniable à notre échelle.



multi quardatique

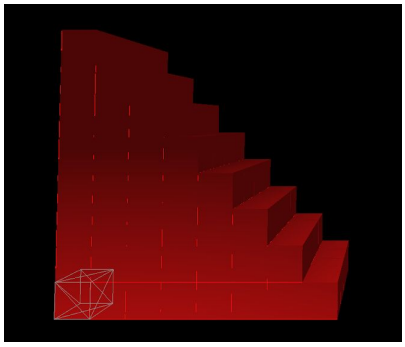


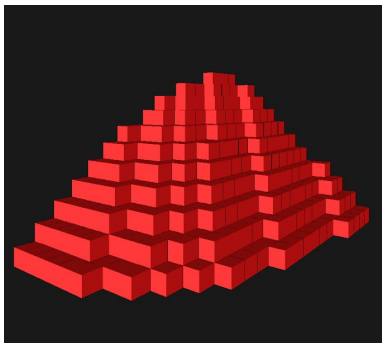
quadratique inverse

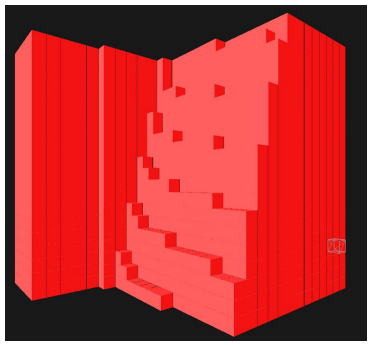


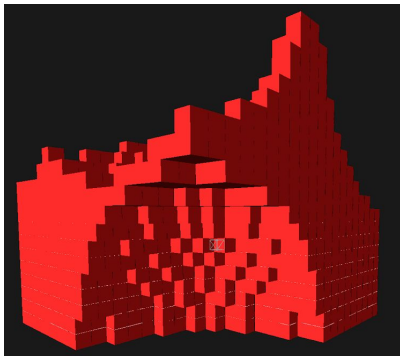
bi harmonique

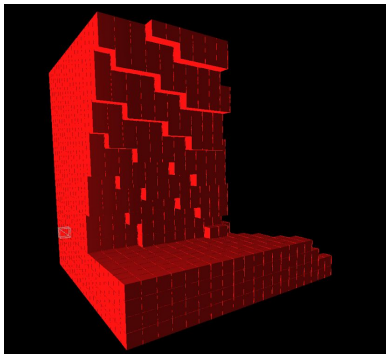
Nous avons réussi plus facilement à créer des reliefs différents avec la fonction Gaussienne. Nous avons donc décidé de continuer en l'utilisant.

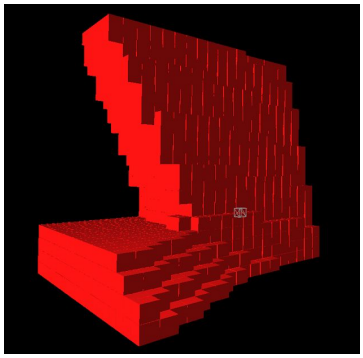










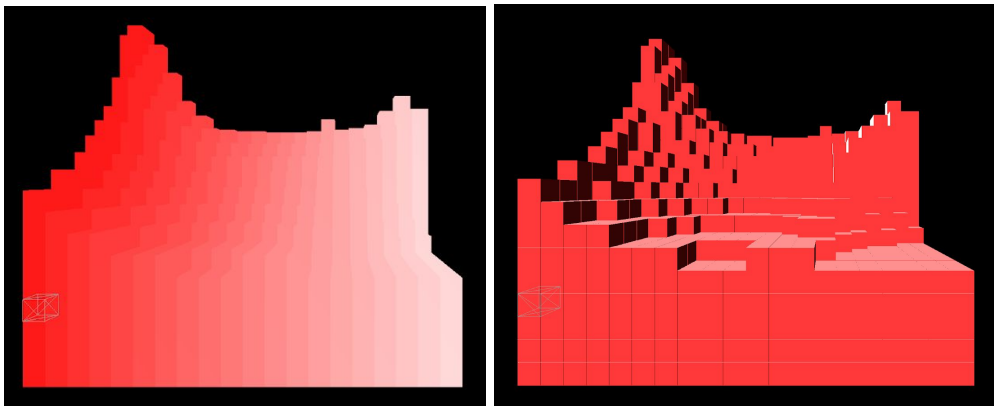


VI) DIFFICULTÉS

A) CUBES

Nous avons d'abord tenté de créer puis supprimer chaque instance de cube à la demande, mais nous avons opté pour un attribut de visibilité dictant si oui ou non le cube sera dessiné. C'est un choix plus coûteux en mémoire puisque tous les cubes sont toujours présents mais ça simplifie l'édition.

Nous avions mal compris l'utilité des normales quant à l'éclairage de la scène. Nous avons du reprendre une partie du code de création des cubes lorsque nous avons commencé à travailler sur les lumières.



Avant et après correction du problème de normales

B) SHADERS

Nous avons rencontré un obstacle auquel nous n'avons pu remédier, que ça soit seul ou avec l'aide de nos camarades. Il nous est impossible d'envoyer plus de 3 variables uniformes aux shaders. Cela nous a notamment empêcher de réaliser la fonctionnalité d'ajout de texture.

C) RADIAL BASIS FUNCTIONS

Nous avions implémenté une première version fonctionnelle des RBF. Cette version donnait un résultat similaire au résultat attendu mais le raisonnement n'était pas le bon.

Nous avons donc revu cette partie du code et utilisé la première version pour créer l'outil Brush.

D) LUMIÈRES

Pour les lumières nous avons commencé par implémenter la lumière directionnelle.
Pour cela nous avions comme idée de suivre un TP d'OpenGL et d'implémenter la fonction et la méthode de *Blinn–Phong*. Nous avions eu plusieurs problèmes lors de cette mise en place, notamment à cause du problème posé par notre restriction de variables uniformes.

Cela nous a pris beaucoup de temps suite à quoi nous avons décidé de changer de méthode avec l'aide d'un de nos camarades.

VII) CONCLUSION

La programmation n'étant pas ce qui me fait me lever le matin, j'ai eu du mal à me motiver, mais le fait de voir des résultats s'afficher, de comprendre où sont les erreurs et comment les corriger m'a permis de me plonger dans ce projet. Il m'a semblé être la continuité des cours et TDs suivis ce semestre. Il m'a pu paraître complexe et inaccessible au début. Cependant la clarté du sujet et des TDs a simplifié les choses, contrairement à l'année dernière par exemple. Nous avons codé l'intégralité du projet à deux, ce qui nous a permis de gagner en efficacité en corrigeant le code de l'autre au fur et à mesure. Cela nous a également permis de gagner du temps. Même si évidemment il y a des manquements et des lourdeurs, nous sommes satisfaits du programme final, nous ne pensions pas pouvoir livrer une application aussi aboutie. Johan Boyer	Ce projet était satisfaisant dans le sens où cela m'a permis de mettre en place et d'appliquer des notions et connaissances acquises en code pour un contexte concret. J'ai réellement été motivée par ce projet et cela m'a permis de laisser de côté mes lacunes et ma perte d'intérêt pour la programmation. Ayant le même niveau de base en code avec mon camarade, nous avons réellement eu la sensation d'avoir progressé par rapport à l'année dernière où la réalisation du projet était plus laborieuse faute de connaissances. Je suis satisfaite car nous sommes parvenu à rendre une application terminée avec toutes les fonctionnalités essentielles implantées, ce que je ne pensais pas être capable de faire au premier abord ou du moins beaucoup plus difficilement. Effectivement nous avons dû faire face à des difficultés mais nous avons réussi à avancer malgré cela sans se décourager en apportant des solutions et en s'adaptant. Elisa Ciavaldini