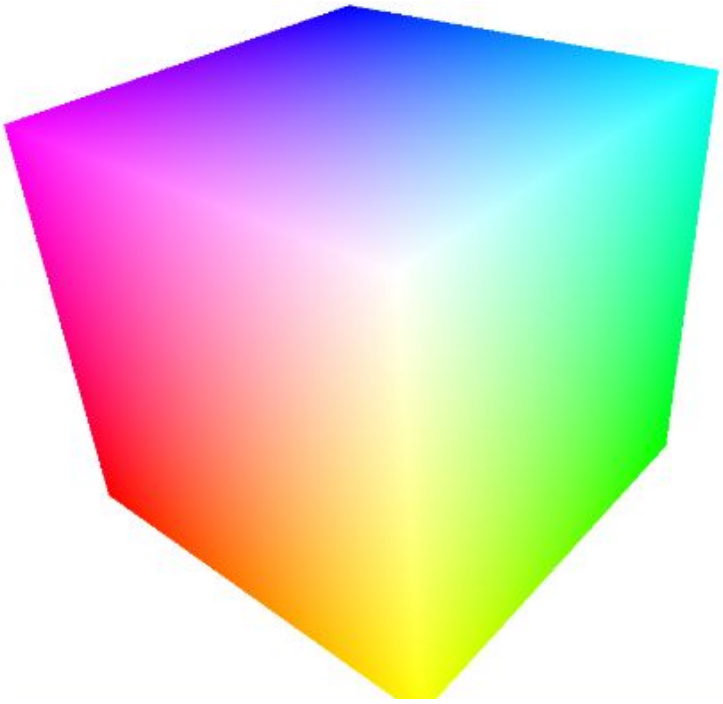
WorldIMaker

Maurel Victorine - Séguy Margaux



Rapport de projet

Introduction

Ce projet alliant programmation, math-infos et synthèse de l'image a pour objectif de faire réaliser un éditeur-visualiseur de terrain et de scène en 3D. Nous avions un cahier des charges complet et précis avec des fonctionnalités à implémenter. Notre binôme se constitue de Victorine Maurel et Margaux Séguy. Nous avons commencé à travailler sur ce projet peu de temps après avoir reçu les consignes. D'abord en réfléchissant "sur le papier" à la structure du projet et à l'implémentation de l'application, puis en passant à la partie code. La répartition des tâches s'est faite assez naturellement, nous avons donc travaillé chacune sur des parties différentes du projet. Nous avons mis en place un répertoire GitHub pour synchroniser notre travail et voir l'avancée du projet dans sa globalité. Ce projet, bien que très intéressant, nous a demandé énormément d'heures de travail.

Plusieurs fonctionnalités telles que l'affichage d'une scène, l'édition des cubes, la sculpture du terrain (create, delete, extrude et dig), la génération procédurale et l'ajout de lumières étaient indispensables. Deux fonctionnalités supplémentaires étaient requises, nous avons choisi sauvegarde/chargement de la scène et chargement de modèle 3D. Cette dernière n'est pas fonctionnelle mais est implémentée, nous reviendrons là dessus plus tard dans le rapport.

Pour faciliter notre travail et permettre une meilleure compréhension du projet nous avons décidé de mettre en place un schéma montrant explicitement la relation entre les différentes fonctions, schéma que nous complétions au fur et à mesure de l'avancée du projet.

L'entièreté du code a été documenté à l'aide de la librairie *doxygen* pour générer la documentation il suffit de faire : make html.

1) Récapitulatif des fonctionnalitées

Fonctionnalité	Implémentée intégralement	Implémentée partiellement	Non implémentée	Précision
Affichage d'une scène avec des cubes	X			
Déplacement dans la scène (Trackball)	X			
Edition des cubes (Curseur)	X			
Sculpture du terrain	X			
Génération procédurale	X			
Ajout de lumière		X		L'utilisateur ne peut pas placer ou déplacer les lumières dans la scène
Sauvegarde/ chargement de la scène	X			L'utilisateur peut choisir où sauvegarder et quel fichier charger via le terminal
Chargement de modèles 3D		X		Implémentée mais ne fonctionne pas
Librairie ImGui			X	

2) Choix d'implémentations

Pour réaliser notre application nous sommes parties de la librairie glimac avec laquelle nous avons travaillé pendant les TP. Nous avons choisi de découper notre code en plusieurs entités en fonction du design pattern de forte cohésion : c'est à dire que nous avons créé plusieurs entités qui contiennent un code cohérent, facile à comprendre et facilement maintenable. Nous avons donc une entité pour chaque "objet" nécessaire à l'éditeur de cube.

Cube : la création d'un cube est inspirée du design pattern *State*, en effet tous les cubes sont présents sur la grille de manière visible ou invisible, c'est donc l'état du cube qui change. De plus, selon si le cube est visible ou non, les différentes méthodes n'ont pas le même effet sur lui. Le cube est créé grâce à un tableau d'indices et des coordonnées de sommets. Pour optimiser le code et éviter la répétition des vertices nous avons utilisé un Index Buffer Object.

C'est la classe *Cube* qui gère la création des cubes. Elle contient un constructeur par défaut pour placer le cube à la position donnée et un constructeur par recopie avec une référence sur le cube. Elle contient aussi des méthodes pour afficher un cube et pour changer la couleur en fonction des paramètres choisis.

Affichage d'une scène avec des cubes : L'affichage de la scène est gérée grâce à la projection des matrices. Par défaut, la scène est composée d'un solide de 20*20*3 cubes soit 1200 cubes au total. A l'origine ce solide est centré dans la fenêtre principale mais il est possible de le déplacer grâce à un mouvement de caméra associé aux touches q d et s z.

C'est la classe *Scene* qui gère la création de la scène. Elle contient notamment une méthode *createAllCubes* qui crée tous les cubes de la scène qu'ils soient visibles ou non, *displayCubes* se charge d'afficher les cubes de la scène et *getCubeAtThisPos* donne l'indice du cube à la position donnée. Enfin il y la méthode de création des variables uniformes pour faire le lien avec les shaders et les méthodes update des matrix pour mettre à jour ces variables uniformes.

Déplacement dans la scène (*Trackball*) : le déplacement dans la scène se fait grâce à une trackball caméra. Elle peut tourner autour du centre de la scène, s'en rapprocher, s'en éloigner et se déplacer de manière verticale et horizontale. Pour cela, on utilise un angle X et un angle Y pour tourner autour de la scène verticalement et horizontalement, ainsi que des distances pour zoomer et déplacer la caméra. Le zoom/dézoom et les rotations sont gérés par la souris tandis que les déplacements sont gérés par les touches q s et z d .

C'est la classe *TrackballCamera* qui gère la caméra et ses mouvements. Elle contient des méthodes pour déplacer la caméra horizontalement et verticalement, pour changer l'angle horizontal et vertical de la caméra et également pour zoomer ou dézoomer. Il est possible de remettre ses paramètres initiaux à la caméra.

Edition des cubes (Curseur) : l'édition des cubes se fait à l'aide d'un curseur que l'on peut déplacer dans la scène grâce aux flèches directionnelles. Le curseur agit comme un *itérateur*

en fonction du changement de position qu'on lui applique. A l'origine le curseur est placé au centre de la scène.

C'est la classe *Cursor* qui gère l'affichage du curseur. Cette classe hérite de *Cube* et contient une méthode qui initialise le curseur à la position passée en paramètre ainsi qu'une méthode pour afficher le curseur dans la scène.

Sculpture du terrain : chaque cube peut subir quatre transformations add, remove, extrud et dig. Ces différentes transformations agissent sur l'*état* du cube et sa visibilité. Add et remove agissent directement par rapport à la position du curseur, alors que extrud et dig agissent sur les cubes du haut de la colonne dans laquelle se trouve le curseur. Il est également possible de changer la couleur du cube sur lequel se trouve le curseur, pour cela il faut entrer les nouvelles valeurs RGBA dans le terminal.

C'est la classe *Scult* qui gère les transformations appliquées à la scène. Il y a donc une méthode par transformation. Ces différentes méthodes sont appliquées en fonction de la position du curseur dans la scène.

Génération procédurale : une scène peut être générée de manière procédurale, pour cela on utilise des radial basis function qui nous permettent d'extrapoler des valeurs, ces valeurs sont par la suite comparées à 0, si la valeur est supérieure à 0 le cube est dessiné sinon la case est laissée vide.

Pour cela tout se passe dans la classe *Generate*, *readControlPoints* parcourt le fichier où sont stockés les points de contrôle afin de les stocker pour pouvoir les utiliser par la suite.
La suite se passe en plusieurs étapes :

- La méthode omega : génère les valeurs d'interpolation, en résolvant un système
- RBF : met en place les fonctions radiales qui servent pour la génération finale
- Quant à elle la méthode *applyRBF* applique les fonctions radiales à notre scène : si la valeur finale pour un point spécifique est supérieure à 0 un cube est généré sinon il ne l'est pas.

Ajout de lumière : il y a deux sortes de lumière dans la scène. Une lumière directionnelle et une lumière ponctuelle. Elles sont appliquées directement dès le lancement du programme. Pour passer du mode jour au mode nuit il suffit de cliquer sur n, pour rétablir le mode jour on clique sur j.

Les lumières sont calculées par le fragment shader. Elles sont basées sur le modèle de réflection de *Bling Phong*.
Les différentes variables nécessaires sont mises en place dans la classe *Scene*. Une autre méthode permet de les appliquer, en envoyant les valeurs des variables mises en place précédemment au fragment shader. Un mode jour/nuit est également disponible (accessible par les touches j et n).

Sauvegarde/chargement de la scène : il est possible de sauvegarder la scène et charger dans la fenêtre une scène déjà sauvegardé. Pour cela il suffit respectivement d'appuyer sur le

f ou le v. Il est demandé par la suite de donner, dans le terminal, le nom du fichier dans lequel on souhaite enregistrer la scène ou le fichier comportant la scène que l'on souhaite ouvrir.

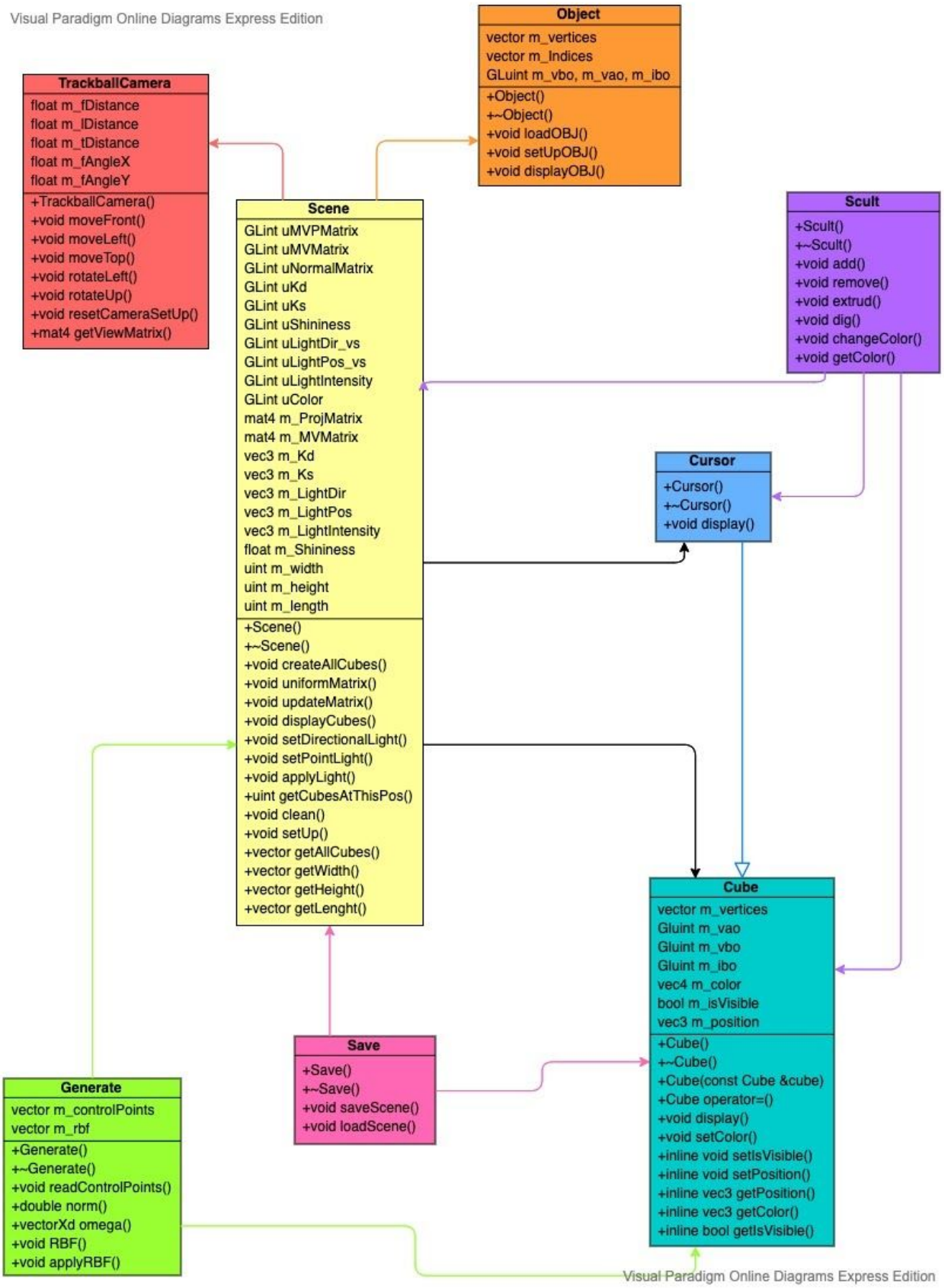
Cette fonctionnalité est gérée dans la classe *Save*, qui comprend deux méthodes une pour la sauvegarde et l'autre pour le chargement.
La fonction de sauvegarde édite un fichier dans lequel elle stocke toutes les informations relatives à la scène : les cubes visibles, ceux invisibles et leurs couleurs.
La méthode de chargement parcourt le fichier et restitue la scène, en modifiant le *vector* de cube qui correspond à tous les cubes de notre scène.
Elle parcourt l'ensemble de la scène et remplace les valeurs par défaut par celles stockées dans le fichier.

Chargement de modèles 3D : cette fonctionnalité a été implémentée mais ne fonctionne pas. Le but étant d'importer un modèle 3D dans notre scène.

Dans le sujet il était conseillé d'utiliser la librairie *assimp*, mais nous avons préféré créer nos propre méthodes afin de simplifier la démarche, le fonctionnement d'assimp restant assez flou malgré la recherche de documentation. En effet nous avons remarqué que chaque fichier *.obj* était construit de la même manière, nous avons donc créé une fonction qui le parcourt et qui stocke les différentes valeurs présentes dans le fichier, tels que la position des sommets.
Une autre méthode initialise les buffers, et enfin une dernière gère l'affichage de l'objet.

Par manque de temps nous avons préféré laisser de côté le chargement de modèle 3D.

Librairie ImGui : cette librairie permet de développer des menus. Nous avions implémenté ImGui pour afficher un menu dans notre fenêtre principale. Malheureusement nous n'avons pas réussi à la faire fonctionner correctement. Nous avons préféré la retirer totalement du projet pour ne pas laisser de code mort et éviter des bugs.


Visual Paradigm Online Diagrams Express Edition
Object
vector m_vertices
vector m_Indices
GLuint m_vbo, m_vao, m_ibo
+Object()
+~Object()
+void loadOBJ()
+void setUpOBJ()
+void displayOBJ()
TrackballCamera
float m_fDistance
float m_lDistance
float m_tDistance
float m_fAngleX
float m_fAngleY
+TrackballCamera()
+void moveFront()
+void moveLeft()
+void moveTop()
+void rotateLeft()
+void rotateUp()
+void resetCameraSetUp()
+mat4 getViewMatrix()
Scene
GLint uMVPMatrix
GLint uMVMatrix
GLint uNormalMatrix
GLint uKd
GLint uKs
GLint uShininess
GLint uLightDir_vs
GLint uLightPos_vs
GLint uLightIntensity
GLint uColor
mat4 m_ProjMatrix
mat4 m_MVMatrix
vec3 m_Kd
vec3 m_Ks
vec3 m_LightDir
vec3 m_LightPos
vec3 m_LightIntensity
float m_Shininess
uint m_width
uint m_height
uint m_length
+Scene()
+~Scene()
+void createAllCubes()
+void uniformMatrix()
+void updateMatrix()
+void displayCubes()
+void setDirectionalLight()
+void setPointLight()
+void applyLight()
+uint getCubesAtThisPos()
+void clean()
+void setUp()
+vector getAllCubes()
+vector getWidth()
+vector getHeight()
+vector getLenght()
Scult
+Scult()
+~Scult()
+void add()
+void remove()
+void extrud()
+void dig()
+void changeColor()
+void getColor()
Cursor
+Cursor()
+~Cursor()
+void display()
Cube
vector m_vertices
Gluint m_vao
Gluint m_vbo
Gluint m_ibo
vec4 m_color
bool m_isVisible
vec3 m_position
+Cube()
+~Cube()
+Cube(const Cube &cube)
+Cube operator=()
+void display()
+void setColor()
+inline void setIsVisible()
+inline void setPosition()
+inline vec3 getPosition()
+inline vec3 getColor()
+inline bool getIsVisible()
Save
+Save()
+~Save()
+void saveScene()
+void loadScene()
Generate
vector m_controlPoints
vector m_rbf
+Generate()
+~Generate()
+void readControlPoints()
+double norm()
+vectorXd omega()
+void RBF()
+void applyRBF()
Visual Paradigm Online Diagrams Express Edition

3) Actions et commandes associées

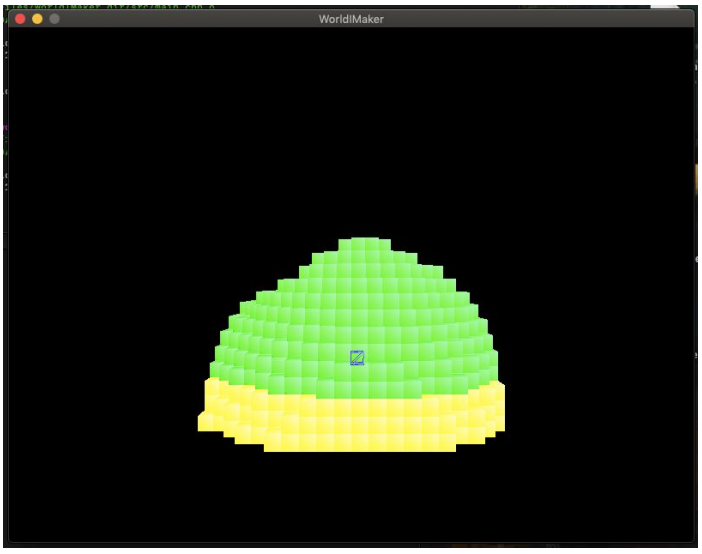
Actions	**Commandes associées**
Déplacer le curseur	Flèches + w et x pour l'axe z
Déplacer la caméra	Touche q d et z s
Zoom	Molette de la souris
Change l'angle de la caméra	Clic + souris
Réinitialiser la caméra	Touche g
Sauvegarder la scène	Touche f et entrer le nom du fichier
Charger une scène	Touche v et entrer le nom du fichier
Supprimer la scène	Touche c
Réinitialiser la scène	Touche b
Mode jour/nuit	Touche n
Ajouter un cube	Touche r
Supprimer un cube	Touche t
Extruder	Touche u
Dig	Touche i
Changer la couleur du cube	Touche l et entrer les valeurs RGBA
Générer un cactus	Touche p
Générer un igloo	Touche m

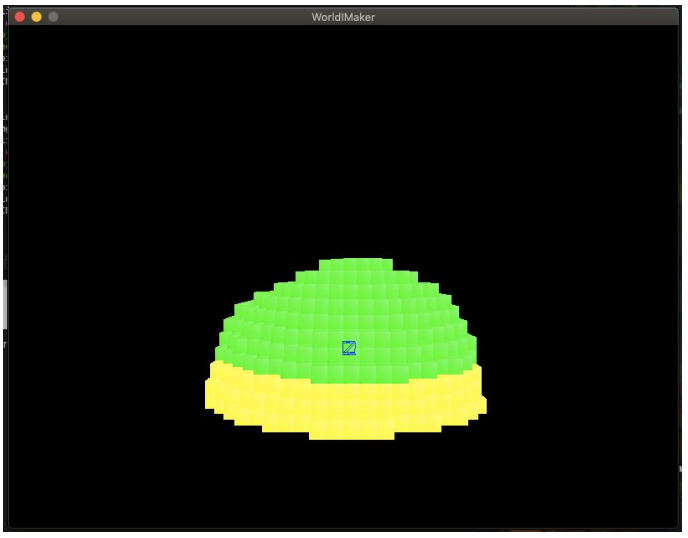
4) Générations procédurales

La résultat de la génération procédurale dépend de nombreux paramètres : les différents points de contrôles choisis, la valeur de l'epsylone et évidemment la fonction radiale choisie.

Afin d'illustrer ces changements nous allons vous montrer différentes générations basées sur les mêmes points de contrôle mais avec des fonctions radiales différentes et des epsylones différents.

Fonction multiquadrique

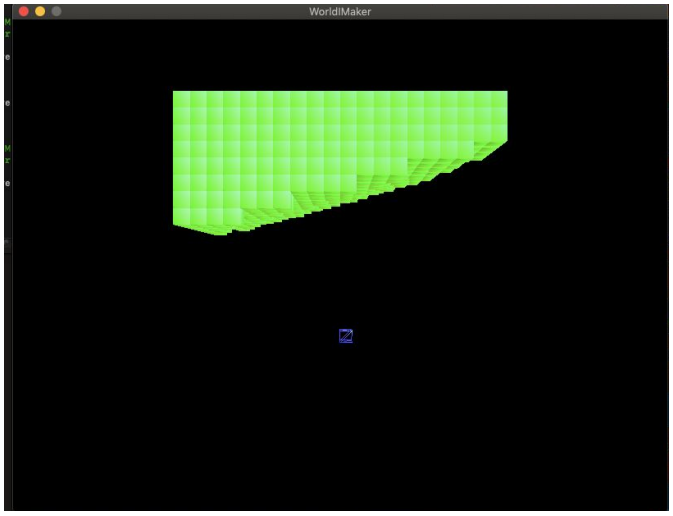


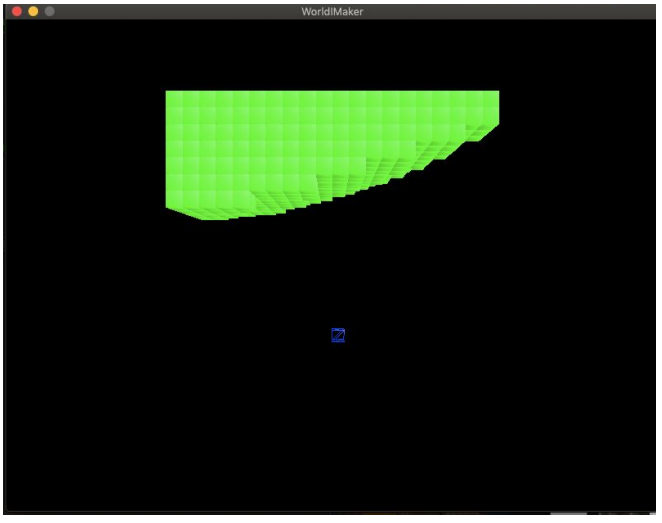


Epsylone = 1 Epsylone = 0.2

Dans les deux cas les cubes forment un dôme, dans la cas où epsylone est le plus petit le cube est grand verticalement parlant.

Fonction inverse quadrique

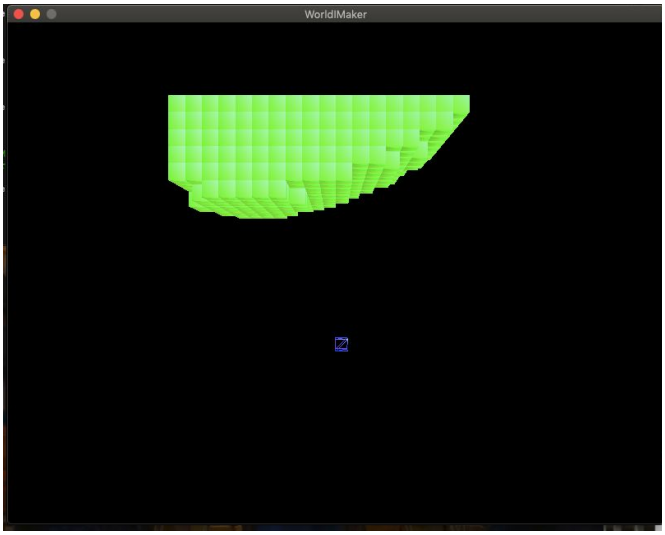


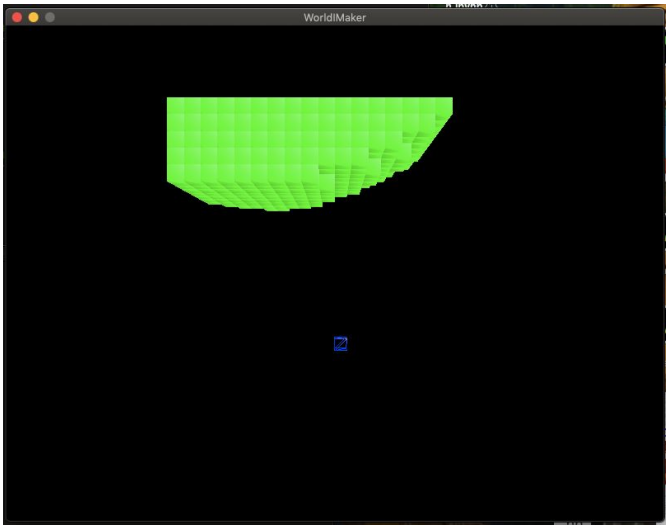


Epsylone = 1 Epsylone = 0.2

On constate une grosse différence entre l'application de la fonction multiquadrique et celle de la fonction inverse quadrique, ce qui est plutôt logique puisque les fonctions sont quand même plutôt différentes. La chose que nous pouvons remarquer est que lorsque l'on a affaire à une fonction inverse les cubes sont tous générés dans la partie supérieure de la scène.
En ce qui concerne la variation de l'epsylone comme dans le cas d'avant moins de cubes sont générés quand epsylone est égal à 1.

Fonction inverse multiquadrique





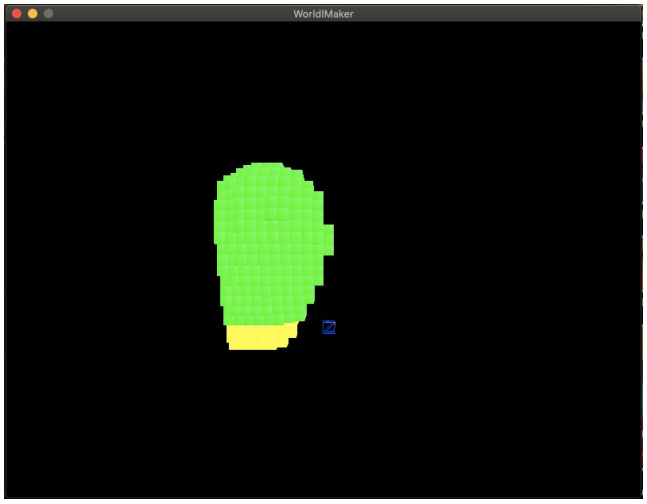
Epsylone = 1

Epsylone = 0.2

On constate une légère différence de génération lorsque l'on utilise une fonction inverse multiquadrique plutôt que la fonction inverse quadrique. En effet dans ce cas là, on peut observer une sorte d'arrondi au bas de l'amas de cube.
Comme dans les cas précédents plus l'epsylone est grand plus il y a de cubes.

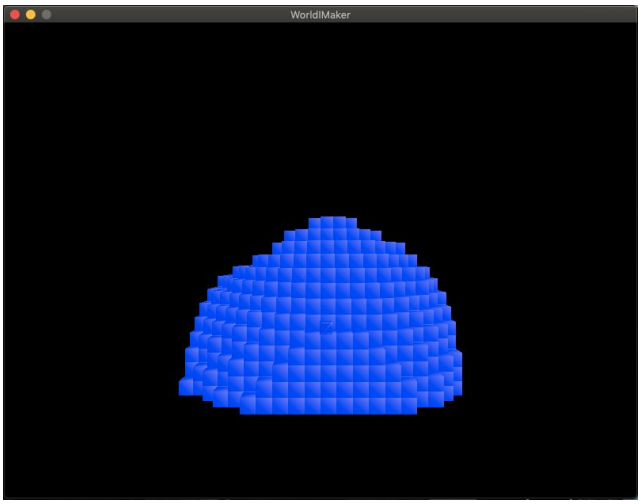
Dans le cadre de notre application nous avons choisi deux générations différentes.

- La première que nous avons surnommé *cactus* :



Cette génération dépend de 8 points de contrôle (qui ont tous des positions et des poids différents), et l'utilisation de la fonction multiquadrique.

- La deuxième que nous avons surnommé *iglou* :



Dans ce cas là la génération dépend de 3 points de contrôle qui ont des caractéristiques différentes. Comme dans le cas précédent c'est la fonction multiquadrique qui est utilisé.

5) Difficultés rencontrées

Ce semestre et surtout la fin de l'année ont été assez dense avec de nombreux projets à réaliser. Il a donc fallu jongler équitablement entre tous ces projets pour pouvoir proposer des rendus corrects dans toutes les matières et également consacrer du temps à la révision des partiels.

La principale difficulté que l'on a rencontré pour réaliser ce projet est donc le temps imparti. Bien que nous ayons eu le sujet relativement tôt, nous n'avions pas fini les TP de synthèse d'image ce qui nous bloquait dans l'avancée du projet. De ce fait , même si nous avons réussi à implémenter une majorité des fonctionnalités principales, nous avons manqué de recul sur le travail réalisé. Nous sommes donc conscientes que notre code peut être optimisé. Nous avons préféré coder toutes les fonctionnalités de manières imparfaite, plutôt que d'en implémenter que quelques unes de manière plus aboutie.

La seconde difficulté à laquelle nous avons été confrontées est l'outil de travail. En effet nos ordinateurs personnels ne sont pas optimisés pour travailler sur de tel projet, ce qui fait que la fenêtre beugue rapidement après quelques actions sur la scène. De plus comme nous n'avons pas codé sur les machines de la fac nous nous sommes aperçues au dernier moment que lorsque nous lancions le programme sur ces dernières le rendu n'était pas le même que sur nos machines personnelles.

Enfin la troisième et dernière difficulté rencontrée est l'utilisation d'outils tels que GitHub ou ImGui pour lesquels nous n'avons eu que très peu d'informations. La documentation et les erreurs liées nous a fait perdre un temps précieux.

6) Améliorations possibles

Bien que notre projet nous semble relativement abouti nous sommes conscientes que des améliorations sont possibles.

- les lumières : pouvoir choisir l'emplacement des différentes lumières et notamment pouvoir positionner plusieurs points de lumières. Mais également créer des buffers indépendants pour la lumière, pour avoir une lumière ambiante et pas une lumière par cube.

- chargement des modèles 3D : utiliser la librairie *assimp* pour implémenter cette fonctionnalité.

- librairie ImGui : implémenter cette librairie pour développer un menu et faciliter l'utilisation du WorldIMaker, en effet nous avons conscience que l'utilisation du clavier n'est pas optimale au niveau de l' expérience utilisateur.

- de manière générale optimiser le code et les fonctions.

7) Retour individuel

“Ce projet a été pour moi une sorte de réconciliation avec l’OpenGL, les TD de synthèse d’image de ce semestre nous ont vraiment permis d’avoir des bases solides pour la réalisation de l’éditeur de cube. Chaque TD étaient détaillés afin que l’on comprenne les notions importantes et Céline toujours présente pour nous aider en cas d’incompréhension, c’était très agréable.
Ça a également été une manière de constater notre progression en programmation. Évidemment le rendu final n’est pas parfait et c’est toujours un peu frustrant de rendre quelque chose qui n’est pas tout à fait abouti, mais j’espère qu’il traduit le travail fourni. En tout cas moi j’en suis plutôt fière.”

- Victorine

“Bien que la programmation ne soit pas mon principale intérêt, j’ai trouvé ce projet plus intéressant et abordable que le précédent. De plus, le fait d’avoir eu de vrais TP de synthèse d’image ce semestre nous a permis de développer nos capacités et d'accroître notre intérêt pour cette matière. Nous pouvons remercier Céline pour le temps qu’elle nous a consacré. Ce projet était l’occasion de mettre à profit ce que nous avons appris durant le semestre. Bien que le rendu ne soit pas totalement terminé, notamment par manque de temps, je suis entièrement satisfaite de ce que nous vous proposons.”

- Margaux