DURAND Zoé
ROSENBERG Maéva

Projet S3

Synthèse d'image / Programmation objet / Mathématiques

World IMAKER

Sommaire

Introduction

Dans le cadre d'un projet transversal entre les cours de synthèse d'image, de programmation objet et de maths, nous avons produit une application qui permet de créer un monde à partir du cube. Nous avons imaginé ce programme comme un programme de voxel art. Comme pour le pixel art mais en 3 dimensions, l'utilisateur peut choisir la taille de son monde, créer et supprimer des cubes dans ce monde, modifier la couleur de ces cubes, tourner autour de la scène, l'enregistrer, charger un monde qu'il avait créé auparavant ou générer un nouveau monde.

Ce logiciel a été réalisé en C++, OpenGL avec les librairies ImGui et GLImac.

I - Manuel d'utilisation

Compilation

Il est conseillé de compiler le programme dans un dossier Imaker-build situé au même emplacement que le dossier Imaker. Le programme se compile avec les lignes :

cmake ../Imaker
make

Pour l'exécution du programme : **./UP/UP_main 3D.vs.glsl dirligthPhong.fs.glsl**

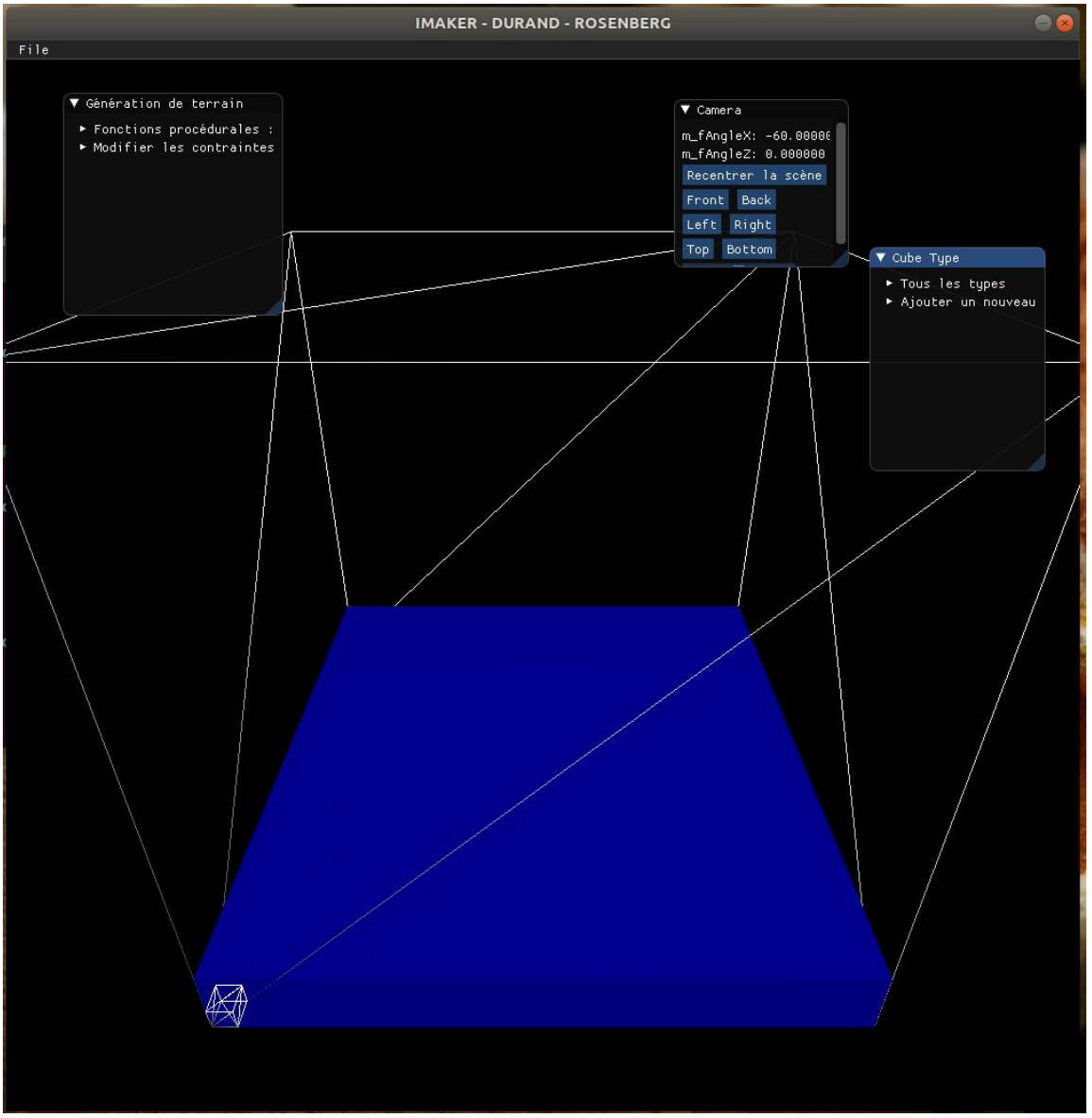

figure 1. vision au lancement

Le logiciel s'ouvre automatiquement sur un monde de 25*25*25 cubes, dont les 3 premiers niveaux sont pleins. Dans la fenêtre [nom de la fenêtre], deux types de cube existent déjà par défaut : Bleu et Blanc.

Caméra

La caméra est également à son emplacement par défaut. Si l'utilisateur tourne autour de la scène, il peut retrouver cet emplacement par défaut en cliquant sur le bouton "Recentrer la scène" dans la fenêtre Caméra. Il peut également placer la caméra aux 6 emplacements automatiques prévus : Front, Back, Left, Right, Top, Bottom. Ces placements automatiques sont inspirés des logiciels de modélisation 3D qui ont en général les mêmes.

L'utilisateur peut tourner autour de la scène en maintenant le clic gauche de sa souris enfoncé. Il peut zoomer avec la molette de sa souris ou les flèches haut et bas de son clavier.

Curseur

Le curseur permet de visualiser l'emplacement où l'utilisateur peut effectuer ses actions. Ce curseur est toujours visible même si son emplacement est caché par d'autres cubes. Pour déplacer son curseur, il doit utiliser les touches du clavier numériques : le 2 et le 8 permettent de se déplacer d'avant en arrière, le 4 et le 6 de gauche à droite et le 9 et le 1 de haut en bas. Cependant le sens de déplacement du curseur ne change pas lorsque l'utilisateur tourne autour de la scène : s'il visualise la scène depuis l'arrière, il verra le curseur se déplacer en miroir par rapport à l'avant de la scène.

Modélisation

Les actions de modélisation se font également au clavier. Ainsi, la touche C permet de créer un cube à l'emplacement du curseur, et la touche V permet de vider cet emplacement. Le cube n'est cependant pas supprimé (voir gestion du monde), s'il est recréé juste après il aura conservé sa couleur. La touche D permet de creuser la colonne de la position du curseur (Dig) et la touche E permet de rallonger cette colonne vers le haut (Extrude).

Changer la couleur d'un cube se fait en allant sélectionner un type de cube dans la fenêtre dédié à cet usage. Cette même fenêtre permet à l'utilisateur de créer

lui-même ses types de cubes, en sélectionnant une couleur et en lui attribuant un nom. Si l'utilisateur sauvegarde son fichier, les types seront enregistrés avec tout le reste de la scène pour que les cubes conservent leur couleur sur-mesure à la réouverture du fichier.

Sauvegarder

L'enregistrement et l'ouverture d'un fichier passent d'ailleurs par la barre de menu en haut de la fenêtre, onglet "File". En faisant juste "Save", le fichier s'enregistre sous son nom actuel (NewFile.imaker s'il n'a pas été changé depuis l'ouverture du nouveau monde). Avec "Open World", l'utilisateur peut ouvrir une fenêtre de recherche de fichiers dans les documents de son ordinateur. La fenêtre est paramétrée de façon à ce que n'apparaissent que les fichiers .imaker, qui est le format que nous avons créé pour ce projet (cf enregistrement des fichiers). En sélectionnant un fichier .imaker et en cliquant sur "OK", l'utilisateur ouvre le fichier en question. En choisissant "New World", il peut créer un nouveau monde vierge et même choisir sa hauteur, sa longueur et sa largeur (entre 3 et 50 cubes pour chaque). Avec "SaveAs", il peut changer le nom de son fichier qui s'enregistrera sous ce nouveau nom. Si ce nom existe déjà, une fenêtre d'avertissement demandera confirmation à l'utilisateur pour écraser le fichier déjà existant.

Afin d'améliorer encore l'expérience utilisateur, on pourrait ajouter un historique des modifications du monde, ce qui permettrait d'annuler une action, la refaire ou vérifier que le fichier a bien été sauvegardé et qu'aucune modification n'a été faite depuis avant de fermer le fichier.

II - Détail des fonctionnalités

a - Vue globale

Fonctionnalité	Implémentée	Commentaires
Affichage d'une scène avec des cubes	Oui	
Edition des cubes : - curseur de sélection qui se déplace - modifier couleur / type du cube	 Oui Oui	Il faut créer un nouveau type avec la couleur de son choix avant de changer le type du cube.
Sculpture : - Ajouter un bloc - Supprimer un bloc - Extrude - Dig	 Oui Oui Oui Oui	
Génération procédurale	Oui	
Lumières	En partie	Les shaders des lumières existent et fonctionnent, mais les points de lumières ne peuvent pas être ajoutés ou retirés à volonté
Sauvegarde / Chargement de la scène	Oui	
Amélioration de la sélection	Commencé	Manque de temps pour finir

b - Gestion des cubes et du monde

La classe Cube permet de créer un cube, l'afficher, le modifier, le cacher ("vider" son emplacement pour l'utilisateur). Un monde est composé de quelques-uns à plusieurs centaines de cubes. Nous avons donc fait le choix de stocker l'espace de création sous la forme de trois vecteurs imbriqués de cubes. Chaque cube est rangé dans les vecteurs comme il l'est dans le monde : un cube à la position x = 2, y = 15 et z = 25 est placé dans le vecteur allCubes[2][15][25];

Ainsi, tous les cubes possibles sont créés en même temps que le monde, ce qui fait que le chargement d'une scène importante peut être un peu long. Chaque cube à un attribut bool visibility qui permet, en le modifiant, de faire apparaître (créer) ou cacher (vider) le cube. Si cela ralenti le chargement d'une scène, il permet ensuite à l'utilisateur de modifier son monde avec fluidité puisque l'accès à un cube se fait immédiatement (compléxité O(1)). L'emplacement du cube dans le vecteur est donc directement déterminé par l'emplacement du curseur de sélection.

La classe World a donc pour fonction principale de contenir tous les cubes du monde. Elle permet de tous les créer et les afficher.

c - L'interface avec Imgui

Pour l'interface, nous avons utilisé comme recommandé la librairie ImGui. Ceci a été notre première vraie difficulté : la librairie est très complète et utile, mais longue à la prise en main.

La création d'un objet interface initialise le contexte SDL qui permet d'afficher le programme. Les fonctions startFrame() et render() permettent d'afficher les fenêtres ou barre de menu, qui sont chacun.e.s défini.e.s dans une fonction différente pour plus de lisibilité. Ainsi, on retrouve la fonction MainMenuBar() directement tirée de la librairie ImGui, browserFile() qui permet d'appeler une fenêtre pour se déplacer dans les dossiers de l'ordinateur et sélectionner un fichier .imaker, createNewWorldWindow() qui est la fenêtre de création d'un nouveau monde, saveWindow(), overwriteWindow() et saveAsWindow() qui sont les fenêtres pop-up qui apparaissent lors de l'enregistrement du fichier selon les circonstances.

d - L'enregistrement des fichiers

Nous avons choisi en implémentation bonus de pouvoir enregistrer le monde et en charger un autre.

La classe File, ainsi que la manière dont sont enregistrés les fichiers .imaker sont en grande partie inspirés du travail similaire que nous avions à faire pour l'Imac Tower Defense de première année. Ainsi, un fichier .imaker à la syntaxe suivante :

```
@Imaker                          -> ceci est un fichier pour le logiciel Imaker
World 25 25 25                   -> La longueur, largeur et hauteur du monde
3                                -> le nombre de type de cube dans ce fichier
Type 0 0 0 255 Bleu              -> premier type de cube (défaut)
Type 1 255 255 255 Blanc         le 3 permet de faire une boucle for pour les
Type 2 99 138 151 autrecouleur   types de cubes à enregistrer
15625                            -> nombre total de cubes dans la scène :
Cube 0 0 0 true 1                    on vérifie que ce nombre correspond à
Cube 1 0 0 true 0                    longueur * largeur * hauteur du monde
Cube 2 0 0 true 2                    puis on fait une boucle for qui push chaque
[...]                                cube à sa position [x][y][z] avec true/false
Cube 24 24 24 false 0                pour sa visibilité et le dernier chiffre
end                                  correspond à son type qui existe forcément
                                     déjà.
                                     -> On finit par end pour être sûr qu'il n'y a
                                     pas plus de cubes que prévu à l'origine
```

La fonction pour ouvrir un fichier créé un monde à partir de toutes ces informations, tandis que la fonction sauvegarder écrit ou réécrit un fichier avec la même syntaxe.

e - Radial Basis Functions (RBF)

Grâce à l'interface, l'utilisateur peut choisir parmi fonctions radiales laquelle il souhaite utiliser. Il peut également ajouter des contraintes en choisissant la position d'un cube de l'espace de création.

Pour gérer les radial basis functions, nous avons créé une classe InterpolationFunc contenant les différentes entitées à prendre en compte (positions, poids, vecteur w). Nous avons pris le parti de faire des height map, nous traitons donc des vecteurs à deux dimensions possédant un poid. Ainsi lorsque l'utilisateur ajoute un point de contrainte de coordonnées (x, y, z), x et y représentent la position de la contrainte et z est son poid.

La classe peut ensuite calculer w grâce à un pivot (librairie Eigen) et une fonction de calcul d'interpolation permet de trouver la valeur z associée à chaque couple (x, y). On rempli ensuite l'espace de construction grâce à la fonction extrude et on obtient un terrain taillé sur mesure.

Un objectif que nous n'avons pas eu le temps d'implémenter mais qui fera prochainement parti du projet et de pouvoir utiliser les RBF sur des vecteurs 3 dimensions. Nous pourrions ainsi créer des environnements plus atypiques (boule au milieu de la scène par exemple).

Pour analyser le comportement des différentes RBF utilisées dans notre programme, nous les avons étudiées dans des conditions équivalentes : 3 points de contrôles assez différents et un alpha égal à -1.

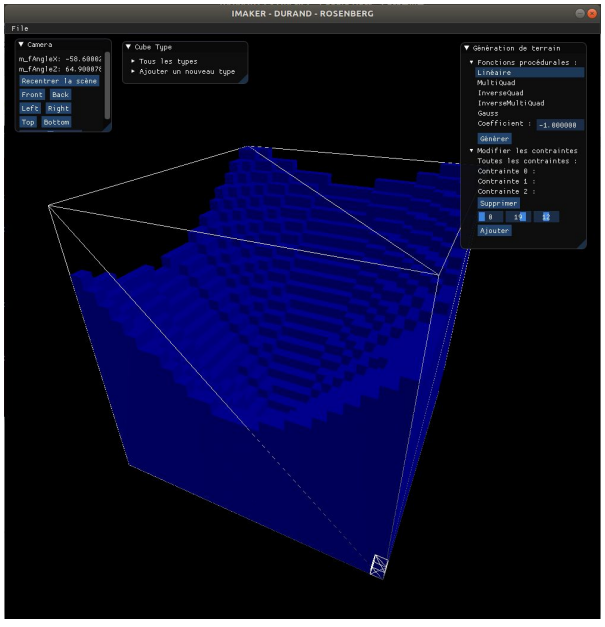

figure 2. RBF linéaire

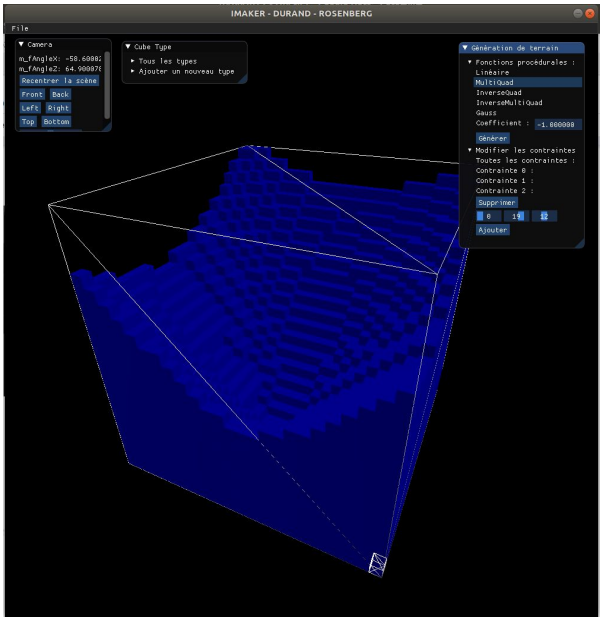

figure 3. RBF multi quadratique

La RBF **linéaire** (f(x)=alpha*x) et la RBF **multi quadratique** (f(x)=sqrt(1+(alpha*x)²) se comportent de manière similaires mais la linéaire est plus aplatie (légèrement sur cet exemple) que la multiquadratique. Quoi qu'il en soit, elles ressemblent plus à des trous ou des plaines (si les poids des coefficients sont moins distants) pour l'utilisation qu'en fera Toto.

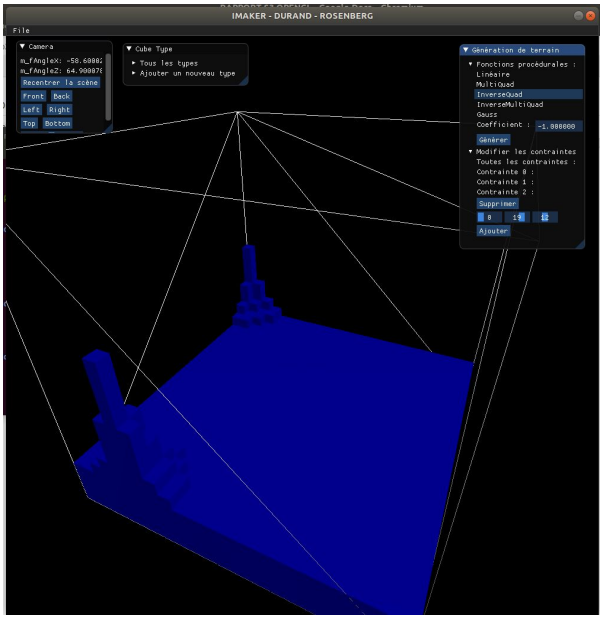

figure 4. RBF inverse quadratique

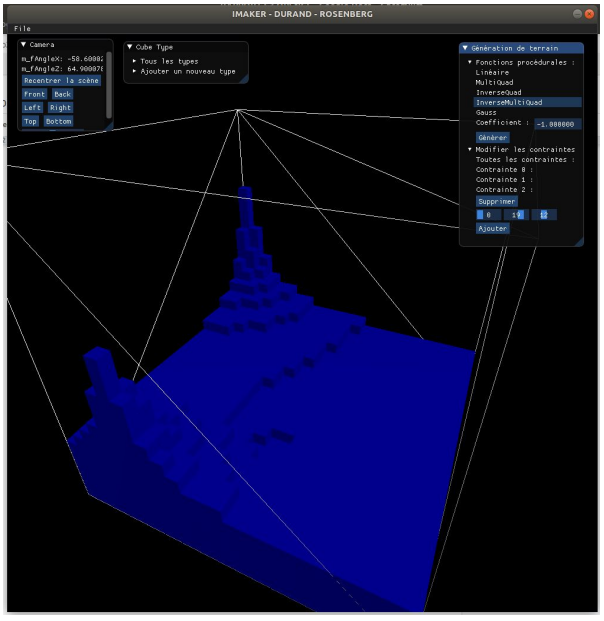

figure 5. RBF inverse multi quadratique

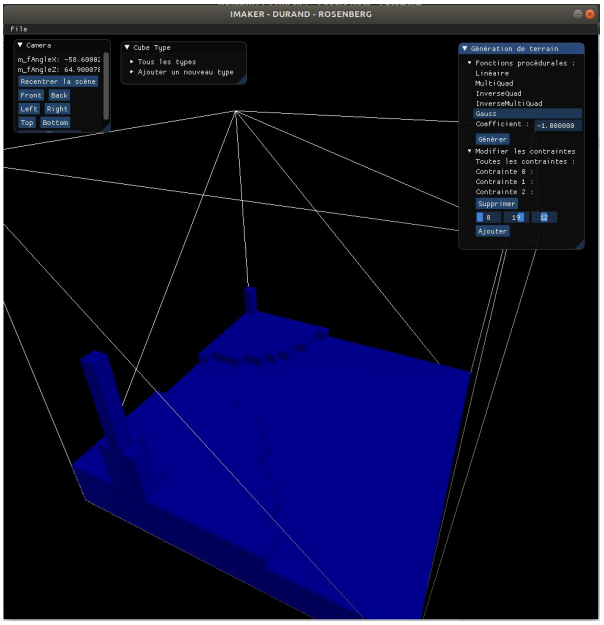

figure 6. RBF Gaussienne

Les RBF **inverse quadratique** (f(x)=1/(1+(alpha*x)²)), **inverse multi quadratique** (f(x)=1/sqrt(1+(alpha*x)²)) et **gaussienne** (f(x)=exp(-(alpha*x)²)) se comportent approximativement pareil mais elles sont plus ou moins contrastées et abruptes dans les reliefs. cependant elles "partent du bas" ce qui signifie qu'en ne posant que quelques clés en hauteur, elles vont d'elles même créer les reliefs. On pourra donc les utiliser pour des montagnes (inverse multi quadratique car plus progressive), des décors de grandes villes avec buildings ou ce genre de reliefs.

III - Analyses Personnelles

a - Zoé DURAND

Je dois avouer que j'ai d'abord été déçue du sujet lorsque nous l'avons eu. En effet je m'attendais à devoir coder un autre jeu auquel je pourrais jouer plus tard et j'ai eu le sentiment que ce sujet était trop "basique" dans l'interprétation (attention je ne sous entends pas qu'il était trop simple techniquement, loin de là, mais que l'appropriation artistique n'étais pas vraiment possible). Je me suis donc lancée dans le projet sans y porter beaucoup plus d'intérêt que celui que j'ai pour la programmation en général. Mais en découvrant le sujet je me suis rendue compte de ce qu'il allait m'apprendre. En effet, ce projet m'a appris beaucoup sur la manière d'appliquer les notions vues en cours de programmation (C++ et openGL) mais également sur l'organisation d'un projet. Nous n'aurions pas pu aller plus loin que l'affichage d'une fenêtre sans penser au préalable à l'organisation globale du projet. J'ai tendance à me lancer au hasard dans les projets et penser aux choses au fur et à mesures qu'elles me viennent mais le fait d'anticiper permet de gagner beaucoup de temps sur la réorganisation potentielle et d'être plus efficace sur la suite du projet.

Pour ce qui est du binôme je suis contente d'avoir fait ce projet avec Maéva car nous avons des compétences complémentaires qui nous permettent de nous conseiller l'une l'autre et de nous expliquer ce que nous ne comprenons pas. En effet Maéva m'a beaucoup aidé sur toute la partie openGL qu'elle comprend mieux que moi ainsi que l'utilisation d'ImGui et cela nous a permit de mener à bien ce projet malgré le temps qui nous manquait.

Cela m'amène à mon troisième point qui est le temps. Ce projet ne me plaisait pas trop au début c'est vrai mais en le faisant je me suis impliquée dedans et j'avais envie de le mener au bout. Cependant le temps et les autres projets ne l'ont pas permis et cela me désole car j'ai encore pleins d'idées pour améliorer cette application à laquelle je porte maintenant beaucoup plus d'intérêt.

Le mot de la fin pour la partie radial basis function qui m'a pris beaucoup de temps car une fois l'aspect mathématique compris je ne savais pas vraiment comment l'implémenter mais j'ai apprécié travailler dessus car cela permet de voir une application concrète de ce que les mathématiques nous permettent de faire si on sait comment les utiliser.

b - Maéva ROSENBERG

J'ai aimé faire ce projet, même si créer un logiciel de VoxelArt est moins amusant et moins sympa à montrer à ses amis ou sa famille qu'un jeu 3D.

Ma plus grosse difficulté a été l'intégration de la bibliothèque ImGui pour créer l'interface. Même si cette dernière permet de gagner énormément de temps (créer une fenêtre multi fonctions en quelques lignes est très pratique), je suis restée bloquée des heures sur des petits problèmes, ou à chercher ce que je voulais précisément dans la fenêtre de démo qui est, certes géniale, mais très fournie donc longue à démêler. Un cours rapide dessus, même 1h, pourrait être très utile et ce dès la première année d'IMAC.

A part ceci, je n'ai rencontré aucune difficulté particulière pour ce projet. J'ai l'impression de mieux saisir comment articuler les fichiers, les classes et les fonctions entre elles. Nous étions toutes les deux d'accord dès le départ sur la manière d'organiser notre code. Cependant je ne pense pas que j'aurai pu faire la partie sur les radial basis functions toute seule, n'étant de base pas très à l'aise avec les maths.

Je suis assez contente du résultat final, ce qui me motive à vouloir m'améliorer en programmation. Or comme tous les projets de programmation, World Imaker était surtout très long à coder et j'aurai donc évidemment aimé avoir plus de temps pour le développer en profondeur, sans empiéter sur les autres projets à rendre à la même période. J'ai aussi l'impression qu'il me manque encore des automatismes pour faire du code plus propre, comme les gestions d'erreurs.

Pour conclure, je dirai que l'apprentissage de la programmation et de la synthèse d'image peut parfois être long et fastidieux, mais faire des projets concrets avec de vrais résultats est ce qui permet de remotiver les personnes comme moi, qui hésitent encore sur le chemin à suivre après l'IMAC.

Conclusion du projet :

Blender n'est pas si dur à maîtriser… Mais on espère que notre logiciel aidera Toto ainsi que tous les passionnés de Voxel Art.