RAPPORT

World IMaker

Eva Benharira - Clara Daigmorte
IMAC 2

Algorithme - openGL - C++ - Mathématiques

Lien du projet : https://github.com/ClawsDevlp/world_Imaker.git

Sommaire

I) RÉCAPITULATIF DE L'AVANCÉE DU PROJET

FONCTION	FAIT	Fait mais bug	Réfléchi	NON FAIT
Créer Cube	X			
Supprimer Cube	X			
Extrud	X			
Creuser	X			
Initialiser une scène et la supprimer	X			
Générer une scène avec des radial basis function	X			
Changer de couleur	X			
Interface interactive	X			
Changer couleurs de plusieurs cube d'un coup			X	
Lumière directionnelle	X			
Lumière point		X		
Sauvegarder	X			
Texture		X (non fonctionnel)		

II) Architecture

build/
Dossier construit et exécutable

glimac/
Librairie glimac + nos classes

imgui/
Librairie imgui

src/
/assets
Dossier de textures
/shaders
Fichiers shaders
CMakeLists.txt
main.cpp
Fichiers sources du projet

third-party/
Librairie third-party

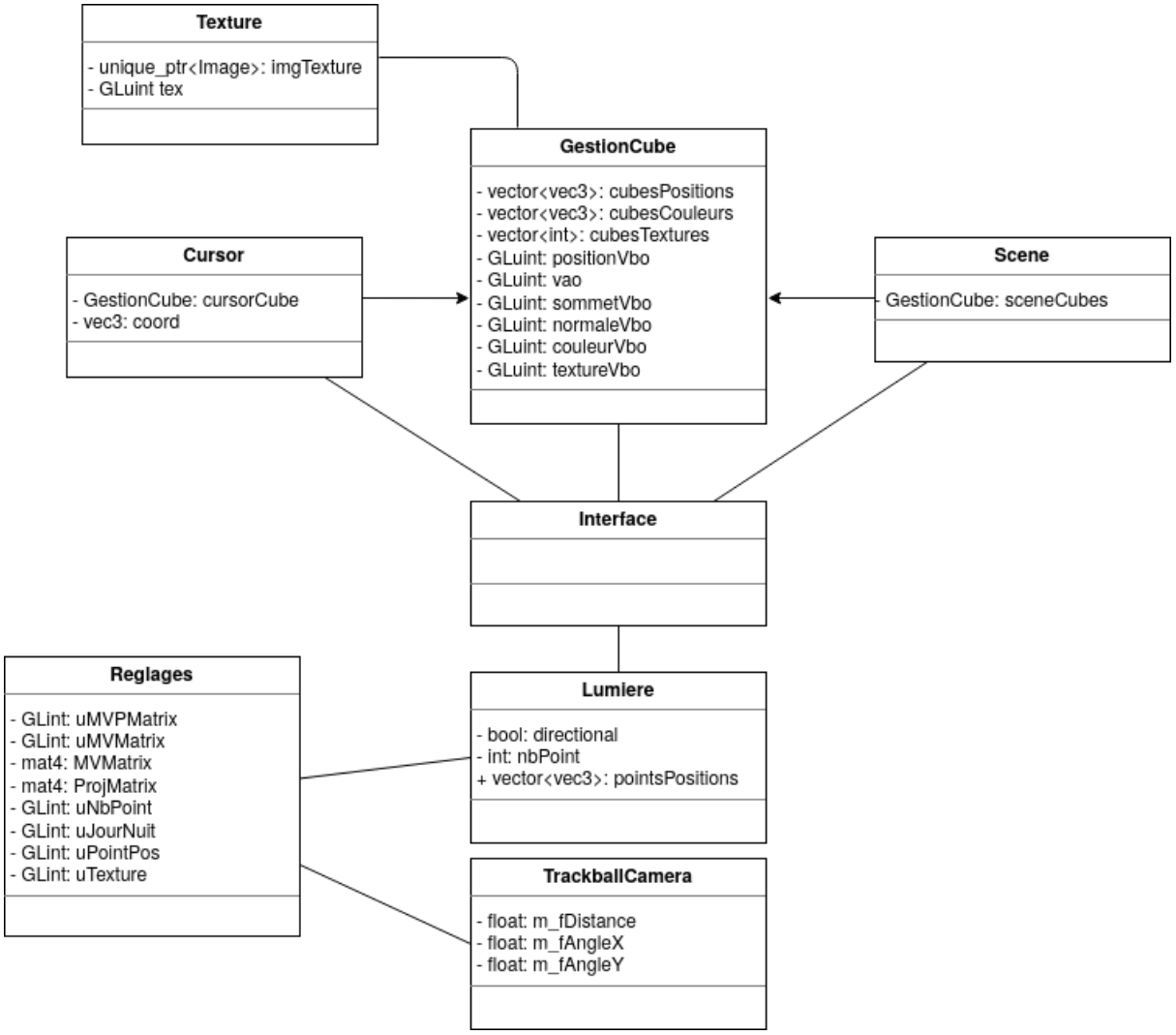
CMakeLists.txt
Fichier principal CMake du projet
sceneColors.txt
sceneCube.txt
Fichiers de sauvegarde

doc/
Rapport.pdf

Structure logicielle du projet

main.cpp	Le fichier main.cpp rassemble les fonctions des structures ci-dessous en un programme fonctionnel.
GestionCube.hpp GestionCube.cpp	Classe GestionCube qui gère tous les cubes dessinés.
Cursor.hpp Cursor.cpp	Classe Cursor, contient un cube. Il peut être déplacé par l'utilisateur.
Interface.hpp Interface.cpp	Classe Interface, elle gère le menu Imgui.
Lumiere.hpp Lumiere.cpp	Classe Lumiere gère la lumière directionnelle et les points lumineux.
rbf.hpp rbf.cpp	Fichiers qui contiennent les radial basis function.
Reglages.hpp Reglages.cpp	Classe Reglages gère les initialisations matrices uniform et renvoie les matrices.
scene.hpp scene.cpp	Classe scene gère la scène de base et celle générée par les radial basis function. Possibilité de sauvegarder et charger la dernière scène sauvegardée.
TrackballCamera.hpp TrackballCamera.cpp	Classe TrackballCamera, gère la caméra (ses mouvements) et renvoie une matrice vue de la scène.
Texture.hpp Texture.cpp	Classe Texture, gère son initialisation, bind et debind. Non fonctionnel.

UML



Dans notre programme, la fonction Scene n'a pas été implémentée de la meilleur manière dès le départ, nous nous en sommes rendues compte assez tardivement, mais nous avions choisi déjà de ne plus toucher au code pour ne pas avoir de mauvaise surprise le jour de la soutenance. C'est pourquoi sur cet UML, la classe Scene a un attribut sceneCubes, qu'elle devrait avoir aussi dans notre programme.

III) ANALYSE FONCTION PAR FONCTION

1) Create cube

Pour créer nos cubes, nous avons décidé d'utiliser des tableaux, contenant les informations de tous les cubes de notre scène. Pour cela, nous nous sommes inspirées de notre gestion des pixels dans le projet minigimp du S1 d'IMAC1.

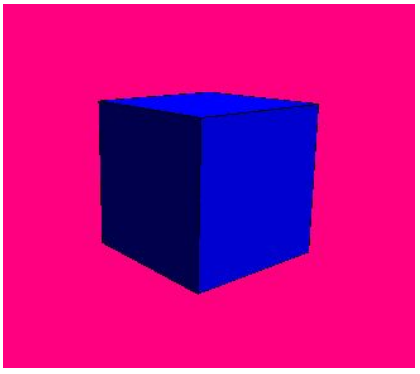
Au même titre que les pixels du projet minigimp, chaque cube possède une position et une couleur qui sont stockées dans un tableau chacune. La seule différence ici est que la position s'exprime en trois dimensions et non en deux dimensions. On a donc 3 valeurs pour déterminer la position au lieu de deux, exprimée en vec3.

Pour la construction du cube en lui-même, quelle que soit sa position, il est construit de la même façon, avec deux triangles par face qui ont toujours les mêmes rapports de tailles et distance.

C'est la classe GestionCube qui prend tout ceci en charge. GestionCube est ainsi composée d'un tableau de cubes et d'un tableau de couleurs.

La fonction ajoutCube se charge d'ajouter aux tableaux les données de position et de couleur de notre nouveau cube.

La fonction drawCube se charge de dessiner tous les cubes de notre tableau à l'écran.



Un cube

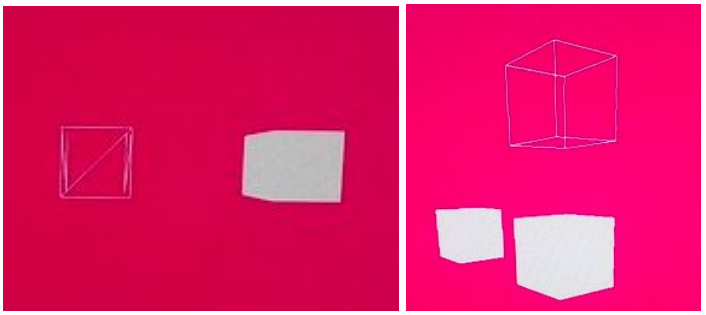
2) Curseur

Sur le modèle de la création de cube, la classe Cursor gère le curseur de notre monde et contient pour cela un cube dont on modifie les coordonnées en temps réel.

Cependant, pour distinguer notre cube curseur des autres cubes, nous avons choisi d'ajouter une fonction DrawCubeWireframe à notre classe GestionCube. Celle-ci permet de ne dessiner que les arêtes de notre cube plutôt que toutes ses faces en utilisant GL_LINE plutôt que GL_TRIANGLE.

Pour faire cela proprement, nous avons dû modifier notre IBO pour ne plus conserver que les sommets de nos faces avant et arrière ainsi que les arêtes qui les relient et non plus

les triangles qui composent les faces de nos cubes. On évite ainsi de tracer une ligne en travers de nos faces.



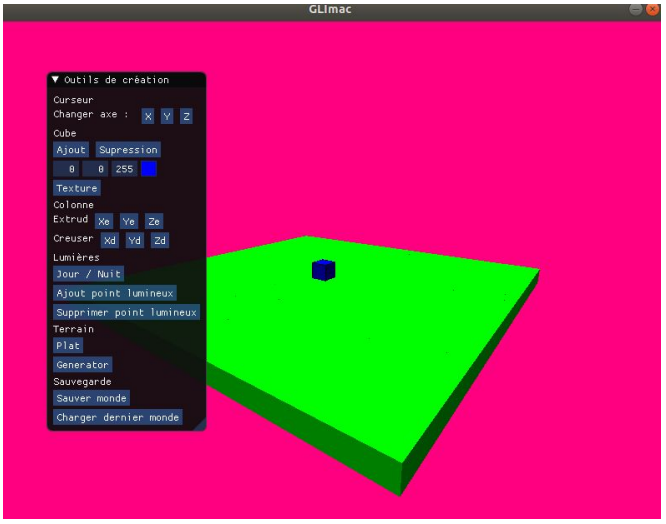
Notre curseur avant et après lifting

3) Initialisation Scene

Nous devons initialiser notre “scène de base” avec un nombre de cube déterminé ainsi que leur placement. Nous avons décidé de le faire sous la forme d’un sol et puisque nous devrons ensuite gérer également une scène générée par des radial basis functions. Nous avons donc choisi de créer une classe scène qui pourra centraliser toutes ces créations globales. Cette classe scène inclue GestionCube pour pouvoir créer une plâtrée de cubes.

La fonction InitScene permet de créer un sol en remplissant le tableau de cube de notre scène en fonction des x, z et y déterminé entre 20 cubes de longs, 20 cubes de larges et 1 d’épaisseur.

La difficulté que l’on abordera plus tard dans le rapport fut de centrer cette création de scène en 0.



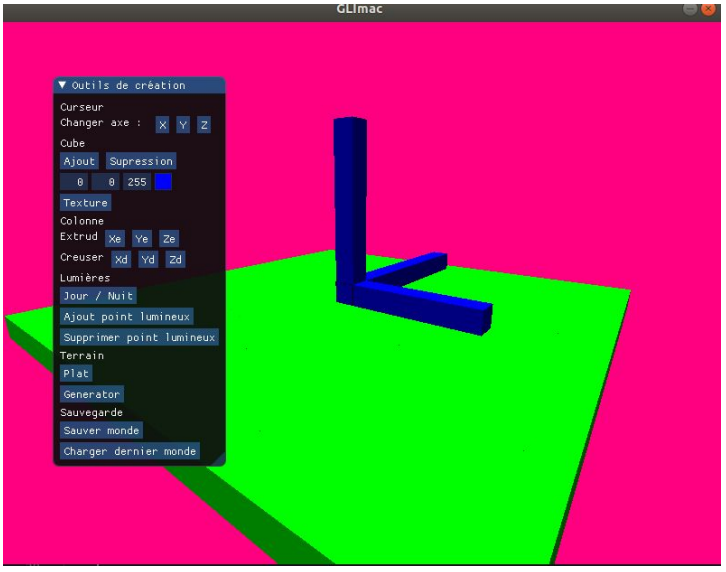
Scène de base

4) Extrud et creuser

Nous avons choisir de fusionner les fonctions extrud et creuser afin de ne pas avoir de code répétitif. Un booléen permet de choisir extrud (qui utilisera donc la fonction ajoutCube), ou dig (avec la fonction suppression). Le fonctionnement est donc le même. Il s'agit d'abord de chercher si un cube se trouve à l'emplacement du curseur. Si non on ne peut rien faire et le message "vous n'êtes pas sur une colonne" s'affiche de le terminal. Si oui, on passe à l'étape suivante.

En partant de notre cube on sélectionne l'axe x, y ou z puis on incrémente de 1 l'axe choisi. Tant qu'il y a un cube à l'emplacement suivant on recommence. Si l'emplacement suivant est vide 2 options :

- pour la fonction extrud : on ajoute un cube à l'emplacement vide puis on recommence les vérifications de même. S'il y a des cubes après on avance jusqu'à ce qu'il n'y en ait plus et on recommence à construire. S'il n'y en a pas on construit un cube chaque fois que l'utilisateur appuie sur extrud.
- pour la fonction creuser, on ne considère les colonnes que tant qu'il y à des cubes. Lorsqu'on arrive à un espace vide sans cube construit on ne va pas chercher plus loin et on commence à supprimer à partir du dernier cube atteint.

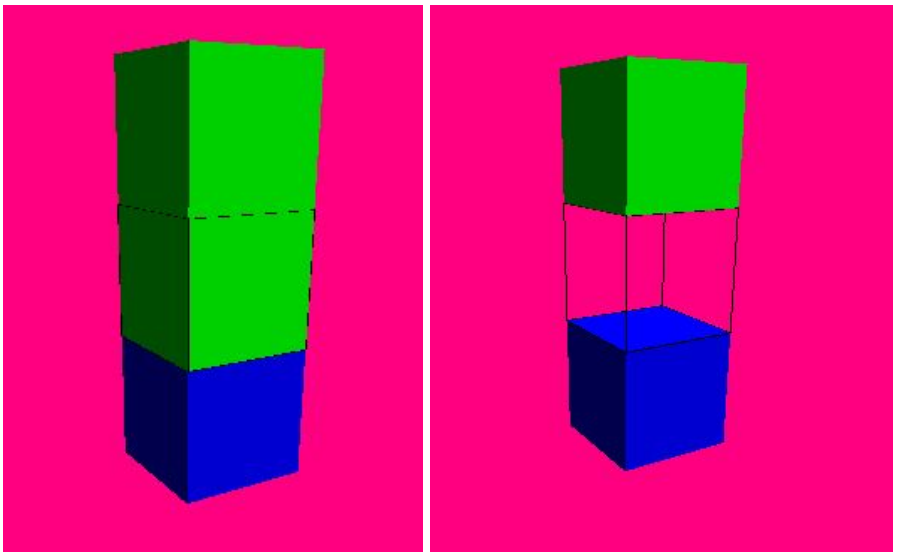


Fonction extrud sur les trois axes

5) Suppression

La fonction suppression reprend les coordonnées de notre curseur pour chercher s'il y a une série de coordonnées correspondantes dans notre tableau de positions de cubes.

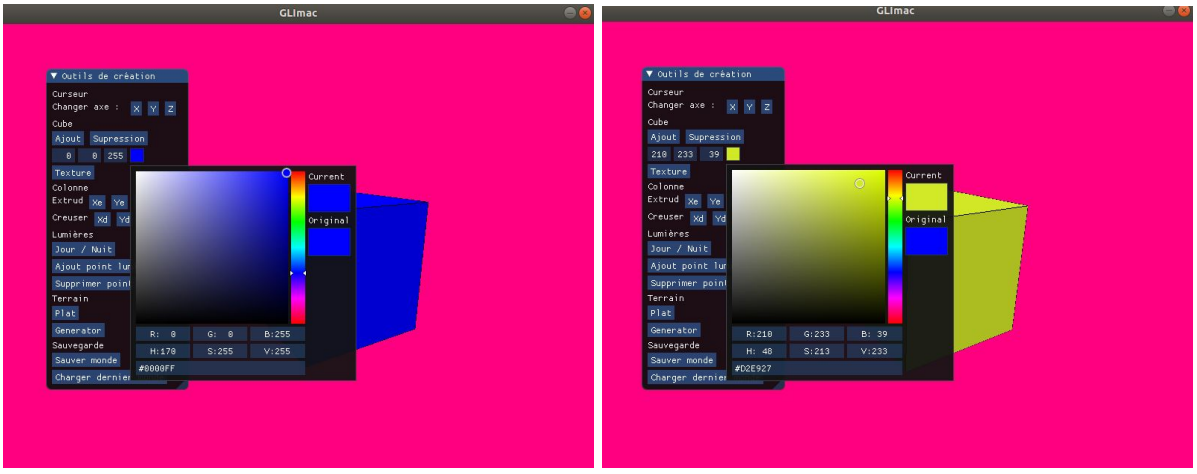
- Si elle l'a trouvée, on stocke l'indice. On supprime alors les vec3 au même indice de chaque tableau (position et couleur) grâce à la fonction erase.
- Si aucune position de cube ne correspond au curseur, on affiche dans le terminal qu'il n'y a pas de cube à cet endroit.



Suppression d'un cube

6) Changement de couleur

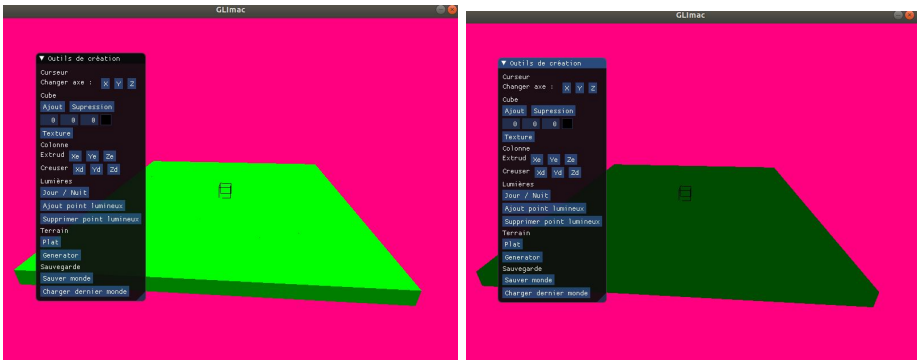
On utilise ImGui et ses fonctionnalités pour changer de couleur nos cube (un à la fois). Tout d'abord nous récupérons les coordonnées de couleurs provenant du cube sur lequel le curseur est positionné. Ses coordonnées sont transmises à imgui pour que la palette soit directement réglée dessus. Lorsque l'utilisateur change de couleur sur la palette, les coordonnées des couleurs du cube sont directement changée en temps réel.



Changement de couleurs grâce à Imgui

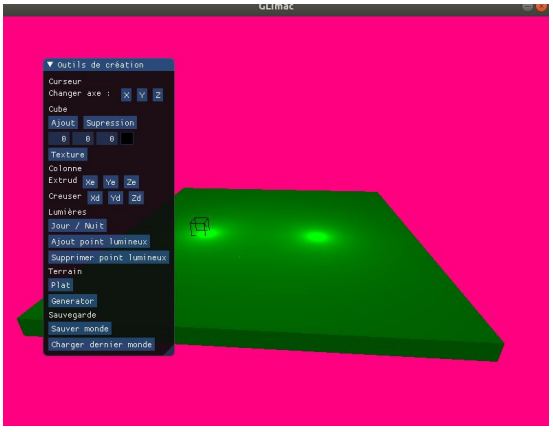
7) Lumières

On a voulu s'appuyer sur les lumières vues en TP mais la partie mathématique (fonctions blinn phong) s'est avérée difficile à modifier. C'est en discutant avec Jules Fouchy que nous avons pu avoir un code beaucoup plus simple à gérer.
Grâce à cela, nous avons une lumière directionnelle qui représente le Soleil. Il est possible de la désactiver pour avoir la nuit.



Jour et Nuit

Une lumière ambiante est présente pour toujours voir un minimum, même si le cube n'est pas directement éclairé.
Aussi, il est possible d'ajouter des points de lumière, pour éclairer. Ils sont visibles lors de la nuit.



Points de lumière vus de nuit

Les points de lumière sont gérés dans la classe Lumière dans un tableau de points. Ce tableau de point est envoyé au fragment shader.

8) Caméra

Pour la caméra nous avons tout simplement repris celle créé lors du TP caméra TrackBall pour laquelle on a créé une classe à part. Il est donc possible de faire un mouvement de rotation de haut en bas, de gauche à droite, ainsi qu'un mouvement vers l'avant ou l'arrière. Nous avons choisi de modifier la distance base par rapport au centre de notre "monde" de manière à pouvoir visualiser toute la scène que l'on initialise avec un sol ou avec les radial basis function.

9) Imgui

Pour gérer les interactions de l'utilisateur avec le monde, l'utilisateur peut utiliser le clavier grâce aux instructions dans le readme, mais pour plus de facilité, nous utilisons ImGui. Nous avons choisi de créer une classe qui y est dédiée : Interface. Dans celle-ci, des

fonctions initialisent et dessinent la fenêtre ImGui à partir de laquelle l'utilisateur peut interagir avec notre logiciel.

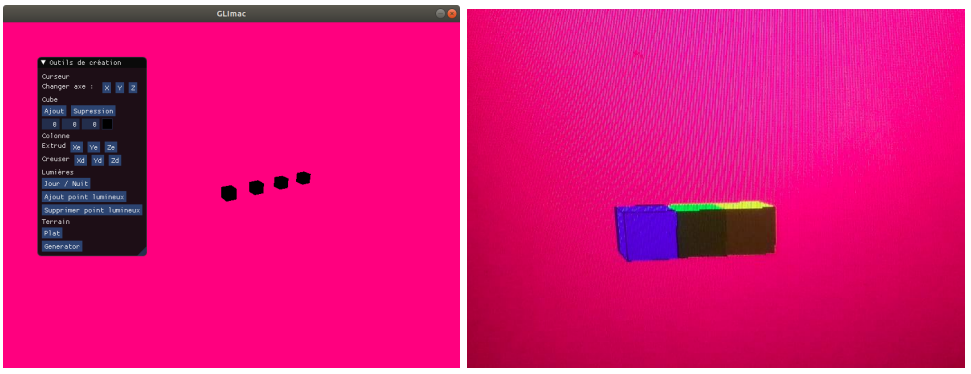
Menu Imgui

10) Sauvegarde

Créer une sauvegarde de notre monde revient à créer une sauvegarde de notre "scène". Nous avons donc choisi d'implémenter de nouvelles méthodes à notre classe scène. L'une qui permettrait d'écrire un fichier sauvegarde à partir de notre scène actuelle, l'autre d'aller chercher ce fichier sauvegarde, de le lire et de construire une scène à partir des données récupérer.

Pour cela on utilise les flux ifstream et ofstream. On récupère les données de notre gestionnaire de cube. Celles-ci sont stockées dans des vecteurs que l'on parcourt. Nous avons d'abord implémentée la fonction sauvegarde pour la position seule en ne parcourant que le vecteur cube position, puis nous avons ajouté la couleur en parcourant cube couleur.

Nous aurions pu rentrer toutes ces données dans un seul fichier, mais pour plus de lisibilité, et de facilité à la lecture, nous avons choisi d'en faire deux : un pour les positions et un pour la couleur.



Scènes chargées grâce à un fichier sauvegarde sans et avec couleurs.

Pour enregistrer la scène on parcourt donc deux vecteur vec3 qui stockent pour l'un les positions x,z,y et pour l'autre les informations colorimétrique RVB. Pour recharger une scène sauvegardée, on parcourt de même les deux documents de sauvegarde et on crée des cubes avec les propriétés récupérées dans ces deux documents. Les informations de ces cubes sont de nouveau stockées dans les vecteurs vec 3 du gestionnaire de cubes de cette nouvelle scène.

IV) RADIAL BASIS FUNCTION

1) Implémentation

Après avoir repris le cours, nous avons choisi d'utiliser Eigen. En effet, la valeur absolue de la différence utilisée dans la formule correspond à la norme qui peut être calculée directement via Eigen en faisant .norm. Eigen nous permet également de manipuler des vecteurs et des matrices.

Après avoir discuté avec Margaux Vaillant, nous avons pu décomposer l'obtentions des radial basis functions (rbf) en plusieurs étapes.

$$g(x) = \sum_{i=1}^{k} \omega_i * \Phi\left(|x - x_i|\right)$$

En partant de la fonction donnée dans le cours, on sait qu'il va nous falloir des points de contrôle, des valeurs à leur associer, des w et des phi.

Pour retourner au final les valeurs associés à chacun des Cubes de ma scène (qui a la même dimension que la scène que j'initialise) j'ai une fonction getMap qui va retourner une matrice de dimension (x,z) et chaque valeurs de la matrice correspondra à la valeur de la hauteur y de la colonne de cube au niveau du cube (x,y).

Il est écrit dans le cours qu'une matrice composée de phi de valeur absolue de xi -xj multiplié par les oméga est égale à nos valeurs pour chaque point de repère. Mathématiquement on en déduit :

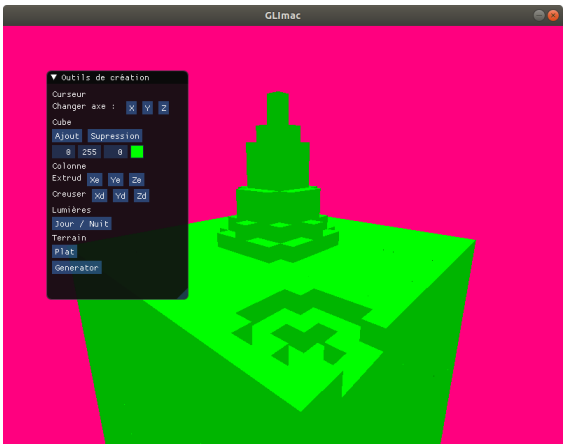
$$A * \omega_i = u_i \iff \omega_i = A^{-1} * u_i$$

On connaît nos u de i puisqu'on les a rentrés, reste à calculer l'inverse de A où A la matrice des phi. On crée donc une fonction trouveOmega qui va créer puis remplir une matrice vide en fonction de notre nombre de points de repères. On la remplie avec deux boucle for qui parcourent lignes et colonnes.

C'est ici que l'usage de Eigen est intéressant car à l'appel de la fonction phi, pour obtenir la valeur absolue de la différence, celle-ci étant la norme, on peut utiliser l'appel .norm() de Eigen.

Reste à déterminer une fonction phi en fonction d'une variable à qui sera la valeur absolue de notre différence, soit notre différence.norm().

On à commencé nos test avec une fonction exponentielle en 3 points à valeurs positives ou négatives, ce qui nous permet d'obtenir l'image ci-dessous.



Génération grâce aux rbf avec phi une exponentielle

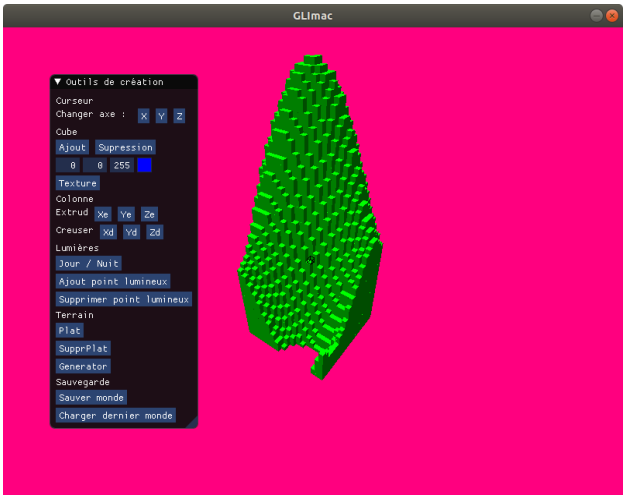
2) Test et observation

a) Exponentielle

La génération ci-dessus est obtenue grâce à l'usage de la fonction exponentielle. Celle-ci nous permet d'obtenir des tours ou des trous plus ou moins grand à chacun de nos point de contrôle. C'est la fonction la plus facilement contrôlable selon nous lors de la génération procédurale pour obtenir à la fois des pics et des creux.

b) Cosinus

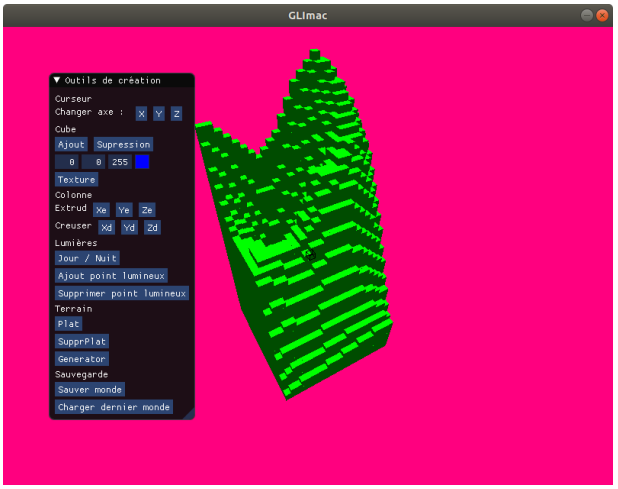
L'emploi de la fonction cosinus pour générer notre scène nous permet d'obtenir une montagne assez progressive selon les valeurs données. Le tout est assez uniforme et donc pratique pour générer une forme de grand mont.



c) Sinus

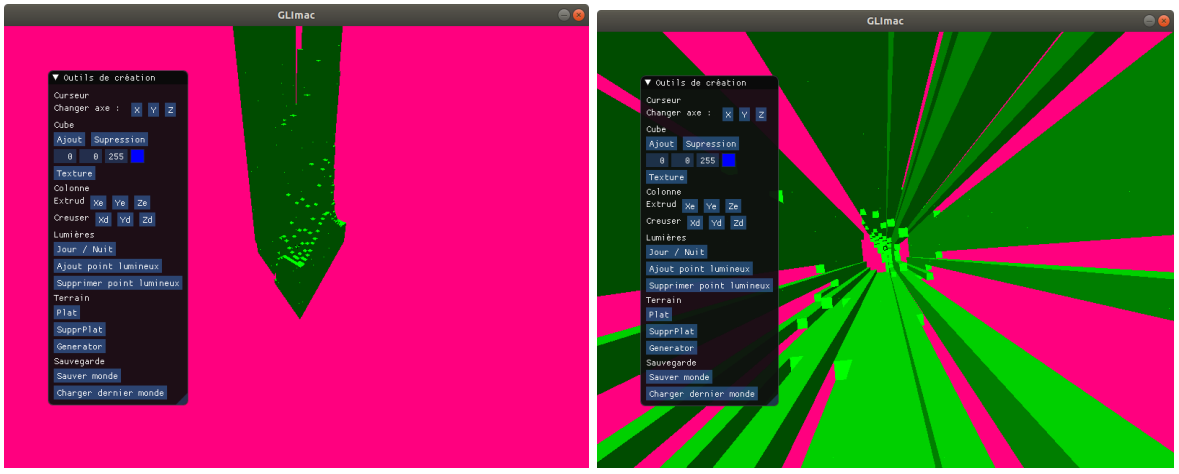
La fonction sinus quant à elle semble ici plus propice à la génération de valon, on observe la formation d'un creux au centre de notre scène. Ce qui est cohérent avec

l'observation précédente de la formation d'un mon grâce à la fonction cosinus. En effet, avec les mêmes points de repère on se situe vraisemblablement aux deux extrêmes haut et bas de chacune de ces fonctions.



d) Tangente

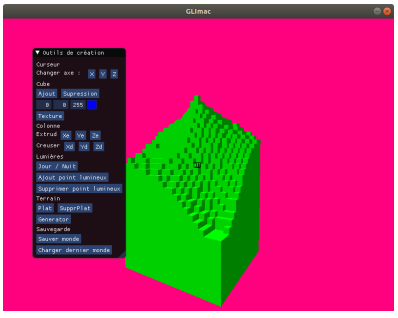
Enfin, nous avons décidé d'essayer d'appliquer la génération procédurale à la fonction tangente pour voir si nous pouvions obtenir un résultat non continu, avec des valeurs égale à 0.



Vue de face et de dessus d'une génération rbf avec tangente

Cependant, la nature de la fonction tangente, veut qu'elle tende vers l'infini positif quand le cosinus tend vers 0, ce qui donne lieu à la formation des pics infinis présent ci-dessus. Il aurait donc fallu gérer ce cas particulier pour rendre la fonction exploitable, ce que nous n'avons pas pris le temps de faire.

e) Equation du cercle



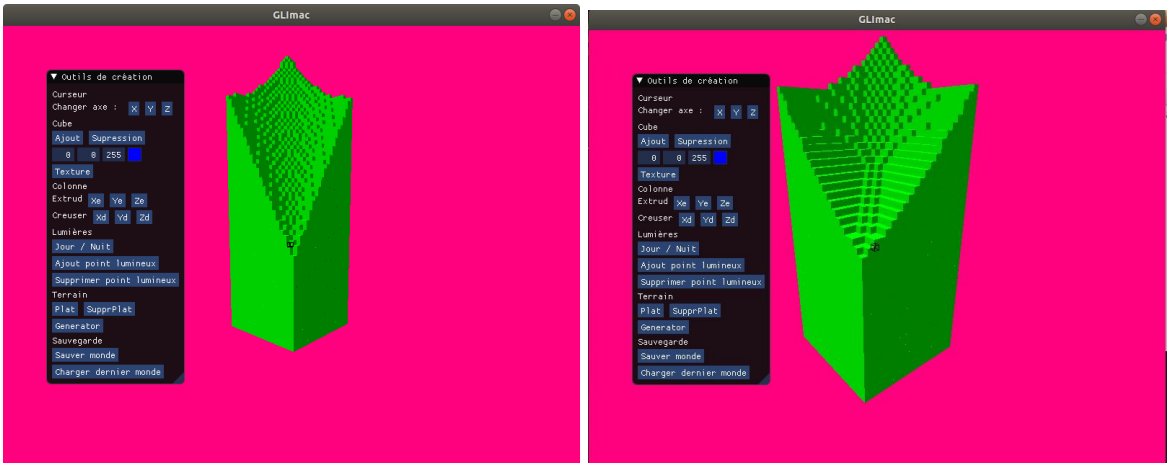
Dans notre recherche de génération, nous avons décidé de tester l'équation du cercle :

$$\sqrt{(x-a^2)}$$

Avec x une constante quelconque, on a choisi de tester avec 5.

Avec nos 3 points de repères de base avec exponentielle, on obtient un terrain aléatoire assez sympathique.

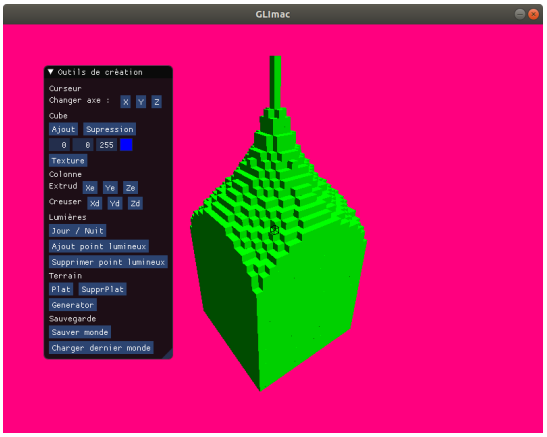
Nous avons ensuite réalisé d'autres tests avec notamment deux points de repères placés à deux coins face à face de notre terrain en leur donnant à l'un une valeur haute et à l'autre une valeur basse. Cela nous permet d'obtenir une forme de quart de colisé assez symétrique.



Vue avec deux points de repères aux extrêmes aux valeurs respectivement petite et grande

f) Racine carrée

Les tests sur l'équation du cercle nous ont poussé à chercher plus loin avec la fonction racine carrée. Également, le choix des points au coin était assez tentant, nous avons donc testé de combiner les deux.



Génération racine carrée avec 5 points de repère

3) Pour aller plus loin

La recherche sur la génération mathématique nous a donné envie d'aller plus loin. Voici donc diverses fonctionnalitées que nous aurions aimé implémenter si nous en avions eu le temps.

- Charger à partir d'un fichier
- Choix random des valeurs
- Choix des points de repère par sélection de cube

V) DIFFICULTÉS RENCONTRÉES

1) Cmake

Au début du projet nous avions décidé dans notre projet d'utiliser des Cmake connu tel que Eva avait vu comment les générer avec Guillaume l'an passé et cette année en cours de soutien, tout en décidant de conserver la librairie glimac dont Clara avait l'habitude de se servir. Nous avons donc décider de créer un fichier cmake hybride pour réunir cette librairie de nos TD d'openGL et notre cmakeList.txt permettant de compiler sur Windows Linux ainsi que d'utiliser Imgui.

Après divers problème de include, de gestion de la SDL2 et autre nous avions un cmake censé être fonctionnel mais qui ne fonctionnait que sur le PC de Eva et pas sur celui de Clara, pour lequel divers éléments de la SDL comme SDL event molette n'était pas reconnu.

Au bout de 3 jours et divers aller-retours par mail qui nous ont quelque peu éclairé, nous avons tenté de reprendre le tout avec l'aide de Pierre Thiel, cependant, lui non plus n'est pas parvenu à voir d'où venait l'erreur.

Nous avons ensemble vu pour rédiger un nouveau cmake qui nous à permis de compiler mais de manière buguée ce que Clara avait commencé à coder en parallèle. En effet, la couleur des cube apparaissait noire au lieu de blanche entre autre.

Nous avons alors choisi d'avancer malgré tout à partir du code que Clara avait rédigé dans ses TDs d'openGL. Ce qui nous a fait passer de notre premier dossier git :

https://github.com/ClawsDevlp/world_Imaker_OLD

à un second : https://github.com/ClawsDevlp/world_Imaker.git.

Nous sommes donc revenues au Cmake sous la forme de celui des TD de Céline. Cette erreur si nous ne sommes pas parvenues à la résoudre nous a cependant permis de nous confronter à la création des cmake et de mieux comprendre ceux utilisés par nos TDs.

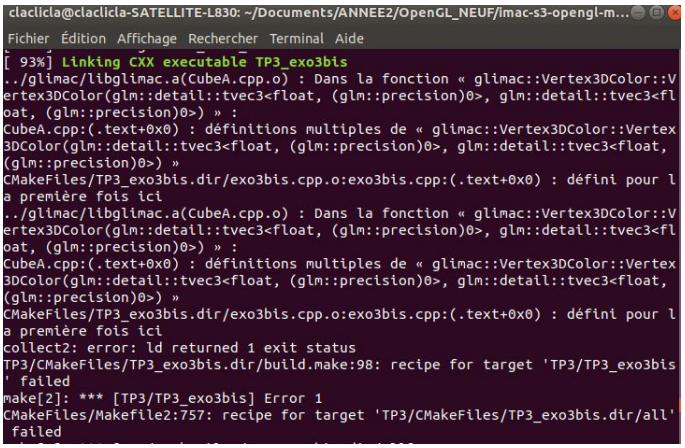
C'est ainsi que nous avons découvert notamment que ceux-ci fonctionnaient avec des CmakeList.txt imbriqués, chaque dossier en contenant un pour compiler.

2) Création cube

Au tout début du projet, nous avons commencé en créant nos cubes un par un, avec un vbo chacun. Nous étions contentes. Mais en discutant avec Jules Fouchy, il nous a très rapidement fait comprendre que ce n'était pas optimisé et qu'il valait mieux utiliser un

tableau de cubes que l'on dessinerait tout en un seul coup. Car effectivement nous ne dessinions que un ou deux cubes à ce moment là, mais lorsqu'il aurait fallu faire une scène avec des dizaines voir des centaines de cubes, cela aurait été une autre histoire pour nos ordinateurs en surchauffe. C'est ce sur quoi nous nous sommes alors penchées.

Aussi pour cela, nous souhaitions utiliser une structure Vertex3DColor (contenant coordonnées, couleur, normale) afin de créer ce tableau de cube. Malheureusement, cela ne fonctionnait pas à cause d'une mauvaise inclusion, et par perte de patience (car cela a été un long problème) nous avons supprimé cette structure pour n'utiliser que des vec3, bien plus simples d'utilisation mais en même temps moins pratique.



La structure maudite

3) Normales et lumières

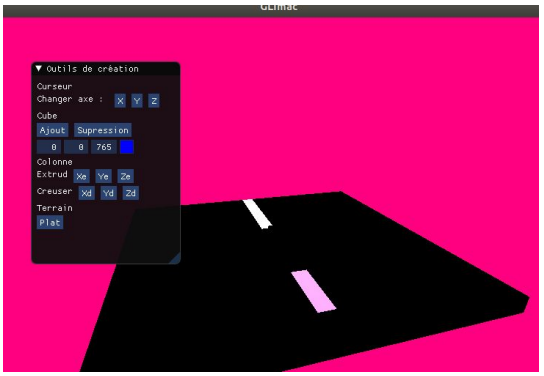
Tout au long du projet, nous avons eu beaucoup de problèmes avec nos normales. Au départ elles étaient mal définies en fonction des faces. Ce qui donnait des lumières particulières sur nos cubes.

Normales mal gérées

Effectivement, après une mauvaise compréhension de notre part des normales, nous n'avions pas rempli correctement leurs coordonnées (voir capture d'écran ci-dessous).

```
//face coté gauche
Vertex3DColor(glm::vec3(-0.5, 0.5, 0.0), glm::vec3(0, 1, 0)),
Vertex3DColor(glm::vec3(-0.5, -0.5, 0.0), glm::vec3(1, 0, 0)),
Vertex3DColor(glm::vec3(-0.5, -0.5, -1.0), glm::vec3(0, 0, 1)),
Vertex3DColor(glm::vec3(-0.5, 0.5, 0.0), glm::vec3(0, 1, 0)),
Vertex3DColor(glm::vec3(-0.5, 0.5, -1.0), glm::vec3(1, 0, 0)),
Vertex3DColor(glm::vec3(-0.5, -0.5, -1.0), glm::vec3(0, 0, 1)),
```

Mais après explication de Céline nous avons enfin compris à quoi correspondait les normales, et avons pu corriger l'erreur.



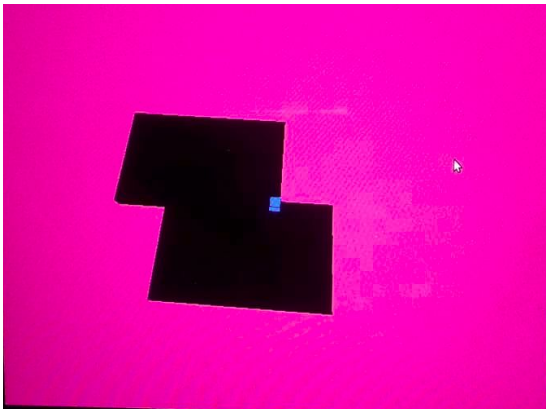
"Route" générée à cause des normales mal gérées

Plus tard, une nouvelle erreur apparu, car nous avions rajouté une fonction en trop :

glVertexAttribDivisor(VERTEX_ATTR_SOMMET_NORMALE, 1);

Qui signifie que chaque normale est attribuée à un seul cube, plutôt qu'à chaques faces. Donc les 6 normales des faces de cube se sont retrouvées "à plat", créant une magnifique route.

4) Construction centrée



Le fait que la caméra soit centrée sur le point (0,0,0) de notre scène nous à pousser à devoir découper en boucle l'initialisation de notre scène de même que notre génération de scène grâce aux radial basis functions. En effet sans cela notre fenêtre aurait semblée décentrée.

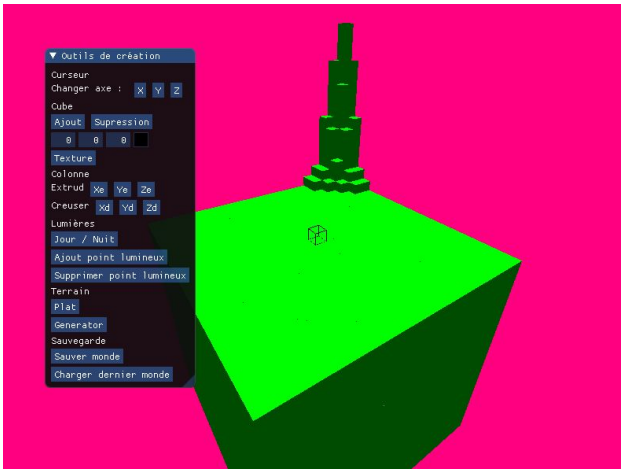
Il a donc fallu pour ça découper notre scène en 4 carrés. Un de -10 à 0 en x et en z, un de -10 à 0 en x et de 1 à 10 en z, un de 1 à 10 en x et de -10 à 0 en z et un de 1 à 10 en x et en z.

Pour faire cela, on a du avoir recours à une imbrication de boucles.

5) Rbf

La première et principale difficultés avec les radial basis function consistait à en comprendre le fonctionnement et à le découper de manière à pouvoir l'implémenter, ce que nous sommes parvenues à faire avec l'aide de Margaux Vaillant.

Le seul autre problème a ensuite été de gérer la génération avec notre problème de placement. En effet, la génération grâce au rbf n'est pas régie comme celle de l'initialisation de notre scène. Ainsi si on veut créer un pic en (0,0) il ne faut pas donner comme point de repère (0,0) ou la "tour" se retrouve à être générée en (-10,-10). Il à juste fallu le remarquer et adapter notre espace et nos boucles.



Génération rbf exponentielle de valeurs 15 au point de repère (0,0) qui sur notre scène correspond au point (-10,-10)

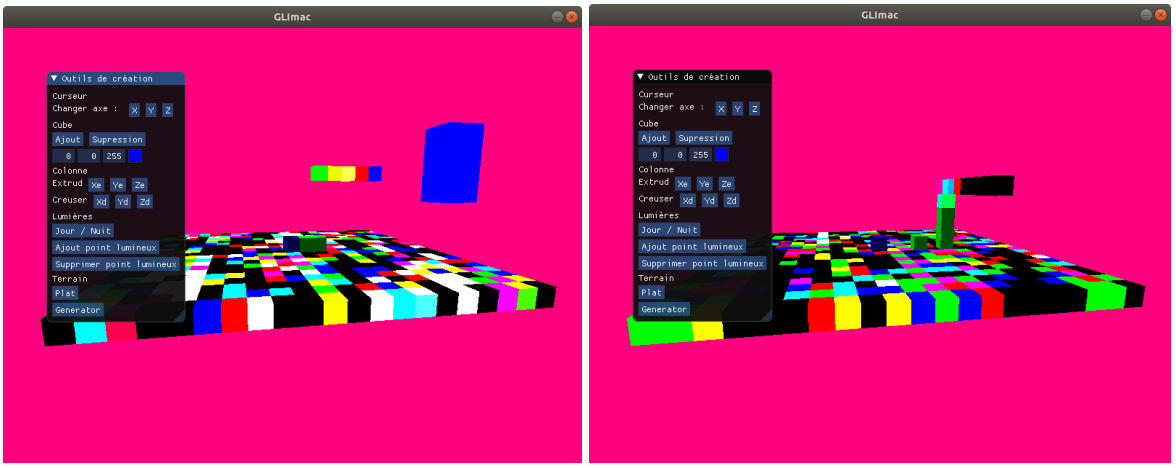
6) Extrud et sauvegarde des couleurs

Nous avons eu une erreur de couleur avec extruDigCube à cause de la valeur de base donnée à la couleur lors de l'ajout d'un cube :

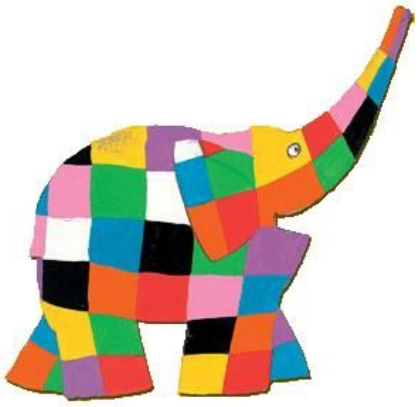
```
ajoutCube(position+incrementVec, glm::vec3(0,0,4));
```

Alors que ce sont censé être des float entre 0 et 1. Cela créait un problème au niveau de l'ombrage de ces nouveaux cubes : ils n'étaient pas influencés par les lumières.

Aussi cela créait des problèmes pour la sauvegarde des couleurs :



Scènes buguées “Elmer”



Dès qu’une scène contenant un cube par extrud était sauvegardée, nous obtenions ce problème un fois la scène chargée. Les cubes étant tous censé être verts.
En fait, nous avions repris la même manière d’enregistrer la position des cubes pour leur couleur. Sauf que cela utilisait des entiers, alors qu’il fallait des float pour les couleurs.

Une fois les deux problèmes réglés, nos scènes s’affichaient et s’enregistraient parfaitement !

7) Textures

Comme options pour le projet, nous souhaitions faire la sauvegarde ainsi que les textures. Ces dernières que nous avions déjà travaillées en TD et réussies à implémenter, nous pensions pouvoir les gérer aisément. C’est vers la fin du projet que nous avons commencé à implémenter cette dernière fonctionnalité mais le manque de temps et de sommeil ayant frappé, nous n’avons pas réussi à finir.
Nous avions commencé à les ajouter comme nous l’avions fait pour les couleurs avec un buffer instancié textureVbo qui dicte quelle texture utiliser sur chaque cube avec un ID.

VI) RETOURS PERSONNELS

1) Clara

a) Points forts du projet

Toujours une des choses que j’apprécie le plus quand je programme, c’est de voir visuellement ce que l’on crée. J’ai été servie : c’est très agréable de voir que l’on peut créer des cubes ! Observer les lumières réagirent en fonction de nos cubes, et les scènes générées par les radial basis functions était ce que j’ai préféré. J’ai pu complètement acquérir les concepts de classe, d’attributs privé / public, const ou non et les méthodes.

b) Points faibles du projet

Pour ma part, une des plus grande difficulté a été de devoir concevoir autant de notions en même temps. Surtout en openGl, ce qui a été déjà difficile à acquérir en TP (comprendre comment fonctionne les shaders, vbo / vao, …), même s’ils sont expliqués pas à pas. Si on n’arrive pas à maîtriser ces concepts, il est impossible de pouvoir réussir ce projet.

c) Ce que je retiens

J’aurai apprécié avoir vraiment le temps de travailler dessus mais je suis heureuse du résultat que nous avons réussi à coder, malgré nos difficultés. Aussi, le langage C++ est bien plus flexible que le C, ce que j’apprécie.

2) Eva

a) Points forts du projet

Ce que j'ai apprécié dans ce projet c'est que je l'ai trouvé plutôt très détaillé et assez en lien avec les TDs notamment même si je n'ai pas pu aller au bout de ceux-ci, c'était rassurant de savoir qu'on avait déjà une première pierre de posée et des connaissances auxquelles nous raccrocher.

La mise en situation si elle est peu crédible, on ne va pas se le cacher est sympathique et participe à un point qui me semble important : rendre le projet accessible. En effet, contrairement aux projets de premier année, j'ai eu la sensation en lisant le sujet que certes les attentes étaient hautes mais que nous avions les clés en main pour réussir à rendre quelque chose, ce qui est très motivant tout au long de la réalisation du projet.

Le caractère visuel du projet est également très appréciable. En effet, j'ai pu cette année installer Linux sur mon ordinateur personnel, ce qui m'a permis de tester le code au fur et à mesure à travers l'affichage. Ceci, comme lors de la réalisation de nos projet processing en IMAC 1 rend le travail plus agréable car matérialisé.

J'ai également apprécié que le sujet fasse appel à divers type de compétence. De l'utilisation de git, à la rédaction des cMake en passant par la conception des classes, l'affichage en synthèse d'image ou la partie mathématique. Cela permet de bien fonctionner en équipe et d'avoir du travail pour chacun à son échelle ce qui est valorisant. J'ai la sensation d'avoir véritablement pu contribuer au projet en en maîtrisant les grandes lignes et en faisant ma part ce qui pour moi qui ne suis pas une "bête du code" est important.

b) Points faibles du projet

Si j'ai apprécié que le projet soit découpé de manière claire avec les différente fonctions à implémenter et un minimum requis de spécifié, je regrette que les fonctions optionnelles ne soient pas si optionnelle que cela. En effet, l'obligation d'en implémenter au moins deux, même si elles sont nombreuses ajoute du stress. D'autant que je pense que le projet aurait pu être améliorable dans notre cas sans passer par ces fonctions supplémentaires mais en se concentrant plus sur la propreté et la rigueur du code que nous avons malheureusement un peu laissée de côté pour avancer et produire le maximum de ce qui était attendu.

Je trouve également que nous demander une réflexion d'ordre de l'arhitecture logicielle à base d'agencement de classe, de schéma UML et de design pattern est intéressant, cependant, sans retours au préalable, avant que l'on ne se lance dans le code je ne vois que difficilement l'intérêt, d'autant plus que pour ceux qui ne sont pas familiers de ces mode de pensée, il s'agit d'une difficultés supplémentaires à laquelle on ne vient jamais vraiment apporter une correction. Je pense qu'il demeure intéressant de conserver cet aspect mais peut-être en amont du projet au tout début puis rencontrer les groupe pour en discuter et les aiguiller. Également proposer une architecture de projet type comme "correction" pourrait ensuite être intéressant pour donner des retours sur les choses à faire ou ne pas faire.

J'ai regretté le placement temporel du projet. En effet, le fait qu'il soit à cheval sur les vacances de Noël ne nous place pas dans les meilleures disposition pour le mener à bien tel qu'on le souhaiterait, surtout avec des partiels à la rentrée.

Je trouve ainsi que nous avons manqué de temps pour aller plus loin dans nos réflexion et le développement de nos fonction. De ce point de vue, je regrette surtout que nous n'ayons pas pu consacrer plus de temps à la rédaction d'un code propre. Ils nous étaient difficile dans le temps imparti de prendre le temps de rechercher et d'appliquer les "bonnes pratiques" vues en cours.

Enfin dans le même ordre d'idée, pour les fuites de mémoires ou les bugs dont on ne parvient pas à trouver les explications ou la source, je regrette qu'il n'y ait pas un temps plus important dédié à ces points avec les professeurs.

c) Ce que je retiens

J'ai encore beaucoup appris lors de ce projet et je sens que je commence à vraiment me familiariser avec la programmation et ses logiques. J'ai beaucoup apprécié de retrouver le langage C++ même si je regrette de ne pas avoir pu pousser plus loin en appliquant plus les éléments vus en cours cette année.

J'ai continué à découvrir les cMake même si je n'ai toujours pas compris la source de l'erreur dans les premiers que j'ai rédigé. Devoir réutiliser ceux des TPs ainsi que mes échanges de mails avec Céline m'ont permis de mieux en comprendre les composantes de manière à mieux les manipuler.

J'ai pu réaliser moi-même le système de sauvegarde, ce qui m'à permis de me familiariser avec l'écriture et la lecture de fichier qui avait sur mes autres projets été prise en charge par mon binôme.

La mise en place des radial basis function m'a permis de mieux comprendre comment il faut décomposer la mise en place d'une fonctionnalité en code.

Je pense également avoir appris à manipuler les vbo et ibo un peu mieux qu'auparavant dans la continuité des TD d'openGL. C'est loin d'être parfait mais ces notions me semble dorénavant plus familières.

Enfin, ce travail dans l'espace 3D en mouvement du fait des modifications de l'utilisateur m'a permis de mieux appréhender la gestion de vectors et de matrices en informatique.

Et par-dessus tout je retiens le travail avec mon binôme. C'est la première fois que je travaillais avec quelqu'un d'autre que Laurelenn en programmation et cela s'est avéré très bénéfique pour moi. Clara m'a laissé prendre des initiatives et elle m'a aidé à me débloquer quand j'en avais besoin. J'ai beaucoup apprécié travailler avec elle !

VII) CONCLUSION

Nous sommes très heureuses de ce que nous avons réussi à produire dans le temps imparti. Cependant, les tests de notre logiciel avec d'autres personnes nous ont permis de nous rendre compte de fonctionnalités qui pourraient être implémentées pour faciliter l'expérience utilisateur. Nous avons ainsi pensé à changer la couleur du curseur selon s'il est ou non sur un cube pour pallier la perte de perspective dûe à sa constante visibilité même en-dessous d'un sol de cube. Également permettre de changer la couleur de plusieurs cubes à la fois soit, toute la scène initialisée, soit colonne par colonne à la manière des fonctions extrud et dig pourrait rendre l'interaction plus ludique.

Pour conclure, nous pensons toutes deux avoir beaucoup appris durant ce projet même si nous regrettons de ne pas avoir pu aller au bout de tout ce que nous voulions faire. Il est sûr qu'en reprenant le même projet aujourd'hui nous éviterions bien des écueils ce qui est la marque de notre progrès.

En espérant que vous et Toto puissiez utiliser ce logiciel avec joie nous vous quittons sur le partage d'une citation papillote, de circonstance en cette période post fêtes et vous souhaitons une bonne année 2020 !

« L'erreur n'annule pas la valeur de l'effort accompli. »

Proverbe africain