

# C++ - Synthèse d'images - Mathématiques

IMAC 2

– 2019-2020 –

---

## Rapport binôme - WorldIMaker

---

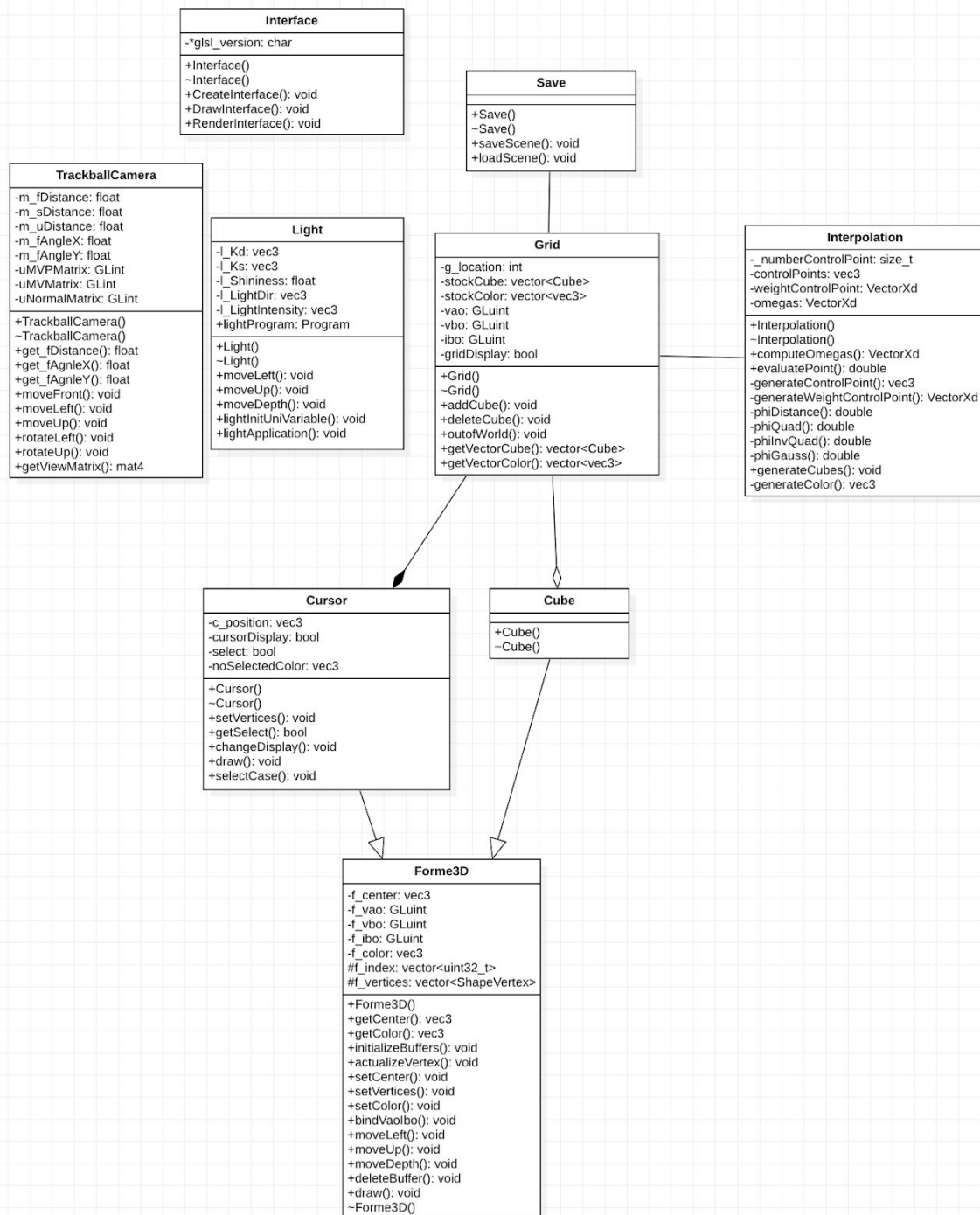
### *I. Projet dans son ensemble*

#### 1. Tableau récapitulatif

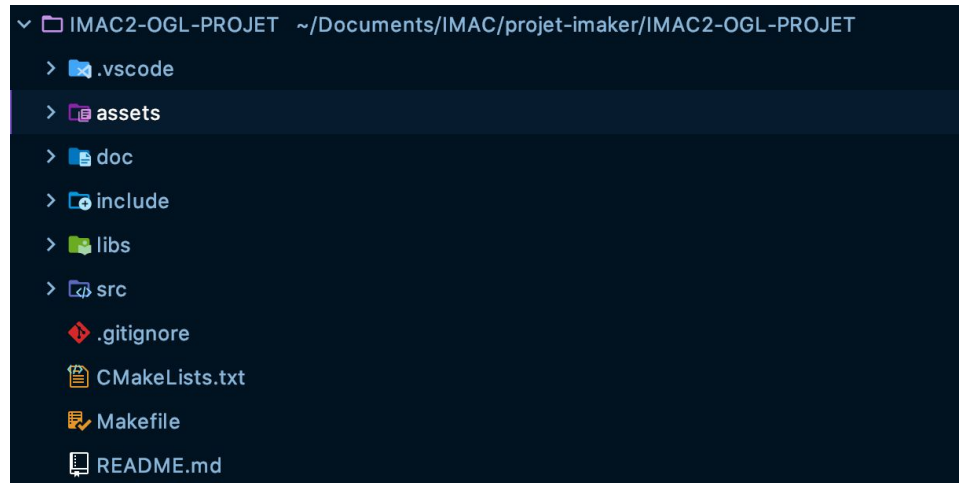
Fonctionnalités	Implémentation	Commentaires
Affichage scène	Intégrée	
Sélection cube	Intégrée	
Changer couleurs cube	Intégrée	
Ajouter cube	Intégrée	
Supprimer cube	Intégrée	
Curseur	Intégrée	
Extruder	Partielle	
Dig	Partielle	
Lumière directionnelle	Intégrée	
Ajouter / Supprimer	Non intégrée	Activer et désactiver la lumière possible
Point de lumière	Intégrée	
Ajouter / Supprimer	Non intégrée	Activer et désactiver la lumière possible
Génération procédurale	Intégrée	
Sauvegarde/Chargement scène	Intégrée	
Texture 6 faces	Non intégrée	



## 2. UML du projet



### 3. Architecture du dossier - projet



- **Assets** : Dossier dans lequel on retrouve les fichiers *Vertex Shaders* et *Fragment Shaders* liés aux cubes, aux lumières et le curseur
- **Doc** : Dossier dans lequel on retrouvera le rapport du projet et la documentation dynamique lié au projet
- **Include** : Dossier dans lequel on retrouve l'ensemble des fichiers .h ou .hpp (*header files*) lié au projet
- **Libs** : Dossier dans lequel on retrouve les librairies qu'on a utilisées lors du projet. Les librairies utilisées et présentes en local sont glimac, glm et ImGui
- **Src** : Dossier dans lequel on retrouve l'ensemble des fichiers .cpp lié au projet
- **.gitignore** : permet d'exclure les fichiers et dossiers non voulus au projet sur le repository (.DS\_Store : Fichier créer par les appareils MacOS)

### 4. Environnement de développement

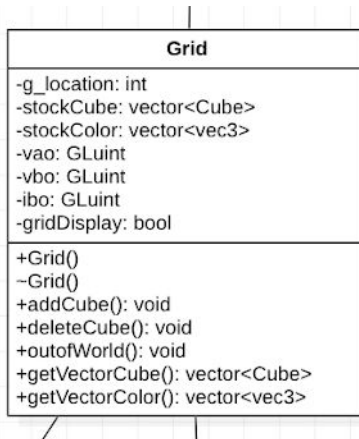
Notre binôme travaillait sous deux environnements différents : Linux et MacOS. Certains ajustements ont dû être opérés particulièrement pour *CMakeList.txt*. Le problème concernant l'environnement MacOS, c'est la difficulté à retrouver les *Find.cmake*, particulièrement celui de *Glew*. Le compilateur *CLang* semble chercher aux bonnes endroits, pourtant la commande *find\_package(GLEW)* ne permettait pas de retrouver le chemin vers la librairie. C'est pourquoi il est possible de retrouver à quelques endroits des conditions.

```
##### ----- Necessary in order to run on Apple's laptop ----- #####  
if (APPLE)  
    include_directories(/usr/local/include)  
    set(GLEW_LIBRARY /usr/local/lib/libGLEW.dylib)  
endif (APPLE)
```

```
#ifdef __APPLE__  
#include <OpenGL/gl.h>  
#else  
#include <GL/gl.h>  
#endif
```

## II. Les fonctionnalités

### 1. Affichage d'une scène avec des cubes



Pour gérer le “monde” dans lequel on travaille, nous avons choisi de créer une classe qui gère l’ensemble des cubes créés et définit un monde fini (20x20x10). Dans cette classe **Grid**, on stocke les cubes dans un vecteur. La grille gère la création et la suppression des Cubes.

```
void Grid::AddCube(const glm::vec3 &position, const glm::vec3 &color)
{
    int x = static_cast<int>(position.x) + 10;
    int y = static_cast<int>(position.y) + 10;
    int z = static_cast<int>(position.z) + 5;
    if (this->g_location[x][y][z] == 0)
    {
        Cube newCube(position, color);
        this->stockCube.push_back(newCube);
        this->g_location[x][y][z] = this->stockCube.size();
    }
}

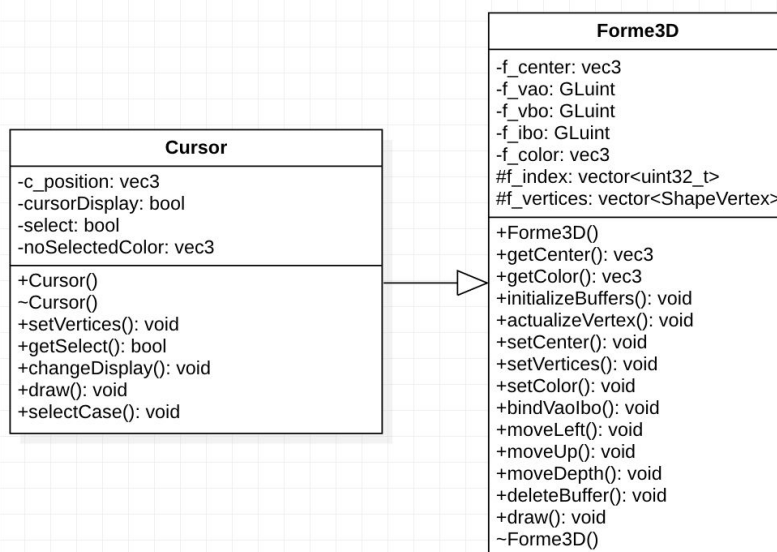
void Grid::deleteCube(const glm::vec3 &position)
{
    int x = static_cast<int>(position.x) + 10;
    int y = static_cast<int>(position.y) + 10;
    int z = static_cast<int>(position.z) + 5;
    for (int i = 0; i < 20; ++i)
    {
        for (int j = 0; j < 20; ++j)
        {
            for (int k = 0; k < 10; ++k)
            {
                if (this->g_location[i][j][k] > this->g_location[x][y][z])
                {
                    this->g_location[i][j][k] -= 1;
                }
            }
        }
    }
    this->stockCube.erase(this->stockCube.cbegin() + (this->g_location[x][y][z] - 1));
    this->g_location[x][y][z] = 0;
}
```

La classe **Grid** possède deux attributs essentiels dans la mise en place du logiciel :

Tableau statique de int qui représente toutes les positions de l’espace de travail ainsi qu’un vecteur de cube afin de stocker et supprimer dynamiquement les cubes.

Le tableau de int est initialisé à 0, à chaque création/suppression de cube, ce tableau est mis à jour : 0 si l’emplacement est vide, n pour le n-ième cube créé. Cette valeur permettra d’abord de vérifier l’état de l’emplacement, pour savoir s’il s’agit d’ajouter ou supprimer un cube mais aussi pour supprimer les cube car cette index sert en fait d’itérateur pour accéder au cube pour le supprimer.

## 2. Gestion du Curseur

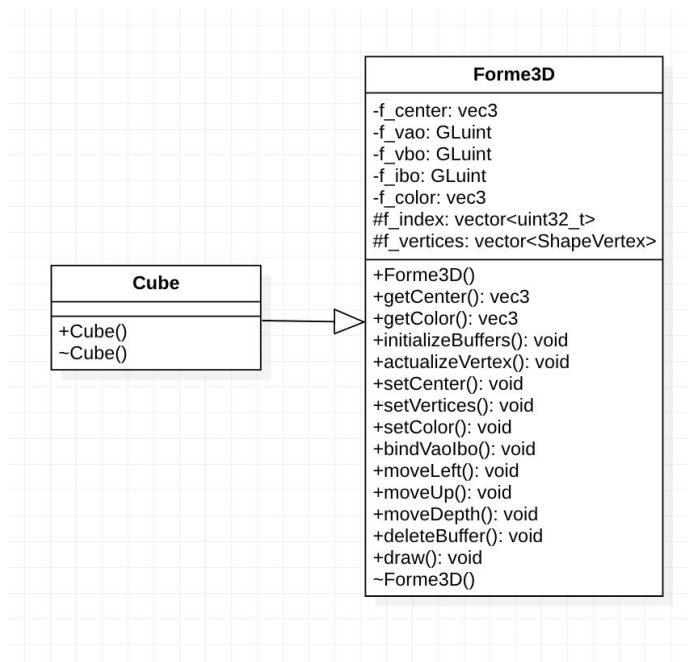


Nous avons décidé de créer une classe **Cursor** dérivée de la classe **forme3D**, tout comme notre classe **Cube**. En effet, nous avons d'abord codé **Cube** et **Cursor** séparément, pour au final se rendre compte qu'il était plus pratique et plus lisible de faire dériver nos deux classes d'une classe mère. **Cursor** ne se trace pas de la même façon qu'un cube et ne répond pas aux mêmes attentes. Il doit à la différence de la classe **Cube** posséder des attributs de "vérification", ou encore, il se trace différemment d'un cube. Comme vous pouvez le voir ci-dessous, trois fonctions **moveDepth()**, **moveUp()** et **moveLeft()** gèrent le déplacement du curseur. En plus des fonctions **getCenter()** pour obtenir la position courante du curseur et **getColor()** renvoie la couleur du curseur, ces cinq fonctions, sont les outils majeurs de l'interaction entre la classe **Grid** et les cubes, puisque **Grid** se sert de la position du curseur pour créer un cube.

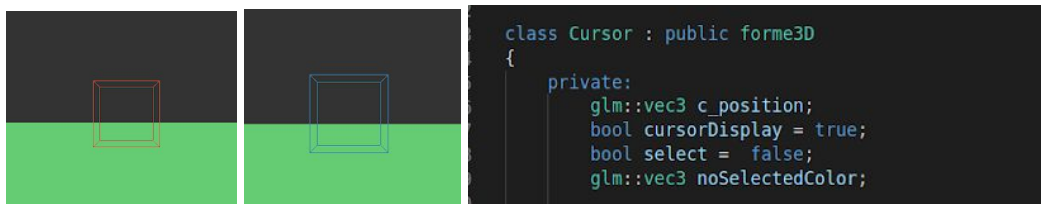
```
167 void forme3D::moveDepth(float delta)
168 {
169     for (int i = 0; i < this->f_vertices.size(); i++)
170     {
171         this->f_vertices[i].position.z += delta;
172     }
173     std::cout << "moveDepth axe z" << std::endl;
174     this->f_center.z += delta;
175 }
176
```

Ces fonctions sont appelées lorsque l'on clique sur A, Z, E, Q, S, D.

### 3. Edition des cubes

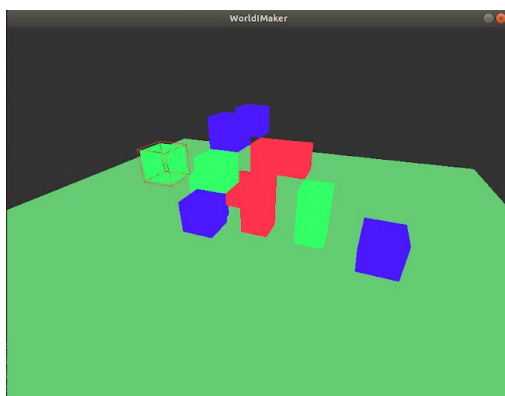


#### a) Sélection des cubes



La sélection des cubes se fait seulement si le curseur est “select”, il est rouge si l’on peut construire le terrain bleu sinon. Cela empêche déjà des erreurs de maladresses.

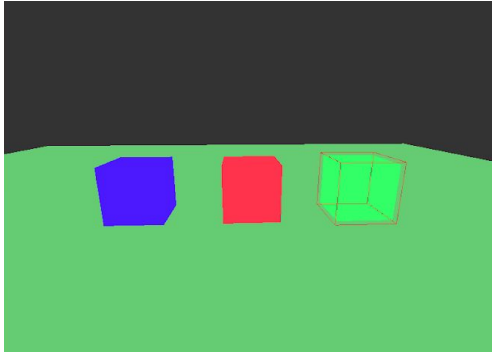
#### b) Changement de couleurs



On peut choisir trois couleurs différentes pour les cubes : rouge (clic “R”), vert (clic “G”) et bleu (clic “B”). Sans oublier de mettre le curseur en mode “select”.

#### 4. Sculpture du terrain:

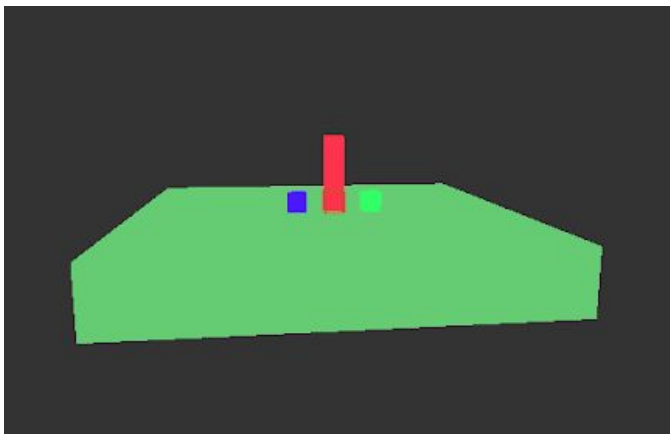
##### a) Ajout - suppression de bloc



Il suffit d'appuyer sur la barre espace pour créer ou supprimer les cubes.

Comme dit précédemment, c'est la classe **Grid** qui gère l'addition (**AddCube(vec3,vec3)**) et la suppression (**deleteCube(vec3)**) de cubes.

##### b) Extrude - Intrude



En cliquant sur "w", on empile (**extrudeCube(vec3,vec3)**) les cubes et en cliquant "x", on effectue l'action inverse (**digCube(vec3)**).

#### 5. Lumières

Light
-l_Kd: vec3 -l_Ks: vec3 -l_Shininess: float -l_LightDir: vec3 -l_LightIntensity: vec3 +lightProgram: Program
+Light() ~Light() +moveLeft(): void +moveUp(): void +moveDepth(): void +lightInitUniVariable(): void +lightApplication(): void

Dans les TP de synthèse d'image, nous avons vu que la lumière est un ensemble de variables envoyé aux shaders. Il ne nous fallait qu'une fonction liant les valeurs de la lumière aux shaders.

Nous avons implémenté trois modes d'aperçu du monde : Sans lumière, avec une lumière directionnelle ou avec un point de lumière. Pour accéder à ces différentes vues, il suffit de cliquer sur "V".

Cela amène au point suivant : la mise en place des shaders

## 6. Les shaders

```
[ 90%] Building CXX object CMakeFiles/WorldMaker.dir/src/main.cpp.o
[100%] Linking CXX executable bin/WorldMaker
[100%] Built target WorldMaker
line:1: GLSL330: /media/line/ubuntu/IMAC2/Synthèse d'image/OPENGL_build$ make
[ 28%] Built target glimac
[ 50%] Built target Input
Scanning dependencies of target WorldMaker
[ 60%] Building CXX object CMakeFiles/WorldMaker.dir/src/main.cpp.o
[ 64%] Linking CXX executable bin/WorldMaker
CMakeFiles/WorldMaker.dir/src/main.cpp.o : Dans la fonction « main » :
main.cpp:(.text+0x31e) : référence indéfinie vers « glimac::Program::operator=(glimac::Program const&) »
collect2: error: ld returned 1 exit status
CMakeFiles/WorldMaker.dir/build.make:333: recipe for target 'bin/WorldMaker' failed
make[2]: *** [bin/WorldMaker] Error 1
CMakeFiles/WorldMaker.dir/build.make:333: recipe for target 'CMakeFiles/WorldMaker.dir/all' failed
make[1]: *** [CMakeFiles/WorldMaker.dir/all] Error 2
CMakeFiles/WorldMaker.dir/build.make:333: recipe for target 'all' failed
make: *** [all] Error 2
line:1: GLSL330: /media/line/ubuntu/IMAC2/Synthèse d'image/OPENGL_build$
```

Nous avons donc choisi de conserver glimac pour la réalisation du projet puisque nous voulions nous resservir des TP réalisés. Cependant, nous avons eu beaucoup de problèmes au niveau des constructeur et destructeur de la classe

**Program** de glimac.

Finalement, nous avons réussi à “contourner” le problème en créant un header **imakerProgram** définissant les programmes liant les shaders à notre code. Nous avons en tout quatre fragments shaders : un utilisé pour un affichage sans lumière, un gérant la lumière directionnelle, un autre le point de lumière et enfin celui gérant le curseur. Chaque programme est appelé directement dans le main.

```
imakerProgram.hpp
> imakerProgram.hpp > ...
#pragma once
#include <../libs/glimac/include/Program.hpp>
#include <../libs/glimac/include/FilePath.hpp>

struct DirLightProgram {
    glimac::Program m_Program;

    GLint uMVPMatrix;
    GLint uWMatrix;
    GLint uNormalMatrix;

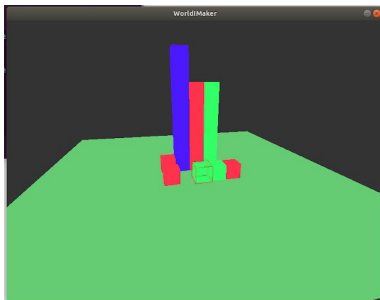
    DirLightProgram(const glimac::FilePath& applicationPath):
        m_Program(loadProgram(applicationPath.dirPath() + "../assets/shaders/3D.vs.glsl",
                                applicationPath.dirPath() + "../assets/shaders/lightShader/dirLight.fs.glsl")) {
        uMVPMatrix = glGetUniformLocation(m_Program.getGLId(), "uMVPMatrix");
        uWMatrix = glGetUniformLocation(m_Program.getGLId(), "uWMatrix");
        uNormalMatrix = glGetUniformLocation(m_Program.getGLId(), "uNormalMatrix");
    }
};

struct PointLightProgram {
    glimac::Program m_Program;

    GLint uMVPMatrix;
    GLint uWMatrix;
    GLint uNormalMatrix;

    PointLightProgram(const glimac::FilePath& applicationPath):
        m_Program(loadProgram(applicationPath.dirPath() + "../assets/shaders/3D.vs.glsl",
                                applicationPath.dirPath() + "../assets/shaders/lightShader/pointLight.fs.glsl")) {
        uMVPMatrix = glGetUniformLocation(m_Program.getGLId(), "uMVPMatrix");
        uWMatrix = glGetUniformLocation(m_Program.getGLId(), "uWMatrix");
        uNormalMatrix = glGetUniformLocation(m_Program.getGLId(), "uNormalMatrix");
    }
};

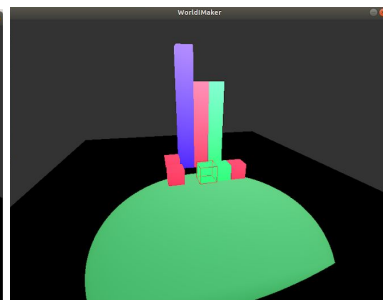
struct SceneProgram {
    glimac::Program m_Program;
};
```



Sans lumière



DirectionLight



PointLight



## 7. Génération procédurale

Interpolation
-_numberControlPoint: size_t -controlPoints: vec3 -weightControlPoint: VectorXd -omegas: VectorXd
+Interpolation() ~Interpolation() +computeOmegas(): VectorXd +evaluatePoint(): double -generateControlPoint(): vec3 -generateWeightControlPoint(): VectorXd -phiDistance(): double -phiQuad(): double -phiInvQuad(): double -phiGauss(): double +generateCubes(): void -generateColor(): vec3

La classe Interpolation regroupe beaucoup de **méthodes** et d'**attributs privés**. Les fonctions sont pour la plupart appelées entre elles, et seul la fonction de génération de terrain doit être public afin d'y avoir accès dans notre *main.cpp*.

La génération procédurale de ce projet repose sur l'**aléatoire**. Elle est appliquée à l'ensemble de notre grille d'une dimension de 20 x 20 x 10. La suppression de la scène pour en régénérer une nouvelle peut donc prendre un peu de temps.

La génération procédurale se base avant tout sur les points de contrôle qu'on attribue. Dans ce projet il était possible de **définir des points de contrôle** directement **dans le code**, mais si on faisait cela, la génération serait que **très peu dynamique**. Il est également possible de **lire un fichier .txt** qu'on édite pour affecter des points de contrôle.

Dans notre cas, nous avons préféré une **génération aléatoire** de ces points de contrôle ainsi que leur poids.

Pour ce faire, nous avons employé des fonctions de la STL : *ranlux24\_base*, *uniform\_real\_distribution*... Nous partons d'une seed définie par le temps.

```
60 std::vector<glm::vec3> Interpolation::generateControlPoint() {  
61  
62     std::vector<glm::vec3> controlPointVec;  
63     // select seed from time  
64     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();  
65  
66     std::ranlux24_base ranlux24Generator(seed);  
67  
68     std::uniform_int_distribution<int> uniformIntDistribution( a: 1, b: 10);  
69     for (size_t i = 0; i < _numberControlPoint; ++i) {  
70  
71         glm::vec3 controlPoint(uniformIntDistribution( & ranlux24Generator),  
72                               uniformIntDistribution( & ranlux24Generator),  
73                               uniformIntDistribution( & ranlux24Generator));  
74         controlPointVec.push_back(controlPoint);  
75  
76     }  
77  
78     return controlPointVec;  
79 }
```

Les fonctions de génération auraient pu former une classe *Generator* à part entière afin d'améliorer son **accessibilité** entre les classes et la **cohérence** du projet. Effectivement la génération de couleurs aléatoire aurait pu être une composante intéressante à employer à d'autres endroits.

Le défaut de l'aléatoire, c'est qu'il est difficile de comparer les fonctions radiales entre elles, car les points de contrôle et leur poids changent constamment. Tout de même nous avons pu observer quelques différences sur la génération.

Une fois les points de contrôle générés, le calcul des omégas est standard. L'inversion de la matrice est effectuée par la fonction `.inverse` de la librairie Eigen. Eigen semble utiliser une **décomposition LU**, soit en pivot partiel (si on pose 4 points de contrôle ou moins) sinon pivot total.

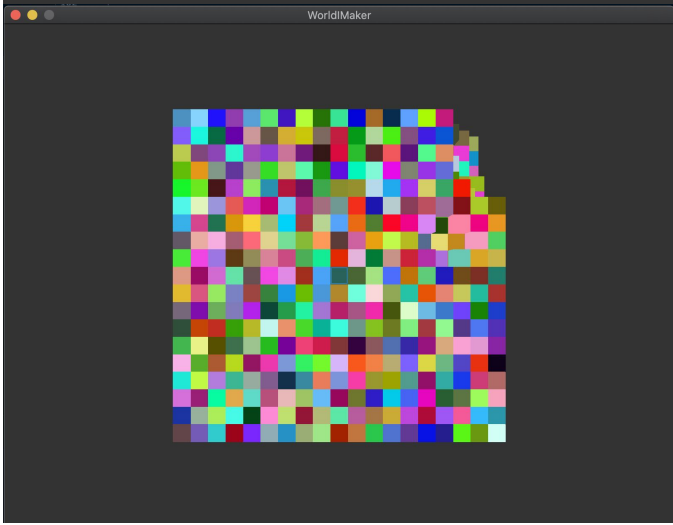
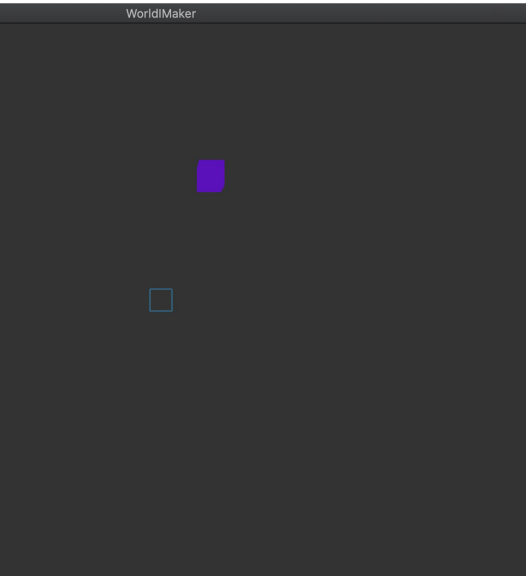
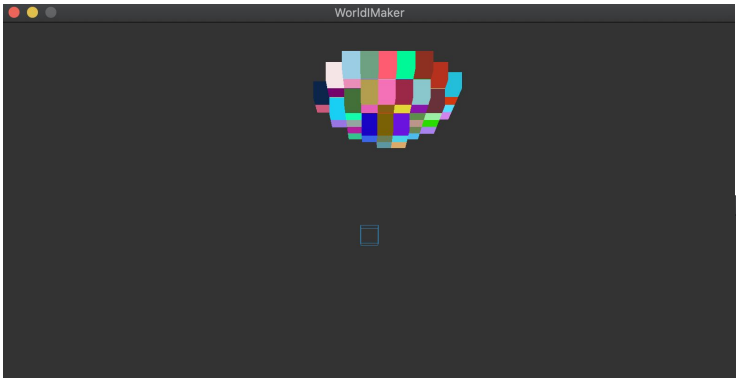
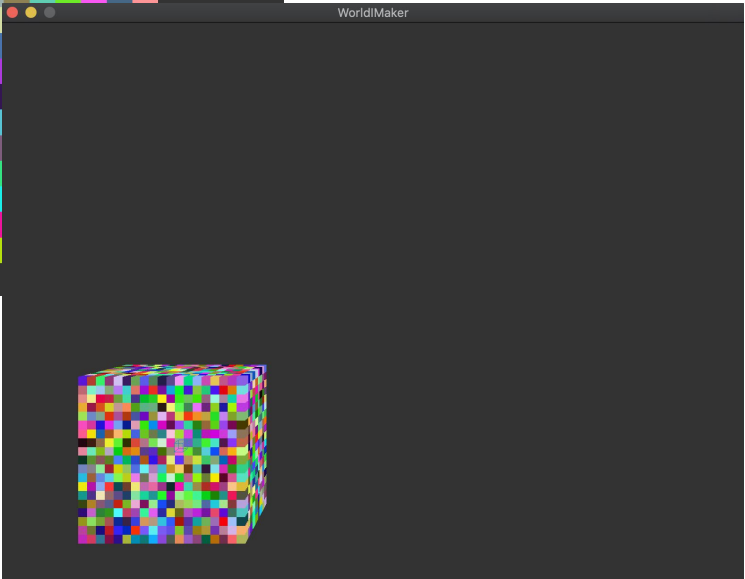
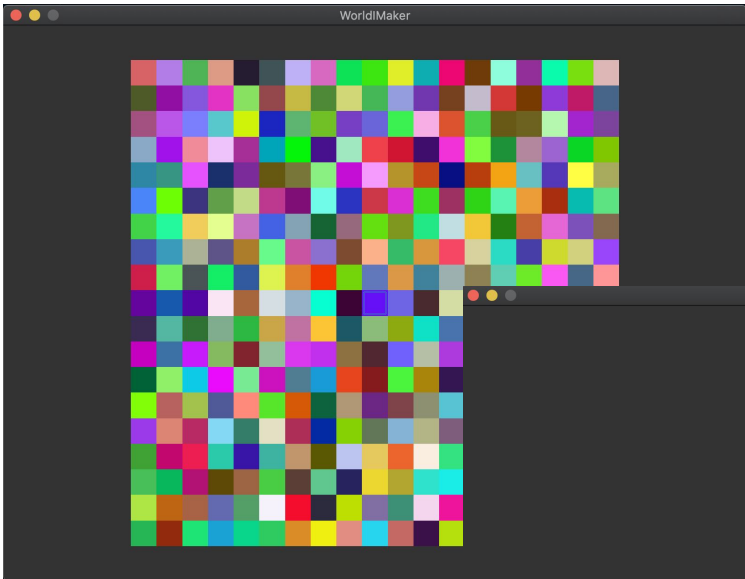
```
31 Eigen::VectorXd Interpolation::computeOmegas() {
32     controlPoints = generateControlPoint();
33     weightControlPoint = generateWeightControlPoint();
34
35
36     Eigen::MatrixXf phiMatrix = Eigen::MatrixXf::Zero( rows: controlPoints.size(), cols: controlPoints.size());
37
38     // Parse our matrix in order to fill with distance between two control points
39     for (size_t i = 0; i < controlPoints.size(); ++i) {
40         for (size_t j = 0; j < controlPoints.size(); ++j) {
41             phiMatrix(i, j) = phiGauss( a: controlPoints[i], b: controlPoints[j]);
42         }
43     }
44
45
46     return phiMatrix.inverse() * weightControlPoint;
47 }
```

La génération de notre scène consiste donc d'évaluer tous les points de notre grille en fonction de nos omegas calculés précédemment. Si le poids en ce point est **positif**, nous **affichons** un cube. La complexité de l'algorithme suivant est de l'ordre  $O(n^3)$  dû aux 3 boucles for afin de parcourir notre grille.

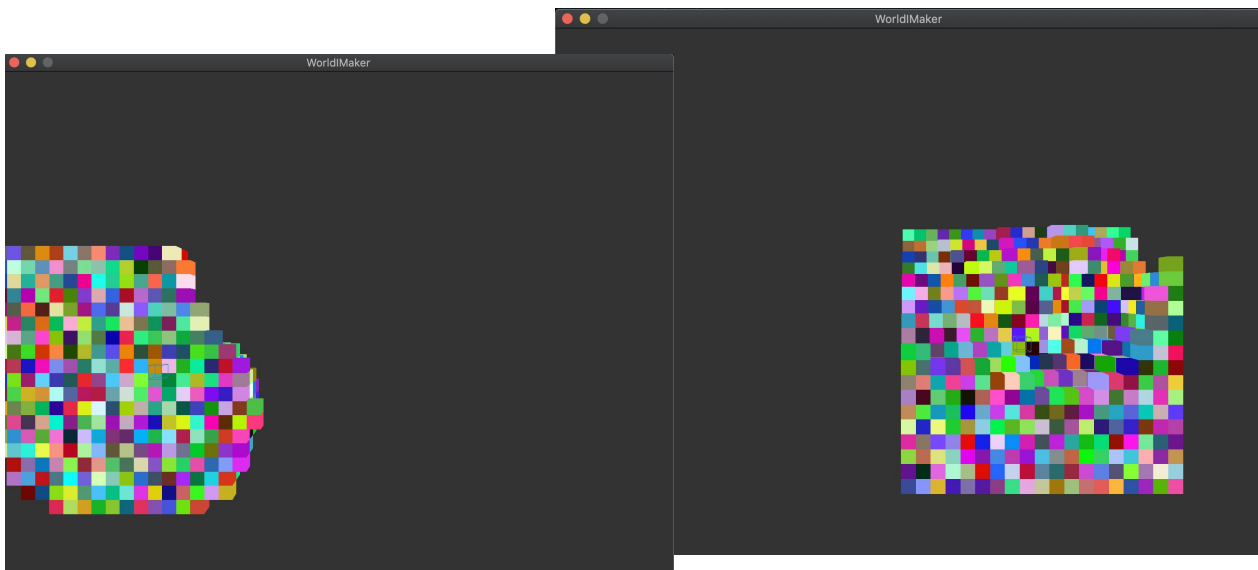
```
116 void Interpolation::generateCubes(Grid &grid) {
117     omegas = computeOmegas();
118     grid.refreshGrid();
119
120     for (int i = -9; i < 10; ++i) {
121         for (int j = -9; j < 10; ++j) {
122             for (int k = -4; k < 5; ++k) {
123                 glm::vec3 currentPoint(i, j, k);
124                 double weightPoint = evaluatePoint(currentPoint);
125
126                 if (weightPoint >= 0) {
127                     grid.AddCube(currentPoint, color: generateColor());
128                 }
129             }
130         }
131     }
132 }
```

- Fonction radiale basique  $\phi(d) = d$  - Points de contrôle : 3

La génération procédurale à 3 points de contrôle rend des scènes très basiques et peu variées. Il est possible de rencontrer plus d'une fois une scène totalement vide ou totalement remplie. Sachant qu'on possède une grille limitée, nous obtenons très souvent une scène en forme de cube.

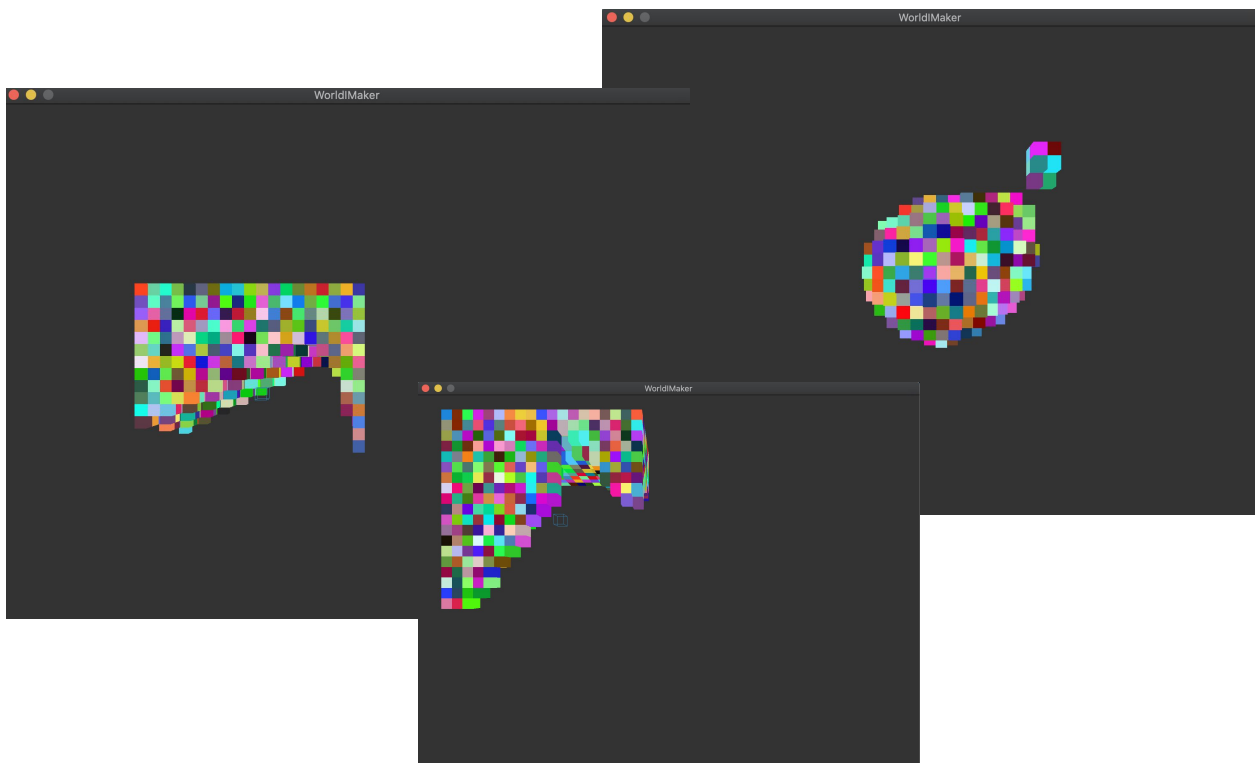


- Fonction radiale basique  $\phi(d) = d$  - Points de contrôle : 8

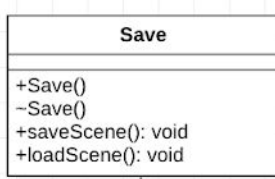


- Fonction radiale gaussienne - Points de contrôle : 8

La génération avec une fonction radiale gaussienne complexifie la scène et sur la génération des cubes.



## 8. Sauvegarde / Chargement de la scène

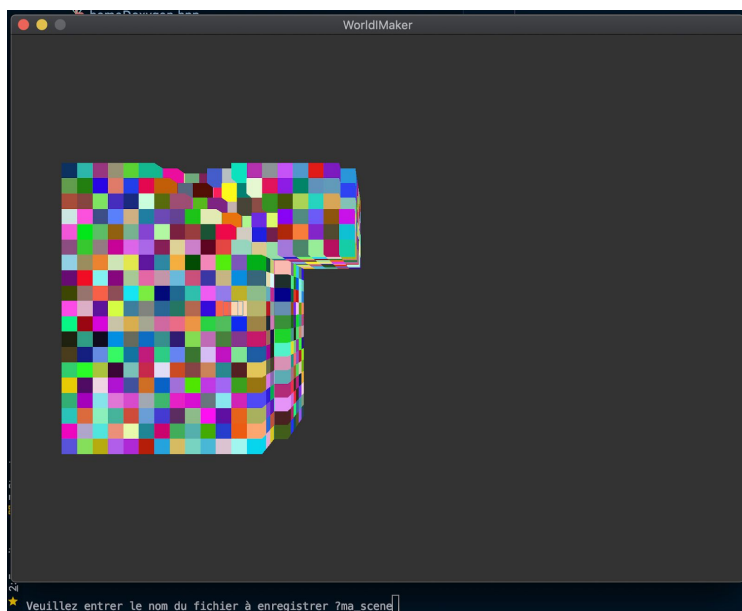


La sauvegarde et le chargement d'une scène s'effectue grâce au clavier : K pour sauvegarder et L pour charger. Étant donné que le projet ne possède pas d'interface graphique, nous utilisons la console pour nommer le fichier exporté ainsi que pour l'import.

Les fonctions présentes dans la classe Save aurait pu être implémentées directement dans la classe Grille. Cependant nous avons préféré créer une classe à part entière pour un soucis de lisibilité. Il est également plus intéressant de segmenter des features pour éviter les conflits lorsque l'on merge la branche en question.

```
1 void Save::saveScene(std::vector<Cube> vectorCube) {
2
3     std::string nameFile;
4     std::cout << "Veuillez entrer le nom du fichier à enregistrer ?";
5     std::cin >> nameFile;
6
7     std::ofstream outfile;
8     outfile.open( s: "./bin/" + nameFile, mode: std::ios::out | std::ios::binary);
9
10    if (!outfile.is_open()) {
11        std::cerr << "Can not create file named " + nameFile + "." << std::endl;
12    }
13
14    outfile << vectorCube.size() << std::endl;
15    for (auto & i : vectorCube) {
16        outfile << i.getCenter().x << " ";
17        outfile << i.getCenter().y << " ";
18        outfile << i.getCenter().z << " ";
19        outfile << i.getColor().r << " ";
20        outfile << i.getColor().g << " ";
21        outfile << i.getColor().b << " ";
22    }
23
24    outfile.close();
25 }
```

Afin de développer ces fonctionnalités, nous avons utilisés les classes de stream (ifstream/ofstream). En parcourant notre vecteur cube, nous écrivons sur un fichier texte la position et la couleur de tous nos cubes. Nous notons également au début du fichier le nombre de cubes inscrit dans ce dernier afin de faciliter son parcours lors de la lecture.



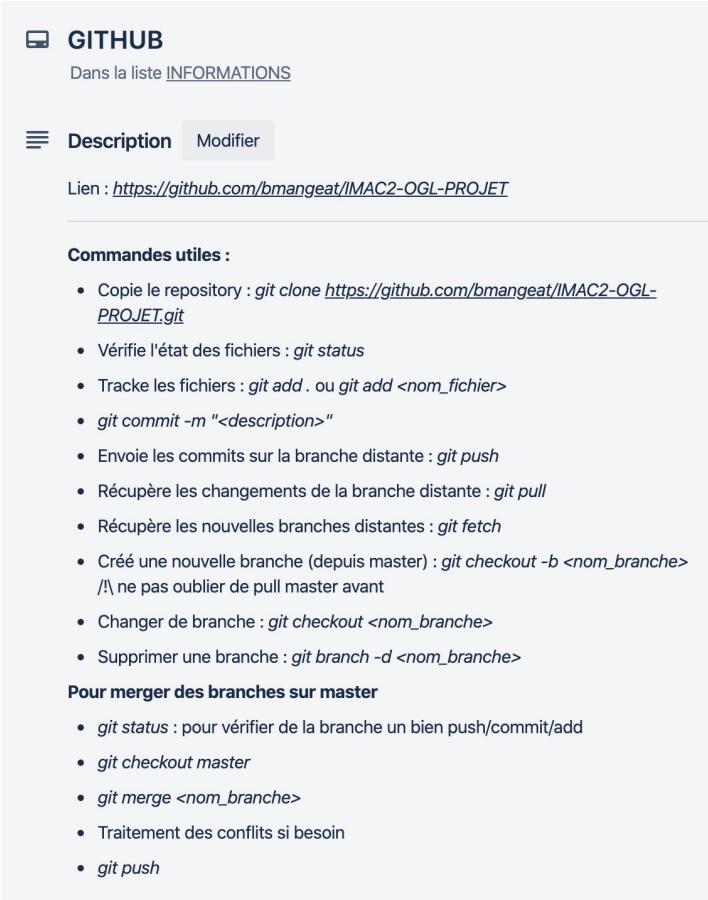
Les exports sont situés dans le dossier *bin* du dossier *build*

## C. Gestion de projet

### 1. Outils

#### - Git :

Le fonctionnement de notre git. Chaque feature est développée sur une branche de notre repository afin de pouvoir concevoir les outils à part et à différents moments. La fin de vie d'une branche sur le projet correspond à la fin du développement de la feature concernée. Les branches ne sont pas supprimées pour pouvoir revenir à une version ultérieure si besoin. L'ensemble des commandes et pseudo-protocole était disponible sur notre Trello.



**GITHUB**  
Dans la liste [INFORMATIONS](#)

**Description** Modifier

Lien : <https://github.com/bmangeat/IMAC2-OGL-PROJET>

**Commandes utiles :**

- Copie le repository : `git clone https://github.com/bmangeat/IMAC2-OGL-PROJET.git`
- Vérifie l'état des fichiers : `git status`
- Tracke les fichiers : `git add` . ou `git add <nom_fichier>`
- `git commit -m "<description>"`
- Envoie les commits sur la branche distante : `git push`
- Récupère les changements de la branche distante : `git pull`
- Récupère les nouvelles branches distantes : `git fetch`
- Créé une nouvelle branche (depuis master) : `git checkout -b <nom_branche>`  
/!\ ne pas oublier de pull master avant
- Changer de branche : `git checkout <nom_branche>`
- Supprimer une branche : `git branch -d <nom_branche>`

**Pour merger des branches sur master**

- `git status` : pour vérifier de la branche un bien push/commit/add
- `git checkout master`
- `git merge <nom_branche>`
- Traitement des conflits si besoin
- `git push`

#### - Documentation :

Doxygen est un outil performant pour réaliser une documentation dynamique et accessible. Nous avons rencontré quelques difficultés pour définir la page main de notre projet. C'est pourquoi nous avons créé un header file qu'on a défini en tant qu'index de notre documentation.

## - ImGui :

La librairie ImGui a été implémentée dans le projet, mais l'interface graphique ne s'affiche pas proprement. La position de l'interface graphique ImGui se positionne en bas de l'écran et ne recouvre pas l'intégralité de l'écran. Par ailleurs, malgré l'affichage d'une fenêtre ImGui, l'interface semble réagir en décalé avec la souris.

*Exemple : Le menu avec l'onglet "Files" ne réagit seulement si on se dirige sur la partie haute de la fenêtre de l'application.*

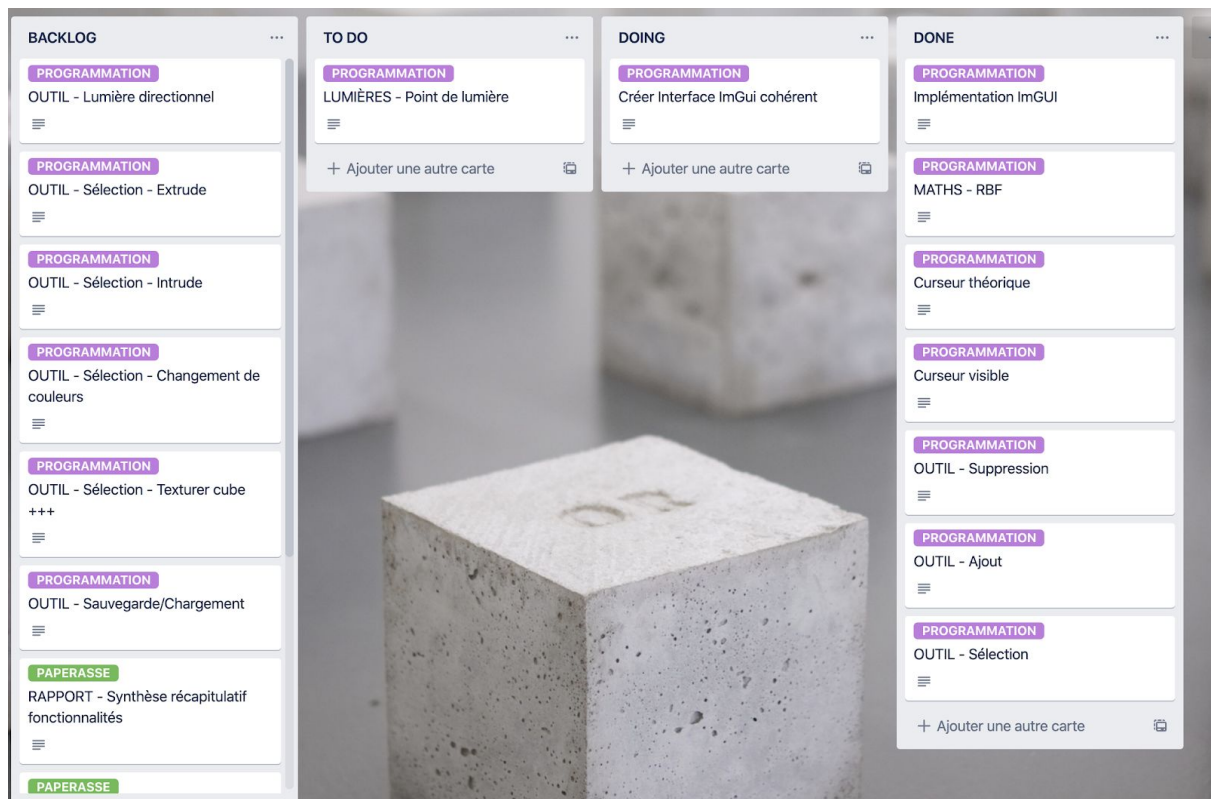


## - Trello :

L'organisation de notre projet réside dans un Trello. Nous avons constitué des tickets pour l'ensemble des fonctionnalités à développer. Les tickets peuvent avoir différents états :

- > backlog : la réserve de ticket pour les tâches qui devaient être réalisées dans une temporalité plus éloignées
- > to do : les tickets à réaliser sous une semaine
- > doing : tickets en cours de traitement
- > to do : tickets terminés

Une gestion des tickets assez standard dans les méthodes agiles.



## C. Rapport individuel

### - Brice Mangeat :

Le projet dans son ensemble s'est bien passé. Dès le début, Line et moi avions le même objectif : un code clair et compréhensible. Nous mettons du temps à concevoir les outils pour comprendre chaque ligne qu'on faisait, nous privilégions la qualité à la quantité. Cependant



cette objectif est dur à tenir lorsque la date du rendu rapproche. Tout de même, nous avons pu poser les bases du projet proprement tels que le CmakeList ou bien la conception de la grille.

L'OpenGL est très intéressant à apprendre et demande beaucoup de compétences différentes. Je ne pense pas les avoir acquis à 100%, mais j'ai saisi la problématique du domaine. Cependant il aurait été intéressant d'avoir davantage de temps pour chercher des fonctions OpenGL qui je pense auraient facilité le développement du projet. En effet, je suppose que nous n'avions qu'une petite partie de l'OpenGL et que tout peut être optimisé.

La partie mathématique abordait la radial basis function que j'ai trouvé très amusant. Par ailleurs utiliser les outils qu'on a vu en travaux pratiques en mathématiques tels les fonctions de génération...

Enfin très intéressant de concevoir un cmakeList fonctionnel pour les deux environnements de développement.

#### **- Line Rathonie :**

Le projet de programmation/synthèse d'image était celui que j'attendais le plus ce semestre. Je voulais en quelques sortes "prendre ma revanche" sur le projet de programmation d'IMAC1 que je n'ai pas pu aboutir comme je le voulais. Avec Brice, j'ai le sentiment que l'on a relevé le défi. Nous n'avons peut-être pas réalisé tout ce que nous voulions faire, à cause d'une mauvaise gestion du temps ou de problèmes longs à résoudre. Cependant, je suis bien plus satisfaite du résultat que l'an passé, même si j'avoue que j'aimerais continuer à développer ce projet qui n'a que les bases. Cette année, je me suis vraiment impliquée dans le projet, en réfléchissant "par moi-même" et j'ai compris tout ce que j'ai pu entreprendre même si j'aurai aimé faire beaucoup plus.

Dans ce projet, je me suis davantage concentrée sur le côté C++ et OPENGL. La programmation est un domaine qui me plaît de plus en plus, j'ai réalisé que je pouvais passer des heures à travailler sur ce projet sans m'en lasser. Même faire face aux erreurs de code m'amusait.