

# Document de design

Présenté à M. Philippe Voyer

Université Laval  
IFT-3100 Infographie

Équipe 04:  
David D'Anjou  
Johny Savard  
Alexis Roberge  
Vincent Laverdure

2 mai 2025

# Table des matières

[Table des matières](#)

[Sommaire](#)

[Interactivité](#)

[Technologie](#)

[Compilation](#)

[Télécharger et installer openFrameworks](#)

[Architecture](#)

[Fonctionnalités](#)

[Module 6: Texture:](#)

[6.1 Coordonnées de texture](#)

[6.2 Filtrage](#)

[6.3 Mappage Tonal](#)

[6.4 Cubemap](#)

[Module 7: Illumination classique](#)

[7.2 Matériaux](#)

[7.3 Types de lumière](#)

[7.4 Lumières multiples](#)

[Module 8:Topologie](#)

[8.1 Courbes paramétriques](#)

[8.2 Surface paramétrique](#)

[8.3 Shader de tessellation](#)

[Module 10: Illumination moderne](#)

[10.1 PBR](#)

[10.2 Métallicité et 10.3 Microfacettes](#)

[10.4 Éclairage environnemental](#)

[Ressources](#)

[Ressources pour la seconde partie du travail:](#)

[Présentation](#)

[Déclaration globale de l'usage de l'intelligence artificielle:](#)



# Sommaire

Ce projet a pour objectif de développer une application interactive permettant à l'utilisateur de créer, manipuler et modifier divers éléments graphiques en 2D et en 3D, incluant des primitives, des formes vectorielles et des images. L'application repose sur openFrameworks en C++, une bibliothèque adaptée aux applications graphiques et interactives.

L'idée initiale était de concevoir un outil de test de performances pour les cartes graphiques, mais le projet a évolué vers un éditeur graphique, permettant de travailler sur des scènes combinant des éléments 2D et 3D.

L'application est divisée en deux volets :

- Mode 2D
  - Dessin de primitives vectorielles (ligne, rectangle, ellipse, etc.).
  - Importation et modification d'images (ajustement des couleurs en RGB et HSB).
  - Application de filtre aux images
    - Flou
    - sharpness
    - Mappage tonal
    - grayscale
  - Sélection et modification des objets vectoriels dans la scène.
  - Courbes paramétriques
- Mode 3D
  - Affichage d'une scène 3D avec contrôle de la caméra.
  - Importation d'objets 3D.
  - Création et manipulation d'éléments géométriques en 3D
  - Ajout d'illumination sur les structures précédentes
  - Environnement permettant une simulation en temps réel (lumière et textures)
  - Surface paramétrique



# Interactivité

Pour l'import d'images, il suffit simplement de prendre un fichier png externe à l'application et le glisser dans la fenêtre. Il est ensuite possible de bouger l'image à l'intérieur de la fenêtre pour la positionner, simplement en cliquant sur l'image et en la glissant à la position voulue. Lors de l'import d'une image, l'application ouvre automatiquement une seconde fenêtre pour représenter l'histogramme de l'image. Pour cesser de le visualiser, il suffit de fermer la fenêtre.

Pour modifier les attributs concernant le blur, le mappage tonal, le grayscale ou le sharpness d'une image importée, il suffit de sélectionner l'image avec le clic gauche de la souris, puis de modifier les valeurs dans l'interface à gauche.

Pour les courbes et surfaces paramétriques, on peut sélectionner chaque point de contrôle dans l'interface avec le slider de points de contrôles (0-7 pour courbe paramétrique et 0-15 pour surface paramétrique). On peut ensuite bouger le slider de hauteur du point de contrôle, pour modifier la hauteur du point de contrôle sélectionné par rapport à sa position d'origine (de -200 à 200). Pour la courbe paramétrique, elle possède 4 points de contrôle par défaut, mais il est possible d'en ajouter jusqu'à 8 avec le slider nombre de points de contrôle. Pour la surface paramétrique, il est possible d'ajouter un certain niveau de tessellation pour modifier la précision de l'apparence de la surface.

Pour l'illumination, il suffit de cliquer sur le type de lumière voulu dans le menu de gauche, puis d'utiliser le bouton droit de la souris pour placer une lumière. Selon le type de lumière, l'orientation et/ou la position de cette dernière sera placée selon l'orientation et la position de la caméra. La position du clique dans l'écran n'a pas d'importance. Lorsque placé, les lumières se retrouvent dans la liste d'objets et peuvent être sélectionnée et supprimé. Un nouveau bouton a été ajouté pour la partie 3D pour supprimer toutes les lumières et les objets de la scène. On peut



aussi appliquer un matériau parmi 3 choix et un par défaut. Les 3 matériaux sont basés sur l'état courant des matériaux de la scène de l'autre fenêtre. Lorsque sélectionné, un matériau semblable au matériau par défaut est utilisé avec une autre couleur.

Pour la partie PBR il y a plusieurs interactions possibles avec le menu principal de gauche dans la scène 3d pour contrôler les niveaux de métallicité, rugosité et les positions des textures. Il est également possible d'activer ou désactiver les lumières avec la touche 1 (pour lumière ambiante) et 3 pour la lumière de la skybox. La touche 2 active ou désactive le déplacement des sphères pour mieux analyser les changements des matériaux. Finalement la touche 0 permet d'alterner entre les 3 skybox de la scène.



# Technologie

Tout comme la première partie de ce projet, l'application est développée en C++17 en utilisant openFrameworks version 0.12.0 qui est la dernière version disponible à ce jour sur le site officiel d'openFrameworks: [download | openFrameworks](#). Ce framework est spécialisé dans la modélisation graphique et basé sur OpenGL. Si ce projet était à recommencé nous aurions peut-être utiliser DirectX à la place, car beaucoup de notre temps était de chercher de la documentation désuète ou tout simplement inexiste.

Pour la gestion des interfaces utilisateur et l'affichage des formes en 2D et 3D, nous avons intégré deux bibliothèques essentielles : ofxGui, qui facilite la création d'interfaces graphiques interactives, et ofxAximpModelLoader, permettant le chargement et le rendu de modèles 3D. Le projet s'appuie également sur la Standard Template Library (STL) de C++, notamment pour le stockage et la manipulation des formes graphiques et des données d'histogramme.

IDE:

Le développement a été réalisé sous Visual Studio 2022, car il est un des logiciels recommandé lors de l'utilisation d'openFrameworks et que plusieurs d'entre nous l'avons déjà utilisé au moins une fois dans le passé. Avec du recul nous aurions peut-être opté pour Clion, car plusieurs d'entre nous avons éprouvé des difficultés lors des "pull" dans GitLab. Il semblerait que les dossiers .vs et obj créait beaucoup de problème et devions alors les supprimer à presque chaque modification d'un autre membre de l'équipe.

Gestionnaire de code:

La gestion du code source, des versions et des collaborations en équipe a été assurée via Git et GitLab qui est la seconde plateforme la plus utilisée pour le travail collaboratif. Comme cette dernière a été utilisée pour le cours GLO-2004 pour plusieurs d'entre nous, il semblait être le choix le plus logique pour ce projet.



Système d'exploitation:

Nous sommes tous sous plateforme x64 avec une combinaison de Windows 10 et Windows 11.

Un des membres de l'équipe a changé de Mac OS à Windows 11 en cours de projet, car plusieurs problèmes de compatibilité se faisaient ressentir avec les nouvelles puces M4 d'Apple.

## Compilation

Télécharger et installer openFrameworks

1. Rendez-vous sur le site officiel : <https://openframeworks.cc/download/>
2. Téléchargez la version correspondant à votre système d'exploitation (Windows, macOS ou Linux).
3. Extrayez l'archive dans un répertoire de travail (ex: [C:/openFrameworks/](#) sous Windows).

Télécharger la dernière version de visual studio à cet endroit:

<https://visualstudio.microsoft.com/downloads/>. Une fois installé, ouvrir un nouveau projet en utilisant l'option de cloner un référentiel en utilisant l'adresse suivante:

<https://gitlab.com/JohnySavard/infographie.git>. Il est important de cloner le projet dans le fichier d'installation d'openframeworks à cet endroit: \openframeworks\apps\myApps/.

Si vous n'utilisez pas le référentiel, simplement télécharger le projet fourni et le déposer au même endroit que mentionné précédemment.

Par la suite, ouvrez le project generator d'openframeworks:

\openframeworks\projectGenerator\projectGenerator.exe. Il suffit de créer un nouveau projet en



sélectionnant 2 addons supplémentaires: ofxAssimpModelLoader et ofxGui. En cliquant sur generate, le projet est lancé dans visual studio, ou il faudra supprimer les fichiers ofapp.h et ofapp.cpp. (il peut être nécessaire de supprimer le fichier .vs qui est caché dans le répertoire, selon votre configuration). Finalement il suffit d'exécuter l'application dans visual studio!

## Architecture

Notre logiciel n'a pas beaucoup changé en termes d'architecture pour la seconde partie du travail.

Notre application repose sur une structure de base, largement inspirée des laboratoires du cours, et adopte une forme simplifiée du modèle MVC (Modèle-Vue-Contrôleur). L'organisation du projet repose sur plusieurs classes ayant chacune un rôle distinct dans le fonctionnement global du programme.

La classe Application joue le rôle de contrôleur principal, en assurant la gestion des événements, des interactions utilisateur et de la logique générale du programme. Elle orchestre le comportement des autres modules et leur permet de communiquer entre eux.

La classe Renderer est responsable du rendu graphique, traitant à la fois l'affichage des formes vectorielles et des images importées. Elle permet d'appliquer différentes transformations et ajustements aux objets affichés à l'écran.

La classe HistogramWindow est dédiée à l'analyse des images. Elle génère un histogramme des couleurs d'une image sélectionnée et le met à jour en fonction des modifications effectuées par l'utilisateur. Cet histogramme offre une représentation visuelle de la répartition des couleurs et permet une meilleure compréhension des ajustements appliqués.

La classe PortalWindow est chargée de gérer un portail offrant une vue alternative sur la scène. Elle permet d'afficher un point de la scène sous une perspective différente en utilisant une



seconde caméra, offrant ainsi une fonctionnalité supplémentaire pour l'exploration et la manipulation d'objets dans l'espace de travail.

Par manque d'expérience avec openFrameworks, l'architecture du projet n'a pas été découpée de manière optimale. Les classes restent volumineuses et regroupent plusieurs responsabilités, ce qui pourrait être amélioré en affinant la séparation des modules et en suivant une approche plus rigoureuse du génie logiciel. Un découpage plus structuré permettrait de rendre le projet plus maintenable et plus flexible dans l'éventualité où nous souhaiterions continuer de le développer.

La classe Lighting prend le renderer de Application lorsque créé. Lorsque le mode de l'application est 2D, on dessine directement le Renderer depuis l'application. Si c'est 3D, Application dessine Lighting qui elle applique la lumière et dessine Renderer lorsque la lumière est activée.



# Fonctionnalités

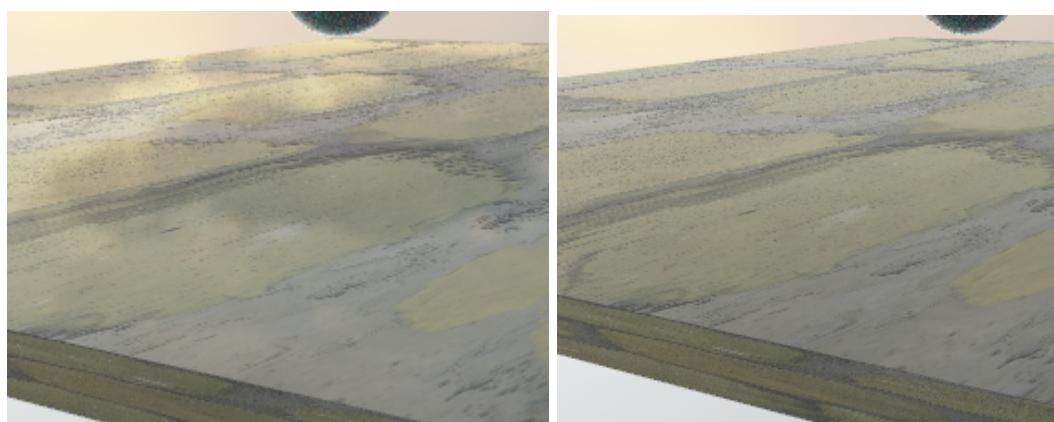
## Module 6: Texture:

### 6.1 Coordonnées de texture

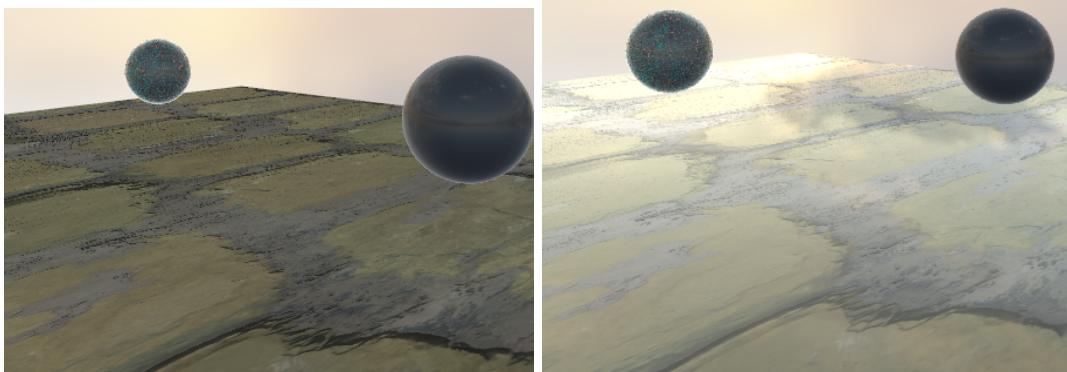
Nous avons implémenté les coordonnées de texture sur la base de notre modèle 3d comme suit:



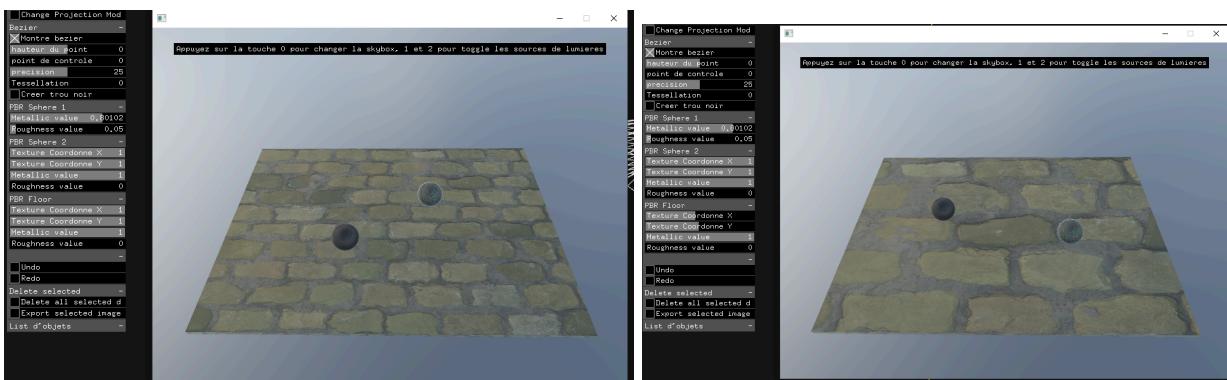
La base en brique est faite avec un maillage géométrique 3d texturé. Nous pouvons augmenter ou diminuer le facteur de rugosité du matériau comme vous pouvez le constater dans l'image suivante



Il est aussi possible de modifier sa métallicité comme dans l'image suivante:

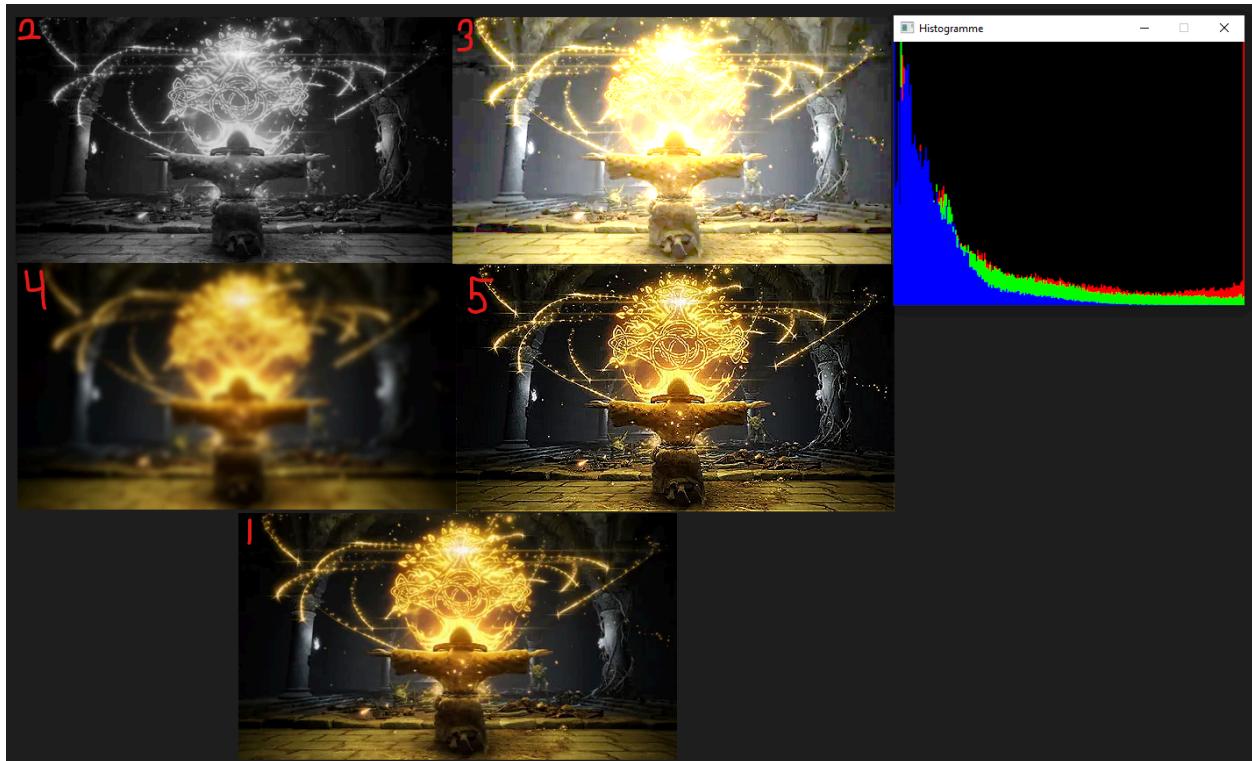


Nous pouvons de plus étirer la texture en x et en y comme suit:



## 6.2 Filtrage

Concernant le filtrage, nous avons choisi d'appliquer trois types de traitements aux images importées : la mise en niveaux de gris (*grayscale*), l'accentuation (*sharpening*) et le flou (*blur*), comme illustré ci-dessous:

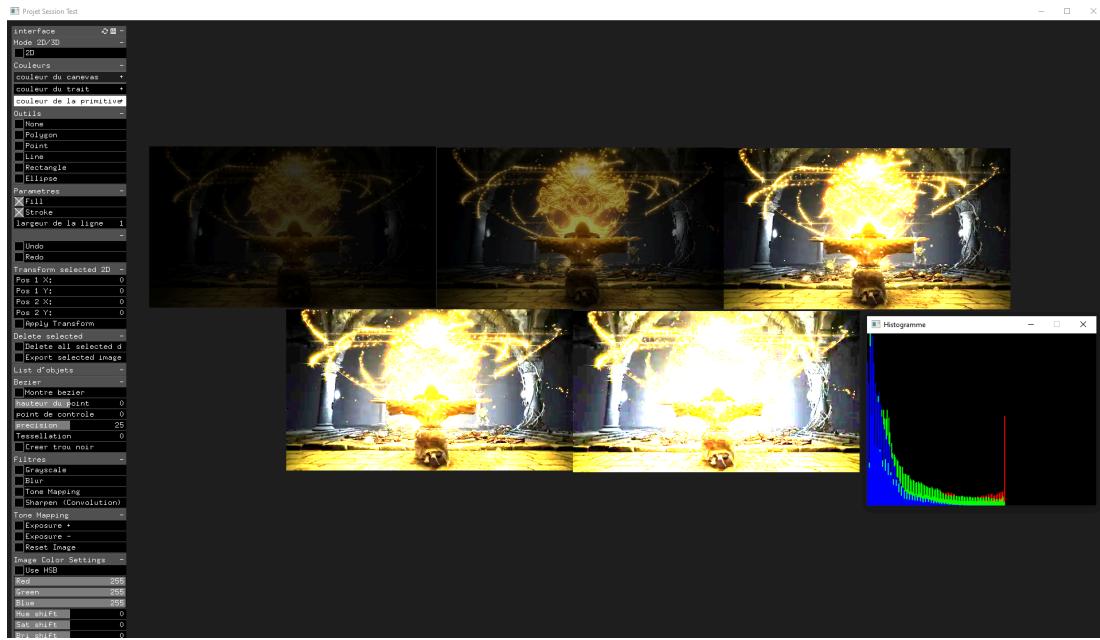


- Image 1: Image de référence
- Image 2: Ton de gris
- Image 4: Floue (blur).
- Image 5: Accentuation (sharpening)

Pour le blur et le sharpening, il est possible de l'augmenter ou le diminuer à l'aide des boutons dans le menu. De plus, un bouton a été mis pour simplement réinitialiser l'image à ses valeurs par défaut. Comme vous pouvez le constater l'histogramme se met à jour en fonction de l'image sélectionnée en incluant ses nouvelles modifications (sera plus facile à constater lors du visionnement de la vidéo).

## 6.3 Mappage Tonal

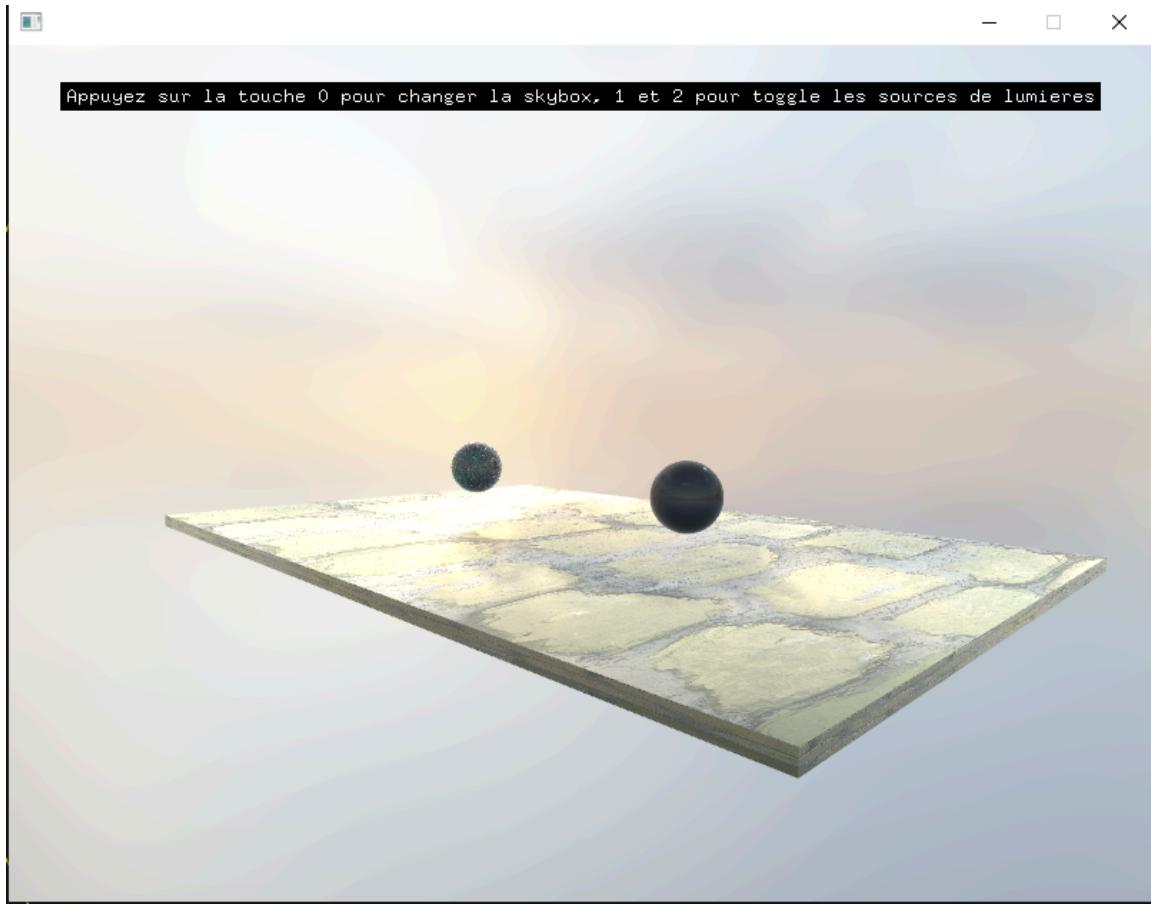
Toujours concernant nos images, nous avons aussi ajouté un algorithme de mappage tonal (tone mapping) comme vous pouvez le constater dans l'image suivante:



Les images ici sont en ordre croissant d'indicateur du mappage tonal. Il est facile dans le menu d'augmenter ou diminuer la valeur comme voulu. Tout comme précédemment, l'histogramme se met à jour pour refléter la dernière modification apportée.

## 6.4 Cubemap

Nous avons décidé d'implémenter le cubemap à la manière d'un skybox comme vous pouvez le voir dans l'image suivante:

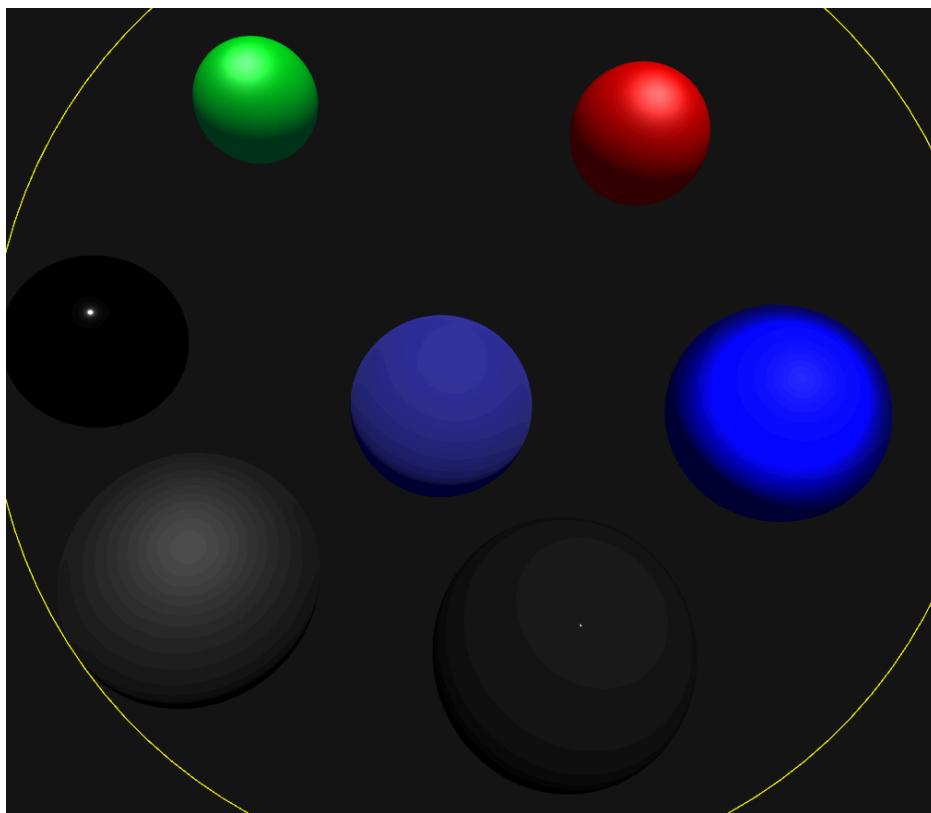


Cela nous permet de simuler un environnement pour notre scène en créant l'illusion d'un ciel lointain et en donnant un aspect de profondeur.

## Module 7: Illumination classique

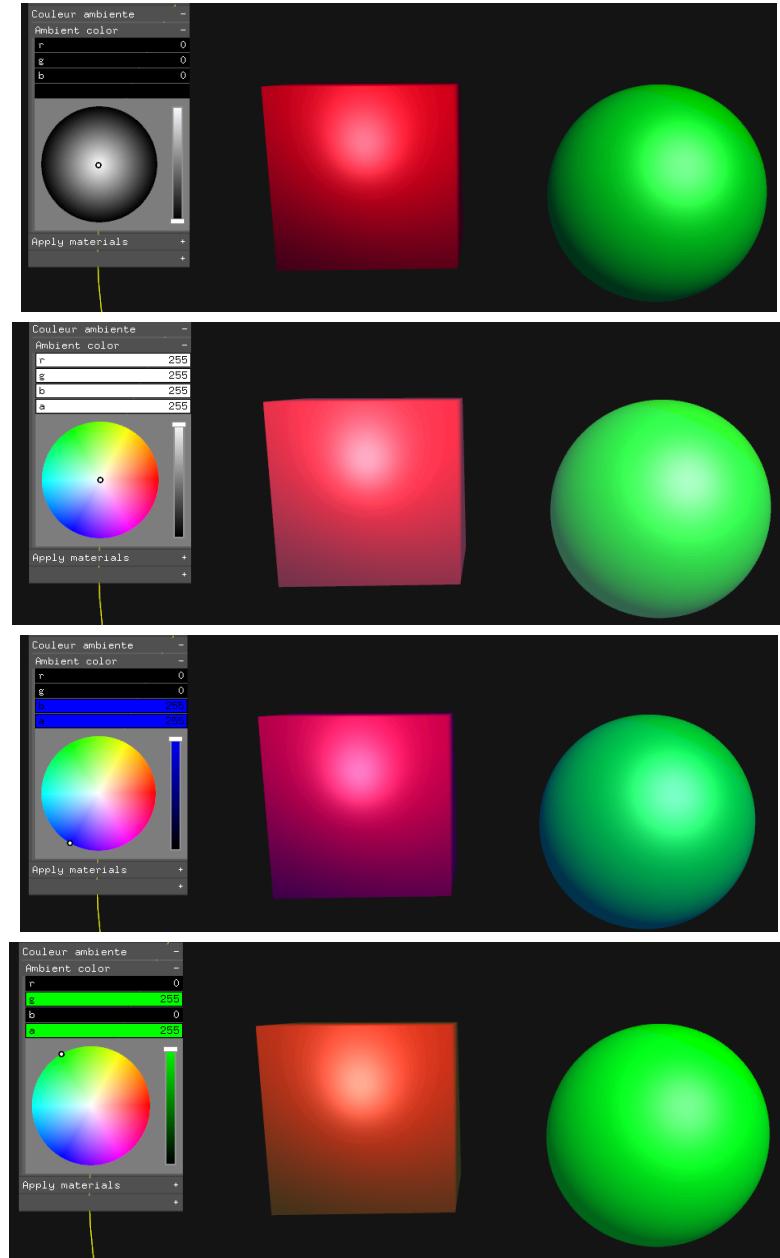
### 7.2 Matériaux

Pour les matériaux, nous avons des options dans l'interface nous permettant d'appliquer dynamiquement des textures aux objets de la scène. Nous avons 4 options, “default”, “material Sphere PBR 1” (le matériau qui est actuellement appliqué sur la Sphere PBR 1), “Material rough” puis “Material metal”. Lors de la sélection, un matériau similaire au matériau par défaut est utilisé avec une autre couleur, le rouge. Pour le matériau de la sphère, on peut modifier l'état courant du matériau puis l'appliquer à une sphère pour qu'elle conserve ce matériau jusqu'à la prochaine application de matériau. Dans l'image suivante, les matériaux noir et gris sont Sphere PBR 1 avec des états différents. Vert est le matériau par défaut et rouge le matériau de sélection. Le premier bleu est le matériau “Material Rough”, le deuxième est “Material Metal”.



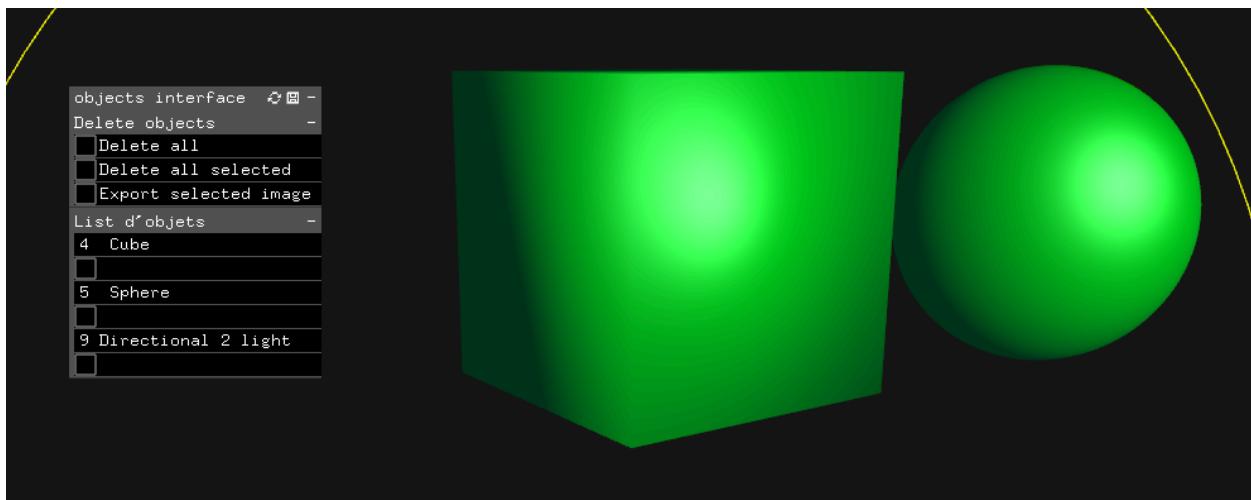
## 7.3 Types de lumière

Nous avons 4 types de lumière. Commençons par la lumière ambiante. Dans l'interface, un menu “Ambient color” nous permet de choisir la couleur et l'intensité de la lumière ambiante.

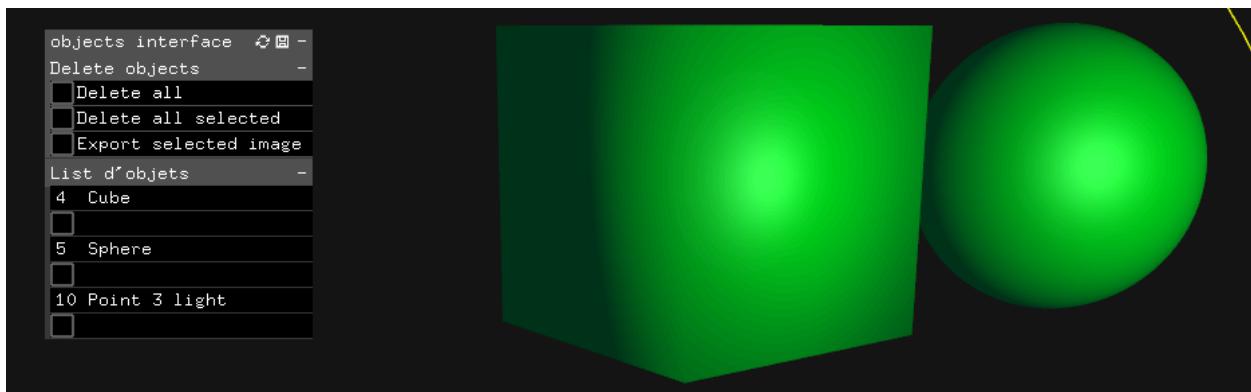


Pour les trois autres types de lumières, on sélectionne le type de lumière voulu dans l'interface, puis on appuie sur le bouton droit de la souris dans la scène pour placer une lumière. Tout dépendant du type de lumière, elle sera placée et/ou orientée de la même manière que la caméra. La position de la souris dans l'écran n'a aucun impact sur le placement de la lumière. Les trois types de lumière disponibles sont: directionnelle, ponctuelle et projecteur. Elles peuvent être sélectionnées et supprimées comme des objets, mais n'ont pas de changement d'apparence lors de la sélection.

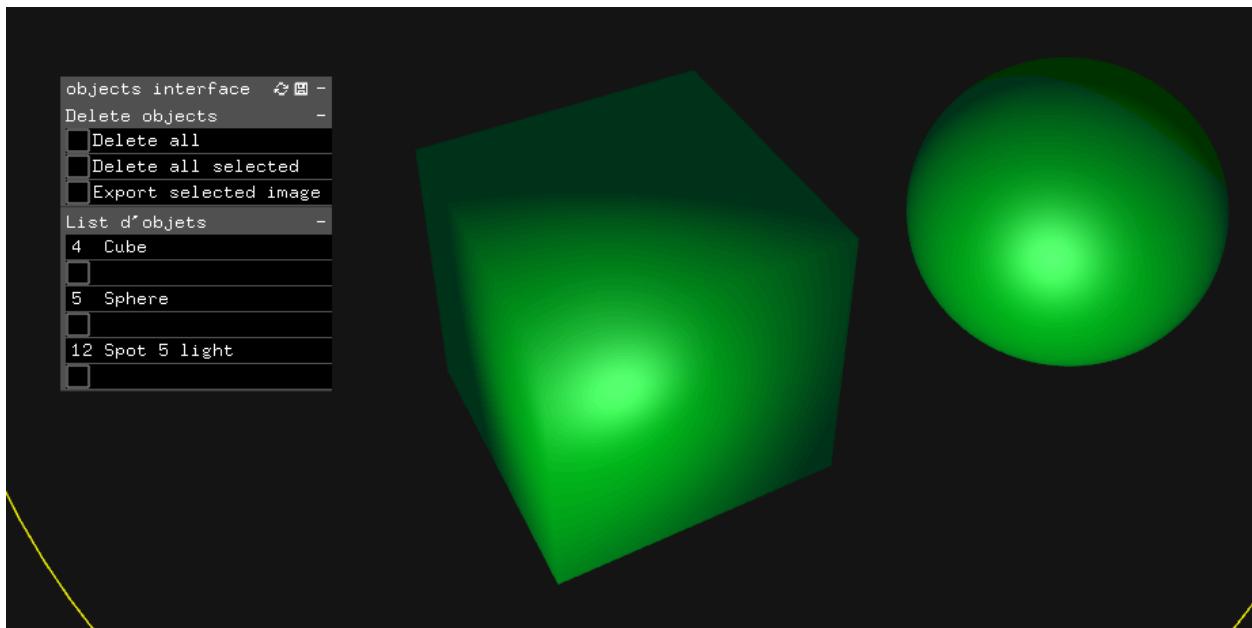
Directionnelle:



Ponctuelle

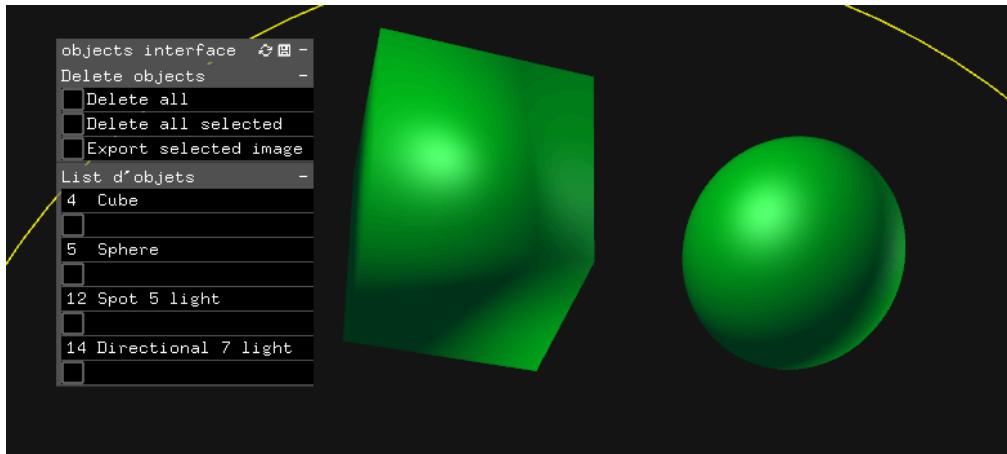


## Projecteur

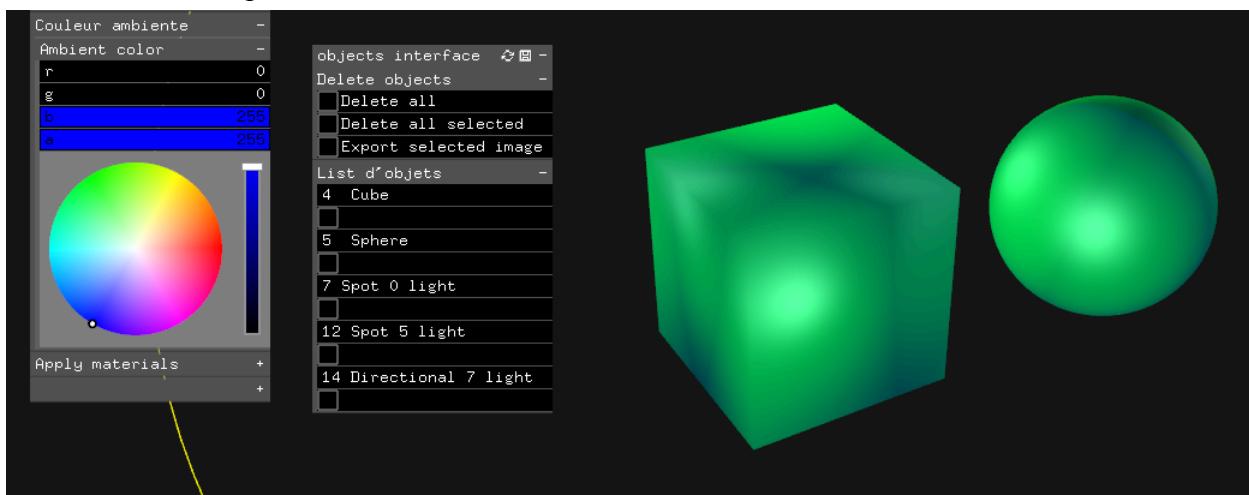


## 7.4 Lumières multiples

Comme mentionné dans 7.3, on peut sélectionner un type de lumière entre directionnelle, ponctuelle et projecteur. On peut ensuite placer jusqu'à 8 lumières dans la scène, peu importe leur type. Nous avons aussi une instance de lumière ambiante modifiable n'importe quand depuis l'interface. Voici un exemple de 2 lumières sur un objet:



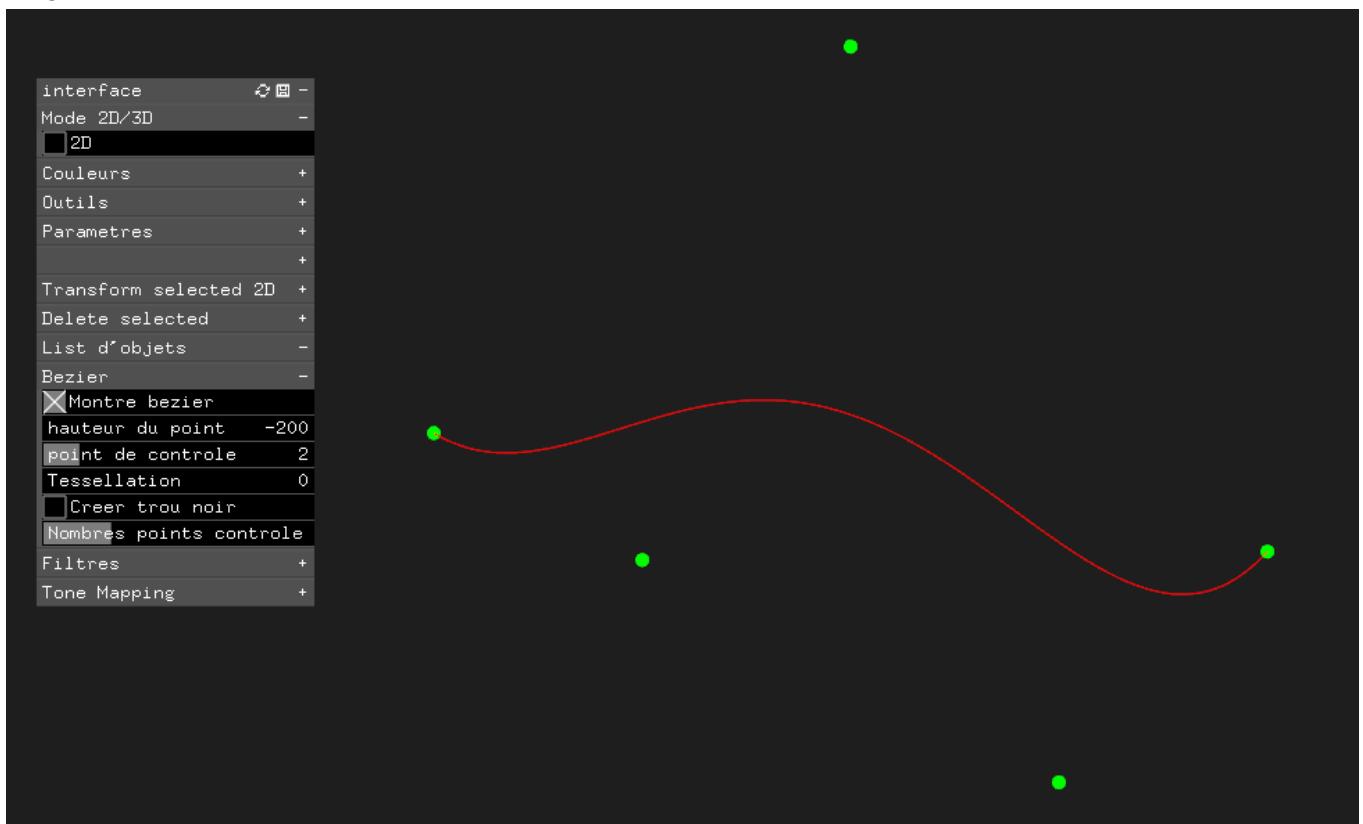
Puis un autre exemple, cette fois-ci avec 3 lumières et la lumière ambiante:



# Module 8:Topologie

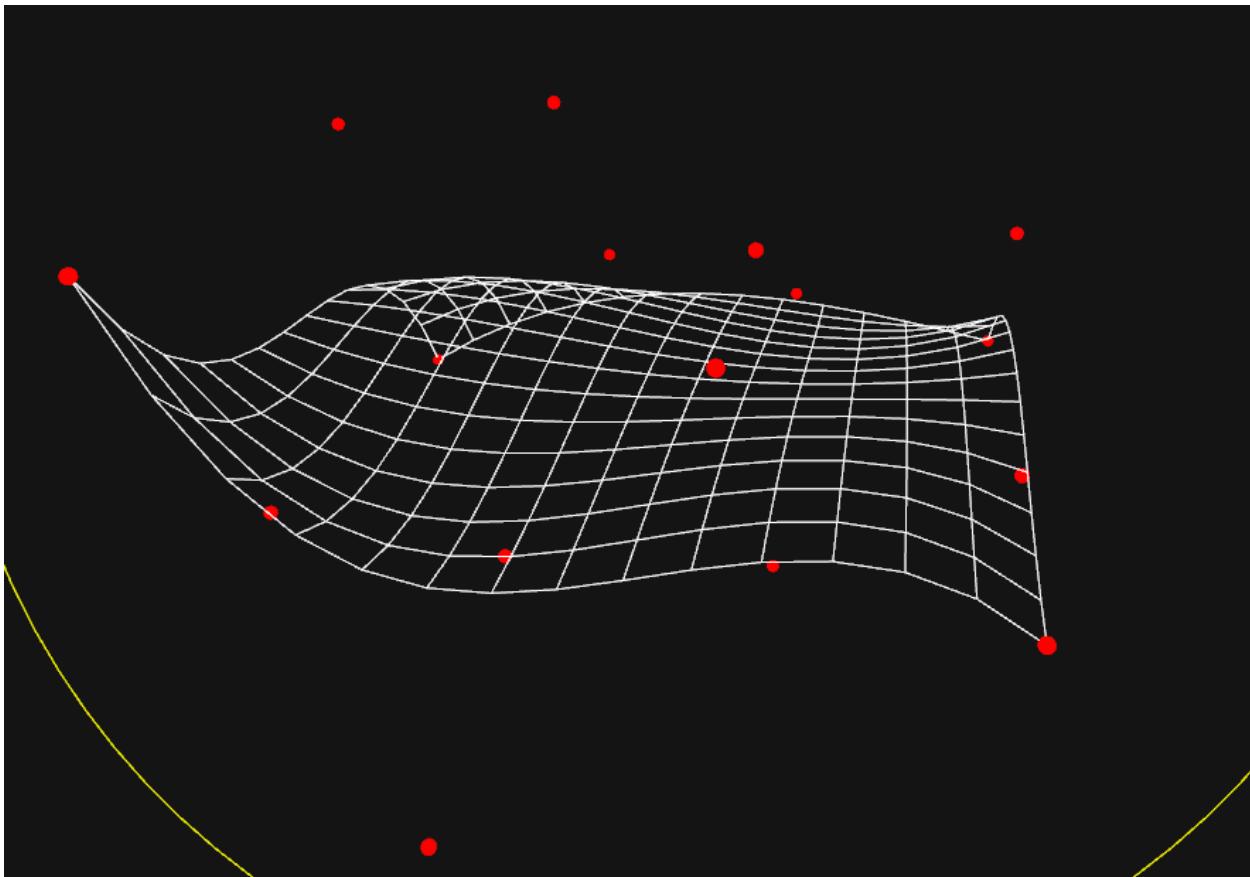
## 8.1 Courbes paramétriques

La scène en 2 dimension affiche une courbe paramétrique de bázier, avec des valeurs aléatoires choisies pour les points de contrôle pour donner une forme unique chaque fois que l'application démarre. La courbe à par défaut 4 points de contrôle (bázier cubique), mais il est possible d'en ajouter pour avoir jusqu'à 8 points de contrôle à l'aide de l'interface. On peut aussi sélectionner chaque point de contrôle manuellement avec l'interface et changer leur hauteur par rapport à la position d'origine, ce qui permet de changer l'apparence de la courbe paramétrique à celle voulue. Les points de contrôle sont illustrés par des cercles verts, et la courbe est dessinée en rouge.



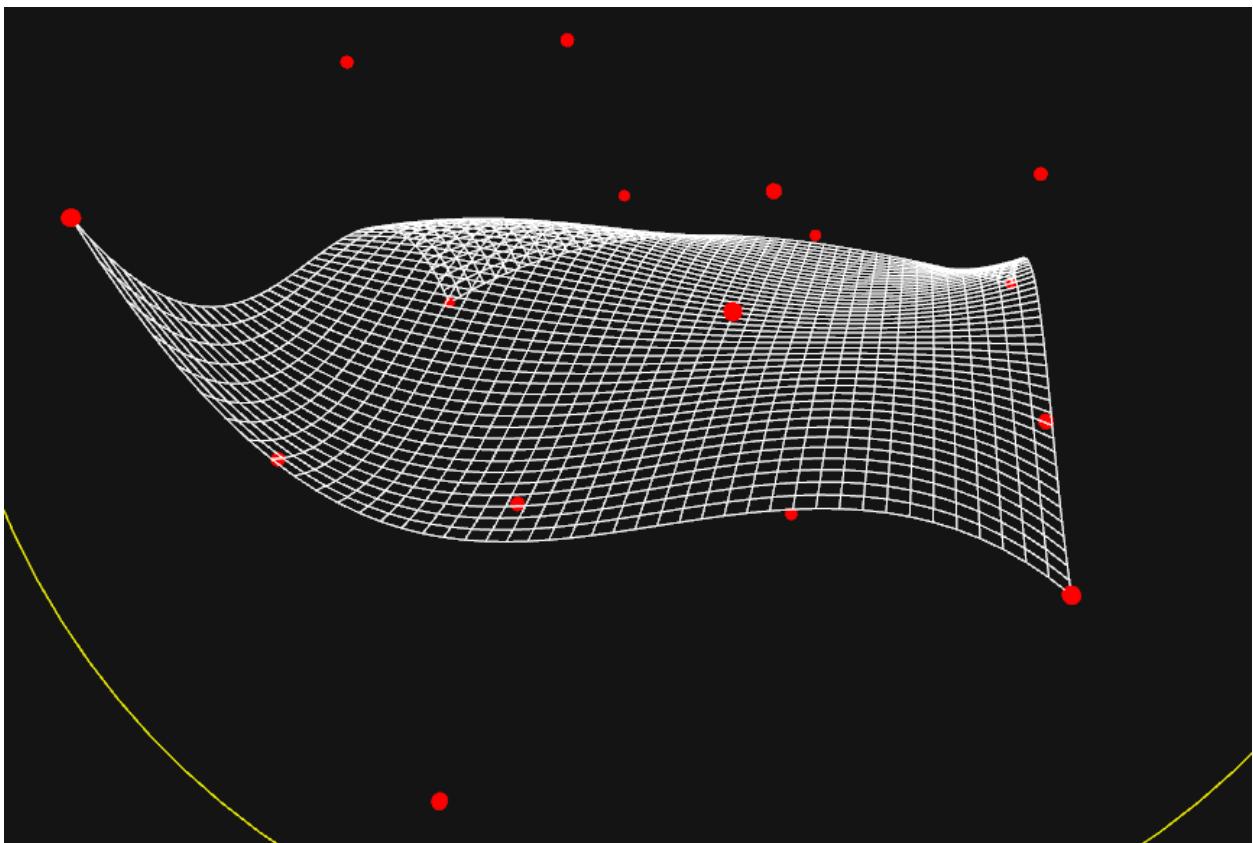
## 8.2 Surface paramétrique

Dans le mode 3D, on peut afficher une surface de Bézier, avec 16 points de contrôle. Chaque point de contrôle peut être sélectionné comme en 2D, et on peut changer sa hauteur par à sa position d'origine. La hauteur d'origine des points de contrôle est aussi calculée avec une valeur aléatoire pour permettre l'apparence d'une courbe directement lors de l'ouverture de l'application. On observe tous les points de contrôle comme étant des sphères rouges.



### 8.3 Shader de tessellation

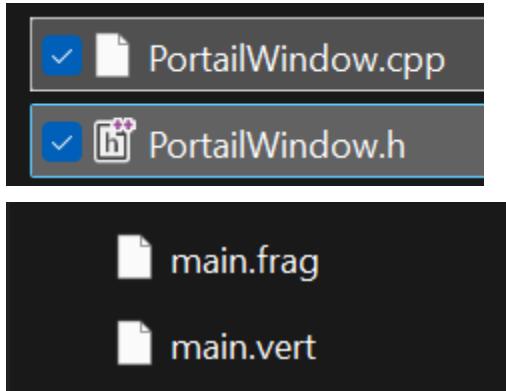
On peut changer le niveau de tessellation de la courbe de bézier montrée au point 8.2 à l'aide de l'interface. On peut choisir entre aucune tessellation , 1 niveau ou 2 niveaux de tessellation. Voici un exemple avec la même surface que dans la section 8.2, mais avec 2 niveaux de tessellation au lieu de 0 :



## Module 10: Illumination moderne

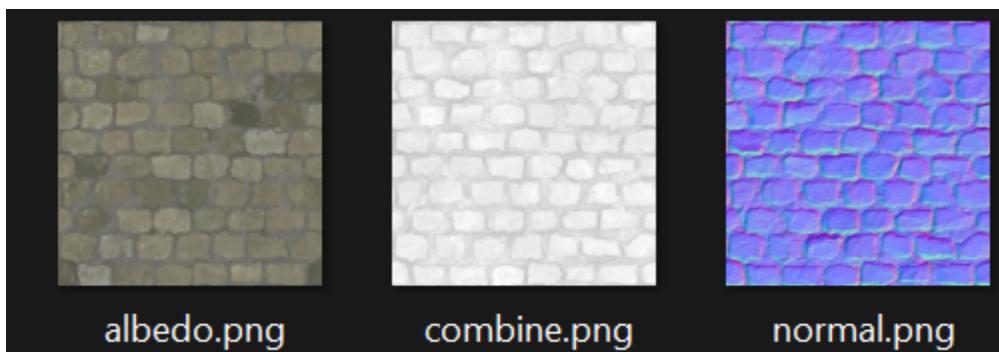
### 10.1 PBR

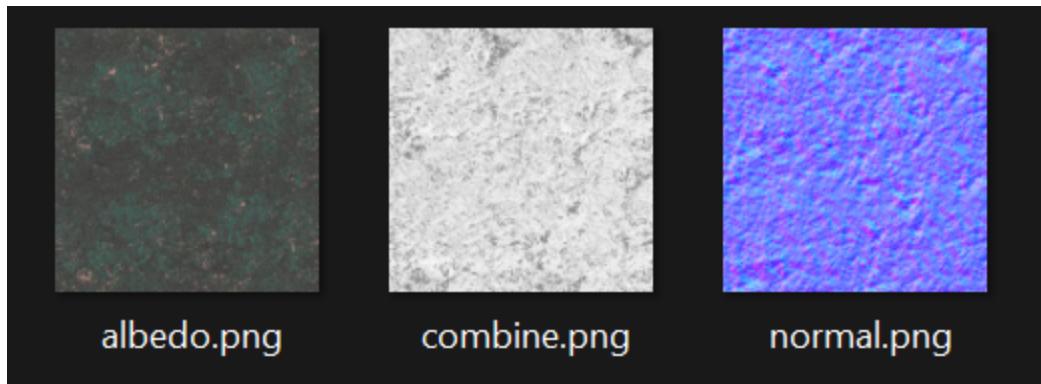
Pour cette partie du travail nous avons créé un shader inspiré des exemples d'OpenFramework directement. Ainsi nous avons pu implémenter le chargement du shader dans la fenêtre PortailWindow, modifiée pour la seconde partie du travail afin de ne pas impacter les autres composantes de notre projet.



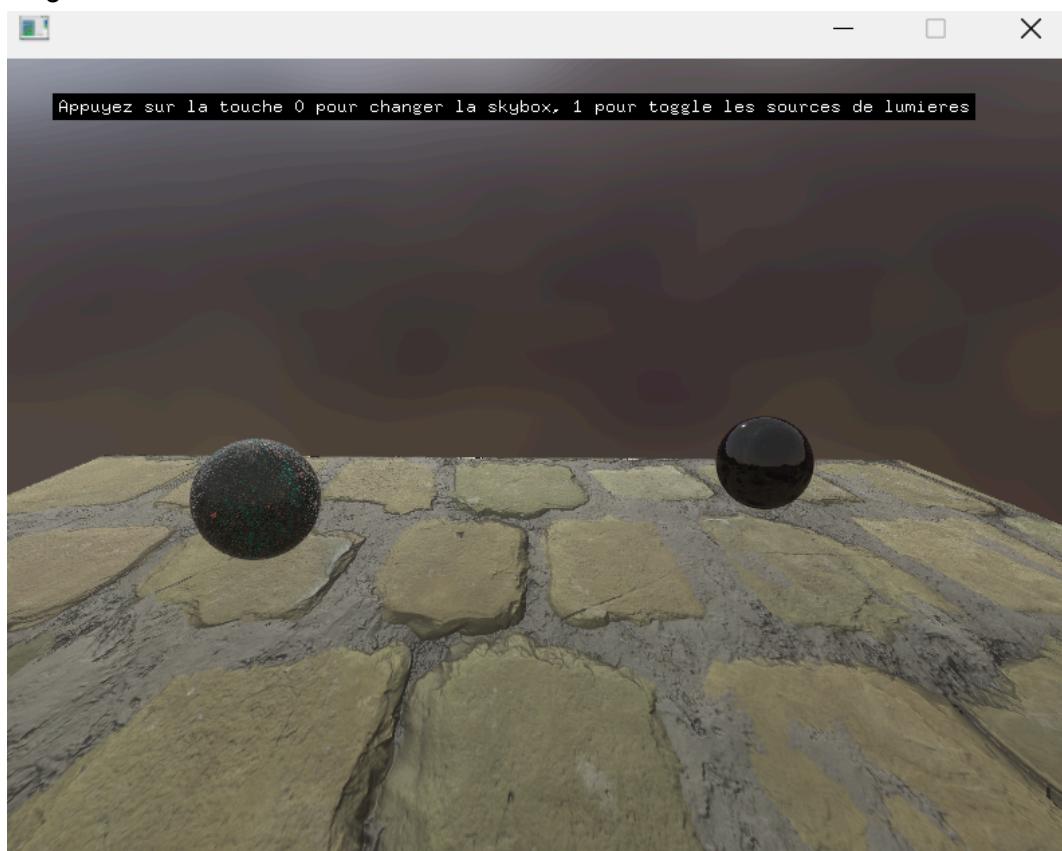
Le shader reste très basique, mais fonctionne comme suit: le vertex shader transmet la position locale pour piloter une animation, et le fragment shader utilise cette position en plus le facteur temps pour animer dynamiquement les paramètres PBR (métallicité., rugosité, albédo) sur le maillage.

Pour appliquer le PBR nous avons combiné le shader et les textures suivantes:



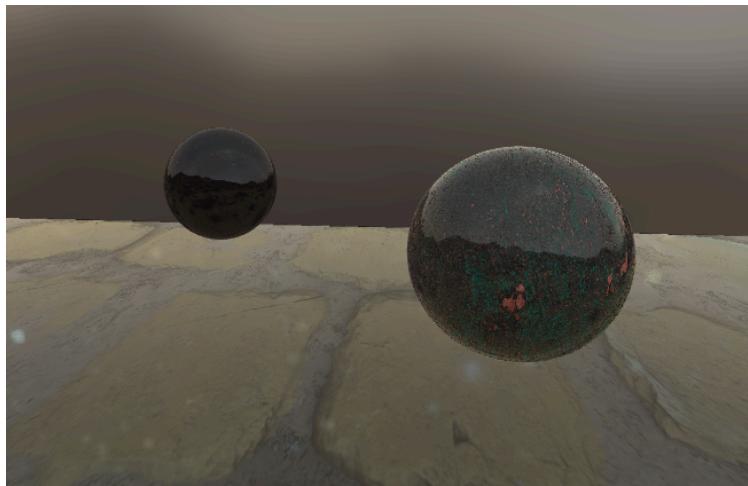


Combine.png est la combinaison de la texture de rugosité et de la texture métallique. Pour ce faire, nous avons utilisé une image RGB pour que chaque texture soit affichée sur sa couleur: vert pour rugosité et bleu pour la métallicité et nous aurions pu ajouter une autre texture sur le rouge au besoin.

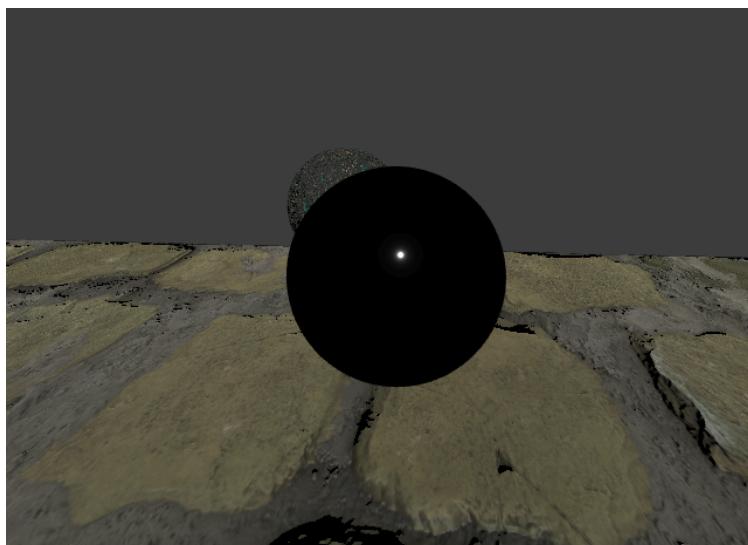


Plusieurs sources de lumière dynamique sont disponibles et peuvent être activées ou désactivées au besoin. Nous avons une lumière type projecteur, une lumière ambiante et la lumière de la skybox.

lumière ambiante désactivée:



Skybox désactivé:

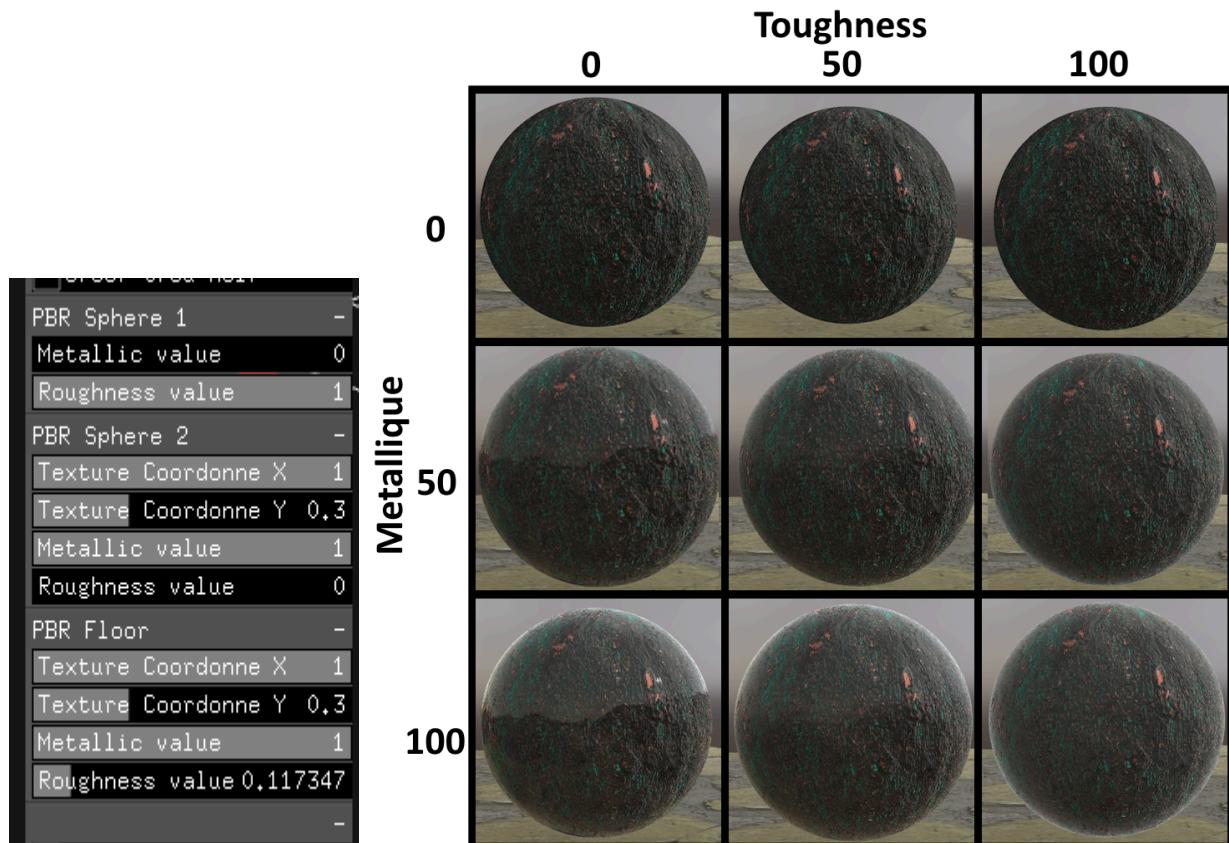


Lumière ambiante et skybox désactivé:



## 10.2 Métallicité et 10.3 Microfacettes

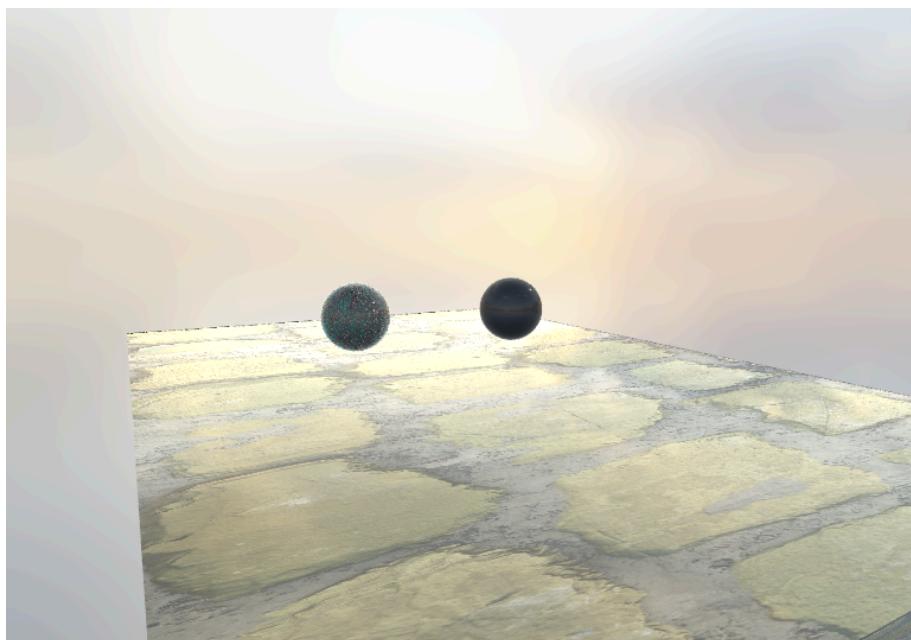
Pour la métallicité et les microfacettes nous avons ajouté un menu permettant de manipuler ces dernières et ainsi voir en temps réel les différences sur les objets.

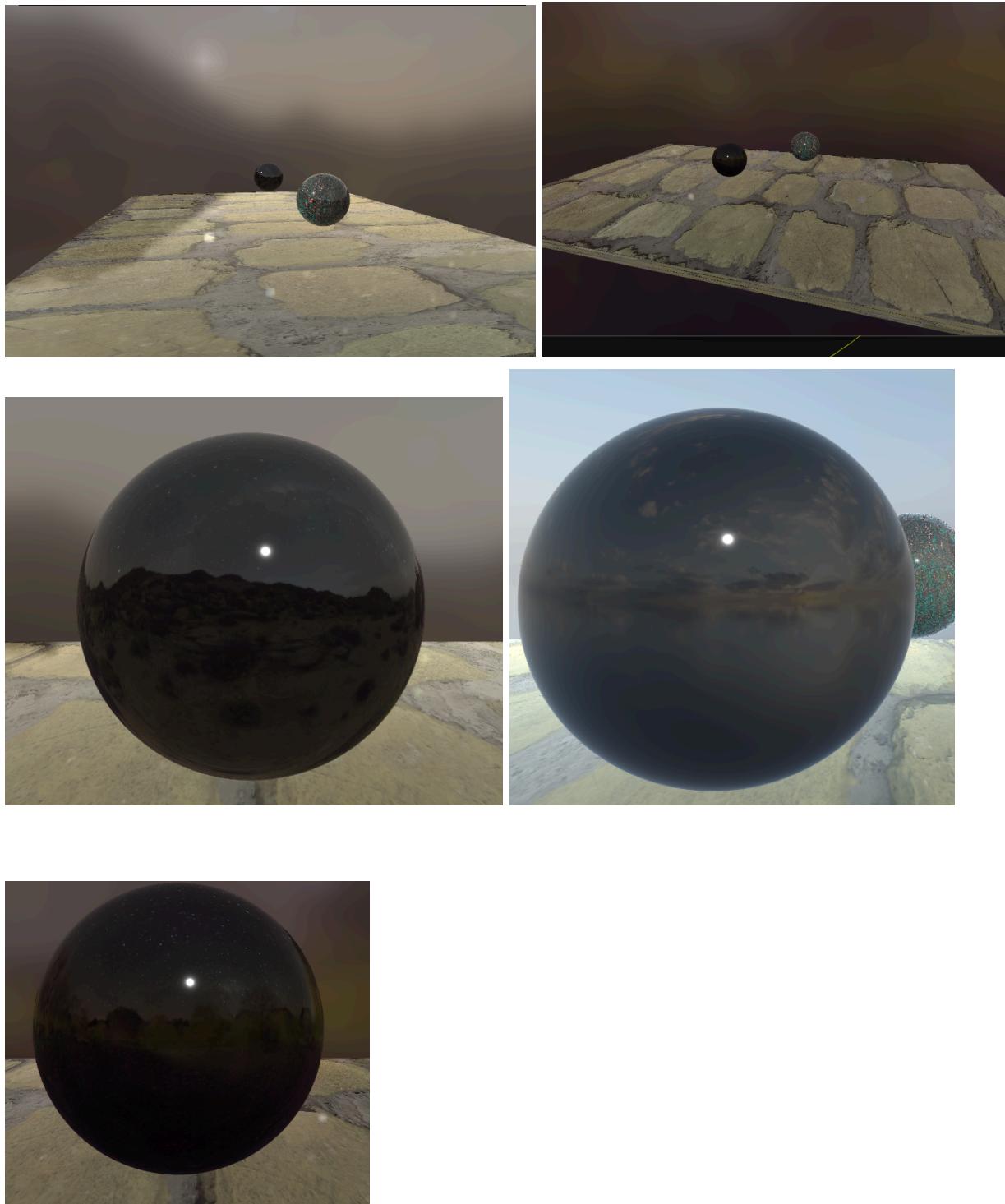


## 10.4 Éclairage environnemental

Pour cet étage nous avons fait en sorte de pouvoir changer entre 3 différentes skybox, qui modifient l'éclairage de la scène. Ceci est alors visible sur les surfaces métallique que la scène:

```
void PortailWindow::changeSky() {
    switch (cptSky) {
        case 0:
            cubeMap.load("kloppenheim_06_puresky_1k.exr", 512);
            cptSky++;
            break;
        case 1:
            cubeMap.load("rogland_clear_night_4k.exr", 512);
            cptSky++;
            break;
        case 2:
            cubeMap.load("satara_night_no_lamps_4k.exr", 512);
            cptSky = 0;
            break;
    }
}
```





# Ressources

Les images importées peuvent être n'importe quelles images, mais voici celles que nous avons utilisées pour le présent document: <https://alphacoders.com/elden-ring-wallpapers>.

Pour nos curseurs, nous avons utilisé le site suivant pour les dessiner par nous-même: <https://www.piskelapp.com/p/create/sprite>.

Pour la prise de vidéo, nous avons utilisés l'outil OBS (open broadcaster software) disponible ici: <https://obsproject.com/welcome>

## Modèles 3D

Gramophone: <https://www.turbosquid.com/3d-models/gramophone-character-vintage-2317851>

Cubone: <https://poly.pizza/m/cc7gCdKaQYU>

\*Dans les captures d'écran et la vidéo, une version grossi du modèle cubone a été utilisée pour mieux montrer les boîtes de délimitations.

Poisson: <https://poly.pizza/m/52s3JpUSjmX>

## Ressources pour la seconde partie du travail:

### Textures

Iron matériel : [Rock Infused with Copper PBR Material - Texture Download](#)

Floor: [Octogon Stone Cobble PBR Material - Texture Download](#)

### Skybox:

[Rogland Clear Night HDRI • Poly Haven](#)

[Satara Night \(No Lamps\) HDRI • Poly Haven](#)

[Kloppenheim 06 \(Pure Sky\) HDRI • Poly Haven](#)



# Présentation

Tous ont participé de manière équivalente au projet, ont respecté les échéances et ont participé aux réunions. Voici les membres de l'équipe:

- David D'Anjou
- Johny Savard
- Alexis Roberge
- Vincent Laverdure

Bien que tous ont participés à toutes les sections, que ce soit en conseils ou en rétro-action, voici un compte-rendu global des efforts principaux de chacun:

6.1	Coordonnées de texture	Johny
6.2	Filtrage	David
6.3	Mappage Tonal	David
6.4	Cubemap	Johny
7.2	Matériaux	Vincent
7.3	Types de lumières	Vincent
7.4	Lumières multiples	Vincent
8.1	Courbe paramétrique	Alexis
8.2	Surface paramétrique	Alexis
8.3	Shader de tessellation	Alexis
10.1	PBR	Johny
10.2	Métallicité	Johny
10.3	Microfacettes	Johny
10.4	Éclairage environnemental	Johny
	Document Design:	David
	Vidéo:	Tous
	Editing:	Johny



# Déclaration globale de l'usage de l'intelligence artificielle:

L'intelligence artificielle a été utilisée principalement comme support pour le débogage, notamment pour interpréter les codes d'erreur et proposer des solutions lorsque nous étions bloqués. Elle a également servi de ressource pour orienter l'implémentation de certaines fonctionnalités, comme mentionné dans la section correspondante.

L'intégration de Copilot à Visual Studio 2022 a permis d'accélérer certaines parties du développement en suggérant automatiquement du code, en particulier pour les structures répétitives et les tâches redondantes et des noms de variables. Toutefois, son utilisation est restée complémentaire aux outils classiques de l'IDE, sans remplacer notre travail d'analyse et d'implémentation des fonctionnalités spécifiques au projet.

## Lien vers la vidéo:

[https://youtu.be/6lcxqlJ\\_wwU](https://youtu.be/6lcxqlJ_wwU)

