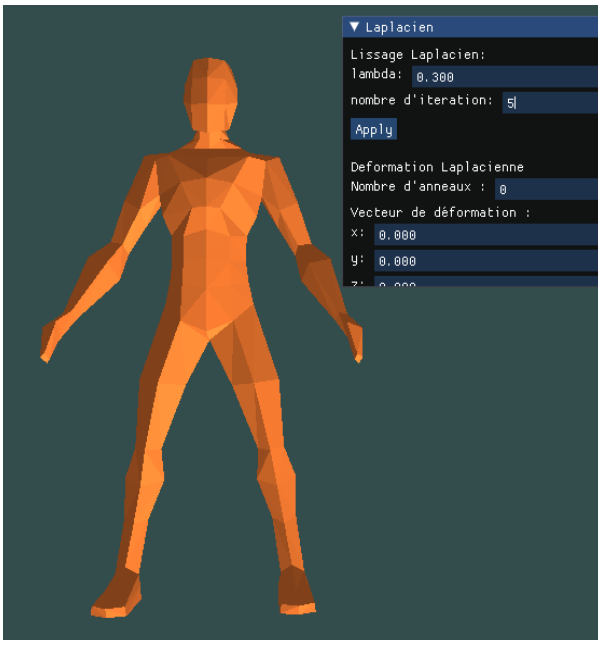
Rapport Projet
Informatique Graphique 3

I) D'où je suis partie et objectifs

Je suis parti d'un code permettant de charger et visualiser de maillage (.obj) avec Opengl fait au semestre précédent pour le projet de modélisation. Pour le projet de l'année dernière je n'étais pas parvenu à faire grand chose, c'était une des première fois que je codais en C, j'ai fait peu d'informatique en licence. Le code dont je suis parti peut charger un modèle et l'éclairage est fait dans un shader qui simule une lumière directionnel avec un éclairage de Phong. Il y a aussi une caméra mobile et dispose d'une petite interface faite avec Imgui/.



Mes objectifs sont:

- Pour la partie rendu:
  - rendre plusieur modèles simultanément
  - rendre plusieur lumière simultanément et d'avoir plusieur type de lumière
  - d'inclure des textures
  - de faire un éclairage micro facettes
  - faire des ombres avec du shadow mapping
- Pour la partie animation:
  - faire l'animation d'un cylindre avec la méthode de skinning comme sur le sujet d'animation présenté sur moodle

II) Travail réalisé pour la partie rendu

a) Plusieur modèles et plusieur lumières

Pour pouvoir rendre plusieurs modèles, j'ai créé une structure "Rendable" qui contient le modèle à rendre, une matrice glm::mat4 toWorld contenant la projection du modèle dans l'espace scène et le shader qui devra être utilisé pour rendre la modèle. La classe de rendu contient donc un vecteur de "Rendable" sur lequel on 'itère dans ma boucle de rendu. Pour chaque "Rendable", on active le shader associé, on définit les uniformes et ensuite on lance le dessin.

```
struct Rendable{
   Model model;
   glm::mat4 toWorld;
   Shader shader;
};
```

Pour ce qui est des types de lumières, les points lights et directional light ont été ajoutés. Elles ont une structure commune avec union du côté C++ et chacune leur propre du côté shader.

```
struct Light{
   Model model = Model("../scenes/cube/cube.obj", false);
   Shader shader = Shader("../shader/light/light.vs",
"../shader/light/light.fs");
   LightType type;
   union{
       glm::mat4 toWorld;
       glm::vec3 direction;
   };
};
```

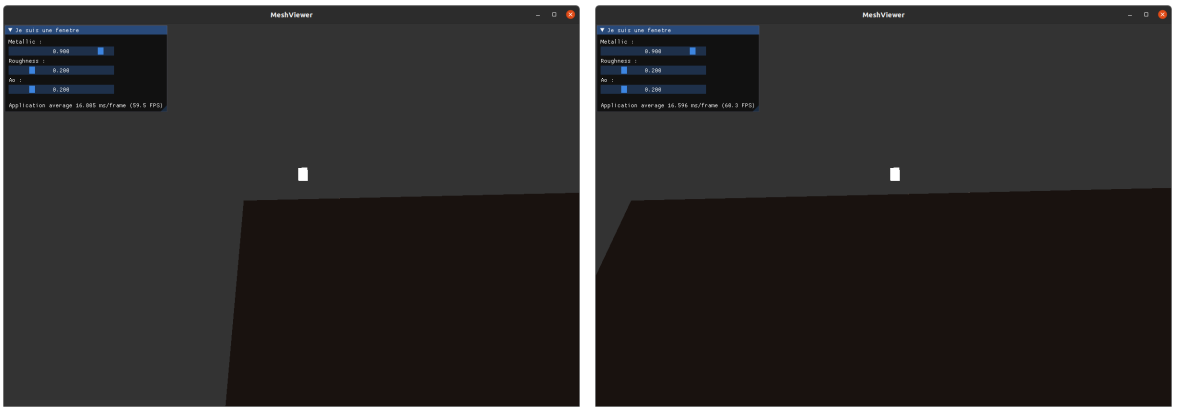
Coté C

```
struct PointLight {
   vec3 position;
   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
   float constant;
   float linear;
   float quadratic;
};
```

```
struct DirLight {
  vec3 direction;
  vec3 ambient;
  vec3 diffuse;
  vec3 specular;
};
```

Coté shader

Les lumières directionnelles suivent la caméra pour maintenir leur direction d'éclairage.



Lumière directionnel

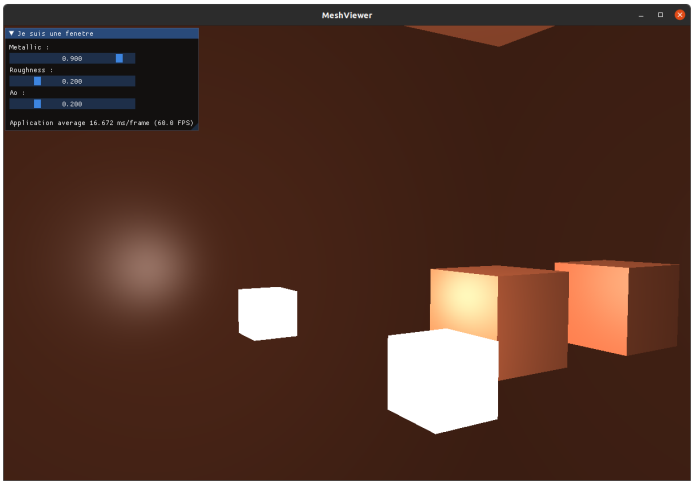
Il manque plus que d'avoir plusieurs lumières actives en même temps. Pour cela on crée un vecteur de "Light" pour stocker toutes les lumières (point light et directional light) de la scène. Pour rendre ça il va nous falloir passer un tableau de uniforme contenant les informations de toutes les lumières au shader pour ensuite itérer sur toutes ces lumières et additionner leurs contributions.

```
#define NB_POINT_LIGHTS 128
uniform PointLight pointLights[NB_POINT_LIGHTS];
uniform int nbPointLight;

#define NB_DIR_LIGHTS 128
uniform DirLight directionLights[NB_DIR_LIGHTS];
uniform int nbDirLight;
```

Coté shader

```
currentModel->shader.setVec3("pointLights[" +
std::to_string(nbPointLight) + "].ambient", 0.2f, 0.2f, 0.2f);
```

Coté C en bouclant sur les lumières de la scène



Plusieur modèle et point lights

b) Inclusion des textures

Jusqu'à présent les caractéristique des surface été caractérisé par les matériaux contenant un triplé RGB pour les composantes ambient, diffuse et spéculaire. Pour ajouter des textures il nous faut les charger, le changement des .obj avec assimp fait déjà le chargement des textures. On y trouve les textures spéculaires et les textures diffuses. Les textures diffuses serviront à la fois pour la composante diffuse et ambiante. Il nous faut ensuite activer une texture et puis on y bind notre texture à notre shader pour pouvoir s'en servir. Du côté du shader on remplace simplement les triplés des matériaux par les valeurs de textures.

```
glActiveTexture(GL_TEXTURE0 + i);
//.. ..//
glBindTexture(GL_TEXTURE_2D, textures[i].id);
```

Coté C++

```
in vec2 TexCoords;
uniform sampler2D texture_diffuse1;
uniform sampler2D texture_specular1;
texture(texture_diffuse1, TexCoords);
texture(texture_specular1, TexCoords);
```

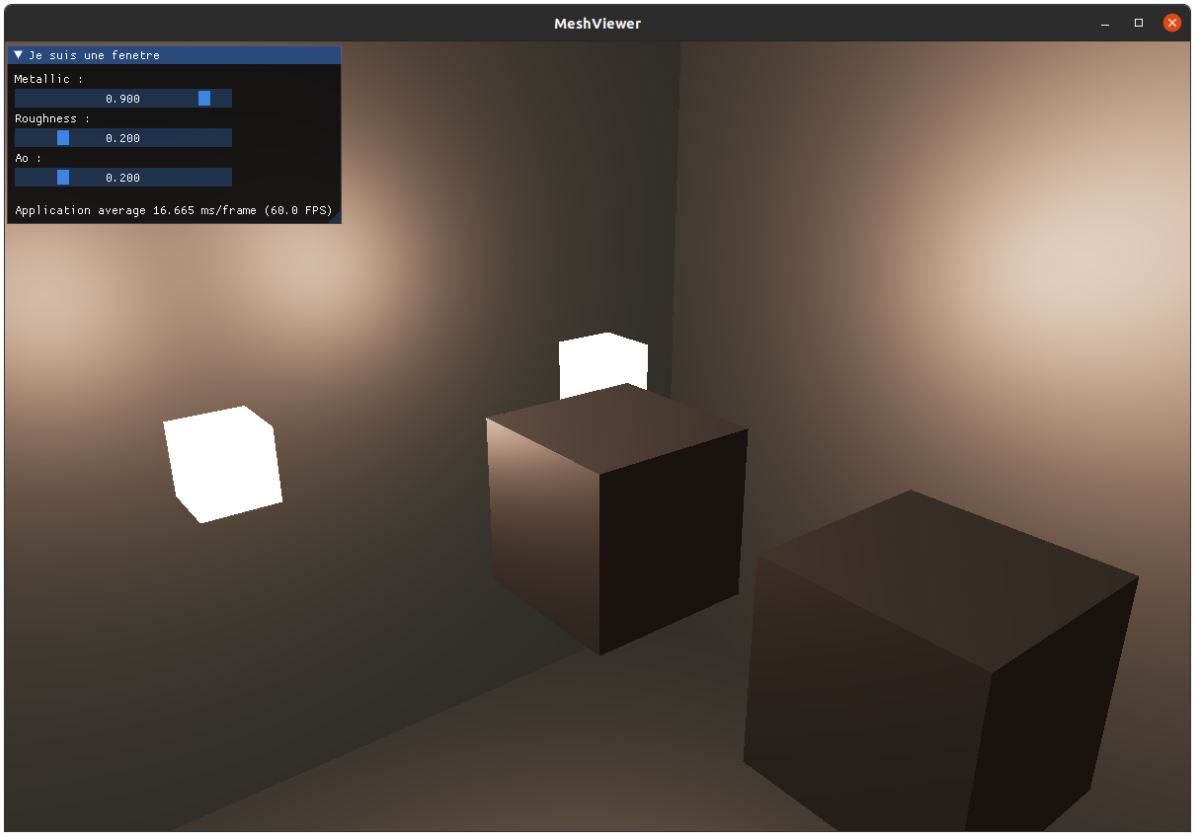
Coté shader

Eclairage de Phong avec textures et multi light

c) Eclairage micro facettes

Pour faire l'éclairage microfacette j'ai suivi le tutoriel de learn opengl. Pour réaliser cet éclairage, on utilise une BRDF de CookTorrance possédant un terme de distribution de normale, de géométrie et de Fresnel. Le premier rend compte de la rugosité de la surface au travers de l'alignement des normales des micro-facettes avec le vecteur de "halfway". Le second décrit l'ombre que font les micro-facettes les unes sur les autres. Et enfin le terme de Fresnel décrit les proportions de lumière réfractée et diffractée.

Nous avons trois paramètres pour influencer le modèle, le premier est l'albédo qui décrit la couleur de la surface. Le second est la métallicité qui caractérise à quel point la matière concentre la lumière dans ces reflets. Et enfin la rugosité dont on a parlé plus haut.



Eclairage par micro facette

d) Les ombres

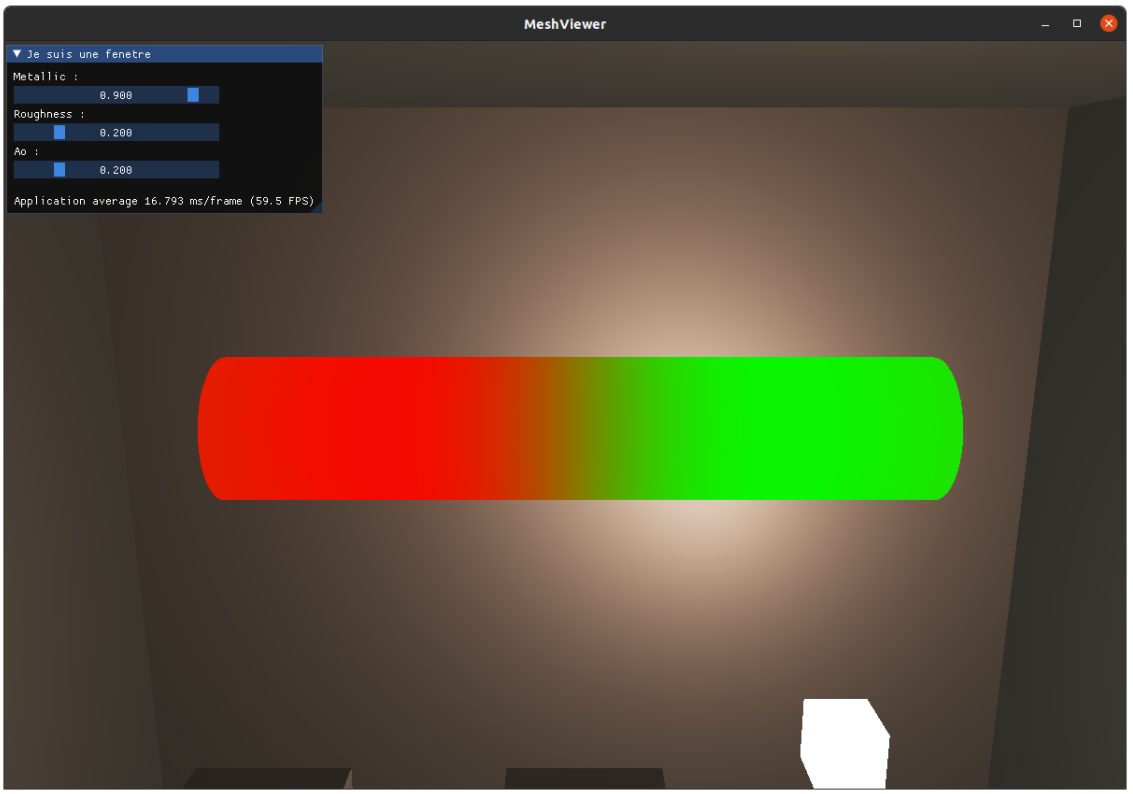
Pour les ombres l’objectif était d’utiliser de shadow map. La principe est de faire une passe rendu en se plaçant à la position de la lumière et dans la direction où elle éclaire pour récupérer la carte de profondeur et la placer dans une texture : une shadow map. Cette carte contient la profondeur et donc la position des points de la scène éclairée par la lumière. De cette manière en utilisant une matrice de changement de base entre la vue caméra et celle de la shadow map on peut comparer la profondeur inscrite dans shadow map et celle du pixel rendu pour voir si il est éclairé ou non.

J’ai essayé de l'implémenter mais je réussi à faire un passe de rendu dans un autre FBO mais je n’ai pas réussi à écrire le résultat du depth buffer dans une texture.

III) Travail réalisé pour la partie animation

Ici l'objectif est d'appliquer une méthode de skinning pour animer un cylindre possédant deux os. Cela nous permet d'animer un squelette et de transmettre son animation au maillage. A noter que j'ai fait cet algorithme avec les explications du cours et le reste par moi même donc désolé d'avance si la méthode employée n'est pas vraiment la bonne.

La première étape est de calculer l'influence de chaque os sur chaque point du maillage et l'os ce qui nous donnera l'influence de l'os qui sera l'inverse de la distance entre le point et le milieu de l'os. Pour finir on normalise cette influence par la somme de toute les influences sur ce point.



.

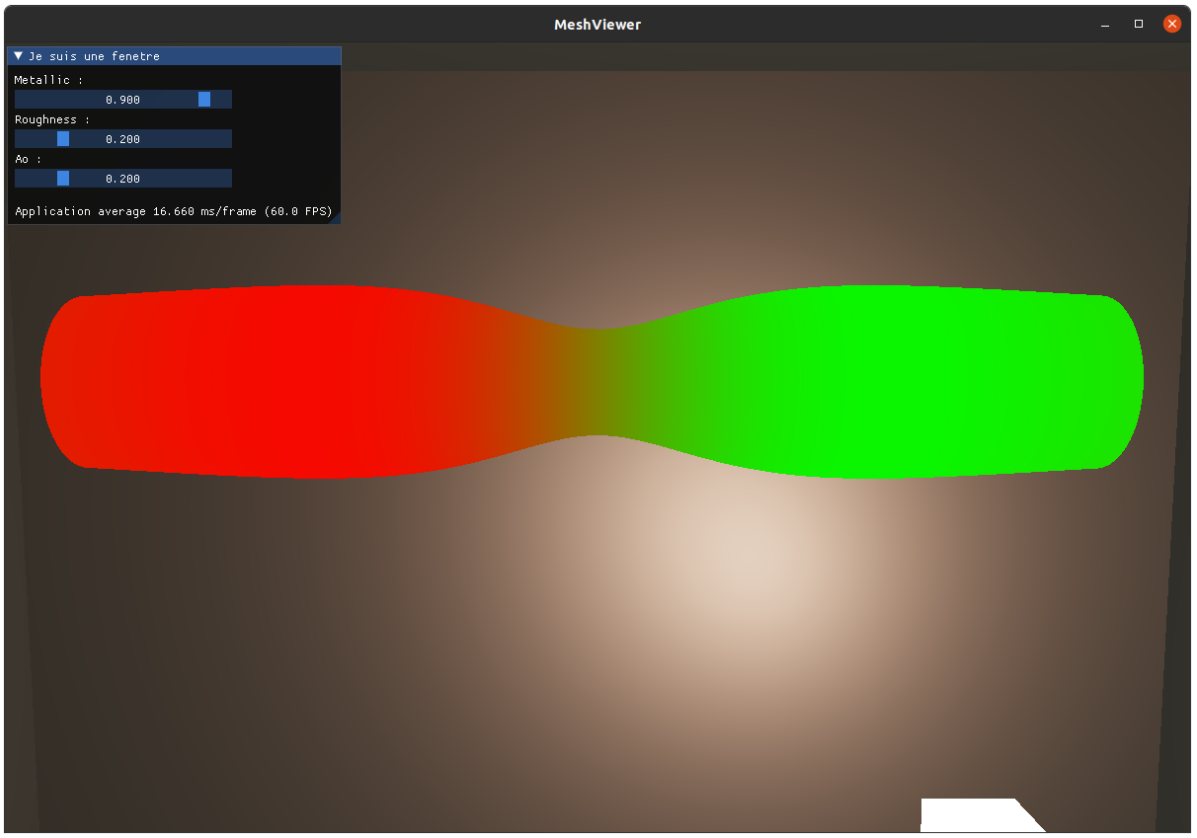
En vert l'influence d'os de gauche et en rouge l'influence de l'os de droite

Premièrement il nous faut définir une base pour l'os pour cela on prend le vecteur de l'os, un autre vecteur avec lequel le produit scalaire est nul et enfin le produit vectoriel des deux. Pour finir on normalise le tout.

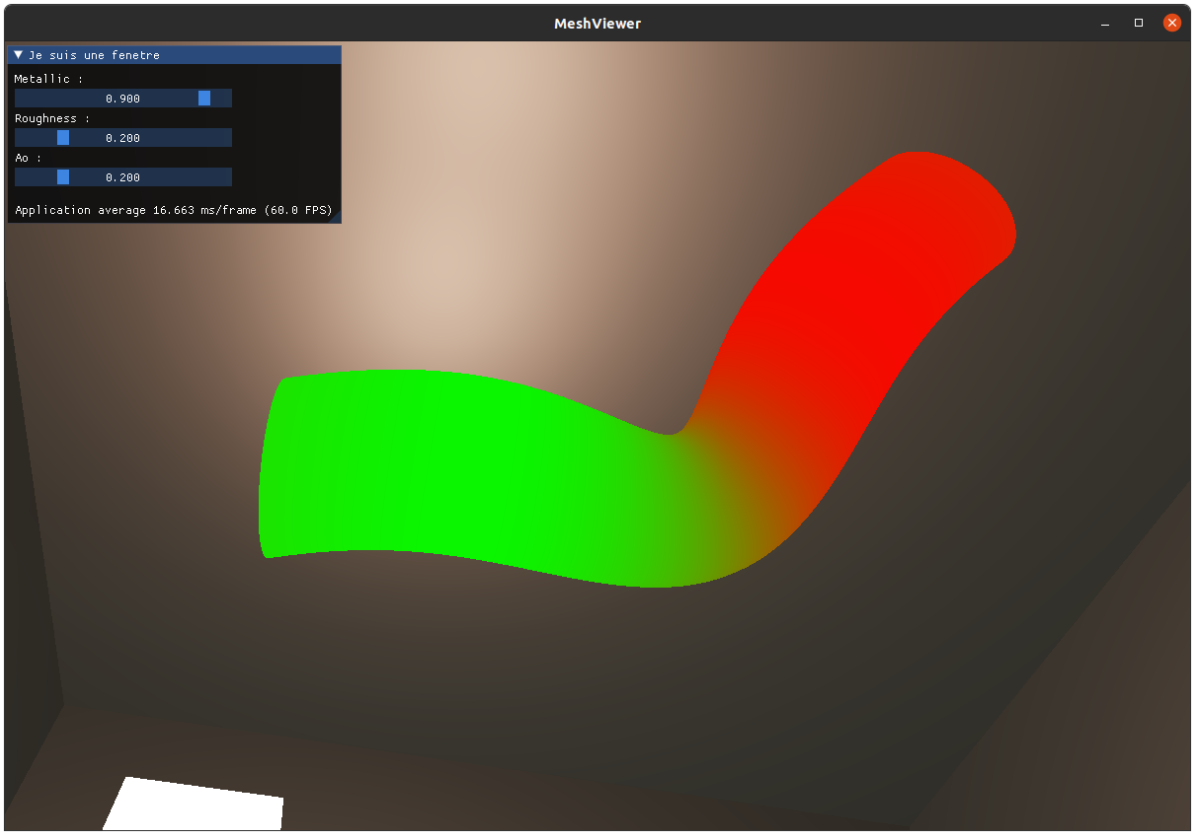
Ensuite il nous faut de matrice de transition entre la base world et os pour pourvoir faire suivre l'animation au point dans la base de l'os et ensuite le repasser dans la base world pour l'afficher. Cette matrice de transition et une matrice de passage entre nos deux bases combinée à une translation de la position de la base de l'os.

```
    glm::vec3 transpose = base.xyz();
    glm::vec4 ax1 = glm::normalize(extremite - base);
    glm::vec4 ax2 = glm::normalize(createColineaire(ax1));
glm::vec4 ax3 = glm::normalize(glm::vec4(glm::cross(ax1.xyz(),
                ax2.xyz()), 0));

    glm::vec4 fill = glm::vec4(transpose,1);
    PBtoW = glm::mat4(ax1, ax2, ax3, fill);

    PWtoB = glm::inverse(PBtoW);
```

On peut donc maintenant bouger les os, passer un point du maillage dans la base de chaque os pour obtenir les positions induites par chaque os. Enfin on somme ces positions en les pondérant par les influences décrites plus haut pour obtenir la position finale des points.



Effet papier de bonbon



Pli de l’articulation

IV) Code

Le code pour la partie animation est les fichiers src/bones.cpp et src/animation.cpp.
Le code pour la partie rendu est dans les fichiers src/renderer.cpp
Les shaders sont dans le dossier shader.
Les configurations de lancement peuvent être modifiées dans le fichier main.cpp entre les lignes 85 et 95.
Le code est disponible sur github: https://github.com/allamika/meshViewer
Le cmake fonctionne avec vcpkg pour :

- glfw
- glad
- imgui
- GLEW

Pour assimp il faut clone le repo dans le dossier meshviewer