Compte rendu TP8-9

Aloys TETENOIRE - 22204713

23 décembre 2022

1 Partie 1

Pour la première partie de ce TP, j'ai commencé par implémenter des shaders suivant le modèle Phong, afin de pouvoir appliquer la bump map par la suite.

1.1 Phong

J'ai réussi à avoir un rendu qui me semble correct (figure 1) en reprenant les shaders présentés dans le cours et en les adaptant à mon code. Pour les paramètres des matériaux utilisant Phong, j'ai utilisé les valeurs par défaut.

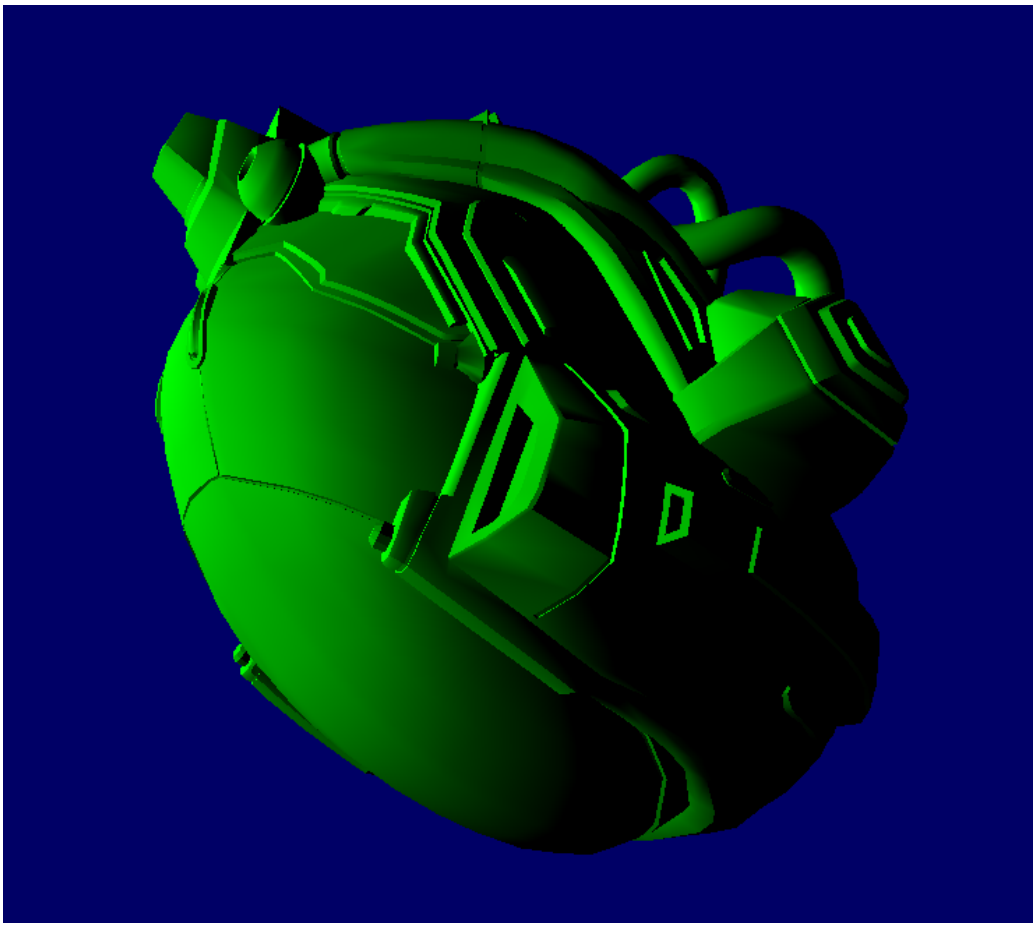

Figure 1 – Rendu Phong

1.2 Bump map

Pour le bump mapping, j'ai essayé d'appliquer la formule du cours afin de calculer de nouvelles normales. Pour cela, j'ai trouvé un moyen de récupérer les texels adjacents de la

bump map. Cependant, je n'arrive quand même pas à un résultat satisfaisant (figure 2). Le code faisant ces calculs est dans le fragment shader Phong, et la ligne utilisant la nouvelle normale est commentée.

Le calcul des texels adjacents se fait de cette manière :

```
ivec2 texSize  = textureSize(bumpTexture, 0);
vec2 texOffset = 1.0 / vec2(texSize.xy);
float color_right = texture(bumpTexture, UV.st + texOffset * vec2(1.0, 0.0))
    .r;
float color_left = texture(bumpTexture, UV.st + texOffset * vec2(-1.0, 0.0))
    .r;
float color_up = texture(bumpTexture, UV.st + texOffset * vec2(0.0, 1.0)).r;
float color_down = texture(bumpTexture, UV.st + texOffset * vec2(0.0, -1.0))
    .r;
```

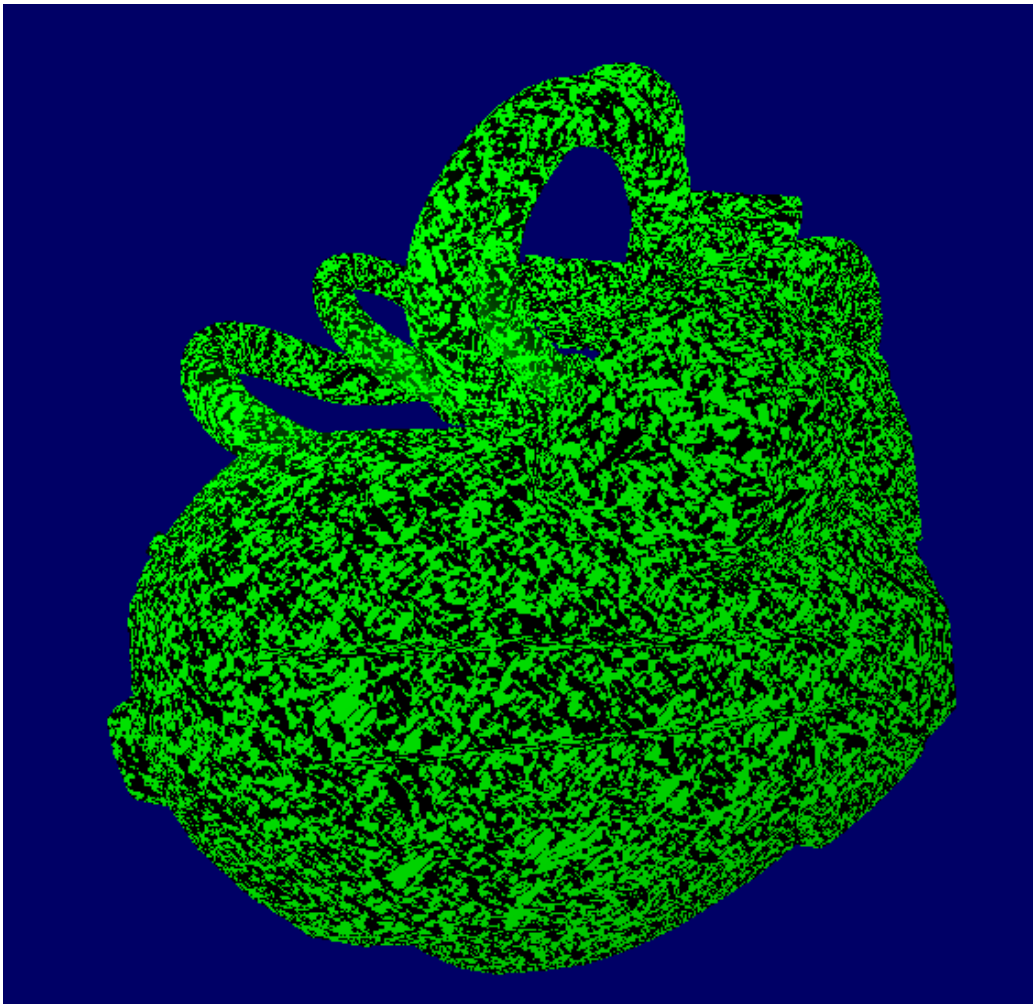

Figure 2 – Rendu avec bump mapping

1.3 Skybox

Pour la skybox, j'ai créé un cube autour de la scène. Pour éviter que cette skybox ne se déplace avec la caméra, j'ai retiré la partie translation de sa view matrix. On obtient donc le résultat de la figure 3.

Figure 3 – Rendu de la skybox

1.4 Environment mapping

J'ai créé un système permettant de changer le type de shaders utilisé dans le contexte. On peut facilement changer entre les shaders Phong, PBR, et Reflective. C'est ce dernier type de shaders qui permet le rendu réflectif. Pour changer le type de shaders, il suffit de changer la ligne 214 dans la fonction *main*().

Le rendu réflectif est parfaitement fonctionnel, comme on le voit dans la figure 4.

Figure 4 – Rendu réflectif

2 Partie 2

Pour le rendu PBR, j'ai tout d'abord commencé par utiliser *assimp* pour importer les matériaux provenant des modèles 3D. J'ai réussi à transférer correctement les paramètres PBR de ces matériaux aux shaders.

```
for (int i=0; i<material->mNumProperties; ++i)
    {
        if (material->mProperties[i]->mKey == aiString("$clr.base"))
        {
            auto * albedo = (float*)material->mProperties[i]->mData;
            mat->m_pbr.albedo = glm::make_vec3(albedo);
        }
        else if (material->mProperties[i]->mKey == aiString("$mat.
    metallicFactor"))
        {
            mat->m_pbr.metalness = *(float*)material->mProperties[i]->mData;
        }
        else if (material->mProperties[i]->mKey == aiString("$mat.
    roughnessFactor"))
        {
            mat->m_pbr.roughness = *(float*)material->mProperties[i]->mData;
        }
    }
```

Cependant, je n'ai pas réussi à appliquer ces paramètres pour le rendu PBR.

3 Organisation

Pour ce TP, j'ai commencé par essayer d'appliquer une bump map sur un objet. Cependant, comme le rendu unlit de base ne permet pas de faire de bump mapping, je n'ai pas pu tester les bump maps directement.

J'ai ensuite écrit la partie qui gère l'import des paramètres PBR dans les matériaux, afin de pouvoir tester le bump maps avec le rendu PBR. Cependant, j'ai eu des difficultés avec le PBR. J'ai donc implémenté le modèle de Phong. Enfin, j'ai ajouté la skybox, et le rendu réflectif.

Conclusion

Pendant ce TP, j'ai appris les bases du rendu PBR. Aussi, j'ai appris à implémenter le modèle de Phong ; je l'avais déjà fait pendant un autre projet, mais en ray tracing, ce qui fait que l'approche est assez différente. J'ai aussi appris à utiliser les cube maps en OpenGL. De plus, je prévois de lire les livres en références afin d'en apprendre plus.

Je pense qu'il serait utile d'avoir plus d'heures de cours sur le PBR avant de faire ce TP, car j'ai trouvé qu'il était assez difficile d'enchaîner le cours et le TP directement. Pour ce qui est de la structure du code, cela pourrait être pratique qu'il soit possible de la débugger en utilisant RenderDoc.