Compte rendu du Projet de Programmation 3D

Phase 1 et Phase 2

Ludovic LONLAS

9 novembre 2022

Workflow

Pour ce projet, j'ai utilisé VSCode ainsi que VSCodium comme IDE, me permettant de compiler mon code en faisant Ctrl+Shift+b. J'ai aussi utilisé une extension me permettant de voir les fichier .ppm exporté et de pouvoir les enregistré en .png . Pour pouvoir codé en OpenGL sur ma machine Windows, j'utilise WSL2 ainsi que Xwindow, cela me permet de pouvoir voir les fenêtres OpenGL.

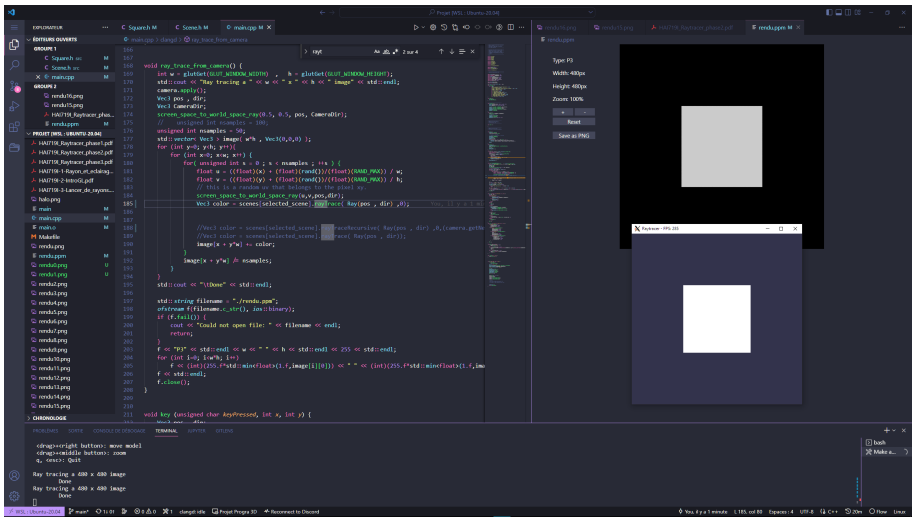

FIGURE 1.1 – Workflow OpenGL

Phase 1

Pour toutes les fonctions d'intersections, je ne suis pas aller chercher sur internet la solution, toutes les fonctions sont donc de moi.

raytrace()

Au début raytrace() ne faisait qu'appeler "spheres[0].intersect(rayStart)" et renvoyait le matériau si il y avait une intersection, par la suite cette ligne à été remplacé par "computeIntersection" pour gérer des scènes plus complexes.

```
1
2  Vec3 rayTrace( Ray const & rayStart, double startdist ) {
3      RaySceneIntersection inter = computeIntersection(rayStart,startdist);
4      if (!inter.intersectionExists) return Vec3(0,0,0);
5      return getRayMaterial(inter).diffuse_material;
6  }
7
```

intersect() Sphere.h

Pour calculé l'intersection entre un rayon et une sphère, je commence d'abords par projeté le centre c de la sphère sur le rayon r, si la distance cc' est moins élevé que le rayon de la sphère alors il n'y a pas d'intersection, sinon je peux calculé grâce au théorème de Pythagore les 2 points d'intersections.
Voici un schéma montrant comment fonctionne la fonction d'intersection.

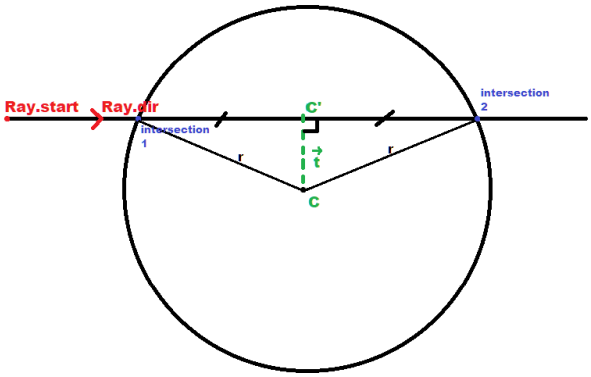

FIGURE 2.1 – Schéma d'intersection de la sphère

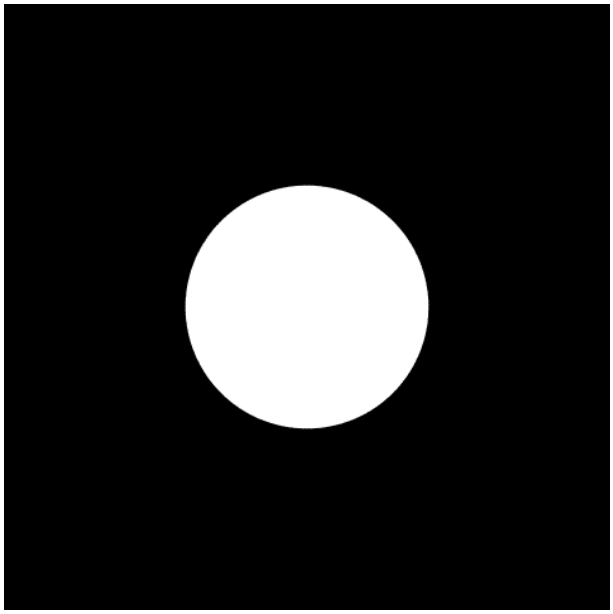

FIGURE 2.2 – Rendu de la sphère dans la scène 1

intersect() Square.h

Pour cette partie au lieu de faire une projection grâce à la normal du Quad, j'ai décidé d'utiliser un changement de plan pour projeté le rayon sur le Quad. J'ai dû implémenter une inversion de matrice. J'ai eu du mal mais ça marche, je pense cependant qu'une projection grâce à la normal aurait pu être plus rapide pour les 2 premières phases.

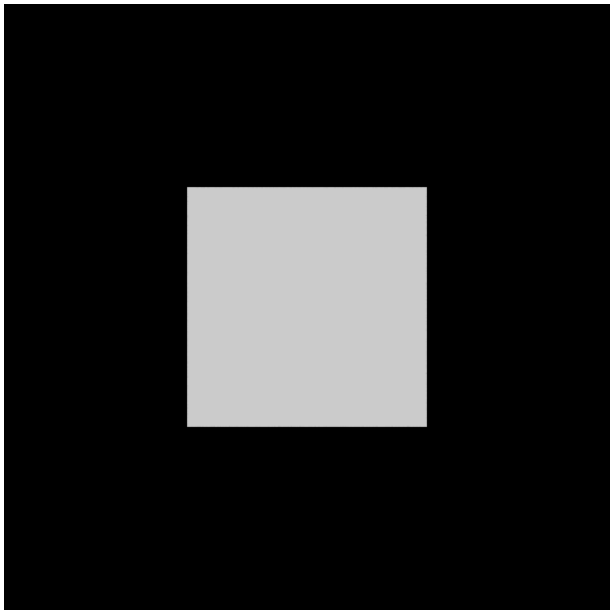

FIGURE 2.3 – Rendu d'un carré dans la scène 2

computeIntersection()

Mon implantation de computeIntersection() prend beaucoup de ligne car je réutilise le code d'enregistrement 3 fois, une fois pour chaque type d'objet.
La fonction appelle intersect() sur tout les objets et renvoie celui avec le plus petit t.

```
RaySceneIntersection closest;
for(int i=0; i<meshes.size();i++){
    RayTriangleIntersection m = meshes[i].intersect(ray);
    if(m.intersectionExists && m.t<closest.t&&m.t!=0&&m.t>mindist){
        closest.intersectionExists = true;
        closest.typeOfIntersectedObject=0;
        closest.rayMeshIntersection=m;
        closest.t=m.t;
        closest.objectIndex=i;
        closest.normal = m.normal;
        closest.pos = m.intersection;
        // closest.bouncedir = m.bouncedir;
    }
```

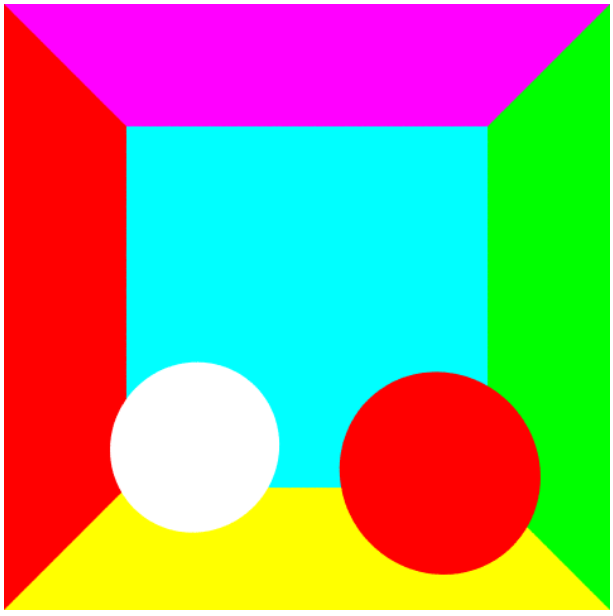

FIGURE 2.4 – Rendu de la scène 3

Caméra

Au début la caméra n'a pas de Z-near ce qui pose problème pour la scène 3, où le mur de devant se trouve derrière la caméra OpenGL mais le mur se trouve devant la caméra de raytracing.
Pour régler ce problème il faut introduire un test de distance pour le premier lancé de rayon en s'assurant d'en faire un plan.
Voici mon implémentassions de ce test :

```
1 Vec3 color = scenes[selected_scene].rayTraceRecursive(
2 Ray(pos , dir)
3 ,0
4 ,(camera.getNearPlane()+LAMBDADIS)/(dir.dot(dir, CameraDir)));
```

Phase 2

La phase 2 est assez courte, il suffit juste d'implémenter l'illumination de Phong et les ombres adoucies

Phong

Ayant réalisé une illumination de Phong lors de l'année précédente, j'ai réutilisé ce code en enlevant les valeurs liée au lumières, elles pourront être ré-implémenté par la suite quand il y aura plus de lumières.

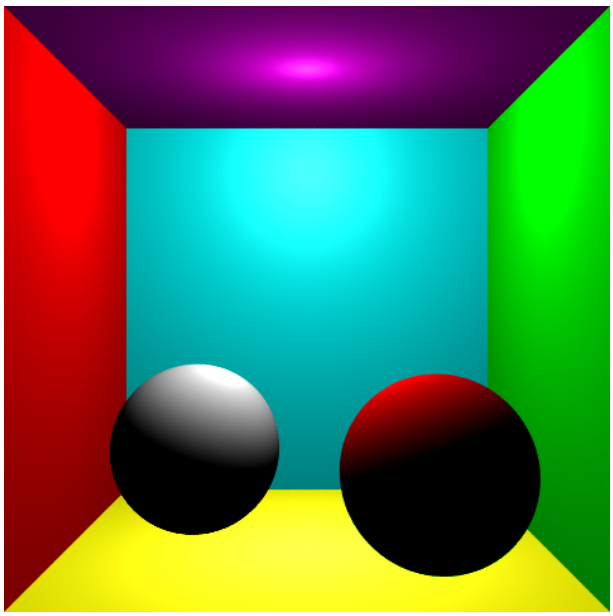

FIGURE 3.1 – Rendu de la scène 3 en utilisant Phong

Ombre adoucies

Pour la dernière partie de la phase 2, j'ai réalisé les ombres portées ainsi que les ombres adoucies en même temps.
Pour réalisé des ombres arrondis j'implémente le type de lumière "LightType_Quad", je lui passe un quad en entrée et quand je calcul les ombres avec ce type de lumière, j'échantillonne aléatoirement le Quad par rapport à ces 2 premiers cotés et je fais la moyenne du nombre de rayon qui peuvent atteindre des échantillons pour obtenir le niveau d'ombre.

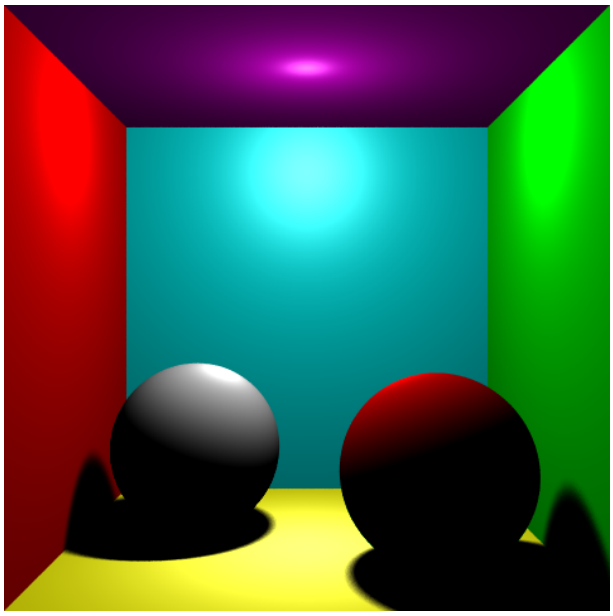

FIGURE 3.2 – Rendu de la scène 3 en utilisant Phong avec des ombres adoucies