Compte rendu du Projet de Programmation 3D

Phase 3 partie 1

Ludovic LONLAS

7 décembre 2022

Pour cette partie du projet j'ai commencé en essayant d'optimiser mes fonctions d'intersections pour les squares pour pouvoir plus facilement implémenter l'intersection avec les mesh, ce qui a créer beaucoup de bugs car je ne faisait pas bien certaines parties de l'intersection du quad.
Après avoir réglé le problème j'ai pu tester la différence de performance entre ma fonction d'intersection et une trouvé sur internet. En moyenne et sur ma machine, mon intersection est 15% plus lente que celle venant du web. Cependant en mettant en mémoire la matrice de transformation ma fonction devient plus rapide d'environ 10% pour chaque intersection après la première.

Pour charger un maillage j'utilise la fonction Load_OFF() présent dans Mesh.cpp cependant pour la deuxième partie du projet, je vais devoir utiliser un autre format de fichier que .off pour plus facilement pouvoir charger les u,v et les textures.

FIGURE 1 – Rendu de Suzanne dans mon programme

Comme vous pouvez le voir sur la figure précédente, j'ai implémenté l'interpolation des normales. Pour cela je passe sur tout les triangles du maillage pour relié chaque vertex à un ou plusieurs triangles, j'associe à ce vertex la moyenne des normales des triangles liées. Voici la fonction qui calcul la normal de chaque point du maillage :

```
1  void Mesh::recomputeNormals () {
2      for (unsigned int i = 0; i < vertices.size (); i++)
3          vertices[i].normal = Vec3 (0.0, 0.0, 0.0);
4      for (unsigned int i = 0; i < triangles.size (); i++) {
5          Vec3 e01 = vertices[triangles[i].v[1]].position
6                      - vertices[triangles[i].v[0]].position;
7          Vec3 e02 = vertices[triangles[i].v[2]].position
8                      - vertices[triangles[i].v[0]].position;
9          Vec3 n = Vec3::cross (e01, e02);
10         n.normalize ();
11         for (unsigned int j = 0; j < 3; j++)
12             vertices[triangles[i].v[j]].normal += n;
13     }
14     for (unsigned int i = 0; i < vertices.size (); i++)
15         vertices[i].normal.normalize ();
16 }
```

Maintenant lors d'une intersection avec un maillage je calcule la normale du triangle en utilisant le Barycentre du point d'intersection avec les normales de ces trois vertex. Je pourrais par la suite utilisé ce même calcul pour interpoler les UVs du maillage.

```
areaV[0]=( Vec3::cross(
    *rightPoint-inter->intersection,
    *upPoint-inter->intersection)).length()/2.0;
areaV[1]=( Vec3::cross(
    *bottom_left-inter->intersection,
    *upPoint-inter->intersection)).length()/2.0;
areaV[2]=( Vec3::cross(
    *rightPoint-inter->intersection,
    *bottom_left-inter->intersection)).length()/2.0;

double totalA= areaV[0]+areaV[1]+areaV[2];
inter->normal = vertices[triangles[i][0]].normal*areaV[0]
            +vertices[triangles[i][1]].normal*areaV[1]
            +vertices[triangles[i][2]].normal*areaV[2];
inter->normal/=totalA;
```

En même temps j'ai rectifié les UVs de mes quads et sphères pour pouvoir par la suite pouvoir y appliquer des textures.

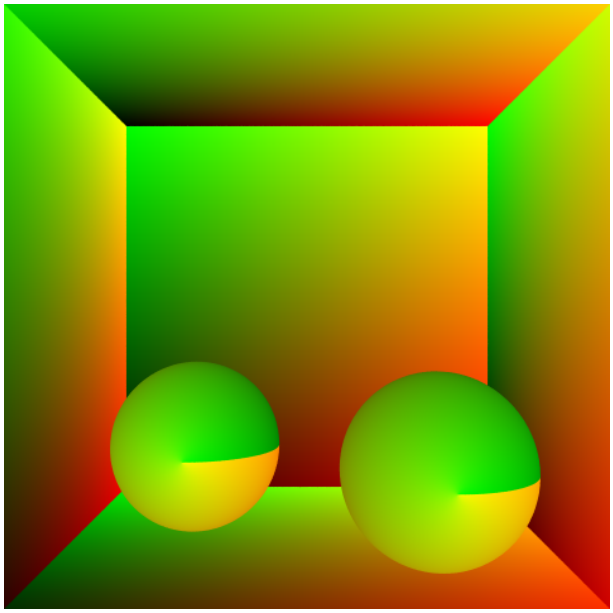

FIGURE 2 – UVs de la Boîte de Cornell - en rouge U et en vert V

J'ai finalement implémenté la sphère miroir, comparé au reste de ce que j'ai codé, ce fut assez simple. Pour le moment le miroir ne reçoit pas d'information des lumières mais il est possible que j'implémente un peu de shininess pour montrer la lumière qui reflète directement sur le miroir, je n'ai pas non plus implémenté pour le moment un calcul plus complexe des ombres prenant en compte le rebond sur un miroir. Je pensais qu'il y avait un bug dans mon

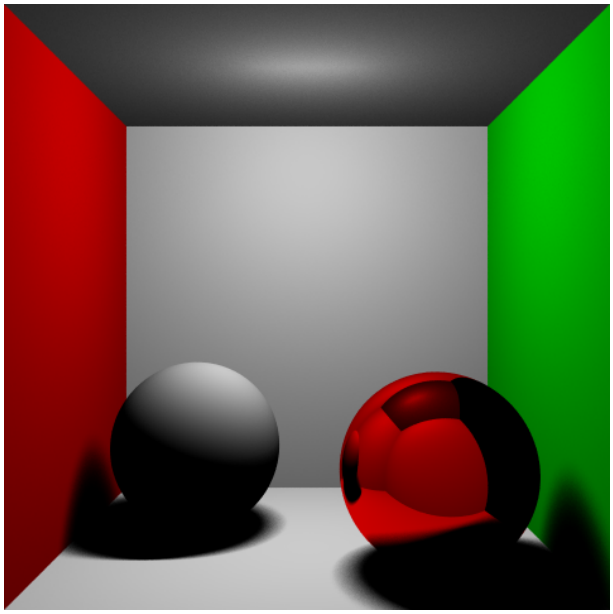

FIGURE 3 – Boîte de Cornell avec un miroir rouge qui absorbe le bleu et le vert

rendu que l'on peut d'ailleurs voir dans la figure 3, le plafond est gris alors que la couleur donnée est blanche, j'ai passé beaucoup de temps à essayer de régler le problème, cependant il ne s'agit peut être pas d'un bug mais du fait que la lumière se trouve très proche du plafond, Phong créerait cet effet.
J'ai préparé pour la prochaine partie une classe boundingBox, je la crée en lui donnant deux coordonnées. Elle permet d'améliorer le temps de calcul et me servira de base pour l'implémentions du k-tree.