Rapport Projet Raytracing

BARRY Timothée

Introduction

Ce travail a été réalisé dans le cadre du MOS 8.5 Informatique graphique. Mon ordinateur portable sur lequel j'ai effectué les rendus est équipé d'un processeur I7-10510U avec 4 cœurs et 8 threads (à prendre en compte pour les temps de calculs).

Objets

Le raytracer peut gérer des sphères, des maillages et des sources de lumière.

Chacun de ces objets est représenté par une classe qui hérite de la classe abstraite Object.

On peut ajouter des objets avec la méthode addObject de la classe Scene.

- Les sphères sont caractérisées par les coordonnées de leur centre, leur rayon, un albedo et leur matériau (cf partie sur les matériaux).

- Les maillages sont caractérisés par les éléments déjà présents dans le template de code (vertices, indices, etc). Ainsi qu'une bounding box, un bvh et un matériau. On peut charger un maillage avec la méthode readOFF, et une texture à partir d'un fichier PNG avec la méthode readPNGTexture.

- Les sources de lumière sont des sphères lumineuses caractérisées par les coordonnées de leur centre, leur rayon et leur intensité.

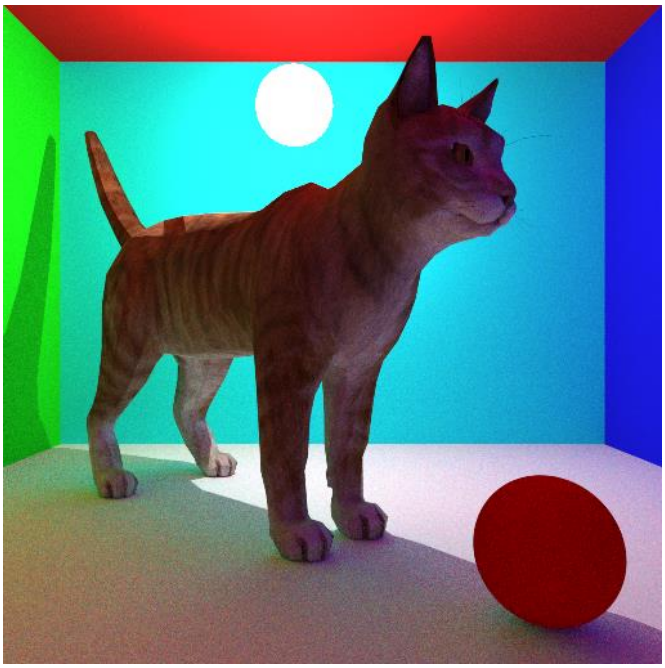

Figure 1 : 512x512, 100 rayons, 5 rebonds, 13min27

Gestion des maillages

On peut effectuer des transformations sur les maillages avec les méthodes translate, rotate et scale de la classe TriangleMesh.

La méthode translate prend en argument un Vector effectue une translation du maillage suivant ce vecteur.

La méthode scale prend en argument un facteur d'échelle effectue une mise à l'échelle du maillage suivant ce facteur, centrée sur le barycentre du maillage.

La méthode rotate prend en argument un angle et un vecteur effectue une rotation du maillage suivant cet angle, et selon l'axe défini par le vecteur, passant par le barycentre du maillage.

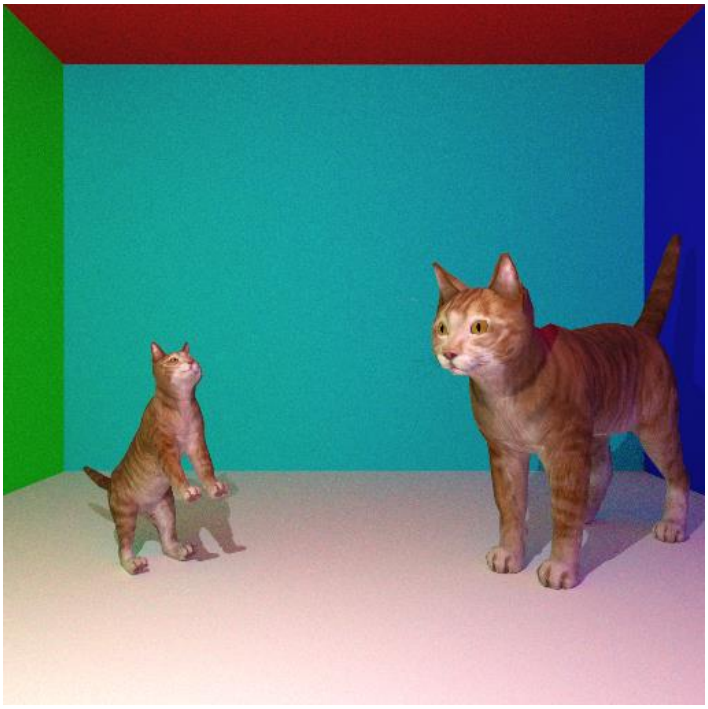

Figure 2 : 512x512, 100 rayons, 10 rebonds, 12min40

Le maillage contient une bounding box globale, ainsi qu'une bvh (Bounding Volume Hierarchy) qui permet de réduire le nombre d'intersections à tester lors du lancer de rayons. La méthode buildBVH construit la bvh du maillage de manière récursive.

La méthode initBVH doit être appelée après avoir ajouté le maillage dans la scène, après les transformations, pour mettre à jour la bounding box et la bvh.

La fonction d'intersection du maillage utilise la bvh pour réduire le nombre d'intersections à tester avec un parcours en profondeur de la bvh: pour chaque noeud, on teste si le rayon intersecte la bounding box du noeud, et si c'est le cas, on teste les enfants du noeud, jusqu'à atteindre une feuille. Si on atteint une feuille, on teste l'intersection de tous les triangles de la feuille.

Enfin, le maillage effectue un lissage de Phong si le fichier le permet, c’est-à-dire si le fichier OBJ contient des approximations de normales au sommets (vn).

J’ai cependant remarqué un problème que je n’ai pas eu le temps de corriger, lié à ce lissage : dans la figure 5, on voit certaines zones noires sur le chat miroir. Cela est du au lissage de Phong. En effet, la normale retourné par la fonction d’intersection est la normale approximée par le lissage. Quand j’effectue le petit hack de faire rebondir le rebond à P + epsilon * N au lieu de P, le N est donc la

fausse normale. Dans certains cas, la fausse normale passe à travers la face, et l'origine du rayon réfléchi est positionné à l'intérieur du chat, d'où les zones noires. Pour limiter ce problème, j'ai considéré qu'il n'y avait pas d'intersection si la distance entre l'origine est le point d'intersection était inférieure à 2* epsilon. (Dans mon cas epsilon = 1e-10)

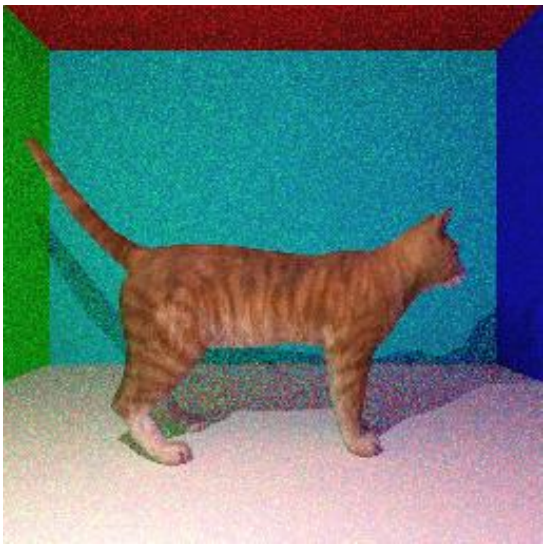

Figure 3 : 256x256, 10 rayons, 5 rebonds

- Sans BVH ni Bounding box: 2min47
- Sans BVH, mais avec Bounding box globale : 40sec
- Avec BVH (maximum de 3 triangles par feuille) : 7sec

Matériaux

Le raytracer peut gérer plusieurs types de matériaux, gérés par une classe abstraite Material.

Chaque matériau hérite de la classe Material et doit implémenter la méthode brdf qui renvoie la couleur du matériau en fonction de l'albedo, de la direction incidente, de la direction réfléchie et de la normale.

L'albedo est ajouté en paramètre de la méthode brdf pour permettre de gérer les textures des maillages.

Parmi ces matériaux, on a :

- Transparent: un matériau transparent caractérisé par un indice de réfraction.

- Mirror: un matériau miroir (parfait) caractérisé par une réflectance entre 0 et 1.

- Diffus: un matériau diffus. Sa brdf est une constante (albedo / PI)

- Blinn-Phong: un matériau suivant le modèle de Blinn-Phong, caractérisé par un coefficient alpha (exposant) et un coefficient de brillance entre 0 et 1. Sa brdf est donnée par la formule de Blinn-Phong: albedo * (n+2) / (2*PI) * cos(theta)^alpha avec alpha l'exposant, et theta l'angle entre la direction de la lumière et la direction de la vue.

Ces matériaux peuvent être ajoutés à des sphères ou des maillages. Par défaut, les objets ont un matériau diffus.

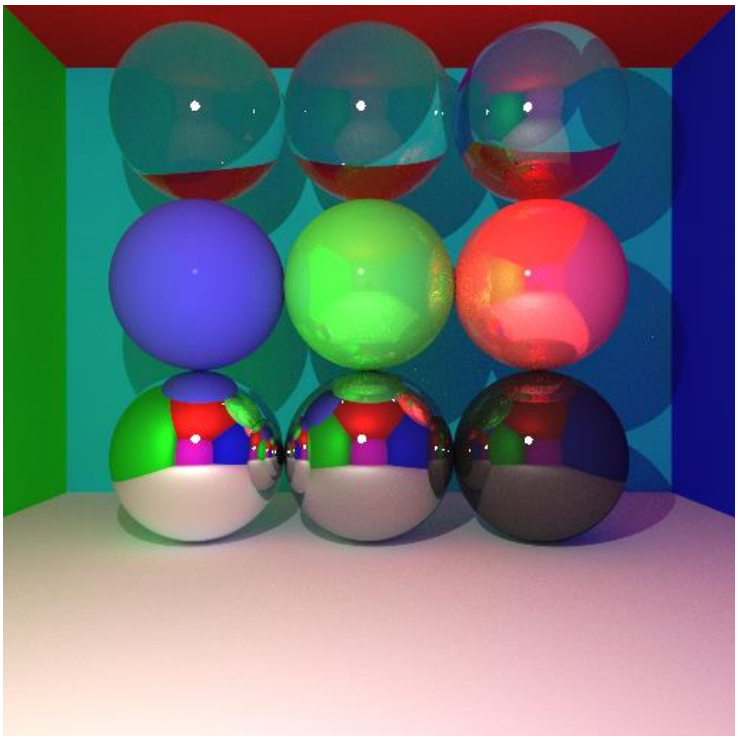

Figure 4 : Du bas vers le haut : 3 boules miroir avec une réflectance différente, 3 boules avec une BRDF de Blinn Phong de alpha 1000 et de brillance variant de 0.01 à 0.1, 3 boules transparentes avec des indices de réfractions différents (eau, verre, diamant). 512x512, 1000 rayons, 5 rebonds, 30min55

Figure 5: 3 chats : Miroir, Blinn Phong et transparent. 512x512, 100 rayons, 10 rebonds, 29min26

Effets

Différents effets peuvent être ajoutés à la scène :

- Anti-aliasing : pour chaque pixel, on lance le rayon dans la direction du pixel avec un léger offset aléatoire dont l'amplitude est fixée et dont la distribution est centrée sur le milieu du pixel. On fait la moyenne des couleurs obtenues pour chaque rayon.

- Depth of field: Cet effet est caractérisé par une ouverture de diaphragme et une distance focale. En plus de l'aliasing, on décale le point d'origine du rayon dans le plan de l'ouverture de diaphragme, et on fait en sorte que les rayons se croisent en un point situé à la distance focale.

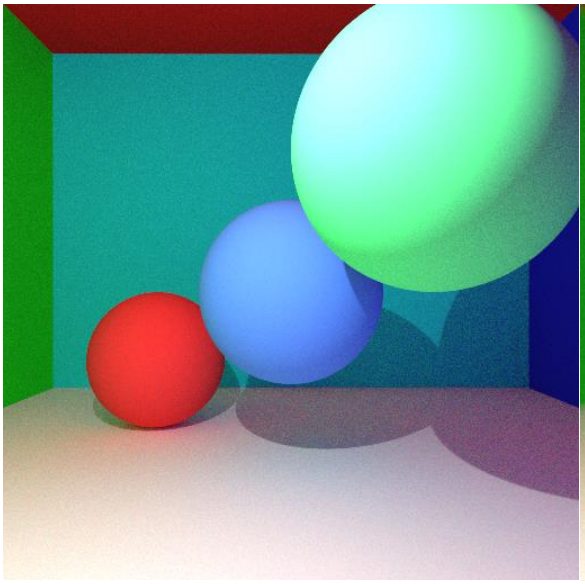

Figure 6: Sans DOF

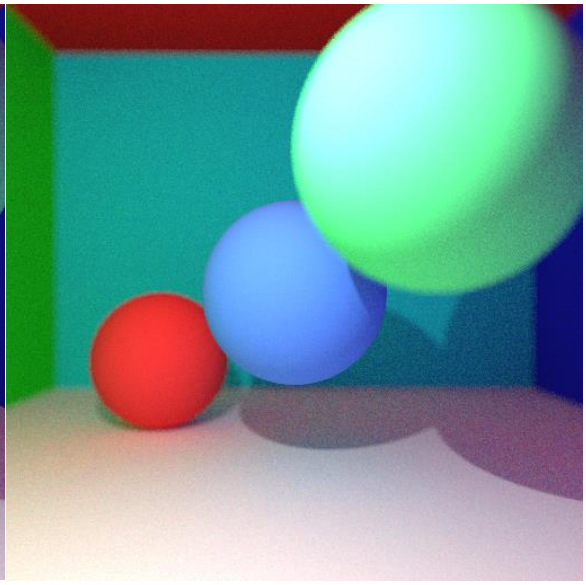

Figure 7: Avec DOF (ouverture = 2, distance focale = 55)

Figure 8 : Sans anti-aliasing

Figure 9: Avec anti aliasing (écart type BoxMuller de 0.4)

Scène

La scène contient une liste d'objets pouvant être des sphères, des maillages ou des sources de lumière. On peut ajouter des objets avec la méthode addObject de la classe Scene.

La scène contient une méthode getColor récursive, qui renvoie la couleur d'un rayon lancé dans la scène. Cette méthode gère la réflexion, la réfraction et la lumière indirecte.

La scène contient plusieurs méthodes qui gèrent le comportement de différents matériaux : miroir, transparence, diffusion et Blinn-Phong.

Optimisations

Pour réduire le temps de calcul, j'ai effectué quelques optimisations.

Pour le calcul des ombres, j'ai ajouté une fonction fastIntersect (dans la classe abstraite objet, et dans la classe Scene), qui prend uniquement un rayon en paramètre, et retourne uniquement un booléen à true ou false selon qu'il y ait une intersection ou non. Cela rend le calcul plus rapide, car dans le cas des ombres, on s'intéresse uniquement à s'il le rayon en direction de la lumière rencontre un obstacle ou non. Au moindre obstacle, que ce soit une sphère ou un triangle de maillage, on retourne true. Cela fait que s'il y a un obstacle, on ne continue pas de vérifier les autres objets car on ne s'intéresse pas à la distance d'intersection, contrairement à la fonction intersect classique qui a besoin de trouver l'intersection la plus proche. De plus, on s'affranchit des calculs du point d'intersection, de la normale et tous les autres paramètres qui nous sont inutiles ici.

Aides externes :

Les librairies et codes que j'ai utilisés sont stb_image / stb_image_write, pour le chargement et l'écriture d'images ainsi que les templates de code fournis par le professeur.

Pour certaines fonctions simples mais peu intéressantes à implémenter comme les opérations vectorielles telles que la rotation d'un vecteur autour d'un axe, j'ai utilisé GitHub Copilot pour gagner du temps. Je ne l'ai cependant pas utilisé pour les parties qui demandaient une logique algorithmique un peu plus poussée, afin d'éviter de créer des bugs que j'allais passer des heures à débuguer.