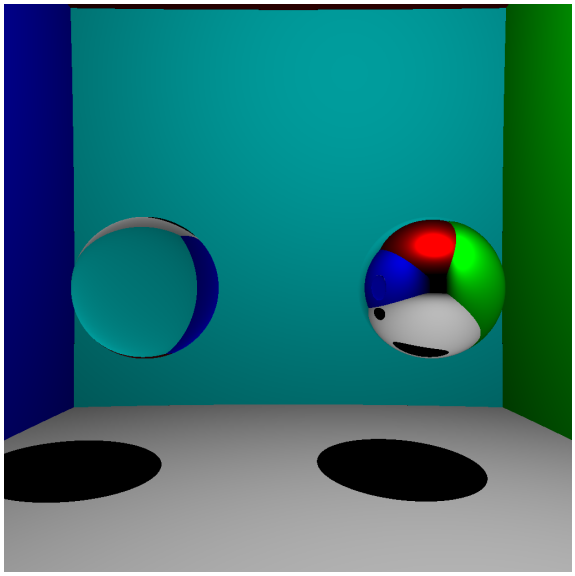
ÉCOLE CENTRALE LYON

Réalisation d'une scène par Raytracing

MOS 2.2 : Informatique Graphique

Etudiants :
Ravi Hassanaly

Enseignant référent :
Nicolas Bonneel



1er avril 2020

1 Mise en Contexte

Dans le cadre de ce cours nous avons étudié pas à pas la constitution d'une scène virtuelle par raytracing. Le principe est d'envoyer des rayons de la caméra à travers chaque pixel de l'image et d'en déterminer la couleur selon les objets intersectés par le rayon. Pour cela on commence par un exemple basique ou on calcule l'intersection du rayon avec une sphère, puis on complexifie graduellement la simulation en ajoutant des paramètres : prise en compte de la source de lumière, de l'éclairage indirect, des propriétés des surfaces des objets, des propriétés optiques... Cela permet de tendre vers un rendu de plus en plus réaliste.

2 Intersection avec une sphère

Nous commençons donc l'étude par le cas le plus simple possible, on crée une sphère de rayon r et on envoie un rayon à travers chaque pixel de notre image : si le rayon a une intersection avec la sphère, alors le pixel est blanc, il est noir sinon.

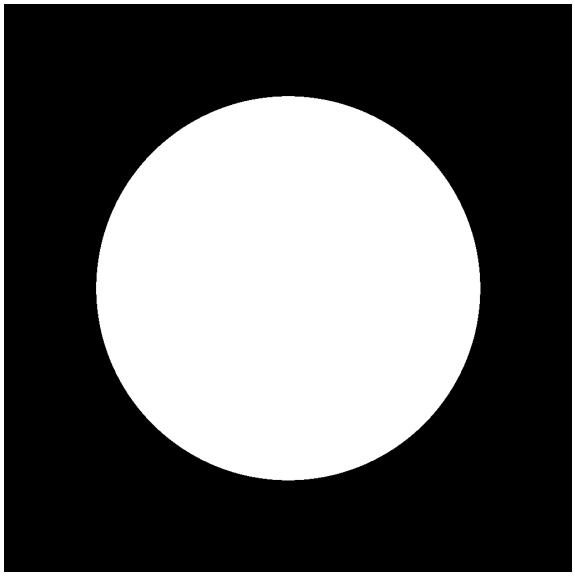


FIGURE 1 – Intersection entre une sphère et les rayons

On obtient ainsi le résultat ci-dessus, on observe un cercle qui correspond à la projection de la sphère sur le plan de l'image.

Ce premier exemple simple permet de mettre en place les principales classes permettant de définir les rayons lumineux ainsi que les sphères et de prendre en main le pipeline d'enregistrement de l'image.

3 Ajout d'une source de lumière

On ajoute ensuite une source de lumière défini par sa position et son intensité. Maintenant, à chaque fois qu'il y a une intersection entre une sphère et la rayon qui part de la caméra, on calcule l'intensité lumineuse réfléchie par ce point de la sphère. Pour cela on calcule l'angle entre la normale à la surface de la sphère et le vecteur directeur de la

direction entre ce point et la source de lumière ponctuelle. Cela va pondérer l'intensité de chaque point.

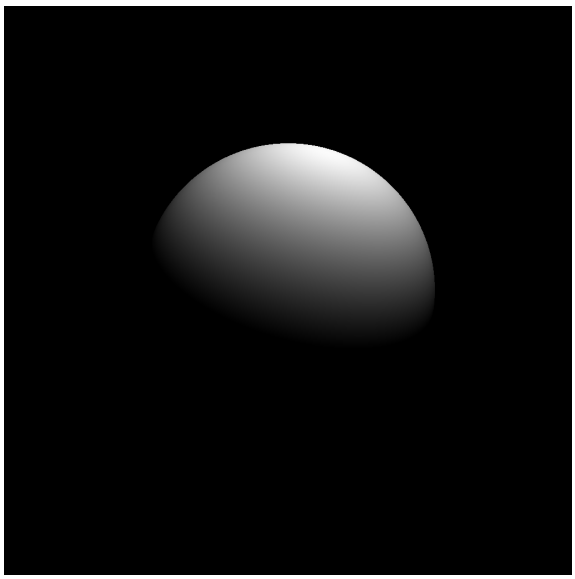


FIGURE 2 – Prise en compte de la source de lumière ponctuelle

Cette technique permet de donnée du volume aux objets de la scène.

4 Couleur

On ajoute maintenant la couleur en tant que propriété de l'objet *sphre* et on calcule l'intensité de chaque couleur du pixel grâce à cette propriété nommé albedo.

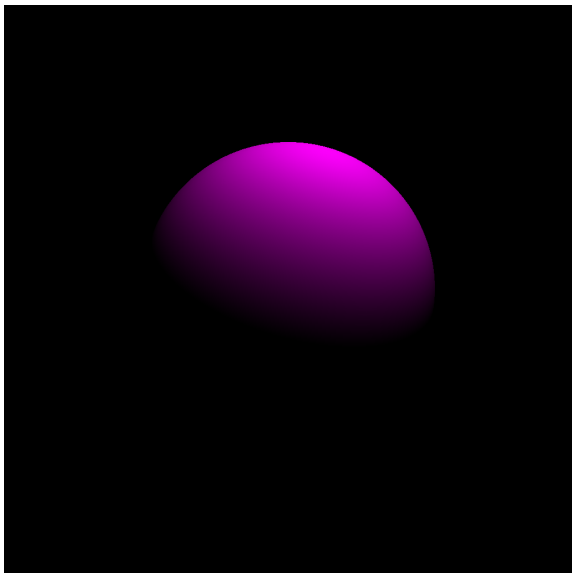


FIGURE 3 – Ajout de la couleur pour la sphère

On obtient bien une sphère colorée, ici on a une valeur de 1 pour le rouge et le bleu et 0 pour le vert ce qui nous donne cette couleur mauve.

On peut maintenant ajouter des sphères géantes de manière à constitué une scène.

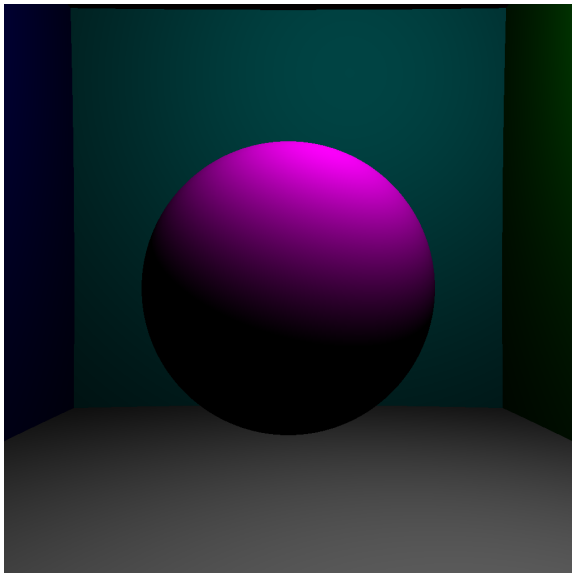


FIGURE 4 – Scène complète

On obtient donc la scène suivante avec les murs et le sol. On a pour cela mis en place la classe *Scene* qui permet de gérer plusieurs sphères.

5 Ajout des ombres

Pour ajouter plus de réalisme à la scène, on peut calculer les ombres des objets. Pour cela, lorsqu'on obtient une intersection entre une sphère et un rayon, au lieu de renvoyer l'éclairage direct comme valeur du pixel, on va vérifier s'il n'y a pas un autre objet entre ce point et la source de lumière. Si on détecte un objet c'est que le point est à l'ombre et on le renvoie en noir.

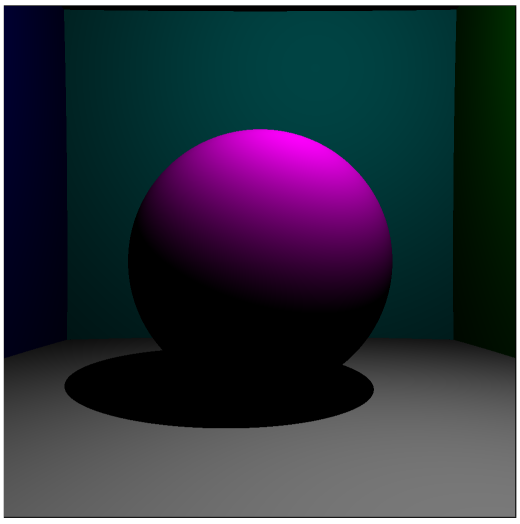


FIGURE 5 – Ajout de l'ombre

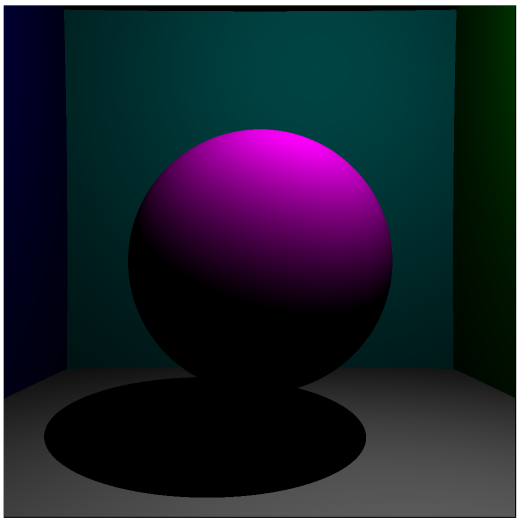


FIGURE 6 – Ajout de l'ombre

On obtient bien les ombres de la sphère principale au centre de la scène sur le sol (vu que la source de lumière est sur le haut).

Pour éviter le bruit, il ne faut pas oublier à décaler légèrement le point d'intersection initial, sinon celui-ci peut être confondu avec lui même à cause des erreurs d'imprécision du codage des nombres flottants.

6 Correction gamma

Afin d'avoir un rendu qui correspond plus à la perception humaine, on applique la correction gamma au valeur des pixels.

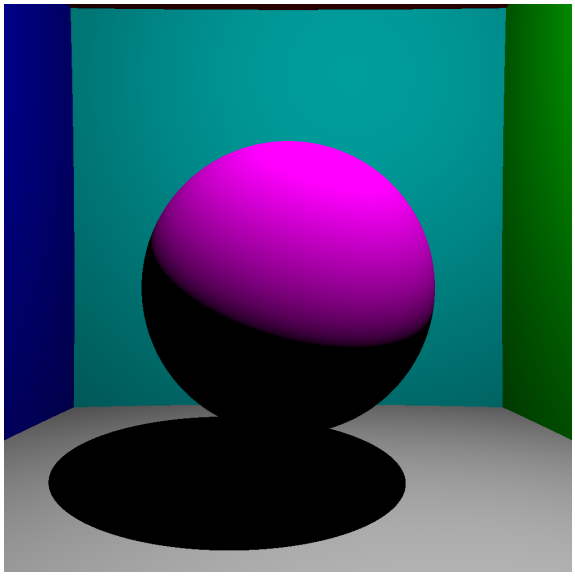


FIGURE 7 – Correction gamma

On obtient ici une figure avec moins de contraste mais perpétuellement plus juste.

On a beaucoup augmenté l'intensité de la lumière pour ne pas avoir une image trop sombre.

7 Surface miroir

On souhaite maintenant ajouter des propriétés aux sphères. On traite dans un premier temps le cas des surfaces miroirs : lorsqu'une intersection est trouvée avec une sphère dans la surface est miroir, on calcule le rayon réfléchi par cette surface et on applique récursivement l'algorithme précédent. Ainsi la couleur du pixel sera celui du premier objet non miroir rencontré par le rayon et ses réflexions.

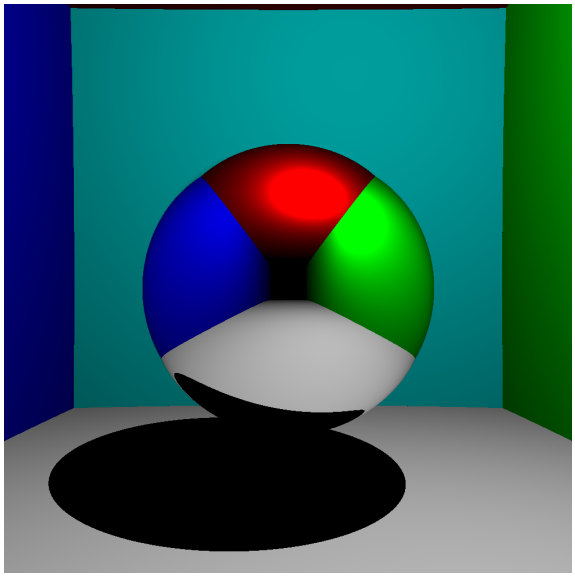


FIGURE 8 – Boule ayant une surface miroir

On peut maintenant observer à travers la boule la couleur rouge du plafond.

Pour éviter d'avoir un algorithme qui ne se termine pas dans le cas par exemple de deux surfaces miroirs l'une en face de l'autre, on limite le nombre de rebonds à 5.

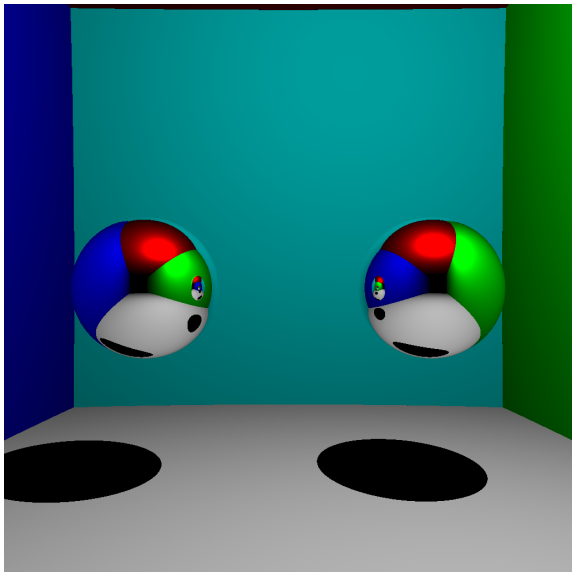


FIGURE 9 – Reflexion avec deux boules miroirs

8 Objet Transparent

On souhaite ensuite traiter le cas des objets transparent (en verre par exemple). Pour cela on applique simplement les lois de l'optique géométrique : la loi de Snell-Descarte pour calculer le rayon réfracté. On calcule ensuite récursivement la couleur du pixel à partir de la nouvelle direction déterminée.

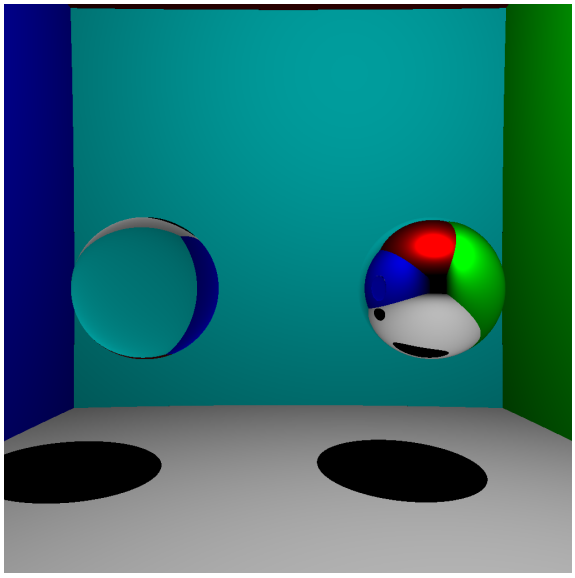


FIGURE 10 – Ajout d'une boule transparente

On observe ainsi la couleur de l'objet derrière la boule transparente à travers celle-ci.

9 Prise en compte de l'éclairage indirect

Jusqu'ici on a pris en compte seulement l'effet de l'éclairage direct dans la simulation de la scène. Pour ajouter du réalisme au rendu, il faut aussi ajouter les effet de l'éclairage indirecte. En effet lorsque qu'on observe une ombre, elle apparaît rarement comme toute noire dû aux réflexions de la lumière dans les autres objets de la scène.

Pour éviter les calculs trop long, on considère seulement quelques rayons aléatoires. On va donc ajouter les contributions à l'éclairage direct. Autrement dit, lorsqu'on calcule la valeur d'un pixel, on compte l'éclairage direct de celui-ci, puis on détermine une direction aléatoire dans laquelle on renvoie un rayon pour obtenir une contribution indirect.

Cette méthode nous donne un résultat très bruité, on va donc faire une moyenne de plusieurs rayons (une dizaine pour éviter un temps d'exécution trop long) afin d'obtenir un rendu plus réaliste.

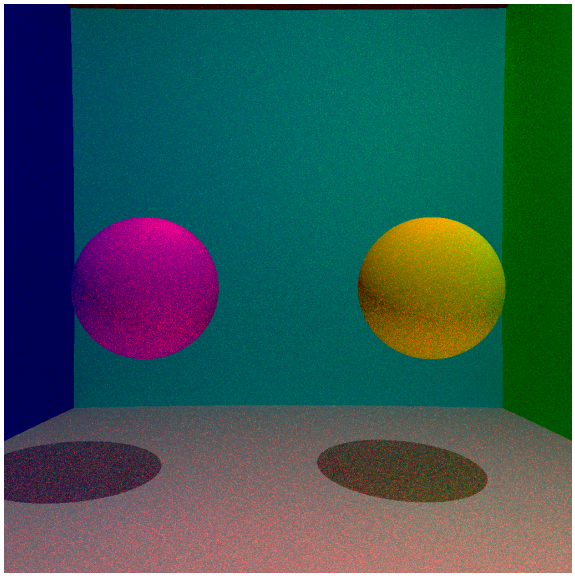


FIGURE 11 – Prise en compte de l'éclairage indirect

On remarque que le résultat est encore un peu bruité, mais je n'ai pas réussi à corrigé ces erreurs. Il doit y avoir un problème dans le programme car le temps d'exécution est déjà très long (une dizaine de minute).

Mais le résultat correspond à peu près à ce que l'on souhaite obtenir, on passe donc à la suite de l'étude.

10 Source de lumière étendue

Jusqu'à maintenant la source de lumière était ponctuelle. On peut changer cela en ajoutant une sphère qui jouera la rôle de la lumière (comme une ampoule par exemple). On peut faire cela naïvement en renvoyant l'intensité de la lumière si un rayon croise la sphère lumière. En comptant sur les contributions indirects on pourrait ainsi reconstruire la scène.

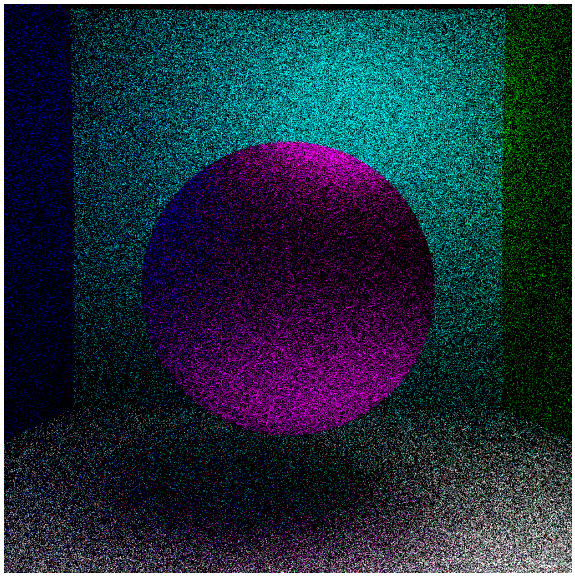


FIGURE 12 – Source lumière étendue

On remarque que l'image est très bruitée, cela est dû au fait que la probabilité d'arriver sur la sphère lumière reste assez faible. On peut changer cela en adaptant la probabilité d'avoir un rayon en direction de la lumière.

Malheureusement après 14 minutes de calcul, l'image obtenue était toute noire et je n'ai pas eu le temps de trouver la source du bug.

11 Anti-aliasing

Dans la méthode utilisé précédemment, on envoie les rayons par le centre du pixel, donc soit l'algorithme trouve une intersection avec une sphère soit il n'en trouve pas. Il en résulte un effet de crénelage au niveau des bords de objets comme l'illustre la figure ci-dessous.

Pour éviter cela on peut mettre en place l'anti-aliasing, qui consiste a envoyer des rayons à travers toute la surface du pixel et non plus seulement le centre de celui-ci. On utilise pour cela la méthode de Box-Muller vu en cours pour avoir des rayons avec une distribution gaussienne centré au centre du pixel et d'écart type 1.

FIGURE 13 – Crenelage

En mettant en place l'anti-aliasing on peut observer l'effet de flou sur la figure 12 au lieu des créneaux.

12 Repository Github

`https://github.com/ravih18/Raytracing`