Ray tracing

Florent Guiotte et Frédéric Becker

17 janvier 2015

Table des matières

1 Introduction

L'objectif de ce TP est d'écrire la méthode qui retourne la couleur à dessiner sur une image dans un projet de ray tracing. Nous nous occupons donc d'implémenter les algorithmes de lancé de rayons [1] avec comme paramètres un vecteur représentant le rayon à lancer à travers la scène, une liste de géométries représentant les objets (entièrement composés de triangles, associé à un matériau) et de la liste des lumières de la scène. Notre méthode sera appelée pour chaque pixel de l'image.

Pour réaliser ce TP, nous avons utilisé le projet et plusieurs classes/objets issus du logiciel de raytracing développé par Fabrice Lamarche.

1. Les algorithmes sont issus des cours de Kadi Bouatouch, notamment ceux du modèle de Phong, la réflectance...

2 Détail de l'algorithme

2.1 Intersection

Le premier calcul à faire est de trouver l'intersection la plus proche de la caméra entre le rayon émis de cette dernière et le triangle d'un objet qui compose la scène (si l'intersection existe).

Nous avons implémenté une méthode qui permet d'obtenir l'intersection la plus proche de la source d'émission du rayon et la scène afin d'alléger la méthode *sendRay* et pour la réutiliser un peu plus loin pour les ombres [2]. Cette méthode parcours les triangles qui composent les objets de la scène, et détecte s'il y à intersection. Si c'est le cas, on la conserve pour pouvoir la comparer si une nouvelle intersection est trouvée. En effet, afin d'avoir le rendu de la face visible à l'écran, il ne faut prendre en compte que celle qui est la plus proche de la caméra, les autres seront «cachées» par cette face, donc n'ont pas besoin d'être dessinées. Techniquement, nous utilisons juste une *intersection* en cache afin de la comparer avec la nouvelle, et de conserver la plus proche des deux (plutôt que d'utiliser un Z-buffer par exemple).

Si il n'y a aucune intersection entre le rayon et la scène, alors on renvois une couleur par défaut (couleur de fond).

2.2 Ombrage

Avant d'appliquer le modèle de Phong, il faut vérifier que l'intersection est éclairée par une lampe. On parcours les lampes de la scène, pour chacune d'entre elle on créé le rayon qui part de la lampe dans la direction de l'intersection trouvée précédemment. On l'appellera rayon d'ombrage.

Il suffit de rappeler la méthode décrite précédemment pour trouver l'intersection la plus proche de la lampe. Si le triangle correspondant à l'intersection de ce rayon d'ombrage est le même que celui qui correspond à l'intersection trouvée dans la première partie, c'est que ledit triangle est éclairé par cette lumière.

2.3 Phong

Lors du parcours des lampes de la scènes, si le triangle correspondant à l'intersection est éclairé par une lampe, alors on applique les calculs du modèle de Phong qui gèrent de manière réaliste l'intensité d'éclairage d'une surface.

Les calculs se décomposent en deux parties, le calcul de l'éclairage diffus de la surface et le calcul du reflet spéculaire (tache lumineuse, reflet d'une source de lumière dans l'objet).

Le calcul du diffus prend en paramètre un vecteur Kd qui est renseigné dans le matériau du triangle (il représente une couleur) à laquelle on multiplie l'intensité de la lumière multiplié par le cosinus [3] de la normale de l'intersection

2. partie 2.2 page 2

3. Pour accélérer le calcul, le cosinus est obtenu avec le produit scalaire entre deux vecteurs normalisés, soit la multiplication de ces deux vecteurs

et la direction de la lumière.

Le calcul du spéculaire prend en compte une rugosité n du matériau en plus de sa couleur *Ks*. Par rapport au calcul du diffus, le cosinus est fait entre la direction idéale du reflet spéculaire (soit la réflection de la direction de la lumière par rapport à la normale) et le rayon de la vue. Le cosinus est élevé à la puissance n afin d'influencer le cône de la tache spéculaire.

L'addition des deux partie renvoie la couleur de l'intersection de l'objet. On additionne aussi les résultat de chaque lampes en cette intersection pour avoir notre couleur.

2.4 Réflectance

En plus de PHONG, nous calculons la réflectance de certains matériaux. Ce calcul permet de rendre certains murs comme des miroirs, ils reflètent les autres objets de la scènes. La réalisation de cette partie utilise la récursivité de notre méthode *sendRay*.

Nous avons lié cet effet à la valeur du reflet spéculaire d'un matériau. Ainsi, en plus du calcul de PHONG, si le matériau est spéculaire et si le niveau de récursion est inférieur à celui précisé au moment du lancé de rayon, alors on construit le rayon réfléchi au point d'intersection. Puis on lance ce rayon avec *sendRay* pour obtenir la couleur (multipliée à la valeur *Ks* à additionner au résultat.

Le résultat de toutes les étapes décrites précédemment sont visibles en figure 1 page 4.

3 Amélioration

3.1 Réfraction

La réfraction permet de rendre un matériau transparent en déformant la direction du rayon le traversant (effet optique propre au changement de milieux). Nous n'avons pas réalisé cette amélioration, cependant, pour l'implémenter il suffirai d'utiliser la récursivité de *sendRay* avec un nouveau rayon (déformé en respectant les propriété du matériau).

3.2 BoundingBox

Pour accélérer le calcul des intersections de la scène, on peut d'abord détecter les intersection avec des objets simplifiés appelés bounding boxes. Même si elles sont implémenté dans le code, nous ne les avons pas prises en compte . La scène de test est composé de trois objets seulement. Et ils sont cubiques (forme la plus simple possible...).

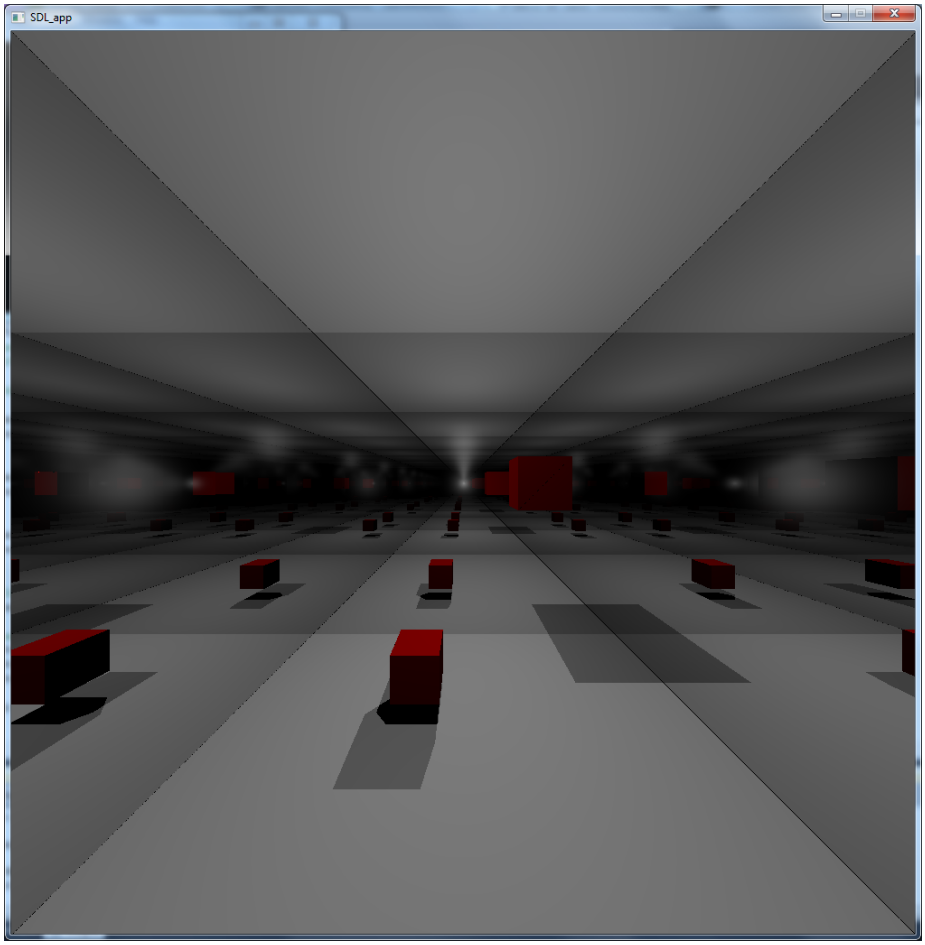

FIGURE 1 – Résultat du lancé de rayon, avec une récursivité max = 15

3.3 Glossy reflection

La réflectance obtenue dans notre implémentation est parfaite. Pour imiter un matériau imparfait il faudrait que les reflets soit plus ou moins déformés en fontion de la distance du reflet. Le calcul théorique est cependant très lourd. À chaque point de reflection il faut lancer un cône, soit plusieurs rayons et les moyenner.

Nous avons tenté d'approcher ce genre de résultat en lançant un nombre fixé de rayons dévié aléatoirement. Le paramètre pour activer cet effet est l'entier *glossyReflection* qui contrôle jusqu'à quelle niveau de récursion l'effet est calculé. Nos résultats sont visibles figure 2 et 3 page 5 et 6.

Ceci est une approche empirique, en étudiant les images obtenues, on peut remarquer des déformations trop importantes dans certaines directions (dévia-

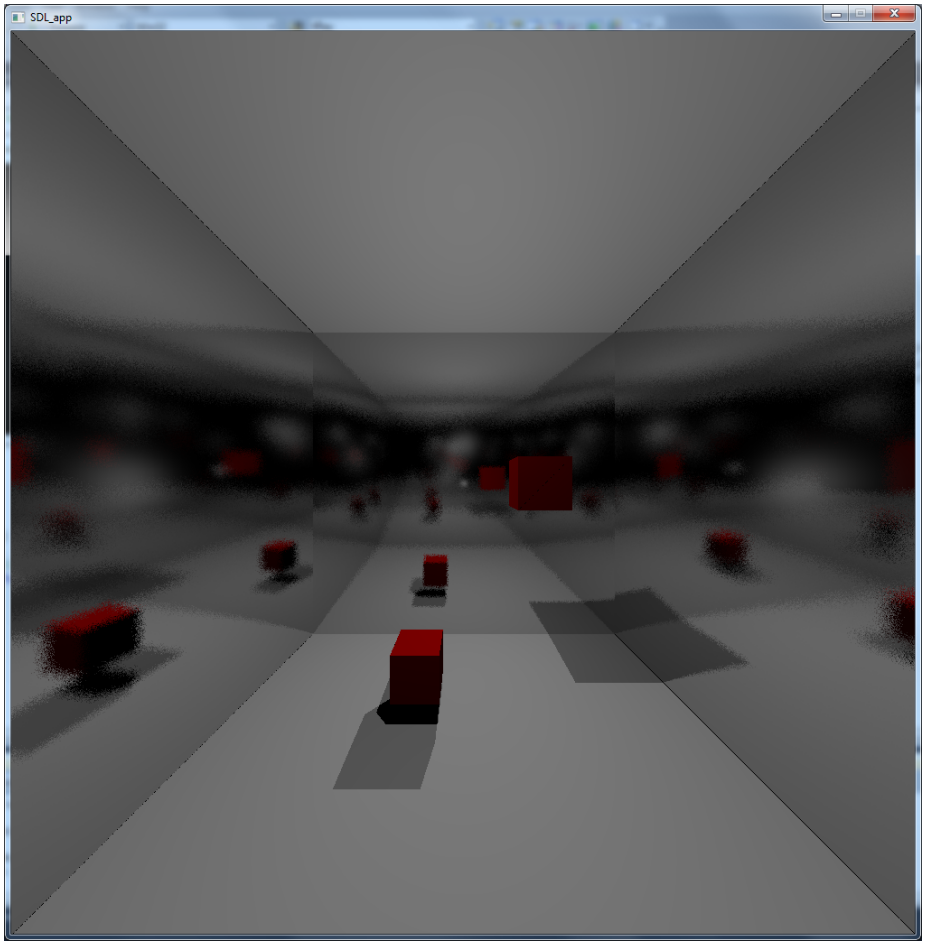

FIGURE 2 – Approche d'une réflectance «glossy» avec 4 rayons réfléchis et une forte déformation

tion des droites, il faudrait vérifier et corriger la façons dont on dévie les rayons réfléchis), mais le reflet obtenu est quand même de plus en plus diffus en fonction de la distance.

3.4 Éclairage ambiant diffus

Pour augmenter le niveau de réalisme de la scène, on devrait pouvoir calculer l'éclairement issus des faces éclairées des autres objets de la scène. On pourrait approcher cet effet en s'inspirant de l'amélioration précédente, mais les temps de calculs seraient exponentiels...

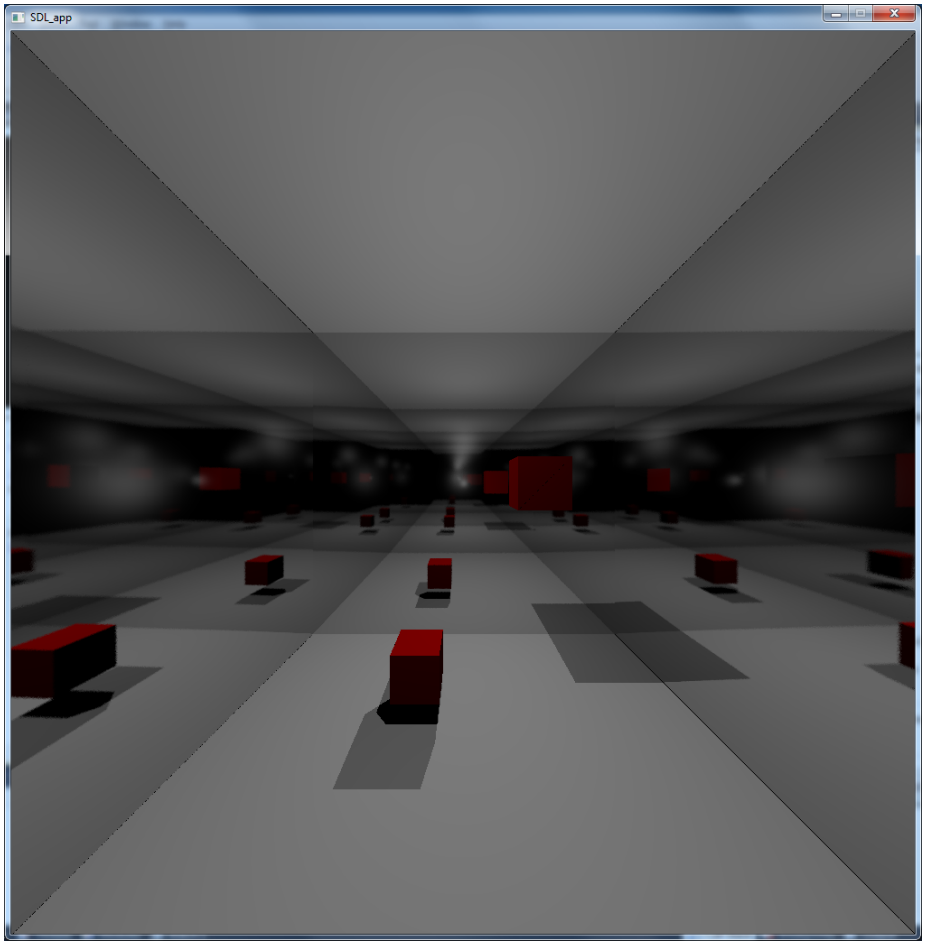

FIGURE 3 – Approche d'une réflectance «glossy» avec 14 rayons réfléchis et une faible déformation

4 Conclusion

Notre implémentation permet d'obtenir le rendu d'une scène 3D en respectant les algorithmes simples tels que le calcul des ombres, Phong et la réflectance. Notre méthode est implémenté en effectuant le moins de copies possible et en respectant la libération mémoire des objets plus utilisés, ce qui nous permet des rendus rapide (moins de 10 secondes pour une image 1080x1080 pixels avec une profondeur max = 15).