



ÉCOLE
CENTRALE LYON

INFORMATIQUE GRAPHIQUE

INTRODUCTION AU RAYTRACING

Léo BESANÇON

19/03/2018

Sommaire

Introduction	3
1 Structure du programme	3
2 Bases de l'affichage de la scène	3
2.1 Premier rendu	3
2.2 Gestion de l'intensité lumineuse	4
2.3 La couleur	4
2.4 Les ombres portées	5
2.5 Correction Gamma	5
2.6 Plusieurs lampes	6
3 Différents types de surfaces	6
3.1 Les surfaces spéculaires	6
3.2 Les surfaces transparentes	7
3.3 Coefficients de Fresnel	8
3.4 La BRDF de Phong	9
4 Amélioration du rendu	10
4.1 Éclairage indirect	10
4.2 Sources de lumières étendues	10
4.3 Depth of Field	10
4.4 Monte-Carlo	11
4.5 Anti-aliasing	11
5 Les maillages	12
5.1 Structures de données et parsing	12
5.2 Affichage du maillage	13
5.3 Structures accélératrices	13
5.4 Gestion des textures	14
5.4.1 Textures procédurales	14
5.4.2 Textures Bitmap	14
6 Animation	15
6.1 De la caméra	15
6.2 Des objets de la scène	15
6.3 Flou de mouvement	15
Conclusion	16

Table des figures

1 Résultat du premier rendu.	4
2 Ajout des variations d'intensité lumineuse.	4
3 Ajout de la couleur.	5
4 Ajout de l'ombre portée.	5
5 Application de la correction Gamma avec $\gamma = 2.2$	6

6	Rendu avec 2 lampes.	6
7	Ici, seule la sphère est un miroir.	7
8	Ici, la sphère ainsi que le mur de derrière sont des miroirs	7
9	Sphère transparente d'indice $n = 1.6$	8
10	Sphère transparente d'indice $n = 2.5$	8
11	Rendu d'une surface transparente avec coefficients de Fresnel	9
12	Rendu d'une sphère avec BRDF de Phong (exposant 20, Couleur : (1, 1, 1), Couleur spéculaire : (0.2, 0.2, 0.2))	9
13	Rendu d'une sphère sous une source de lumière étendue.	10
14	Rendu de deux sphères de distance différentes, avec Depth of Field.	11
15	Rendu d'une sphère avec Monte-Carlo.	11
16	Rendu avec AA (gauche) et sans (droite).	12
17	Rendu du maillage du T-Rex avec un matériau diffus.	13
18	Rendu du maillage avec texture procédurale.	14
19	Rendu du maillage avec texture bitmap.	15
20	Rendu du maillage flou de mouvement lors d'une rotation autour d'un axe vertical.	16

Introduction

L'objectif de ce projet est d'appliquer différentes méthodes de raytracing afin de réaliser le rendu réaliste d'une scène. Pour les premiers rendus, j'ai repris la scène proposée au début du cours pour faciliter le débogage.

1 Structure du programme

Voici un résumé des objectifs des fichiers principaux de mon programme :

- `stdafx.h` : Définit les fonctionnalités de rendu par des macros.
- `Raytracer.cpp/.h` : Il s'agit de la classe principale, qui permet de gérer les paramètres de la scène, créer la caméra et les objets, et faire tourner la boucle principale du programme.
- `Scene.cpp/.h` : Classe qui gère les objets de la scène, les lumières, et les calculs d'intersection.
- `Sphere.cpp/.h` : Classe définissant le comportement d'un objet sphérique.
- `Mesh.cpp/.h` : Classe définissant le comportement d'un maillage (texture, parsing du fichier...). Elle hérite de `Box` afin de créer facilement des boîtes englobantes récursivement.
- `Material.cpp/.h` : permet de créer différents types de matériaux (miroirs, matériaux diffus...) facilement.
- `IAnimatable.cpp/.h` : Classe abstraite dont les objets animables (sphères, maillages, caméra...) doivent hériter. Une animation est définie dans la struct associée, et définit une translation / mise à l'échelle / rotation, mais également un temps de début et de fin. Ainsi, un objet peut posséder plusieurs animations différentes, qui se chevauchent dans le temps ou non !
- `Camera.h` : Définit les paramètres de la caméra.
- `Ray.cpp/.h` : Définit un rayon. Cette classe est surtout utile pour ses routines de génération aléatoire de rayon.
- `Vector.cpp/.h` : Classe outil qui permet de définir des vecteurs 3D, utilisés pour les points dans l'espace mais également les couleurs RGB.

2 Bases de l'affichage de la scène

2.1 Premier rendu

Dans un premier temps, le but est de pouvoir afficher une scène en noir et blanc : un pixel est blanc s'il pointe en direction d'une sphère, noir sinon (FIGURE 1).

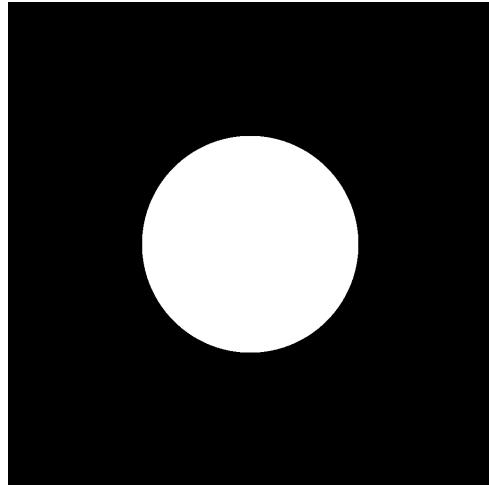


FIGURE 1 – Résultat du premier rendu.

2.2 Gestion de l'intensité lumineuse

Ce premier rendu fonctionne bien, mais nous ne voyons pas encore le relief de la scène. Pour cela, la sphère devrait avoir des points plus ou moins lumineux en fonction de leur position par rapport à la lumière (FIGURE 2).

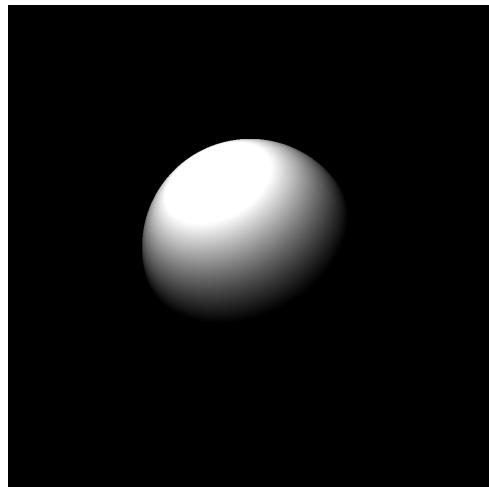


FIGURE 2 – Ajout des variations d'intensité lumineuse.

2.3 La couleur

Afin de gérer et pouvoir afficher des pixels de différentes couleurs, il faut associer à un point trois intensités lumineuses différentes : une pour le rouge, une pour le vert, et une pour le bleu. Chaque objet de la scène possède alors un attribut *couleur*, qui est un vecteur décrivant l'intensité de réflexion pour chacune des composante rouge, vert et bleu (FIGURE 3).

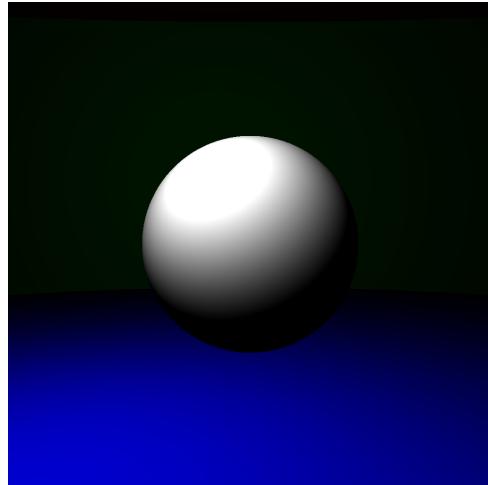


FIGURE 3 – Ajout de la couleur.

2.4 Les ombres portées

La lumière de la lampe de notre scène se propage pour l'instant à tous les points de la scène. Cependant, si un objet se trouve entre la lumière et un autre objet, alors ce dernier ne pourra pas être éclairé : le rayon lumineux sera bloqué par le premier objet.

Pour rendre le rendu plus réaliste, nous regardons lors du calcul de l'intensité lumineuse associée à un point de la scène s'il y a une obstruction de la lumière depuis ce point. Si c'est le cas, alors nous fixons la couleur du point à noir (FIGURE 4).

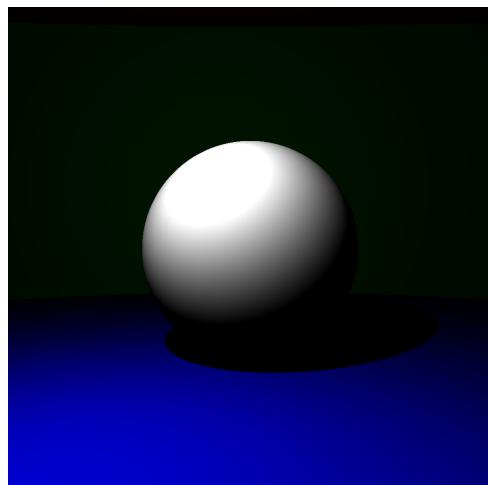


FIGURE 4 – Ajout de l'ombre portée.

2.5 Correction Gamma

Une fois la correction Gamma appliquée, nous devons augmenter la luminosité de nos lampes (FIGURE 5).

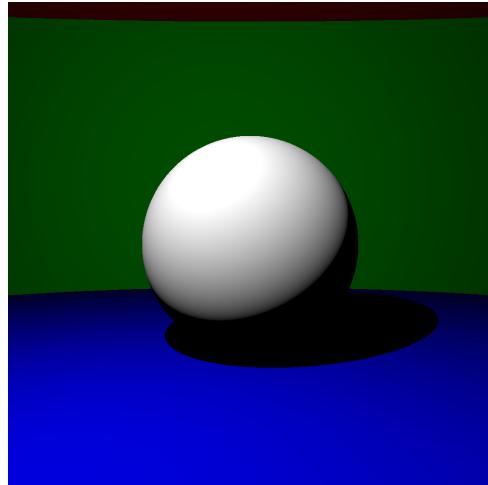


FIGURE 5 – Application de la correction Gamma avec $\gamma = 2.2$.

2.6 Plusieurs lampes

À partir de ce que nous avons déjà, il est très simple d'ajouter plusieurs lumières dans notre scène. Il suffit d'ajouter les intensités que chaque lampe produit (FIGURE 6).

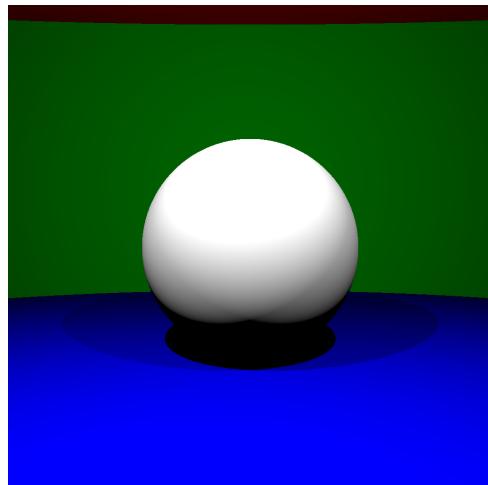


FIGURE 6 – Rendu avec 2 lampes.

3 Différents types de surfaces

3.1 Les surfaces spéculaires

Les surfaces spéculaires correspondent à des miroirs. Pour calculer l'intensité lumineuse en un point de la surface, nous devons faire rebondir le rayon correspondant et calculer récursivement l'intensité lumineuse du rayon rebondit (FIGURE 7 et FIGURE 8).

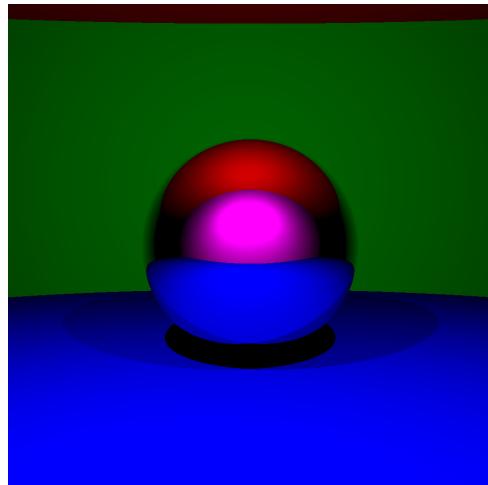


FIGURE 7 – Ici, seule la sphère est un miroir.

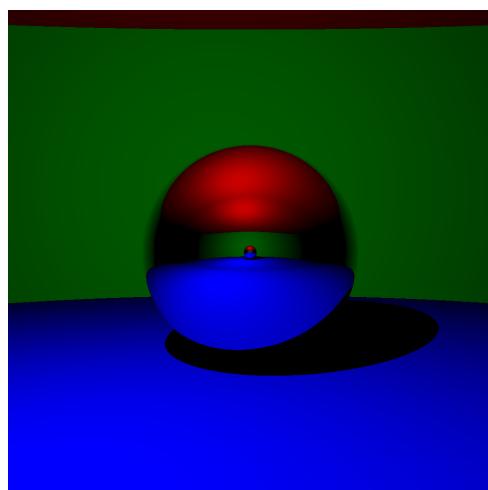


FIGURE 8 – Ici, la sphère ainsi que le mur de derrière sont des miroirs

3.2 Les surfaces transparentes

Ici, nous voulons gérer les objets transparents (qui dévient les rayons lumineux selon la loi de Snell-Descartes) (FIGURE 9 et FIGURE 10).

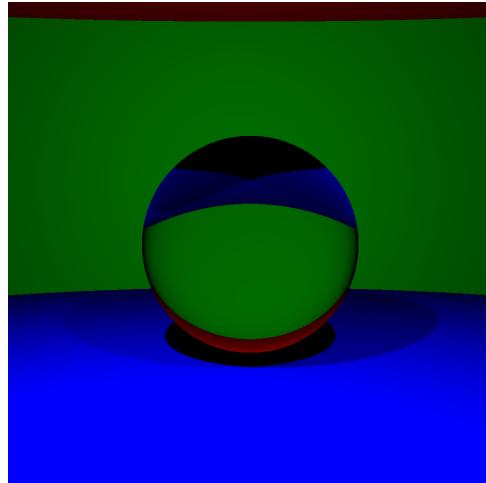


FIGURE 9 – Sphère transparente d’indice $n = 1.6$

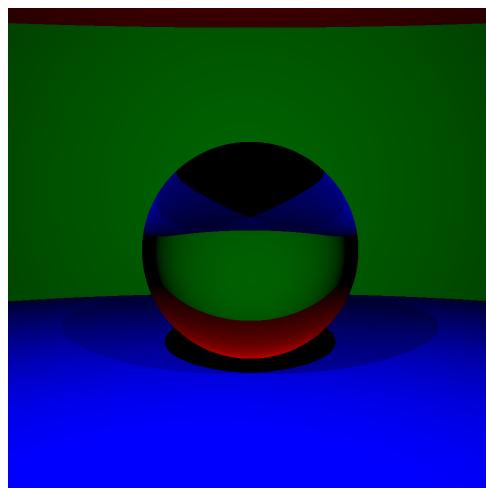


FIGURE 10 – Sphère transparente d’indice $n = 2.5$

Nous remarquons par ailleurs que le calcul de l’ombre portée n’est alors plus juste : un objet transparent dévie les rayons mais ne peut les bloquer totalement ! Nous ne corrigéons pas ce comportement pour l’instant.

3.3 Coefficients de Fresnel

Pour avoir un rendu plus réaliste, nous prenons en compte le fait que les matériaux transparents reflètent également une partie de la lumière. Les coefficients de Fresnel permettent de calculer les coefficients de réfraction et réflexion en fonction des indices de l’objet et de l’air, ainsi que de l’angle entre le rayon incident et la normale à la surface.

Dans la FIGURE 11, les deux disques blancs sont les réflexions d’une source de lumière étendue. La réflexion de gauche est directe, alors que la réflexion de droite correspond à une suite de réfraction, réflexion et réfraction : la réflexion se fait à l’intérieur de la sphère. Note : cette figure comporte également des sources de lumières étendues, décrites dans la section suivante.

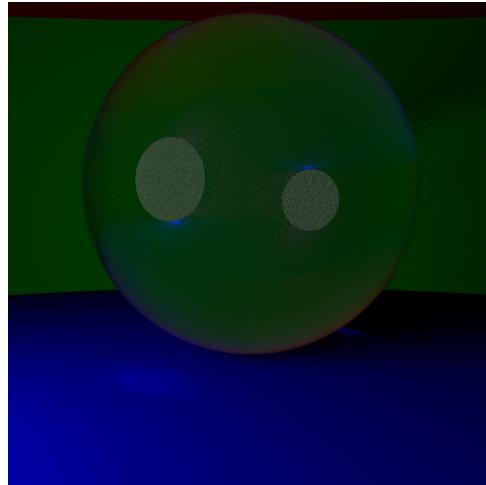


FIGURE 11 – Rendu d'une surface transparente avec coefficients de Fresnel

3.4 La BRDF de Phong

La BRDF de Phong permet de simuler des matériaux plus réalistes : des matériaux à mi-chemin entre diffus et des miroirs. Les rayons réfléchis sont, comme pour les matériaux diffus, aléatoires, mais au lieu d'avoir une distribution de probabilité uniforme dans le demi-hémisphère extérieur au matériau, une préférence est donnée aux rayons proches de la trajectoire des rayons d'un miroir.

On définit alors "l'exposant de Phong", qui décrit le rapprochement des rayons, ainsi qu'une couleur spéculaire, qui définira la teinte de réflexion des rayons (dans la plupart des cas, la couleur spéculaire ne sera qu'un multiple de la couleur diffuse du matériau).

Dans la FIGURE 12, on remarque la caustique sur le sol causé par la réflexion partielle des rayons.

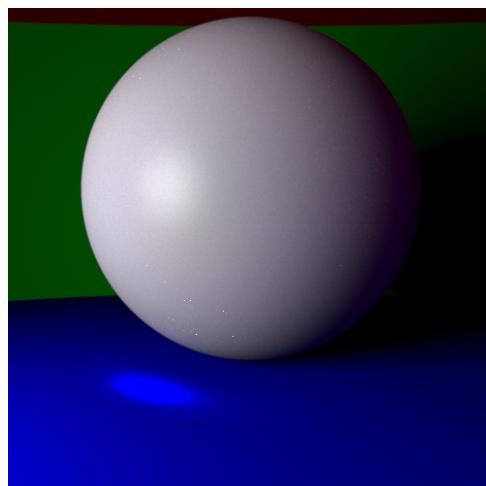


FIGURE 12 – Rendu d'une sphère avec BRDF de Phong (exposant 20,
Couleur : (1, 1, 1), Couleur spéculaire : (0.2, 0.2, 0.2))

4 Amélioration du rendu

4.1 Éclairage indirect

L'éclairage indirect consiste à faire rebondir les rayons de la scènes plusieurs fois, en prenant en compte la couleur des intersections de chaque rebond.

Dans la FIGURE 13 (sous-section suivante), on remarque que le sol est plus clair sous la sphère qu'aux autres endroits : les rayons lumineux rebondissent sur la sphère et atteignent le sol en plus grande quantité à cet endroit.

4.2 Sources de lumières étendues

Dans la FIGURE 13, on remarque que l'ombre est adoucie.

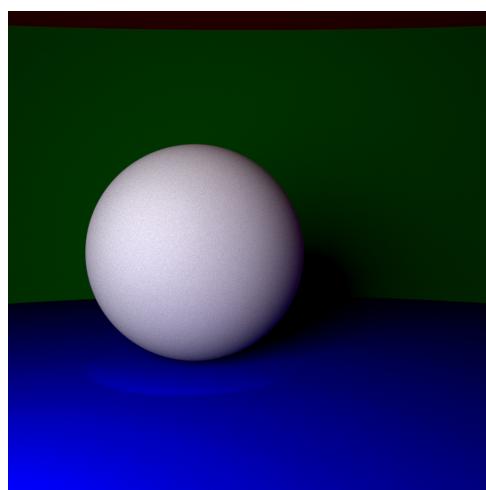


FIGURE 13 – Rendu d'une sphère sous une source de lumière étendue.

4.3 Depth of Field

Le Depth of Field permet de mieux modéliser la caméra. Dans la vie réelle, la caméra ne pourra rendre des images nettes que pour des objets étant à un intervalle de distance spécifique de la caméra. Des objets en dehors de cet intervalle seront flous.

Dans la FIGURE 14, j'ai placé deux sphères à des distances différentes de la caméra, et cette dernière est focalisée sur la sphère blanche. La sphère jaune, elle, est floue puisque trop près de la caméra. On remarque que le mur du fond est également flou, puisqu'il est trop loin de la caméra.

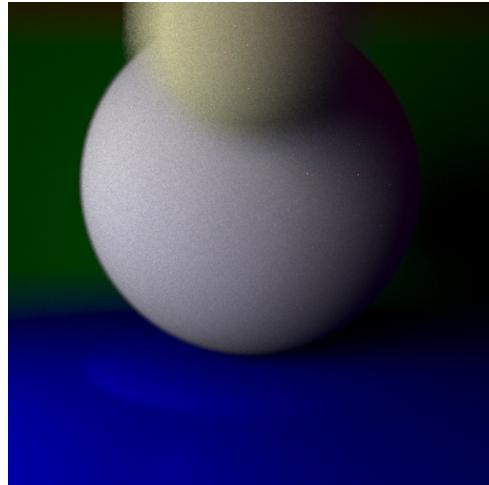


FIGURE 14 – Rendu de deux sphères de distance différentes, avec Depth of Field.

4.4 Monte-Carlo

Dans la FIGURE 15, on remarque la réduction du bruit de l'image par rapport aux premiers rendus.

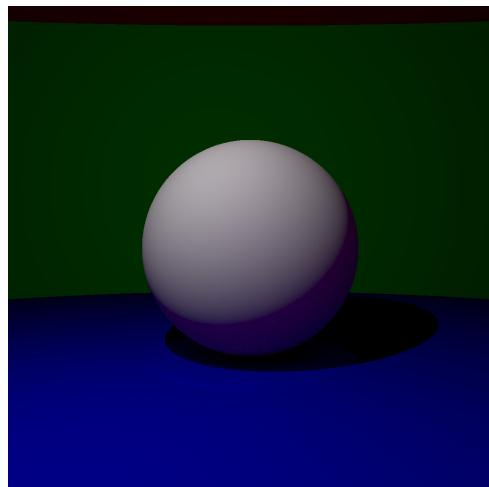


FIGURE 15 – Rendu d'une sphère avec Monte-Carlo.

4.5 Anti-aliasing

L'anti-aliasing peut être réalisé de différentes manières (post-processing, downscaling...), mais la plus adaptée à notre cas est dévier aléatoirement les rayons autour d'une gaussienne, puis de prendre une moyenne des couleurs des rayons obtenus.

Dans la FIGURE 16, nous voyons la différence entre les deux rendus, avec et sans anti-aliasing. Les bords de la sphère ont l'effet classique d'escalier dans l'image de droite, qui n'est pas présent dans l'image de gauche.

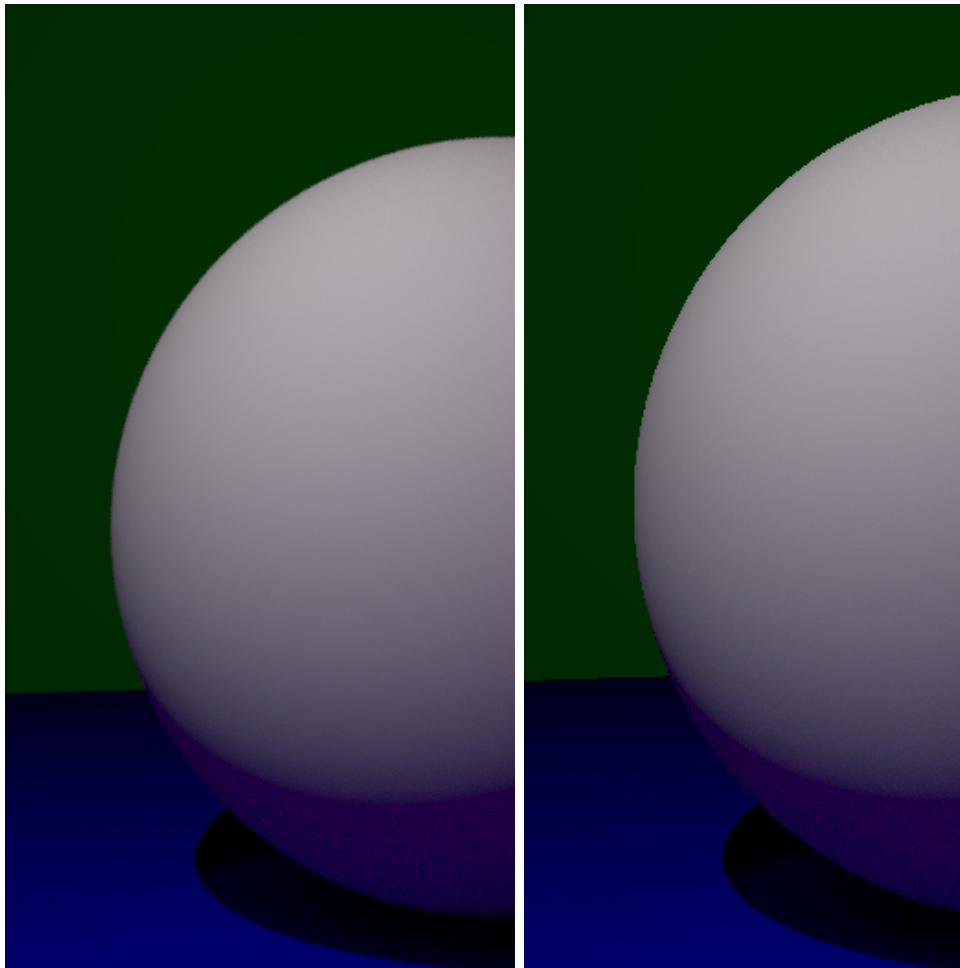


FIGURE 16 – Rendu avec AA (gauche) et sans (droite).

5 Les maillages

5.1 Structures de données et parsing

Les fichiers objets (*.obj) permettent de représenter un modèle 3D en décrivant les sommets et triangle du maillage.

D'autres informations, comme les normales à chaque sommet ou les coordonnées de textures sont également présentent dans le fichier du modèle.

J'ai récupérer un modèle de T-Rex (<https://free3d.com/3d-model/t-rex-by-zino-25516.html> - licence pour utilisation personnelle) comprenant également des images de textures Bitmap.

Le parsage du fichier est basique, mais il m'a fallu faire attention, certaines descriptions de faces du maillage ne comprennent pas d'information de texture. J'ai défini la couleur pour ces faces à noir.

5.2 Affichage du maillage

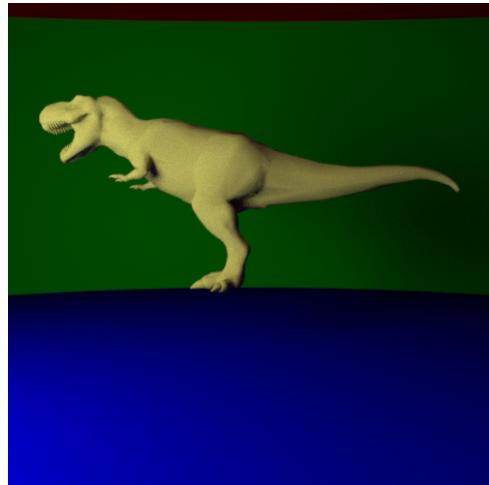


FIGURE 17 – Rendu du maillage du T-Rex avec un matériau diffus.

5.3 Structures accélératrices

Afin d'éviter de réaliser les tests d'intersection avec l'ensemble des triangles du maillage (il y en a plus de 10 000 dans le modèle du T-Rex !), nous utilisons le concept de boîte englobante (bounding-box, de type AABB puisque les axes des boîtes seront les axes (x, y, z) de l'espace).

Ainsi, nous créons un arbre de boîtes récursivement (chaque boîte possède jusqu'à 8 boîtes filles). Nous ne créons pas de sous-boîte si le nombre de triangles de la boîte mère est inférieur à une valeur seuil afin de ne pas en créer trop. Ce seuil est pris à 20, et donne de bons résultats en mémoire et temps d'exécution.

Dans mon code, la classe représentant le maillage, `Mesh`, hérite de `Box`. Ainsi, dans la routine d'intersection d'une boîte, nous testons au préalable si le rayon intersecte la boîte. Si ce n'est pas le cas, nous n'avons pas besoin d'aller plus loin et gagnons beaucoup de temps.

Si c'est le cas, nous testons récursivement les boîtes filles (ou les triangles si elle ne possède pas de boîte fille).

Pour les routines d'intersection avec les boîtes (Rayon-Boîte et Triangle-Boîte), j'ai repris les snippets de code donnés dans le document support au cours, qui avaient de bien meilleures performances que ce que j'ai tenté de réaliser (je suppose que je faisais des tests inutiles supplémentaires).

5.4 Gestion des textures

5.4.1 Textures procédurales

Les textures procédurales sont assez simple à gérer. Il suffit de créer une fonction qui retourne la couleur voulue de notre texture en fonction des paramètres voulus (normale, position dans l'espace, ...).

J'ai choisis de ne pas les gérer pour les sphères, même si les changements à effectuer sont minimaux.

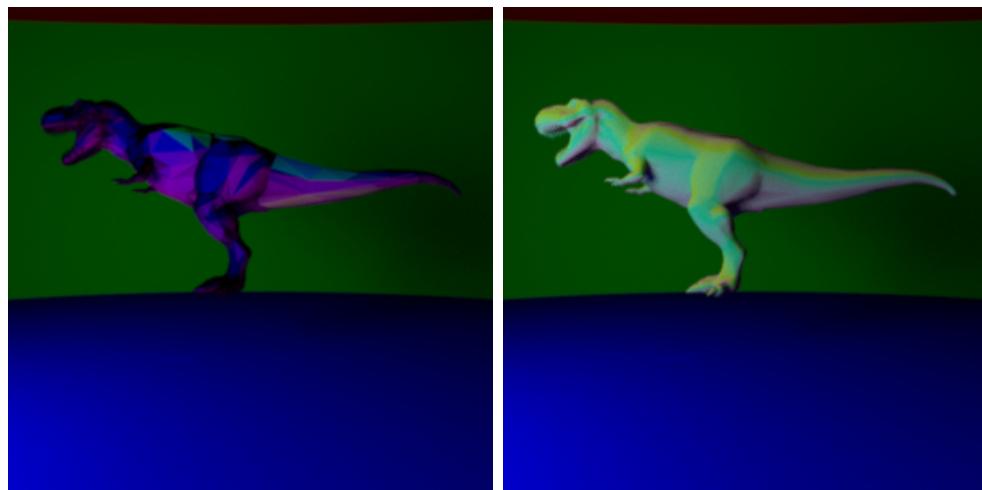


FIGURE 18 – Rendu du maillage avec texture procédurale.

5.4.2 Textures Bitmap

Les textures Bitmap, elles, sont plus complexes. Il faut récupérer l'image BMP de la texture à appliquer, puis déterminée les coordonnées de textures correspondants au point d'intersection, en prenant le barycentre des coordonnées de texture des 3 sommets du triangle du maillage intersecté par notre rayon.

Je n'ai ainsi pas géré les textures Bitmap pour les sphères.

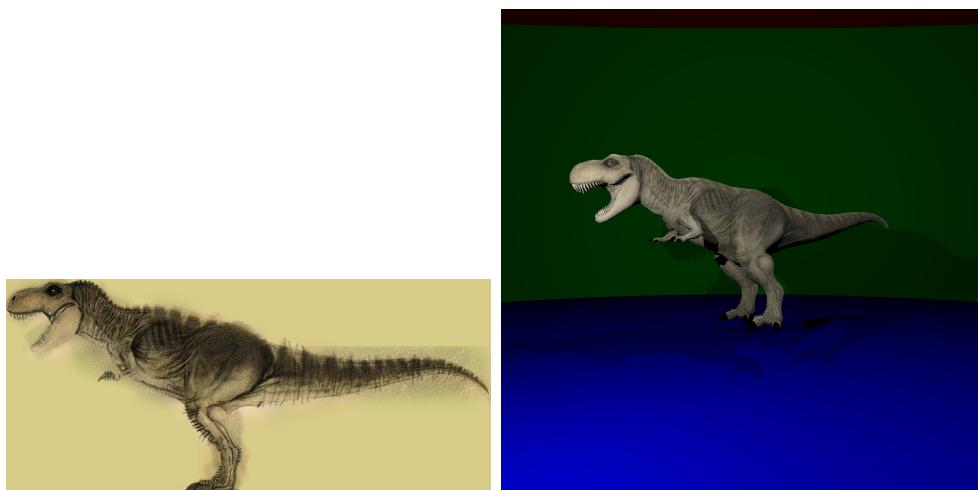


FIGURE 19 – Rendu du maillage avec texture bitmap.

6 Animation

Un fichier .gif est disponible dans le dossier du projet pour visualiser le rendu réalisé avec animation des objets.

6.1 De la caméra

Afin de déplacer la caméra dans notre scène, il nous suffit d'appliquer la transformation voulue aux rayons envoyés.

6.2 Des objets de la scène

Le déplacement des différents objets de la scène (lumière, sphères et maillages) sont plus complexes, mais reposent sur le même principe que précédemment : nous préférons bouger les rayons que les objets eux-mêmes !

6.3 Flou de mouvement

Une fois les animations générées, il est possible d'ajouter un effet de flou de mouvement à nos rendus. Pour cela, au lieu d'avoir un temps unique pour les rayons d'une même image, nous devons choisir ce temps aléatoirement dans un intervalle défini par la durée d'ouverture du diaphragme.

Il serait également possible de modéliser une caméra avec "rolling-shutter", et comparer avec les résultats de vidéos réelles, mais je n'ai pas implémenté cette modélisation.

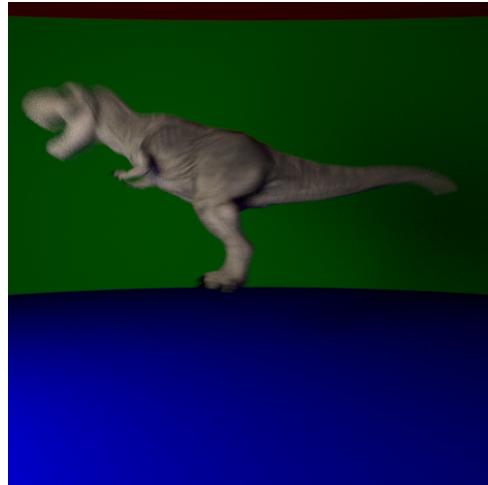


FIGURE 20 – Rendu du maillage flou de mouvement lors d'une rotation autour d'un axe vertical.

Conclusion

Finalement, nous avons pu voir au travers de ce projet les grands principes du Ray-tracing.

Les principales améliorations que j'aurai pu implémenter sont liées :

- À l'optimisation des temps de calculs, en profilant mon programme afin de déterminer les points lents de ce dernier. Un style de programmation plus proche du C, avec par exemple de nombreuses macros permettrait sans doute de gagner considérablement en performances, mais j'ai préféré prioriser la clarté du code.
- Aux fonctionnalités gérées, comme les milieux participants, qui sont très intéressants mais plus complexes à implémenter.