



ÉCOLE  
**CENTRALE**LYON

## ÉCOLE CENTRALE DE LYON

MOS 2.2  
INFORMATIQUE GRAPHIQUE

# Raytracing

*Étudiant :*  
Fabien LECLERC

*Enseignant :*  
Nicolas BONEEL

## Table des matières

<b>Introduction</b>	<b>2</b>
<b>1 Capacité du Raytracer</b>	<b>3</b>
1.1 Affichage de sphères colorées avec éclairage direct . . . . .	3
1.2 Correction gamma . . . . .	3
1.3 Ombre portées . . . . .	3
1.4 Éclairage indirect . . . . .	5
1.5 Ombres douces . . . . .	7
1.6 Profondeur de champ . . . . .	7
1.7 Affichage de triangles . . . . .	8
1.8 Maillage . . . . .	8

## Introduction

Ce document présente les capacités du raytracer développé en c++ dans le cadre du cours MOS 2.2 Informatique Graphique.

# 1 Capacité du Raytracer

## 1.1 Affichage de sphères colorées avec éclairage direct

Le premier objet élémentaire affiché par le moteur de raytracing sera une sphère. Le programme calculera les intersections entre les rayons et les points de la sphère. Plutôt qu'un affichage binaire de l'intersection, la lumière ambiante est prise en compte. L'intensité du pixel de la sphère va être calculée en fonction de l'intensité de la lumière et de l'angle d'incidence de la lumière. Une lumière rasante aura une contribution moindre qu'un rayon de lumière avec une incidence normale. Pour cela on envoie maintenant un deuxième rayon vers la source de lumière. Ce rayon part du point d'intersection entre le premier rayon et la sphère. L'intensité est calculée de la manière suivante :

$$\text{Intensité}_{pixel} = \text{Intensité}_{lumière} * \max(0, \vec{l} \cdot \vec{n}) / d^2 \quad (1)$$

Pour donner des couleurs aux objets affichés on implémente l'albédo. L'albedo correspond à un vecteur RVB (Rouge,Vert,Bleu)  $\rho \in [0, 1]^3$ . Chaque composante du vecteur correspond à l'intensité réfléchie par l'objet pour chaque sous-pixel. Une composante totalement absorbée et non réfléchie aura une valeur à 0, et à l'inverse 1 si cette dernière est totalement réfléchie par l'objet. On peut donc créer des objets de toutes les couleurs en combinant différentes valeurs entre 0 et 1 pour chaque composante.

Les sphères à afficher sont dans un objet *Scène*. Pour chaque rayon envoyé on calculera l'intersection avec les différentes sphères de la scène et on retiendra la sphère la plus proche pour l'intensité et la couleur du pixel. En effet seule la sphère la plus proche de la caméra apparaît, les autres sont cachées derrière.

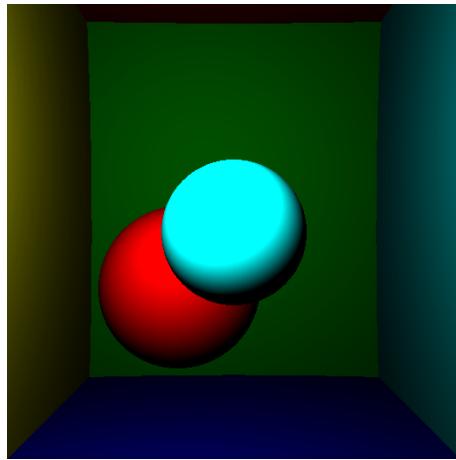


FIGURE 1 – Affichage d'une scène contenant plusieurs sphères de différentes couleurs

L'image ci-dessus a été obtenue avec un temps de calcul de 250ms environ. On y observe également que l'on a pu réaliser des "murs" pour la scène grâce à des sphères de rayon très grand.

## 1.2 Correction gamma

Les images obtenues jusqu'à présent ont un rendu assez peu réaliste. Certaines zones ont une intensité lumineuse très élevée dans laquelle on ne distingue rien, et dans les zones peu éclairées on n'observe quasiment rien. Cela vient du fait que les écrans ont un facteur gamma qui élève l'intensité de la lumière à la puissance 2.2. Pour compenser ce phénomène les valeurs d'intensités calculées se verront appliquées une correction gamma, c'est à dire qu'elles seront élevées à la puissance  $\frac{1}{2.2}$ .

## 1.3 Ombre portées

Pour faire afficher les ombres portées, il faut regarder si un objet se trouve entre le point d'intersection rayon-scène calculé précédemment et la source de lumière. Pour faire cela, un rayon secondaire est envoyé depuis le point d'intersection avec l'objet vers la source de lumière et on cherchera à savoir si ce rayon a une intersection avec d'autres objets de la scène. Si on trouve une intersection, il suffira de comparer sa distance à celle de la source de lumière pour savoir si l'objet intersecté bloque le passage de la lumière.

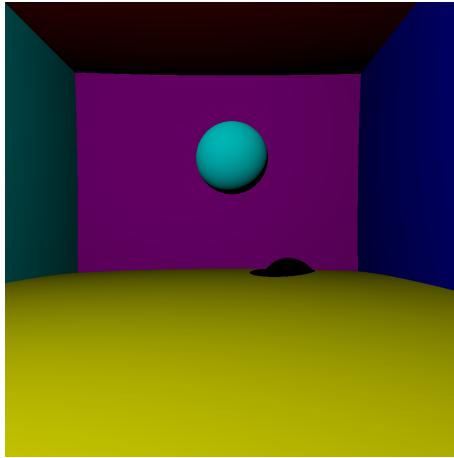


FIGURE 2 – Ombres portées et correction gamma sur la scène affichée

L'ombre est bien projetée sur les murs de la scène de manière réaliste. Même dans les angles comme on peut l'observer sur la figure 2. De plus la correction gamma rend la luminance de la scène plus homogène avec plus de détails dans les zones d'ombres. Cette image a été obtenue en 385ms environ. L'augmentation du temps de calcul vient principalement du calcul des ombres qui nécessite l'envoi d'un rayon de test entre le point d'intersection et la source de lumière.

#### Surfaces spéculaires

Les sphères affichées jusqu'à présent sont des objets diffus. Pour afficher des objets miroirs il faut faire un calcul itératif de raytracing avec les rayons des directions réfléchies à chaque objet réfléchissant rencontré. La valeur d'un pixel d'un objet miroir rencontré par le rayon sera celle obtenue par le rayon partant de l'objet miroir dans la direction réfléchie. Ce calcul sera donc réalisé récursivement avec comme condition d'arrêt, l'intersection d'un rayon avec une surface diffuse. Il faut cependant fixer un nombre de rebond maximal, correspondant au nombre d'objets réfléchissant par exemple, afin de s'assurer de la terminaison de l'appel récursif.

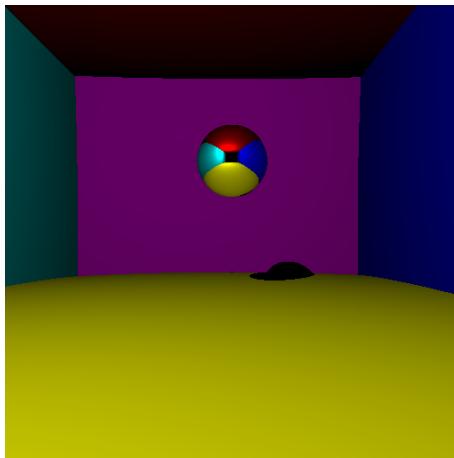


FIGURE 3 – Sphère miroir qui reflète les murs de la scène

Le temps d'exécution ne varie quasiment pas avec cet effet miroir pour une seule sphère. Cependant si l'on commence à augmenter le nombre de miroir le temps de calcul augmente. Par exemple l'image ci-dessous a été obtenue en 530 ms environ en raison d'un nombre de rebond plus important pour les rayons :

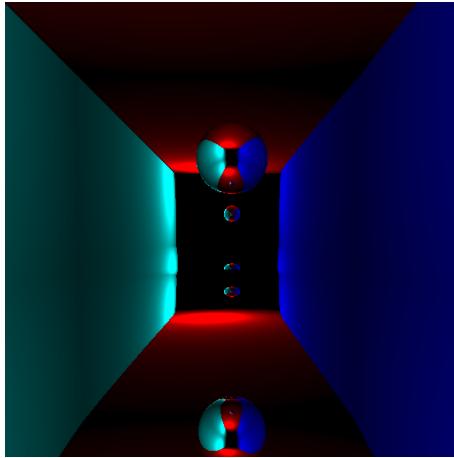


FIGURE 4 – Sphère miroir dans une scène avec deux murs miroirs

#### 1.4 Éclairage indirect

L'éclairage secondaire va maintenant être pris en compte dans le rendu. Lorsque la lumière rebondit sur un objet, celui-ci va agir comme une source de lumière secondaire. Pour calculer cette contribution indirecte, un second rayon est envoyé dans une direction aléatoire proportionnelle à l'angle solide d'incidence avec la surface. La contribution totale sera approximée par échantillonage et intégration de Monte Carlo. Un certain nombre de rayon fixé est envoyé.

Le modèle rendu calcule maintenant l'équation du rendu :

$$L_0(x, w_0, \lambda) = L_e(x, \vec{w}_0, \lambda) + \int_{S^2} L_i(x, \vec{w}_i, \lambda) f(x, \vec{w}_i, \vec{w}_0, \lambda) \langle \vec{w}_i, \vec{N} \rangle d\vec{w}_i \quad (2)$$

Cette intégrale est donc calculée récursivement avec des rayons envoyés dans des directions  $\vec{w}_i$  aléatoires selon la BRDF  $f$  qui émettent une lumière  $L_i$ . Dans cette intégrale récursive, la probabilité de tomber sur la source de lumière ponctuelle est nulle, d'autant plus pour un faible nombre de rayons. Pour prendre en compte sa contribution et ne pas avoir une image toute noire, un rayon sera toujours envoyé en direction de cette source de lumière en plus des rayons envoyés aléatoirement.



FIGURE 5 – Rendu avec éclairage indirect par méthode de Monte Carlo avec 8 rayons par pixel (30s)

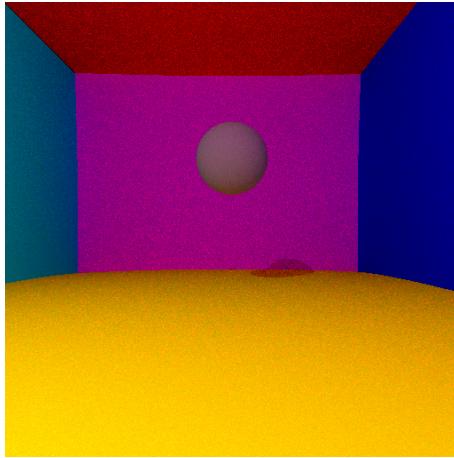


FIGURE 6 – Rendu avec éclairage indirect par méthode de Monte Carlo avec 100 rayons par pixel (400s)

Le temps de calcul devient très long pour un grand nombre de rayons utilisés. Or pour réduire le bruit, il faut utiliser le plus grand nombre de rayons possible. Étant donné que chaque pixel peut être calculé indépendamment, les calculs peuvent être réalisés en parallèle. En parallélisant le calcul on obtient des temps d'exécution de l'ordre de 5 secondes pour 8 rayons par pixel et 70 secondes pour 100 rayons parpixel, soit un temps de calcul réduit d'un facteur 6.

#### Anti-aliasing

L'aliasing ou "crénelage" est un phénomène qui apparaît lorsqu'on considère uniquement un rayon passant par le centre du pixel. En effet si la sphère n'est présente que sur une partie du pixel, elle n'apparaîtra que si le centre du pixel recouvre la sphère. Cela donne des contours de sphères pixellisés comme le montre la figure 7. Pour corriger ce phénomène une technique d'anti-aliasing consiste à envoyer plusieurs rayons pour un pixel dans des directions légèrement différentes qui ne passent pas que par le centre. Cependant les rayons du centre du pixel auront toujours un poids plus importants que ceux des bords. Plutôt que de pondérer les intensités lumineuses obtenues par chaque rayon en fonction de sa position sur le pixel, on réalise un échantillonage gaussien centré sur le centre du pixel. Cela aura pour effet de produire plus d'échantillons au centre que sur les bords du pixel. L'algorithme de Box-Muller permet de réaliser ceci. Il suffit d'ajouter ce calcul de variation aléatoire de direction pour chaque rayon dans la méthode de Monte-Carlo précédemment implémentée. Pour constater l'effet de l'anti-aliasing, deux simulations sont réalisées avec 100 rayons : une avec anti-aliasing et l'autre sans

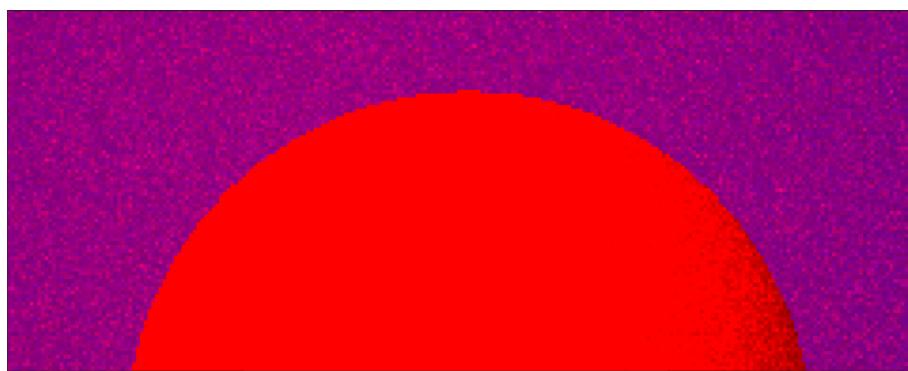


FIGURE 7 – Crénelage observé sur le contour de la sphère sans anti-aliasing

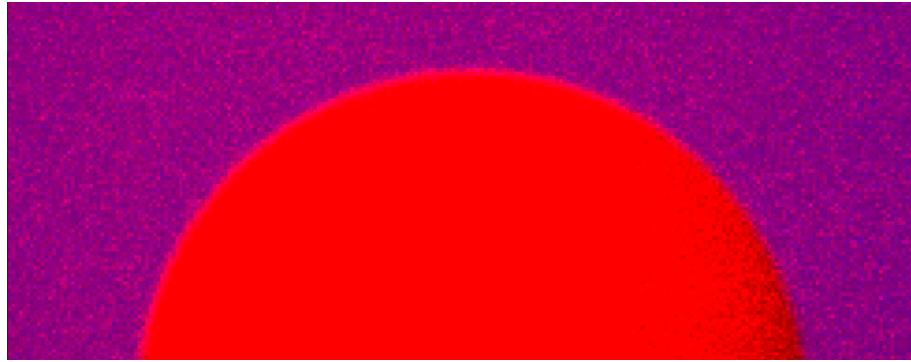
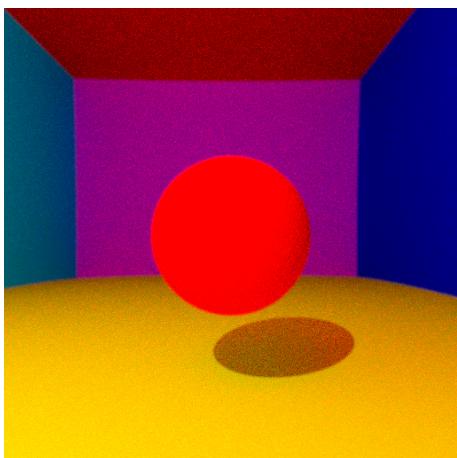


FIGURE 8 – Contour plus flou et moins pixellisé grâce à l’anti-aliasing

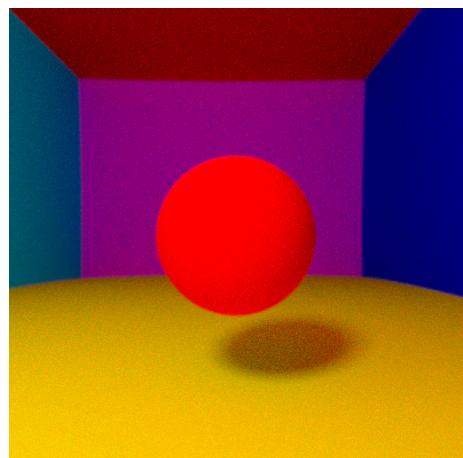
Les deux simulations ont un temps d’exécution similaire de 84 secondes. En effet le même nombre de rayons sont envoyés dans les deux. On observe que le filtre anti-aliasing a bien l’effet attendu : le contour de la sphère est moins pixellisé.

## 1.5 Ombres douces

On considère maintenant des sources de lumières diffuses, c’est à dire des sphères. La contribution de l’éclairage direct sera calculée en envoyant un rayon dans une direction aléatoire autour de la direction entre le point d’intersection et le centre de la lumière.



(a) Lumière ponctuelle



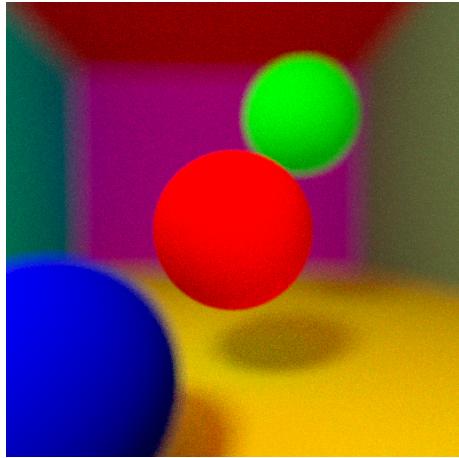
(b) Lumière diffuse

FIGURE 9 – Comparaison des ombres en lumières ponctuelle et diffuse. Rendus réalisés avec 100 rayons par pixels

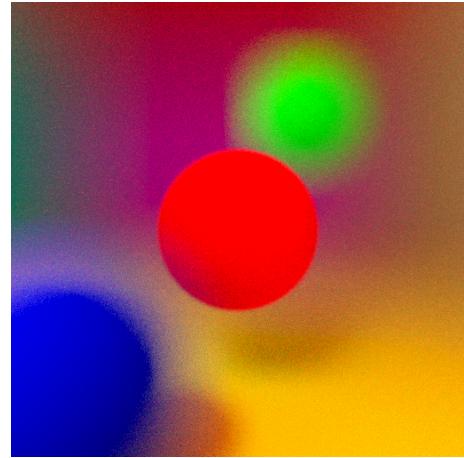
Le rendu ci-dessus permet d’observer que le contour des ombres est adouci en prenant en compte une source de lumière diffuse. Cela vient du fait que l’objet peut bloquer seulement une partie de la lumière de la source étendue alors qu’avec une source ponctuel le passage de la lumière était binaire. Ce calcul par génération de rayons aléatoires a ajouté de la complexité au calcul. L’image ombre douce a été obtenue en 135 secondes contre 80 secondes en lumière ponctuelle.

## 1.6 Profondeur de champ

Sur une vraie caméra on peut observer de la profondeur de champ en raison de l’ouverture du diaphragme. Plus le diaphragme est ouvert et plus les points éloignés du plan focal objet de la lentille forment de grandes tâches et donc la profondeur de champs diminue. Pour simuler le diaphragme dans le modèle de raytracing, plusieurs rayons sont envoyés dans des directions qui recouvrent le diaphragme supposé carré pour des raisons de simplifications. Une distance de mise au point à laquelle les objets seront nets est également définie.



(a) Ouverture du diaphragme = 1



(b) Ouverture du diaphragme = 5

FIGURE 10 – Influence de l'ouverture du diaphragme sur la profondeur de champ. Rendus réalisés avec 100 rayons par pixels

Tout comme sur un appareil photo, une plus grande ouverture entraîne une plus petite profondeur de champ et donc plus de flou devant et derrière la zone de mise au point. Le temps de calcul des rendus reste autour de 130 secondes. En effet on a simplement décalé les rayons déjà envoyés avec l'anti-aliasing.

### 1.7 Affichage de triangles

L'implémentation d'un objet triangle avec les 3 coordonnées des sommets permet d'afficher des triangle grâce à un calcul d'intersection basé sur les coordonnées barycentriques :

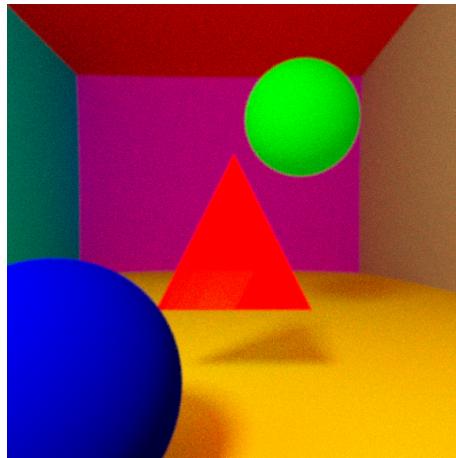


FIGURE 11 – Affichage d'un triangle

Les fonctionnalités précédentes d'éclairage indirect et de mise au point fonctionnent toujours comme on peut le voir sur l'image.

### 1.8 Maillage

Les maillages permettent d'afficher des objets plus complexes. Des fichiers obj qui contiennent les coordonnées de milliers de triangles sont utilisés pour représenter des formes diverses. Étant donné le grand nombre de triangle, on économise des calculs en ne testant les intersections avec les triangles du maillage que pour les rayons intersectant une boîte englobante du maillage calculée à partir des coordonnées maximales et minimales des triangles selon les différents axes.

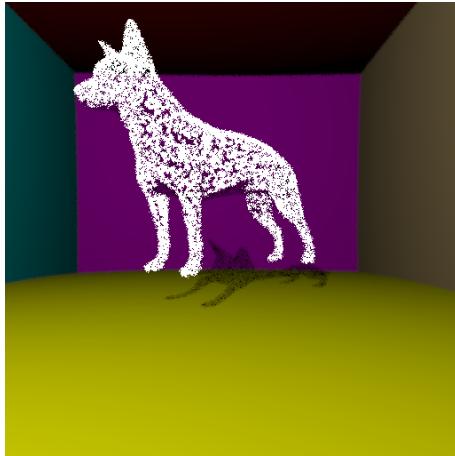


FIGURE 12 – Affichage d'un maillage

Le maillage affiché ci-dessus a été réalisé avec 8 rayons. L'éclairage indirect a été désactivé pour réduire le temps de calcul et la taille de l'image a été réduite (1024x1024 à 512x512) 8 rayons ont été utilisés pour l'antialiasing. L'image a nécessité près d'une heure de calcul. Cela montre que le grand nombre d'objets triangles dans le maillage alourdi considérablement les calculs des intersections, même avec la boîte englobante. Une amélioration serait de créer des boîtes englobantes hiérarchique sur le maillage afin de tester successivement des boîtes plutôt qu'un grand nombre de triangles.