

**CENTRALE  
LYON**

ÉCOLE CENTRALE LYON

INFORMATIQUE GRAPHIQUE  
RAPPORT

---

# Moteur graphique en C++

---

*Élèves :*  
Damien DELPRAT

*Enseignant :*  
Nicolas BONNEEL

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Intersections rayon sphère(s)</b>	<b>2</b>
<b>3</b>	<b>Ombres portées, surfaces spéculaires et transparentes</b>	<b>2</b>
3.1	Ombres portées . . . . .	2
3.2	Surfaces spéculaires . . . . .	2
3.3	Surfaces transparentes . . . . .	3
<b>4</b>	<b>Intégration de Monte-Carlo et éclairage indirect</b>	<b>3</b>
<b>5</b>	<b>Anti-aliasing et ombres douces</b>	<b>5</b>
<b>6</b>	<b>Intersections rayon plan(s) et gestion des maillages et textures</b>	<b>6</b>

# 1 Introduction

Ce rapport décrit la réalisation d'un moteur graphique basé sur le ray tracing en C++. Le code associé est disponible sur Github à l'adresse suivante : <https://github.com/ddelprat/3D-Engine>.

La réalisation du code se divise en 5 étapes :

- Calcul d'intersection rayon sphère(s)
- Ombres portées, surfaces spéculaires et transparentes.
- Intégration de Monte-Carlo et éclairage indirect
- Anti-Aliasing et ombres douces
- Intersections rayon plan(s) et gestion des maillages et textures

Je n'ai en revanche pas réussi à mettre en place le parallélisme des calculs donc les temps de calculs peuvent être un peu longs parfois.

## 2 Intersections rayon sphère(s)

La première étape est de calculer l'intersection d'un rayon projeter depuis la caméra vers une sphère. L'intersection est calculée en résolvant l'équation :

$$t^2 + 2t \langle u, OC \rangle + OC^2 R^2 = 0$$

Ensuite, pour connaître la couleur du pixel, pour rayon qui intersecte une sphère en un point P, on projette un rayon de ce point vers la source lumineuse et on utilise la formule :

$$\text{couleur} = \frac{\rho}{\pi} \frac{I \langle \vec{N}, \vec{L} \rangle}{d^2}$$

avec  $I$  l'intensité de la source lumineuse,  $\vec{N}$  la normale à la sphère au point d'intersection,  $\vec{L}$  la direction du point d'intersection vers la source lumineuse et  $d$  la distance du point avec la source.

## 3 Ombres portées, surfaces spéculaires et transparentes

### 3.1 Ombres portées

Lorsqu'un rayon touche une sphère, il est nécessaire de savoir si le point d'intersection est éclairé par une source lumineuse ou si il est dans l'ombre ou masqué par un autre objet entre lui et la source de lumière. Pour le savoir, lorsqu'un rayon intersecte une sphère en un point P, on vérifie que le rayon projeté vers la source lumineuse n'intersecte aucune autre sphère à une distance inférieure. Si il intersecte une sphère à une distance inférieure, le point est considéré à l'ombre et n'est donc pas éclairé (noir).

### 3.2 Surfaces spéculaires

Dans le cas d'une surface avec des propriétés réfléchissantes, on cherche à déterminer le rayon réfléchi sur la sphère issu du rayon incident qui peut être calculer grâce à la

normale locale de la sphère. On calcule ensuite la couleur obtenu par intersection avec le rayon réfléchi.

### 3.3 Surfaces transparentes

Dans le cas d'une surface transparente, on détermine la déviation du rayon pénétrant dans la sphère par diffraction grâce aux lois de Descartes. On calcule ensuite la proportion du rayon qui est réfléchie et la proportion traversante. Celles-ci régiront la probabilité d'un rayon de traverser ou réfléchir sur la surface pour chaque simulation de Monte-Carlo.

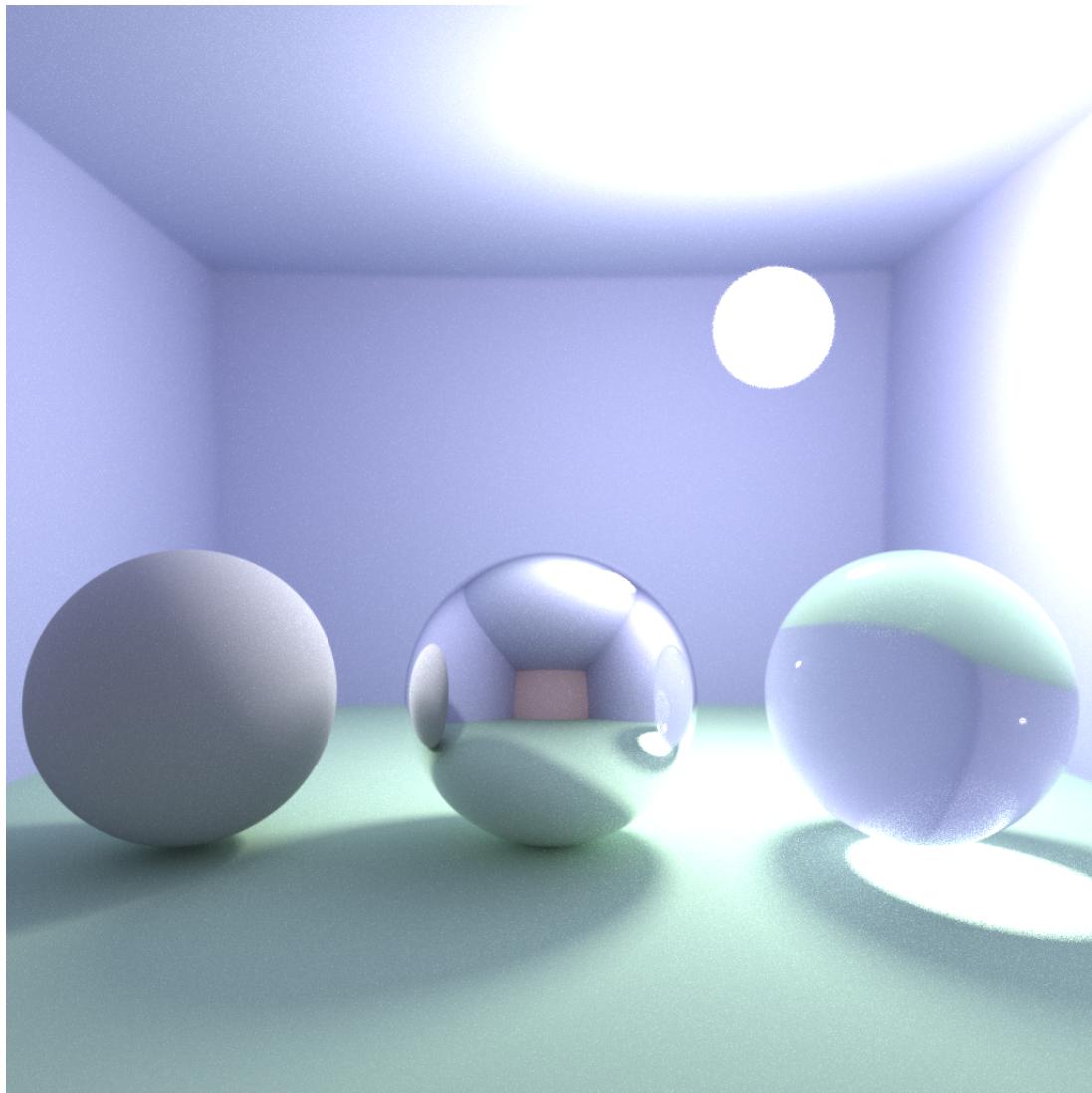


FIGURE 1 – Image générée en 1024x1024 avec 1024 simulations et 10 rebonds

## 4 Intégration de Monte-Carlo et éclairage indirect

Afin de simuler le reflet de la lumière sur les surfaces diffuses, on décompose la lumière en  $\text{éclairage} = \text{éclairage\_direct} + \text{éclairage\_indirect}$ . L'éclairage indirect est calculé en calculant le rayon réfléchi sur la surface et en appliquant la fonction permettant de calculer

la couleur de manière récursive. Le nombre de rebonds à calculer est un paramètre de la simulation.

Pour rendre le rendu plus réaliste, on ajoute une variation aléatoire à l'angle de réflexion autour de la valeur exacte calculée en utilisant une BRDF. Puis on utilise la Méthode de Monte Carlo pour moyenner, pour chaque pixel, la couleur obtenue par simulation. Le nombre de simulations effectuer est un paramètre propre à une image.

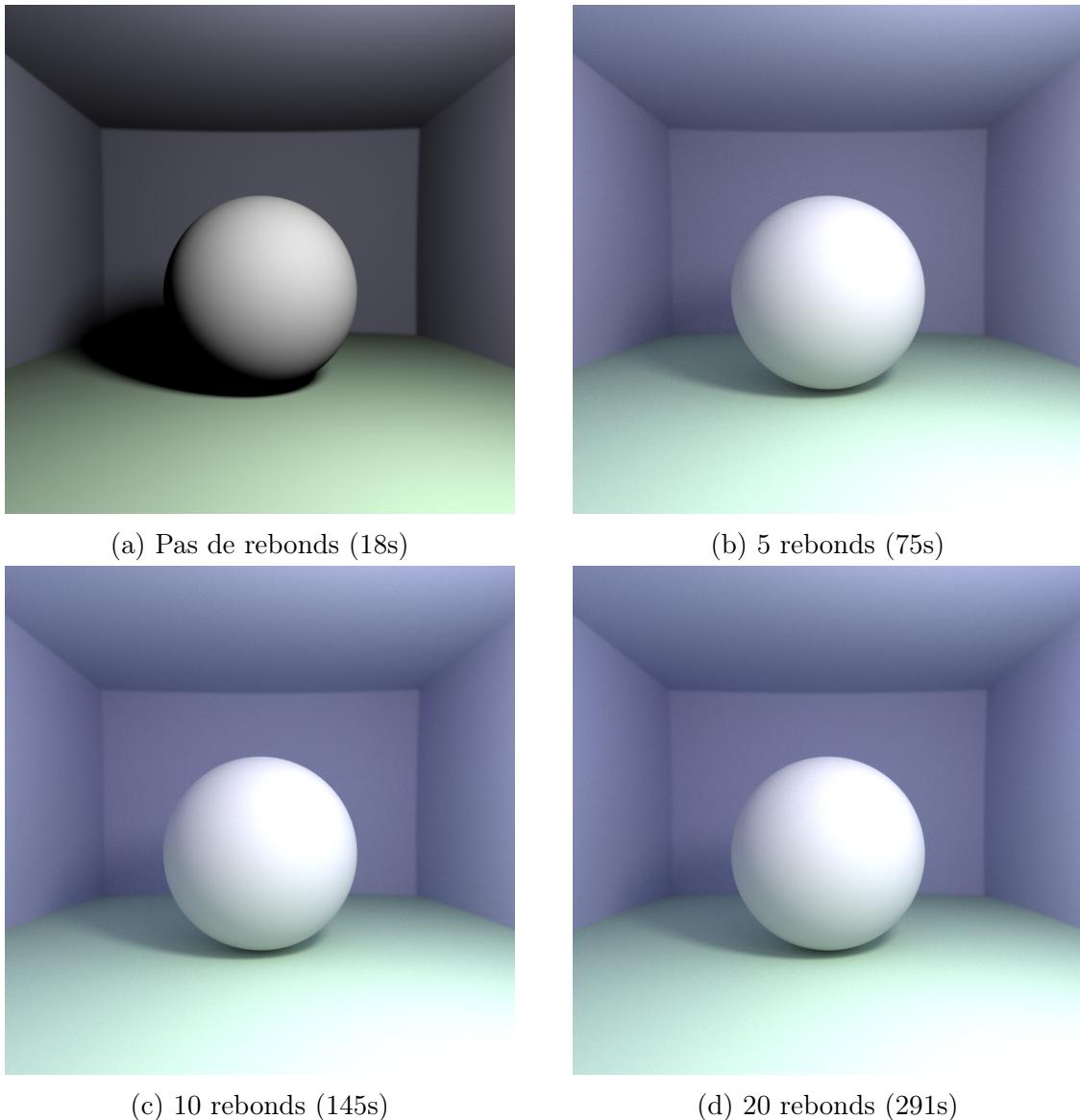
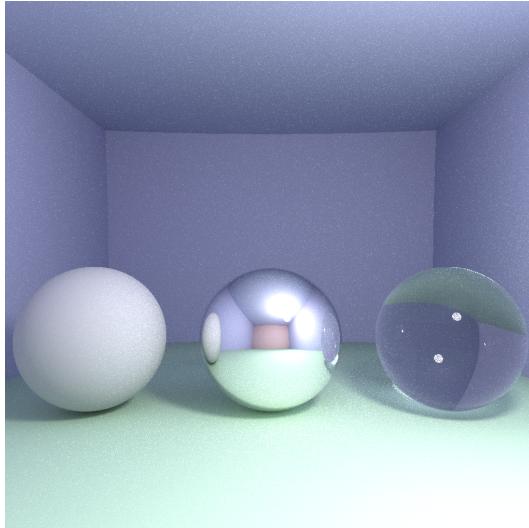


FIGURE 2 – Génération de la scène en 1024x1024 avec 256 simulations

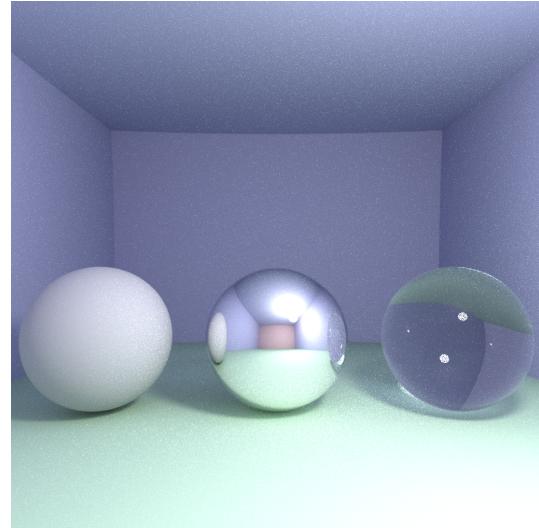
On remarque qu'avec l'augmentation du nombre de rebonds, la luminosité de la scène augmente mais converge pour des valeurs élevées (20 ici). On remarque également que la boule blanche prend un peu la couleur des murs et que l'ombre est moins marquée sauf juste en dessous de la sphère.

## 5 Anti-aliasing et ombres douces

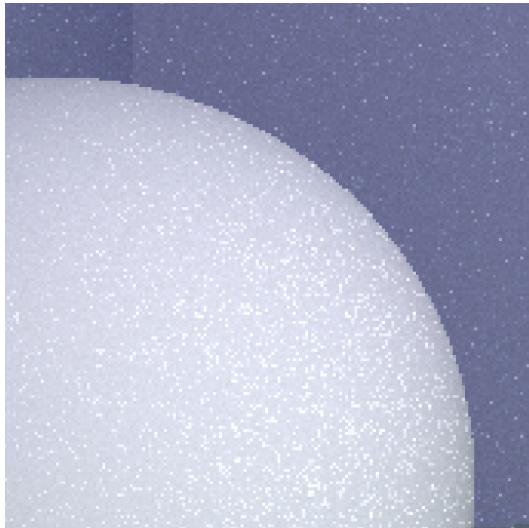
L’anti-aliasing consiste à rendre aléatoire la position de départ du rayon sur chaque pixel pour chaque simulation. Cela permet d’avoir des bordures moins marquées sur les objets observés.



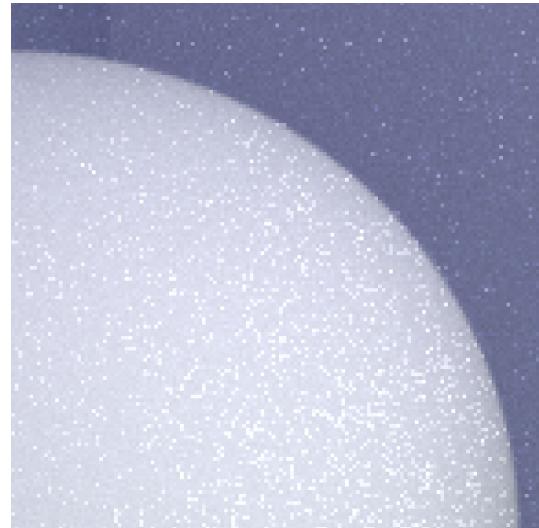
(a) Sans antialiasing (115s)



(b) Avec antialiasing (106s)



(c) Sans antialiasing (zoomé)

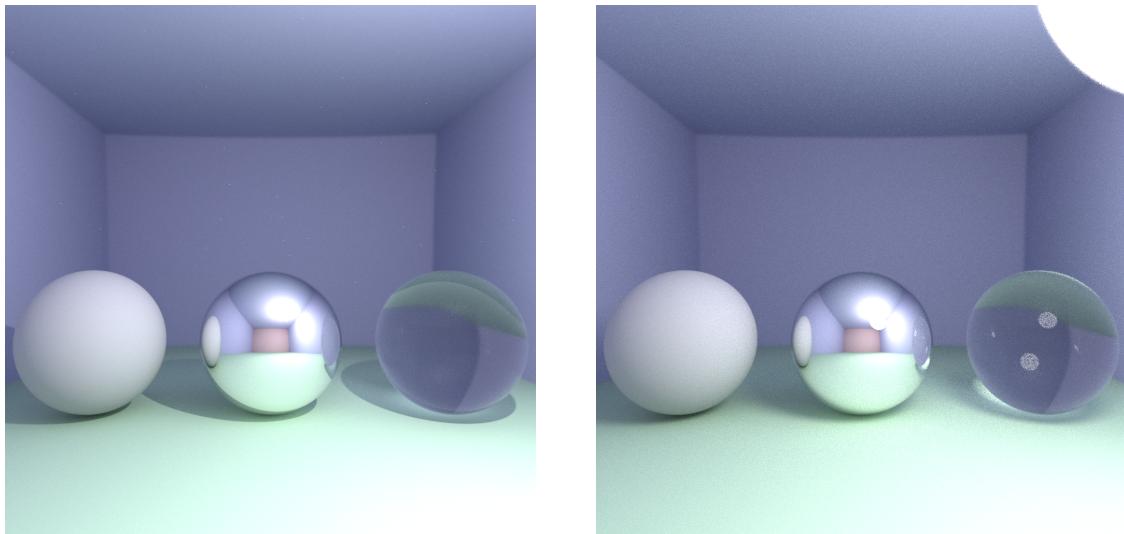


(d) Avec antialiasing (zoomé)

FIGURE 3 – Images générées en 1024x1024 avec 256 simulations et 10 rebonds

On remarque qu’avec l’antialiasing, les contours sont moins pixelisés et légèrement flous ce qui leur donne un effet plus naturel.

Les ombres douces sont calculées en utilisant pour source lumineuse une sphère lumineuse plutôt qu’un simple point. Ainsi, pour calculer la couleur d’un pixel, après avoir calculé le premier point d’intersection de ce point avec un objet, on échantillonne un point sur la sphère lumineuse et on considère ce point comme source lumineuse. Cela permet d’avoir des ombres en nuances de gris plutôt que des ombres très noires et marquées. Les deux images ci-dessous comparent un éclairage avec une sphère éclairante grande et une très petite qui peut donc être considérée ponctuelle.



(a) Source de rayon 0.1 (143s)

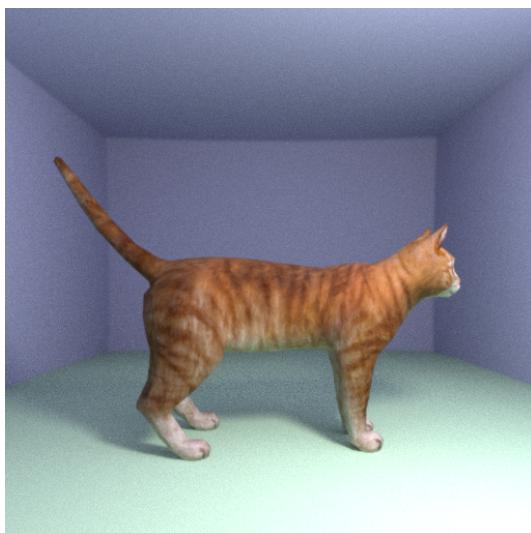
(b) Source de rayon 10 (137s)

FIGURE 4 – Images générées avec une source de rayon 0.1 et une source de rayon 10 en 1024x1024 avec 256 simulations et 10 rebonds

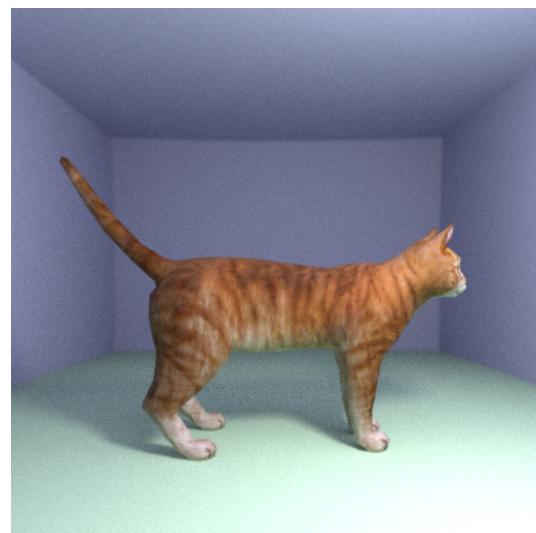
On remarque bien que dans le cas d'une source de rayon 10, les bords des ombres sont moins nettement marquées.

## 6 Intersections rayon plan(s) et gestion des maillages et textures

Les objets 3D sont généralement modélisés par des maillages de triangles. Afin de pouvoir les représenter, on calcule l'intersection d'un rayon avec un plan déterminé par le triangle du maillage. Afin de réduire les temps de calcul, on englobe chacun des maillages dans une boîte. Si un rayon n'intercepte pas cette boîte, alors il n'est pas nécessaire de vérifier s'il intercepte l'un des triangles du maillage. Enfin, afin d'accélérer davantage le processus, on utilise la Hiérarchie des volumes englobants (BVH) qui représente cette boîte sous la forme d'un arbre de sous-boîtes permettant de chercher les triangles concernés par des intersections en complexité logarithmique. Cela permet de réduire nettement le temps de calcul pour une image sans altérer le résultatat (cf photos ci-dessous).



(a) Sans BVH (572s)



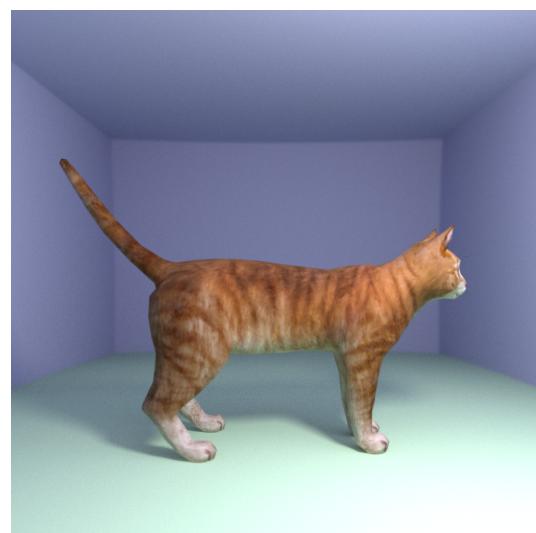
(b) Avec BVH (15s)

FIGURE 5 – Images générées avec et sans BVH en 512x512 avec 128 simulations et 5 rebonds

Enfin, l'interpolation des surfaces permet d'avoir des surfaces lisses pour ne pas pourvoir distinguer les triangles du maillage. L'effet apparaît nettement sur les images suivantes :



(a) Sans interpolation (225s)



(b) Avec interpolation (242s)

FIGURE 6 – Images générées en 1024x1024 avec 256 simulations et 10 rebonds