



ÉCOLE  
**CENTRALE**LYON

# ÉCOLE CENTRALE LYON

## MOS RAPPORT

---

# Projet Raytracing

---

*Élèves :*  
Antoine VALENTIN

*Enseignant :*  
Nicolas BONNEEL

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Séance 1</b>	<b>2</b>
2.1	Création d'un cercle blanc . . . . .	2
2.2	Ajout du relief . . . . .	3
2.3	Gestion d'une scène de plusieurs sphères . . . . .	4
<b>3</b>	<b>Séance 2</b>	<b>5</b>
3.1	Correction gamma . . . . .	5
3.2	Ajout des ombres portées . . . . .	6
3.3	Sphère miroir . . . . .	7
3.4	Sphère transparente . . . . .	8
<b>4</b>	<b>Séance 3</b>	<b>9</b>
4.1	Eclairage indirect . . . . .	9
4.2	Anti-aliasing . . . . .	10
<b>5</b>	<b>Séance 4</b>	<b>11</b>
5.1	Ombres douces . . . . .	11
5.2	Ombres douces améliorées . . . . .	12
5.3	Profondeur de champ . . . . .	13
<b>6</b>	<b>Séance 5</b>	<b>14</b>
6.1	Maillage . . . . .	14
6.2	Maillage avec BoundingBox . . . . .	15
<b>7</b>	<b>Séance 6</b>	<b>16</b>
7.1	BVH . . . . .	16
7.2	Maillage plus lisse . . . . .	17
<b>8</b>	<b>Séance 7</b>	<b>18</b>
8.1	Textures . . . . .	18
8.2	Déplacement caméra . . . . .	19
<b>9</b>	<b>Conclusion</b>	<b>20</b>

# 1 Introduction

Deux méthodes de rendus existent principalement. Tout d'abord, les méthodes par rasterization sont très rapides, mais assez peu précises. Elles sont donc adaptées notamment aux jeux vidéos ou aux simulations interactives. L'autre type, qu'on a étudié lors de ce cours, est celui par Raytracing : on lance un rayon, et on simule son cheminement dans une scène. Ces méthodes sont beaucoup plus lentes, mais elles permettent de simuler beaucoup plus facilement différents phénomènes physiques, et c'est justement ce que nous avons fait lors de ce cours.

## 2 Séance 1

### 2.1 Création d'un cercle blanc

On commence par créer 3 classes : **Vecteur** (à laquelle on rajoute une fonction pour obtenir la norme au carré du vecteur, et une autre pour normaliser le vecteur), **Rayon** (caractérisé par le centre du rayon, et la direction - tous deux des vecteurs) et **Sphère** (caractérisée par le centre de la sphère - un vecteur, et le rayon - un double).

Pour chaque pixel de l'image, on génère un rayon partant de la caméra et dirigé vers ce pixel (donc de direction  $V = (j\text{-largeur}/2+0.5, i\text{-hauteur}/2+0.5, -hauteur/(2\tan(\text{fov}/2.)))$ ). On calcule ensuite s'il y a une intersection entre ce rayon et la sphère, et s'il y en a une, on met le pixel blanc (sinon noir).

Pour vérifier l'intersection, on résout l'équation  $\|C + t \cdot V - O\|^2 = R^2$  ( $C$  est le centre de la caméra,  $V$  la direction,  $O$  le centre de la sphère et  $R$  son rayon).

Cela équivaut à résoudre  $t^2 + 2 < V, C - O > t + \|C - O\|^2 - R^2 = 0$ , il y a donc intersection si le discriminant est positif (ou nul), ce qui est réalisé dans la fonction intersection de **Sphère**.

On obtient l'image 1, après moins d'une seconde d'exécution.

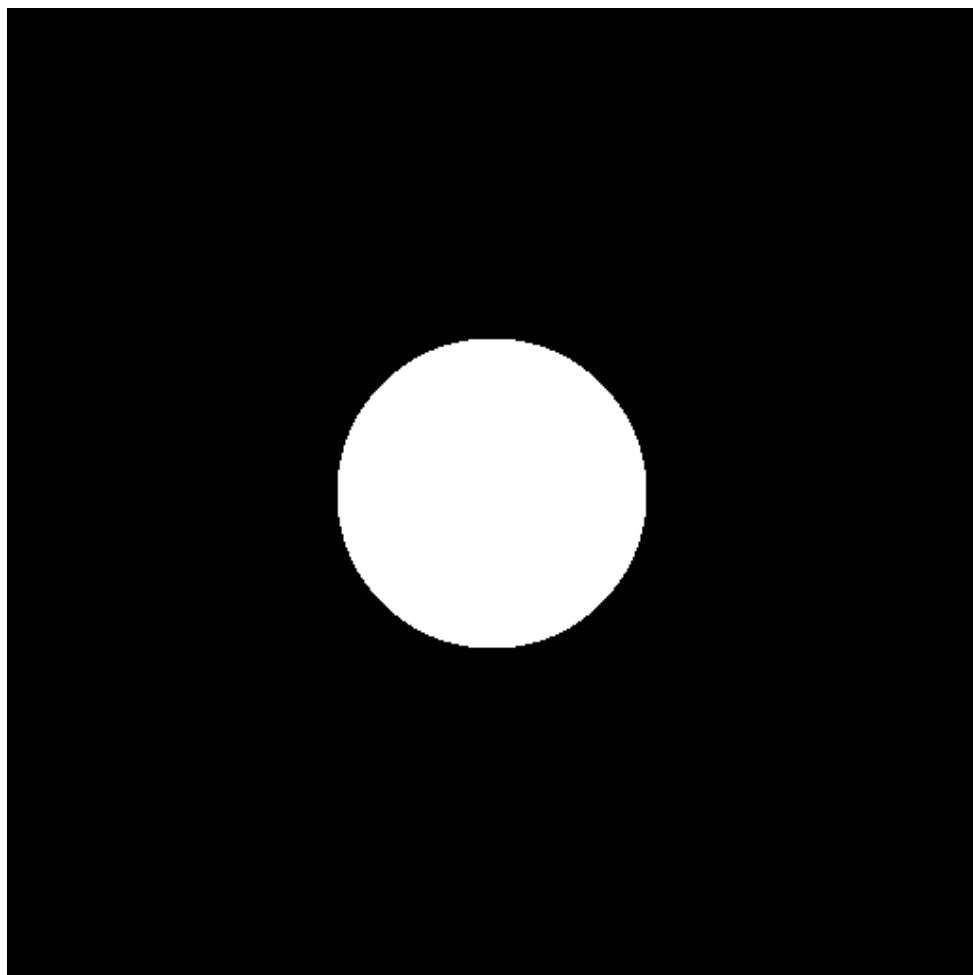


FIGURE 1 – Cercle blanc

## 2.2 Ajout du relief

Pour rajouter du relief à l'image, il ne faut plus rendre le pixel blanc en cas d'intersection, mais donner à ce pixel la valeur suivante :

$$\frac{I}{4*\pi*\|L-P\|^2} * \vec{N}, \frac{\vec{P} \cdot \vec{L}}{\|\vec{P} \cdot \vec{L}\|} > * \frac{\rho}{\pi}$$

avec P qui est le point d'intersection du rayon avec la sphère (qu'il faut donc désormais calculer dans la fonction intersection, avec  $P=C+tV$ ), N la normale à ce point d'intersection, et  $\rho$  qui est l'albedo, un vecteur de 3 composantes (R,V,B) qui représente la couleur de la sphère. Avec un albedo de (1,0,0), on obtient la figure 2, après moins d'une seconde d'exécution.

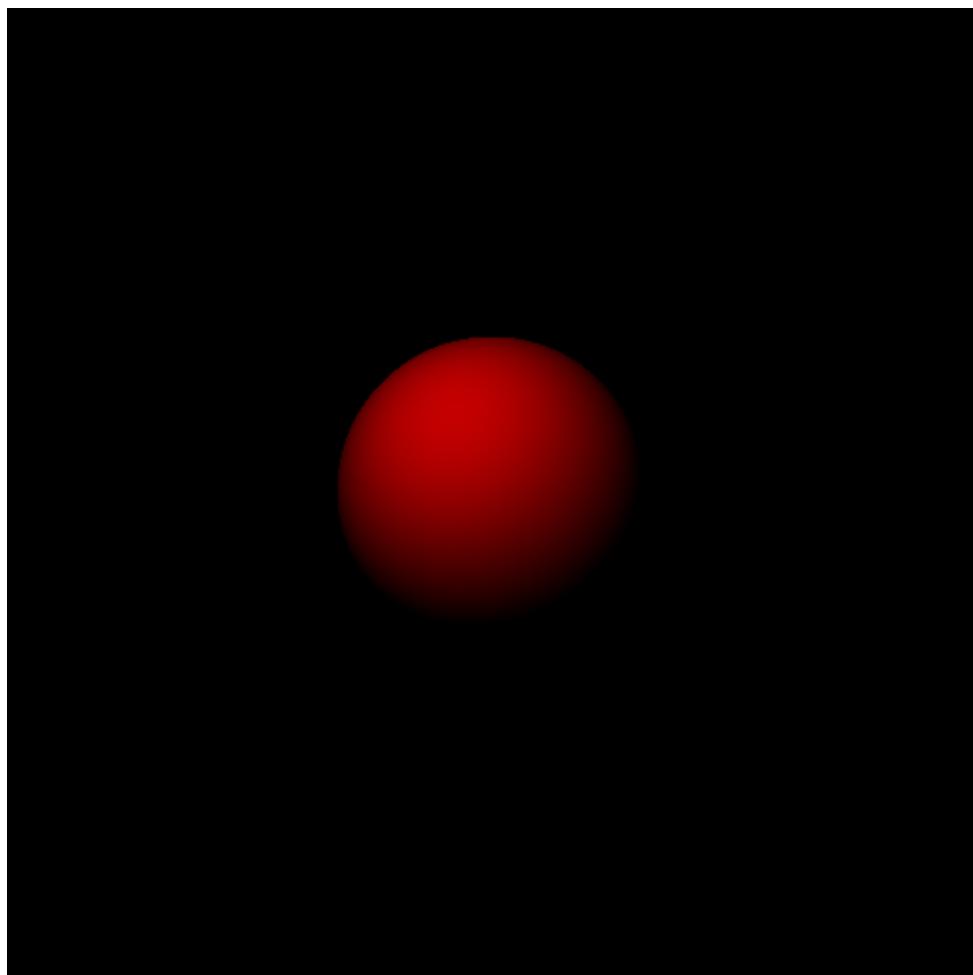


FIGURE 2 – Sphère avec du relief

### 2.3 Gestion d'une scène de plusieurs sphères

Pour pouvoir gérer plusieurs sphères, on crée une nouvelle classe, **Scene**, qui comporte une ou plusieurs sphères. Pour chaque pixel, on regarde désormais si le rayon intersecte la scène, via une nouvelle routine d'intersection : on réalise une boucle sur toutes les sphères de la scène, pour chercher les différentes intersections avec les sphères, on ne garde que celle la plus proche de la caméra (donc avec le  $t$  le plus faible), et on affiche la couleur du pixel correspondant à cette sphère.

On obtient l'image 3 avec plusieurs sphères, après moins d'une seconde d'exécution.

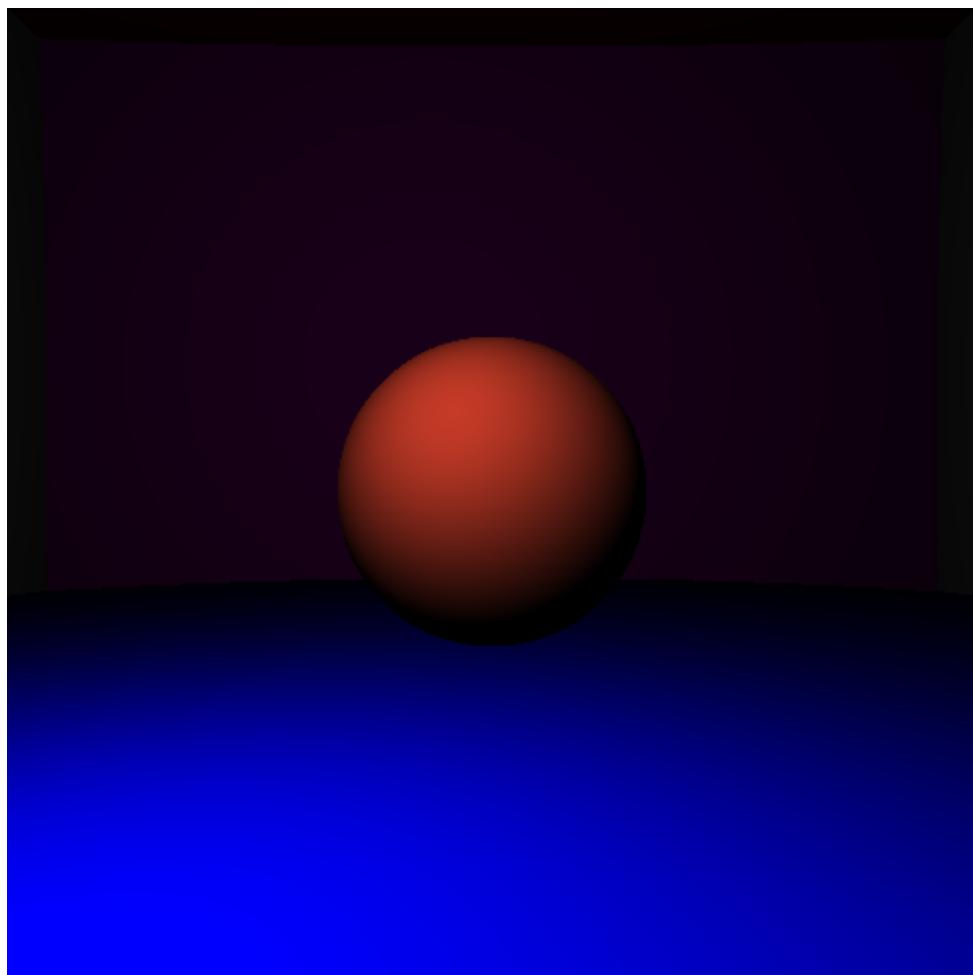


FIGURE 3 – Scène avec plusieurs sphères

## 3 Séance 2

### 3.1 Correction gamma

Les écrans ont un facteur gamma, qui fait que pour nos sphères, lors de l'affichage d'une couleur à 127, cela ne semble pas être à mi-chemin entre une valeur à 0 et une à 255. Pour corriger cela, on met les valeurs des couleurs des pixels à la puissance  $1/2.2$ .

On obtient l'image 4 (l'albedo de la sphère centrale n'est plus le même que sur la figure précédente), qui est beaucoup plus réaliste que la précédente, après moins d'une seconde d'exécution.

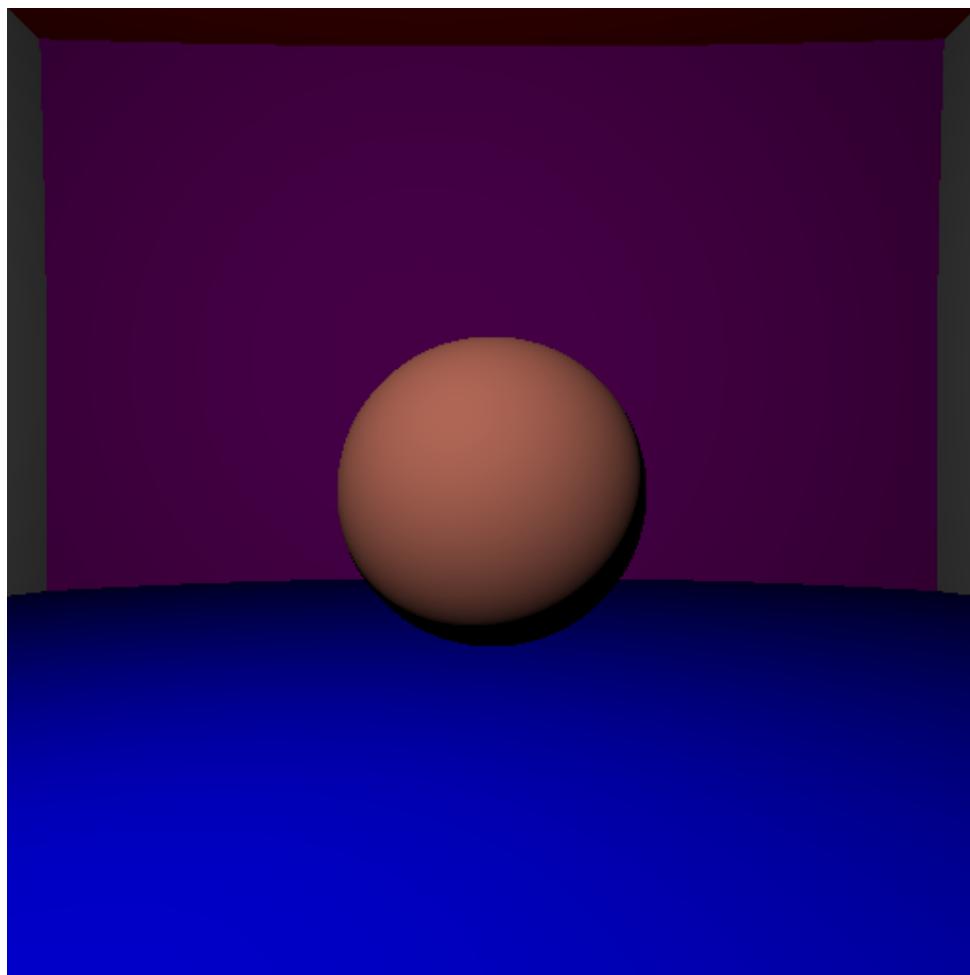


FIGURE 4 – Correction gamma

### 3.2 Ajout des ombres portées

Pour rajouter des ombres portées, le principe est simple : lorsque l'on trouve une intersection, on envoie un nouveau rayon vers la source de lumière et provenant de cette intersection (pour des raisons d'imprécision numérique, on décolle très légèrement l'origine de la surface de la sphère). Si ce rayon intersecte un objet et que cet objet est entre l'origine du rayon et la lumière (donc si  $t_{ombre} < \|\vec{PL}\|$ ), alors il y aura une ombre, donc le pixel sera noir.

On obtient la figure 5, on voit bien l'ombre de la sphère centrale, après moins d'une seconde d'exécution.

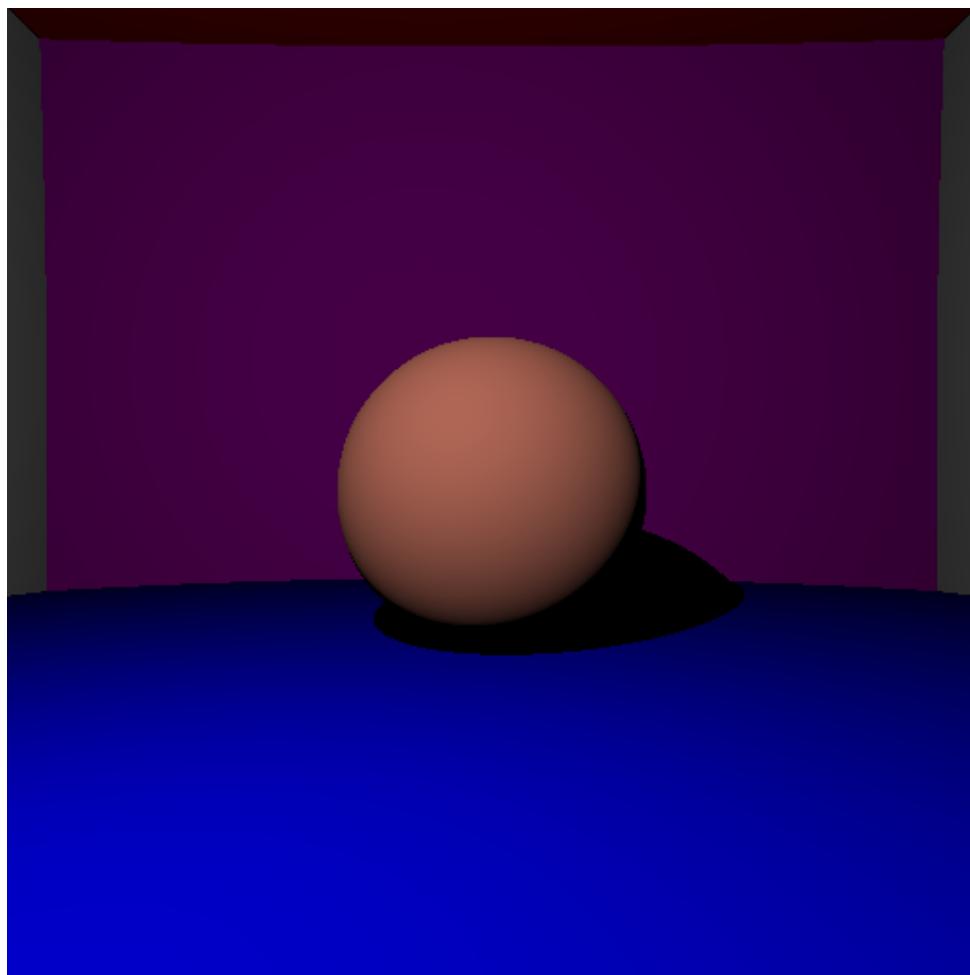


FIGURE 5 – Ombre portée

### 3.3 Sphère miroir

On veut ensuite pouvoir rajouter des surfaces miroirs sur les sphères, qui reflètent la lumière. Au lieu de continuer à calculer directement la couleur dans la fonction `main()`, on va désormais avoir besoin d'une nouvelle fonction, `obtenirCouleur`, qui a comme inputs le rayon et un nombre de rebonds. Dans le cas d'une sphère diffuse, pas de changement, on renvoie la couleur de la même façon qu'auparavant. En revanche, pour une surface miroir, on calcule la direction réfléchie ( $r = i - 2 \langle i, N \rangle N$ ), ce qui permet d'obtenir un rayon réfléchi, d'origine  $P$  (en réalité pour des raisons toujours d'imprécision numérique,  $P + \epsilon N$ ) et de direction  $r$ , et on calcule `obtenirCouleur( $P + \epsilon N$ , nombre de rebonds + 1)`. `obtenirCouleur` est donc une fonction récursive, qui s'arrête soit dans le cas d'une surface diffuse, soit si le nombre de rebonds atteint une certaine valeur (par exemple 5).

On obtient la figure 6, après moins d'une seconde d'exécution.

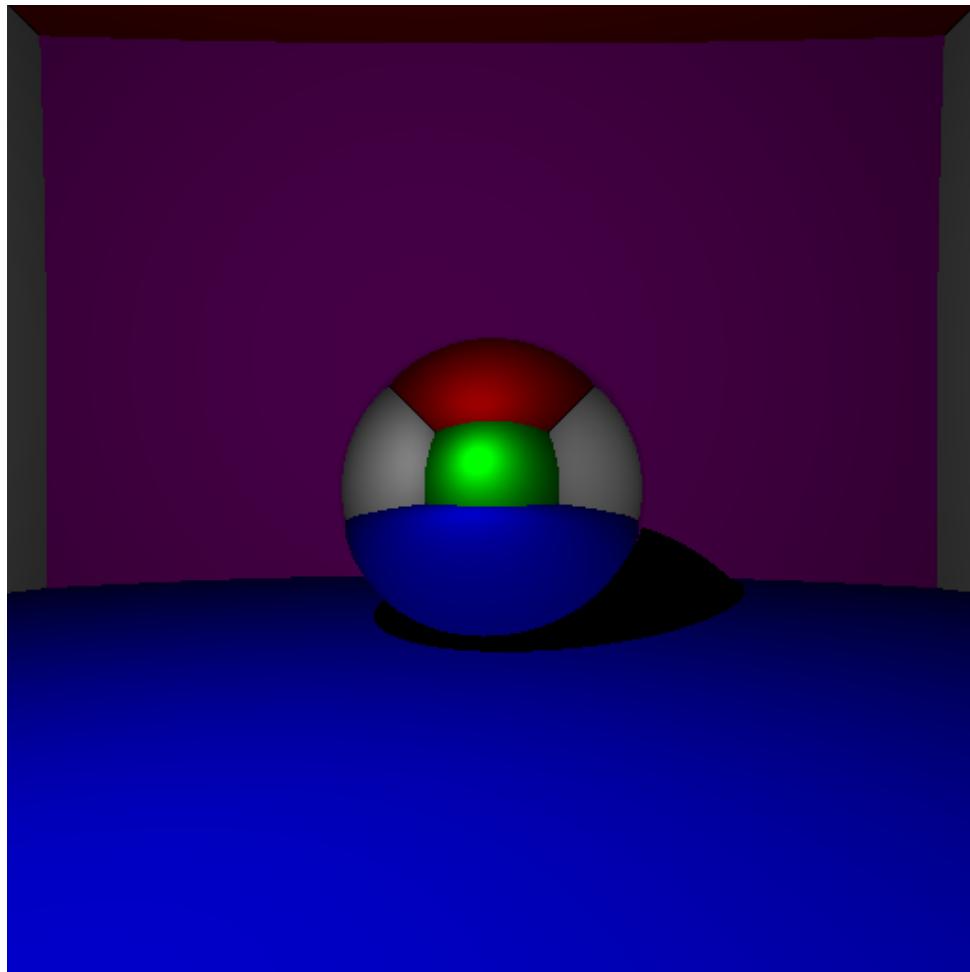


FIGURE 6 – Miroir

### 3.4 Sphère transparente

Finalement, on veut aussi gérer les surfaces transparentes, via un rajout dans la fonction obtenirCouleur.

En utilisant Snell-Descartes, on peut arriver pour la direction du rayon réfracté à :

$$\vec{r} = \frac{n_{air}}{n_{sphere}} \vec{i} - \left( \frac{n_{air}}{n_{sphere}} < \vec{i}, \vec{N} \rangle + \sqrt{1 - \left( \frac{n_{air}}{n_{sphere}} \right)^2 (1 - < \vec{i}, \vec{N} >^2)} \right) \vec{N}$$

(dans le cas où le rayon rentre dans la sphère transparente, sinon  $\frac{n_{air}}{n_{sphere}}$  devient  $\frac{n_{sphere}}{n_{air}}$  et N se change en -N)

Cette formule est valable si ce qu'il y a à l'intérieur de la racine est positif, sinon la direction est la même que pour le miroir (il y a réflexion). On doit ensuite renvoyer le rayon correspond à cette direction, via un nouvel appel à obtenirCouleur.

On obtient la figure 7, après moins d'une seconde d'exécution.

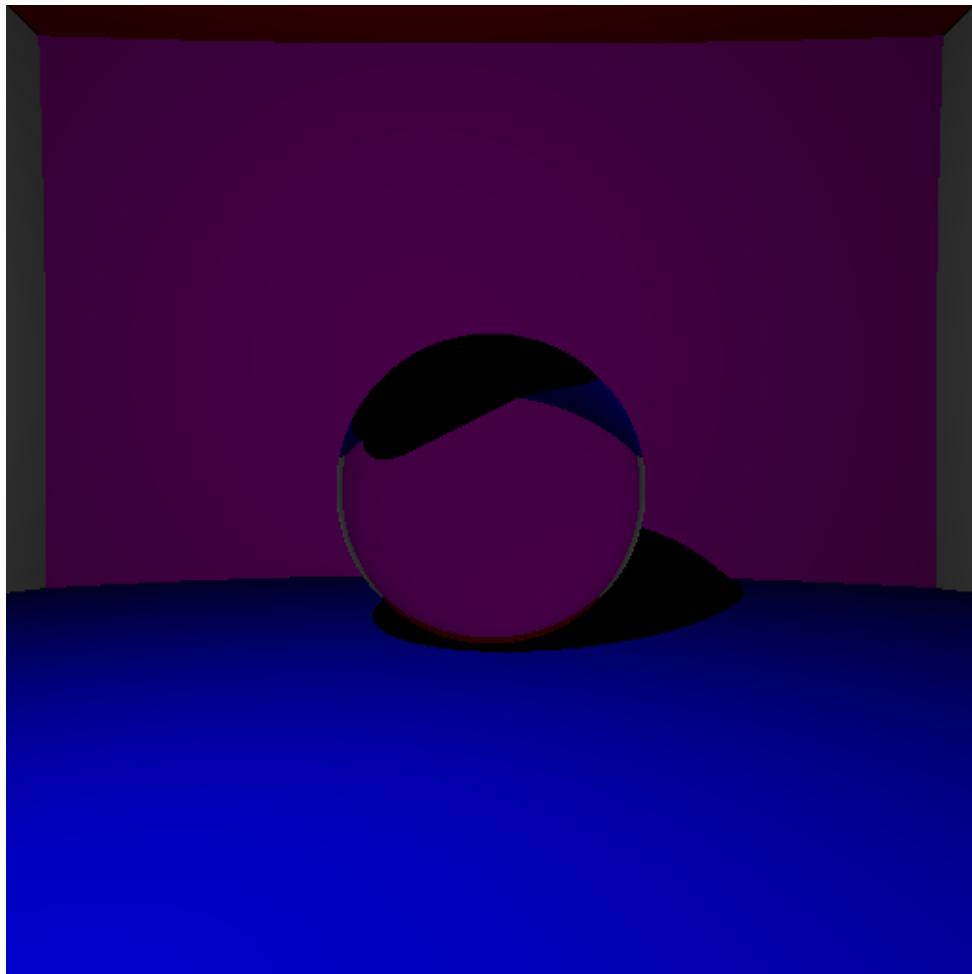


FIGURE 7 – Transparente

## 4 Séance 3

### 4.1 Eclairage indirect

On veut désormais implémenter l'éclairage indirect. On part de l'équation du rendu :

$$L_o(x, \vec{o}) = E(x, \vec{o}) + \int f(\vec{i}, \vec{o}) L_i(x, \vec{i}) \cos \theta_i d\vec{i}$$

(avec  $f$  la BRDF, constant pour une surface diffuse)

L'intégrale peut être approximée via Monte-Carlo :

$$\int f(\vec{i}, \vec{o}) L_i(x, \vec{i}) \cos \theta_i d\vec{i} \simeq \frac{1}{n} \sum_1^n f(\vec{i}_i, \vec{o}) L_i(x, \vec{i}_i) \cos \theta_i / p(\vec{i}_i)$$

On génère :

$$x = \cos(2r_1)\sqrt{1 - r^2}, y = \sin(2r_1)\sqrt{1 - r^2} \text{ et } z = r_2$$

avec  $r_1$  et  $r_2$  des nombres aléatoires entre 0 et 1.

On obtient donc une direction aléatoire  $zN + xT_1 + yT_2$  (avec  $N$  la normale à l'intersection, et  $(N, T_1, T_2)$  un repère) - implémentée dans la fonction qui s'appelle aleatoire, et on envoie un rayon dans cette direction (via un appel à obtenirCouleur(), et on rajoute la couleur obtenue (correspond à l'éclairage indirect) à celle obtenue par éclairage direct. Pour que l'éclairage indirect marche convenablement, on doit envoyer beaucoup de rayons (100 par exemple) initialement vers chaque pixel de l'image, et prendre finalement la moyenne des couleurs obtenus pour chacun de ces rayons.

On obtient l'image 8 avec 100 rayons (moins de 10 secondes d'exécution).

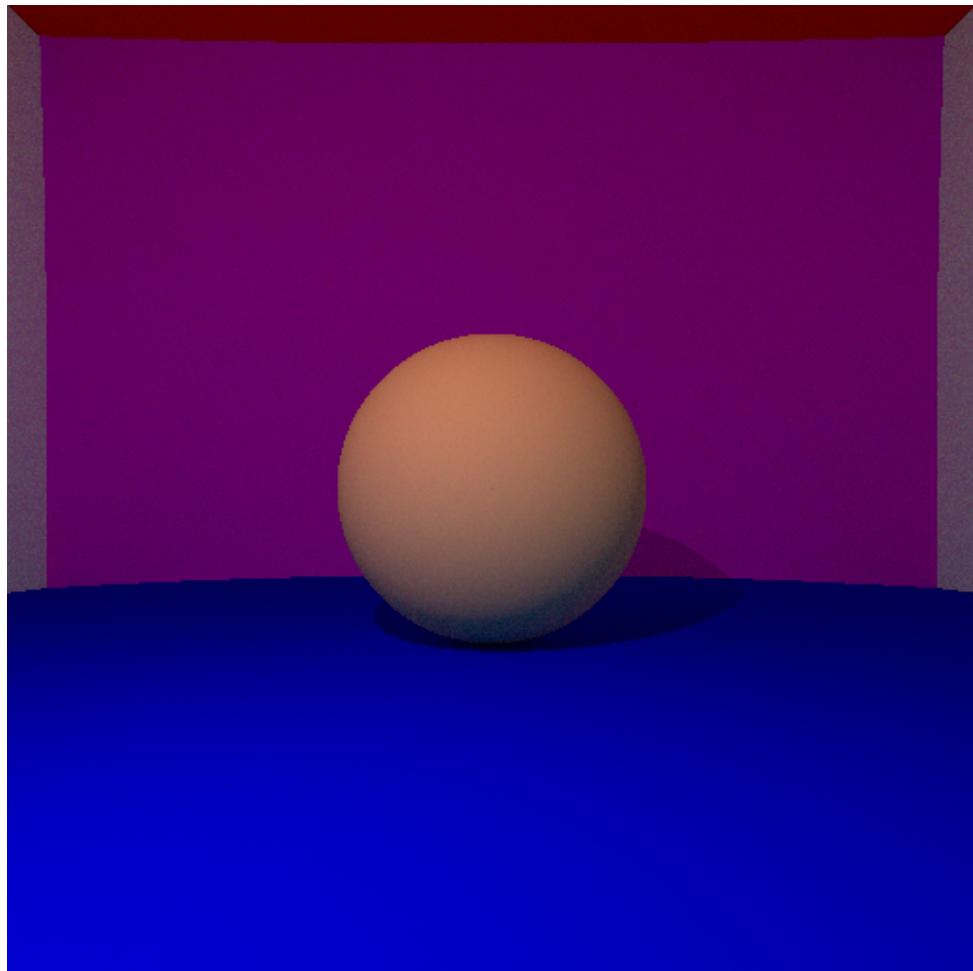


FIGURE 8 – Éclairage indirect avec 100 rayons

## 4.2 Anti-aliasing

C'est peu visible de loin, mais si on zoomé sur les contours de la sphère centrale, il y a de l'aliasing, c'est à dire que ces contours sont crénelés. C'est lié au fait que tous les rayons envoyés visent le milieu du pixel.

Pour corriger cela, on peut faire en sorte de ne plus viser uniquement le centre du pixel, mais un endroit aléatoire proche de ce centre (via une gaussienne centrée au centre du pixel). Chacun des par exemple 100 rayons visant un pixel, ne viseront donc plus uniquement son centre, mais chacun un endroit a priori différent proche de ce centre, ce qui évitera l'aliasing.

On obtient l'image 9 (moins de 10 secondes d'exécution).

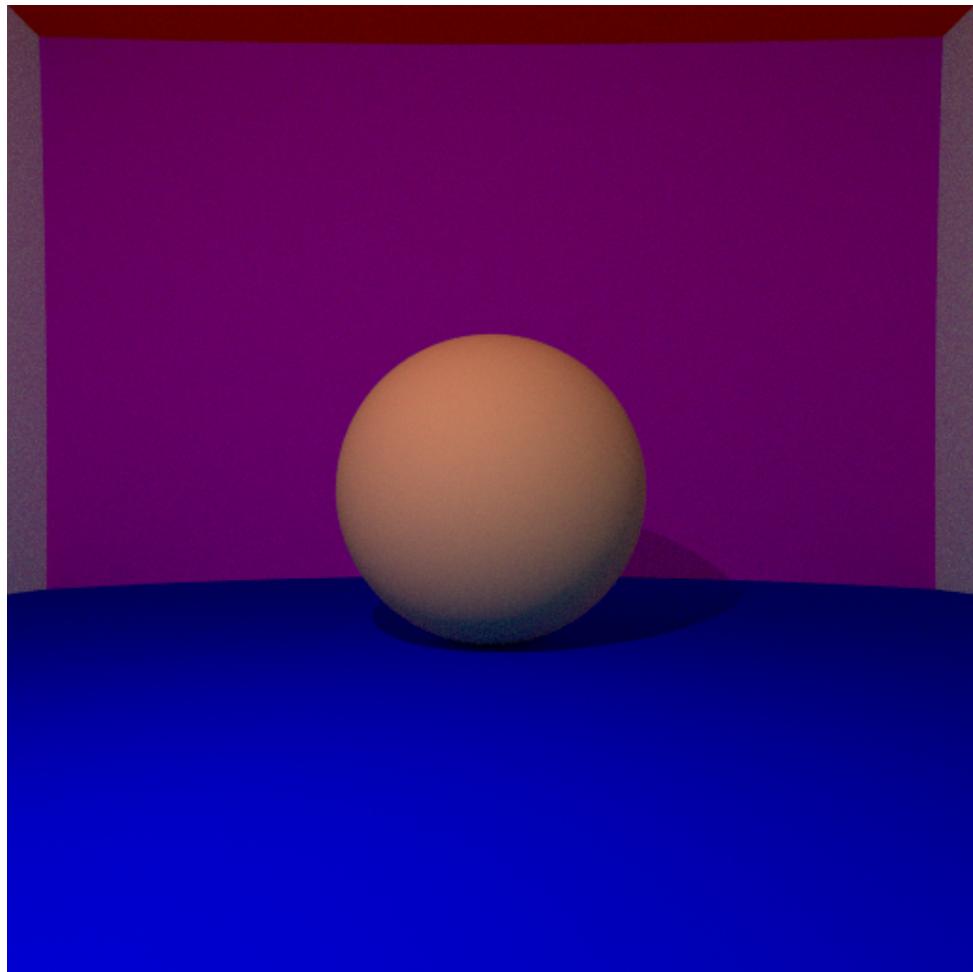


FIGURE 9 – Anti-aliasing

## 5 Séance 4

### 5.1 Ombres douces

Jusque là, notre source de lumière était ponctuelle. Pour pouvoir implémenter les ombres douces, il faut la remplacer par une sphère de lumière. Une première version de cette implémentation consiste à retirer la contribution de l'éclairage direct : il n'y a plus que l'éclairage indirect. En outre, lorsque le rayon arrive sur la sphère de lumière, cela renvoie la couleur  $\frac{I}{4\pi^2R^2}$ . Il faut donc que les rayons aléatoires de l'éclairage indirect atteigne cette sphère de lumière, ce qui oblige à la rendre très grosse, sinon l'image sera bruitée (puisque les rayons n'atteindront quasiment jamais la source de lumière, et donc les pixels seront tous presque noirs).

On obtient l'image 10, déjà bruitée alors que le rayon de la sphère lumineuse est assez gros, 15 pixels (moins de 10 secondes d'exécution).

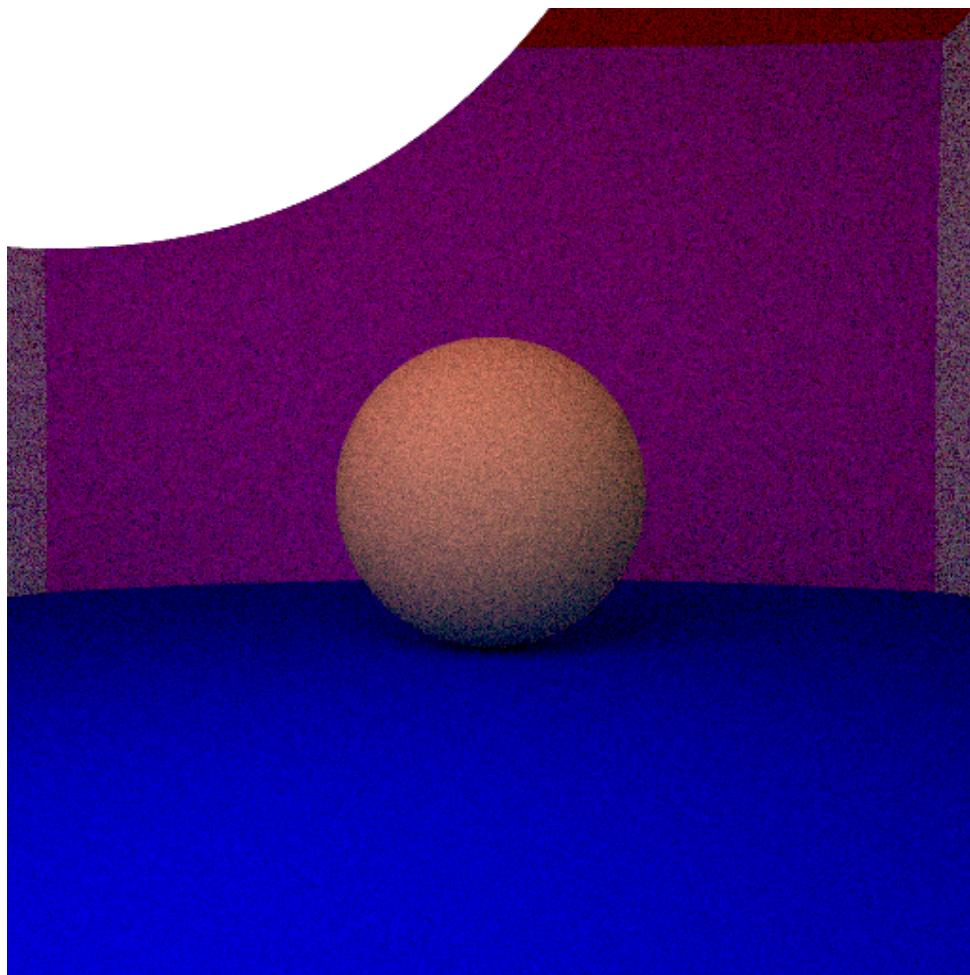


FIGURE 10 – Ombres douces

## 5.2 Ombres douces améliorées

Pour régler cela, on réimplémente une sorte d'éclairage direct en plus de l'indirect : cet éclairage direct est très similaire à celui qu'on utilisait lorsque la source de lumière était ponctuelle, sauf qu'on envoie un rayon aléatoire qui a plus de chance d'être dirigé vers la source de lumière, et on prend la couleur correspondante.

On obtient la figure 11, de bien meilleure qualité que la précédente (après 15 secondes d'exécution).

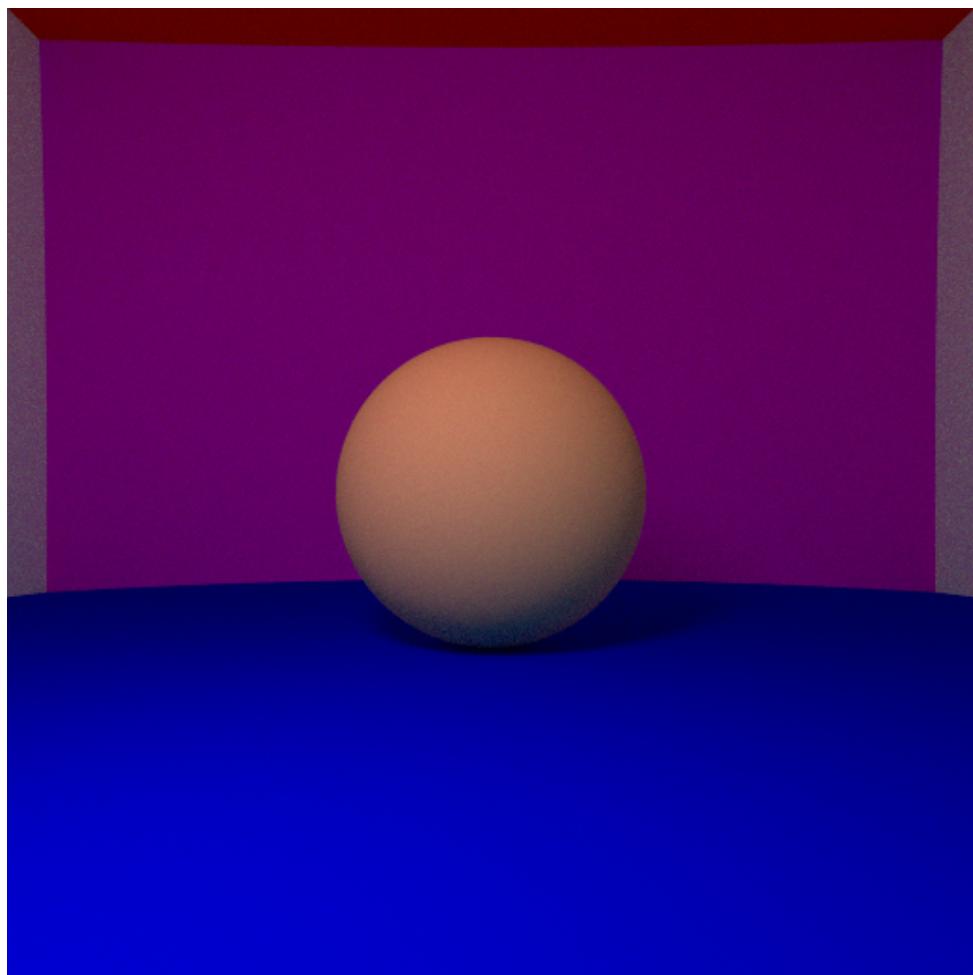


FIGURE 11 – Ombres douces améliorées

### 5.3 Profondeur de champ

Pour pouvoir implémenter la profondeur de champ, il faut d'abord calculer le point de focus  $F = C + 55 \vec{V}$  (avec  $C$  le centre de la caméra,  $\vec{V}$  qui est la direction du rayon et 55 qui est lié à la distance de focus). Au lieu de partir de  $C$ , on se place sur un point  $C'$  qui est pris aléatoirement sur le disque d'ouverture, et on envoie comme rayon celui d'origine  $C'$  et de direction  $\vec{C'F}$ .

On obtient l'image 12.

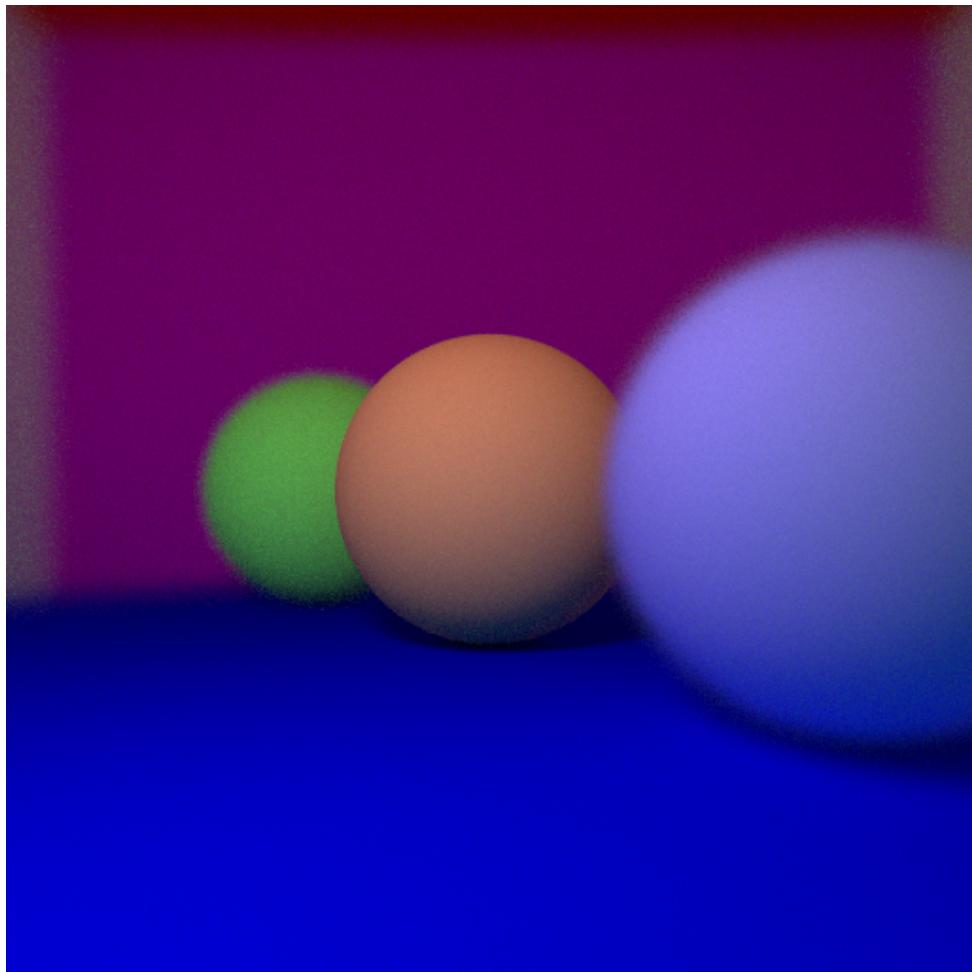


FIGURE 12 – Profondeur de champ

## 6 Séance 5

### 6.1 Maillage

Jusque là on n'affichait que des sphères, mais on veut aussi pouvoir afficher des maillages plus complexes : pour cela, on lit des fichiers .obj, qu'on met sous forme de **TriangleMesh**, comportant tous les triangles du maillage. Pour faire l'intersection avec les rayons envoyés, il s'agit d'une intersection rayon-triangle : le point d'intersection P peut s'écrire  $\alpha A + \beta B + \gamma C$  (avec A, B et C les sommets du triangle, et la somme des trois inconnues égale à 1) (ou  $A + \beta \vec{AB} + \gamma \vec{AC}$ ), et on peut calculer ces trois inconnues, ainsi que l'inconnue t (telle que  $P = C + tV$ ). En cas d'intersection avec le maillage, on ne prend en compte que l'intersection avec le triangle le plus proche de la caméra.

En prenant un maillage de Jeep (fourni sur le Github et venant de <https://www.cgtrader.com/free-3d-models/car/suv/jeep-renegade-a-5-doors-compact-suv-from-2016>), on obtient l'image 13, dont la taille a été diminué à 128x128, et avec 10 rayons (temps de calcul : 35 secondes).

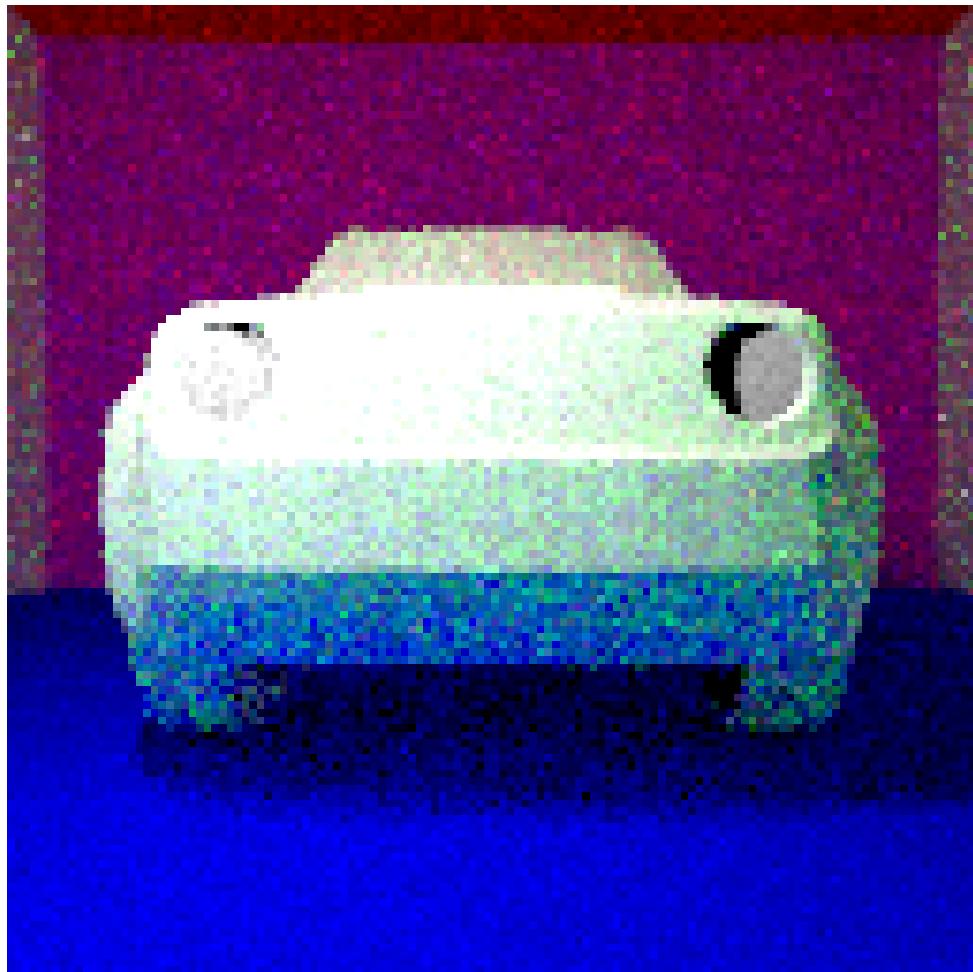


FIGURE 13 – Affichage d'un maillage de Jeep

## 6.2 Maillage avec BoundingBox

Pour faire en sorte que le calcul soit plus rapide, une première méthode est de ne plus tester directement si le rayon intersecte chaque triangle du maillage, mais avant cela, tester s'il intersecte la boîte englobante du maillage. En effet, s'il ne l'intersecte pas, il n'a aucune chance d'intersecter le maillage. On définit donc dans le code la boîte englobante du maillage dans une classe **BoundingBox**, et l'intersection d'un rayon avec cette boîte est une intersection rayon-boîte.

On obtient l'image 14, qui est quasiment la même que la précédente, sauf qu'elle n'a pris que 15 secondes à se calculer, donc moitié moins de temps.

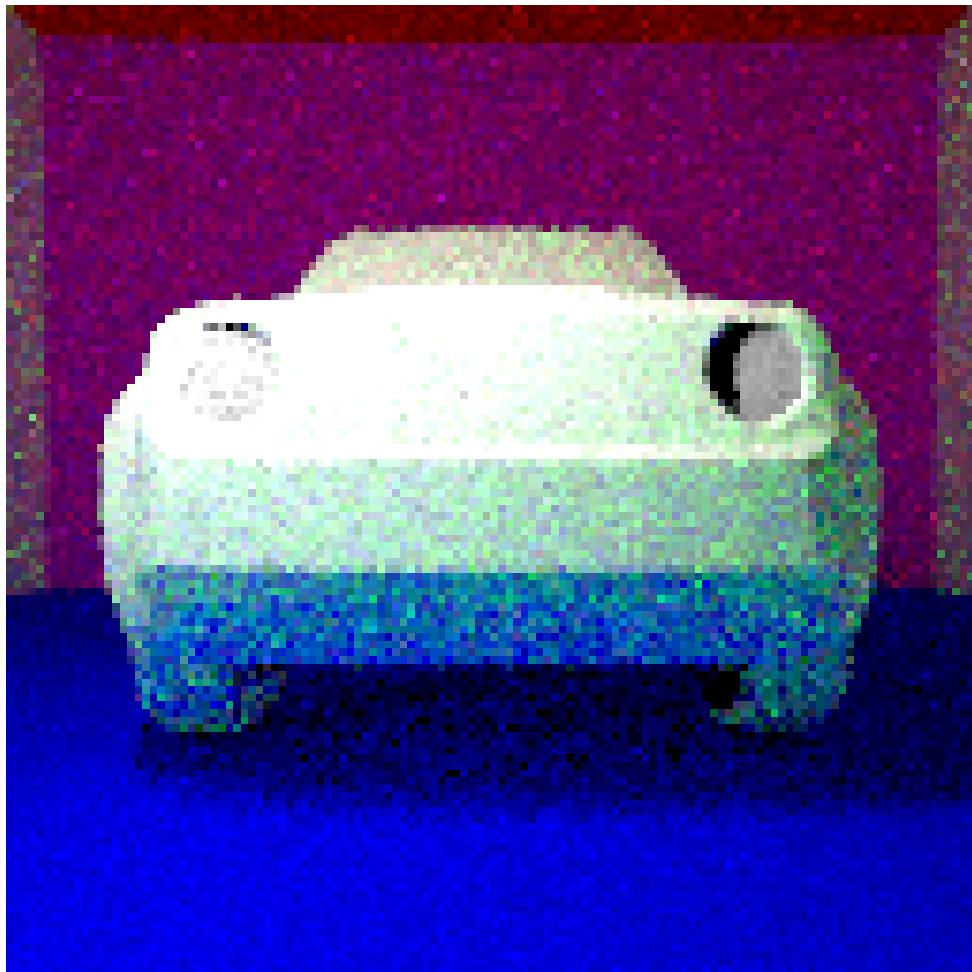


FIGURE 14 – Affichage d'un maillage de Jeep, en utilisant la méthode des Bounding Box

## 7 Séance 6

### 7.1 BVH

Pour encore améliorer le temps de calcul, on implémente une structure d'accélération encore plus efficace : la BVH. Il s'agit d'un arbre binaire, dont la racine est la Bounding Box du maillage complet. On prend ensuite la dimension la plus longue de cette boîte, et on classe les triangles du maillage selon que leur barycentre est avant ou après le milieu de cette dimension de la Bounding Box (en pratique, cela est fait via un Quick Sort). On obtient donc deux ensembles de triangles, et on peut calculer les Bounding Boxes de ces deux ensembles, qui seront les deux enfants de la Bounding Box précédente. On peut ensuite continuer à calculer les enfants de ces deux nouvelles boîtes, et caetera (jusqu'à atteindre des ensembles de seulement 5 triangles). Pour calculer l'intersection, on fait un parcours en profondeur : si le rayon intersecte la première Bounding Box, alors on teste l'intersection avec ses deux enfants (en commençant par la gauche), et on continue ainsi jusqu'à arriver à la dernière Bounding Box, où en cas d'intersection, on teste l'intersection rayon-ensemble des triangles de cette Bounding Box.

Pour 50 rayons, une image de taille 512x512, on obtient l'image 15, en seulement une minute.

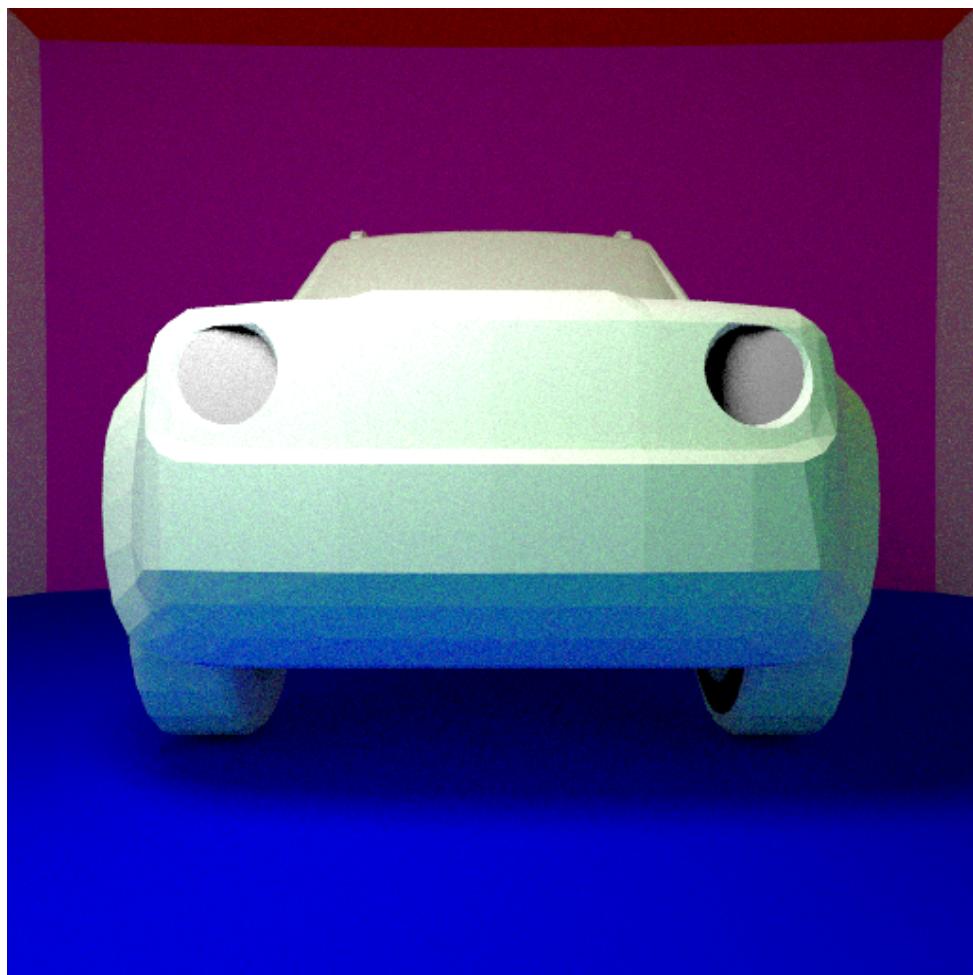


FIGURE 15 – Affichage d'un maillage de Jeep, en utilisant la méthode du BVH

## 7.2 Maillage plus lisse

Pour avoir une image plus lisse, plutôt que de prendre les normales calculées, on prend celles définies par l'artiste ayant fait le modèle 3D (on a les normales aux sommets des triangles, il suffit donc pour avoir la normale au point d'intersection de prendre  $\alpha N(A) + \beta N(B) + \gamma N(C)$ ).

On obtient l'image 16, avec les mêmes paramètres que la précédente (et le même temps de calcul), on voit bien que l'image est plus lisse, il n'y a plus des sortes d'arêtes entre les polygones.

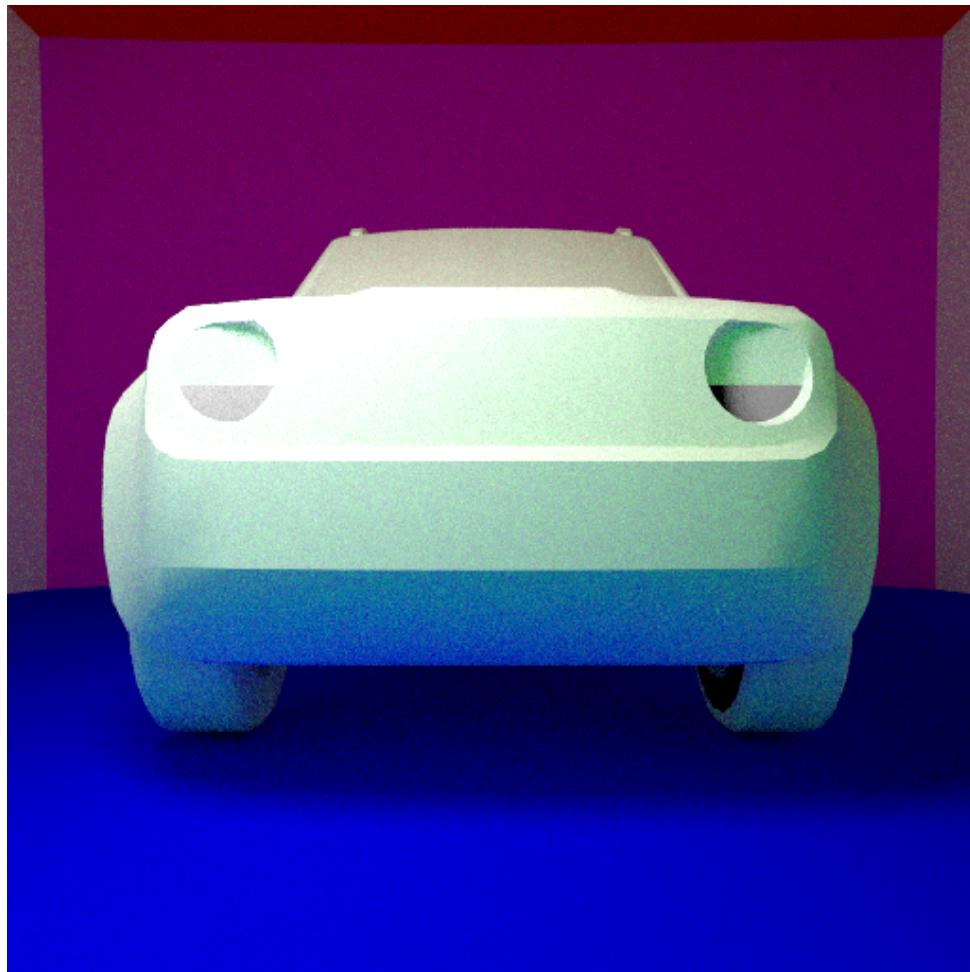


FIGURE 16 – Affichage d'un maillage de Jeep, mais en utilisant les normales du fichier obj

## 8 Séance 7

### 8.1 Textures

On veut désormais implémenter la texture sur le maillage, qui vient d'un fichier .jpg, qu'on charge dans un vecteur textures. Dans le fichier obj initial sont fournies les coordonnées UV : pour chacun des sommets des triangles, on a le pixel de l'image jpg correspondant. De la même façon que pour les normales, pour avoir l'UV correspond au point d'intersection rayon-triangle, on prend  $\alpha UV(A) + \beta UV(B) + \gamma UV(C)$ . Pour l'implémentation, il faut bien faire en sorte que ces coordonnées UV correspondent bien à un pixel de l'image (et donc ne soient pas négatives, ou pas juste entre 0 et 1). Il faut également appliquer un facteur gamma à la couleur obtenue.

On obtient finalement l'image 17, toujours après 1 minute de calcul.



FIGURE 17 – Affichage d'un maillage de Jeep avec sa texture

## 8.2 Déplacement caméra

Pour pouvoir déplacer la caméra, il faut créer trois vecteurs : up, qui correspond à la verticale souhaitée de la caméra, right qui pointe vers la droite, et Direction = - up  $\wedge$  right, qui est la direction de visée. A partir de la direction V du rayon que l'on avait auparavant, on a  $V_{new} = \text{right } V[0] + \text{up } V[1] + \text{Direction } V[2]$ .

En rendant la caméra plus plongeante, on obtient l'image 18 (toujours 1 minute de calcul). Dans le dossier sur Github où il y a le code, il y a également une vidéo qui montre un déplacement de caméra (entre un angle de plongée de 0° jusqu'à 30°).



FIGURE 18 – Affichage d'un maillage de Jeep avec sa texture et une caméra plus plongeante

## 9 Conclusion

Lors de ce cours, nous avons donc étape par étape amélioré notre Raytracer, pour qu'il soit finalement capable d'afficher des maillages complexes, avec les textures associées à ces maillages.