ÉCOLE CENTRALE LYON

Informatique Graphique

Partie pratique

Développement d'un Ray Tracer

Auteur :
M. Miguel Reuter

Professeur :
Pr. Nicolas Bonneel

22 mars 2018

Table des matières

Introduction

Parmi les méthodes de rendu, 2 familles existent :

— Les méthodes de rasterization qui affichent des formes sur l'écran. C'est cette méthode qui est utilisée dans les jeux vidéos par exemple pour afficher les triangles. Cette famille a pour avantage d'obtenir un rendu rapide mais n'est pas très réaliste, car ne prend pas en compte de manière "native" les effets physiques comme la transparence, le reflet de lumière sur une surface etc.

— Les méthodes de lancer de rayons (ray tracing) quant à elles simulent le cheminement (inverse) de la lumière dans une scène. Ces méthodes sont beaucoup plus réalistes car se basent sur des équations physiques mais sont beaucoup plus lentes. Pour le rendu d'une image, plusieurs heures de calcul peuvent être nécessaires.

Dans le cours d'informatique graphique, les notions nécessaires à la construction d'un ray tracer ont été développées. Ce présent rapport a pour but d'expliquer brièvement l'implémentation du ray tracer en langage C++ et de présenter les résultats obtenus. Les éléments théoriques ne seront pas abordés ici.

Chapitre 1

Implémentation générale du Ray Tracer

Nous présentons ici l'architecture globale du programme. Le fichier *main.cpp* gère l'ensemble du programme de Ray Tracer.

Les différents fichiers, avec headers, sont les suivants :
— Object
— Mesh
— Sphere
— Vector
— Light
— Scene
— Ray

Vector :

La classe *Vector* correspond à un objet représentant un ensemble de 3 double. Ils sont utilisés pour désigner une position, un vecteur ou même une couleur RGB. Les principales méthodes (addition de vecteurs, normalisation, produit scalaire etc.) sont définies dans cette classe.

1.1 La lumière

Une classe *Light* a été définie et représente les sources ponctuelles de lumières. Différentes caractéristiques sont données :
— sa position
— la couleur de la lumière
— son intensité
— son rayon (pour la prise en compte des lumières étendues)

Dans la seconde partie du rapport, les premiers résulats sont donnés en n'utilisant que des sources de lumière ponctuelles.

1.2 Les rayons

Pour calculer les intersections, des rayons sont envoyés depuis chaque pixel de la caméra. Ces rayons sont définis dans la classe *Ray* par :

— une origine
— une direction

1.3 Les objets visualisables

La classe abstraite *Object* correspond aux objets que l'on souhaite faire afficher par le ray tracer. Un tel objet est constitué de propriétés comme sa couleur, son comportement optique (caractère diffus, spéculaire ou transparent), et si nécessaire ; sa couleur spéculaire et son coefficient de Fresnel.

2 Classes filles ont été définies :

— Sphere : sphères définies par sa position et son rayon
— Mesh : maillage défini par un ensemble de triangles et potentiellement une texture.

Une méthode qui calcule l'intersection avec un rayon donné est définie pour ces 2 classes. Cette méthode renvoie entre autre le point d'intersection s'il y en a une, la normale, la couleur en ce point...

1.4 La scène

La classe *Scene* contient tous les objets définis précédemment : les meshes, les sphères et les lumières de la scène. On peut ajouter des objets dans la scène et c'est cette classe qui va s'occuper de calculer pour un rayon lancé depuis la caméra, la couleur résultante (en appelant les méthodes d'intersection des objets visualisables).

1.5 Programme en général

Le programme de ray tracing, d'après les données de la scène et de quelques paramètres va calculer une image et la sauvegarder dans le répertoire courant.

Des paramètres peuvent être changés, au début du fichier *main.cpp* :

```
#define NUM_RAYS 50
#define NUM_BOUNDS 3

#define ANTIALIASING true
#define DOF false

double shutter_size = 1.0;
double focus_distance = 75;
```

La valeur NUM_RAYS correspond au nombre de rayons lancés par pixel, NUM_BOUNDS correspond au nombre de rebonds par rayon lancé depuis la caméra. Les booléens ANTI-ALIASING et DOF permettent de prendre en compte ou non l'anti-aliasing et la profondeur de champs. Les valeurs de focus_distance et shutter_size correspondent à des paramètres pour la profondeur de champs (respectivement la distance à laquelle un objet sera net, et l'importance de l'effet de profondeur de champs, qui correspond en fait à la taille de l'obscurateur).

Chapitre 2

Résultats obtenus

2.1 Image binaire

Dans un premier temps, l'implémentation de l'intersection entre un rayon et une sphère a été effectuée. Nous initialisons une scène toute simple composée juste d'une sphère et d'une caméra. Pour l'instant, la lumière n'est pas prise en compte. La sortie est une image binaire montrée en figure 2.1 : un pixel appartenant à la sphère est blanc, sinon il est noir.

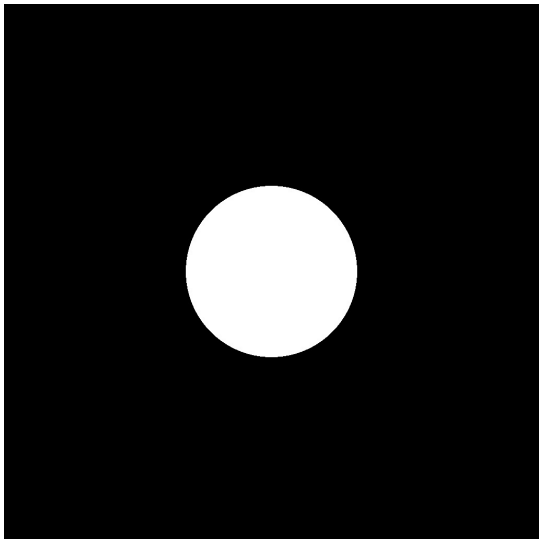

FIGURE 2.1 – Image binaire d'une sphère

2.2 Prise en compte de la couleur et la lumière

Nous prenons en compte maintenant la lumière et la couleur de la sphère. La lumière est ponctuelle pour l'instant et est définie par :

— une couleur
— une intensité lumineuse
— une position dans la scène

Nous obtenons les figures 2.2 et 2.3.

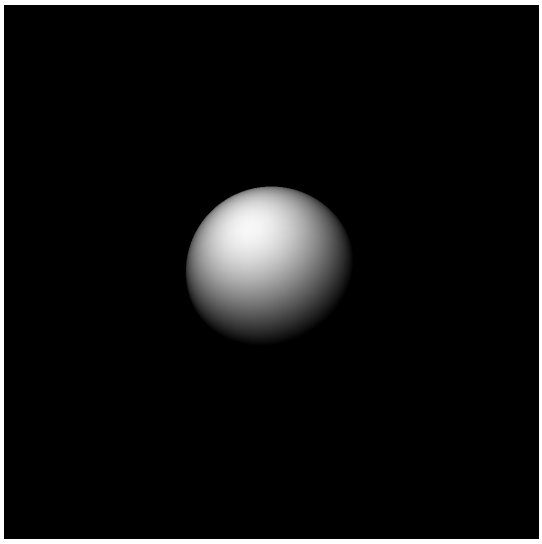

FIGURE 2.2 – Prise en compte de la lumière. La lumière est située à gauche de la sphère

Pour les sphères, l'information de couleur correspond en fait à un coefficient d'absorption pour chaque composante RGB. La couleur de la lumière est également prise en compte en modulant ces composantes.

Une correction gamma a été appliquée (non présent sur les figures précédentes) en modifiant la valeur de chaque composante d'un pixel :

$$pixel' = pixel^{1/2.2}$$

2.3 Les ombres portées

Les ombres portées peuvent être prises en compte en faisant des tests d'intersections supplémentaires. En effet, pour chaque rayon qui part de la caméra et qui arrive sur une surface (sphères pour l'instant), on relance un rayon depuis ce point d'intersection vers la lumière de la scène. S'il y a un autre objet entre la lumière et le précédent point, il y aura une ombre portée et la couleur sera donc noire. On n'oublie pas de décoller le rayon de

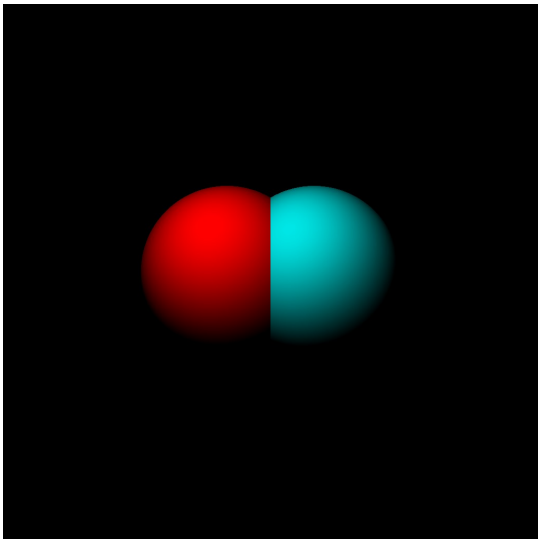

FIGURE 2.3 – Prise en compte de la couleur. Cas de plusieurs sphères s'intersectant pour vérifier le rendu de la frontière.

la surface pour éviter d'avoir du bruit (dues aux imprécisions, la moitié des rayons seront considérés comme étant à l'intérieur de la sphère ce qui n'est pas le cas). Le résultat est visible en figure 2.4.

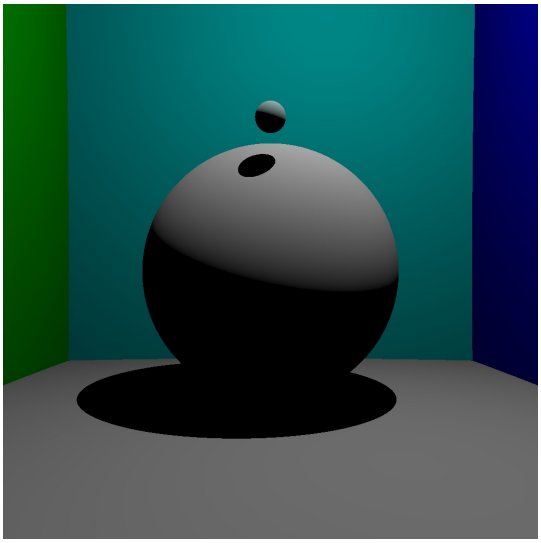

FIGURE 2.4 – Ombre portée

2.4 Surfaces spéculaires et transparentes

Pour l'instant, un modèle de lumière diffuse est utilisée pour les sphères présentes dans la scène. Nous souhaitons maintenant prendre en compte les surfaces spéculaires (miroir) et transparentes. Dans les 2 cas, de nouveaux rayons seront envoyés dans des directions précises depuis ces surfaces. Le rendu de ces surfaces sont visibles en figure 2.5.

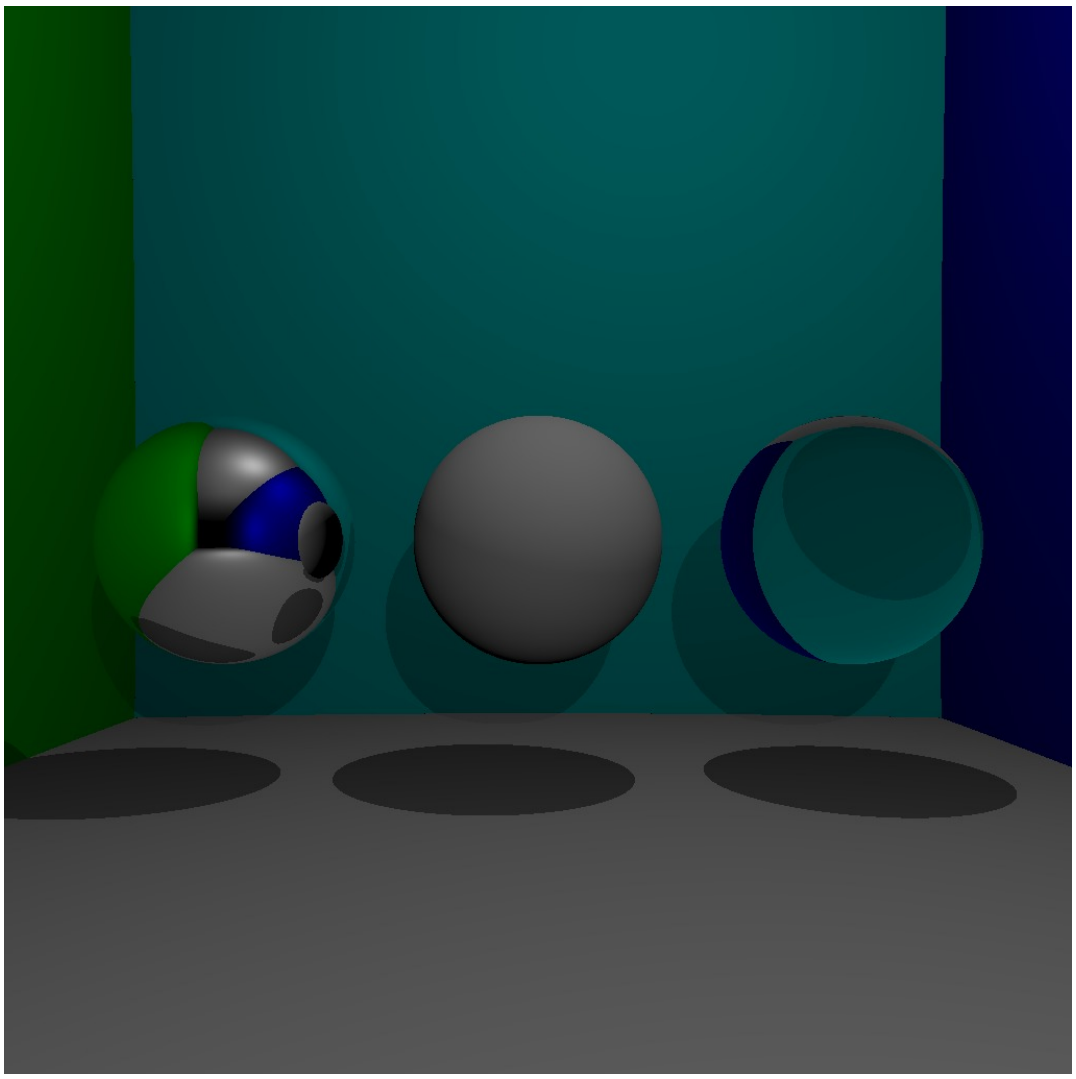

FIGURE 2.5 – De gauche à droite : surface spéculaire, diffuse et transparente. 2 lampes ont été utilisées, d'où la présence d'ombres portées sur le mur et sur le sol.

2.5 Anti-aliasing

En zoomant sur les images précédentes, on peut observer du crénelage : une surface lisse, comme le contour d'une sphère, sera affichée avec des sauts correspondant aux pixels. L'anti-aliasing consiste à n'envoyer plus un rayon pour chaque pixel, mais plusieurs par pixel avec des directions légèrement différentes. La couleur d'un pixel est alors moyennée sur l'ensemble des rayons lancés. On obtient une surface plus lisse comme présenté en figure 2.6.

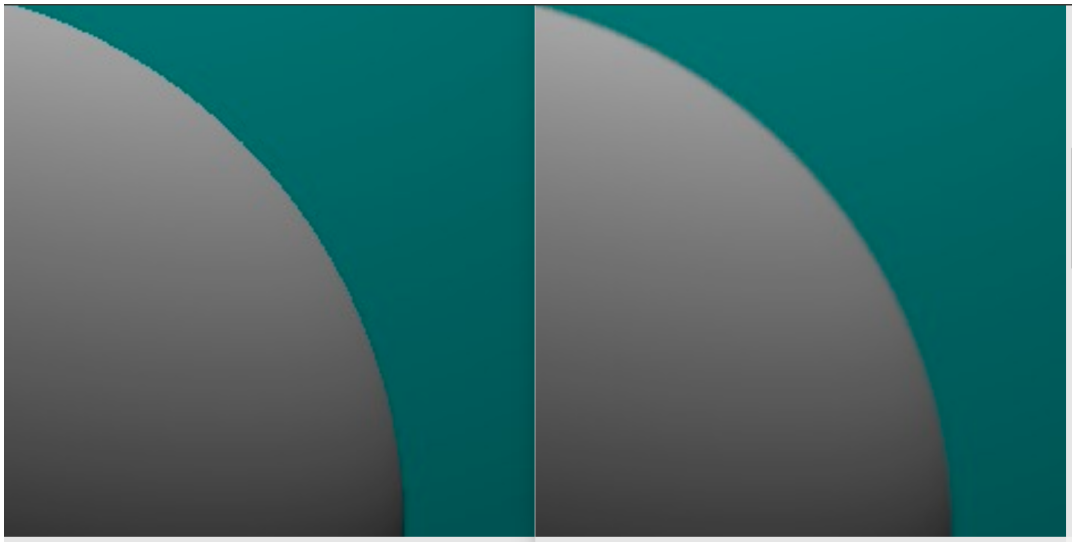

FIGURE 2.6 – image sans et avec anti-aliasing (gauche et droite). 50 rayons ont été lancés par pixel dans l'image anti-aliasée

Il est nécessaire d'avoir un nombre de rayons assez important car sinon l'image est bruitée.

2.6 L'éclairage indirect

La notion de BRDF vue en cours a été implémentée (pour le cas diffus) dans le ray tracer. Il s'agit de prendre en compte l'éclairage indirect en chaque point de la scène. le nombre de rayons et de rebons influent sur les résultats obtenus.

Ici aussi, si on veut éviter le bruit sur l'image, un grand nombre de rayons sont nécessaires. On observe en figure 2.7 que les lumières sont plus douces (au niveau des ombres par exemple) et la scène paraît plus réaliste que l'image présentée en figure 2.5. On observe également une teinte correspondant aux objets environnants sur la sphère diffuse (une teinte verte sur le côté gauche de la sphère et bleue à droite). L'anti-aliasing n'a pas été utilisé sur cette image.

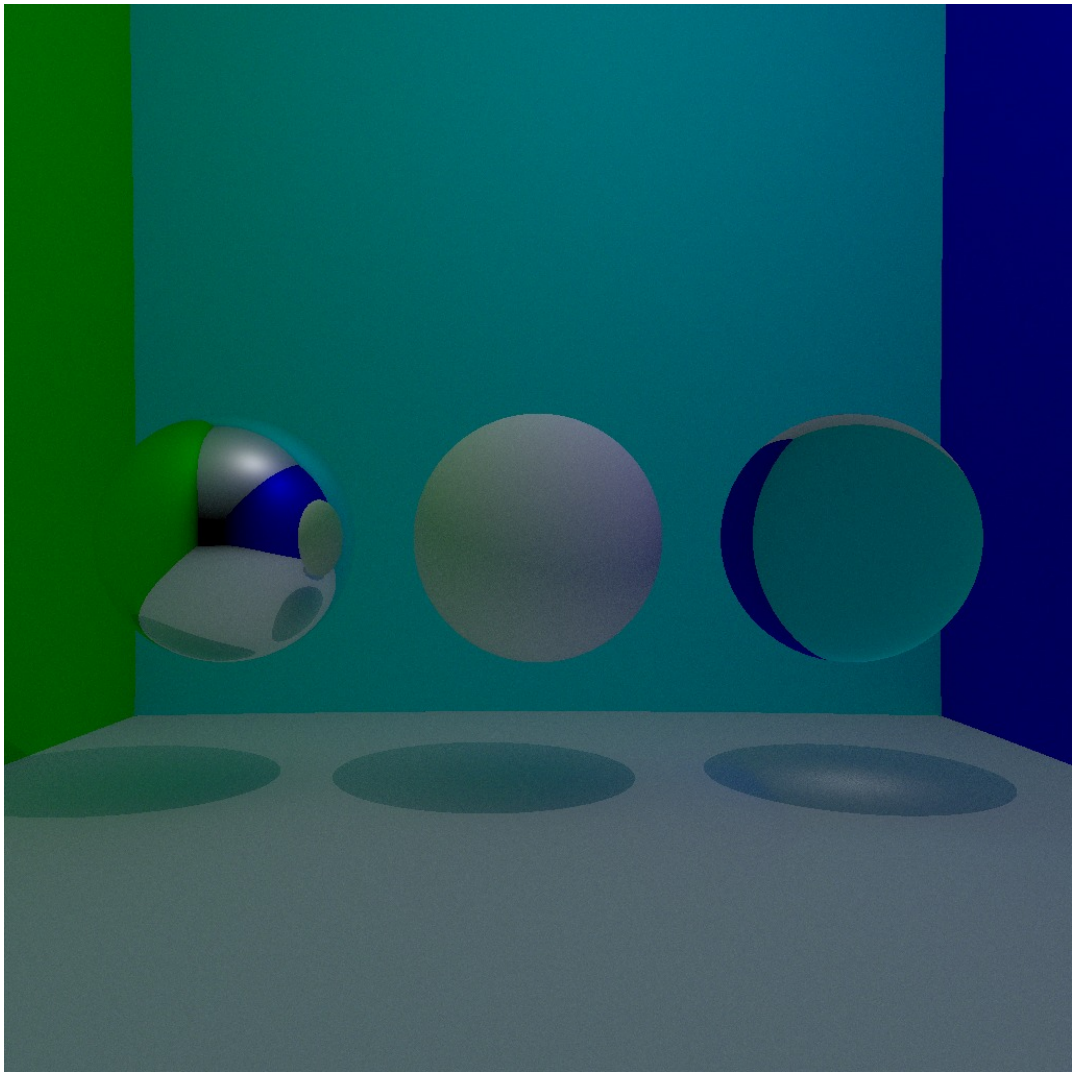

FIGURE 2.7 – Scène en prenant compte l'éclairage indirect. 1000 rayons et 5 rebonds sont utilisés dans cette image

2.7 Effet de profondeur de champs : Depth of Field

Un effet de profondeur de champs a été implémenté. Le principe est que pour chaque pixel, les différents rayons sont lancés depuis une position légèrement différente du centre de la caméra et une direction légèrement changée. Ces modifications sont faites de manière aléatoire. On simule de cette manière un modèle de caméra avec une lentille et un obturateur, ce qui permet d'avoir l'effet de profondeur. Sur la figure 2.8, on voit que seule la sphère du milieu est nette, car elle se situe à la distance de focus de la caméra. L'image est encore bruitée, il faudrait augmenter le nombre de rayons.

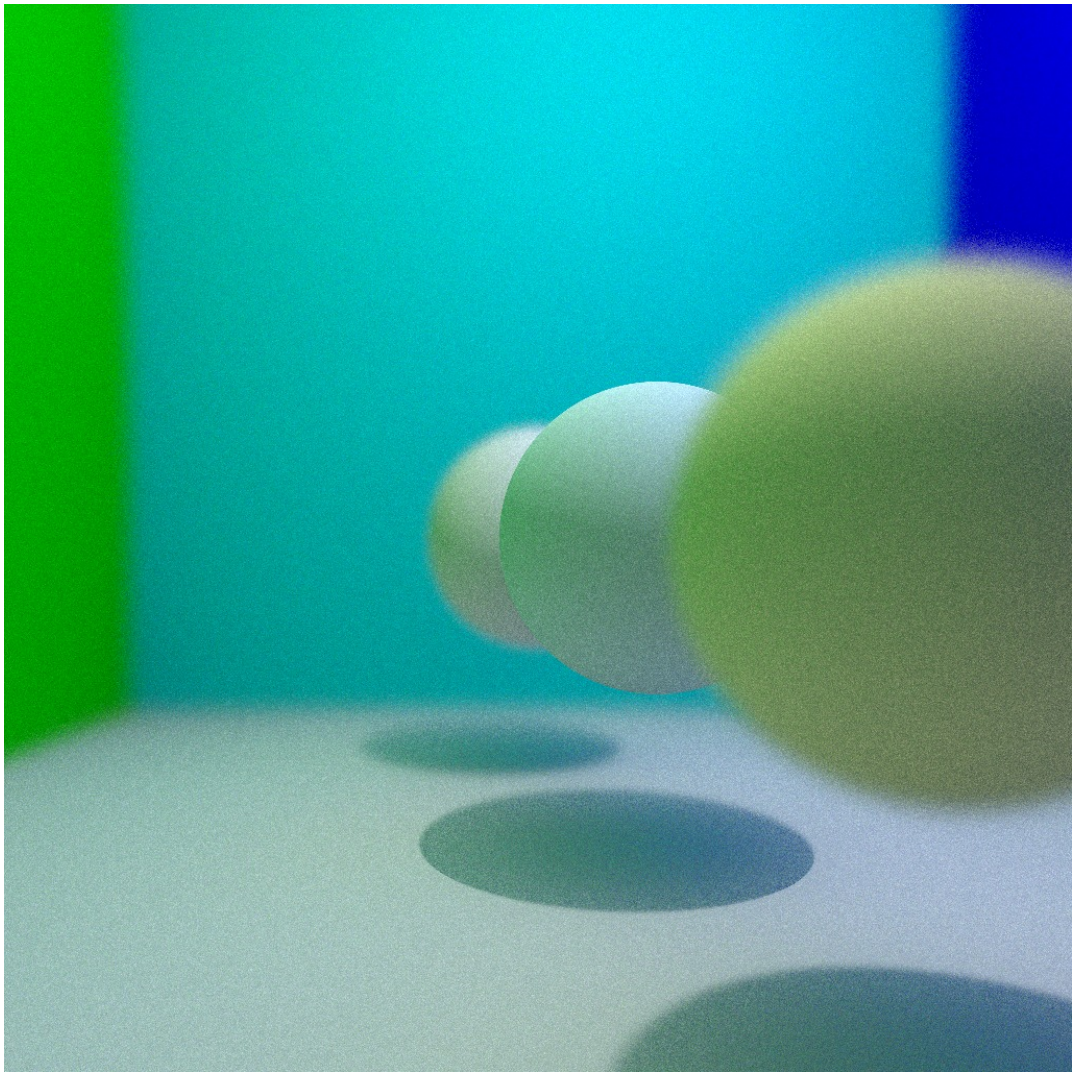

FIGURE 2.8 – Effet de profondeur pris en compte. 50 rayons et 3 rebonds utilisés.

2.8 Affichage d'un maillage

Jusqu'à présent, les objets affichés sont seulement des sphères. Nous améliorons notre ray tracer en implémentant les intersections entre un rayon et un triangle défini par 3 points. Grâce à cette nouvelle fonctionnalités, il est alors possible d'afficher des triangles et donc un maillage entier composé de triangles.

Pour définir comment sont agencés les triangles et les points d'un maillage, il est courant d'utiliser un fichier *.obj*. Un code a été donné pour lire les fichiers .obj et le traduire en un ensemble de triangles.

Pour le calcul des normales en tout point d'un maillage, on procède de la manière

suivante :

— Les normales pour chaque vertex du maillage sont données dans le fichier .obj
— Ayant un point d'intersection P dans un triangle, on calcule la normale du point P depuis les normales des vertices en donnant comme poids les coordonnées barycentriques de P.

Cette méthode de calcul des normales correspond à un lissage des normales. Le résultat est présenté en figure 2.9. L'anti-aliasing est activé et l'effet de profondeur de champs désactivé.

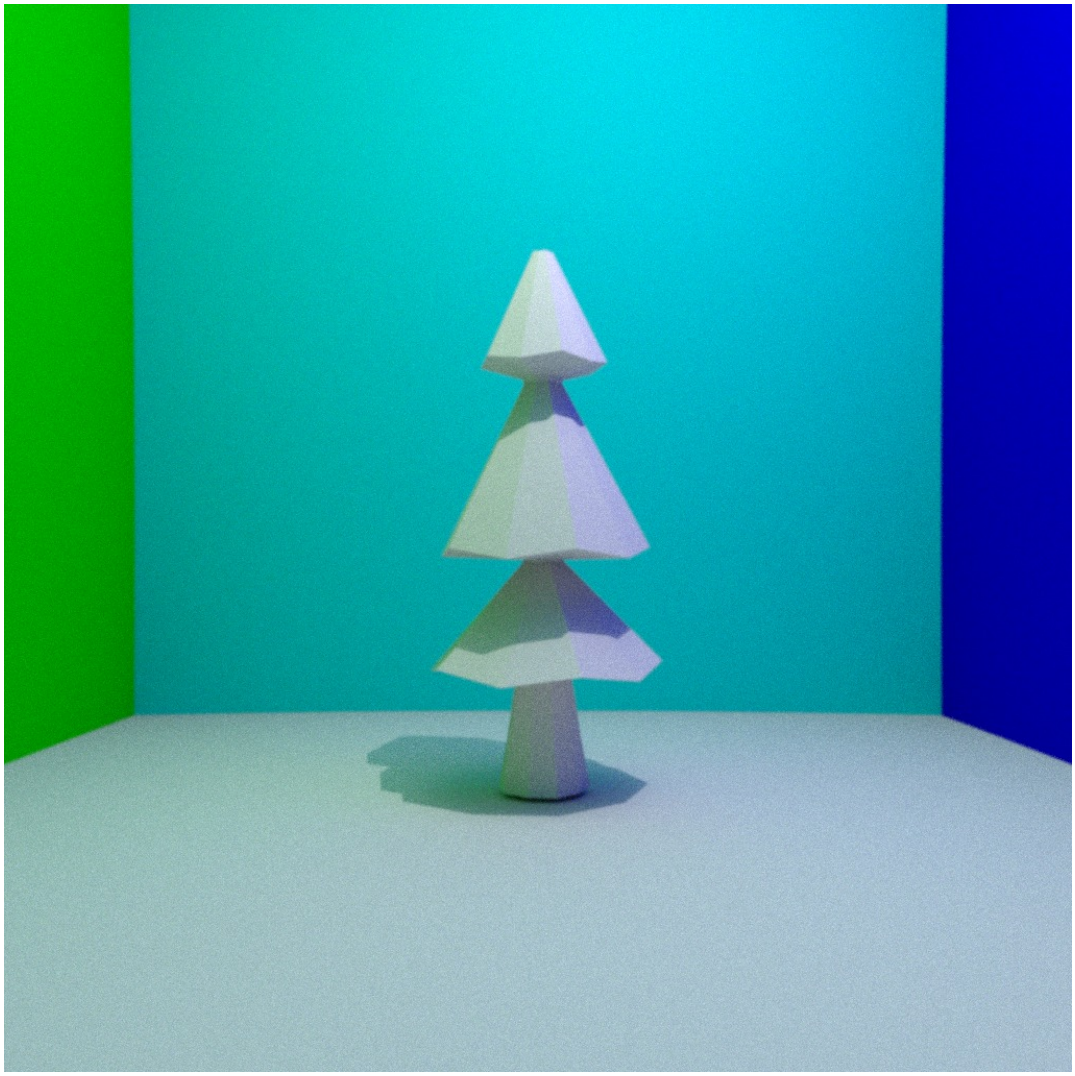

FIGURE 2.9 – Affichage d'un maillage avec lissage de Phong. 100 rayons et 3 rebonds utilisés.

En pratique, les structures d'accélération vues en cours ont été utilisées pour réduire le temps de calcul.

2.9 Les textures

2.9.1 Les textures procédurales

Une fois le maillage affiché, nous souhaitons rendre les textures des objets. Il s'agit pour chaque point du maillage qui est touché par un rayon, de renvoyer une certaine couleur. Dans un premier temps, les textures procédurales ont été utilisées. Il s'agit de moduler les composantes RGB en fonction de paramètres, comme les coordonnées des points de la surface. On obtient par exemple la figure 2.10.

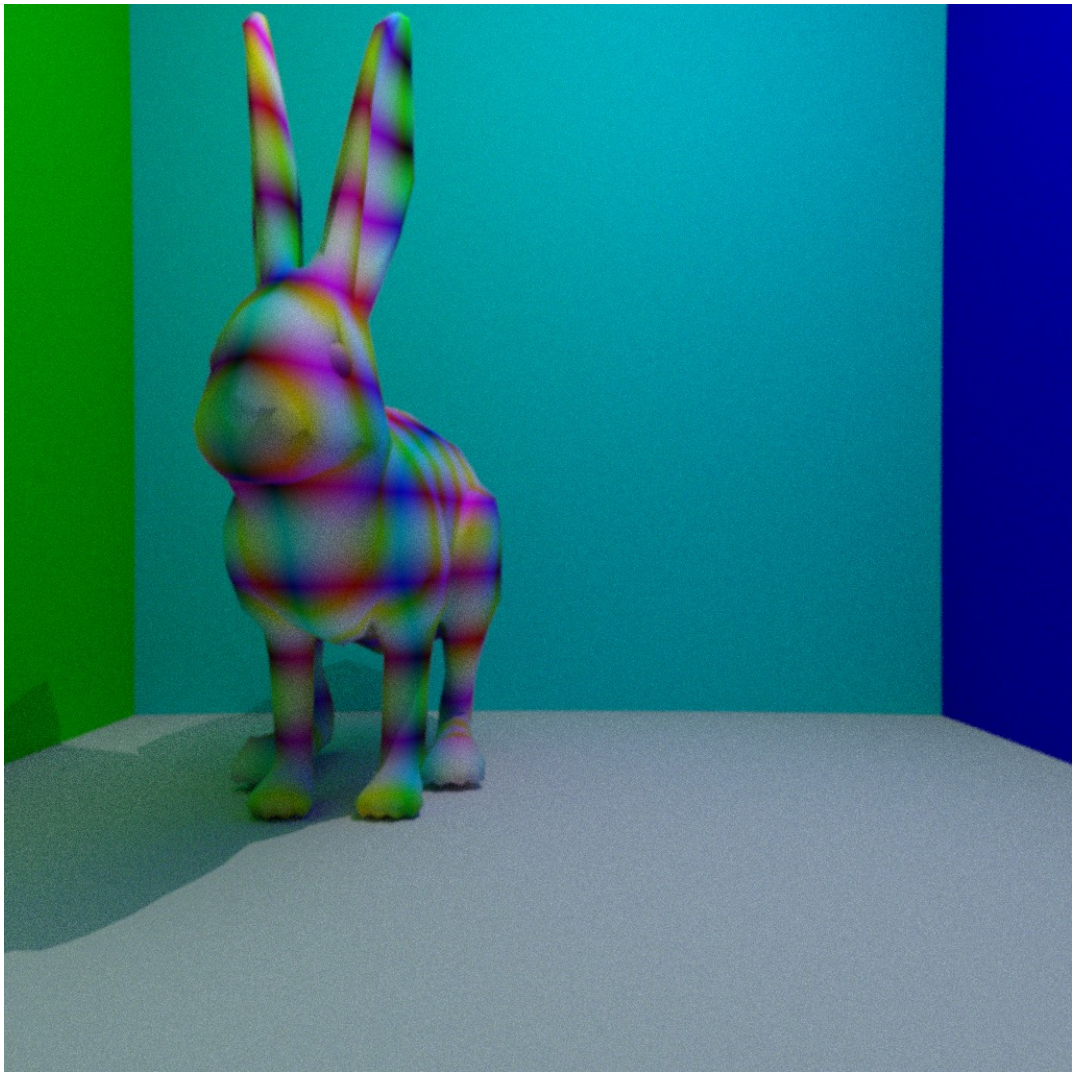

FIGURE 2.10 – Affichage d'un maillage avec texture procédurale

2.9.2 Les textures bitmap

Les fichiers .obj incluent une information sur les coordonnées de texture pour chaque vertex. Un fichier image (ou plusieurs) est joint au fichier .obj, correspondant à la texture. Connaissant la position dans le triangle correspondant de chaque point d'un maillage, il est possible de connaître la couleur du pixel correspondant. On obtient par exemple la figure 2.11.

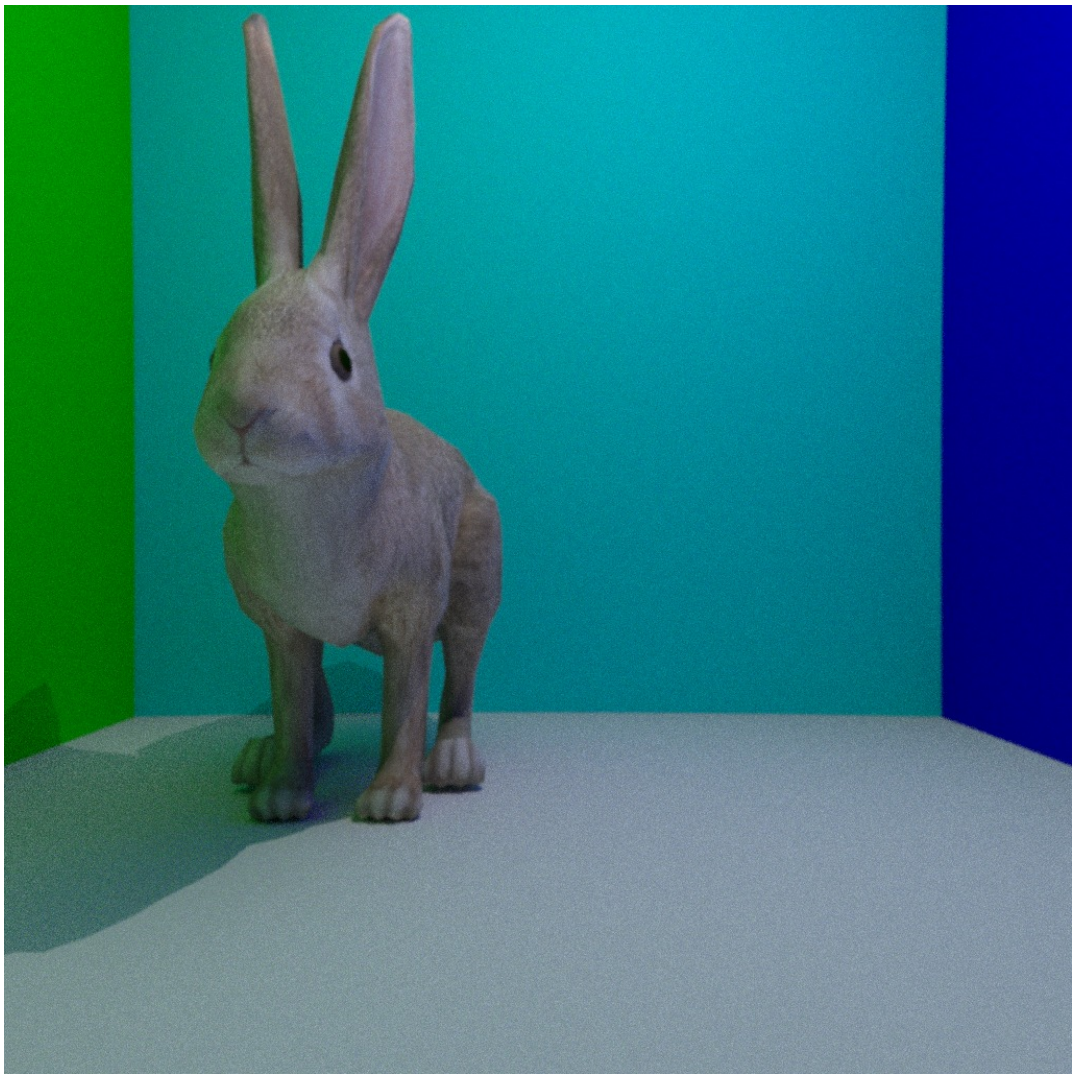

FIGURE 2.11 – Affichage d'un maillage avec texture bitmap

Dans le code, le chemin de la texture est rentré en dur.

2.10 Les lumières étendues

Nous implémentons les lumières étendues qui adoucissent les ombres. Au lieu d'utiliser des sources ponctuelles de lumières, les lumières sont maintenant des sphères. Le résultat est visible en figure 2.12.

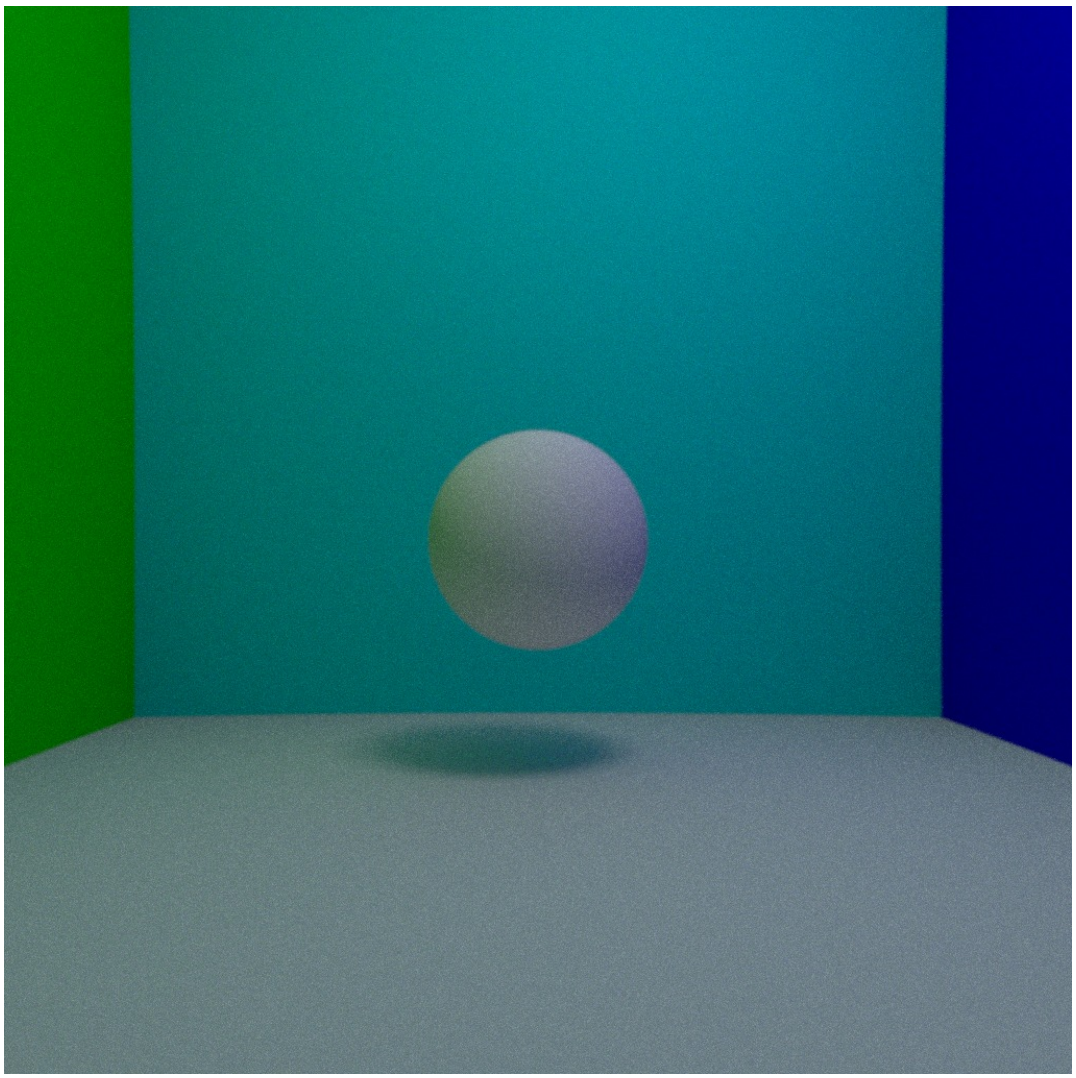

FIGURE 2.12 – Lumière étendue. 50 rayons et 3 rebonds utilisés.

2.11 Quelques résultats finaux

On présente dans cette dernière partie quelques rendus obtenus.

Pour le rendu figure 2.13, l'anti-aliasing, l'effet de profondeur et les lumières étendues

ont été utilisées. 50 rayons sont lancés par pixel, avec 3 rebonds. Le modèle du lapin fait 1910 faces et le rendu se fait en 12 minutes.

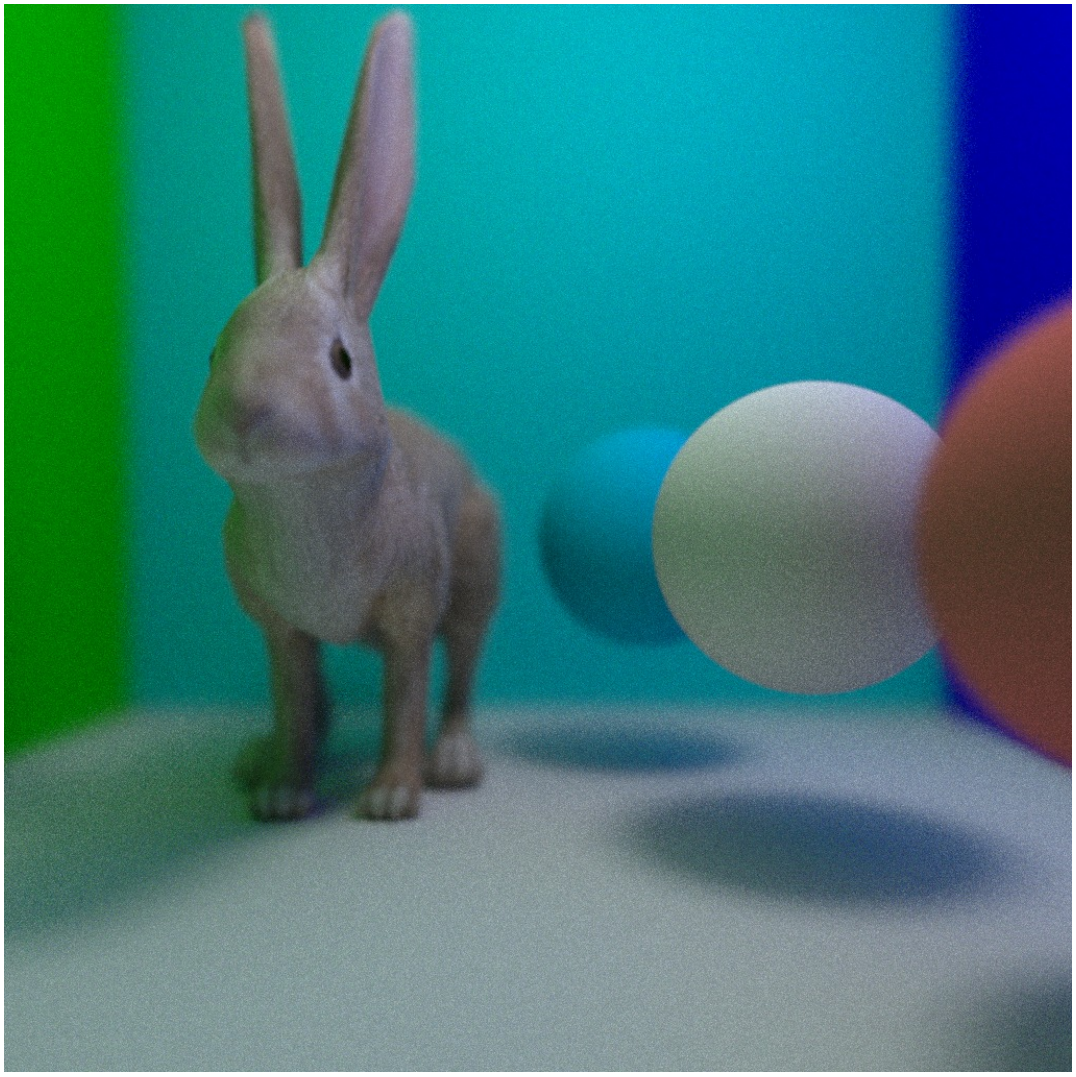

FIGURE 2.13 – Un joli lapin !

Pour le rendu de l'image en figure 2.14, une sphère transparente et le lapin n'est plus texturé mais remplacé par une surface spéculaire. La couleur spéculaire a été également définie à une teinte violette. La distance de focus de la caméra est réglée au niveau de la 2^{nde} sphère. 50 rayons et 4 rebonds ont été utilisés, pour un temps de rendu d'environ 11 minutes. Le lissage des normales est bien visible sur le lapin (autrement on verrait la réflexion sur les différentes faces).

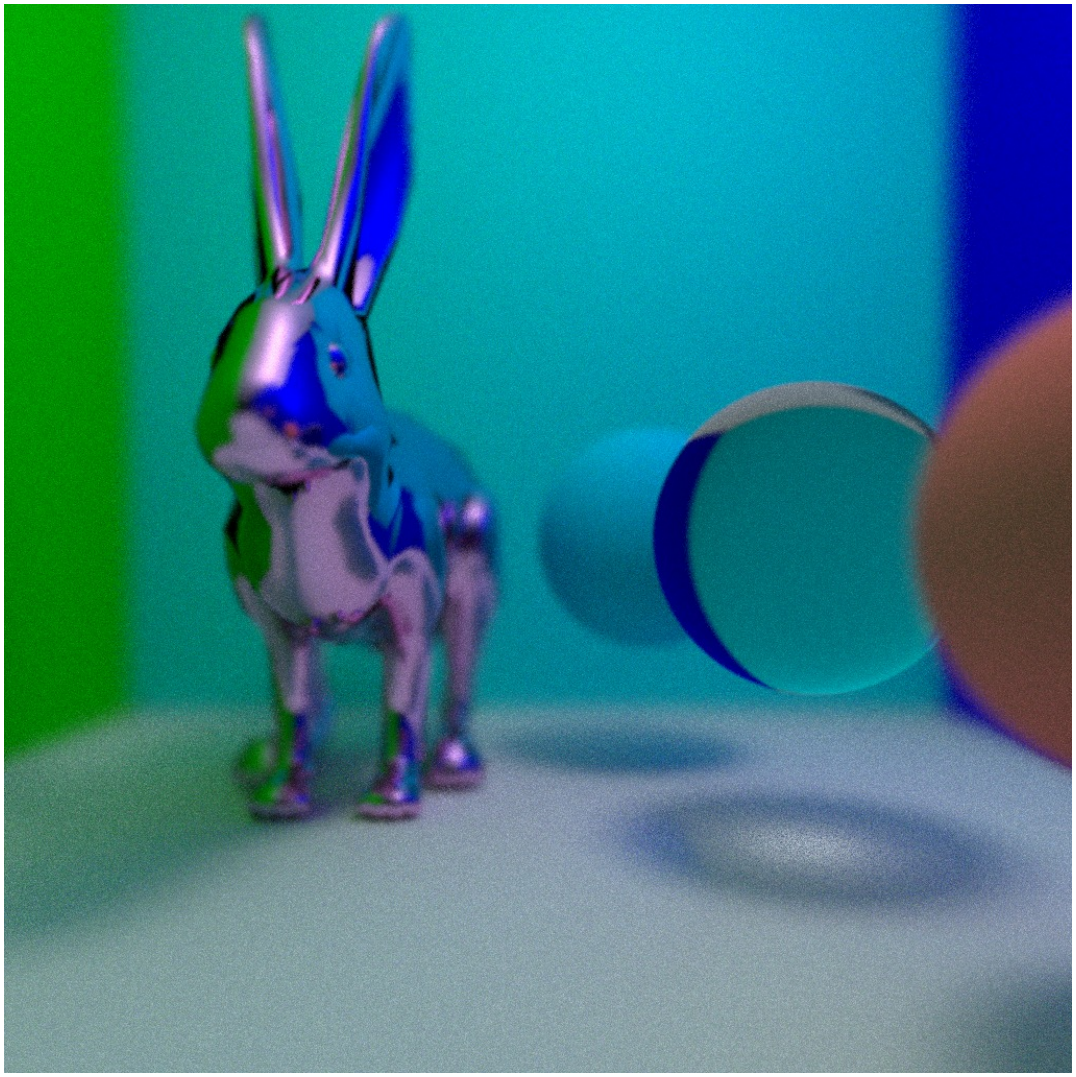

FIGURE 2.14 – Autre exemple de rendu

Conclusion

Les méthodes de ray tracing ont été vues dans ce cours. Ce rapport présente, sans rentrer dans les détails, leur implémentation ainsi que les résultats obtenus. Pour rappel, les différentes fonctionnalités ont été mises en place dans le ray tracer :

- L'anti-aliasing
- Les surfaces spéculaires et transparentes
- L'éclairage indirect avec les BRDFs
- L'effet de profondeur de champ
- L'affichage de sphères puis de maillages. Pour les maillages, des fichiers .obj sont lus. Des structures d'accélérations ont été utilisées, le lissage des normales ainsi que des textures bitmap sont rendues.
- Les lumières étendues

Des rendus sont donnés tout au long du rapport ainsi qu'à la fin, présentant toutes les fonctionnalités implémentées du ray tracer.