

MOS 2.2 : Informatique graphique

Programation d'un raytracer

Théo Cavignac

22 mars 2020

Introduction

Le *raytracing* ou «lancer de rayon» est une technique de rendu de scène 3D qui consiste à simuler le trajet de rayons lumineux hypothétiques arrivant sur le capteur d'une caméra virtuelle. Cette technique se basé sur l'optique géométrique permet d'obtenir des images très réaliste contenant des effets de lumière variés en suivant une méthode assez simple. Elle s'oppose au *rasterizing* qui est une technique plus légère consistant à projeter des triangles sur le capteur de la caméra virtuelle et à faire des approximation pour calculer l'éclairage.

Ce document rend compte de la réalisation d'un code de *raytracing* en C++ au cours du MOS 2.2. Il est rédigé en trois parties. La première partie présente les bases géométriques utilisé et les résultats obtenu au cours des premières étapes de développement. La partie 2 présente les effets de lumières possibles grace aux lancer de rayons stochastiques. La partie 3 présente l'ajout des maillages comme primitives, les structures d'accélération associé et l'implémentation du support des textures. En annexes se trouvent d'autres rendu produit avec des paramètres plus lourds (et donc des résultats plus propres).

1 Base du raytracing

Les premières étapes (tags step1 à steps_5_6) du développement consistent à implémenter la capacité à rendre des sphères de couleurs ombré.

La méthode consiste à lancer des rayons de lumière pour chaque pixel dont l'origine est le centre de la caméra et la direction est donnée par la position du pixel sur le «capteur».

On commence par détecter une sphère placé devant la caméra virtuelle. Pour cela on implémente une fonction permettant de déterminer l'existence d'une intersection entre une sphère et un rayon. En présence d'une telle intersection on rend le pixel correspondant blanc. Sinon on rend le pixel noir. On obtient la figure 1.

On va ensuite utiliser la position de l'intersection pour calculer la luminosité réfléchié par la surface vers la camera. Pour cela on considère que la surface est blanche et parfaitement diffusive, ce qui correspond à une surface ayant une microstructure très rugueuse comme du plâtre. La formule correspond est implémenté en C++ dans la figure 2.

Cette formule exprime deux fait important :

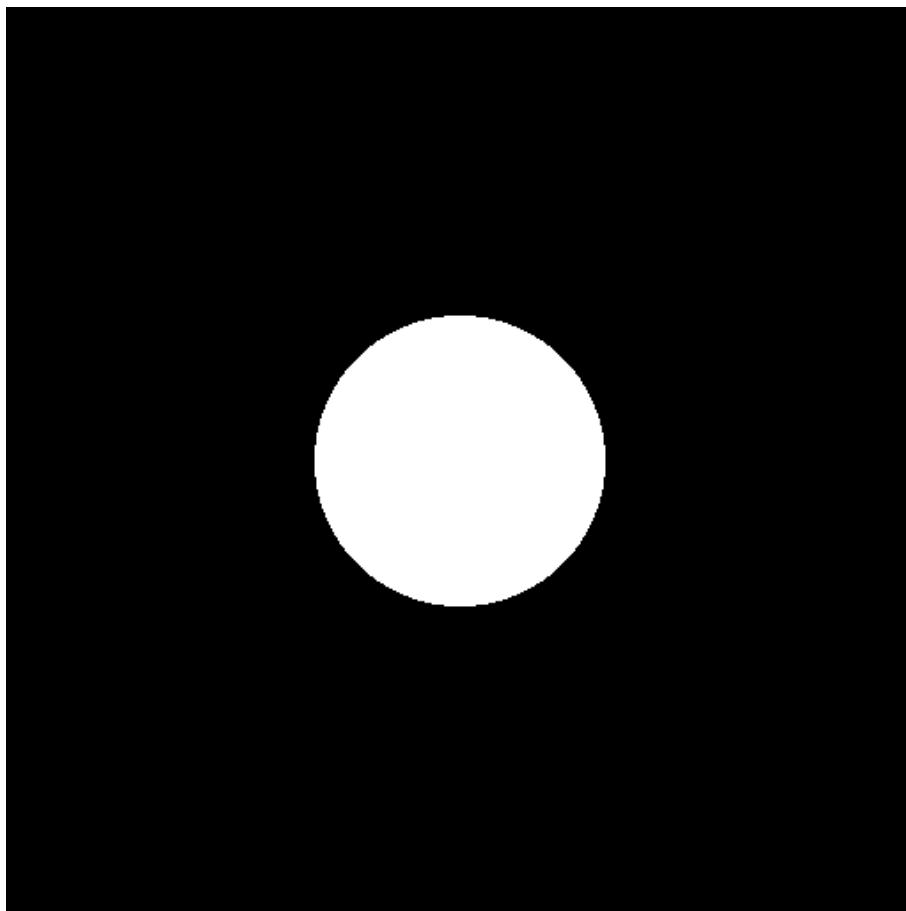


FIGURE 1 – Détection d'une sphère devant la camera

```

// s est une instance de Sphre et r une instance de Ray
// la méthode intersection retourne un vecteur correspond
// à la position de l'intersection
Vec p = s.intersection(r);

// n est le vecteur normal à la surface au point
// d'intersection
Vec n = (p - s.origin()).normalized();

Vec vl = light - p;

// I est l'intensité de lumière reçu pas la caméra
double I = I0 * std::max(0.0, n.dot(vl.normalized()))
    / vl.norm_sq();

```

FIGURE 2 – Calcul de l'intensité reçu par la caméra après réflexion sur la surface de la sphère

- si le rayon de lumière provenant de la source éclaire la surface de façon tangente, aucune lumière n'est réfléchi, si le rayon est normal à la surface un maximum de lumière est réfléchi
- l'intensité rééfléchie ne dépend pas de la direction du rayon allant vers la caméra car la lumière est réfléchie de façon homogène dans toutes les directions (sauf si le rayon est derrière la surface)

Le résultat est visible dans la figure 3

L'étape suivante consiste à refactoriser le code pour :

- faire le rendu de multiples spheres
- attribuer une couleur au spheres

Pour cela on a défini une classe **Sphere** qui est définie par un centre, un rayon et un albedo (un vecteur 3D correspondant à la couleur en format RGB). On a également une classe **Scene** qui contient plusieurs instances de **Sphere** ainsi qu'une instance de **Light**. Cette classe est chargée de calculer la couleur d'un rayon donné avec une méthode `get_color`. Cette méthode restera au centre du programme tout au long de son développement.

On définit une scène dans laquelle cinq sphères de rayon 1000 formeront des murs encadrant un sphère de rayon 10 au centre de la sphère. Pour supporter les couleurs on réutilise la formule donnée dans la figure 2 mais multipliant I par l'albedo de la sphère pour obtenir le vecteur couleur du rayon final.

On obtient la figure 4.

On applique aussi pour la première fois une transformation gamma inverse qui permet de rendre compte de la non linéarité de l'œil humain dans la perception de la lumière ce qui a pour effet de diminuer les contrastes de l'image.

Enfin on ajoute les ombres en cherchant si un objet se trouve sur le chemin entre la source de lumière et le point d'intersection entre la surface et le rayon. Si un tel objet existe le point en question n'est pas (directement) éclairé et apparaît donc noir. On obtient la figure 5.

Plus tard l'albedo de **Sphere** est remplacé par une instance d'une classe **Material** qui

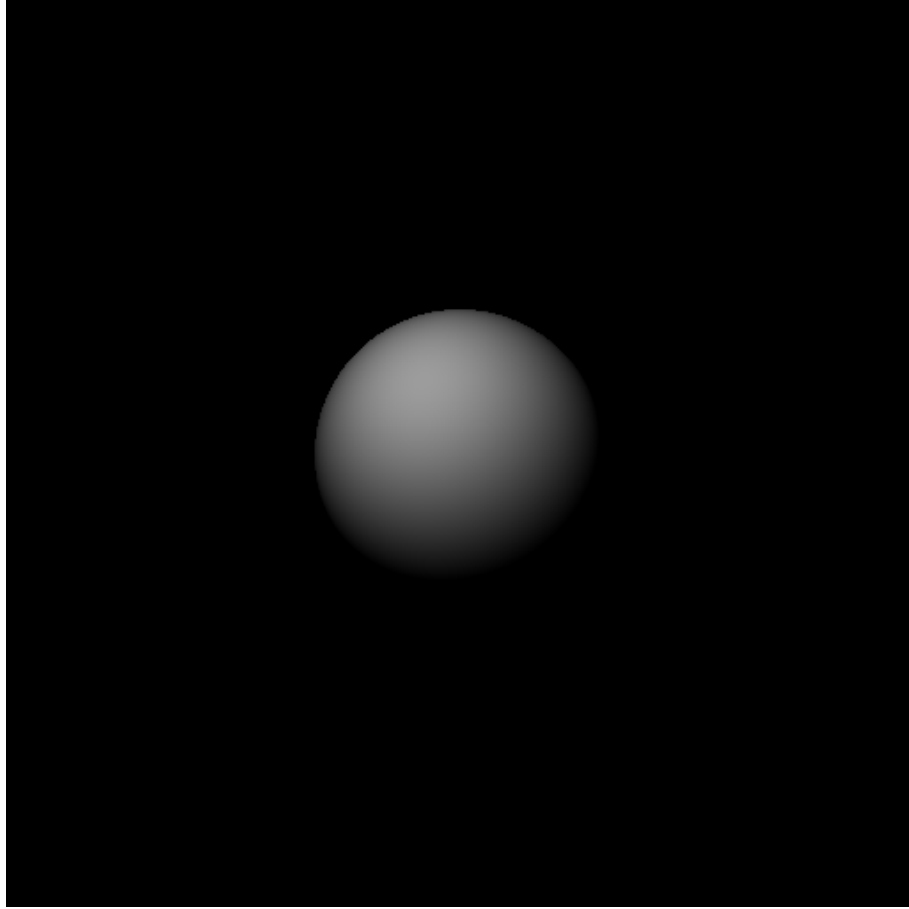


FIGURE 3 – Sphere diffusive ombré

servira de classe mère pour différent type de matériaux correspondant à différentes BRDF. On ajoute notamment un matériaux réfléchissant (d’abord représenté par un drapeau booléen `mirror` dans la classe `Material` puis par une classe fille spécialisé `Reflective` cf. step7) dont la BRDF consiste en un dirac dans la direction du rayon réfléchi, c’est à dire que toute la lumière reçu est réfléchi dans la même direction.

Cet effet est le premier qui traduit le phénomène d’éclairage indirecte que l’on généralise dans l’étape suivante. On obtient la figure 6.

2 Effets de lumière réaliste

Dans cette partie (tags `steps_5_6+` à `step7`) on introduit le lancer de rayon stochastique pour réaliser divers effets de lumière réalistes tel que l’éclairage indirecte qui permet les ombres douces, la transparence, les sources étendues et autres.

Le premier effet est la transparence qui consiste à relancer un rayon depuis le premier point d’intersection dans une direction donnée par la loi de Descartes. Comme il y a dans la plupart des cas un rayon réfléchi et un rayon réfracté on choisi aléatoire l’une de ces deux directions pour lancer le nouveau rayon sans avoir une croissance exponentielle du nombre

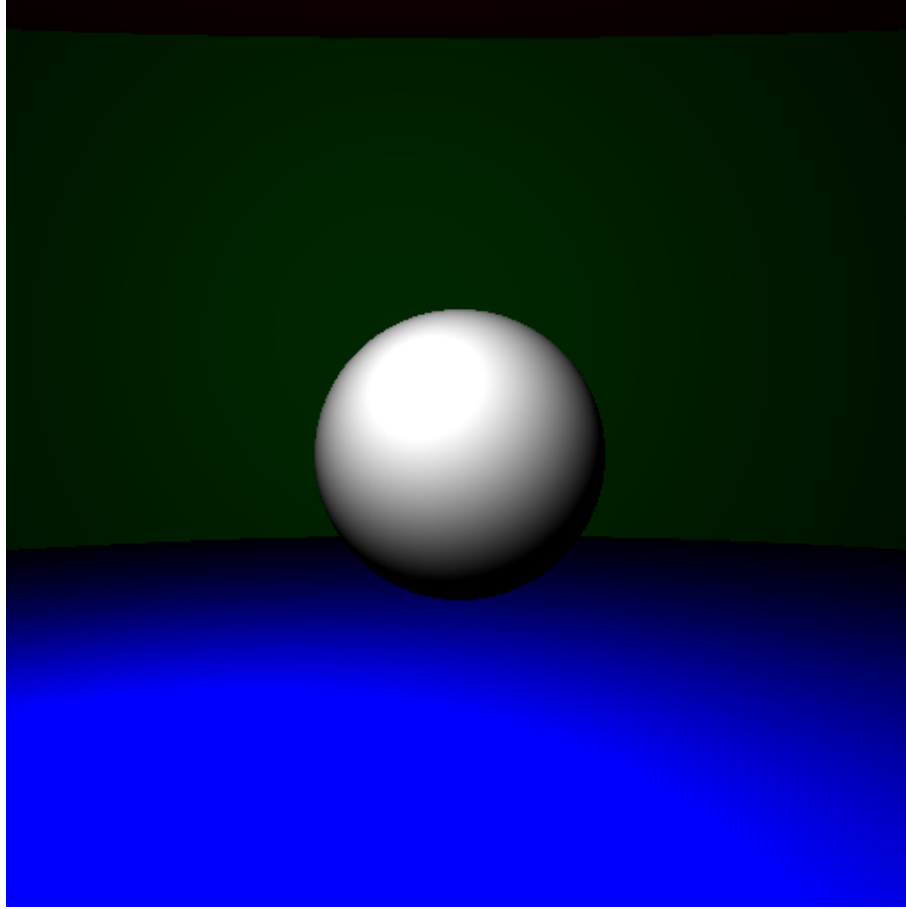


FIGURE 4 – Support d’une scene contenant plusieurs spheres coloré

de rayon. La probabilité de chaque rayon est donnée par les facteurs de reflexion et de transmission en puissance approximé par la formule de shlick. Pour lisser cet effet aléatoire on lance donc plusieurs rayons pour chaque pixel et on moyenne les intensités obtenus. Pour lancer le nouveau rayon `get_color` s’appelle récursivement avec une profondeur de récursion finie. A chaque fois que l’intersection tombe sur un surface transparente, un rayon réfléchi ou réfracté est relancé et ainsi on construit la lumière finale vue par la caméra. L’aspect stochastique fait apparaitre un peu de bruit dans l’image qui est lissé en lançant plus de rayons par pixel. On peut voir cet effet dans la figure 7.

On ajoute ensuite l’éclairage indirecte. Cela consiste à relancer un rayon également pour les surfaces diffuse et a utiliser la couleur de ce rayon comme éclairage supplémentaire (en plus de l’éclairage directe). C’est ici que la création des classes `Material` commence à faire sens car elle permet d’unifier la réflexion totale des surfaces réfléchissantes et la réflexion stochastique des surfaces diffuse au travers d’une méthode `reflex_dir` qui associe à un rayon émergent et une normale, un rayon incident. Dans le cas de la réflexion totale cette méthode est déterministe. En revanche dans le cas des surfaces diffuse elle est stochastique et renvoie une direction selon un distribution qui permet toute les directions de l’hémisphère en favorisant les directions proches de la normale. Cette fonction ne dépend pas du rayon émergent car la surface est parfaitement diffuse. La figure ?? montre la fonction `random_cos`

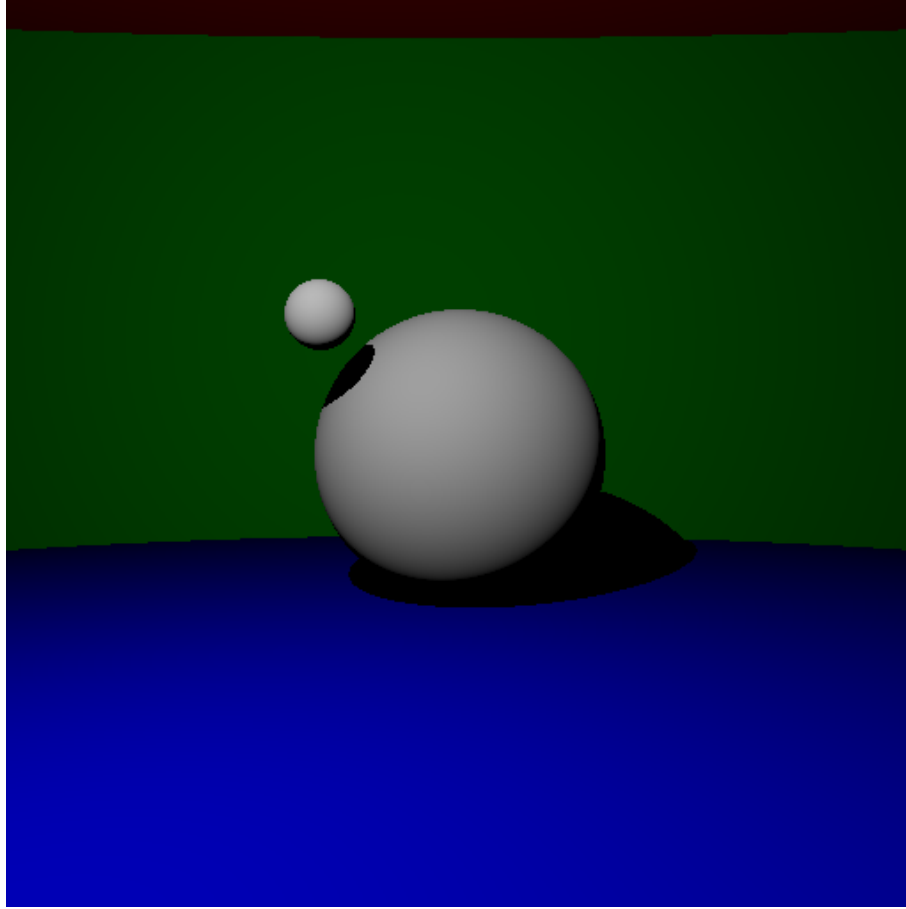


FIGURE 5 – Sphères avec des ombres portés

qui permet de générer une direction et fourni également la probabilité de cette direction particulière qui sert de facteur dans la méthode de Monte-Carlo qui se cache derrière cette méthode.

Concretement l'effet de cette modification est que les ombres ne sont pas totalement noires, car elles reçoivent un peu de lumière réfléchie sur d'autres objets.

Un autre effet cousin de celui-ci est d'étendre la source de lumière qui était jusqu'ici ponctuelle. On considère maintenant une sphère qui emet de façon homogène une lumière blanche. Cette sphère est représenté par la classe `Light` qui contient maintenant un rayon.

Lorsque l'on calcule l'éclairage directe on utilise `random_cos` pour choisir un point sur la surface de la sphere lumière. Ce point sert ensuite de source ponctuelle pour le point courant. La luminausité du point correspond à la luminosité nominale de la source pondéré par le cosinus de l'angle entre la normale au point sur la sphère lumineuse et la direction vers l'objet éclairé.

Le résultat de ces deux ajouts est visible dans la figure 8.

Enfin les derniers effets ajouté utilisant le lancer de rayons stochastique sont l'effet de profondeur de champ qui consiste a simuler l'effet de l'optique d'une caméra sur a netteté des différents plan de l'image et l'anti-crénelage.

Ces deux effets consistent à perturber le rayon initiale. La profondeur de champ est

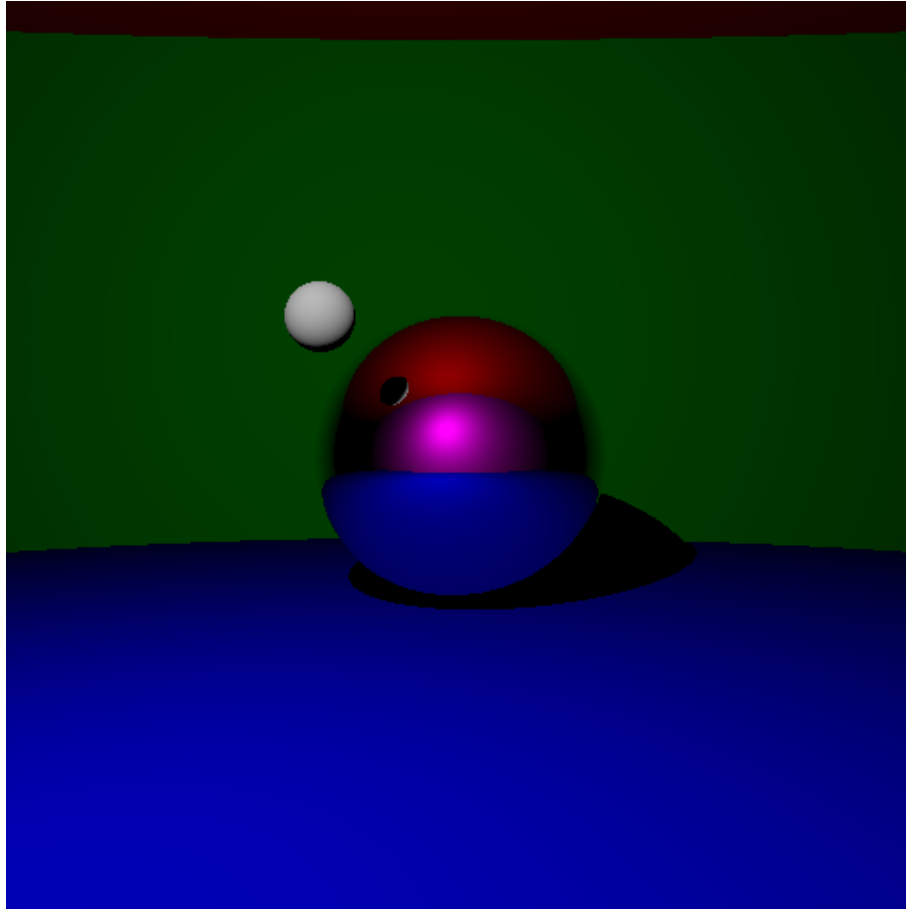


FIGURE 6 – Introduction des surfaces réfléchissantes

obtenu en perturbant l'origine du rayon, mais en choisissant la direction de sorte que quelle que soit l'origine les rayons convergent au même point d'un plan parallèle au plan du capteur. Ce plan est défini par la longueur de champ que l'on fixe pour la figure ?? à 55 unités, soit le plan dans lequel se trouve la sphère miroir.

L'anti-crénelage consiste à perturber légèrement la direction du rayon de sorte qu'il ne passe pas systématiquement au même endroit du pixel. On obtient donc des transitions plus douces entre les surfaces qui font disparaître les pixels.

3 Maillage et structures d'accélération

Dans cette partie on ajoute le support des maillages ainsi qu'un certain nombre d'améliorations du code, comprenant des refactorisations facilitant la maintenabilité et le débogage.

Entre autre la possibilité de spécifier tous les paramètres du rendu (position de la caméra, composition de la scène, paramètres de convergence tel que le nombre de rayons par pixel etc...).

TODO : Rendu du maillage, structure d'accélération, textures

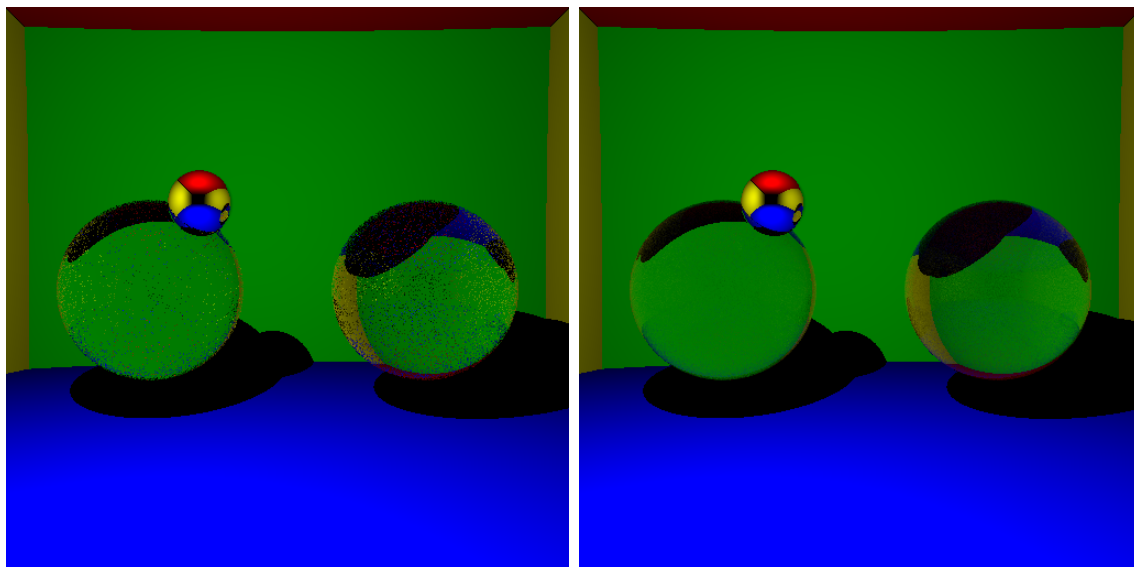


FIGURE 7 – Rendu avec une sphère transparente en lançant 2 rayons par pixel (droite) et 30 rayons par pixel (gauche)

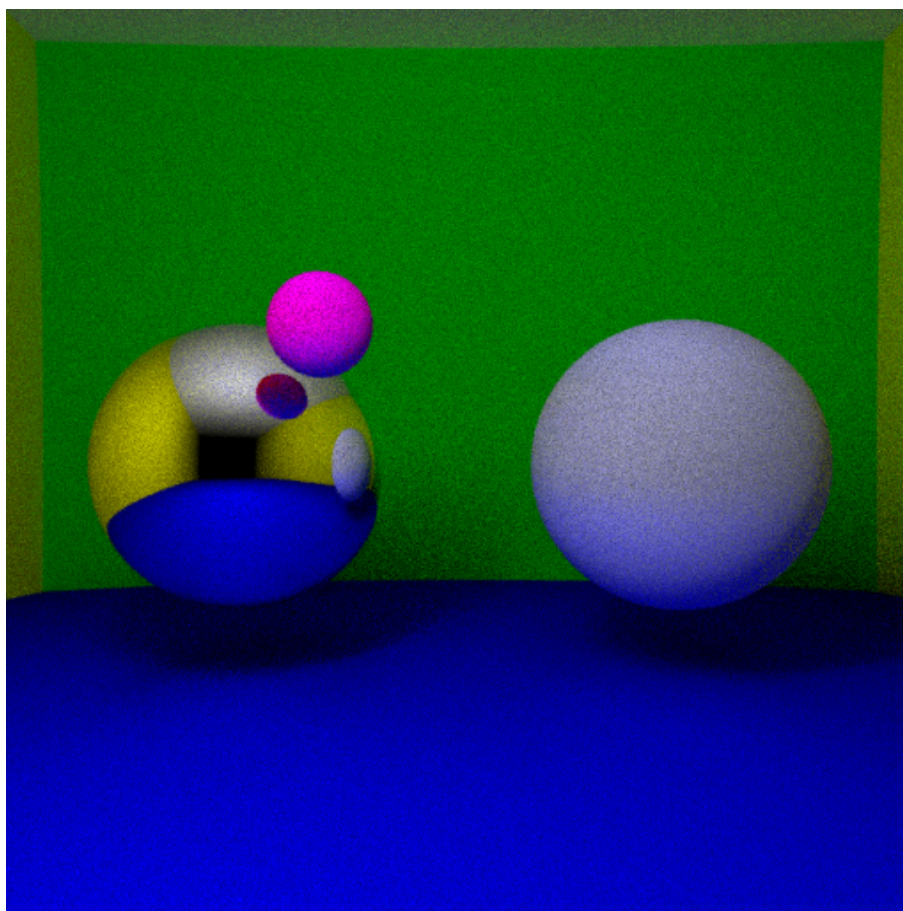


FIGURE 8 – Ajout de sources étendu et d'éclairage indirecte

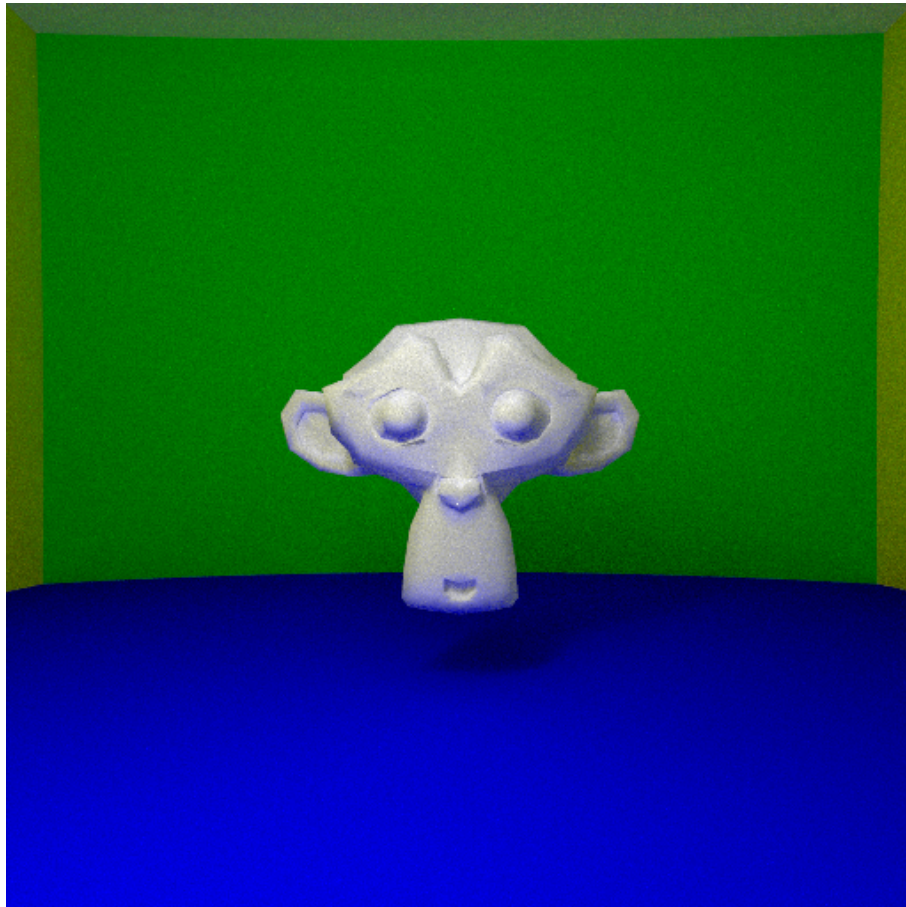


FIGURE 9 – Rendu de maillages simple

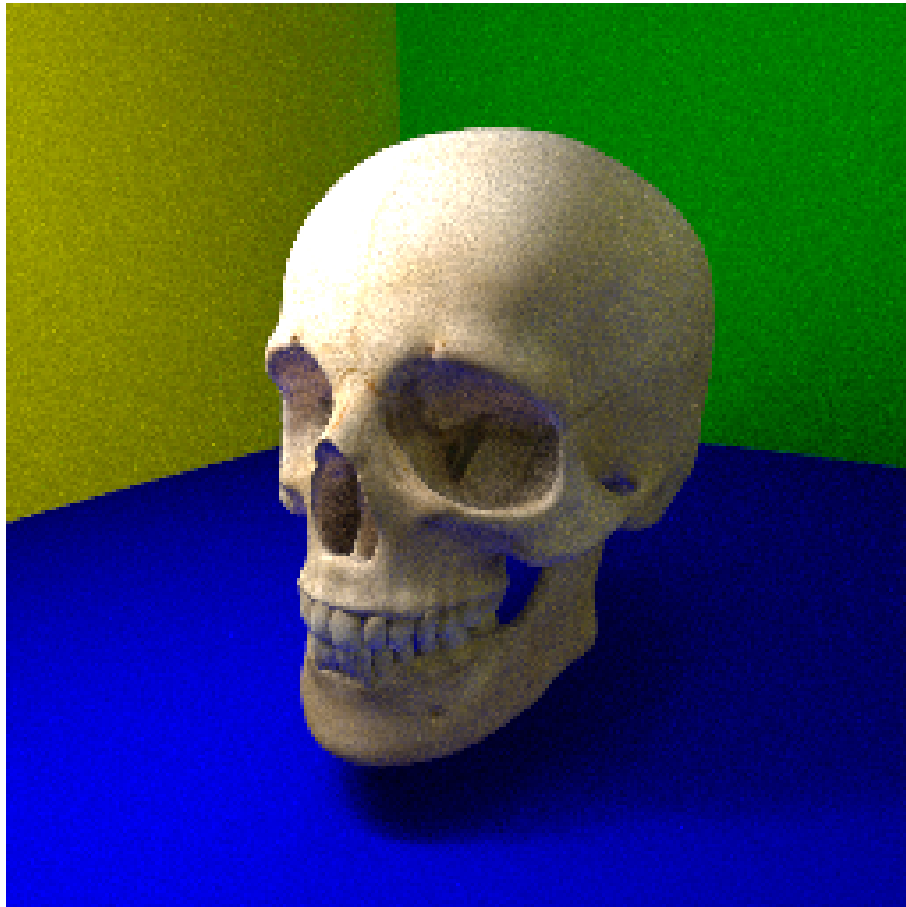


FIGURE 10 – Rendu d'un maillage complexe avec une texture