ÉCOLE CENTRALE LYON

Rapport de fin d'action de formation
Raytracing

S9 MOS 2.2 2019-2020

Informatique Graphique

Enseignant : Nicolas Bonneel

Prégniard Lancelot

21/03/2020

Sommaire.

Table des figures.

I. Introduction

Le but de ce court rapport est d'exposer les différentes méthodes développées au cours de cette action de formation ainsi que les résultats obtenus. Nous n'aborderons que le Raytracing, ou lancer de rayons, sur des sphères.

Le Ray-tracing est la version la plus fidèle du rendu informatique. Il s'agit d'une méthode lente et réaliste qui permet d'atteindre des images d'une qualité très proche du photoréalisme, voire même d'atteindre l'exactitude physique. Les applications du Ray-tracing sont multiples, notamment dans le cinéma pour les images de synthèse ou en architecture pour la simulation de l'éclairage.

II. Initialisation

a. Principes de base

Nous appliquons ici la méthode du Backward Raytracing, fondé sur le principe de Helmhotz qui stipule que l'on peut suivre les rayons de la lumière en sens inverse et obtenir le même résultat. Nous enverrons ainsi les rayons depuis la caméra plutôt que depuis les sources de lumière, réduisant ainsi fortement le temps de calcul car s'affranchissant de tous rayons n'atteignant pas la caméra.

La modélisation de base de la caméra est celle de la caméra sténopé (pinhole) dont le principe remonte a la fin du 19ème siècle. Il s'agit d'un modèle simpliste car ne prend pas en compte les effets de profondeur de champ et autres effets liés à une ouverture circulaire comme sur un système optique réel (oeil, objectif d'APN).

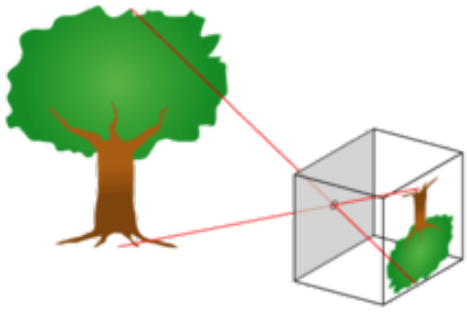

Figure 1 - pinhole camera, source: wikipédia

Ainsi, des rayons sont envoyés depuis la caméra à travers chaque pixel d'un écran positionné à une certaine distance du trou. Le principe de base peut se résumer à l'aide du schéma suivant :

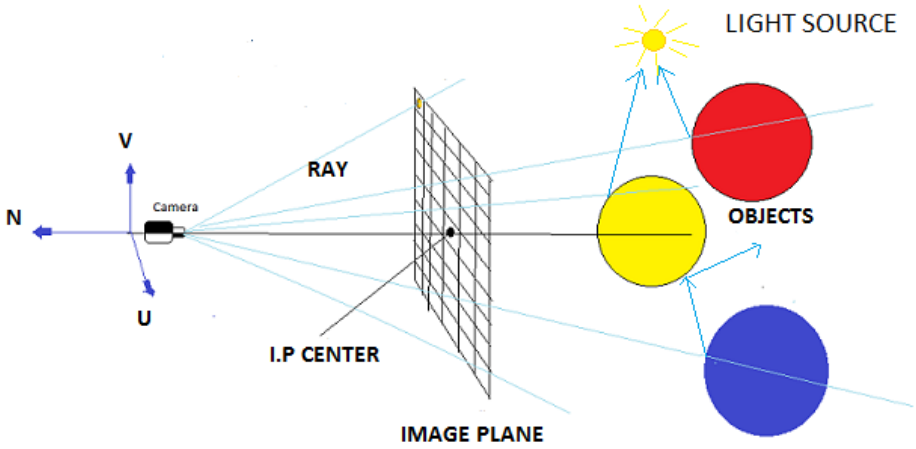

Figure 2 - schéma de base du Raytracing, source: ray-tracing-conept.blogspot.com

b. Éléments de la scène

Les rayons sont représentés par des demi-droites de centre 0 et de direction u.
La scène est composée :

- De sphères pleines (à surface diffuse, puis miroir ou transparentes).
- De plans, représentés par des sphères de très grand rayon.
- De sources de lumière (ponctuelle, puis étendues).

c. Choix de programmation

Le Raytracer est programmé objet en C++. Les différentes classes que nous utiliserons sont :

- Les vecteurs : classe **Vector(x, y, z)**
- Les rayons : classe **Ray(origine, direction)**
- Les sphères : classe **Sphere(centre, rayon, albedo, miroir, transparent)** avec albedo le pouvoir réfléchissant de la sphère, ici utilisé comme le terme de couleur de la sphère, miroir et transparent les booléens indiquant la nature de la sphère.
- La scène : classe **Scene(spheres, position_lumiere, intensité_lumière, sphère_lumière)** avec sphère_lumière si éclairage étendu, et position_lumière si éclairage ponctuel.

L'IDE choisi est Microsoft VS Code, pour ses plug-ins facilitant le compilage, l'éxécution et la syntaxe du code C++.

III. Premiers développements

a. Calcul d'intersection entre les rayons et les objets de la scène

La première étape consiste à calculer les intersections entre les rayons et les sphères de la scène. On montre par un simple calcul géométrique que cela revient à résoudre l'équation du second degré :

$$t^2 + 2 * t * < u, C - O > + \left|\left|C - O\right|\right|^2 - R^2 = 0$$

Avec :

$t : distance\ du\ point\ d'intersection$
$u : direction\ unitaire\ du\ rayon$
$C : centre\ de\ la\ sphère$
$O : origine\ du\ rayon$
$R : rayon\ de\ la\ sphère$

Le calcul de cette intersection est gérée par la routine intersect qui renvoie un booléen si intersection, et modifie la valeur du paramètre t, du point d'intersection P et de la normale N au point d'intersection.

Un premier test, ou l'on considère une sphère centrale de rayon R = 20 et qui renvoie un pixel blanc si intersection, et un pixel noir sinon, renvoie l'image suivante :

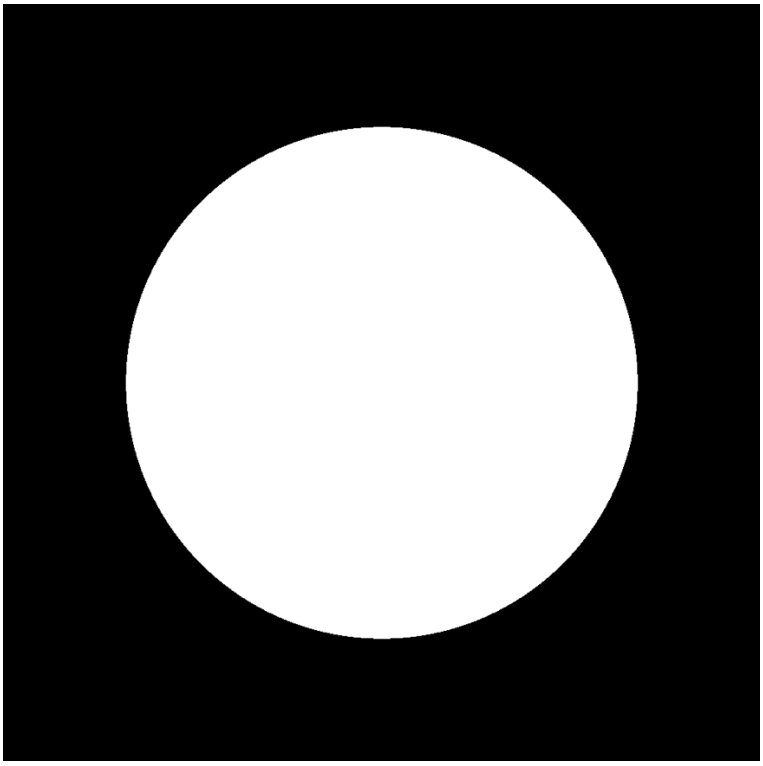

Figure 3 - intersection simple

b. Calcul de l'éclairage

Il convient maintenant d'ajouter une source de lumière ponctuelle, et de calculer l'éclairage de la sphère. Pour chaque point intersecté P, le calcul de l'éclairage se fait par la formule suivante, sous l'hypothèse de surfaces isotropes réflectantes :

$$i = \frac{\rho}{\pi} * I * \frac{< N, L >}{d^2}$$

Avec :

ρ : $albedo\ (réflectance)\ de\ la\ sphère$
I : $intensité\ de\ la\ source\ ponctuelle$
N : $vecteur\ normal\ au\ point\ d'intersection$
L : $vecteur\ partant\ de\ P\ et\ orienté\ vers\ la\ lumière$
d : $distance\ PL$

On obtient ainsi, pour la même sphère, avec une intensité I = 10 000 000 :

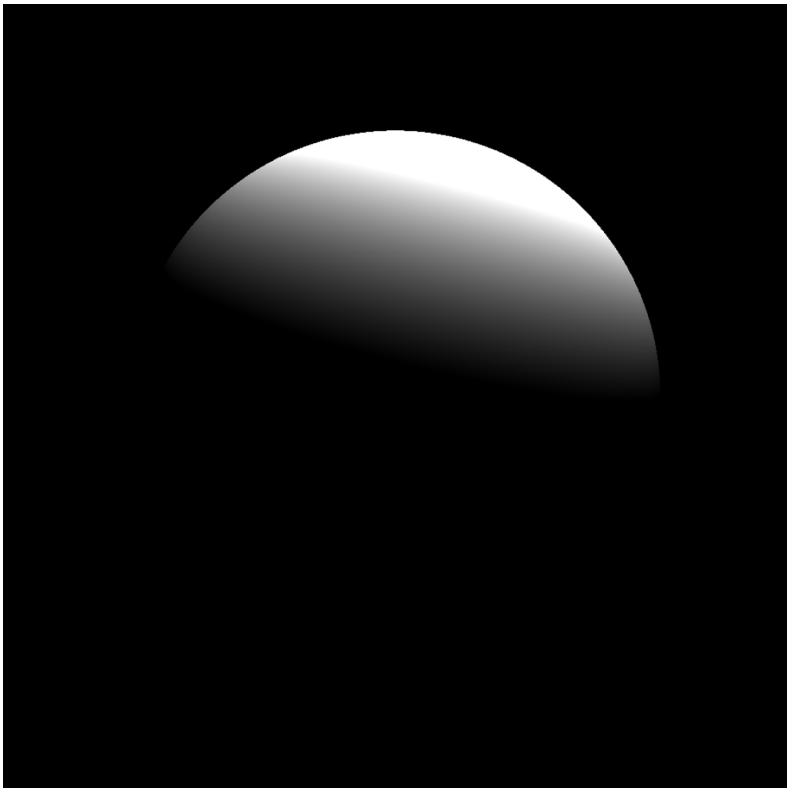

Figure 4 - ajout de l'éclaire direct

Puis, en ajoutant des murs, un sol et un plafond, et des couleurs arbitraires :

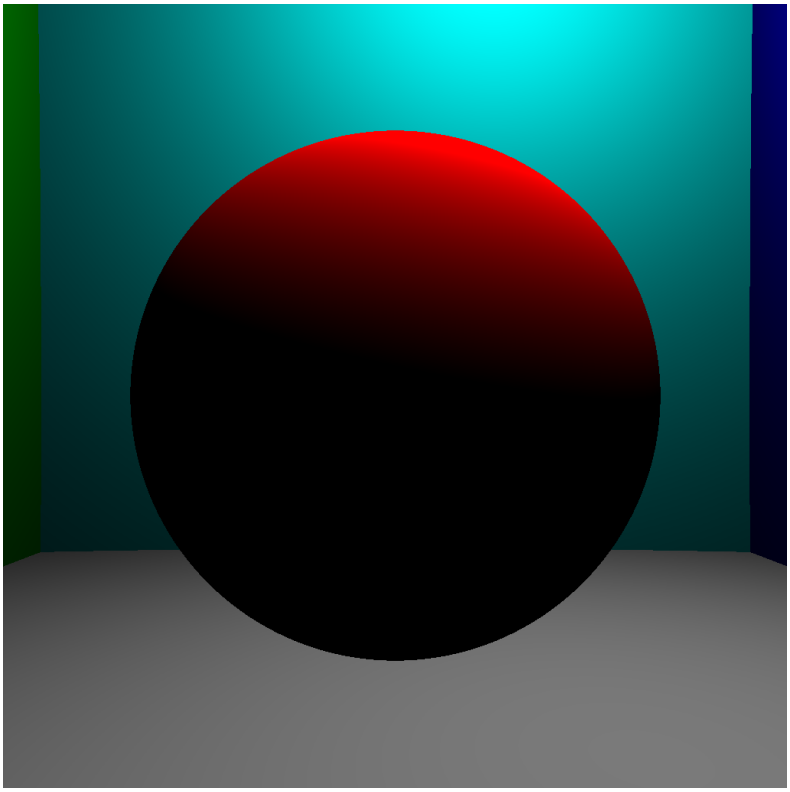

Figure 5 - ajout de sphères de grand rayon

On remarque que l'image n'est pas réaliste car elle ne prend pas en compte l'éclairage indirect (réflexion de la lumière sur le sol et les murs), ni les ombres.

c. Ajout des ombres portées

Pour prendre en compte les ombres portées, on utilise le principe du Path-Tracing, qui consiste à lancer un nouveau rayon depuis le point d'intersection précédent. Ce nouveau rayon, dirigé vers la lumière, donne l'information de position du point d'intersection par rapport à la lumière, et à un éventuel objet rencontré sur le chemin de la lumière.

En modifiant le code du Raytracer de telle manière à réaliser ce test, on obtient l'image suivante :

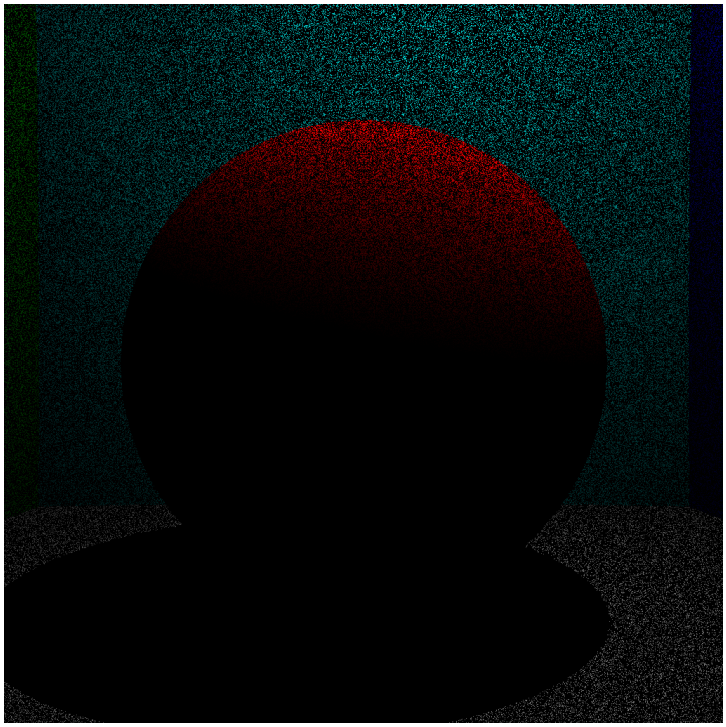

Figure 6 - ajout des ombres portées

On remarque ici que les ombres portées sont bien affichées, mais que l'image est extrêmement bruitée. Ceci est dû à un problème de précision numérique lors du calcul d'intersection. En effet, il arrive que le rayon envoyé à partir du premier point d'intersection se considère lui même comme le point suivant d'intersection, masquant ainsi le pixel alors qu'il ne devrait pas l'être. Pour corriger ceci, on "décolle" un peu le point de départ du second rayon, par un facteur 0.01 sur la normale. En corrigeant ce bug, on obtient :

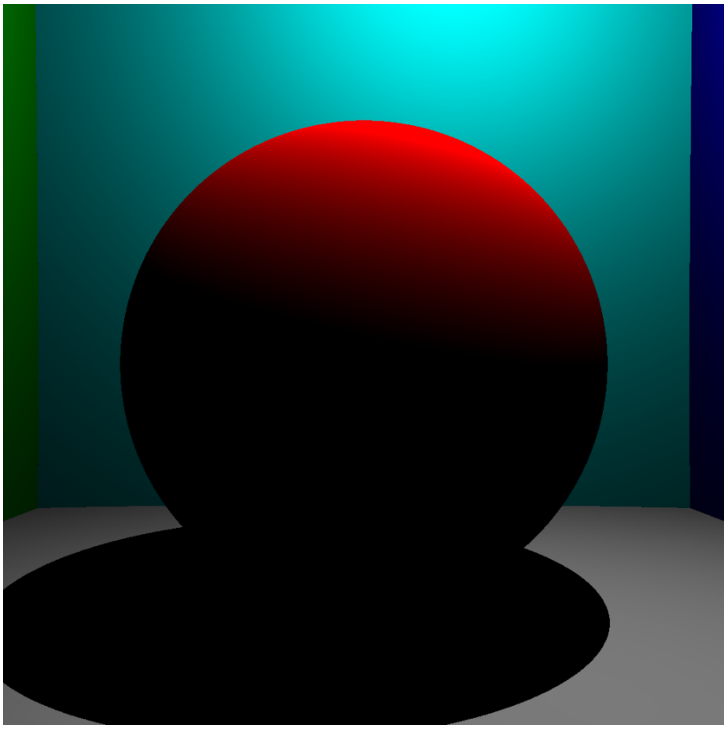

Figure 7 - ombres portées, corrigé

On remarque ici que l'image n'est toujours pas très réaliste, car l'ombre est trop nette. En réalité, l'ombre est plus diffuse.

d. Correction Gamma

Nous apportons ici une petite correction appelée correction Gamma, qui consiste à mettre les intensités RGB à la puissance $\frac{1}{2,2}$, pour rendre compte de la capacité de l'oeil humain à être plus sensible aux intensités faibles. La valeur 2,2 est un vestige hérité des écrans cathodiques.

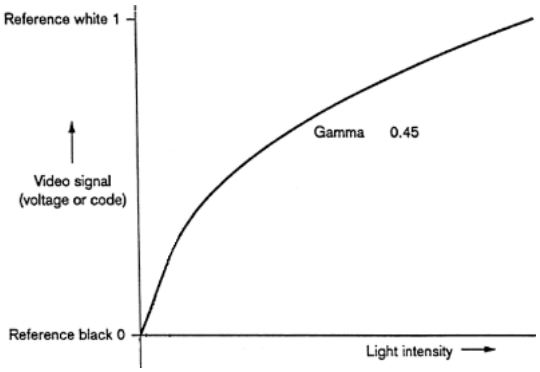

Figure 8 - courbe gamma. source pics.idemmdito.org

Cela donne :

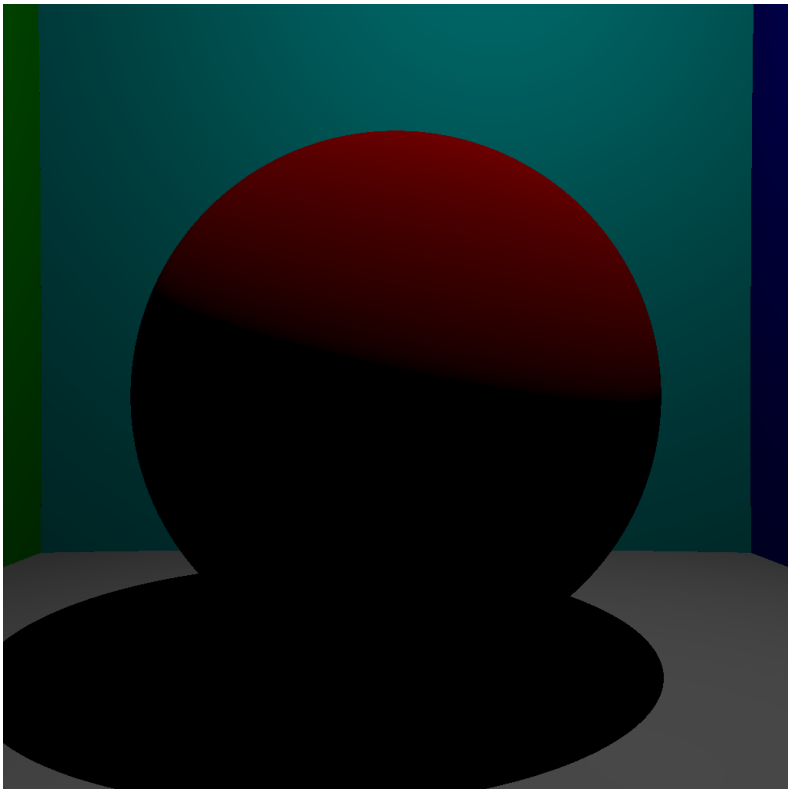

Figure 9 - correction gamma corrigée

Il n'est pas simple de remarquer un quelconque changement, si ce n'est que l'intensité nécessaire est dorénavant plus élevée.

e. Gestion des surfaces spéculaires

Nous allons adapter maintenant le code pour gérer le surfaces miroirs, donc totalement réfléchissantes. Le reste des surfaces est toujours considéré diffus.

Pour ces surface, le path-tracing est adapté pour gérer les lois de la réflexion. On a, pour le rayon réfléchi au point P, sur une surface de normale N :

$$R = -2 * < I, N > N$$

Avec :

$$I : rayon\ incident$$
$$R : rayon\ réfléchi$$
$$N : normale\ \text{à}\ la\ surface$$

Il s'agit ici d'écrire une boucle de récursion, qui envoie un rayon réfléchi dès lors que le rayon incident (donc le rayon réfléchi de la boucle précédente) tombe sur une sphère miroir. On considèrera un nombre maximum de rebond pour écarter le risque de récursion infinie (par exemple, si deux miroirs sont en face).

On obtient, avec cette nouvelle partie du code :

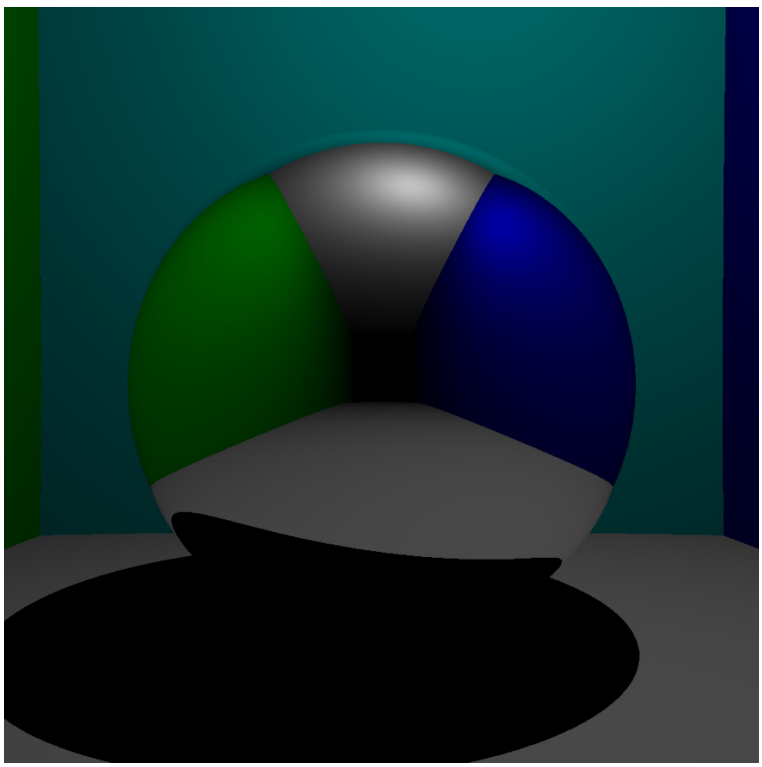

Figure 10 - ajout de la surface miroir

Il est intéressant de noter la présence du plafond blanc sur le haut de la sphère, qui n'est pas visible sans surface miroir. On peut également attester du bon contionnement de la boucle de récursion par l'ajout de deux petites sphères l'une en face de l'autre :

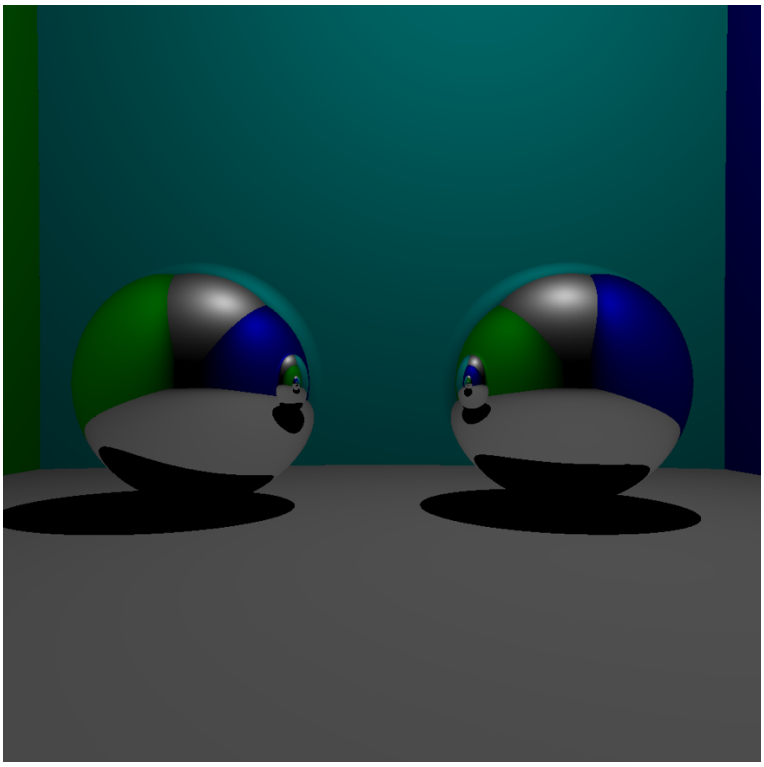

Figure 11 - Test de récursion infinie

f. Gestion des surfaces transparentes

Les surfaces transparentes se traitent de la même façon, mais en respectant les lois de la réfraction (Snell-Descartes). Il faut maintenant considérer de nouvelles grandeurs pour notre problème : les indices de réfraction du milieu environnant et des sphères transparentes. Le rayon renvoyé depuis le point P est désormais le rayon réfracté.

Le calcul géométrique donne :

$$T = T_t + T_n$$

$$T_t = \frac{n_1}{n_2} * (I - < I, N > * N)$$

$$T_t = -N * \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 * (1 - < I, N >^2)}$$

Avec :

$$T : rayon\ transmis\ (réfracté)$$
$$n_1, n_2 : indices\ de\ réfraction$$
$$T_t, T_n : composante\ tangentielle\ et\ normale$$

Notons que le comportement dans le code doit prendre en compte l'origine du rayon incident, car le comportement n_1 vers n_2 n'est pas le même que de n_2 vers n_1. Avec les sphères précédentes, une intensité doublée, et des milieux (air, n=1) et (eau, n=1,3), on obtient :

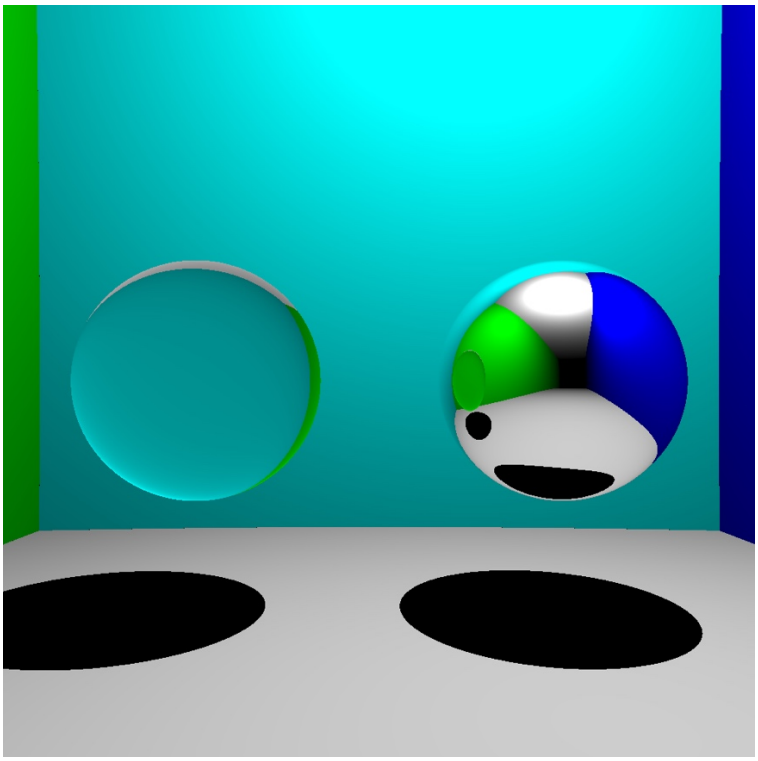

Figure 12 - ajout d'une surface transparente

IV. Seconds développements

a. Équation du rendu

Jusqu'à maintenant, l'unique éclairage provient de la source de lumière ponctuelle. Or, dans la réalité, toute surface diffuse agit comme une source de lumière secondaire, c'est pourquoi un objet non éclairé directement est tout de même visible.

Pour prendre en compte ce phénomène, nous utilisons l'équation du rendu :

$$L_o(x, w_o, \lambda) = L_e(x, w_o, \lambda) + \int_{S^2} (L_i(x, w_i, \lambda) * f(x, w_i, w_o, \lambda) * < N, w_i >) * dw_i$$

Avec :

L_o : $lumière\ réfléchie\ au\ point\ x, dans\ la\ direction\ w_o, à\ la\ longueur\ d'onde\ \lambda$
L_e : $lumière\ émise\ au\ point\ x, dans\ la\ direction\ w_o, à\ la\ longueur\ d'onde\ \lambda$
L_i : $lumière\ incidente\ au\ point\ x, dans\ la\ direction\ w_i, à\ la\ longueur\ d'onde\ \lambda$
f : $fonction\ de\ réflectance\ (BRDF)$
$< N, w_i >$: $terme\ attestant\ de\ l'incidence\ plus\ ou\ moins\ rasante\ du\ rayon$
S : $surface\ au\ point\ x$

Cette équation régit l'éclairage au point x en fonction des éclairages incidents. C'est une équation intégrale récursive, car pour tout point x, on a :

$$L_i(x) = L_o(x')$$

La BRDF (bidirectional reflection distribution function) atteste de la probabilité de réflection d'un photon. En d'autres termes, elle indique comment la lumière est réfléchie sur les objets. Elle peut être mesurée par un appareil physique. Il en existe de nombreuses, selon le matériau. Nous avons par exemple utilisé précédemment, pour les surfaces spéculaires, la BSDF :

$$BSDF = \delta_{w0}$$

Pour résoudre cette équation, une méthode adaptée est la méthode de Monte-Carlo avec échantillonnage par importance. On intègre ici sur les directions w_i au point d'intersection P, lors du lancé du rayon pour path-tracing. Or on ne peut intégrer sur toutes ces directions faute de puissance de calcul, d'autant plus que celle-ci augmente avec le nombre de rebonds (on serait par exemple en 6D au 3ème rayon). La méthode de Monte-Carlo stipule donc un échantillonnage aléatoire des w_i pour approximer l'intégrale. Cette méthode fonctionne mais converge en $\sqrt{n}$ (assez lente).

On aura :

$$L_o(x, w_o, \lambda) \approx \sum_{j=1}^{n} f(x, w_i^j, w_o, \lambda) * L_i^j(x, w_i^j, \lambda) *< N, w_i > \frac{1}{p(w_i^j)}$$

Avec :

$$p : la\ loi\ de\ probabilité\ des\ échantillons\ w^j$$

Plus la probabilité p est proche de $f(x, w_i^j, w_o, \lambda) *< N, w_i >$, plus l'approximation est précise. Pour la suite, on générera donc des rayons avec une distribution proche de la BRDF. Ici, comme les surfaces sont diffuses, la BRDF est constante (probabilité uniforme sur une hémisphère centrée au point P), donc il suffit d'échantillonner le facteur :

$$< N, w_i > = \cos(\theta)$$

Pour d'autres types de matériau, il faut échantillonner aussi la BRDF, ce qui peut se révéler être un challenge. De nombreux échantillonnages sont cependant connus à ce jour.

b. Gestion de l'éclairage secondaire

Pour notre Raytracer, il s'agit de réaliser le path-tracing avec des directions cette fois aléatoires et non plus déterministes comme pour les ombres portées ou les surfaces spéculaires. Il faudrait idéalement tirer un rayon par pixel, puis pour chaque intersection, générer n directions aléatoires. Seulement, ceci demanderait $W * H * n^r$ rayons par pixel, ce qui est trop couteux (avec W * H le nombre de pixels, et r le nombre de rebonds). La stratégie adoptée est de tirer n rayons par pixels, ce qui donne $W * H * n * r$ rayons. L'expérience montre que le résultat est similaire.

Pour rappel, on avait, sans éclairage indirect :

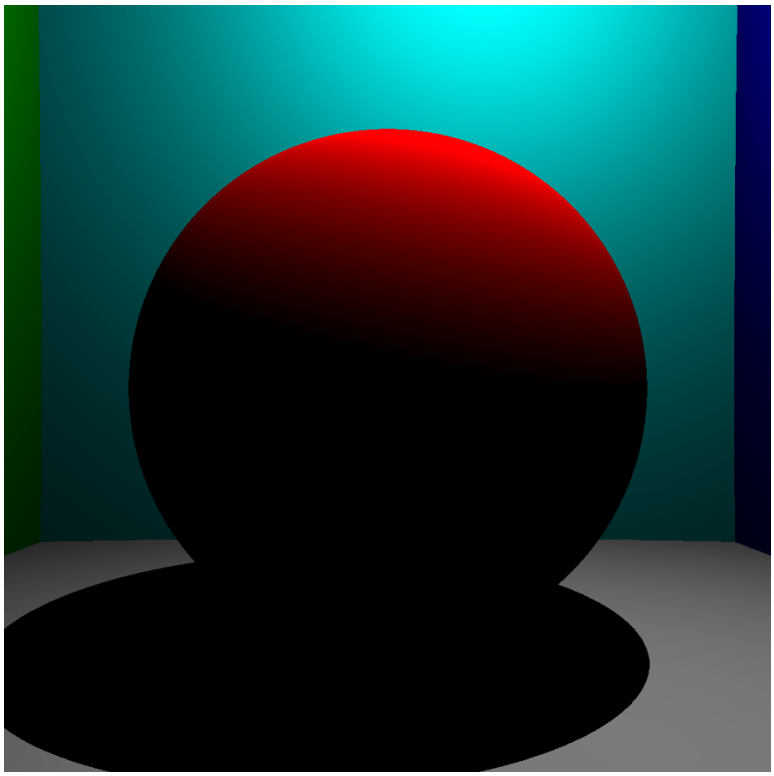

Figure 13 - Rappel de l'éclairage direct

Et on obtient ici, avec 2 rayons par pixels :

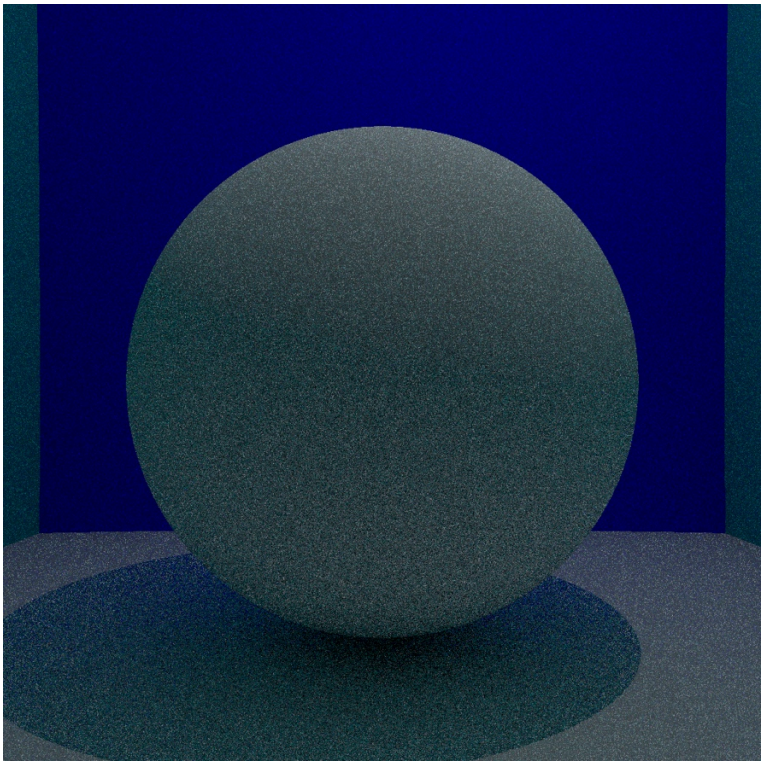

Figure 14 - Ajout de l'éclairage indirect, 2 rayons

On remarque un éclairage bien plus réaliste sur le bas de la sphère. On a cependant besoin de plus de rayons.

Avec 4 rayons par pixels :

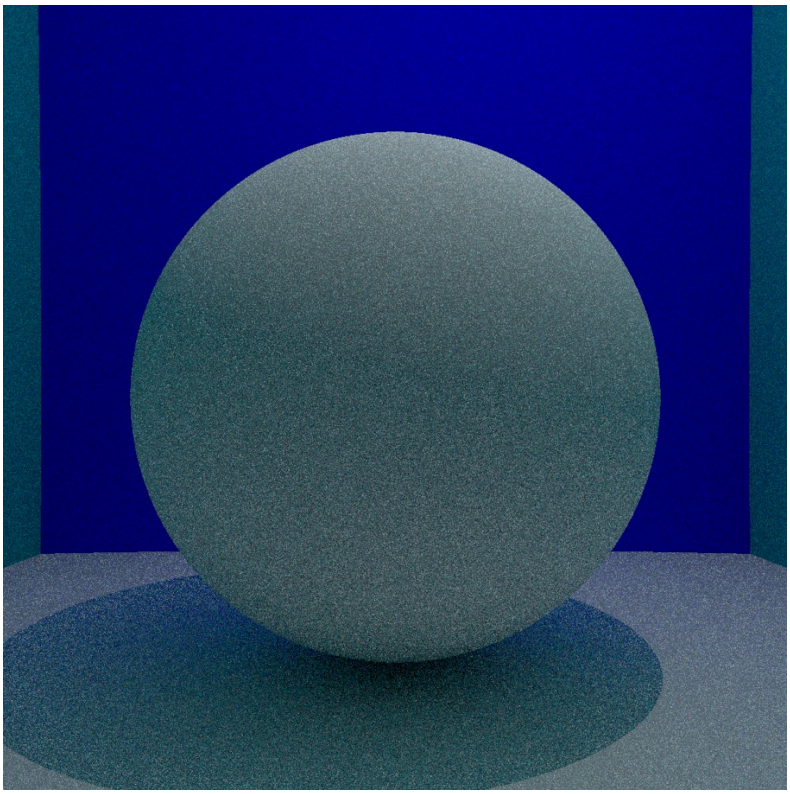

Figure 15 - Ajout de l'éclairage indirect, 4 rayons

Puis, 10 :

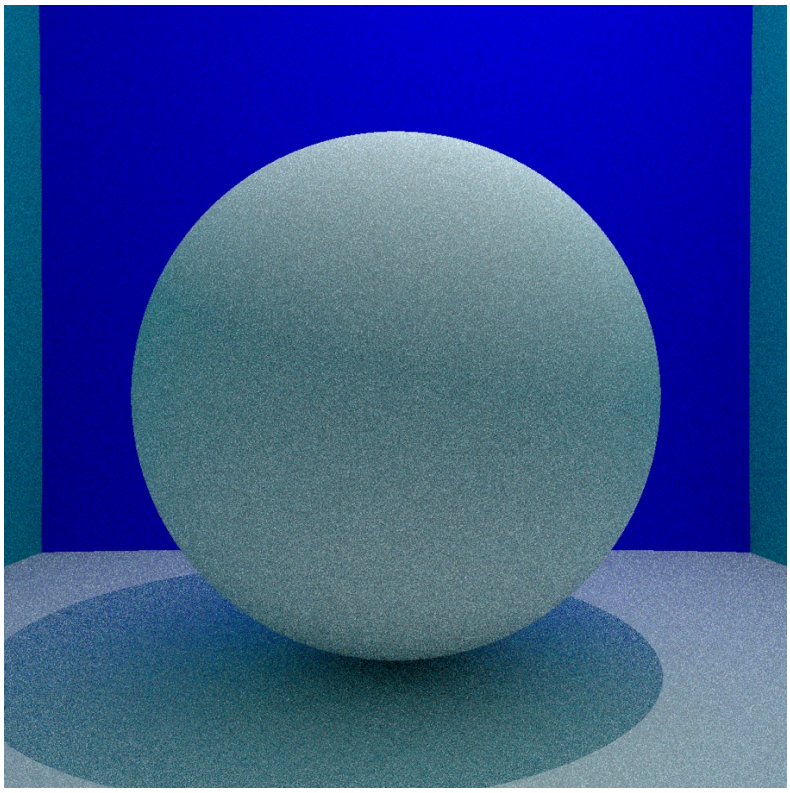

Figure 16 - Ajout de l'éclairage indirect, 10 rayons

Le résultat est bien meilleur ainsi, mais toujours relativement bruité. Il faudrait lancer une simulation avec bien plus de rayons pour obtenir un résultat plus précis. On s'approche d'un résultat correct, mais les ombres sont toujours trop nettes.

c. Correction de l'aliasing

En zoomant sur l'image précédente, on remarque un artefact que l'on va tenter de corriger :

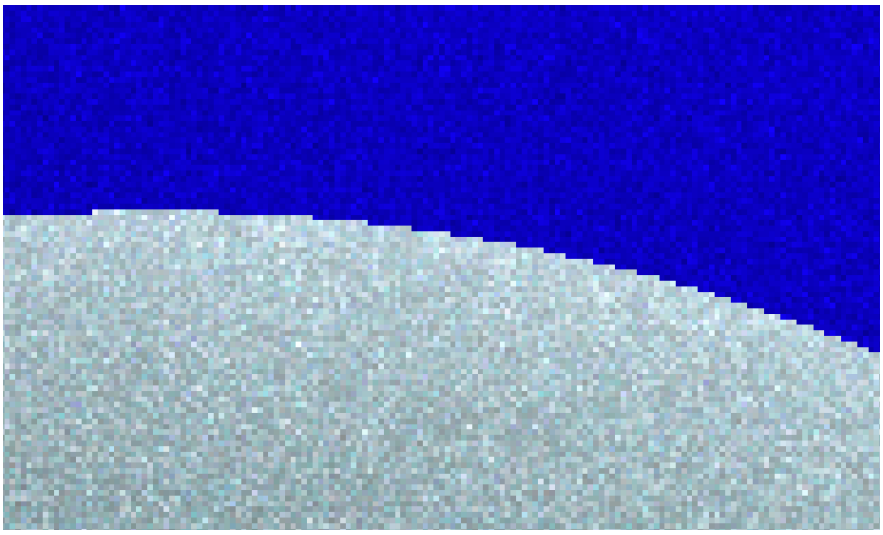

Figure 17 - aliasing, ou crénelage

Cet effet de créneau est dû au fait que tous les rayons sont tirés au centre du pixel, ce qui confère un état binaire aux pixels. Pour contrer celà, nous ajoutons une méthode d'anti-aliasing qui consiste à envoyer des rayons aléatoirement dans chaque pixel, selon une gaussienne centrée au milieu du pixel. La méthode utilisée pour générer les échantillons gaussien est la méthode Box-Muller.

Grâce à cette méthode, on obtient un résultat plus doux :

Figure 18 - avec anti-aliasing

d. Gestion de l'éclairage étendue et des ombres douces

Considérons maintenant des lumières étendues plutôt que des lumières ponctuelles. Ces éclairages correspondent mieux à la réalité, et produiront des ombres plus douces, donc plus réalistes.

On remplace donc les lumière ponctuelle par des sphères de lumière. On affichera le pixel lorsque le rayon réfléchi intersecte la sphère de lumière. Il s'agit ici de gérer différemment l'éclairage direct de l'éclairage indirect. L'éclairage indirect est géré de la même façon que précédemment, mais on ne considère pas l'intersection avec la sphère de lumière. L'éclairage direct quant à lui est réalisé en échantillonnant les directions avec importance vers la sphère de lumière, pour augmenter les chances d'intersection. Plus la sphère de lumière est petite, moins la probabilité d'intersection est élevée, d'ou l'échantillonnement par importance.

On obtient, avec une sphère de lumière :

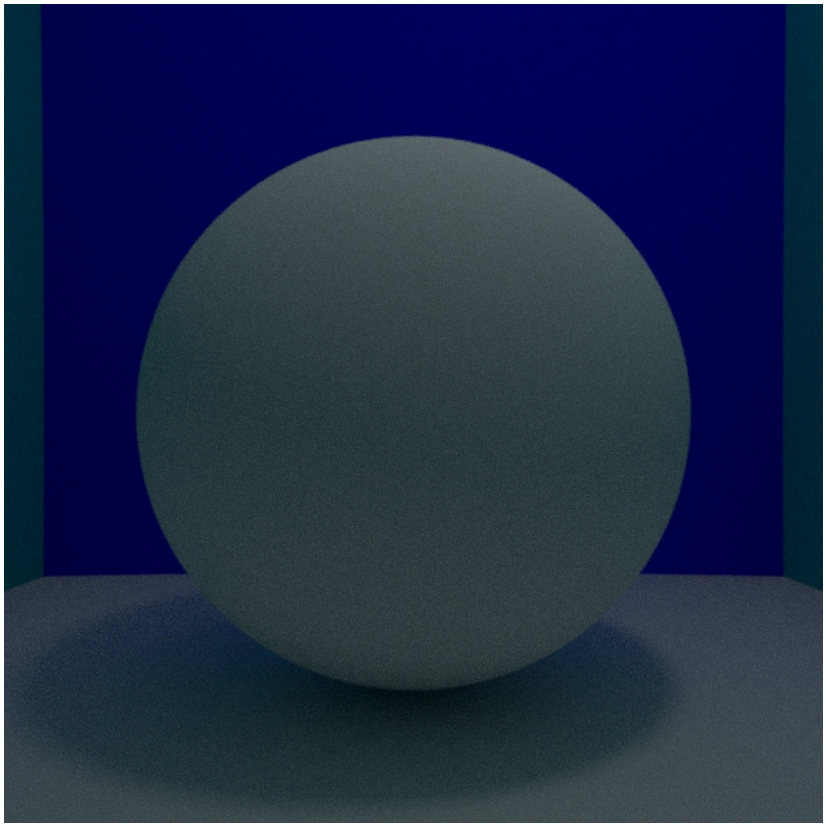

Figure 19 - éclairage étendu

On remarque bien une ombre plus douce.

e. <u>La profondeur de champ</u>

Pour terminer, nous allons utiliser un autre modèle que le modèle sténopé décrit en II. Ce modèle ne permet pas de prendre en compte l'effet de profondeur de champ. Pour palier ce problème, on considère une ouverture carrée, simplifiant la réalité (ouverture circulaire avec diaphragme pour l'oeil ou les caméras), à travers laquelle on tirera les rayons. Ainsi, ces différents rayons auront une origine différente à l'intérieur du carré, et une destination (direction) se positionnant au niveau de la distance focale.

De cette façon, tout objet sur le plan focal sera net, et les objets se situant en amont et en aval de ce plan, flou.

En implémentant cette solution pour 3 sphères décalées selon l'axe de profondeur, on obtient :

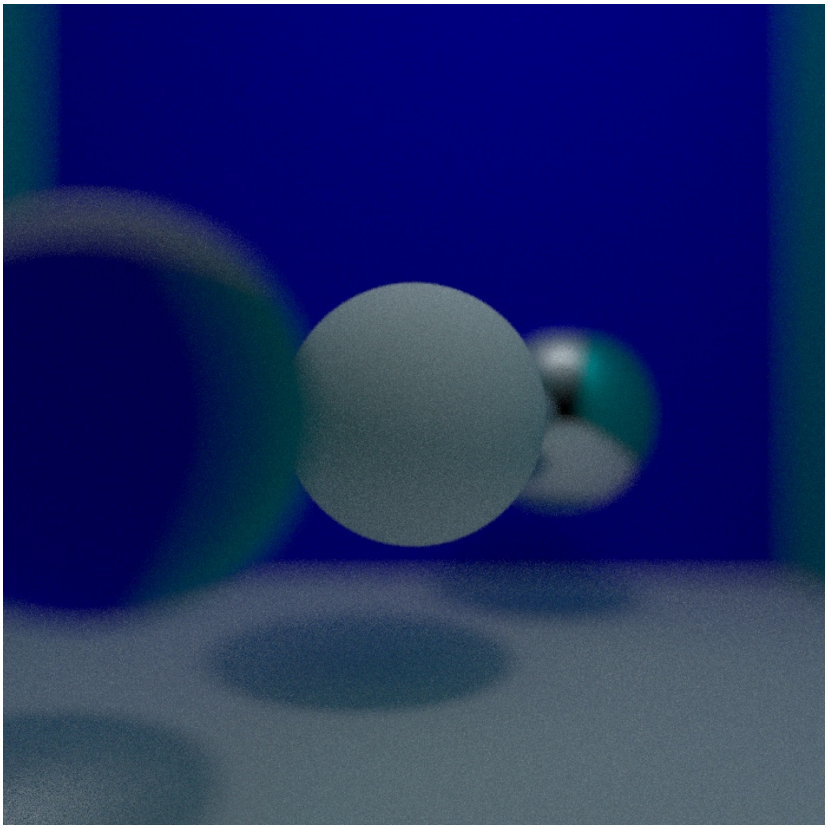

Figure 20 - Application de la profondeur de champ

V. Pour aller plus loin

Les développements possibles qui n'ont pas été abordés dans ce rapport sont :

- Les maillages triangulaires, pour diversifier la scène à d'autres objets
- Le déplacement de la caméra, pour des scènes dynamiques
- Les milieux participatifs
- Utilisation d'autres gammes de BRDF
- Utilisation de textures