ÉCOLE CENTRALE DE LYON

MOS 2.2 - INFORMATIQUE GRAPHIQUE
RAPPORT DE RAVAUX PRATIQUES

Programmation d'un Ray Tracer en C^{++}

Étudiant
Mehdi Elion

Enseignant
Nicolas Bonneel

15 Mars 2017

Table des matières

Introduction

Le présent rapport a pour objectif de présenter les résultats obtenus à l'issue du cours d'informatique graphique sur le « Ray Tracing ». Les algorithmes, formules et codes utilisés ne seront pas détaillés et on mettra plutôt l'accent sur les différentes modélisations expliquées en cours et implémentées dans le code en les illustrant par des images. Pour certaines de ces images, on donnera également le temps d'exécution afin d'estimer l'efficacité du code, lequel sera joint à ce rapport.

Avant d'illustrer les différentes modélisations, rappelons dans les grandes lignes le principe général de notre « ray tracer ». L'idée est de concevoir « virtuellement » une scène composée par différents objets, chacun ayant ses propres surfaces et propriétés optiques (diffusion, réflexion, transparence, émissivité…). On positionne dans cette scène une « caméra » dont la taille est déterminée par la taille de l'image (i.e. le nombre de pixels) de l'image que l'on souhaite générer. Le but est donc de déterminer la couleur de chacun de ces pixels.

Pour cela, on utilise le principe de réciprocité d'Helmholtz qui permet de suivre les rayons de lumière en sens inverse. Pour chaque pixel, on imagine un rayon (i.e. une demi-droite) qui « part » de l'origine de la caméra vers centre de ce pixel (alors qu'il s'agit dans la réalité d'un rayon qui arrive sur la caméra), puis on calcule la couleur du pixel en fonction de l'intersection du rayon avec la scène et, si intersection il y a, en fonction des propriétés et de la position de l'objet intersecté par rapport à la source lumineuse et aux autres objets de la scène.

1 Eclairage Direct

Notons tout d'abord que pour toutes les modélisations mentionnées dans ce rapport, on utilisera le principe de réciprocité d'Helmholtz qui permet de suivre les rayons de lumière en sens inverse.

La première étape du ray tracing (ou plutôt du path tracing) consiste à traiter l'éclairage direct et les surfaces diffuses.

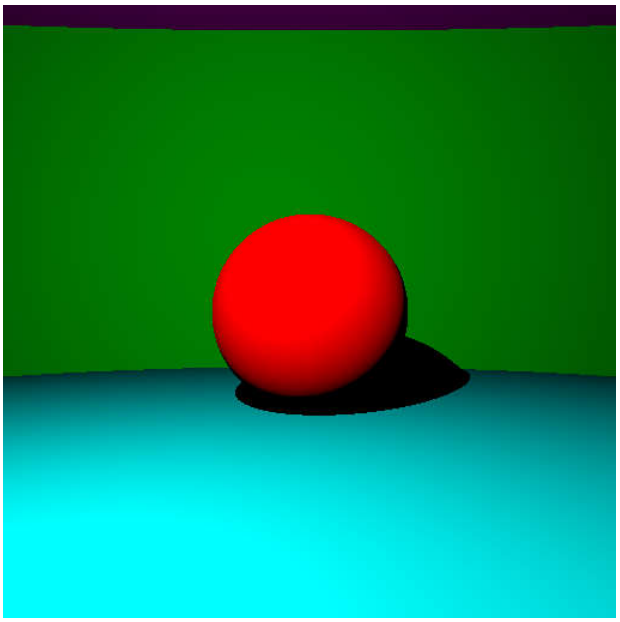
Une première modélisation consiste donc à que chaque objet n'est éclairé que par la source lumineuse (considérée ponctuelle ici), que l'on appelle composante directe, et non pas par la lumière réémise par les autres objets de la scène, que l'on appelle composante indirecte et que l'on traitera plus tard.

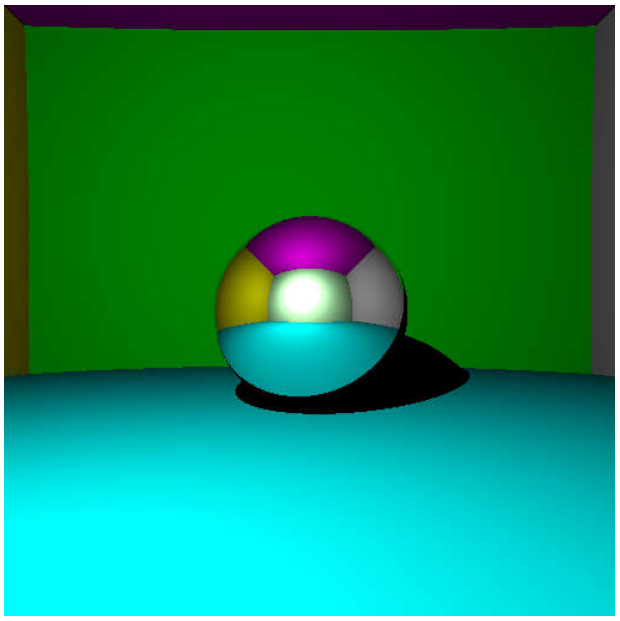
La procédure est la suivante :
- on tire un rayon depuis la caméra
- s'il n'intersecte pas la scène, on colorie le pixel en noir
- s'il intersecte la scène, on retient l'intersection la plus proche de la caméra, puis on calcule le point d'intersection, la normale à la surface en ce point. On tire ensuite un rayon vers la source lumineuse.
- si ce rayon est intersecte une autre surface avant d'atteindre la source lumineuse, on colorie le pixel en noir.
- si ce rayon atteint la source lumineuse, on calcule la couleur du pixel en fonction de la distance du point d'intersection à la source lumineuse, de l'intensité de la source et de l'angle entre la normale et le vecteur directeur du rayon lumineux.

On prendra également soin, lorsque l'on tire un nouveau rayon à partir de ce point d'intersection, de décaler légèrement son origine sans quoi on obtiendra une image bruitée à cause des imprécisions numériques.

Notons qu'ici, toutes les sphères sont diffuses. On traitera également les surfaces spéculaires et transparentes en tirant, après intersection par le rayon incident, un rayon réfléchi ou réfracté dont la direction est donnée par les lois de l'optique.

Voici un premier rendu obtenu avec cette procédure. La scène de gauche est composée de 6 sphères géantes qui font office de murs, plafond et sol, d'une sphère rouge placée devant la caméra et d'une source lumineuse ponctuelle. La scène de droite est semblable mais la sphère centrale possède une surface spéculaire.





On obtient un rendu sur lequel on peut d'ores et déjà émettre quelques remarques.

Premièrement, on observe que les bords des objets sont crénelés. Cela est dû au fait que l'image est composée de rangées de pixels qui ne peuvent rendre compte des formes arrondies des sphères. Chaque rayon initial est tiré depuis le centre du pixel. En passant d'un pixel à l'autre, on peut tirer un rayon qui intersecte un objet différent Ainsi, un pixel permet de visualiser un seul objet alors qu'en réalité, plusieurs objets peuvent être à cheval sur un même pixel. On proposera par la suite une méthode d'anti-aliasing permettant de réduire l'effet de crénelage.

Ensuite, on remarque que l'ombre de la sphère rouge est totalement noire, ce qui n'est pas très réaliste. Cela est dû au fait que cette implémentation ne prend pas en compte la composante indirecte de l'éclairage. En effet, la partie du sol qui est à l'ombre de la sphère rouge devrait aussi recevoir de la lumière réémise par les murs par exemple.
En outre, on remarque que l'ombre n'a pas cet aspect estompé ou diffus que l'on obtient en réalité. Cela est dû au fait que la source lumineuse est ponctuelle, et non étendue, auquel cas on obtient des ombres plus réalistes. On proposera dans la section suivante une méthode permettant de palier à ces problèmes.

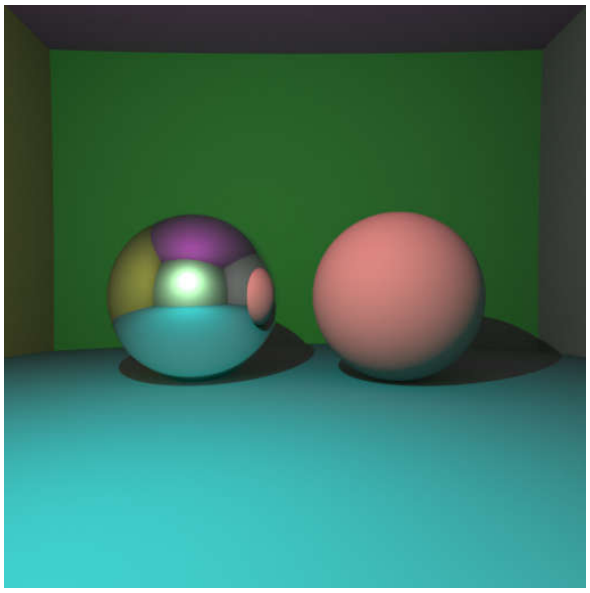
4 Intégration de Monte-Carlo et Eclairage Indirect

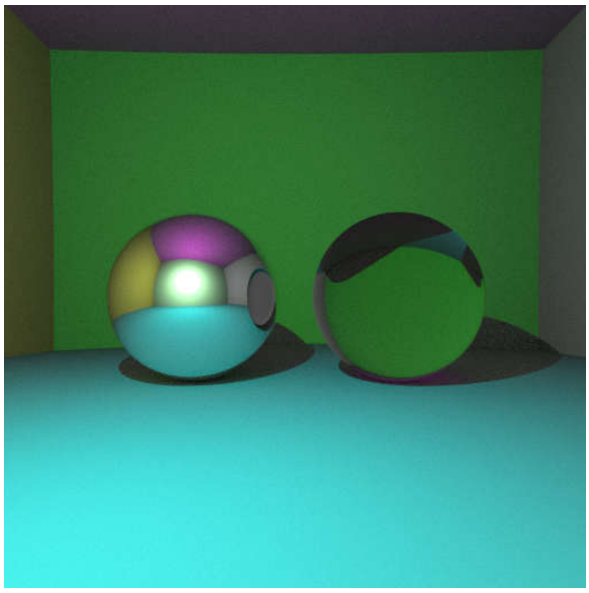
Dans un premier temps, nous implémentons la composante indirecte de l'éclairage afin d'obtenir des ombres plus douces.

À chaque intersection d'un rayon avec une sphère diffuse, on calcule une couleur en sommant la contribution directe et la contribution indirecte. La contribution directe est obtenue comme précédemment, mais la contribution indirecte est obtenue en tirant un rayon dans une direction aléatoire autour de la normale au point d'intersection, puis en récupérant la couleur obtenue à l'intersection (s'il y en a une) de ce rayon aléatoire. On effectue ensuite récursivement un certain nombre de rebonds jusqu'à atteindre un nombre maximal de rebonds que l'on a fixé au préalable.

Toutefois, l'expression analytique suggère qu'à chaque rebond, la composante indirecte soit obtenue en intégrant sur toutes les directions dont elle peut provenir (hémisphère dont le plan sécant est celui défini par la normale au point d'intersection). Ainsi, pour atteindre une précision suffisante sans induire de croissance exponentielle du nombre de calcul à effectuer, on tire, pour chaque pixel, N rayons qui vont tous aboutir sur un chemin aléatoire tel que défini plus haut. On moyenne ensuite, pour chaque pixel, les résultats des N tirages : c'est un estimateur de Monte-Carlo.

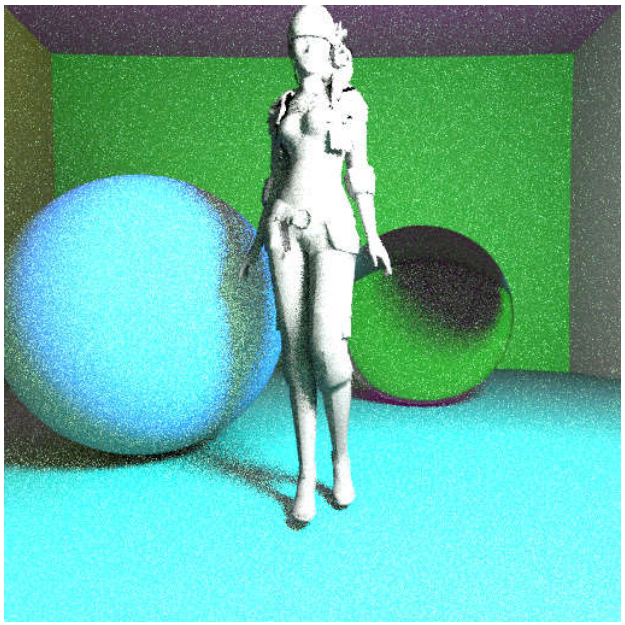
Voici deux premiers rendus avec une sphère diffuse, une sphère miroir et une sphère transparente obtenu avec la méthode décrite ci-dessus.



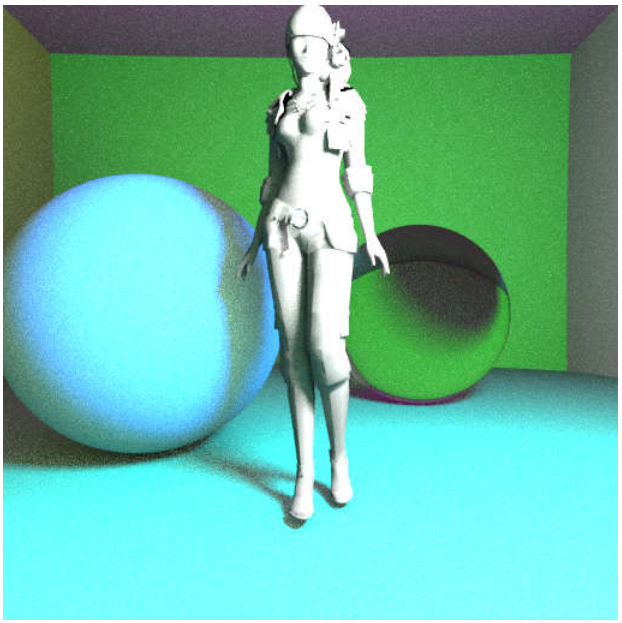


On remarque effectivement que les ombres sont passées de totalement noires à plus claires. En revanche, la forme de l'ombre est toujours aussi marquée plutôt qu'avoir l'aspect estompé que l'on obtient avec des sources étendues. On implémentera les ombres étendues par la suite.

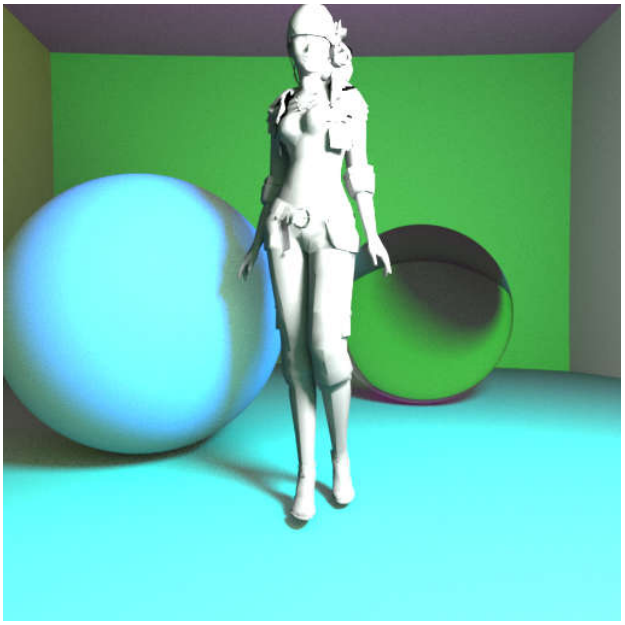
Les images ci-dessous illustrent l'importance du nombre N de chemins aléatoires tirés dans la précision de l'estimateur de Monte-Carlo. Ces images comportent un maillage ainsi qu'une source lumineuse étendue mais nous nous attarderons sur ces deux points plus tard.



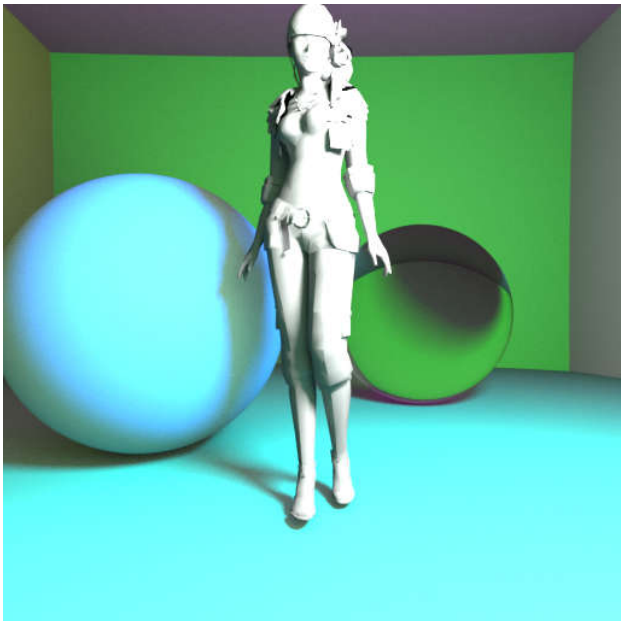
10 tirages (10s)



100 tirages (1min 5s)



1000 tirages (12 min)



2000 tirages (25min)

On remarque qu'avec un faible nombre de tirages, l'image résultante est assez bruitée. En revanche, à mesure que l'on augmente le nombre de tirages, l'image devient de moins et moins bruitée, jusqu'à devenir relativement nette, comme avec l'image à 2000 tirages par exemple. En revanche, le temps de calcul augmente aussi avec N, ce qui peut poser problèmes dans certaines applications qui nécessitent un rendu en temps réel comme les jeux vidéo. Il faudra donc trouver un compromis entre un temps de calcul acceptable, le matériel disponible pour effectuer ces calculs et une qualité d'image acceptable.

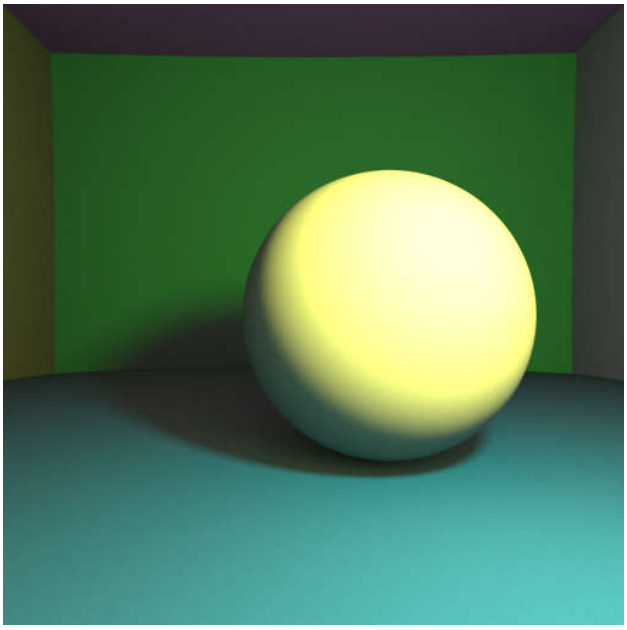
5 Source de lumière étendue

L'implémentation d'une source de lumière étendue permet d'obtenir des ombres plus douces à la forme moins marquée qu'avec une source ponctuelle.

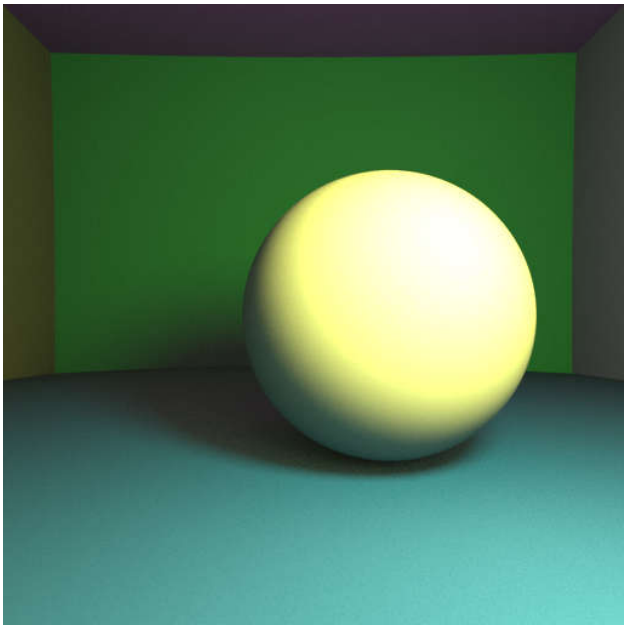
Pour modéliser notre source de lumière étendue, nous allons considérer qu'il s'agit d'une sphère lumineuse. Nous réutiliserons donc la structure de sphère utilisée pour les objets composant la scène. Ainsi, à chaque rebond sur une surface diffuse, on calculera la composante directe par rapport à cette sphère lumineuse, mais la méthode de calcul change.

En théorie, le point d'intersection en question est éclairé par l'hémisphère de la sphère lumineuse qui lui fait face. On devrait donc intégrer la composante directe sur toute la surface de cet hémisphère, ce qui demanderait trop de calculs. Par conséquent, nous allons choisir un point aléatoire sur cet hémisphère et calculer la composante directe an fonction de celui-ci. La méthode de Monte-Carlo permettra ensuite d'obtenir un rendu convenable en moyennant les différents chemins tirés.

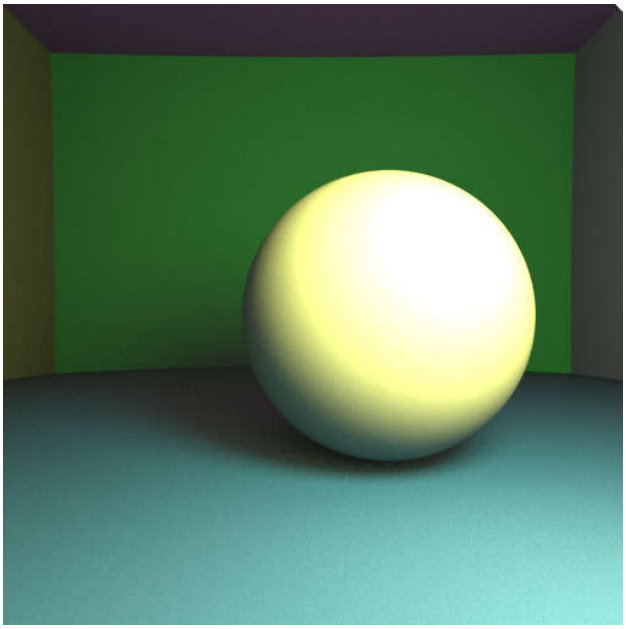
Les images ci-dessous ont été réalisées avec une source de lumière étendue (sphérique en l'occurrence) dont on augmente progressivement le rayon afin d'observer son effet sur les ombres. L'échantillonnage de Monte-Carlo est fait avec 1000 rayons par pixel, et les temps de calcul sont indiqués.



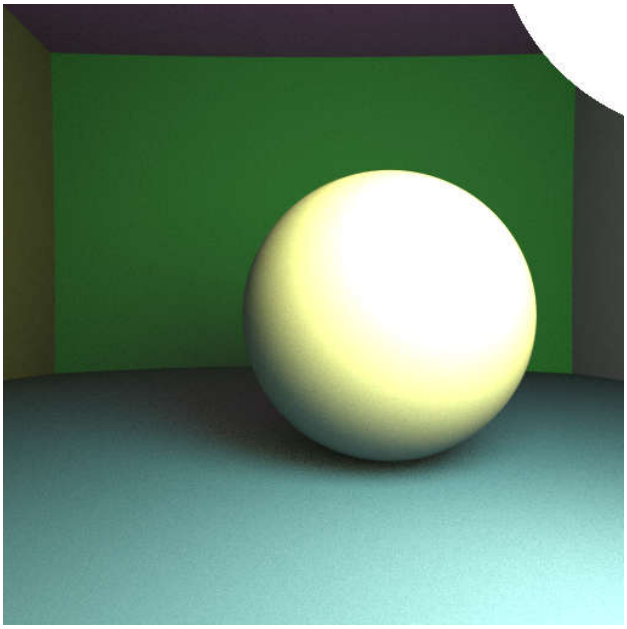
Rayon = 2 (4min1s)



Rayon = 4 (3min57s)



Rayon = 6 (3min58s)



Rayon = 8 (3min48s)

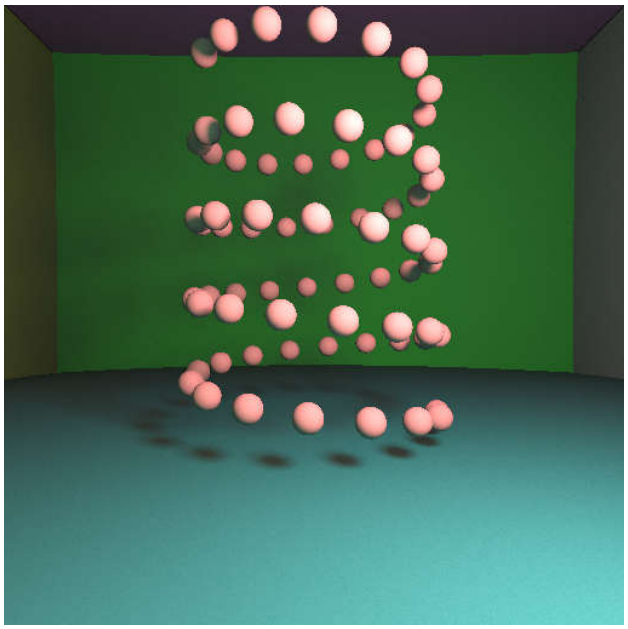
On constate que l'ombre de la sphère jaune à l'avant de la scène devient de plus en plus douce à mesure que le rayon de la sphère lumineuse augmente. Inversement, lorsque le rayon de la sphère lumineuse est faible, plus l'ombre est marquée et plus ses bords sont sombres.

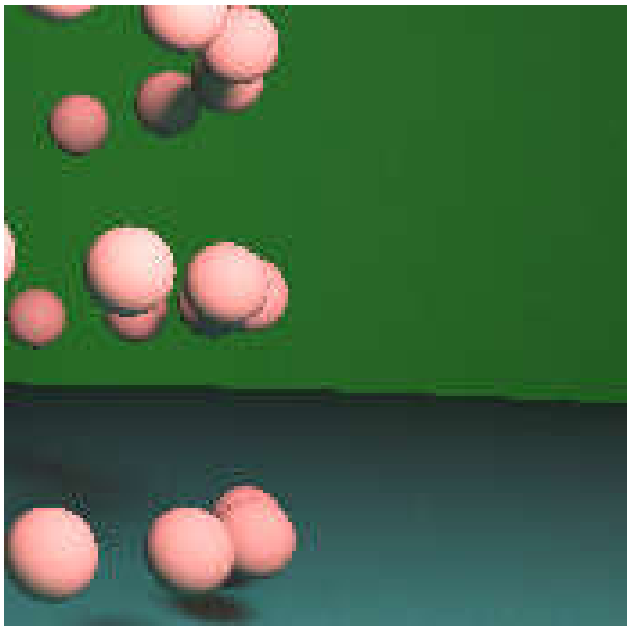
6 Anti-aliasing

Il existe plusieurs moyens de réaliser un anti-alisasing (ou anti-crénelage). Nous optons ici pour une méthode qui consiste à tirer chacun des N rayons aléatoirement à l'intérieur du pixel, plutôt que de tous les tirer depuis le centre du pixel. Pour cela, perturbe aléatoirement la direction de chaque rayon afin qu'ils balaient la superficie du pixel.

On souhaite également accorder plus de poids aux rayons qui sont proches du centre du pixel. Pour faire cela nous tirons, dans le plan de l'écran, les coordonnées polaires (par rapport au centre du pixel) de la destination du rayon avec une loi uniforme, puis nous les convertissons en coordonnées cartésiennes. On obtient alors des cordonnées cartésiennes dont la distribution est centrée autour du centre du pixel.

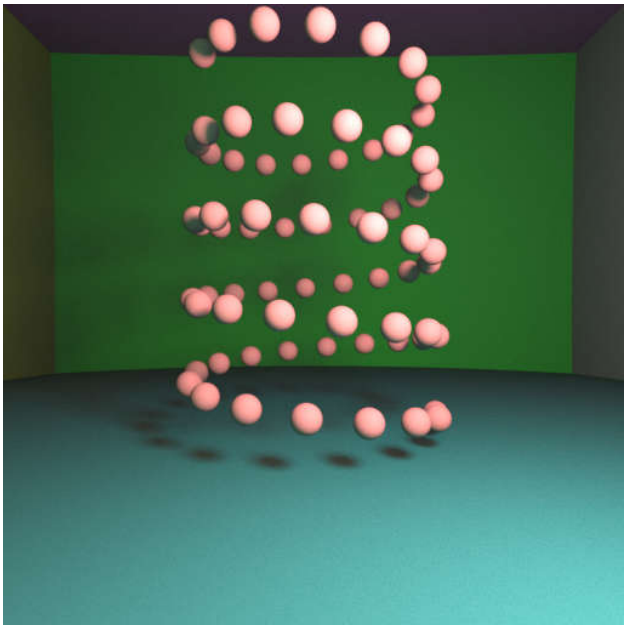
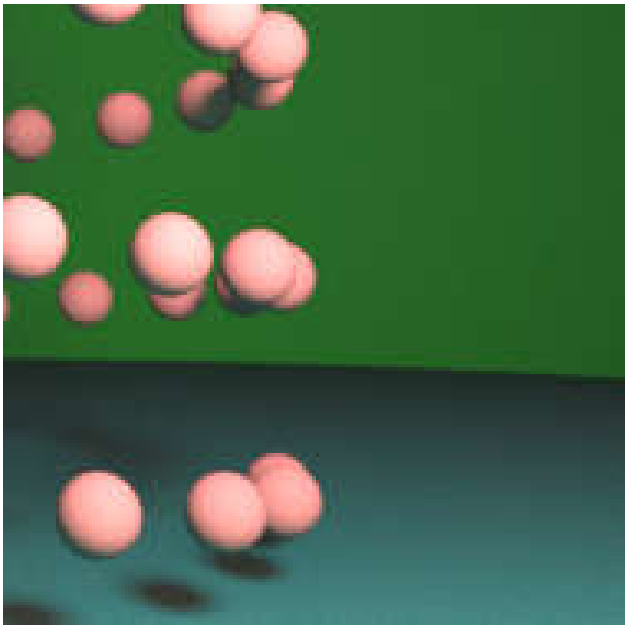
Afin de mieux apprécier l'effet de cette méthode, voici quelques images réalisées avec et sans anti-aliasing.





1000 tirages (6min 44s) - Sans Anti-Aliasing

Sur ces images sans anti-aliasing, on constate clairement l'effet de crénelage, que ce soit les sur les sphères rouges ou bien sur les sphères géantes qui constituent les murs, le sol et la plafond.

1000 tirages (7min 7s) - Avec Anti-Aliasing

En revanche, sur ces images réalisées avec l'anti-aliasing décrit plus haut, on constate que l'effet de crénelage s'est bien estompé et que les formes paraissent plus lisses, même après avoir zoomé sur l'image.

7 Maillages et Textures

Dans notre programme, nous utilisons des maillages téléchargeables sur internet au format obj. Ces fichiers contiennent des informations sur :

- les coordonnées des sommets
- les triplets de sommets définissant chacune des faces
- les coordonnées UV des sommets ainsi que les fichiers permettant de gérer les textures
- les normales aux sommets (fournies pour certains maillages seulement)

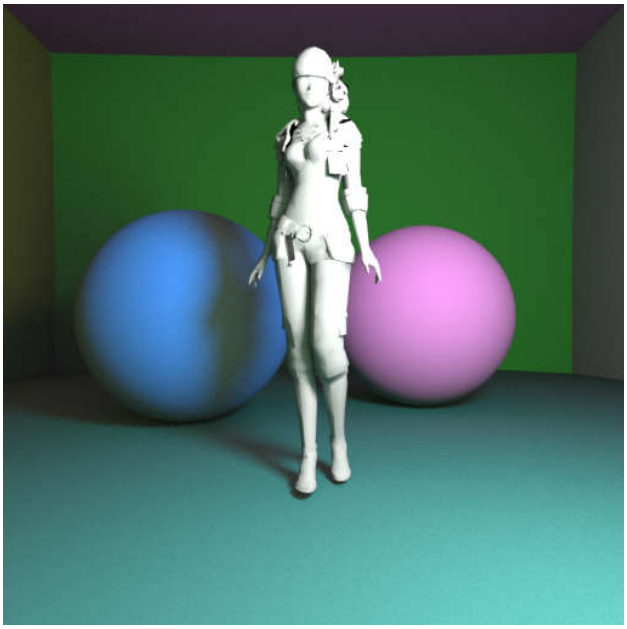
Gestion des maillages

Un maillage est un ensemble de sommets qui délimitent des faces triangulaires. La gestion des maillages consiste donc dans un premier temps à implémenter une routine d'intersection entre un rayon et une face triangulaire, ce que nous faisons par une méthode utilisant les coordonnées barycentriques. Nous utilisons également ces coordonnées

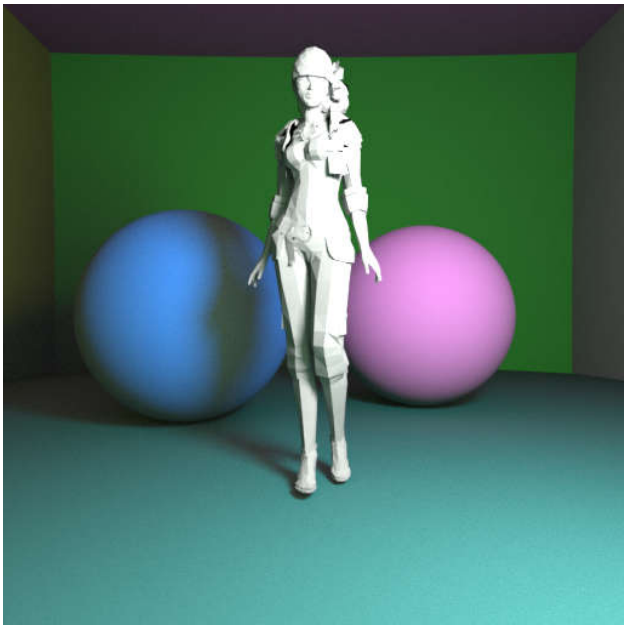
barycentriques pour calculer la normale au point d'intersection et effectuer un lissage de Phong à lorsque les normales aux sommets nous sont fournies.

Nous implémentons également une structure permettant d'accélérer les calculs. Dans un premier temps, nous implémentons une boîte englobante dont les plans sont parallèles aux axes du repère de la scène. Afin d'éviter les appels inutiles à la routine d'intersection avec le maillage (qui elle-même vérifie l'intersection avec tous les triangles du maillage), on ne lance ces derniers uniquement lorsque le rayon intersecte la boîte englobante. Dans un deuxième temps, nous effectuons une scission récursive de cette boîte en deux sous-boîtes en coupant en deux la boîte mère selon la dimension la plus allongée. On obtient alors une structure d'arbre binaire appelée BVH (Bounding Volume Hierarchy) dont les nœuds sont des boîtes englobantes et dont les feuilles contiennent les faces triangulaires. Une fois que l'on dispose de cette structure, on ne parcourt que les sous-boîtes qui sont intersectées par le rayon, ce qui permet de lancer moins de routines d'intersection sur les faces.

Voici deux images permettant de visualiser les résultats de la gestion de maillage et du lissage de Phong. La BVH a été implémentée, les temps d'exécution qui sont précisés en tiennent donc compte.



1000 tirages (33min 11s) - Avec lissage de Phong



1000 tirages (31min 51s) - Sans lissage de Phong

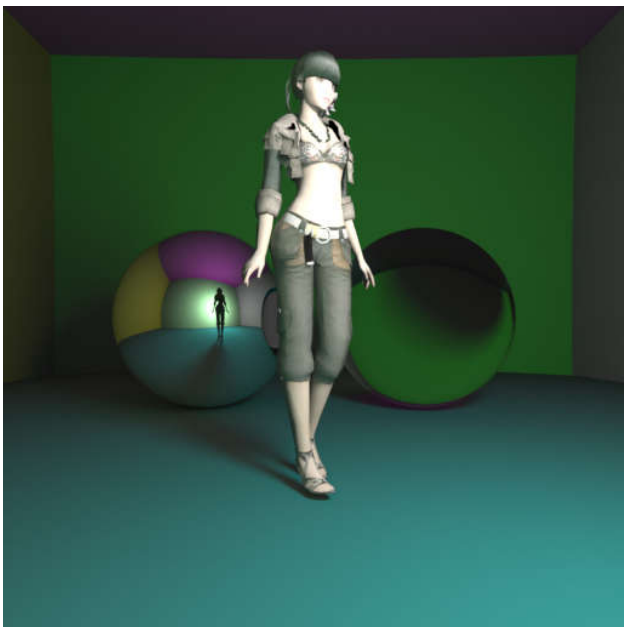
On constate, sur l'image sans lissage de Phong, un aspect laissant apparaître des « facettes » qui ne paraissent pas naturelles. On remarque également que le lissage de Phong permet d'annuler cet effet et d'obtenir un rendu plus réaliste.

Gestion des textures

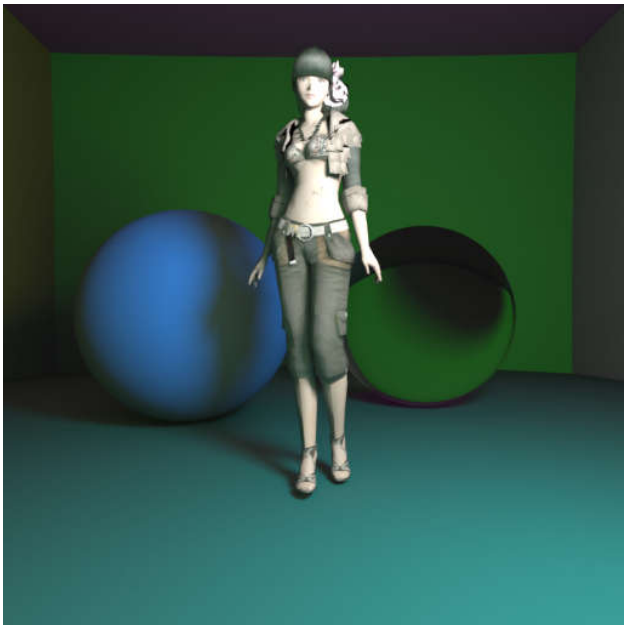
La gestion des textures consiste à récupérer, pour chaque point d'intersection avec une face, les coordonnées UV donnant la couleur à assigner à partir du fichier texture correspondant. Pour obtenir les coordonnées UV d'un point d'intersection sur une face, on l'interpole à l'aide des coordonnées UV des sommets et des coordonnées barycentriques calculées par la routine d'intersection.

Voici quelques images réalisées avec le même maillage que précédemment. Notez que l'on peut également pivoter le maillage en appliquant une matrice de rotation aux coordonnées de ses sommets. On peut également faire pivoter la caméra en appliquant une matrice de rotation aux coordonnées des vecteurs directeurs des rayons initiaux.

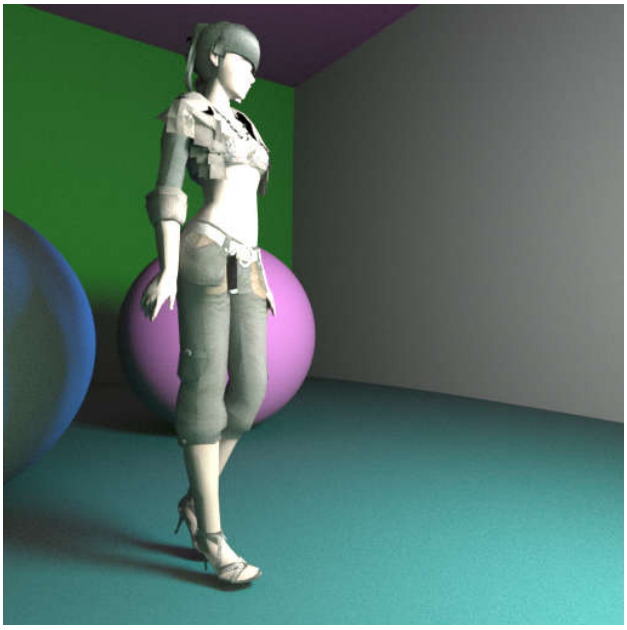
2000 tirages (58min 12s)



2000 tirages (54min 50s)

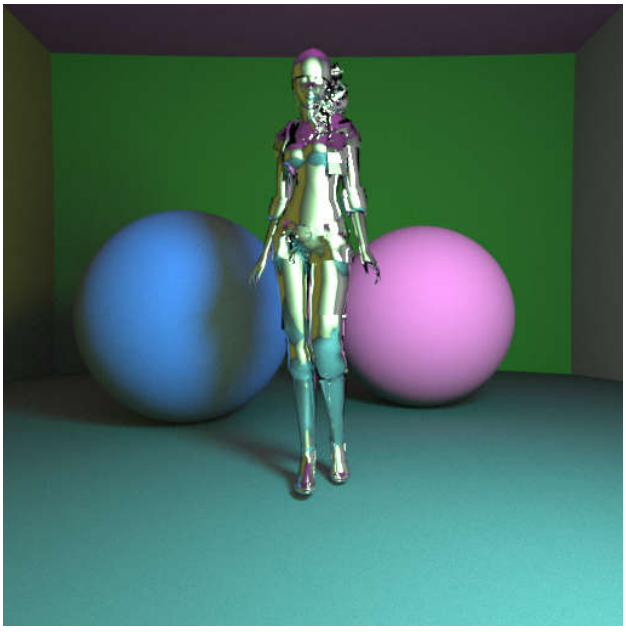


2000 tirages (1h 6min 55s)



500 tirages (16min 41s)

On peut également modifier les propriétés de surface des maillages pour en faire des surfaces spéculaires par exemple.



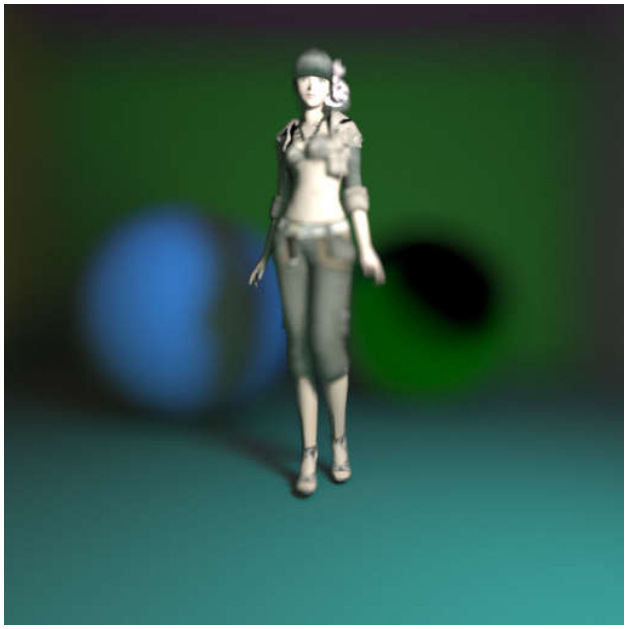
2000 tirages (21min 15s)

8 Profondeur de Champ

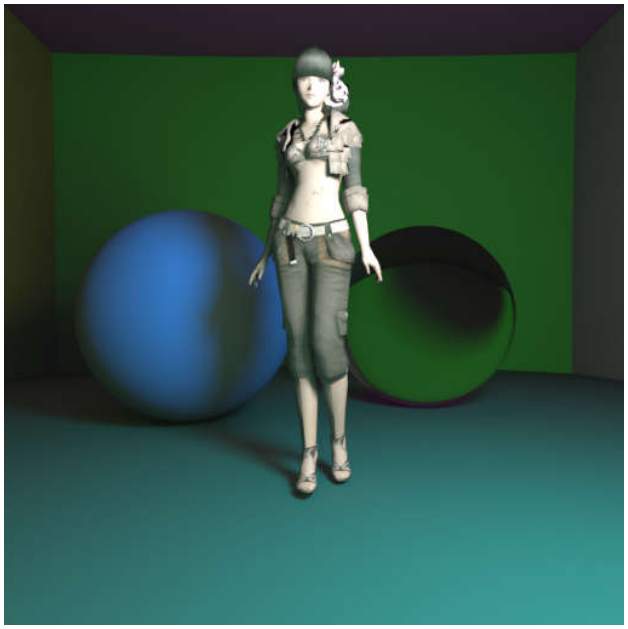
Pour introduire des effets de flou dans nos images, nous allons modéliser une caméra de type « pinhole ». Ces caméras sont composées d'une lentille et d'un obturateur. Jusqu'ici, nous avions simplifié l'étape correspondant à la lentille en lançant directement les rayons depuis un centre fixe de la caméra. Il ne nous reste donc qu'à modéliser l'obturateur, que nous allons considérer comme un carré parallèle au plan de l'image et placé à distance de mise au point de celui-ci. La direction des rayons est toujours décalée aléatoirement par rapport du centre du pixel (anti-aliasing) mais leur origine est décalée aléatoirement de sorte à couvrir le carré de l'obturateur.

Ainsi, les objets placés avant ou après distance mise au point seront flous, tandis que ceux qui sont situés à distance de mise au point seront nets.

Voici quelques images obtenues avec la méthode évoquées ci-dessus.

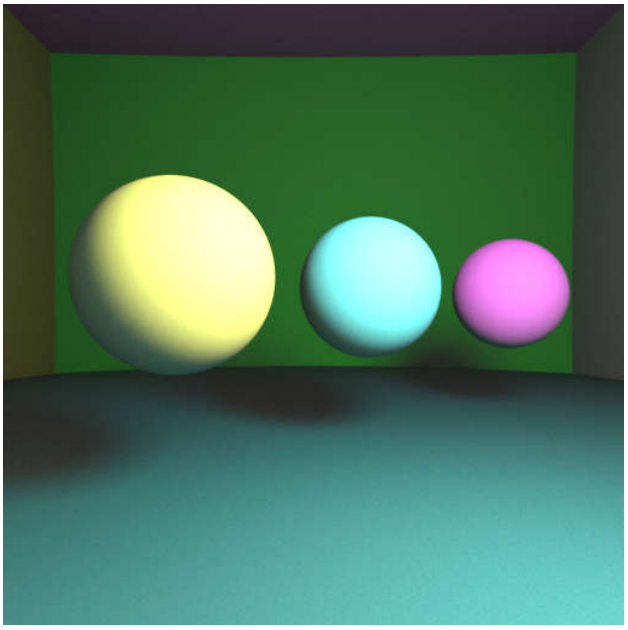


1000 tirages (28min 30s) - Avec profondeur de champ

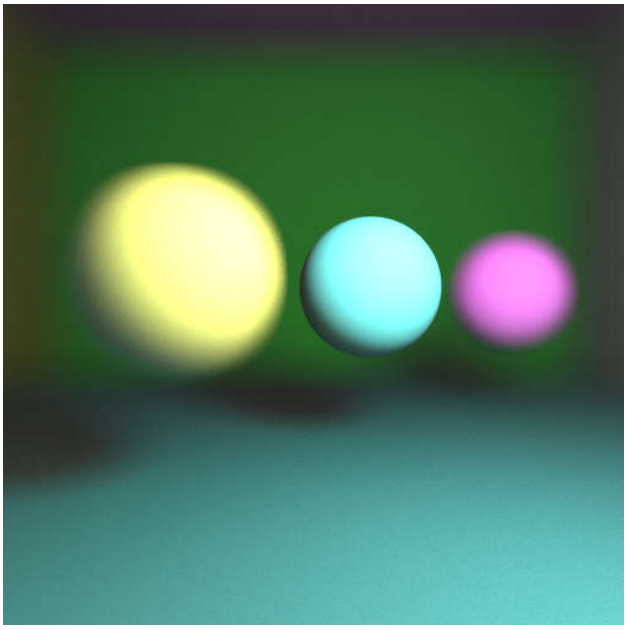


2000 tirages (1h 6min 55s) - Sans profondeur de champ

Sur les deux images ci-dessus qui ont été réalisé avec (à gauche) et sans (à droite) anti-aliasing, on a volontairement placé la fille à distance de mise au point, et les deux sphères au-delà de cette distance. On constate effectivement que la fille est beaucoup plus nette que les deux sphères.



1000 tirages (3min 9s) - Sans profondeur de champ

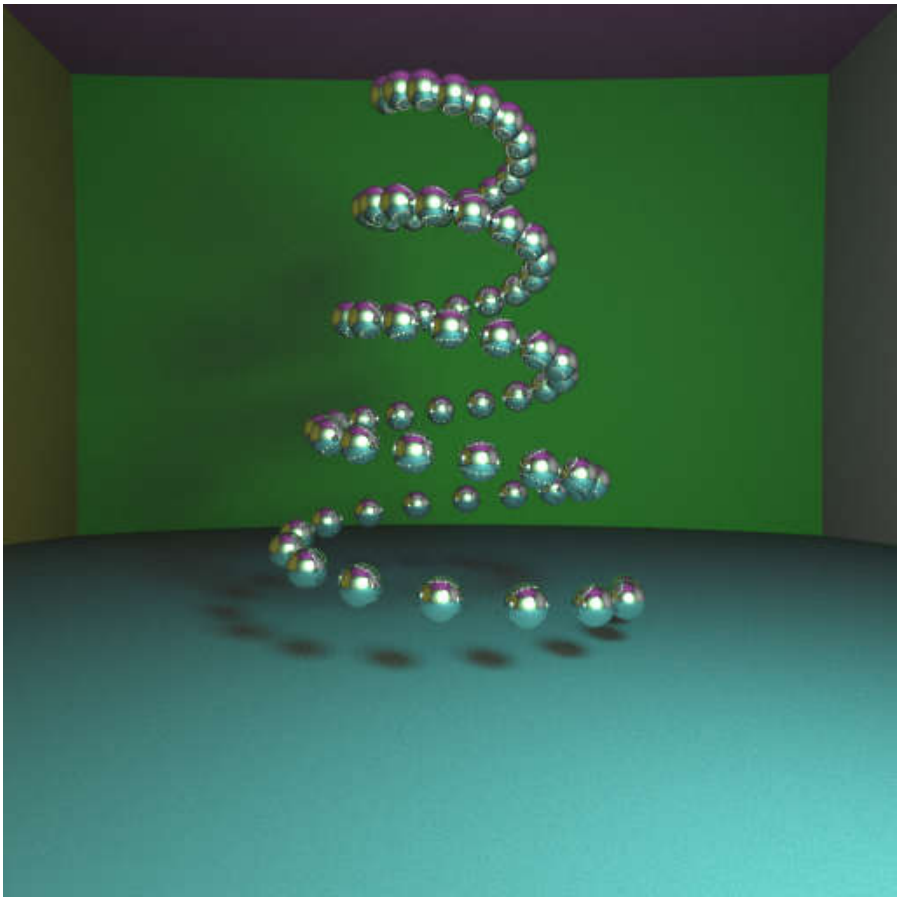


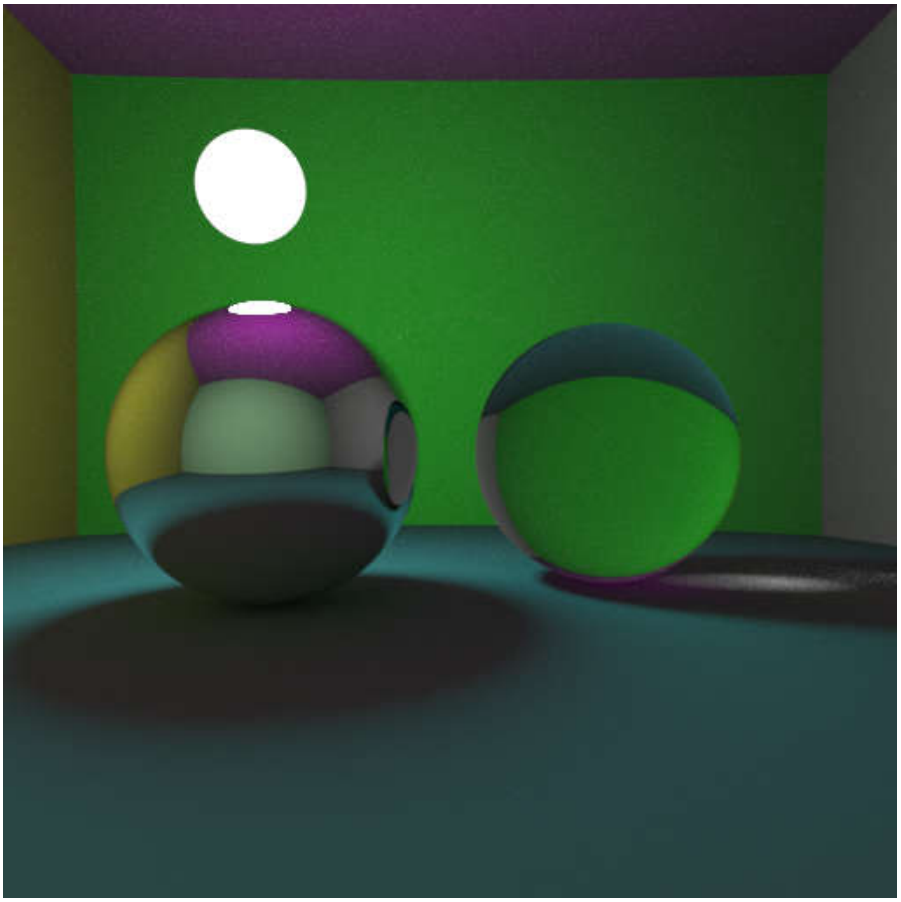
1000 tirages (3min 29s) - Avec profondeur de champ

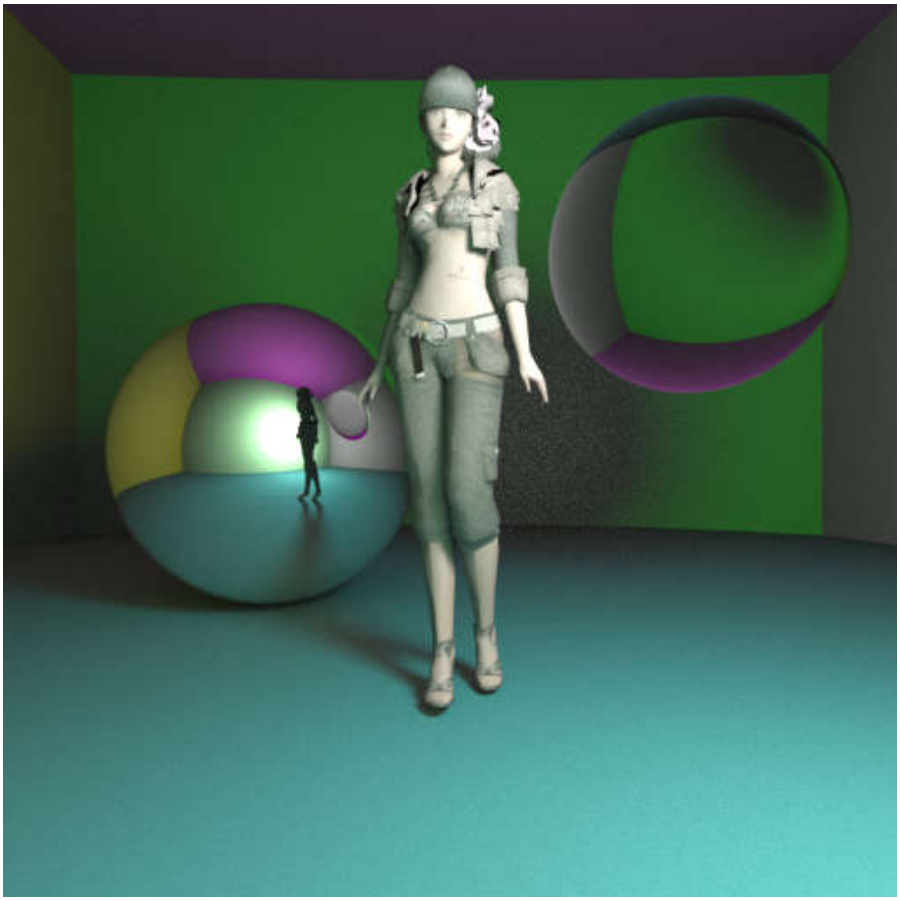
De même, sur les deux photos ci-dessus, on a volontairement placé la sphère bleue à distance de mise au point, la sphère jaune avant la distance de mise au point et la sphère rose après la distance de mise au point. On constate également que la sphère bleue est bien nette tandis que les sphères jaune et rose sont floues, de même que les murs en arrière-plan.

9 Sélection d'Images

Cette section contient une petite sélection d'images réalisées avec les procédures décrites ci-dessus. En raison de l'altération des images dans le rapport dû à l'export au format PDF, les images incluses dans ce rapport seront aussi jointes au format PNG afin de pouvoir les consulter dans leur qualité d'origine.







Conclusion

Au cours de ce TP, nous avons pu expérimenter et implémenter des un algorithme de Ray Tracing à travers plusieurs de ses aspects tels les équations de rendu, l'anti-aliasing, la profondeur de champ, les propriétés de surface et les interactions entre rayons et surfaces. Nous avons également appris à traiter les maillages en travaillant notamment sur les textures et sur des structures d'accélération permettant de diminuer les nombre d'opérations superflues.