Rapport du Projet

MOS 2.2 Informatique Graphique

Projet Informatique Graphique

Auteurs :
M^me^ Rima EL KAMEL

Responsables :
Pr. Nicolas Bonneel

Année Universitaire :
2019-2020

Table des matières

Table des figures

Introduction

L'Informatique Graphique étudie la synthèse d'images virtuelles à partir de modèles d'objets tridimensionnels. C'est une discipline indispensable à l'industrie du loisir (effets spéciaux, films d'animation, jeux vidéo), au design industriel conception et visualisation de prototypes) et à la réalité virtuelle (simulateurs de vol et de conduite, visites virtuelles interactives).

En informatique, une interface graphique (en anglais GUI pour graphical user interface) ou un environnement graphique est un dispositif de dialogue homme-machine, dans lequel les objets à manipuler sont dessinés sous forme de pictogrammes à l'écran, de sorte que l'usager peut utiliser en imitant la manipulation physique de ces objets avec un dispositif de pointage, le plus souvent une souris.

Ce domaine, assez récent, constitue un moteur puissant dans la réalisation des jeux vidéos, des films, des visites virtuelles, ...

L'objectif de ce projet est de découvrir comment faire les lancer de rayons ou ce qu'on appelle le raytracing.

Chapitre 1

Initiation

1.1 Définition

L'infographie signifie dessiner des images sur un écran d'ordinateur.
Le lancer de rayons est une technique pour générer une image en traçant le chemin de la lumière à travers les pixels dans un plan d'image. La technique est capable de produire un très haut degré de photoréalisme ; généralement supérieur à celui des méthodes de rendu de ligne de balayage typiques, mais à un coût de calcul plus élevé.

1.2 Principe

L'idée du lancer de rayons est de calculer la couleur de chaque pixel indépendamment (un pixel étant un point sur la rétine) en reconstruisant le trajet inverse de la lumière. C'est un rendu réaliste malgré qu'il est plus lent. On part de l'oeil et on détecte où ce rayon va tomber dans la scène. Si ce rayon tombe sur un objet réfléchissant, on fait rebondir le rayon (réflexion) et on continue. Si ce rayon tombe sur un objet transparent, on le traverse (réfraction). Si ce rayon tombe sur un objet mat on calcule la couleur en regardant où sont les sources de lumière et on s'arrête (diffusion). Cette méthode permet par exemple de générer des images de synthèses pour des films, ou faire des simulations d'éclairage pour le domaine d'architecture.

Ce projet va nous permettre de se familiariser avec un nouveau langage C++, que je l'utiliserai pour la première fois, de créer des objets pour prendre en main le concept du raytracing et la manipulation de quelques objets graphiques pour avoir à la fin un rendu graphique assez satisfaisant.

Le principe du raytracing est de simuler le comportement de la lumière. J'ai pris la photo qui est déjà dans le lien du cours pour pouvoir comprendre et assimiler le principe

du raytracing.

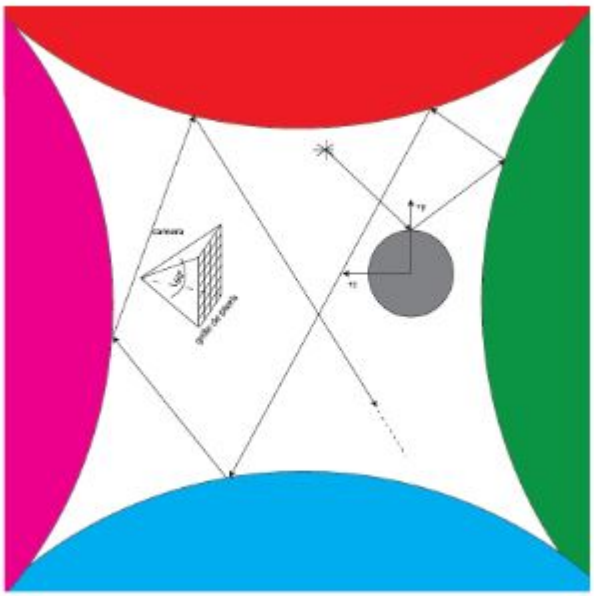

FIGURE 1.1 – Principe du raytracing

Ce que je vais faire en première partie, est de lancer des rayons depuis la caméra vers l'image pour pouvoir voir les interactions comme expliquées dans le cours.

Conclusion

Ce chapitre avait pour but d'introduire le cours de ce mos, et pour la suite je vais aborder la partie code et explication à chaque fois de la partie codée.

Chapitre 2

Réalisations

2.1 Sphères

En raytracing, on a des primitives données par des équations => le but est d'intercepter des rayons lumières avec des sphères. J'ai commencé par le développement de quelques classes de bases de géométrie utiles pour le développement des sphères : Nous avons premièrement la classe Vector qui présente des opérations élémentaires pour de vecteurs. Nous disposons des principales classes suivantes :

- La classe Ray : pour gérer les rayons.
- La classe Vector : pour gérer les vecteurs ayant des coordonnées x, y et z.
- La classe Sphère : pour créer des sphères et qui est définie par une origine et un rayon.

Une boucle qui balaye les pixels de l'écran (hauteur de l'image et largeur de l'image) on génère un rayon R qui aura une origine et une direction(direction du pixel) et tester s'il y a une intersection entre le rayon et la sphère.(bool inter= if_intersect(R,s)) s'il y a une intersection alors le pixel va être en blanc sinon le pixel sera noir. La première image obtenue est la suivante :

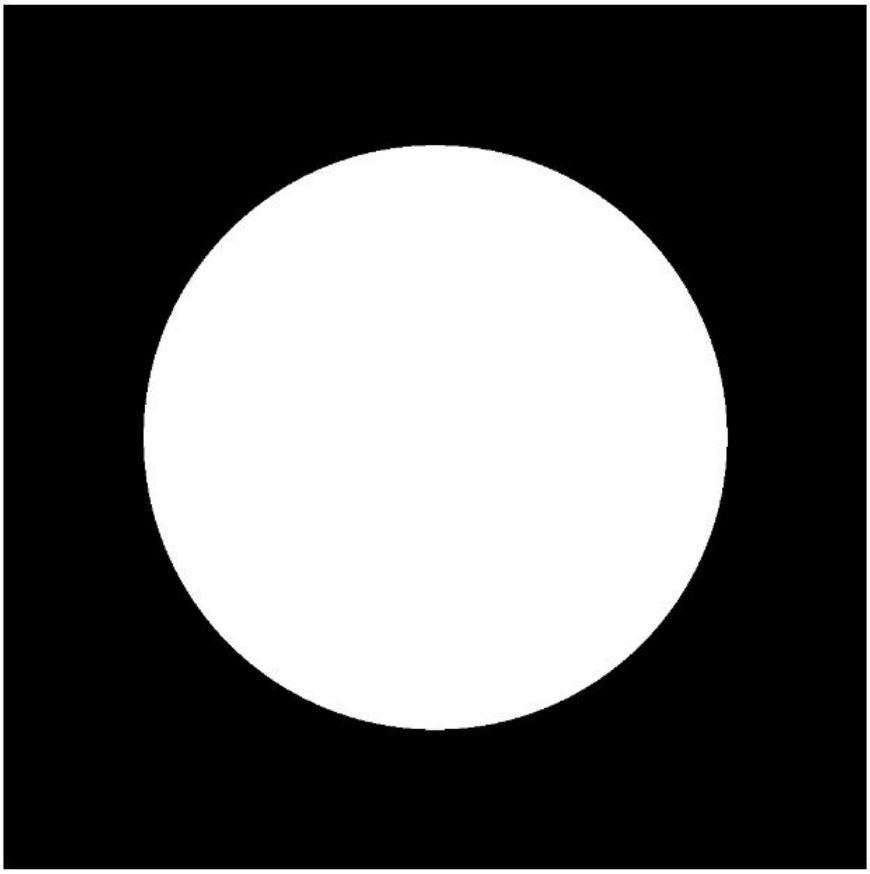

FIGURE 2.1 – Intersection Rayon-Sphère

Il s'agit d'un disque : une sphère illuminée de manière uniforme et s'il y aura une intersection du rayon avec la sphère , on met le pixel en blanc sinon en noir.

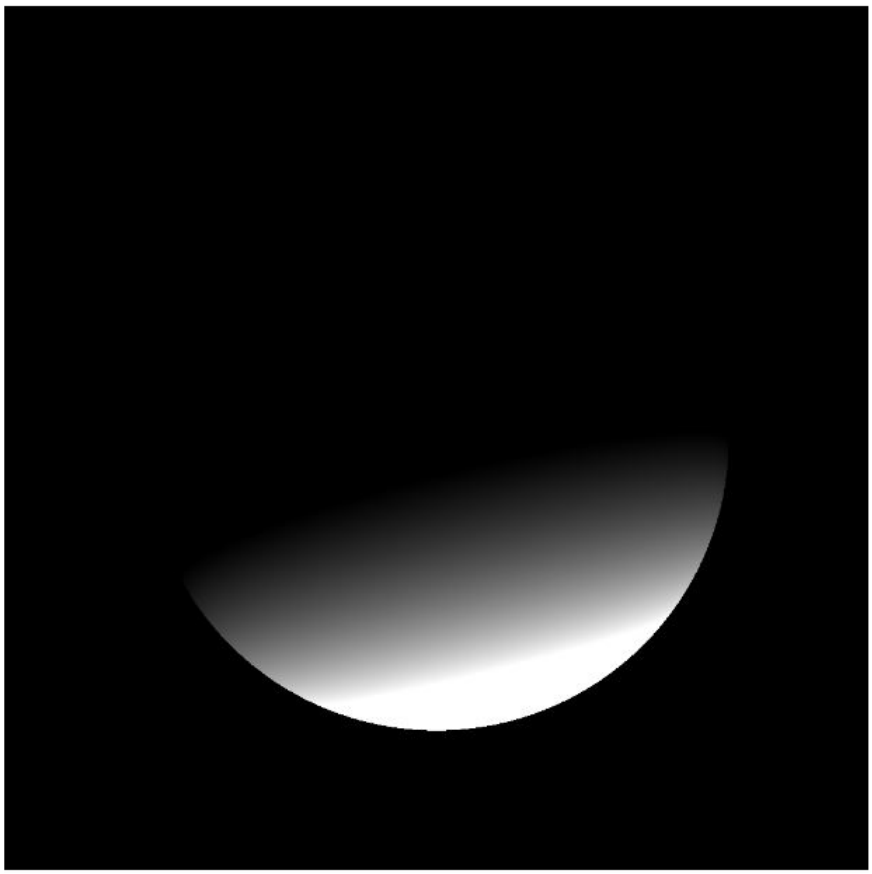

FIGURE 2.2 – Ajout de l'intensité des pixels

En jouant avec certains paramètres comme la luminosité, la couleurs la sphère, j'ai pu obtenir ces 3 images :

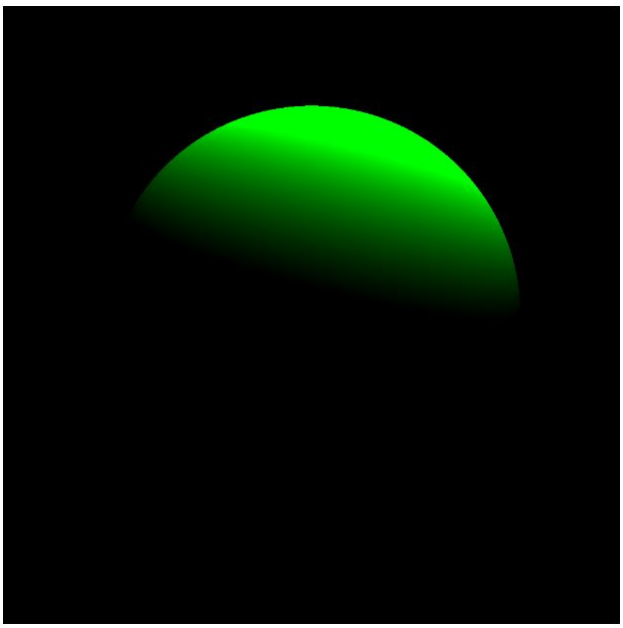

FIGURE 2.3

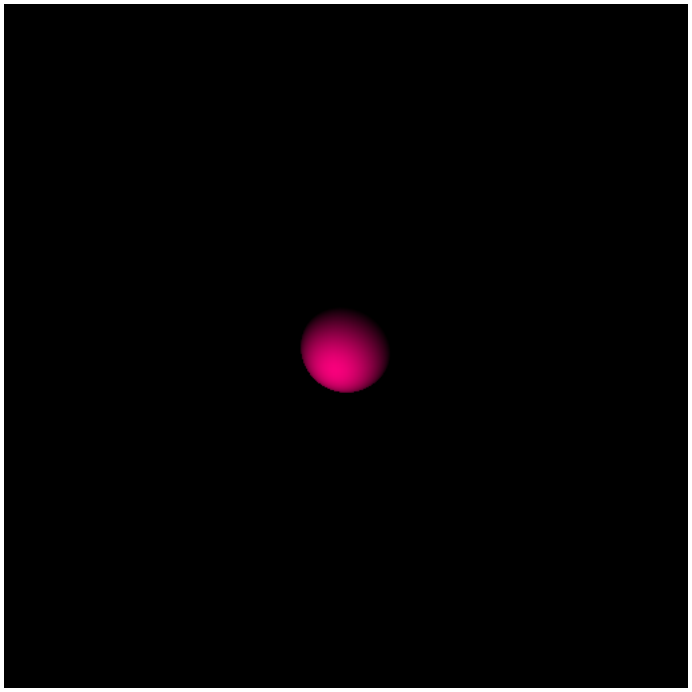

FIGURE 2.4

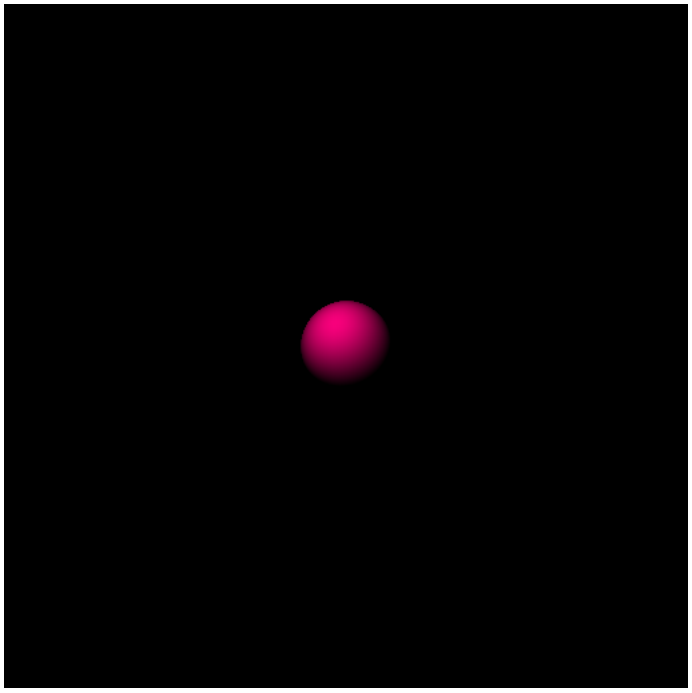

FIGURE 2.5

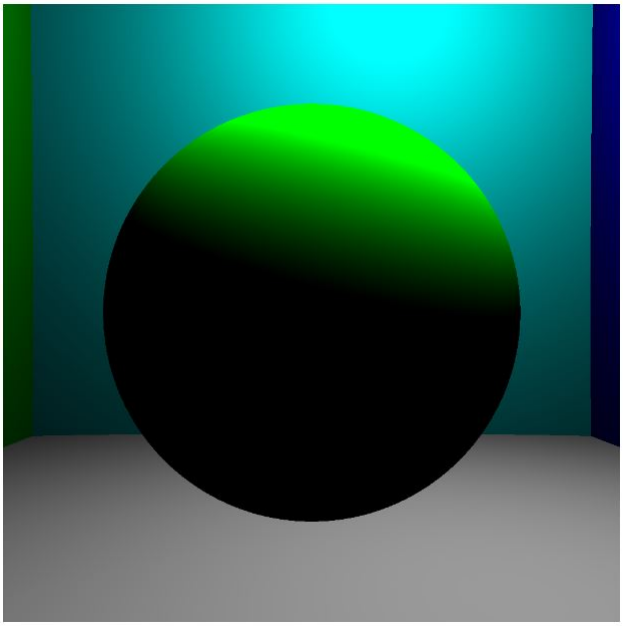

FIGURE 2.6 – Scène de sphère

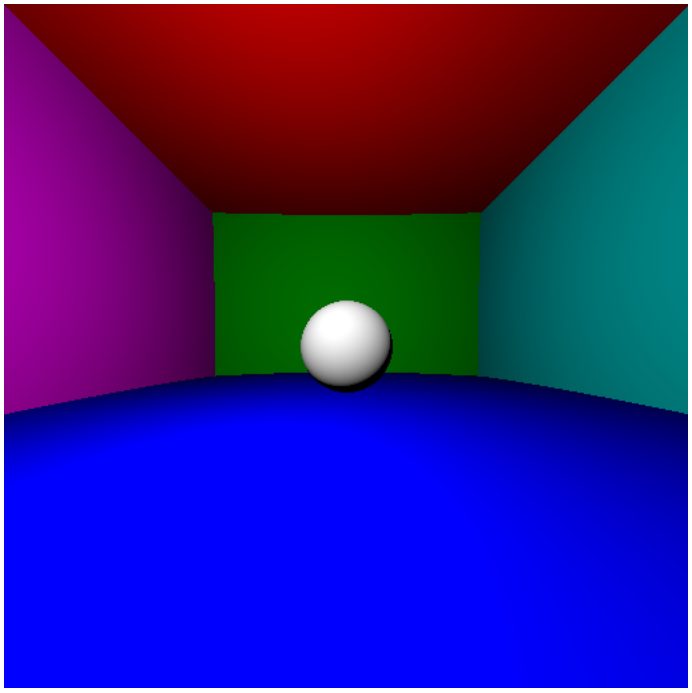

FIGURE 2.7 – Scène de sphère en faisant apparaître les différents plans (Plafond, gauche, droite, fond, sol)

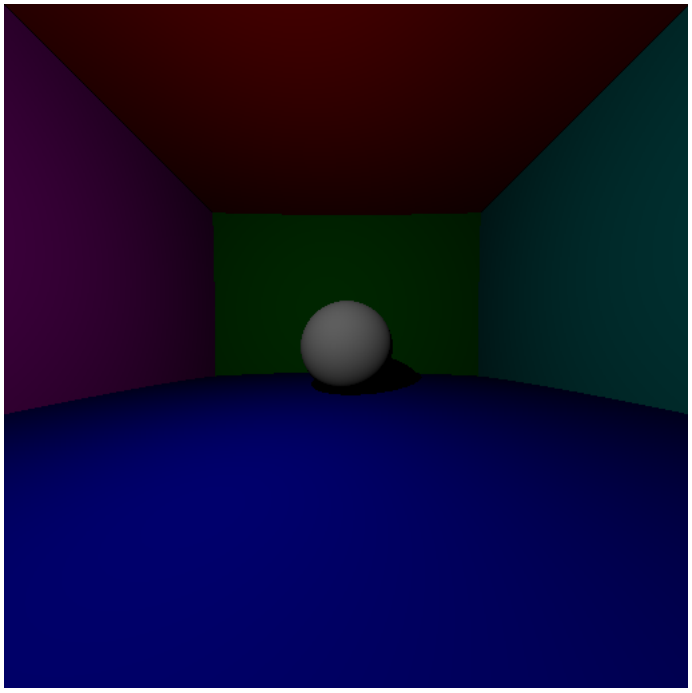

FIGURE 2.8 – La même scène que la précédente en jouant sur l'intensité pour voir l'ombre de la sphère principale de la scène

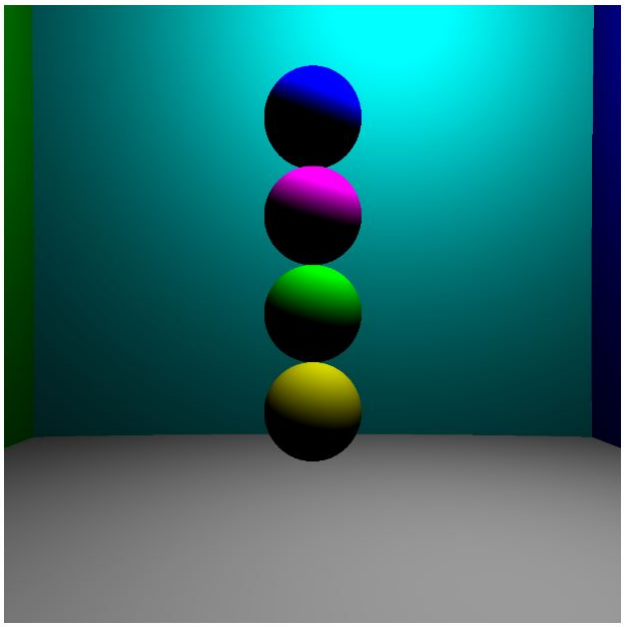

FIGURE 2.9 – Une scène avec 9 sphères

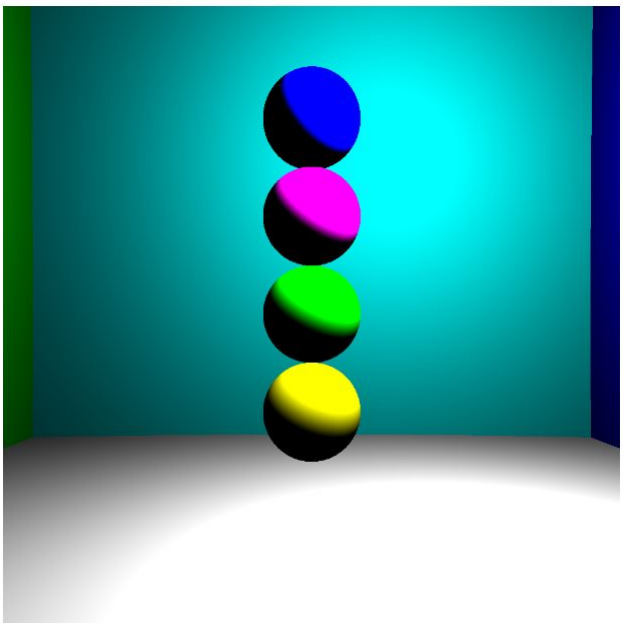

FIGURE 2.10 – Une scène avec 9 sphères en changeant la position de lumière pour voir son influence sur la coloration des sphères

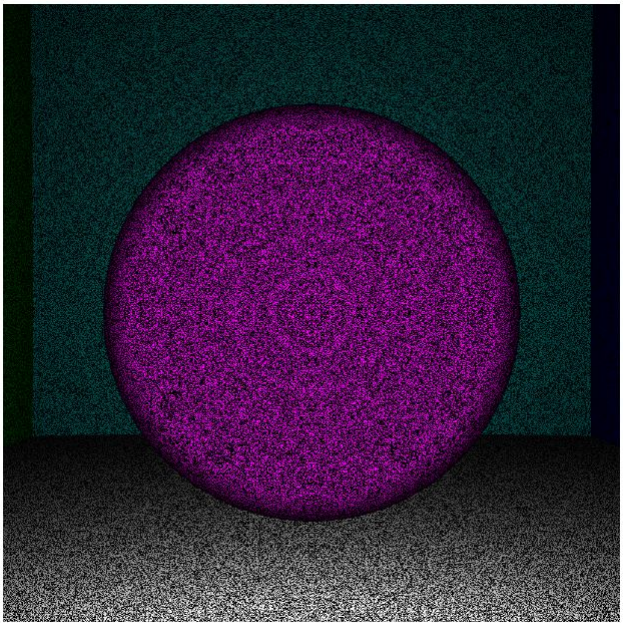

FIGURE 2.11 – Image bruité à cause au problème de précision numérique lors du calcul de l'intersection

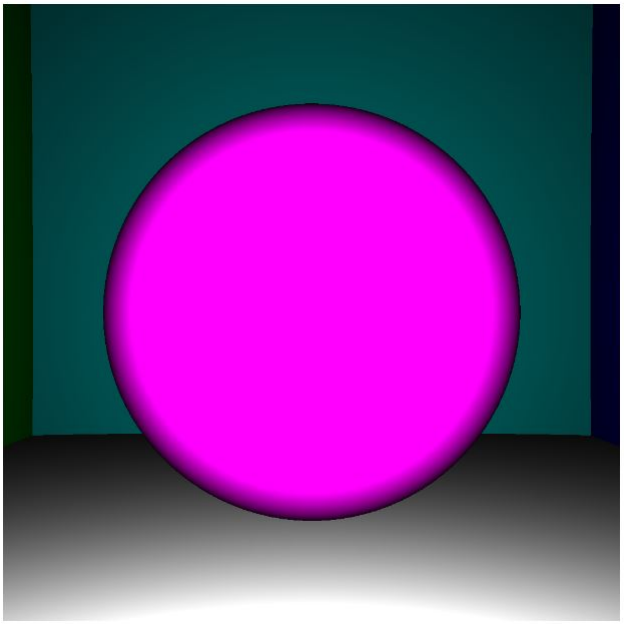

FIGURE 2.12 – Image corrigée

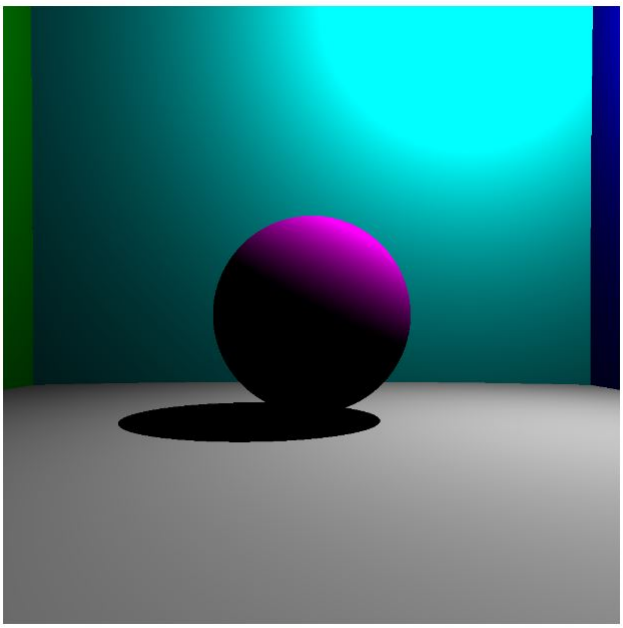

FIGURE 2.13 – Image corrigée avec modification de la position du lumière pour une meilleure visualisation de la sphère ombrée et modification de la taille de la sphère pour un rendu meilleur

Les intensités lumineuses affichées à l'écran ne semblent pas varier linéairement avec les valeurs qu'on lui donne : une intensité de 127 ne semble pas à mi-chemin entre du noir pur (0) et du blanc pur (255). En fait, les écrans ont un facteur gamma : une fonction de type luminosité = intensitéĝamma, où gamma est souvent pris comme le coefficient 2.2. Nous avons fait la correction gamma pour régler le problème des intensités.

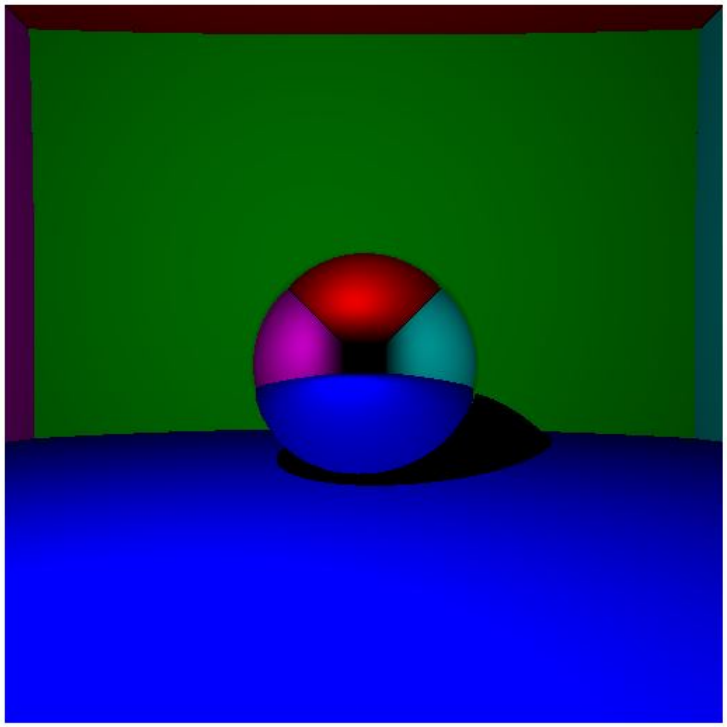

FIGURE 2.14 – Effet miroir d'une sphère

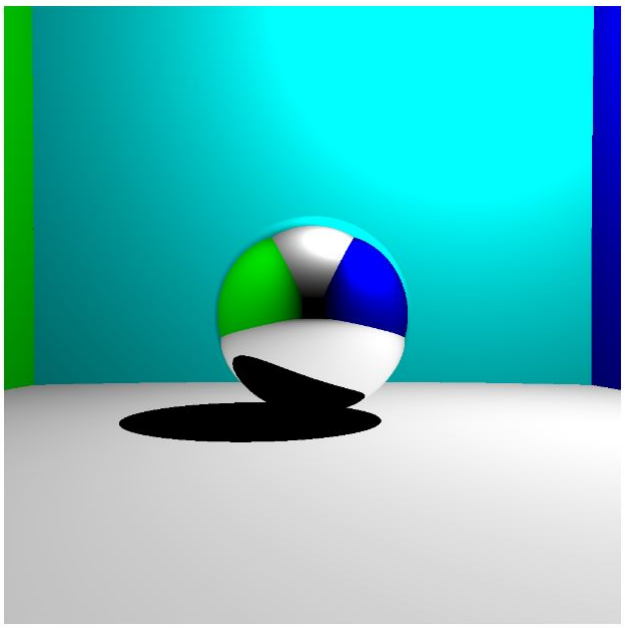

FIGURE 2.15 – Effet miroir d'une sphère en changeant les couleurs des sphères considérées comme plan, la position de la lumière et l'intensité de lumière

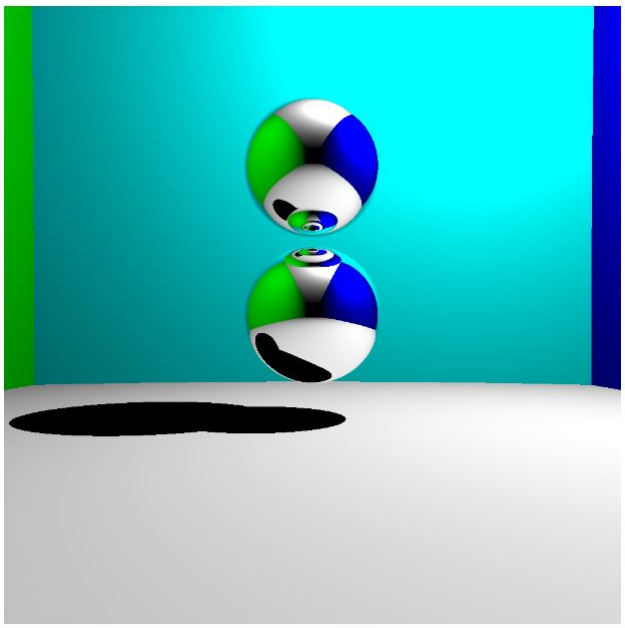

FIGURE 2.16 – Deux sphères miroirs : les petites gouttes en haut et en bas des deux sphères reviennent au fait qu'on a en réalité un nombre infini de reflexions et dans notre code nous avons stoppé la récursivité de la fonction rendu_couleur.

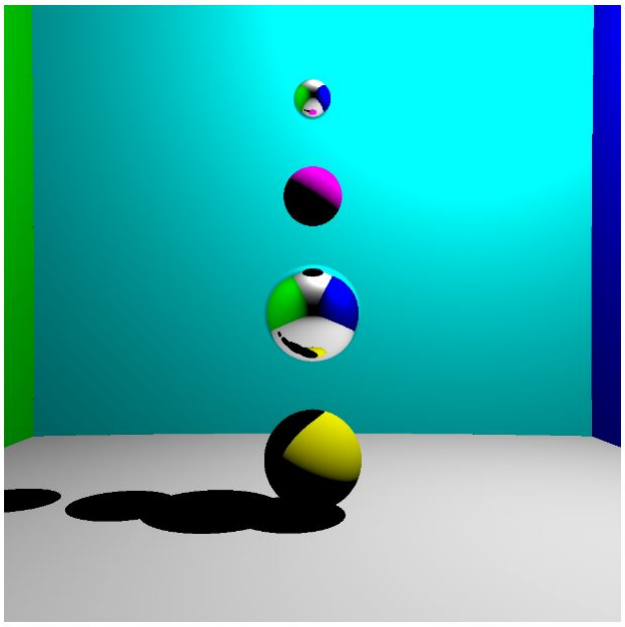

FIGURE 2.17 – Utilisation de 4 sphères avec 2 en miroir et 2 non

Le principe de la transparence suit celui de la réfraction, en d'autres terme cette figure

illustre bien ce principe :

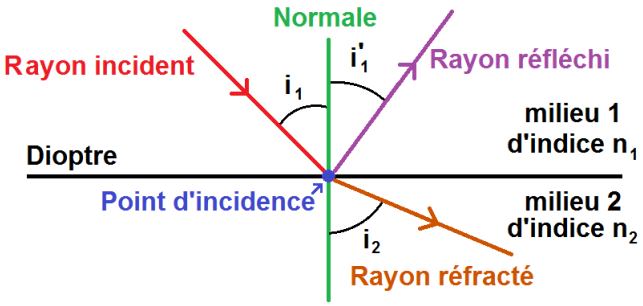

FIGURE 2.18 – Principe de réfraction

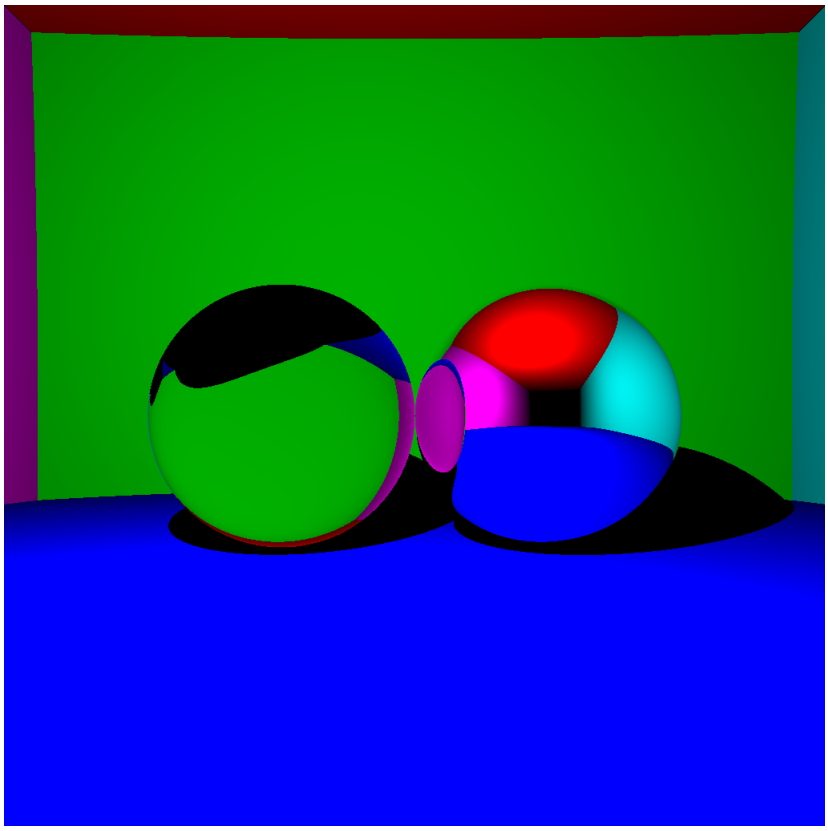

FIGURE 2.19 – Deux sphères une miroir et une transparente en changeant les couleurs

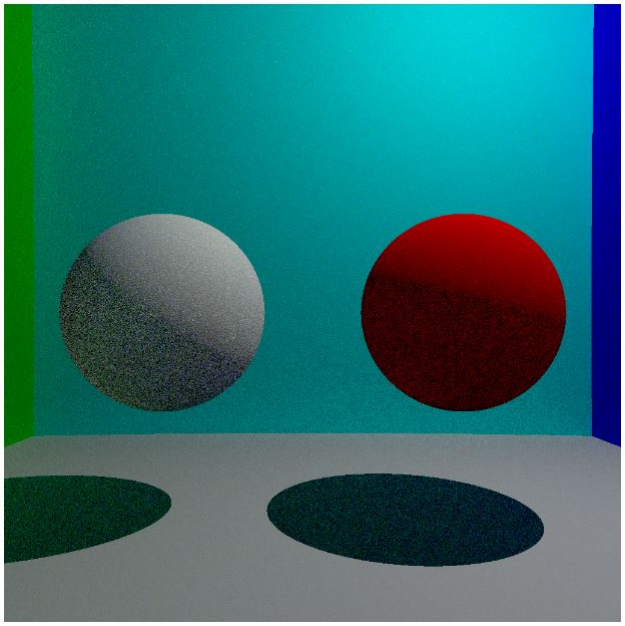

FIGURE 2.20 – Ajout de l'éclairage indirect

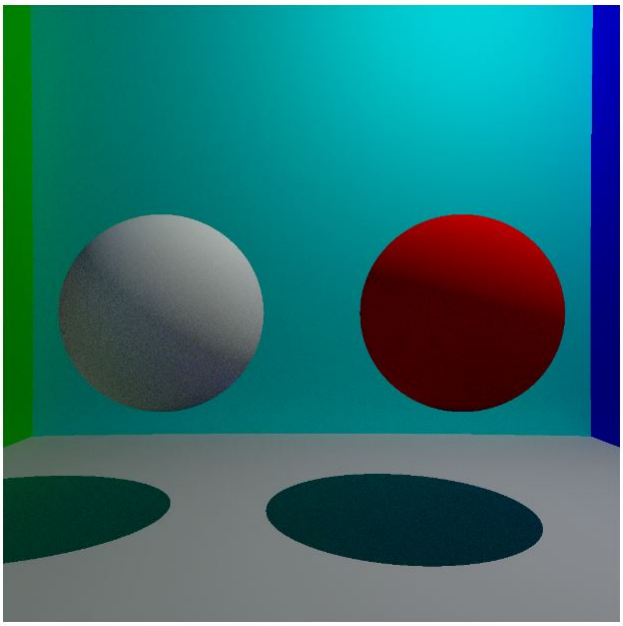

FIGURE 2.21 – Ajout de l'éclairage indirect avec un nombre de rayons plus élevé pour un rendu moins bruité

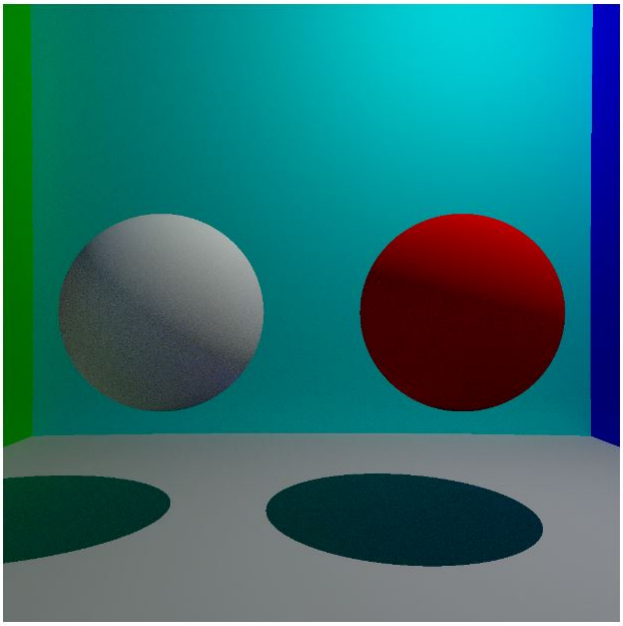

FIGURE 2.22 – Ajout de la classe Triangle pour avoir une autre forme

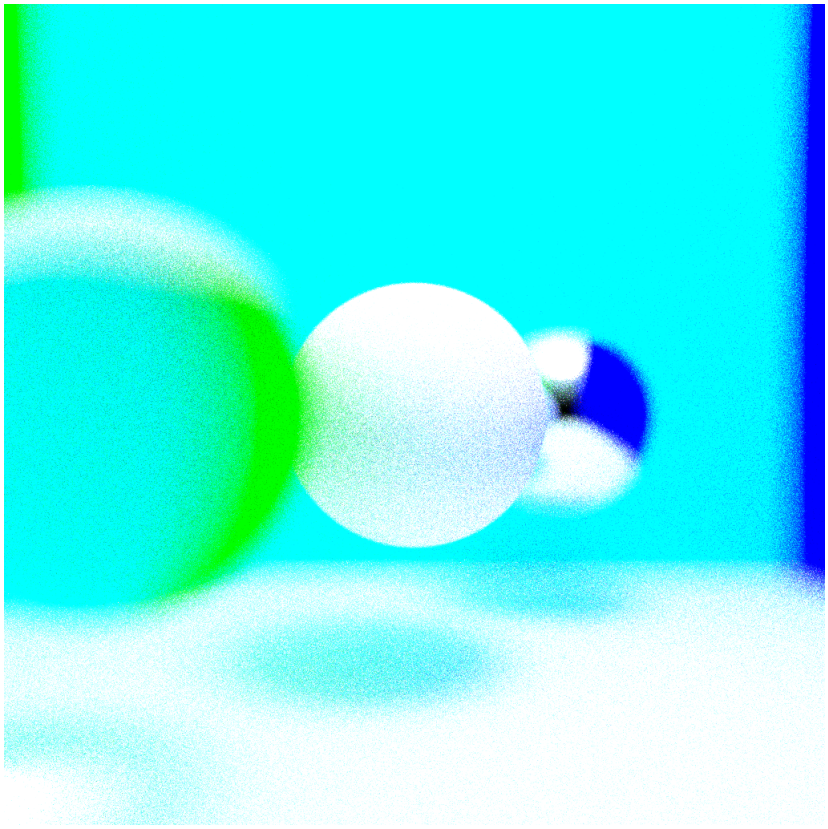

FIGURE 2.23 – Effet de profondeur de champ

2.2 Triangle et Sphère

Nous avons ajouté dans cette partie un triangle à notre scène, (et on peut tout de même ajouter plusieurs formes géométriques à notre scène).

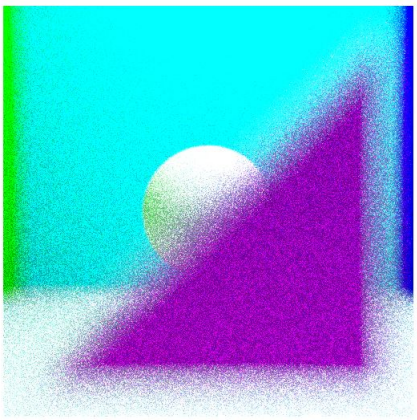

FIGURE 2.24 – Triangle ajouté à une scène

2.3 Beautiful Girl

Pour le traitement de l'objet de la fille, j'ai pu obtenir ce résultat et pour la partie de texture, le pc plante vu que la génération de l'image nécessite une puissance et plusieurs coeurs pour l'avoir. J'ai essayé tant de fois, le code ça compile bien mais le pc plante lors de la génération de l'image. La seule image obtenue est la suivante :

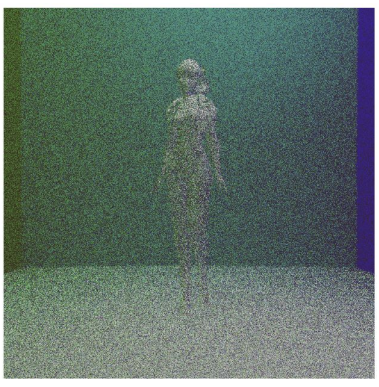

FIGURE 2.25 – Effet de profondeur de champ

Conclusion

Ce cours était pour moi assez intéressant, vu que c'est ma première expérience avec le langage c++. J'ai pu appronfondir mes connaissances pour ce langage orienté objet, j'ai essayé d'avoir une décomposition du code assez adéquate malgré que dans certaines parties j'ai pas trouvé la bonne manière pour architecturer le code, mais il marchait très bien.

Ce domaine aussi de l'informatique graphique qui me permet d'aborder le principe de lancer de rayons était pour moi très constructive.

En effet, l'informatique graphique est indispensable au prototypage virtuel industriel (design d'objets ou de mécanismes destinés à être fabriqués), aux simulateurs d'apprentissage et autres "serious games", ainsi qu'à la visualisation scientifique, par exemple pour l'exploration visuelle de données ou de résultats de simulation.