Ecole Centrale de Lyon

Option : informatique

Rapport informatique graphique

Fait par :

Aymane LAMHARZI ALAOUI

Encadré par :

M. Nicolas BONNEEL

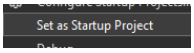
Table des matières

1 Introduction

Ce rapport portera sur le projet de Ray tracing fait au MOS 2.2 Informatique Graphique à l'Ecole Centrale de Lyon.

Le ray tracing est utilisé dans la création d'images 3D. c'est une technique de rendu utilisée dans la production d'images de synthèse pour simuler la façon dont la lumière se propage dans un environnement 3D. Cette technique permet de créer des images plus réalistes en prenant en compte les effets de la réflexion, de la réfraction et de la diffusion de la lumière sur les surfaces des objets. Dans cette rapport, nous allons explorer quelques implémentations du Ray Tracing.

2 La solution Github

Dans le projet Github, vous trouverez une solution contenant 8 projets différents, chacun représentant une implémentation de la technique de Ray Tracing. Pour sélectionner le projet à utiliser, vous devrez cliquer sur le fichier "set as starting project". 

Dans certains projets, vous trouverez également du code commenté. Comme il n'est possible d'utiliser qu'une seule fonction principale dans chaque projet, j'ai pensé que cette méthode était la plus appropriée. Pour utiliser le code commenté, il vous suffira de décommenter cette section de code et de commenter la section que vous ne souhaitez pas utiliser.

Chaque projet est conçu pour illustrer une technique ou un aspect spécifique du Ray Tracing vue en cours.

- Architecture du Github :

Pour savoir ce que fait le code commenté, voici l'architecture du projet Github :

1.Lightsphere :
   - Sphère en noir et blanc
   - Sphère en couleur

2.FirstScene :
   - Première implémentation d'une scène
   - Scène avec correction gamma
   - Scène avec ombres

3.SphereTextures :
   - Sphère miroir
   - Sphère transparente

4.Antialiasing :
- Ajout d'un éclairage indirect
- Ajout d'anti-aliasing

5.SoftShadows :
- Ajout d'ombres douces

6.DepthOfField :
- Essai de profondeur de champ

7.Maillage :
- Rendu de modèles 3D d'un fichier .OBJ

8.Textures :
- Ajout de textures au modèle

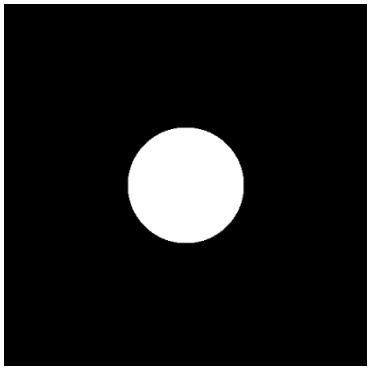
3 Code Review

1. Lightsphere :

La premiere partie de code crée une image d'une sphère blanche sur un fond noir en utilisant le lancer de rayons.

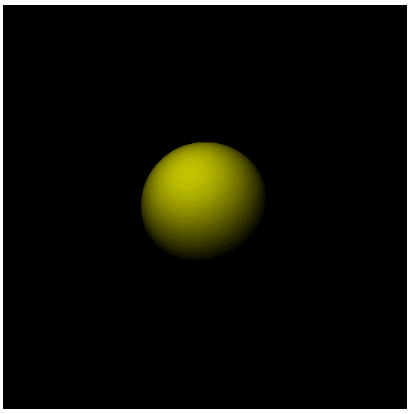
La classe Vector3 représente un vecteur tridimensionnel avec x, y et z comme coordonnées. La classe Ray représente un rayon avec une origine (position) et une direction. La classe Sphere représente une sphère avec un centre et un rayon.

La fonction principale crée une image en bouclant à travers chaque pixel de l'image. Pour chaque pixel, un rayon est lancé à partir de la position de la caméra vers la direction du pixel. Si le rayon intersecte la sphère, la couleur du pixel est définie sur blanc, sinon elle est noire. L'image est ensuite sauvegardée sous forme de fichier PNG avec le nom "whitesphere.png".



La deuxième partie du code crée une image d'une sphère colorée dans une scène illuminée en utilisant le lancer de rayons. Il améliore le code précédent en utilisant une structure de classes plus robuste et en calculant la couleur des pixels en fonction de la luminosité de la scène.

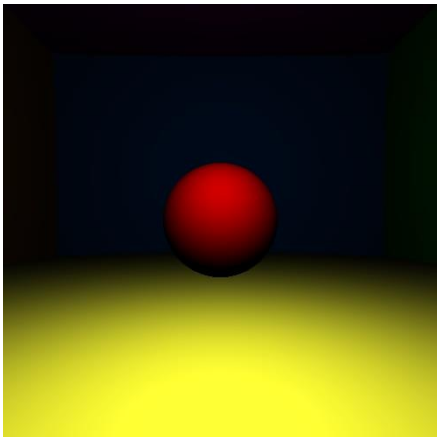
Si le rayon intersecte la sphère, la couleur du pixel est calculée en fonction de la luminosité de la scène et de la position de la source de lumière. L'image est ensuite sauvegardée sous forme de fichier PNG avec le nom "coloredsphere.png".



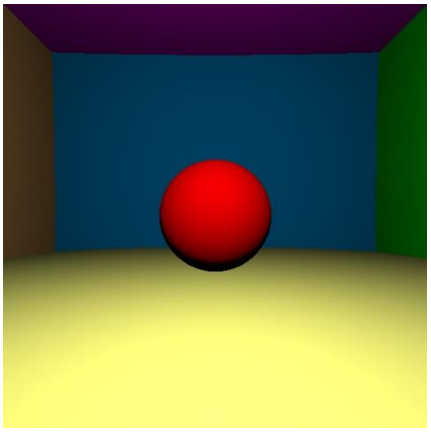
2. FirstScene

La premiere partie du code dessine une scène composée de plusieurs sphères.

Une classe Scene est définie pour représenter une scène qui peut contenir plusieurs sphères, et une méthode Intersection est définie dans cette classe pour calculer l'intersection entre un rayon et toutes les sphères de la scène, et renvoyer la position d'intersection, la normale à la sphère en cette position, et la couleur de la sphère la plus proche du rayon. Le reste du code implémente la boucle principale de rendu, qui itère sur tous les pixels de l'image, calcule le rayon correspondant à ce pixel, teste s'il intersecte un objet de la scène, et calcule la couleur du pixel en fonction de l'intersection. Enfin, l'image résultante est sauvegardée en format PNG à l'aide de la bibliothèque stb_image_write.h.



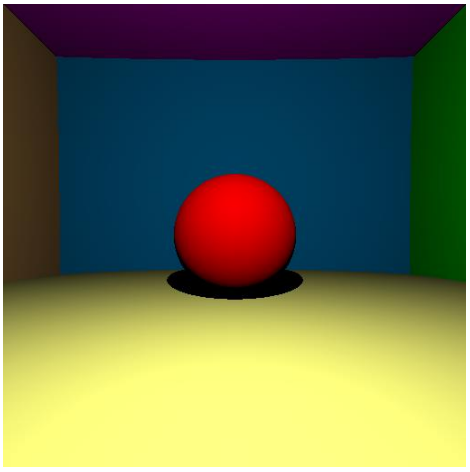
La deuxieme partie implement la correction du gamma :



Dans ce programme, l'ombrage est calculé en simulant la propagation de la lumière à partir d'une source de lumière ponctuelle située à une position fixe (dans ce cas, la variable L dans la fonction main). Pour chaque pixel de l'image, un rayon est lancé depuis la caméra vers la scène, et s'il rencontre un objet dans la scène, on calcule la couleur du pixel correspondant en fonction de l'interaction de la lumière avec cet objet.

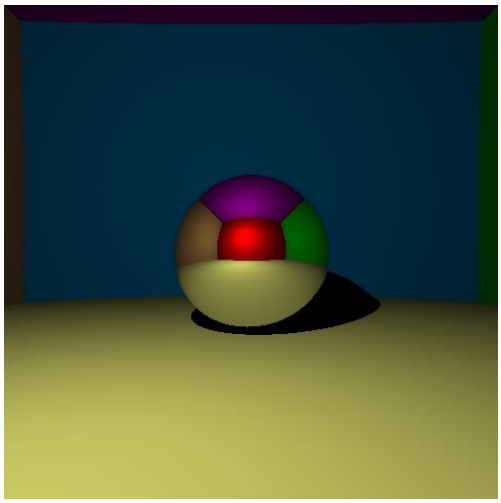
Pour calculer l'ombrage, on vérifie si le point d'intersection avec l'objet est directement éclairé par la source de lumière. Si ce n'est pas le cas, cela signifie que le point se trouve dans l'ombre d'un autre objet, et on ne prend pas en compte la contribution de la lumière directe dans le calcul de la couleur du pixel correspondant. Sinon, on calcule la contribution de la lumière directe en utilisant le modèle de réflexion de Lambert, qui suppose que la lumière est réfléchie uniformément dans toutes les directions par la surface de l'objet.

Pour détecter l'ombre, on lance un second rayon depuis le point d'intersection vers la source de lumière, et on vérifie s'il rencontre un autre objet avant d'atteindre la source de lumière. Si c'est le cas, cela signifie que le point se trouve dans l'ombre de cet objet, et on ne prend pas en compte la contribution de la lumière directe dans le calcul de la couleur du pixel correspondant.

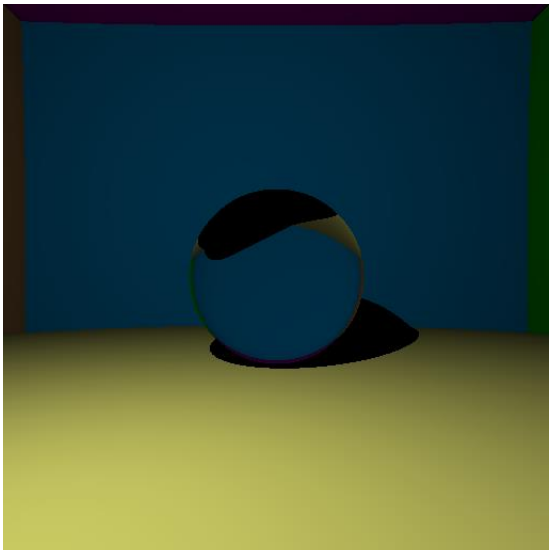


3. SphereTexture

Dans La première partie du code, l'attribut isMirror est un booléen de la classe Sphere qui est utilisé pour déterminer si la sphère est un miroir ou non. Si isMirror est vrai, cela signifie que la sphère est un miroir et la fonction getColour gère la réflexion de la lumière sur cette sphère en calculant le rayon réfléchi et en appelant récursivement getColour avec ce rayon réfléchi et une profondeur de récursion accrue.



La deuxième partie fait la même chose mais en rendant la sphère transparente :

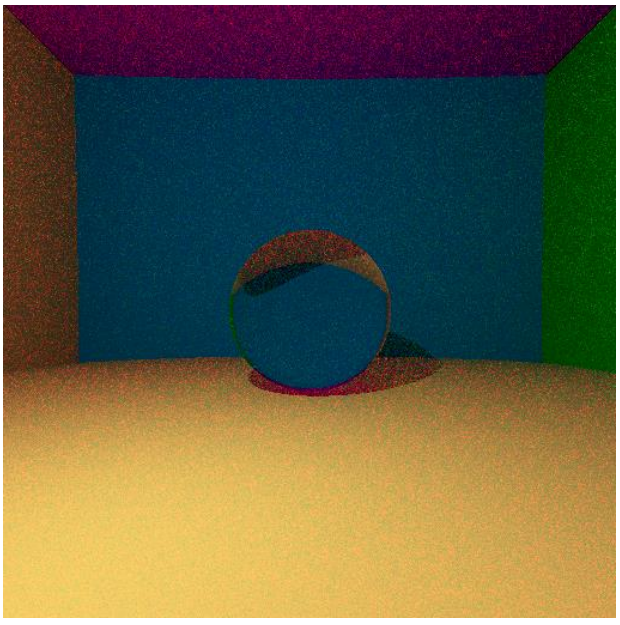


4. Antialiasing :

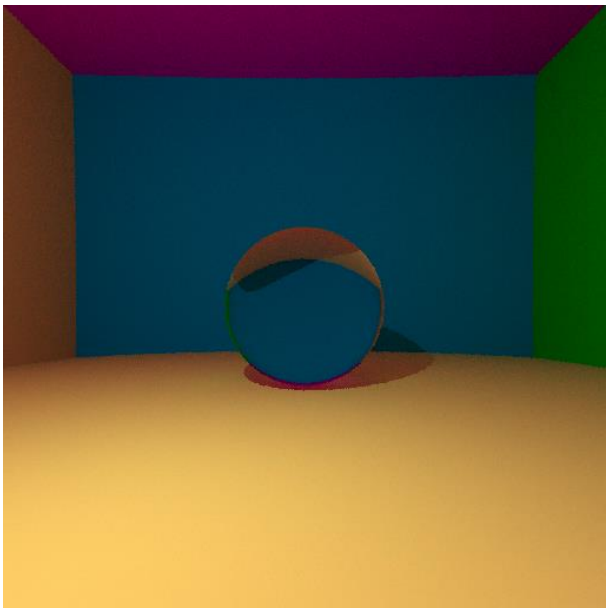
Dans ce code, l'éclairage direct est calculé à partir des sources de lumière visibles depuis le point d'intersection, ce qui signifie que la lumière qui frappe la surface est directement visible. Dans la fonction getColour, pour chaque rayon incident, si l'objet intersecté est réfléchissant, un rayon est tracé dans la direction de réflexion et la fonction getColour est récursivement appelée avec le rayon réfléchi pour obtenir la contribution de la réflexion spéculaire.

Si l'objet n'est pas réfléchissant, la fonction calcule l'éclairage direct en vérifiant si l'objet est ombragé par un autre objet. Si tel est le cas, la contribution de la lumière est nulle, sinon la contribution est calculée en multipliant l'intensité de la lumière par la couleur de l'objet, puis par le produit scalaire entre la normale de la surface et le vecteur allant de la surface à la source de lumière. La contribution est ensuite multipliée par un facteur de normalisation pour prendre en compte l'angle d'ouverture de la source de lumière.

L'éclairage indirect, en revanche, prend en compte la lumière qui est réfléchie par les autres surfaces de la scène et atteint finalement la surface actuelle, après avoir subi des réflexions multiples. Dans la fonction getColour, si l'objet intersecté n'est pas réfléchissant, un rayon est tracé dans une direction aléatoire à partir du point d'intersection, et la fonction getColour est appelée récursivement avec le rayon aléatoire pour obtenir la contribution de l'éclairage indirect. La contribution de l'éclairage indirect est ajoutée à la contribution de l'éclairage direct pour obtenir la couleur finale de la surface.

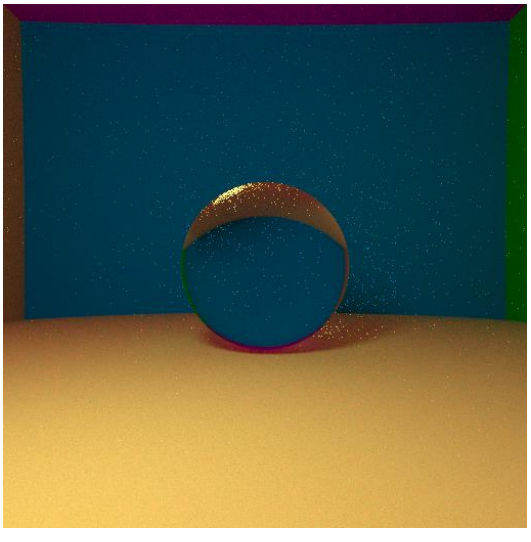


On applique par la suite l'antialiasing, pour faire cela, on utilise une technique appelée "échantillonnage de la fenêtre de Poisson". Au lieu de tirer un rayon unique pour chaque pixel, on tire plusieurs rayons autour de chaque pixel en utilisant une distribution aléatoire régulière. Ces rayons supplémentaires permettent de calculer une moyenne de couleurs pour chaque pixel, ce qui réduit l'aliasing (les effets d'escalier et les bords dentelés). Dans ce code, on tire nmbRays (100) rayons pour chaque pixel et on calcule la moyenne de leurs couleurs pour obtenir la couleur finale du pixel.



5. SoftShadows

Pour ajouter des ombres douces dans ce code de raytracing, on peut utiliser une technique de soft shadows appelée "distributed ray tracing". Cette technique consiste à tirer plusieurs rayons depuis la source de lumière vers un point donné sur l'objet, et à faire une moyenne des couleurs obtenues pour chaque rayon. Pour cela, on peut modifier la fonction "obtenircolor" de la classe Scene pour ajouter une boucle qui tire plusieurs rayons depuis la source de lumière. Ensuite, on peut utiliser la moyenne des couleurs obtenues pour chaque rayon pour déterminer la couleur finale du point.

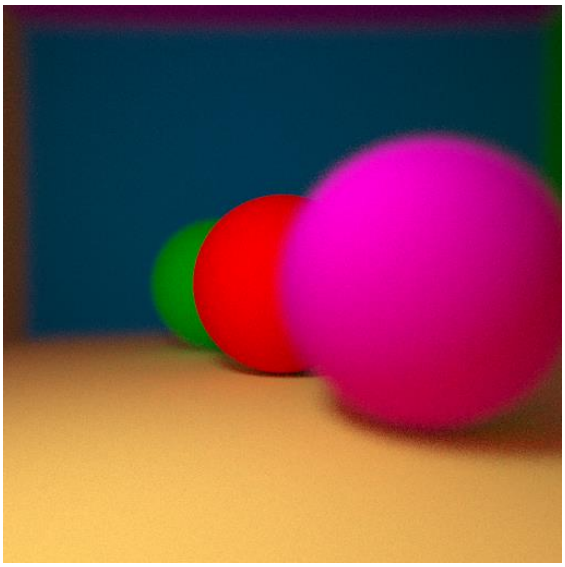


6. DepthOfField

Dans ce code, la profondeur de champ est implémentée en utilisant des rayons provenant de différentes directions. La variable nmbRay représente le nombre de rayons utilisés pour chaque pixel. Plus le nombre de rayons est élevé, plus l'image est nette.

Pour chaque pixel de l'image, le code calcule nmbRay rayons dans des directions différentes. Pour chaque rayon, il calcule le point de focalisation. La distance de focalisation est définie par la variable focalLength. Pour chaque rayon, le code calcule le point de focalisation en ajoutant une certaine quantité de bruit à la direction du rayon. La quantité de bruit est contrôlée par les variables u3 et u4 qui sont des nombres aléatoires uniformément distribués entre 0 et 1.

Une fois que le point de focalisation est calculé, le code crée un nouveau rayon en utilisant le point de focalisation comme point de départ et la direction originale du rayon comme direction. Le code calcule ensuite la couleur de ce nouveau rayon en utilisant la fonction GetColor qui calcule la couleur d'un rayon en fonction des objets de la scène.



7. Maillage

Les fichier .OBJ se trouvent sur « Gramophone retro free VR / AR / low-poly 3D model | CGTrader »

Une partie du code a été prise de ce github : « hedejing/ZJUCGFinal: ZJU Compute Graphics Final Project (github.com) » qui utilise les bounding boxes

Le fichier obj est un format de fichier de modèle 3D couramment utilisé pour stocker des données de modèles 3D. Il peut être ouvert et rendu dans divers logiciels de modélisation 3D.

Dans le code fourni, le fichier obj est lu en utilisant la fonction readOBJ qui prend en entrée le nom du fichier obj. Cette fonction utilise la bibliothèque standard C++ pour ouvrir le fichier et parcourir chaque ligne du fichier pour extraire les informations du modèle.

Les données extraites du fichier obj sont stockées dans les vecteurs vertices, normals, uvs et indices. Les données de vertex, de normale et de texture (si disponibles) sont stockées dans les vecteurs vertices, normals et uvs, respectivement. Les indices des sommets, des normales et des coordonnées de texture pour chaque triangle sont stockés dans le vecteur indices.

Pour afficher le modèle, le programme doit parcourir les triangles et les afficher un par un.

8. Textures

On ajoute par suite les textures :