ÉCOLE CENTRALE LYON

BE : INFORMATIQUE GRAPHIQUE

Réalisé par :Abouelfadl khaoula

ENCADRÉ PAR :PR NICOLAS BONNEEL

2022/2023

Table des matières

Raytracing

Création d'un cercle blanc

Le principe du raytracing repose sur l'envoi de rayons. Pour l'instant, notre scène ne contient qu'une seule sphère et chaque pixel de l'image reçoit un rayon. Si le rayon envoyé depuis la caméra entre en intersection avec la sphère, alors le pixel sera blanc, sinon il sera noir.

Les classes de base implémentées sont les suivantes :

- une classe **Vecteur** : elle contient les coordonnées du vecteur et différentes méthodes pour surcharger les opérations d'addition, soustraction, produit scalaire et produit par une constante ainsi que de calcul de la norme
- une classe **Sphere** : elle contient le centre et le rayon de la sphère
- une classe **Ray** : elle contient un point du rayon et son vecteur directeur.

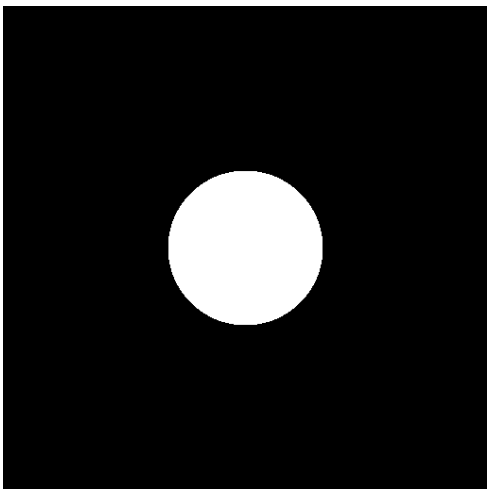

Figure 1:cercle blanc généré de cercle_blanc.cpp // Temps de calcul :1s

Ajout de l'intensité lumineuse

nous ajoutons une source de lumière ponctuelle de coordonnées L et d'intensité lumineuse I

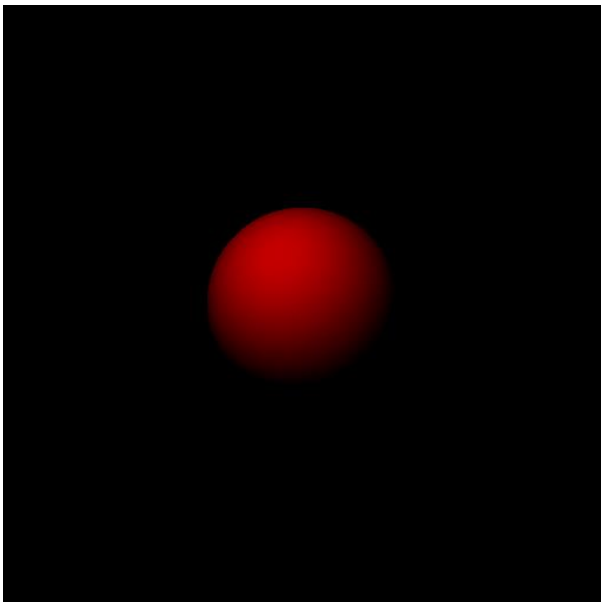

Figure 2:sphere génerée par image_lumiere.cpp// Temps de calcul :1s

Création d'une scène

Maintenant que nous avons établi l'éclairage direct sur notre sphère, nous allons définir le décor de la scène.

La scène aura un plafond et un sol ainsi que quatre murs . Les éléments de décor seront des sphères très éloignés
de la caméra ayant un rayon très important pour que nous ayant l'impression d'avoir des murs plans.

Cela nous permet de créer la classe « Scene » qui contient elle aussi une méthode ' intersect ' qui fera appelle à la méthode intersect des sphères qui la composent.

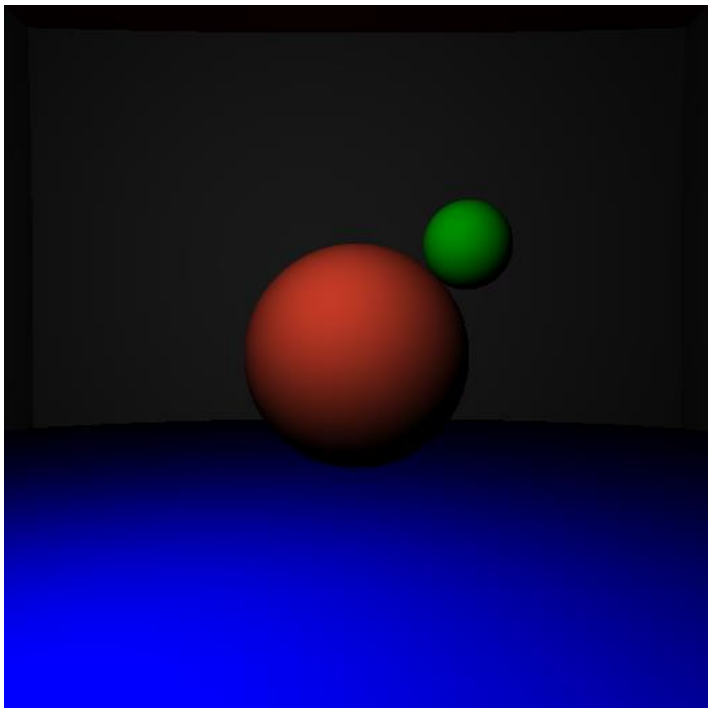

Figure 3:scene géneré de scene.cpp // Temps de calcul :1s

Correction Gamma

Nous avons réussi à afficher une scène constituée de plusieurs sphères. Cependant, nous avons du mal à distinguer les murs de la scène et les écrans appliquent un facteur 1,22 aux images générées. Pour cela, nous allons faire de la correction Gamma.

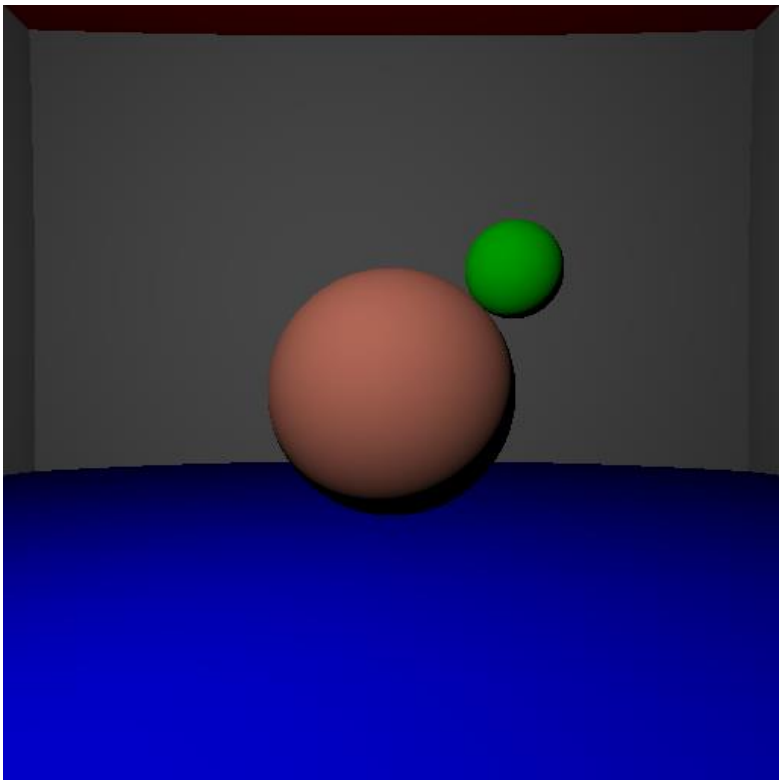

Figure 4:scene generé par correction gamma.cpp // Temps de calcul :1s

Ombres portées

Nous arrivons à afficher 2 sphères dans la scène mais comme tout objet éclairé, il y a une ombre portée qui lui est associée. S'il y a une intersection sphère-rayon, nous envoyons un rayon depuis ce point d'intersection P vers la source de lumière L. S'il y a une autre intersection située en P et L, le pixel où est situé P sera noir.

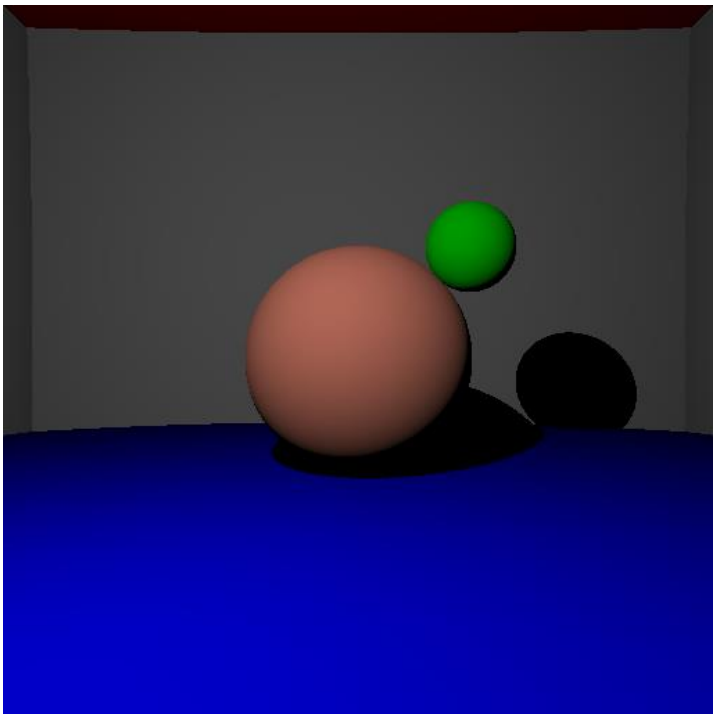

Figure 5:scène à ombre généré par ombre portee.cpp // Temps de calcul :1s

Surfaces miroir

Nous arrivons à afficher des sphères diffuses. Nous nous intéressons maintenant à des sphères miroirs. Un miroir signifie que nous devons prendre en compte le phénomène de réflexion.

Le calcul de la couleur des pixels de l'image est placé dans la méthode 'getColor' de la classe ' Scene 'afin de pouvoir l'appeler de manière récursive avec le rayon réfléchie.

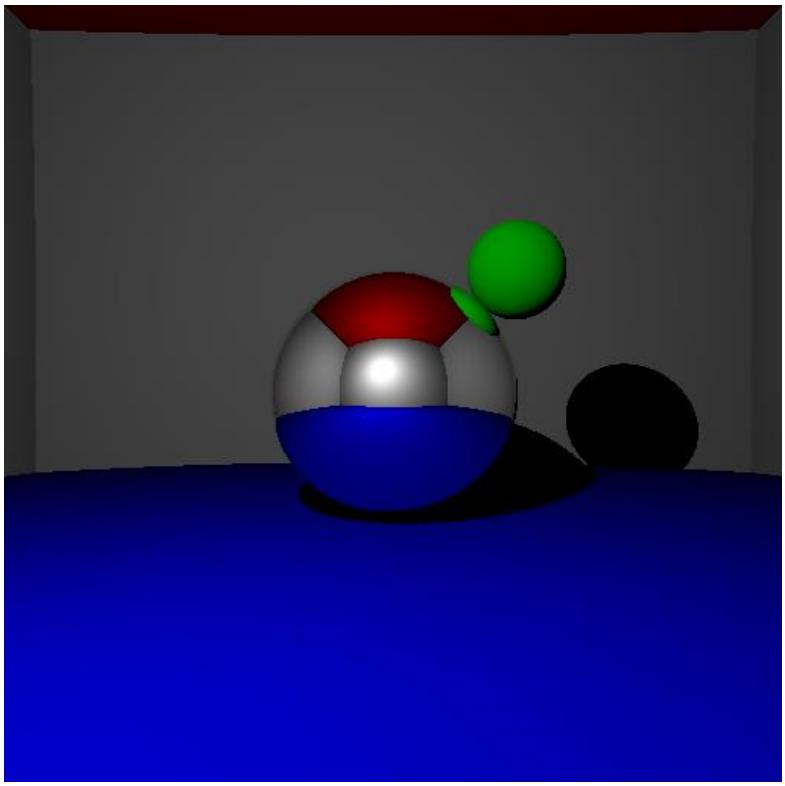

Figure 6:scene avec la sphere S1 mirroir génerée par image mirroire.cpp // Temps de calcul :1s

Surfaces transparentes

Pour le milieu transparent, nous appliquons la deuxième loi de Snell-Descartes qui nous détermine la direction du rayon réfracté. Nous supposons que le milieu transparent à pour indice optique celui du verre. Lorsque le rayon sort de la sphère, nous inversons les indices optiques et les normales à la surface de la sphère

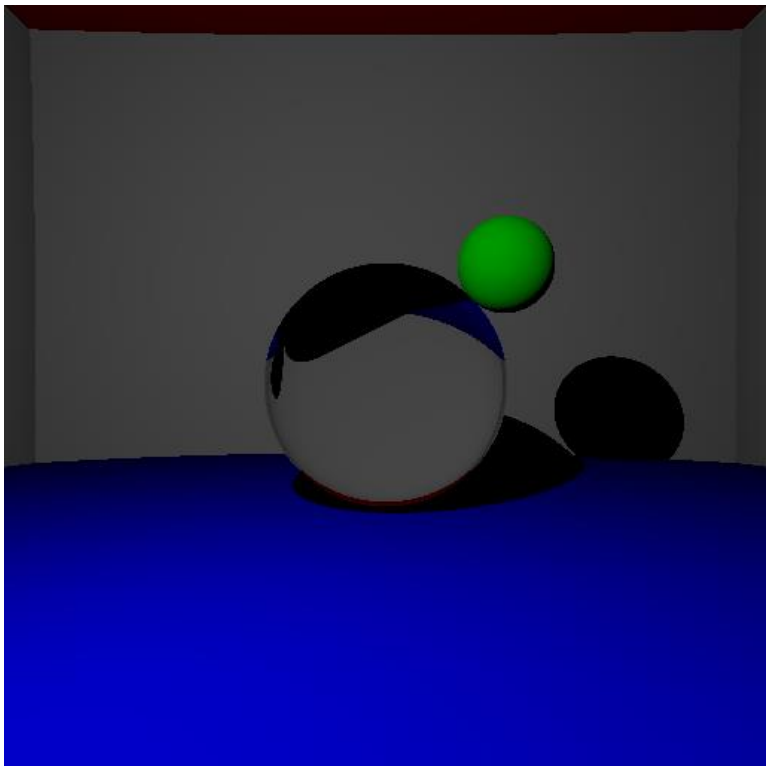

Figure 7:scene avec la sphere S1 transparente génerée par image transparente.cpp // Temps de calcul :1s

Eclairage indirect

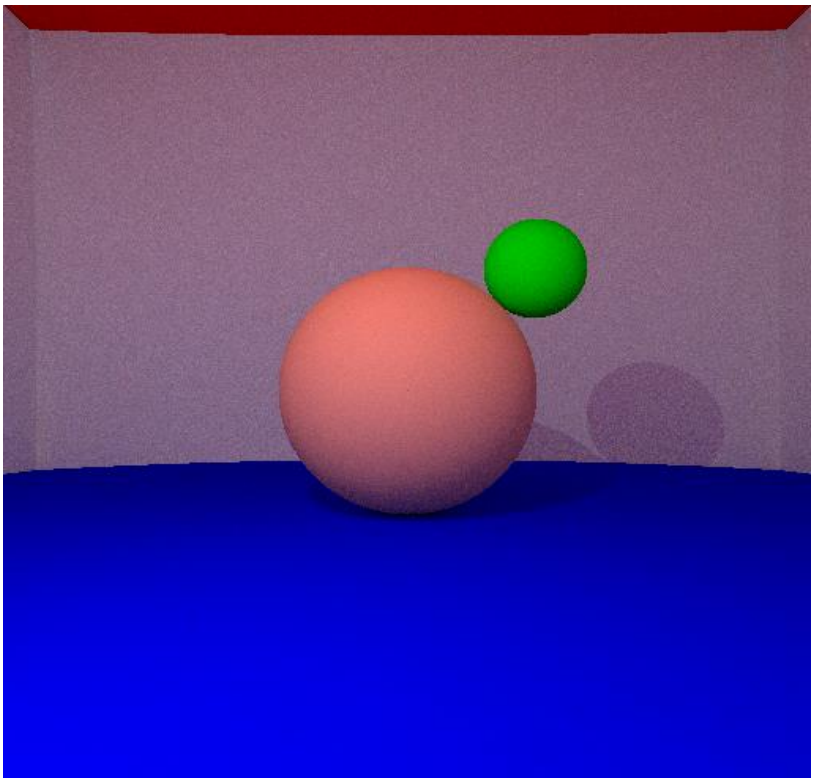

Figure 8:scene en lumiere indirect génerée par lumiere_indirect.cpp // Temps de calcul :4mn

Anti aliasing

En zoomant sur les bordures des images précédentes, nous pouvons observer un effet de créneau de château sur les courbes de la sphère. Cet effet visuel s'appelle le crénelage. Pour supprimer ce défaut, nous envoyons le rayon aléatoirement dans une zone proche du pixel ciblé. L'aléatoire utilisée repose sur des échantillons gaussiens obtenue par la méthode de Box-Muller.

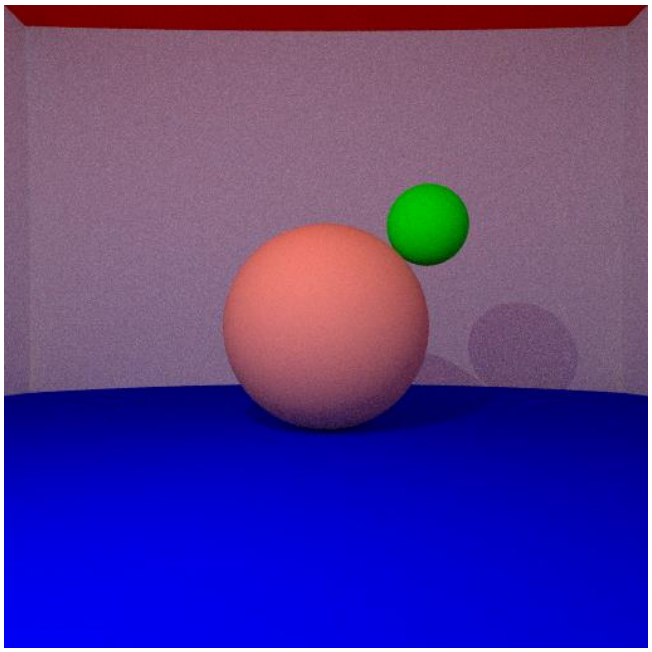

Figure 9:scene antialiasing genérée par antialiasing.cpp // Temps de calcul :6mn

Ombres douces

L'éclairage indirect que l'on a n'est pas suffisant. En effet les ombres ne sont pas assez douces. Pour pouvoir implémenter les ombres douces, il faut remplacer la lumière ponctuelle par une sphère de lumière. La première version de cette implémentation consiste à retirer la contribution de l'éclairage direct : il n'y a plus que l'éclairage indirect

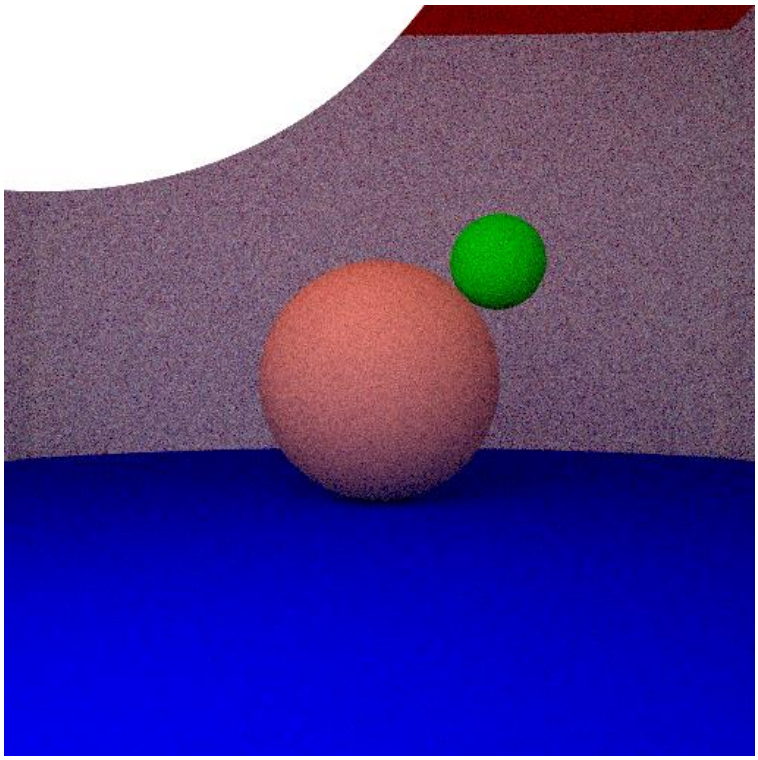

Figure 10:Scène à ombre douce//temps de calcul 7mn

Ombres douces améliorées

On réimplémente une sorte d'éclairage direct en plus de l'indirect :
cet éclairage direct est très similaire à celui qu'on utilisait lorsque la source de lumière était ponctuelle, sauf qu'on envoie un rayon aléatoire qui a plus de chance d'être dirigé vers la source de lumière, et on prend la couleur correspondante.

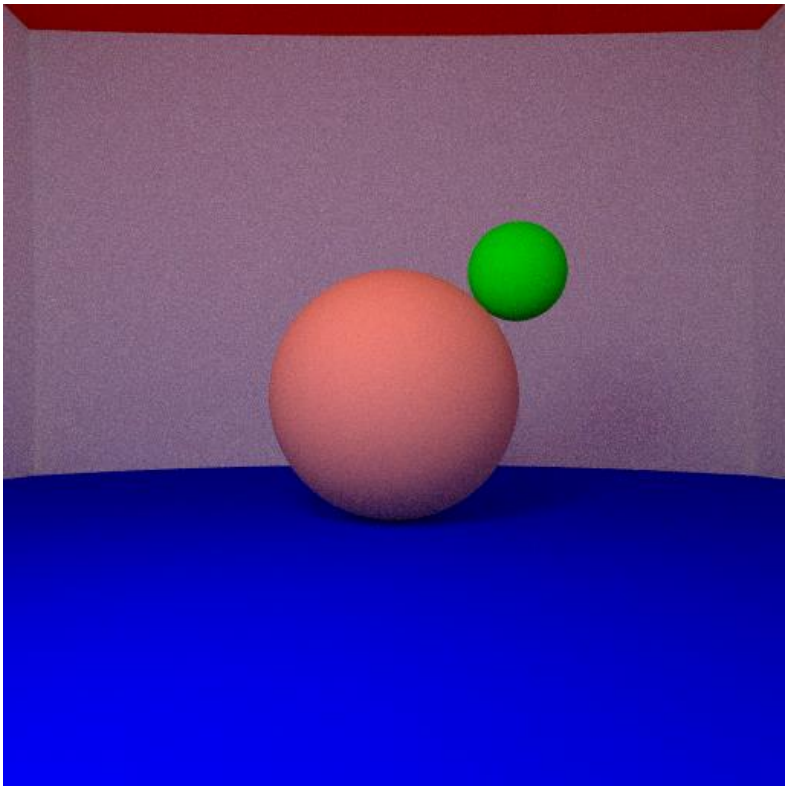

Figure 11:Scène à ombre douce améliorée//temps de calcul 12mn

Profondeur de champs

On peut également ajouter un effet de profondeur de champs.

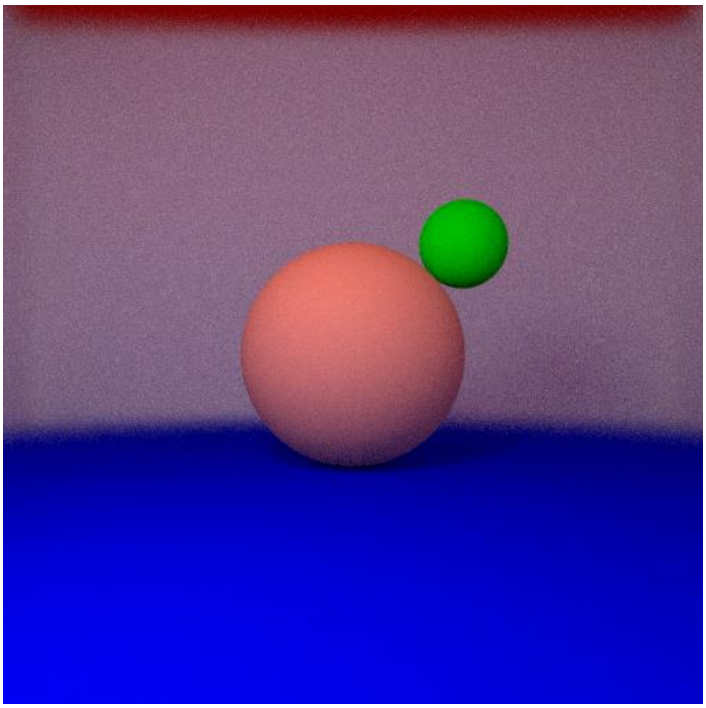

Figure 12: Scène à ombre douce améliorée à profondeur de champs//temps de calcul 12mn

Maillage Approche naïve

Pour le moment, nous n'avons fait que des sphères. Cependant, pour réaliser des objets plus complexes, cela n'est pas suffisant. On s'intéresse donc à des maillages qui sont plusieurs triangles formant ensemble un objet complexe. Nous avons téléchargé le maillage d'un chat et nous allons essayer de le représenter.

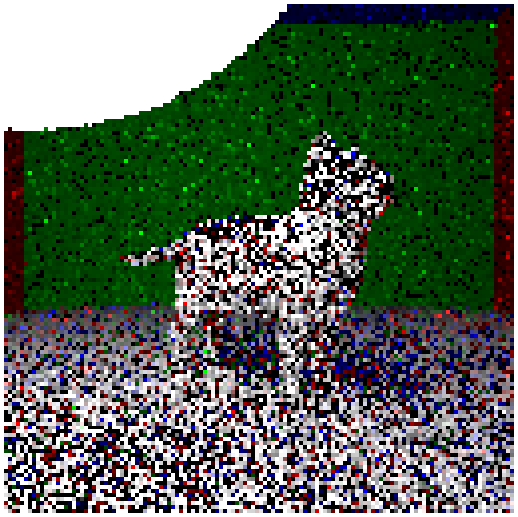

Figure 13:maillage naive de chat (1 rayon)//temps de calcul :6mn

Boîte englobante

Nous avons un temps de génération d'image assez long pour une faible résolution. Heureusement, il existe des méthodes pour accélérer la génération d'image. Pour commencer, nous pouvons calculer la boîte englobante du maillage. Nous créons la classe BoundingBox qui devra qui déterminera s'il rentre en intersection avec un rayon. S'il y a intersection, alors nous pouvons regarder s'il y a intersection au niveau du maillage

Figure 14:chat avec boite englobante (10 rayon)//temps de calcul :15mn

Dichotomie

Une autre façon d'accélérer le calcul est, lorsque le rayon est dans la bounding box, de procéder par dichotomie pour trouver le triangle d'intersection. Cela nous permet de gagner un facteur 20 par rappport à la bounding box.

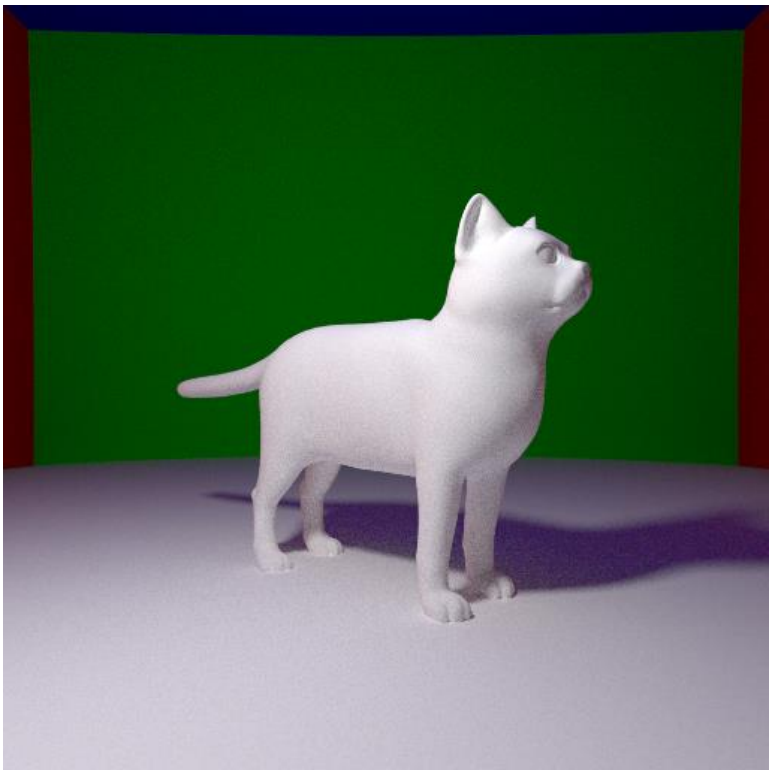

Figure 15:scene de chat en dichotomie (300 rayons) génerée par dichotomie_bb.cpp //temps de calcul : 1 h15mn

Lissage du maillage

Nous arrivons à afficher des fichiers OBJ assez rapidement sur des images de bonnes résolutions. Maintenant nous pouvons lisser le maillage afin d'enlever le côté "boule à facette" sur les objets affichés.

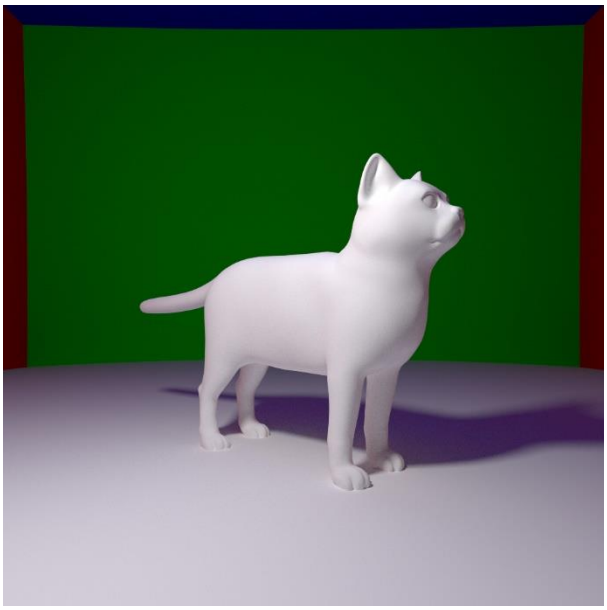

Figure 16:chat Lisse Dichotomie (300 rayon)//temps de calcul :50mn

Textures

Maintenant que nous avons des maillages lisses, nous pouvons appliqués les textures qui leur sont liées. Il suffit de rajouter la texture créée par l'artiste. Pour cela, une correspondance est faite entre les triangles et l'image du chat.

Figure 17:scene de chat (5OO rayons) en texture généré par texture.cpp//temps de calcul 56mn