



ÉCOLE CENTRALE LYON

RAPPORT
LIEN GITHUB

Informatique Graphique

Étudiants :
Victor LUDVIG

Encadrants :
Nicolas BONNEEL

22 mars 2024

Table des matières

I Intersection rayon-sphère	2
II Ombres et correction gamma	2
III Réflexions et réfractions	3
IV Loi de Fresnel	4
V Éclairage indirect et anti-aliasing	4
VI Maillage, BVH, shading normal	5
VII Textures UV et procédurales	6
VII Barycentre	7
IX Rotation	7
X Rendu final	8
XI Jolis bugs	9

I Intersection rayon-sphère

Tous les temps sont calculés pour des images 512x512.

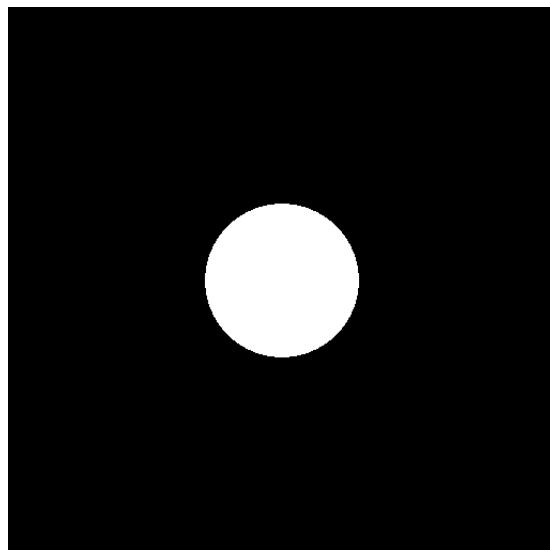


FIGURE 1 – Calcul de l'intersection avec la sphère
Temps de calcul : 0.075s

II Ombres et correction gamma

Les ombres sont ajoutées pour les surfaces Lambertiennes.

Le temps de calcul est plus long que dans le poly, car à ce stade je n'utilisais qu'une seule référence pour le calcul de t , mais la normale N et le point d'intersection P étaient calculés après la fonction *intersect*, à partir de t . J'ai gardé cette configuration jusqu'à l'affichage de maillages. Après j'ai été obligé d'utiliser des références dans la fonction *intersect*, car le calcul de la normale et du point d'intersection est différent selon l'objet intersecté (sphère ou maillage). Les temps de calcul sont donc diminués lors de l'affichage du chat.

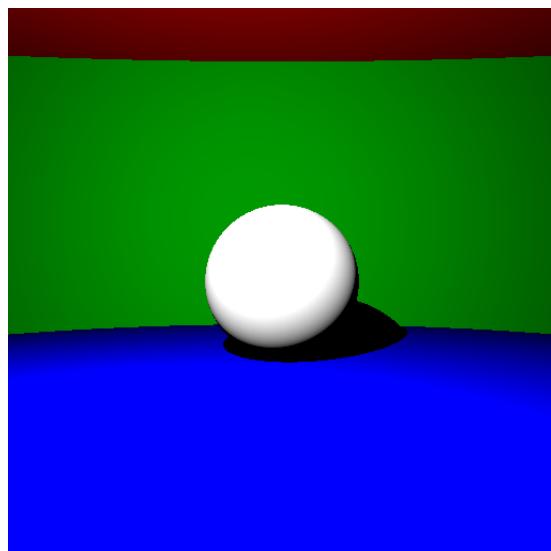
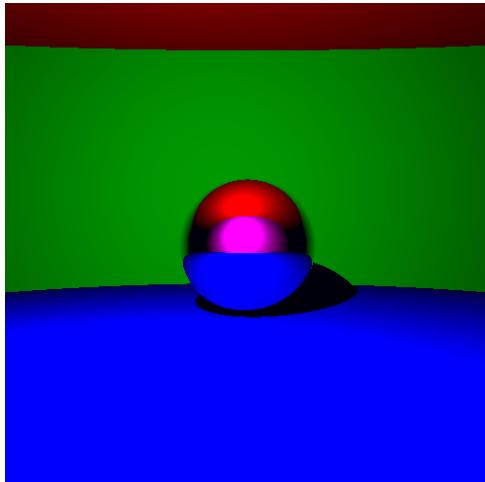
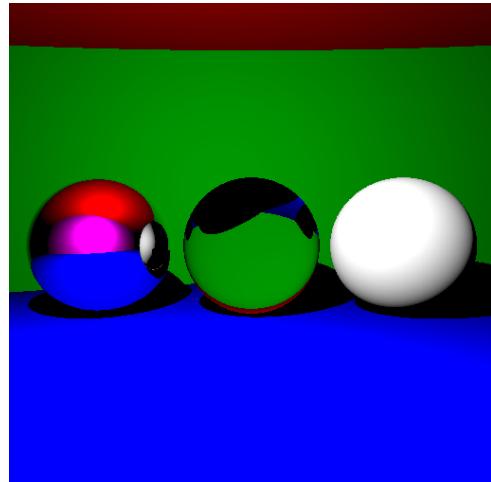


FIGURE 2 – Ajout des ombres et correction gamma
Temps de calcul : 0.344s

III Réflections et réfractions



(a) Ajout des réflexions.
Temps de calcul : 0.359s



(b) Ajout des réfractions.
Temps de calcul : 1.29172s

IV Loi de Fresnel

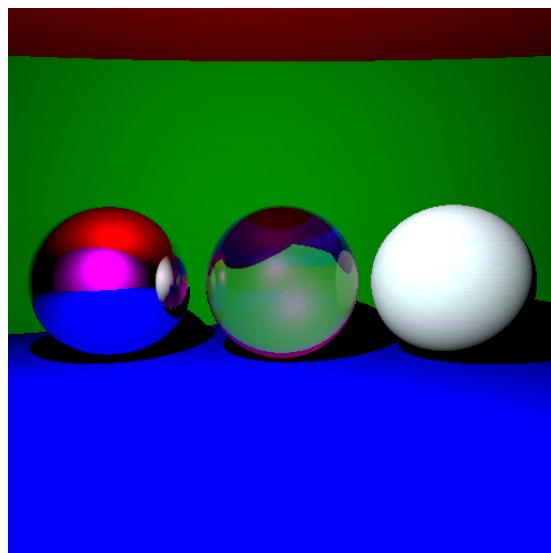


FIGURE 4 – Ajout de Fresnel.

Rayon par pixel : 1000

Temps de calcul : 2min36s

Parallélisation avec 8 threads.

V Éclairage indirect et anti-aliasing

Le code est assez lent, probablement car j'ai utilisé des attributs privés avec getters/setters, mais il semble que les getters renvoient font une copie de l'objet. C'est une bonne pratique dans d'autres langages mais peut-être pas en c++...

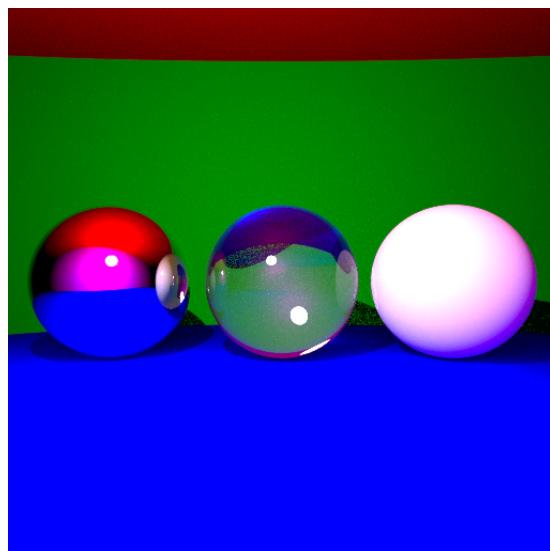
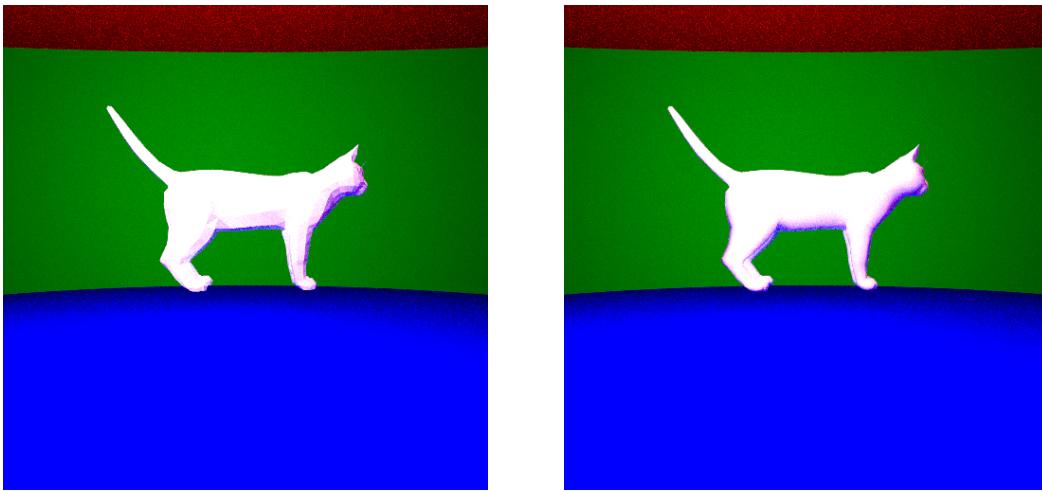


FIGURE 5 – Ajout de l'éclairage indirect et de l'antialisaing.

Rayons par pixel : 1000
Profondeur maximale : 5
Temps de calcul : 43min36s
Parallélisation avec 6 coeurs.
Rayon de la lumière : 8 pixels

VI Maillage, BVH, shading normal

La translation et le scaling ont été implémentés. Pour le rendu final, j'ai implémenté un scaling à partir du barycentre, pour qu'il soit indépendant de la position.

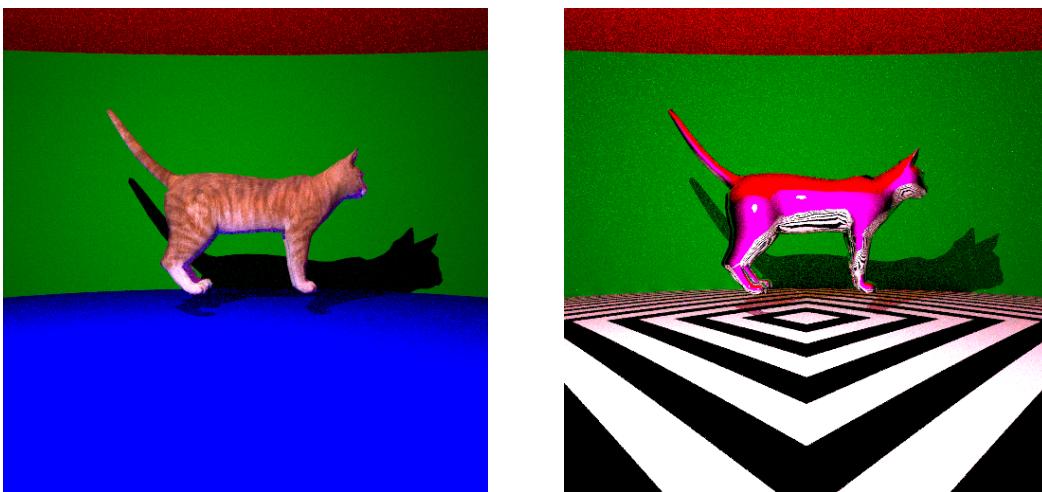


(a) Chat avec BVH

(b) Shading normal

FIGURE 6 – Le temps de calcul est divisé par environ 20 avec l'ajout de l'arbre BVH. On passe d'environ 20 minutes à 1 minute, mais je n'ai pas les temps exacts.
À ce stade il y avait un problème avec les ombres, que j'ai corrigé ensuite (j'avais oublié le terme de visibilité, l'image avec Fresnel a été générée avec la version corrigée.)

VII Textures UV et procédurales



(a) Texture avec coordonnée UV.

(b) Texture procédurale et chat miroir

FIGURE 7 – Rayons par pixel : 32
Profondeur maximale : 5
Temps de calcul : 1min22s
Parallélisation avec 8 threads.

Les ombres ne démarrent pas aux pieds du chat car les pieds ne touchent pas le sol.

VIII Barycentre

Pour le rendu final, je fais des rotations autour du barycentre, donc il faut que je calcule le barycentre du maillage. Une première approche naïve consiste à faire la moyenne des sommets, cependant certaines zones avec des détails ont une plus grande densité de sommets, donc cela fausse le résultat. Une seconde approche considère les barycentres de chaque triangle pondéré par l'aire du triangle.

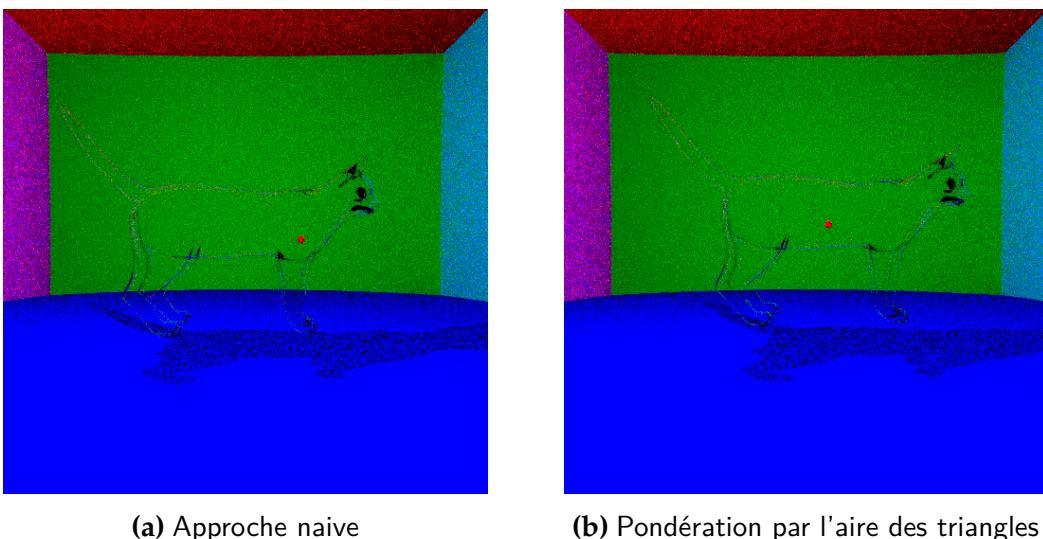


FIGURE 8 – Calcul du barycentre

Rayons par pixel : 5

Profondeur maximale : 5

Temps de calcul : 27s

Le chat est transparent.

IX Rotation

J'ai implémenté une rotation autour du barycentre, selon un axe de rotation et angle quelconques. La formule est donnée par cet article. Voir la fonction init_rotation_matrix de mon code pour l'implémentation.

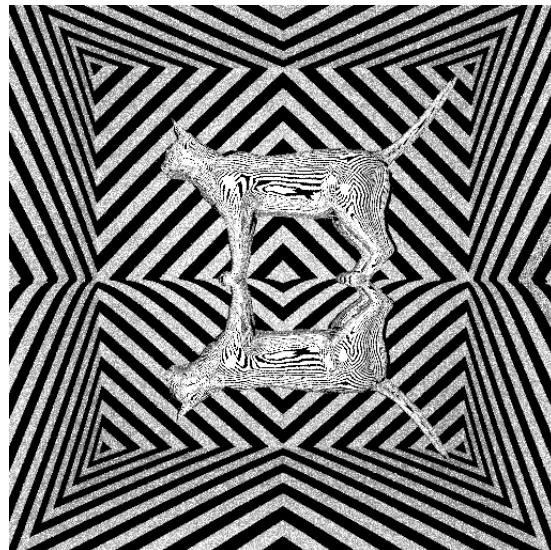


FIGURE 9 – Rotation de 180 degré selon l'axe vertical.

Rayons par pixel : 50

Profondeur maximale : 5

Temps de calcul : 7min07s

Le temps de calcul est environ 9 fois plus long lorsque le chat est tourné, pour une raison inconnue.

X Rendu final

J'ai changé la disposition des murs pour rendre le scène plus proche du chat, et pour qu'on voit plus les murs. Le sol est un miroir, tous les autres murs ont une teinte procédurale. Le chat tourne autour de son barycentre selon l'axe vertical. J'ai fait en sorte d'avoir un tour pour 10 secondes avec 30 fps, ce qui donne 300 images. Sur le readme du github la vidéo est à 38 fps pour pouvoir faire un tour complet tout en ne dépassant la limite de 10MB. Chaque image a 50 rayons. Le chat tourne de 1.2 degré à chaque instant. Le chat est positionné de tel sorte qu'on voit sa réflexion lors de la rotation. Plus d'une centaine d'images ont été générées afin d'obtenir les bons paramètres des différents objets, avant de générer les images de la vidéo. Dans la première vidéo, je n'avais pas fait tourner les normales, donc certains rayons repartent à l'intérieur du chat lorsqu'il est tourné, ce qui donne un effet artistique.

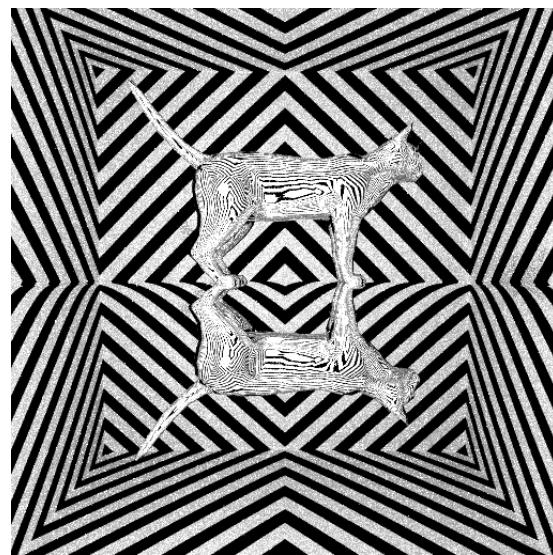


FIGURE 10 – Disposition initiale
Rayons par pixel : 50

Temps de calcul pour 300 images : environ 10 heures, avec 24 threads et intel core i7
13700KF.

XI Jolis bugs

Voici quelques jolis bugs artistiques.

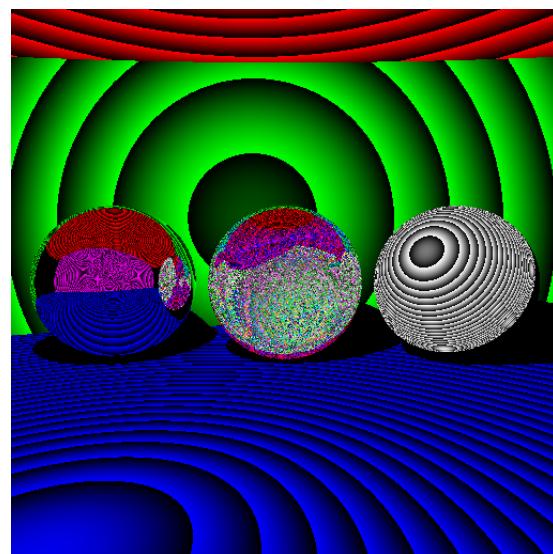


FIGURE 11 – Bug lié au modulo 255 des couleurs.

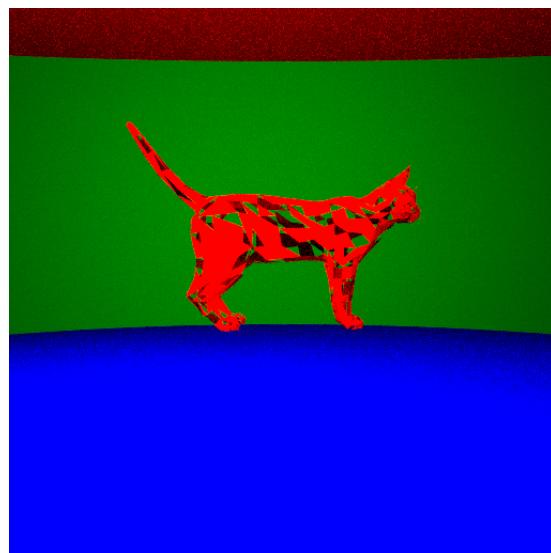


FIGURE 12 – Chat troué lié à un bug du BVH.

Dans le parcours des feuilles, la boucle for s'arrêtait un indice trop tôt, donc un triangle par feuille est systématiquement oublié. C'est pour cette raison que les trous sont répartis uniformément sur le chat.

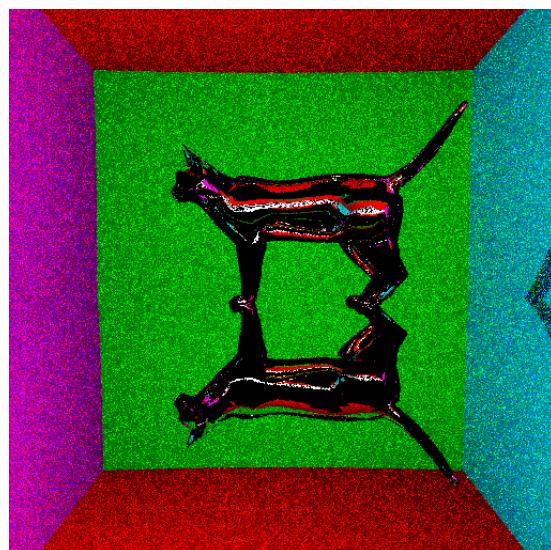


FIGURE 13 – Bug après rotation de 180 degrés.

Les normales du vecteur normals n'étaient pas incluses dans la rotation, donc cela pose problème lors du renvoie du rayon.