

ECOLE CENTRALE DE LYON
MOS 4.4 - Informatique Graphique

RAYTRACING

Paul FLAGEL

19 mars 2022

Résumé

Ce rapport synthétise les différents résultats obtenus lors de la construction d'un *pathtracer*. Cette technique de rendu consiste à simuler le parcours inverse de la lumière afin de générer des images de synthèse.

Note : Les rendus ont été effectués sur un Macbook Air M1/16Go RAM dans un premier temps. Puis pour les rendus plus longs j'ai utilisé les serveurs de l'école puisque le chauffage fonctionnait déjà très bien chez moi. Cela explique les différences des temps de rendus pour des scènes similaires. Si cela peut aider les futurs étudiants, j'ai réussi à faire fonctionner openMP sans manipulation supplémentaire grâce au compilateur g++-11.

Table des matières

1 Boules	2
1.1 Eclairage lambertien et ombres portées	2
1.2 Surfaces spéculaires et transparentes	2
1.3 Correction gamma	3
1.4 Eclairage indirect	3
1.5 Antialiasing	4
1.6 Ombres douces	4
1.7 Profondeur de champ	5
1.8 Coefficients de Fresnel (optionnel)	5
2 Maillages	5
2.1 Ajout de triangles	5
2.2 Mesh	6
2.3 Textures	7
2.4 Mouvements de caméra	7
3 Feedback sur le cours	8

1 Boules

1.1 Eclairage lambertien et ombres portées

Les images générées ont une taille de 512x512 pixels. Nous instancions une scène constituée de deux sphères et des murs qui sont en fait de très grosses sphères. Nous plaçons notre caméra en $C = (0, 0, 55)$ et ajoutons une lumière ponctuelle en $L = (-10, 20, 40)$ avec une intensité $I = 7 \times 10^7$ watts. Le champ de vision est de 60° .

Nous ajoutons ensuite un effet d'ombre portée en vérifiant pour chaque point s'il y a un obstacle entre ce point et la lumière. Si oui, un pixel noir est renvoyé.

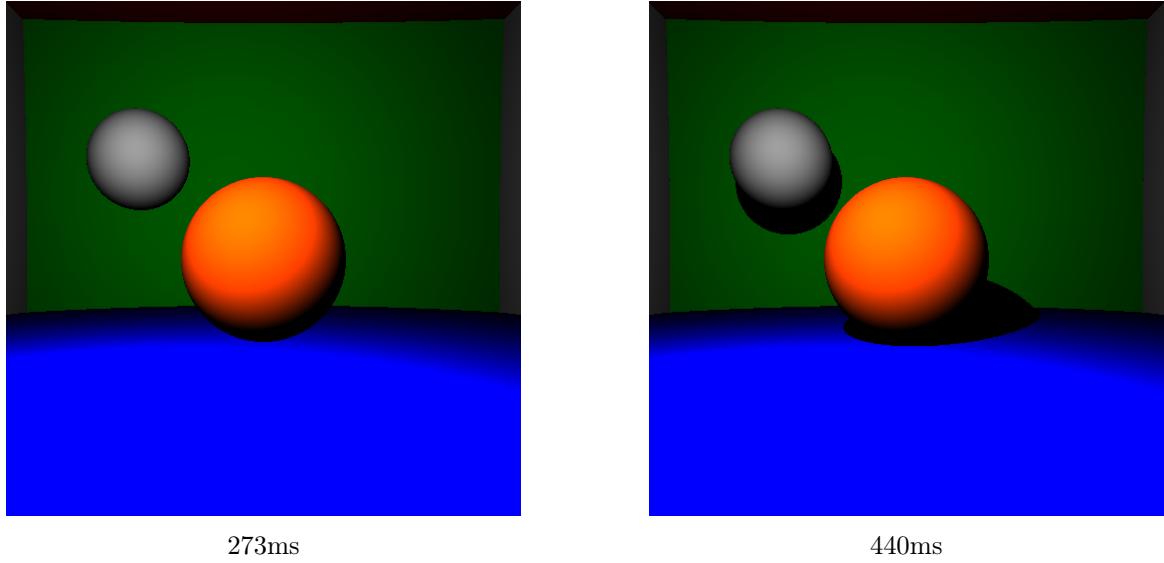


FIGURE 1 – Scène sans ombres à gauche et avec ombres portées à droite.

1.2 Surfaces spéculaires et transparentes

Jusqu'ici, les sphères éclairées étaient constituées d'un matériau diffus. Nous modifions la fonction `getColor` de notre scène afin de l'appeler récursivement lorsqu'on arrive sur une surface possédant la propriété miroir.

Nous implémentons également l'effet de transparence en ajoutant les lois de Snell-Descartes. Nous prenons une boule avec un indice de réfraction $n = 2$. On constate bien que l'image est retournée.

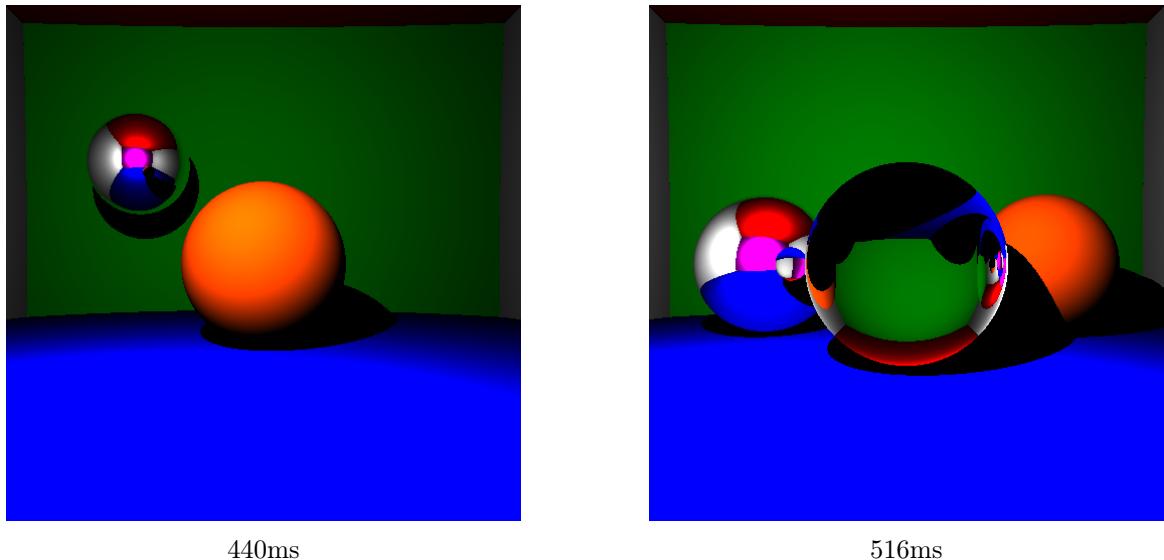


FIGURE 2 – Boule miroir à gauche et transparente à droite.

1.3 Correction gamma

L'intensité lumineuse affichée ne varie pas linéairement. Les écrans affichent les pixels avec un facteur $\gamma \approx 2.2$ tel que la luminosité est de la forme $L \propto I^\gamma$. Pour compenser cela, on applique une puissance $1/\gamma$ à la luminosité, ce qui a pour effet de diminuer les contrastes, et de mieux éclairer les zones sombres. Pour compenser la diminution de luminosité, on augmente l'intensité lumineuse $I = 5 \times 10^{10}$ watts.

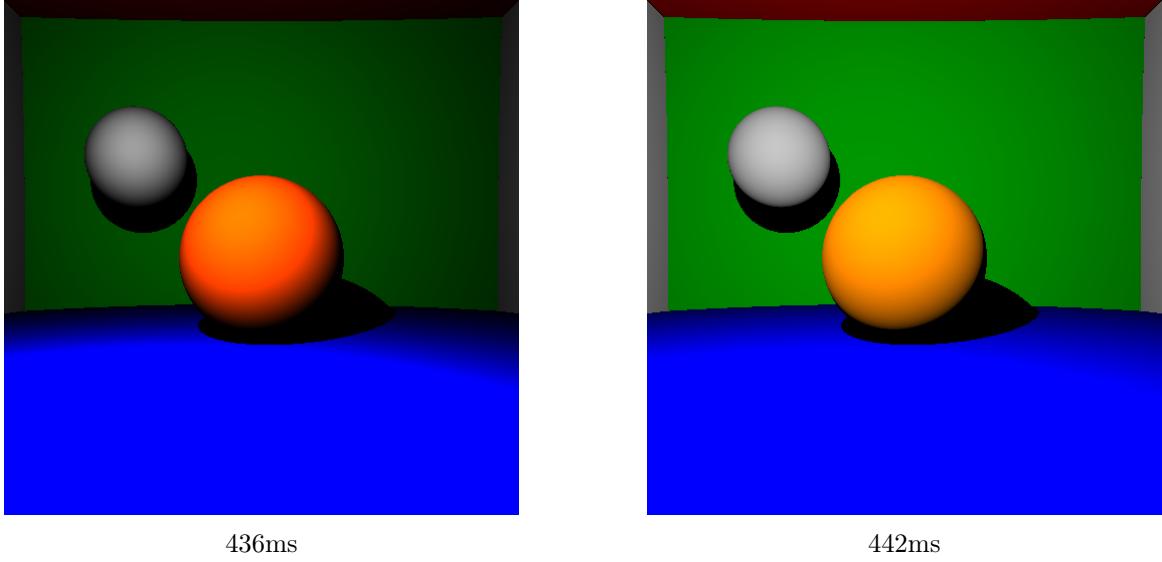


FIGURE 3 – **A gauche** : sans correction gamma. **A droite** : avec correction gamma.

1.4 Eclairage indirect

Nous ajoutons l'effet de l'éclairage indirect correspondant à la lumière réfléchie par l'environnement. L'idée est de générer plusieurs rayons qui ajouteront la contribution indirecte issue de directions aléatoires depuis le point P . La contribution indirecte correspondant à une intégration de l'équation du rendu de type Monte-Carlo, nous obtenons des images de moins en moins bruitées à mesure que le nombre de rayons augmente (l'erreur diminue en \sqrt{N}). On constate que pour 64 fois plus de rayons, le temps est multiplié par environ 60 ce qui est cohérent.

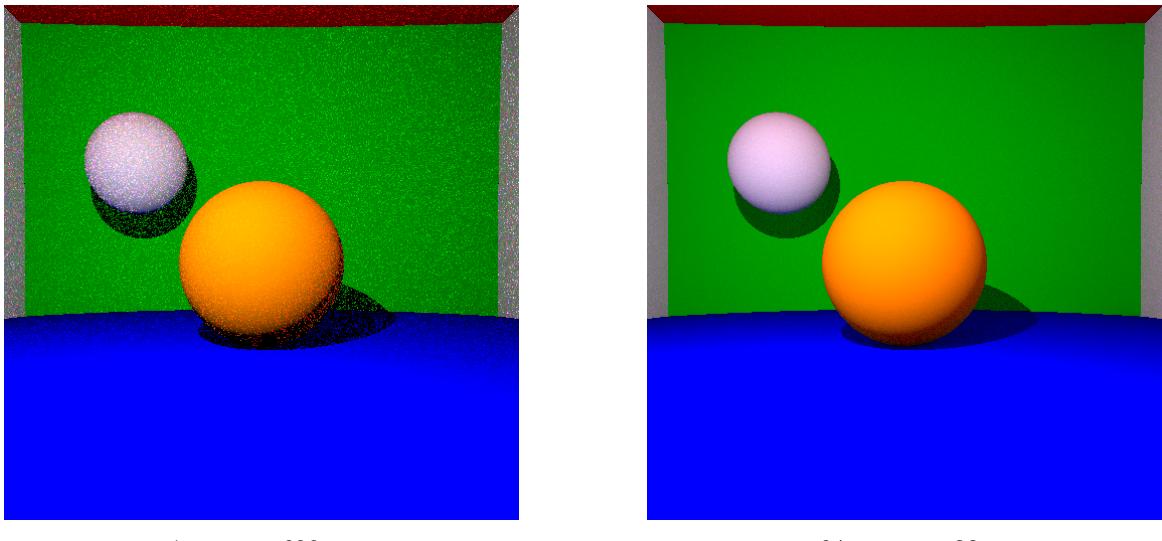
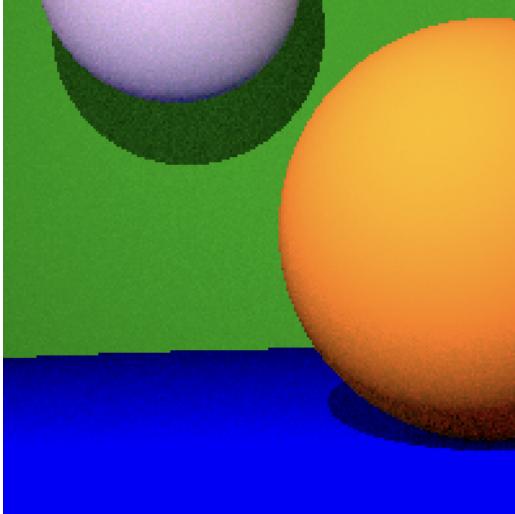


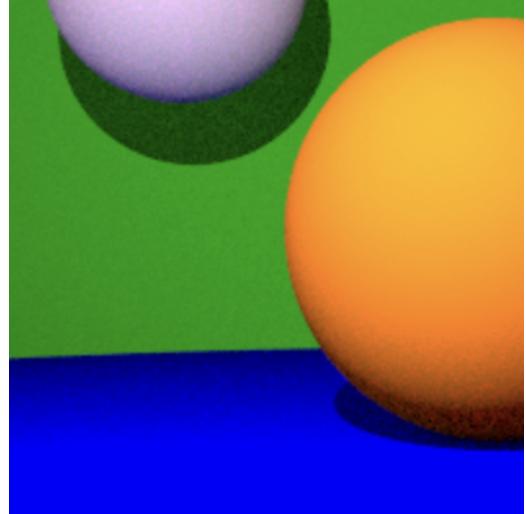
FIGURE 4 – Eclairage indirect : influence du nombre de rayons

1.5 Antialiasing

Depuis le début, les courbes ont un aspect crénelé car les rayons intersectent au milieu des pixels. Ainsi, si un bout de la courbe est dans le pixel mais ne passe pas par le centre, le pixel n'aura pas la couleur de la courbe mais celle de la surface du centre du pixel. Pour remédier à cela, on décale légèrement la direction dans laquelle on envoie chaque rayon suivant une loi normale ($\mu = 0, \sigma = 0,3$). Cela permet de moyennner les couleurs du pixel et de faire disparaître le crénelage.



33s



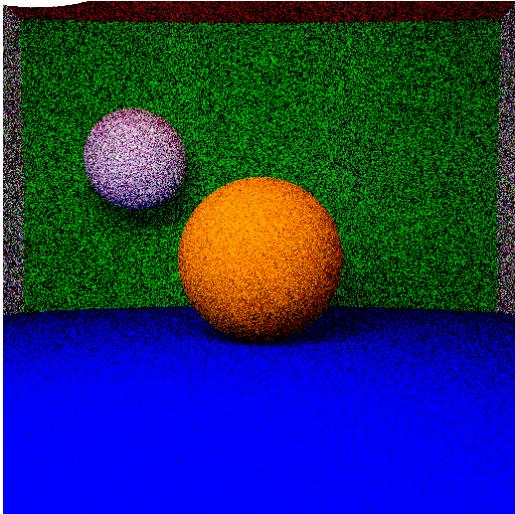
28s

FIGURE 5 – Scène sans antialiasing (à gauche) et avec antialiasing (à droite).

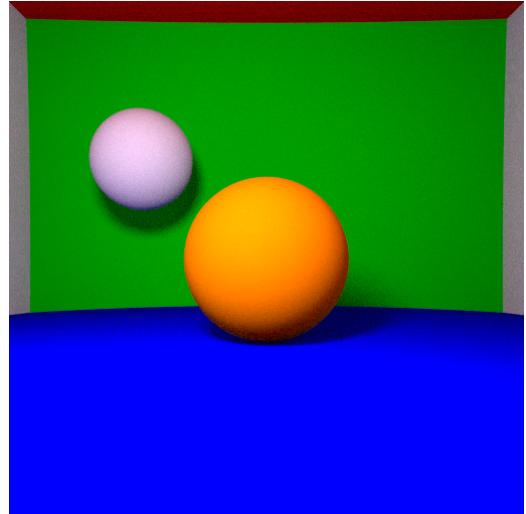
1.6 Ombres douces

Les ombres portées ont une délimitation très franche. Nous pouvons les adoucir en utilisant une source de lumière sphérique et non plus ponctuelle. Nous avons d'abord implémenté une approche naïve où l'on attend de toucher la sphère de lumière par contribution indirecte, mais on aboutit à des résultats très bruités et dépendants de la taille de la sphère de lumière.

Il est plus efficace d'intégrer sur la demi-sphère de lumière orientée vers la surface éclairée en la discrétilisant en N points tirés aléatoirement. On réalise le calcul d'ombre portée pour chaque point et on aboutit donc à des ombres douces.



15s



38s

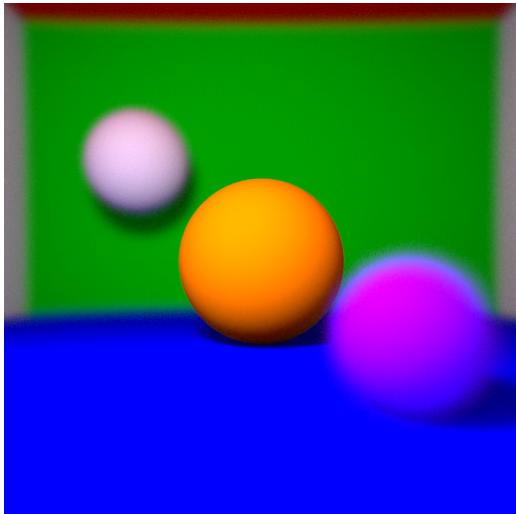
FIGURE 6 – Approche naïve (sphère lumineuse $R = 10$) à gauche et approche intégrale à droite.

1.7 Profondeur de champ

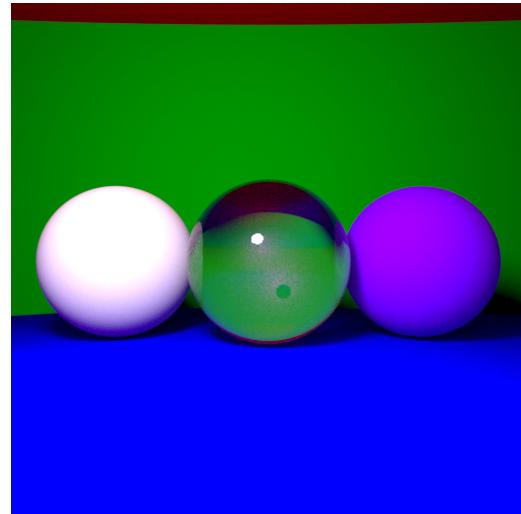
Nous ajoutons enfin un effet de profondeur de champ qui modélise une caméra au capteur non-ponctuel. À chaque rayon, le point C de la caméra est légèrement déplacé autour de la position initiale suivant une loi normale. On a défini la distance de netteté $ddof = 55$ ce qui a pour effet de rendre les objets flous lorsqu'ils sont à une distance inférieure ou supérieure à cette valeur (Figure 7 gauche).

1.8 Coefficients de Fresnel (optionnel)

Pour terminer, on ajoute un effet "miroir" aux sphères transparentes afin de les rendre plus réalistes. Cet effet repose sur les coefficients de transmission et de réflexion de la lumière déterminés par les équations de Fresnel dont on réalise ici une approximation.



38s



58s

FIGURE 7 – Ajout d'un effet de profondeur de champ **à gauche**. Sphère transparente avec coefficients de Fresnel (256 rayons/pixel) **à droite**.

2 Maillages

2.1 Ajout de triangles

Nous avons dans un second temps ajouté au code un parser permettant de lire des fichiers `.obj` encodant des maillages (*meshes*). Il a fallu implémenter les méthodes d'intersection fonctionnant avec les triangles. Pour vérifier que tout fonctionne, un fichier contenant un triangle a été créé. Nous désactivons l'éclairage indirect, l'antialiasing, les ombres douces et la profondeur de champ pour diminuer le temps de rendu.

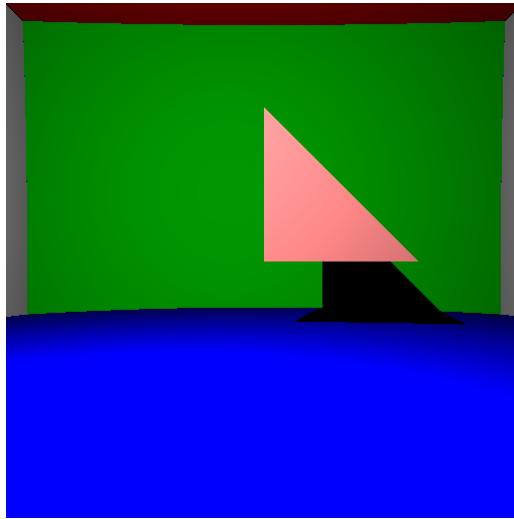


FIGURE 8 – Triangle - 173ms

2.2 Mesh

Nous chargeons maintenant des maillages réels, en désactivant tous les effets pour diminuer le temps de rendu. Dans un premier temps, nous testons pour chaque pixel s'il y a une intersection avec l'intégralité des triangles et renvoyons le triangle le plus proche. Cette méthode fonctionne mais le rendu est extrêmement long notamment lorsque les maillages comportent plusieurs dizaines de milliers de triangles. Nous utilisons le mesh *Australian Cattle Dog V3* et obtenons un **temps de rendu d'environ 900s** avec cette méthode naïve.

Puis nous calculons la boîte englobante autour de l'objet. Ainsi, avant de tester l'intersection avec chaque triangle, on teste d'abord si on intersecte la boîte englobante. Avec cette méthode, le **temps de rendu tombe à 282s**.

Nous implémentons l'algorithme BVH (*Bounding Volume Hierarchy*) permettant de découper le maillage en sous-maillages et de calculer la boîte englobante de chacun des sous-maillages. L'idée est de construire un arbre contenant les sous-maillages et de le parcourir à chaque test d'intersection. Le résultat est extrêmement satisfaisant : **2 secondes de rendu** (tous effets désactivés). Ce qui constitue un gain de temps d'un facteur 450 par rapport à la méthode naïve.

Enfin, nous implémentons l'interpolation des normales permettant de lisser la surface des maillages ainsi que des méthodes `scale` et `rotate` pour orienter le maillage dans la scène.

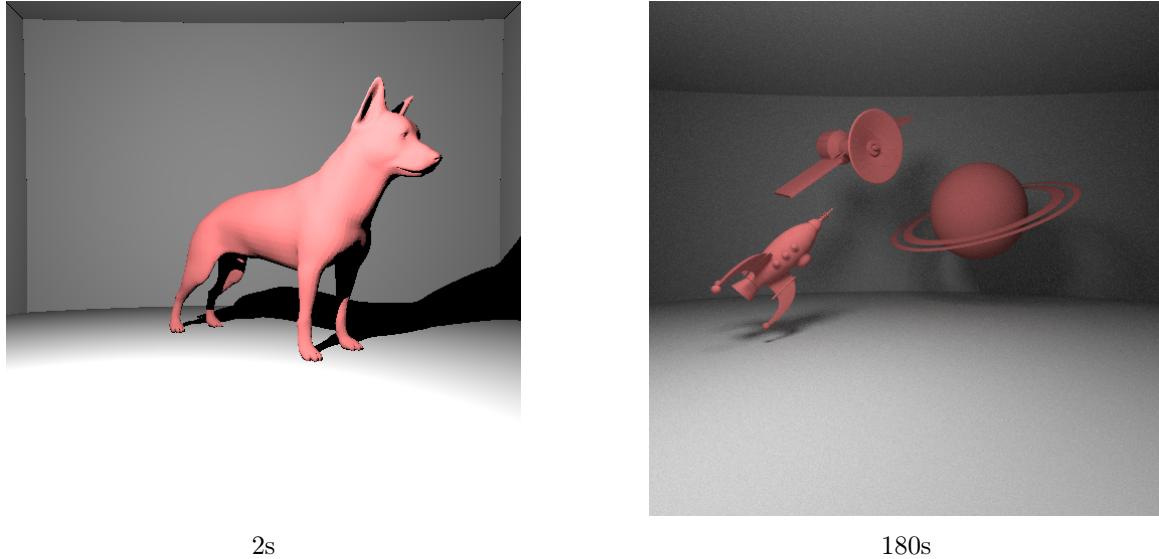


FIGURE 9 – Rendus avec interpolation des normales et BVH. Chien avec effets désactivés à gauche. Scène avec éclairage indirect, antialiasing, ombres douces (64 rayons) et $\text{fov}=80^\circ$ à droite.

2.3 Textures

Nous ajoutons enfin les textures aux objets de notre scène. Pour cela, nous utilisons les coordonnées UV permettant d'effectuer le mapping avec un fichier image (.jpg ou .png) contenant les textures. Dans mon implémentation, il faut faire attention à bien charger les textures dans le bon ordre pour les fichiers qui en contiennent plusieurs.

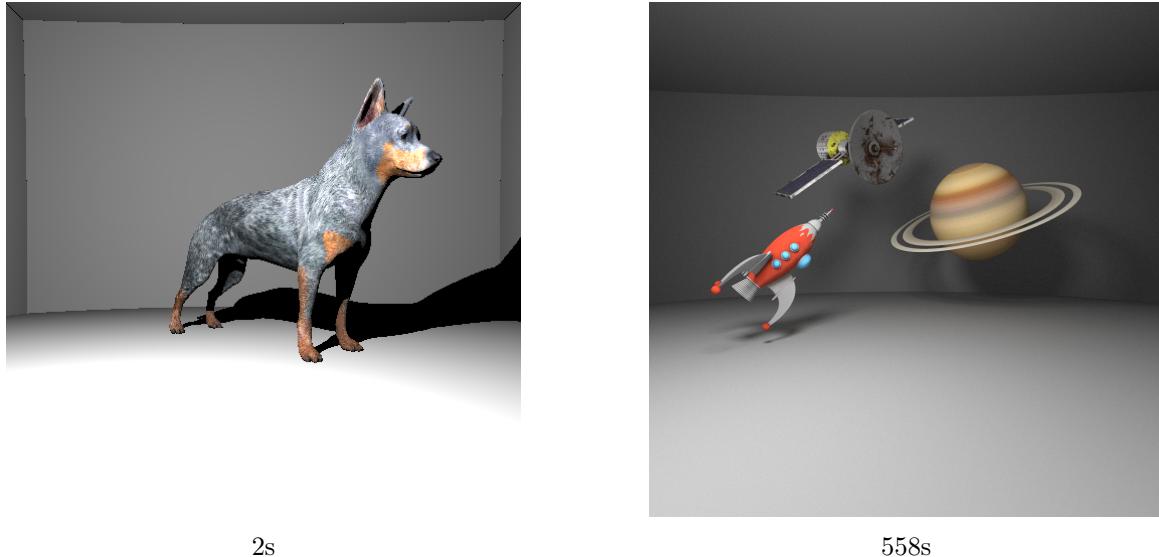


FIGURE 10 – Ajout des textures. Chien avec effets désactivés à gauche. Scène 1024x1024 avec éclairage indirect, antialiasing, ombres douces (128 rayons) et $\text{fov}=80^\circ$ à droite.

2.4 Mouvements de caméra

Enfin, nous avons implémenté des mouvements de caméra afin de générer plusieurs images et créer un gif ([cf. repo GitHub](#)). Pour cela il suffit de déplacer la caméra et d'appliquer une matrice de rotation sur le vecteur qui pointe dans la direction de la scène. Le résultat est plutôt satisfaisant.

3 Feedback sur le cours

J'ai trouvé que le passage de la théorie au code est parfois un peu musclé, notamment quand on commence à parler de maths. Il faut digérer des concepts compliqués pour les implémenter de façon non-triviale (je pense particulièrement à l'effet de la lumière indirecte). Peut-être que passer un peu de temps à expliquer le pseudo-code (comme vous l'avez déjà fait parfois) sur les concepts compliqués pourrait aider.

J'ai beaucoup apprécié le cours que j'ai trouvé très bien construit et bien enseigné. Mixer cours et pratique est une super idée. La difficulté augmente graduellement et permet à chaque séance d'améliorer le code, ce qui est très satisfaisant. De plus, la physique est plutôt accessible et la difficulté repose surtout sur l'implémentation, ce qui est cohérent pour un module d'informatique. Encore loin d'être expert en C++, le projet pousse à se plonger dedans et à apprendre beaucoup. Avec un peu d'efforts on trouve vite les ressources nécessaires. Cependant, cela demande beaucoup de temps quand on n'a aucune expérience et donner des resources avant le premier cours serait très utile.