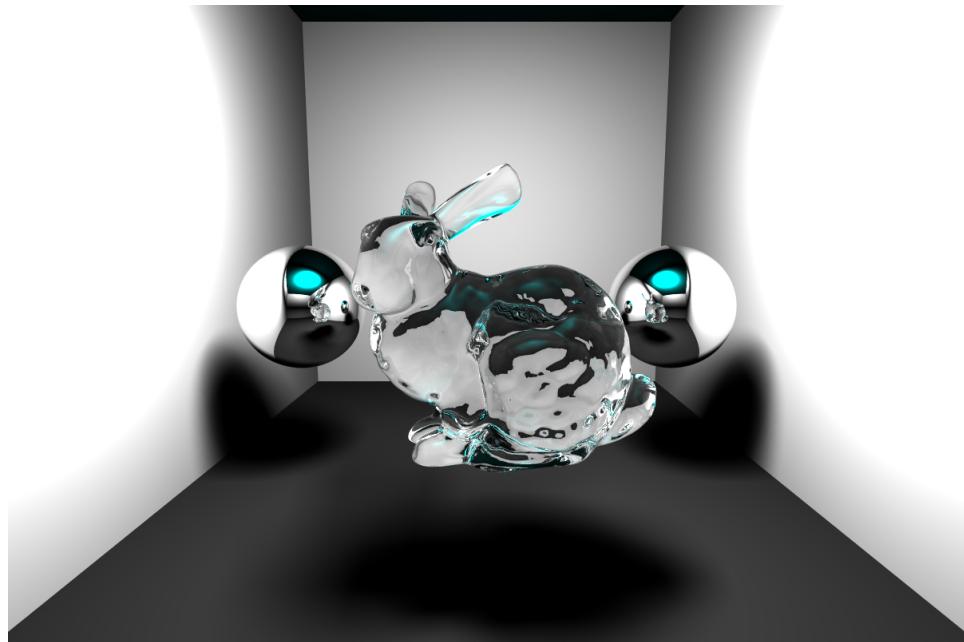




Rapport de projet

Introduction à la synthèse d'image réaliste

Antoine COUTY



Encadrant : Maxime MARIA

9 mai 2022

1 Instruction

Durant ce projet pour l'UE Introduction à la synthèse d'image réaliste, l'objectif premier était de réaliser les sept TPs permettant d'implémenter diverses fonctionnalités d'un ray tracer. Une fois les TPs terminés il a fallu penser à des pistes d'amélioration. Concernant la plus grande amélioration, pour ma part ce fut l'implémentation d'un path tracer en suivant les directives étudiées en TDs.

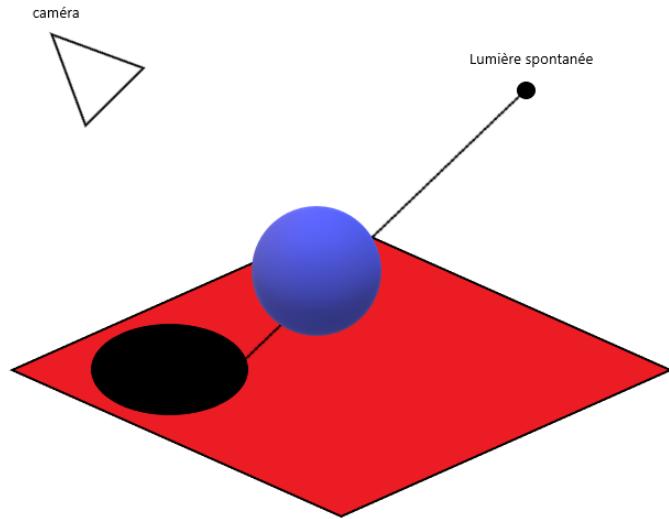
Avant de rentrer dans les explications des différentes implémentations, le fonctionnement de la scène est régie par une succession de DEFINE permettant d'initialiser notre scène via plusieurs configurations. De cette façon, les explications qui suivront dans le rapport posséderons tous un DEFINE renvoyant directement à une scène.

2 Explications des TPs

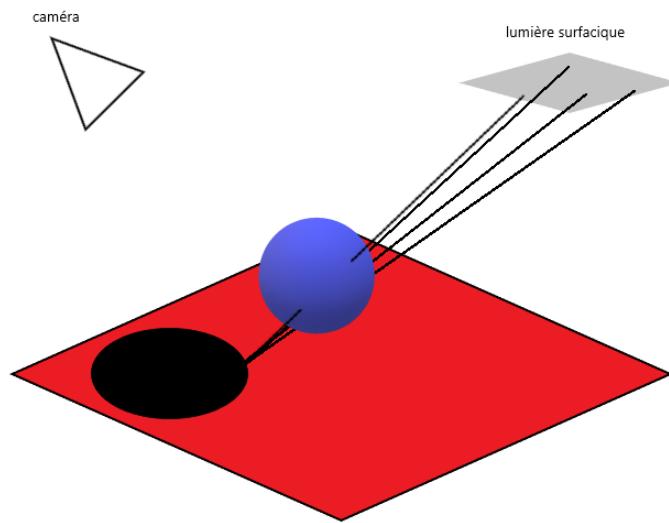
Pour ce qui est de cette partie je ne vais pas présenter les TPs un par un mais plutôt présenter les lignes directives ainsi que les implémentations un peu plus complexe.

2.1 Lumières

Pour ce qui est des lumières deux types se distinguent, les lumières spontanées et surfaciques. Le but en ayant des sources lumineuses dans notre scène est de pouvoir observé l'ombre d'un objet. Pour ce faire, quand un rayon est lancé depuis la caméra et qu'il intersecte un objet de la scène, on va envoyer un rayon (appelé rayon d'ombrage) depuis ce point vers les sources lumineuses. Si un autre objet de la scène intersecte ce rayon d'ombrage alors une ombre doit apparaître à ce point. On effectue ce processus pour chaque source lumineuse on puis additionne la contribution de chacune pour connaître l'éclairage sur le pixel cible.



(a) Rayon d'ombre pour lumière spontanée



(b) Rayon d'ombre pour lumière surfacique

FIGURE 1

Pour ce qui est des lumières spontanées elles ne sont physiquement pas réaliste mais permettent une vitesse de calcul supérieur. Aucune lumière est représenté par un seul point (même la laser est un faisceau) et en conséquence les ombres apportés par ce type de lumière sont abruptes ne possédant aucune

atténuation. Leur comportement est décrit ci-dessus où les sources spontanées n'envoient qu'un seul rayon d'ombrage (cf figure 12a).

Les lumières surfaciques vont quand à elle simulées toutes les lumières présentent dans notre entourage (soleil, spot, etc.) et permettant d'apporter des ombres douces à notre scène (dégradé dans les ombres). Pour cela, au lieu de lancer un seul rayon d'ombre par lumière, on va envoyer des rayons depuis notre intersections vers des points choisis aléatoirement sur notre lumière (nombre de rayon prédéfini). Une fois les la valeur du pixel calculée pour chaque rayon d'ombrage, on les moyennes pour obtenir la valeur final du pixel (cf figure 12b). Pour ce qui des TP, seul une lumière en forme de quad est implémentée. Pour ce qui est du code pour modifier le nombre de rayon lancé sur une lumière, il faut aller dans le .h des intégrateur DirectLighting, Whitted ou Path selon celui utilisé et modifié la variable privée `_nbLightSamples`.

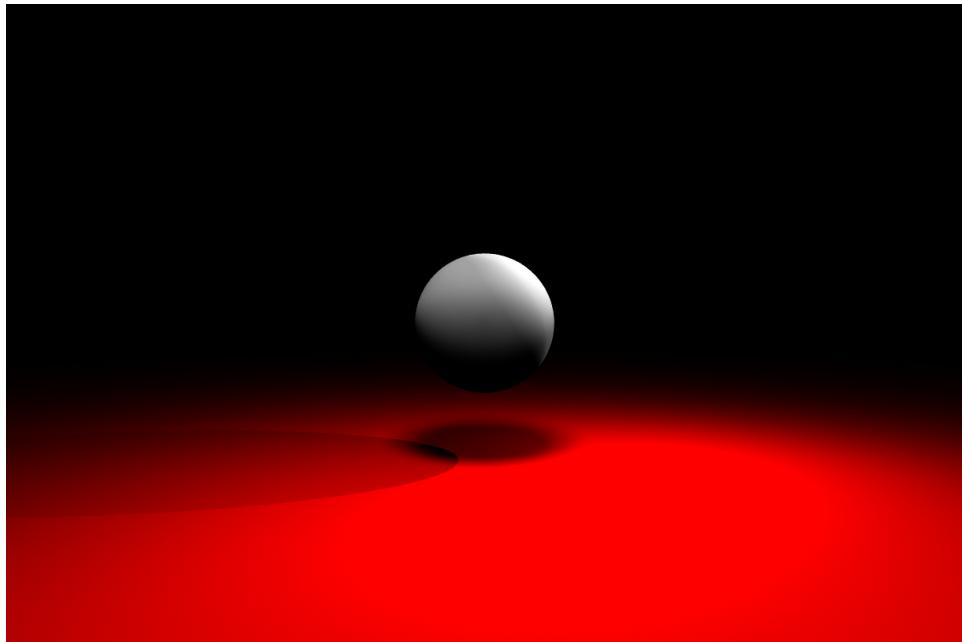


FIGURE 2 – Lumière Quad au dessus et Point sur la droite

2.2 Matériaux

Les matériaux réagissent tous différemment quand un rayon de lumière vient les intersecer. Des qu'un rayon incident (w_i) va intersecer une surface, celle-ci ne va pas complètement l'absorber, elle va le renvoyer dans toutes les directions faiblement si le matériau est diffus (dans un lobe diffus) ou bien le renvoyer autour une direction qui correspond au rayon réfléchi par rapport à la normale (w_r) correspondant à un lobe spéculaire. Ce fonctionnement est représenté par le schéma de la figure 3.

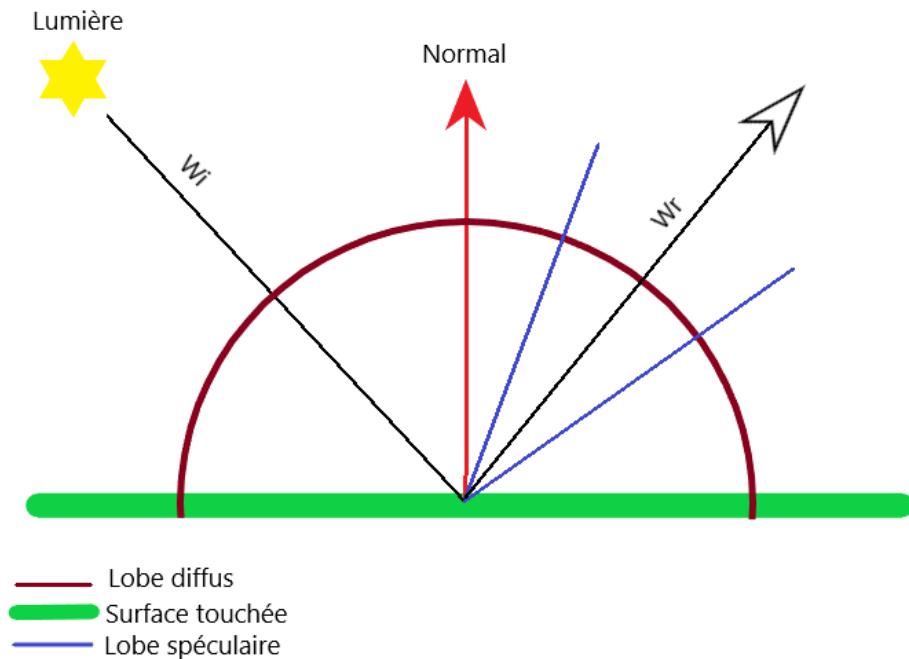


FIGURE 3 – Comportement d'un rebond de la lumière

2.2.1 Matériaux standards

Le point important suivants qui est abordées dans les TPs correspond aux matériaux mais plus précisément à la BRDF élément important dans l'équation de la lumière. La figure 4 représente les différents matériaux "standards" implémentés à savoir Lambert en magenta, Oren-Nayar en rouge, Cook-Torrence en dorée et Phong (gauche) et Blinn-Phong (droite) en cyan. Je ne vais pas m'attarder sur Phong, Lambert et Blinn-Phong dans ce rapport étant donné que leur BRDF sont relativement simple à calculer.

Pour ce qui concerne Oren-Nayar, nous avons à faire à un matériau complètement diffus et rugueux (manipulé par un paramètre sigma). Pour calculer sa BRDF, il a fallu passer en coordonnées cylindropolaires afin de trouver les composantes Phi et Theta pour les rayons d'observation et incident avec la normal du point touché par le rayon comme repère. Il a ensuite suffit d'appliquer la formule donner dans le TP avec `cos(PhiWI-PhoWO)=glm::dot(PhiWI,PhoWO)` avec les deux valeurs uniformisés au préalable.

L'autre matériau qui était demandé permettant d'obtenir un aspect métallique utilise la BRDF d'Oren-Nayar comme composante diffuse et celle de Cook-Torrence comme composante spéculaire. La composante de Cook-Torrence à la particularité d'être un matériau à micro-facette ce qui se traduit

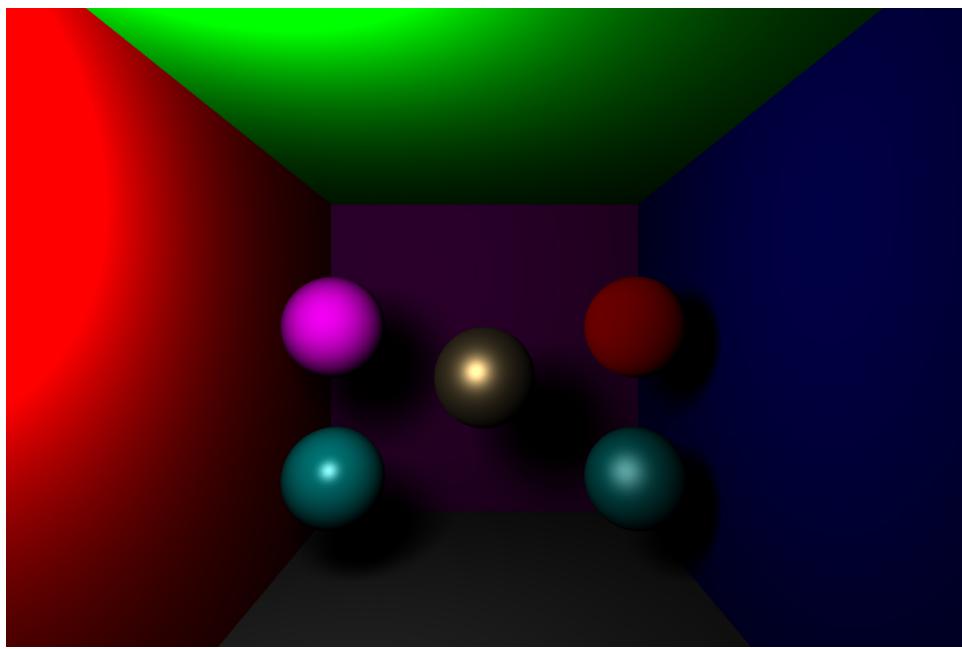


FIGURE 4 – Les différents matériaux

par grand facteur, D représentant la distribution des normales et créant ainsi notre "height-map", G représentant l'occlusion ambiente apportée permettant t'obscurcir les zones cachées par les miroirs et F étant un facteur de Fresnel calculant l'importance de la réflexion. De plus un facteur de metalness entre 0 et 1 est ajouté en plus permettant de gérer la proportion entre la valeur diffuse et spéculaire de l'objet. Plus celle-ci est élevée, plus le matériau sera foncé et aura de grande propriété spéculaire.

2.2.2 Matériaux avec réflexion/réfraction

Les deux matériaux suivants miroir et transparent ne sont utilisables que par le biais des intégrateurs WHITTED et PATH que l'on peut visualiser sur la figure 5.

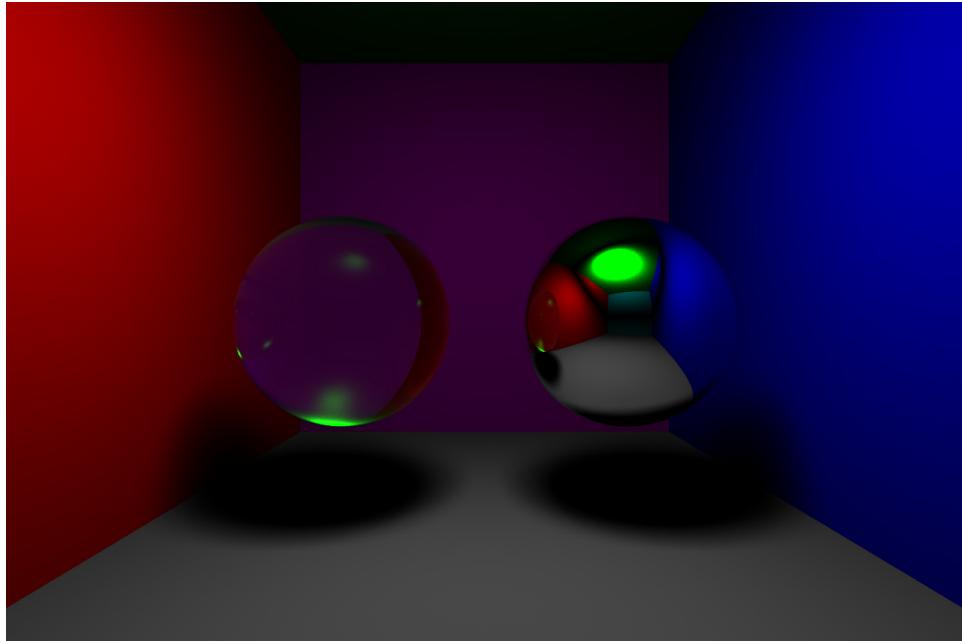


FIGURE 5 – Matériaux transparent et miroir

Le miroir est un matériau particulier étant donné qu'il est intégralement spéculaire. Autrement dit des qu'un rayon va intersecter un objet miroir, celui-ci va renvoyer un rayon réfléchi du rayon d'observation par rapport à la normale là ou les autres vont également diffuser de la lumière en accord avec leur matériau.

Pour ce qui est des matériau transparent, leur particularité est d'en plus de réfléchir de la lumière, il l'a réfracte passant ainsi à l'intérieur du matériau. Pour cela, nous utilisons les facteurs de Fresnel. Au niveau du code, j'ai implémenté deux méthodes différentes permettant de retourner les facteurs de Fresnel. La première implémentation permet de calculer ces facteurs de façon exact en suivant ces formules. Pour la deuxième, tout comme Oren-Nayar, il s'agit d'une approximation mais ce coup si celle de Snell-decartes utilisé couramment dans les moteur de rendu en temps réels. On peut observer la différence entre les deux implémentations avec la figure 6. Il est possible d'activer ou non l'approximation en passant un booléen au moment de la création d'un TransparentMatérial.

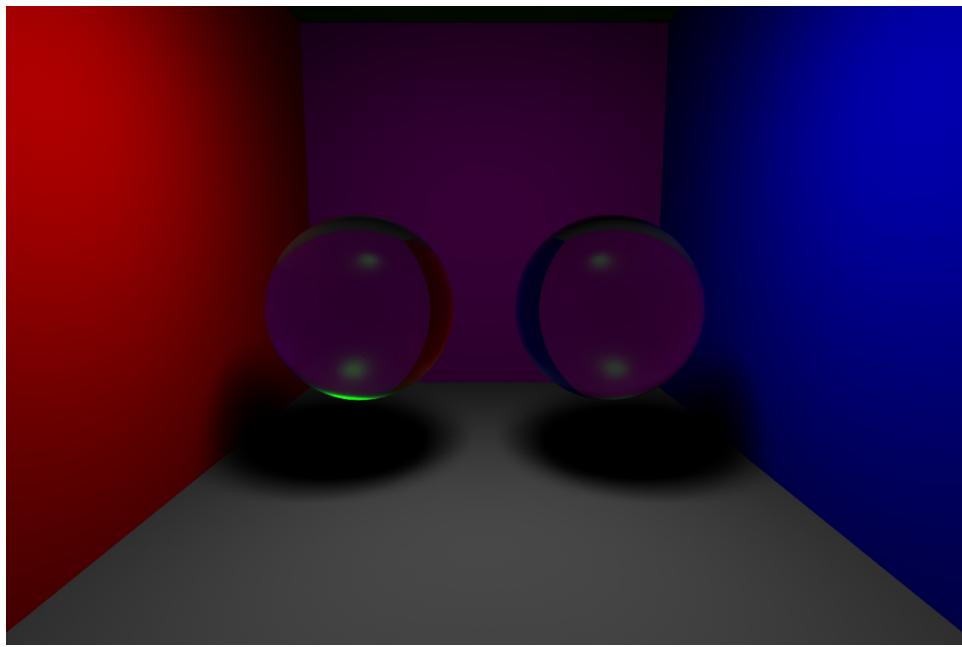


FIGURE 6 – Fresnel à gauche, Snell-Descartes à droite

Afin de rendre possible ces résultats, il a fallut passer l'intégrateur en récursif afin qu'il soit capable de gérer les rebonds de rayons surtout qu'un miroir peut rebondir dans un autre miroir et ainsi de suite.

2.3 Maillage

L'étape suivante était de pouvoir récupérer et calculer les intersections avec des objets récupérer dans des fichiers .obj, pour cela, les enjeux étaient multiples. Pour commencer, un maillage étant composé de triangles, il a fallut calculer l'intersection avec une nouvelle géométrie, le triangle. Pour se faire, j'ai repris la méthode Moller-trumbore.

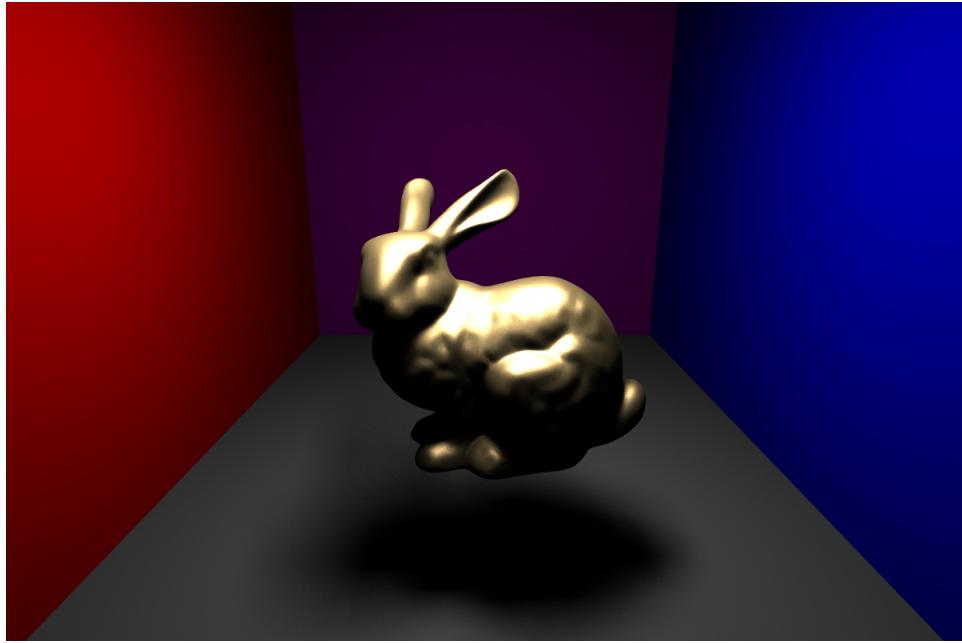


FIGURE 7 – Affichage de l'objet "bunny"

Une fois ces étapes implémentés, il est possible d'afficher n'importe quel objet. Le problème étant que plus l'objet est gros, plus le nombre de triangles est important, plus le temps de calcul sera long. Pour palier à cela, nous utilisons les structures accélératrices. Pour commencer, nous utilisons un système de boite englobante AABB.

Le AABB aura pour principe de récupérer les points représentants les valeurs maximales et minimales d'une géométrie permettant de créer ainsi un cube autour de notre géométrie. Une fois ces valeurs sauvegardées, on va vérifier avant de calculer une intersections probables avec un triangle si on intersecte notre boite amglobante.

Pour calculer l'intersection, tout d'abord il a fallu résoudre l'équation suivante en intégrant l'équation du rayon :

$$\text{Origin} + \text{Direction} * t > \text{Min} \quad \text{Origin} + \text{Direction} * t < \text{Max}$$

Pour cela, nous calculons les coordonnées maximal et minimal valant $\text{Origin} - (\text{min ou max}) / \text{direction} = t$. Puis nous résonnons par l'absurde si une coordonnée du vecteur minimal est supérieur à une valeur maximal alors notre rayon passe à coté de notre aabb.

Grace à cette méthode le temps de calcul est amélioré mais reste encore trop important étant donné que des qu'un rayon touche le aabb tous les triangles du maillage sont testés. Pour cela, nous utilisons un BVH qui va découper notre géométrie en plusieurs AABB de manière à segmenter des zones de façon a ne pas avoir à tous tester mais seulement les zones intéressantes (cf schéma 8).

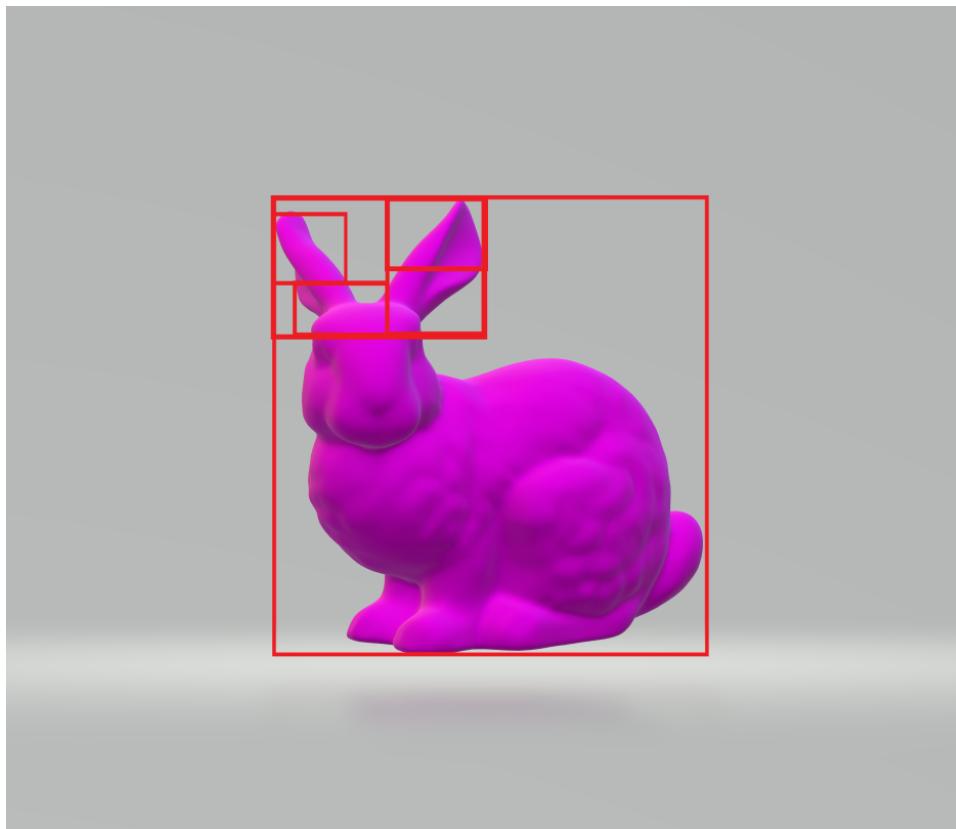


FIGURE 8 – Segmentation du BVH pour la partie gauche de bunny

Pour cela, nous ajoutons un aabb à chaque triangle, etant donné que la méthode extend permet d'agrandir un aabb en ajoutant un autre aabb. Ensuite nous ajoutons tous les triangles dans le aabb racine du BVH qui va regrouper toute la géométrie. L'étape suivante va être de trier tous les triangles selon l'axe dominant (la plus grande diagonal) puis de couper en deux le tableau en choisissant arbitrairement le milieu et les passer à un noeud droit et gauche. Puis on répète la manœuvre successivement jusqu'à obtenir une profondeur de l'arbre fixé ou un nombre de triangle maximal par feuille.

Cette méthode va nous permettre d'augmenter grandement les calculs et nous permettre d'afficher des scènes tels que la figure ?? sans laisser tourner le programme pendant des heures.



FIGURE 9 – Affichage de la scène conférence

Afin d'optimiser cet algorithme, au lieu de couper au milieu de notre aabb sans se soucier des vides ou autres qu'il peut apparaître dans notre géométrie, j'utilise une note SAH sur mon BVH. Pour cela, à chaque fois qu'il faut couper, on test toutes les divisons de boite en deux en lui attribuant une note. Cette note est calculer de la façon suivante :

```
SAH value = (SA_left*nleft+SA_right*nright) / SA_parent
```

Avec SA_*** correspondant au périmètre de la boite en question et n**** au nombre d'élément dans la boite. Cette technique permet en effet de diminuer le temps de calcul d'environ 20% mais à contrario augmente le temps de création du BVH de façon exponentiel. Le SAH est activé par défaut mais peut être désactivé en commentant define USE_SAH dans le fichier bvh.cpp.

2.4 Surface implicite

Le dernier point important abordé dans les TP est l'usage de surface implicite. Pour cela, la première étape a été de créer un sphère tracing. Pour cela, chaque surface implicite possède une sdf retournant la distance de sa géométrie par rapport à un point donné. Nous nous servons donc de ça pour tracer un cercle de notre caméra vers la surface. Ensuite la distance étant retournée nous obtenons le paramètre t pour positionner le centre de notre prochain cercle et on recommence. On itère de cette façon sur l'ensemble du rayon jusqu'à dépasser le tMax imposé ou bien si la distance entre notre point

sur le rayon et la surface descend sous un seuil, dans ce cas une intersection est retournée.

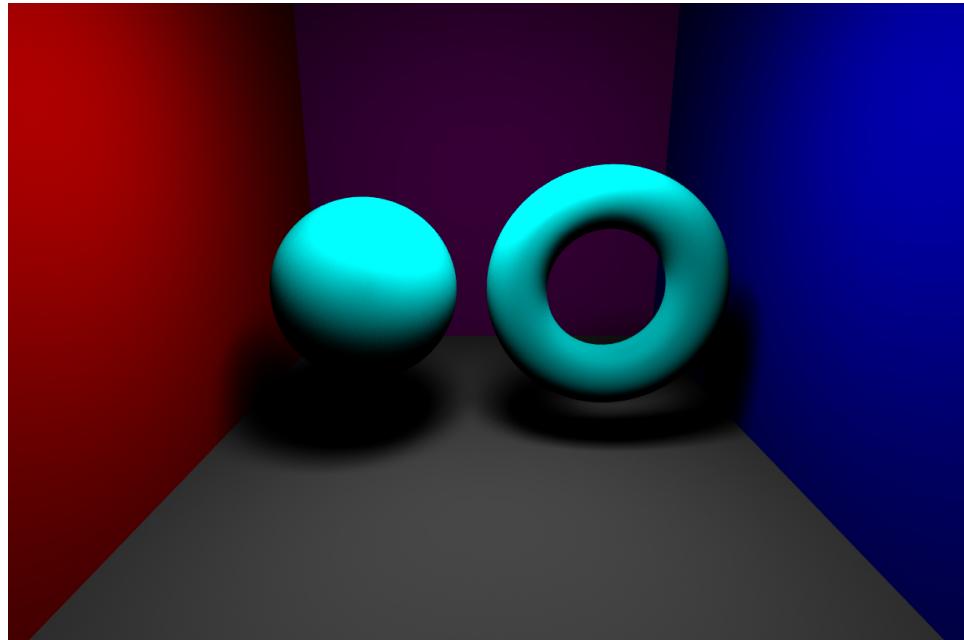


FIGURE 10 – Surface implicite sphère et tore

Concernant les sdf, elles sont toutes données par le biais d'un lien du TP7, il n'y a qu'à soustraire la position souhaitée de la surface au point testé pour afficher la surface au bon endroit. Trois SDF sont implémentés, une sphère, un tore et le mandleBulb comme fractale.

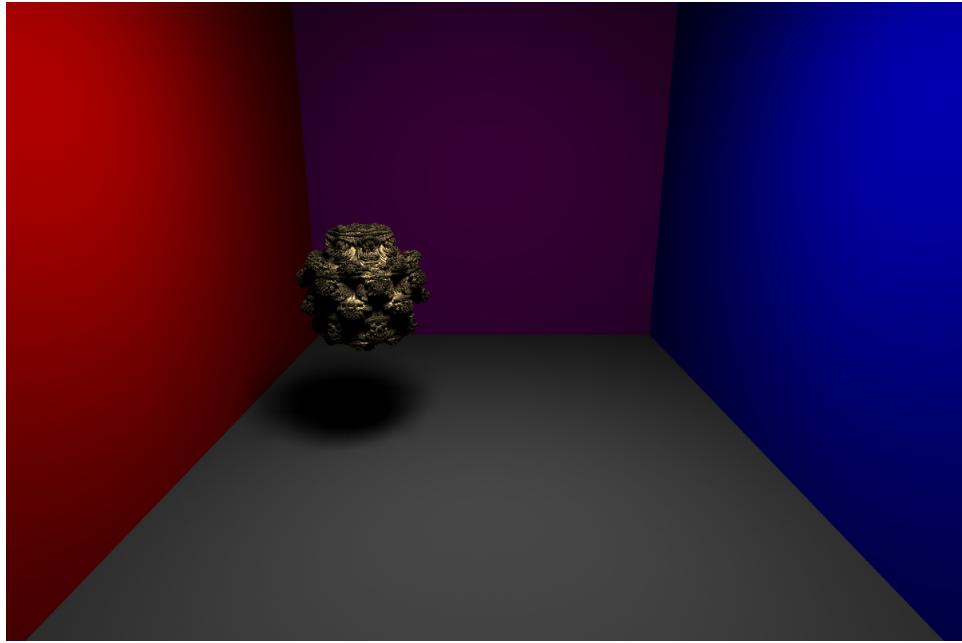
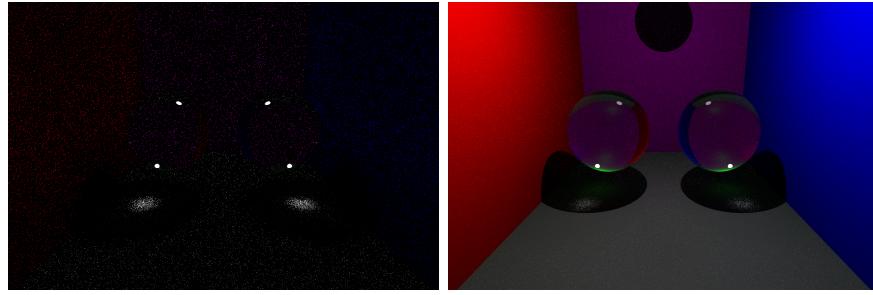


FIGURE 11 – Le mandlebulb

3 Pour aller plus loin

Afin d'améliorer le moteur de rendu, j'ai voulu implémenter un nouvel intégrateur utilisant le path tracing. Pour cela, à chaque boucle au niveau du rendu, en plus de renvoyer l'éclairage direct de notre scène nous allons path tracer l'éclairage indirect. Pour l'éclairage indirect, nous lançons nPath dans directions uniformément aléatoire depuis la première intersection trouvé. On effectue cette démarche pour un nombre de rebond fixé. Une fois tous les paths calculés, on les moyennes pour obtenir la contribution de l'éclairage indirect pour notre pixel. Pour la distribution de rayon uniform j'utilise la méthode mentionnée ici.

Cette méthode combiné au matériaux transparents avec les équations de Fresnel à pour principal qualité de nous fournir des caustiques gratuites. Cependant par manque de temps je n'ai pas pu terminer l'implémentation de ce path tracer. Le résultats obtenus jusqu'à maintenant sont les suivants :



(a) Eclairage indirect

(b) Eclairage direct + indirect

FIGURE 12

4 Conclusion

Pour conclure, l'ensemble des TP sont réalisés avec une grande partie de leurs optionnelles cependant quelques éléments possède quelques soucis dont je n'ai pas parlé avant, une spotLight est implémentée mais présente un problème sur son radius (forme un ovale au lieu d'une sphère). De plus bien que le path Tracer soit implémenté, par manque de temps il n'est pas complètement opérationnel. Mes ambitions étaient en plus de cela d'implémenter la lecture de texture ainsi qu'un algorithme de type stratified pour l'antialiasing que je compte bien continuer même après rendu de ce projet. Cette matière m'aura apporté beaucoup d'élément vis à vis du ray tracing surtout sur les différentes type de rendu qu'il est possible d'effectuer (ray tracing, sphere tracing, path tracing, etc.) et comme dit précédemment je compte continuer ce projet afin d'avoir un moteur solide dont je pourrais me servir d'exemple plus tard. Pour finir " qu'on est les meilleurs profs du monde ".