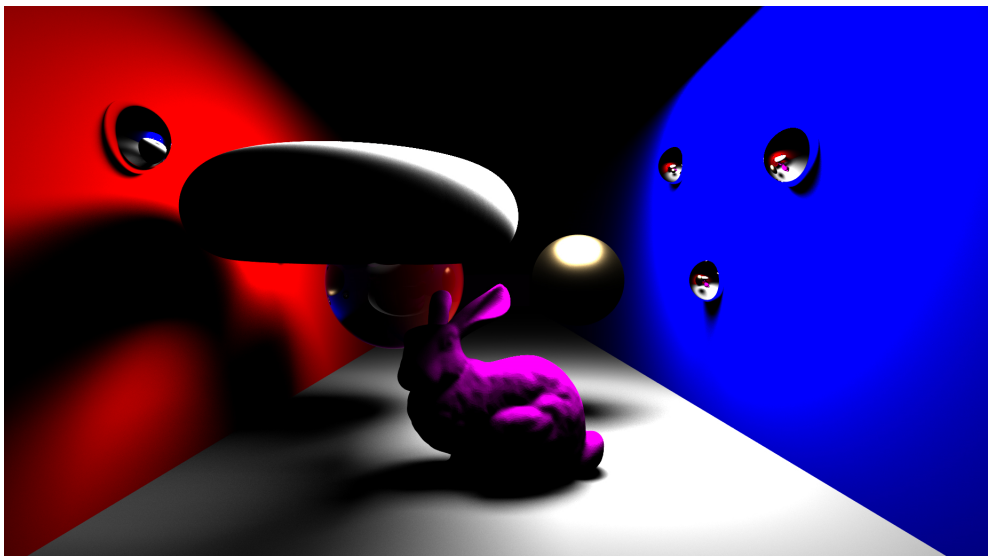
Université de Limoges

ISIR
Rapport de Projet

Yoann SOCHAJ

Mai 2022



1 Introduction

Le but du projet dans sa globalité était de rendre les TP faits durant l'UE, introduction à la synthèse d'images réalistes. Le but de ces TP étaient de reprendre le cours afin de reproduire ceci sur machine à l'aide de notre moteur à

lancer de rayons. Cela nous permet de mieux comprendre les aspects théoriques vu que nous les développons sur machine.

2 TP1

Le premier TP avait pour but de reprendre les bases vu en cours afin de nous préparer à coder notre moteur de lancer de rayon. Mais nous ne partions pas de 0, nous avions à disposition une architecture faite par Monsieur Maxime MARIA. Il fallait donc aussi commencer à se familiariser avec celle-ci.

2.1 Mise de place de la caméra

Dans notre moteur, c'est la camera qui va nous permettre de lancer notre rayon primaire. Ce rayon va nous servir à nous dire ce qui est visible depuis le point de vue. Dans un premier temps, pour calculer la direction de nos rayons primaires, il va falloir initialiser notre fenêtre virtuelle où les rayons vont être lancés.

3 TP2

3.1 Introduction

Le but de ce TP va être de nous introduire aux sources de lumière simple afin de prendre en compte des lumières dans notre moteur, ainsi que de l'ombre.

3.2

Le plan Dans un premier temps, nous allons ajouter un nouvel objet, un plan. Ce plan va nous servir à voir l'ombre de notre sphère. Un plan est composé d'une normale pour avoir son orientation, et d'un point lui appartenant. Il faut donc comme pour la sphère implémenter notre méthode intersection de plan. Pour intersecté un plan, il faut rappeler qu'il est représenté par un point, un vecteur et une normale. Le vecteur, on va pouvoir l'obtenir grâce à la normale et au point. Ensuite, il suffit juste d'injecter notre rayon (O+t*D) dans l'équation du plan (P-Po)*N=0). En résolvant l'équation nous arrivons à trouver une valeurs de t. Si T est positif alors notre rayon intersecte notre plan, à une distance de taille t.

3.3 Prise en compte de notre lumière

Jusqu'à présent, nous ne prenions pas en compte de lumière à proprement parler. Nous allons donc devoir implémenter une lumière, point light. Une lumière simple va être définie par sa couleur et sa puissance. La classe mère de toutes les lumières va posséder une fonction sample qui va falloir définir dans chaque classe de lumière. Celle-ci nous permet de retourner un lightSample qui va pour un point donné, nous renvoyer plein d'informations concernant la luminance à

ce point. Il faut donc construire un objet de type lightSample à renvoyer à notre intégrateur quand il demande la luminance d'un point. La radiance va prendre un compte un facteur d'atténuation qui va nous permettre de réduire sa puissance en fonction de la distance. Nous obtenons donc le résultat de la figure1. Maintenant, il faut calculer les ombres !

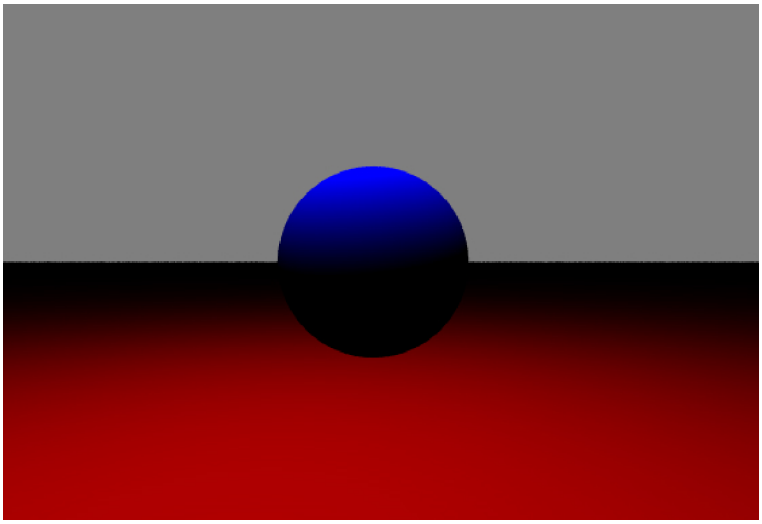


Figure 1: Résultat TP2

3.4 Les ombres

Pour calculer l'ombre en un point donné, il faut lancer un rayon d'ombrage. Pour lancer un rayon d'ombrage, il faut récupérer la position que touche notre rayon primaire, et lancer un rayon vers la lumière. S'il intersecte un objet de la scène alors on est dans l'ombre et on affiche du noir, sinon on va calculer l'éclairage. On va donc devoir récupérer notre lightSample du point afin de savoir dans quelle direction est la lumière. On lance notre méthode pour savoir si avec ce rayon, à partir du point du rayon primaire, on touche un objet de la scène. Mais il va falloir aussi penser à décaler l'origine du rayon, par rapport à la normale, sinon le premier point d'intersection peut être lui-même. Il faut aussi penser à changer PTMax, la variable qui nous dit jusqu'à quelle distance regarder. Car nous voulons seulement aller jusqu'à la lumière. Une fois, ceci fait, on peut voir le résultat attendu dans la figure 2.

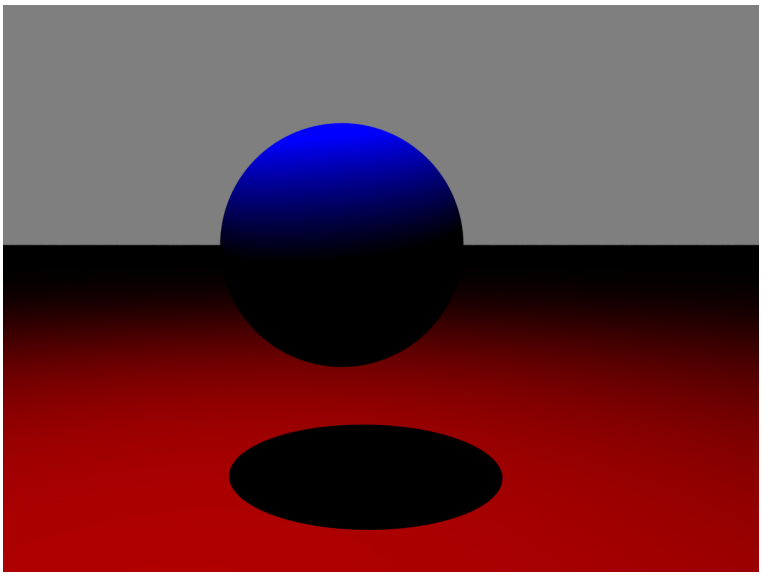


Figure 2: Résultat TP2

C'est un bon rendu, mais ce sont seulement des ombres dures, nous n'avons pas pris en compte les ombres douces, que l'on prendra en compte plus tard. D'abord, il fallut optimiser un peu notre programme. On peut noter que lorsque l'on fait notre rayon d'ombrage, nous n'avons pas besoin de récupérer le point d'intersection, nous avons juste besoin d'un vrai ou faux. On va donc créer une méthode nous permettant de réduire les calculs inutiles.

4 TP3

4.1 Introduction

Comme dis durant la dernière partie, jusqu'à présent nous ne prenons qu'en compte les ombres dures, ce qui ne s'approche pas du tout de la réalité. Pour avoir du rendu réaliste, il va falloir calculer les ombres douces grâce à des sources lumineuses surfaciques. Grâce à cette source lumineuse, on va pouvoir distinguer 3 états différents, soit le point est visible de toute notre surface, alors il est complètement éclairé, soit il n'est pas visible de toutes notre surface, alors il se serra une ombres complètes. Soit le point est visible par certains points de notre light, alors on aura notre zone de pénombre, d'ombre douce.

4.2 La QuadLight

Afin d'implémenter notre quadlight, il faut dans un premier temps la définir. Une quadLight va être définie par une position et de deux vecteurs qui vont

former les coté du quad à partir de ce point. Comme pour une lumière classique, nous aurons aussi une couleur et une puissance. Pour générer un lightSample, il va falloir prendre un point au hasard sur notre quad. Quand on a ce point, on peut tracer le rayon entre le point observé et la direction sur le quad. Et on peut calculer la radiance grâce à tout ça. Mais le rendu n'est pas parfait, car il y a beaucoup de bruit. C'est en partie à cause de la génération aléatoire, et aussi au fait que nous n'avions pas d'ombre douce. Il va donc falloir lancer plusieurs rayons d'ombrages dans le cas d'une lumière surfacique, afin d'obtenir le résultat comme dans la figure 3.

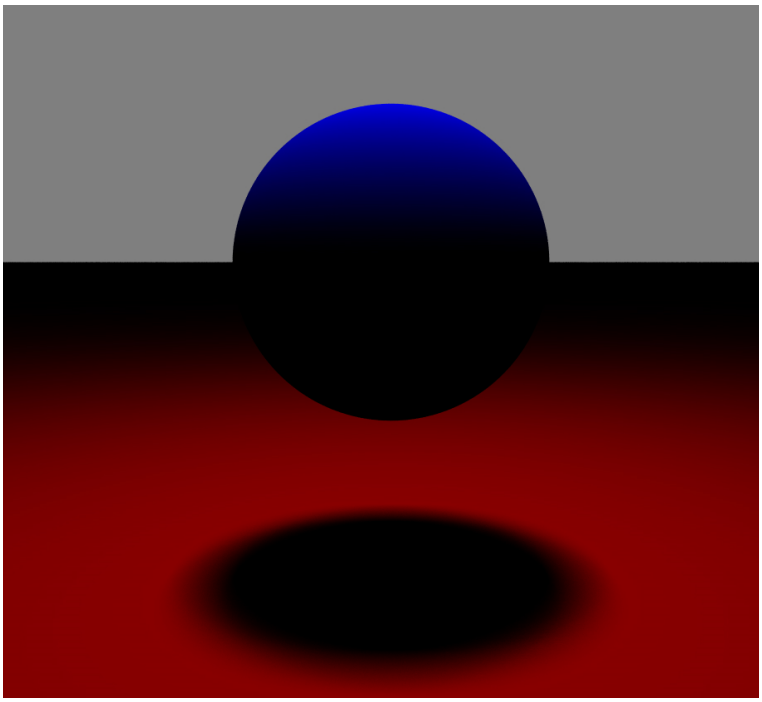


Figure 3: Résultat TP3

Si on combine plusieurs lancer de rayon d'ombre, avec antialiasing, nous allons avoir un résultat très propre, car nous allons lancer plusieurs rayons par pixel, et par ces rayons calculé plusieurs rayons d'ombrage. Nous allons donc calculer une bonne moyenne sur plusieurs échantillons.

5 Les matériaux

Jusqu'à présent, nous n'avions aucun matériau que ce soit spéculaire ou diffus. Durant ce TP, nous allons apprendre à appliquer différents matériaux à des objets dans notre moteur. Nous allons donc dans un premier temps implémenter le modèle de Lambert (parfaitement diffus) et le modèle de phong (spéculaire)

qui sont des modèles de bases, mais très peu réalistes. Le modèle de Lambert va représenter une surface complètement diffuse, c'est-à-dire que la lumière est réfléchie uniformément dans toutes les directions comme dans la figure4.

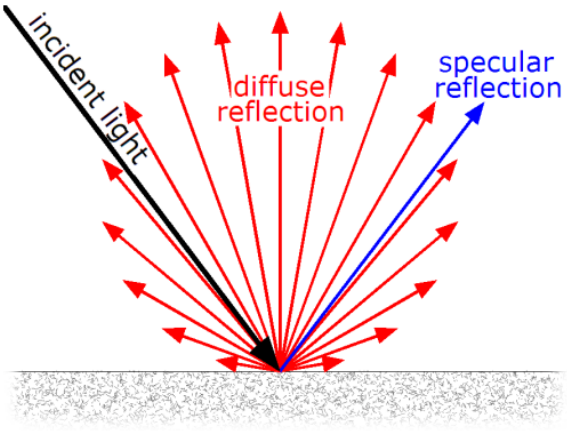


Figure 4: La diffusion

Le modèle de phong lui va représenter le reflet spéculaire (bleu) aligné dans la direction miroir au rayon incident.

5.1 Lambert

Pour l'implémentation, nous allons créer différentes BRF, en fonction du modèle. Pour Lambert la bref sera une constante, car vu que la lumière est réfléchi uniformément dans toutes les surfaces, alors elle ne prends pas en compte les directions d'incidence et d'observations. Mais ceci n'est pas du tout semblable à la réalité, c'est pour cela que nous avons implémenté un autre, le modele de Oren-Nayar.

5.2 Phong

Contrairement à Lambert, nous n'aurons pas une constante pour notre BRDF. En effet, notre brdf va être dépendante de notre angle entre la direction d'observation et la direction incidente. Nous allons aussi pouvoir manipuler la brillance d'un matériau, donc plus il est brillant, plus notre lobe spéculaire serra long et fin. Le modele de phong etant parfaitement symetrique n'est pas du tout correct. Il faudrait que le reflect speculaire se deformer en fonction de la

5.3 Matériau physiquement réaliste

Nous allons implémenter un modèle à microfacettes, celui de COOK-TORRANCE, qui va prendre en compte la rugosité d'un matériau. On va pouvoir définir une surface à petite échelle, afin de représenter une surface entière. On va donc avoir une surface représentée par des "miroirs parfaits", ces miroirs parfait vont donc nous renvoyer de la lumière seulement pour un wh donné, vu que le miroir parfait, va renvoyer la lumière que dans une direction. Nous allons donc, calculer la probabilité que des miroirs nous renvoient de la lumière sur une surface donnée. Cette valeur va déprendre de la rugosité. Nous avons ensuite calculé la qualité de lumière perdue par masquage et ombrage. Le masquage est la lumière qui va être cacher de notre oeil, car elle serra dans les creux d'un matériau, l'ombrage, c'est quand la lumière, elle-même, va être bloquée par une bosse. Un dernier terme Fresnel, va nous permettre d'avoir la proportion de réflexion de la surface pour un angle donné. Ces trois fonctions regroupées vont nous permettre d'avoir la BRDF à microfacettes . Après l'implémentation des calculs mathématiques, nous avons obtenu les résultats de la figure 5.

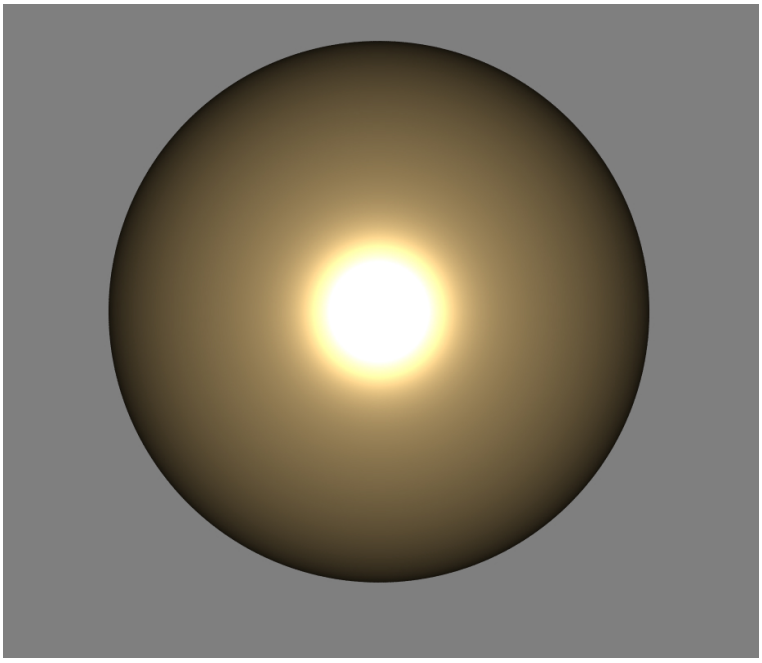


Figure 5: Cook Torrance

6 TP5

Lors de ce tp nous allons apprendre à modéliser les phénomènes de réflexion et de réfraction. Nous allons utiliser une méthode simple, en récursion, qui nous dit qu'un rayon peut être réfléchi ou réfracte.

6.1 Matériau parfaitement spéculaire

Dans le principe, simuler un miroir parfait est assez simple. Dans notre intégrateur quand notre avons une intersection sur un miroir, on doit alors lancer un nouveau rayon vers la direction de réflexion par rapport à la normale comme le montre la figure 6 pris du cours de monsieur Frederic CLAUX.

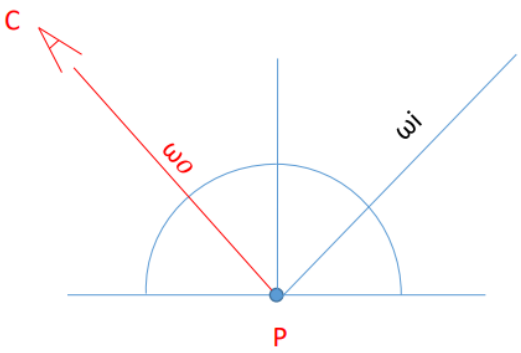


Figure 6: Miroir parfait

Si c'est encore un miroir alors on répète le processus, si ce n'est pas un miroir, alors on récupère la lumière en ce point. Il faut faire attention, par contre à avoir une variable qui va compter le nombre de rebonds, pour éviter de trop rebondir, voir d'avoir des boucles infinies.

6.2 Matériau transparent

Pour le cas d'un matériau transparent, c'est plus compliqué. En effet, dans ce cas-là, nous aurons un rayon réfracté, et un rayon réfléchi. Et grâce à l'équation de Fresnel, on aura la proportion de lumière réfléchie. Nous devons aussi prendre en compte les différents indices de réfraction, car quand nous sommes dans un objet, ou dans le vide, l'indice de réfraction sera différent. Après avoir calculé notre lumière réfléchie et notre réfracté, nous arrivons à obtenir la lumière finale en utilisant Fresnel. Il faut juste penser à prendre en compte le cas de réflexion totale. S'il a lieu, alors nous n'avons pas besoin de calculer la réfraction. Après l'implantation de nos deux sphères, nous obtenions les résultats de la figure.

Malheureusement le résultat ne prends pas en compte la réfraction pour les rayon d'ombrages.

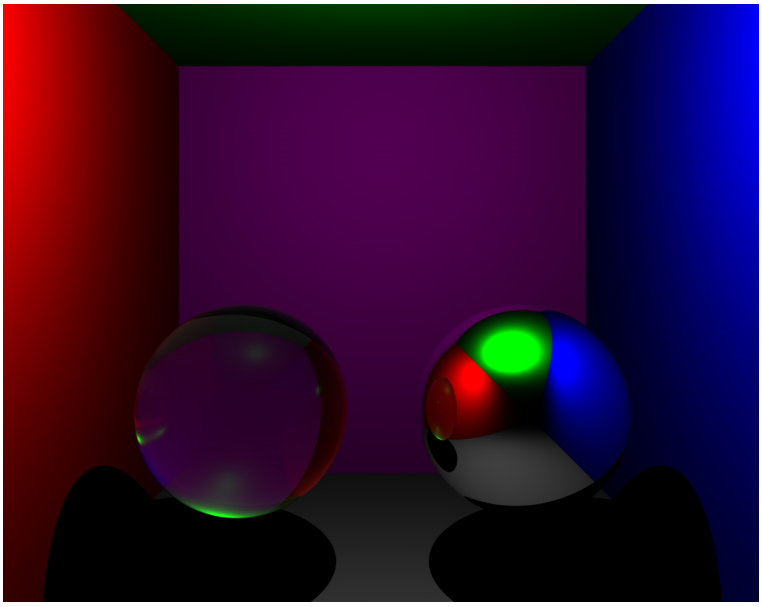


Figure 7: Résultat TP5

7 Les maillages

Le but de ces projets de ce TP, est de pouvoir prendre en compte des objets dans notre scène. C'est-à-dire prendre en compte les maillages de plusieurs triangles.

7.1 Premiers maillages

Pour prendre en compte les maillages dans notre moteur de lancer de rayon, il va falloir ajouter plusieurs méthodes. Déjà, nous avons une méthode fournie par Monsieur Maxime MARIA nous permettant de charger un modèle 3D. Ensuite, il a fallu mettre en place une méthode pour intersecté des triangles. Pour ce faire, on va projeter notre point sur le plan du triangle, puis on regarder si dans cette projection le point est dedans ou pas grave aux coordonnées barycentrique.

8 Amélioration

Le résultat est beau en visuel, mais en terme de performance, ce n'est vraiment pas bon, il va falloir utiliser des méthodes d'intersections plus rapides. Car en effet quand on intersecte, on va à chaque fois essayer tous les triangles de l'objet par pixel. On va donc dans un premier temps utiliser des boites englobantes. Ces boites vont nous permettre de limiter le nombre de calcul, car on testera les triangles seulement si nous intersectons la boite englobante. Ensuite, pour notre programme, chaque triangle va être englobé par son aabb. Chaque fois qu'on va ajouter un triangle à un mesh, on augmente notre aabb du mesh avec le aabb du triangle. Et ainsi, de suite, on va avoir notre AABB du mesh. Grâce à cette technique, nous allons pouvoir gagner du temps, mais il faut maintenant implémenter la méthode pour intersecté un AABB. On va dans un premier temps séparer notre AABB, en axes parallèles. Sur chacun de ces axes nous allons pouvoir localiser l'endroit ou le ray passe. Il suffit apres de verifier pour chaque axe, si il passe entre le min et le max de l'aabb.

Nous pouvons remarquer que nous gagnions beaucoup de temps ! Mais ce n'est pas la seule optimisation disponible, on peut mettre au point un BVH afin d'augmenter nos performances, et d'importer de plus gros mesh.

Le BVH, va être un arbre, où aux feuilles, nous allons retrouver certains triangles. Nous allons avoir deux méthodes principales. Une méthode pour intersecter un bvh et une autre pour le construire. Pour construire notre bvh, on va devoir d'abord choisir le nombre minimal de triangle maximum par feuille. Par exemple, à partir de 8 triangles, nous allons arrêter la récursion, et ajouter nos triangles à la feuille courante du bvh. Ceci va donc être notre critère d'arrêt de récursion. Ensuite, afin de construire un arbre, il faut pouvoir diviser les triangles. Pour ce faire, nous allons devoir localiser le plus grand axe. Nous allons calculer le milieu du plus grand d'axe. Puis nous allons utiliser partial sort les triangles associés au nœud. Grace au milieu calculé précédemment, on va pouvoir récupérer l'id du triangle au moment où l'on dépasse ce milieu par rapport à l'axe le plus grand. Ceci va alors composer nos deux partitions. On va donc relancer la fonction récursive à gauche sur le premier triangle jusqu'au milieu, puis à droite du milieu jusqu'au dernier. Nous avons cette figure 8 qui montre bien comment les triangles vont se repartir par rapport au plus grand axe.

Pour la fonction intersection du bvh. Il va falloir descendre jusqu'aux feuilles.

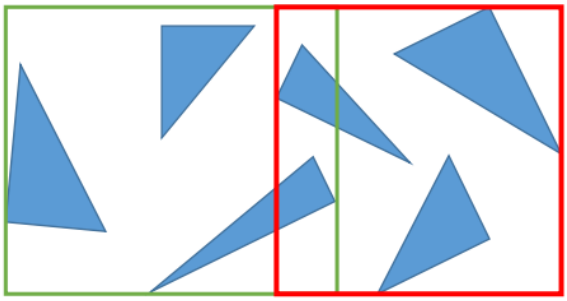


Figure 8: Séparation des triangles

Arrivé aux feuilles, nous allons alors intersecter tous les triangles présents dans celle-ci. Quand on retourne les résultats du nœud droit et gauche, on va alors les comparer pour retourner le plus proche. Afin de descendre dans les bons nœuds de l'arbre, on va seulement lancer la récursion sur les enfants où leurs AABB sont intersectés. Grâce à ce principe cela va nous permettre d'importer de plus gros objets, ou d'en importer plusieurs ! Nous avons voulu implementer une SAH pour notre bvh, afin de reduire le temps de calcul lors de l'intersection. Pour se faire, il faut,

Nous obtenons le résultat de la figure 9.

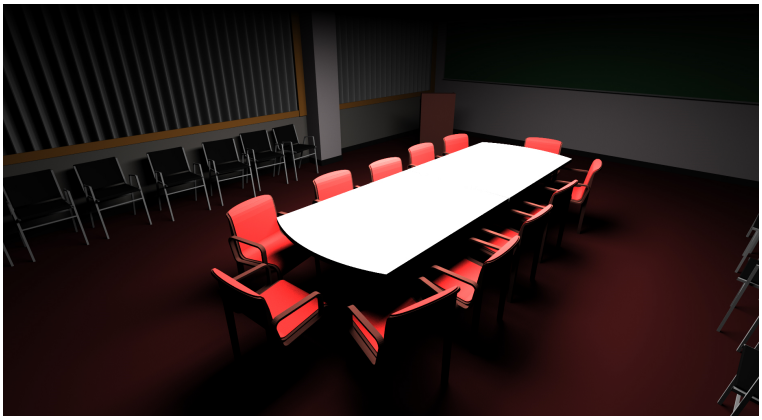


Figure 9: Résultat TP6

9 TP7

Le but de ce tp était de reprendre le cours sur les surfaces implicites afin de mieux le comprendre. Nous avons utilisé la méthode du sphère tracing, afin de calculer les intersections. La méthode des surfaces implicites est séparée en deux parties, déjà notre forme va posséder une méthode sdf, qui va nous renvoyer pour un point donné la distance minimale à la forme. Ceci va pouvoir être utilisé pour le sphère tracing, car cela veut dire qu'on peut avancer sur notre rayon d'au moins cette distance. Si la sdf s'approche de notre distance minimale choisis alors nous allons considérer qu'on intersect, et alors remplir le hitRecord. Nous avons réussis à implémenter plusieurs formes comme la sphère et un Tore. Nous avons aussi vu comment appliquer une rotation et une translation a notre objet implicites. Il suffit, d'ajouter une point courant une rotation selon l'axe que l'on veut grâce au formule de cosinus et de sinus.

Apres ceci, nous nous sommes pas mal renseigner concernant les fractales afin d'essayer d'en proposer une. A l'aide de ressource sur internet sur le MandelBulb, nous arrivions à comprendre certain point. Ce qu'il y a d'abord a comprendre, c'est qu'un nombre complexe existe sur deux axes, l'axe reel et l'axe imaginaire. On peut multiplier un nombre complexe à un autre, il faut alors multiplier les normes, et ajouter leurs angles. Au cours d'une boucle si on augmente z au carré, on va alors au fur à mesure s'eloigner de l'origine. Donc quelque soit la distance a laquelle on l'observe, elle gardera la même forme globale. Notre fractale va donc découler l'ensemble de mandelbrot, mais cette fois si en 3D. Nous sommes arriver au resultat de la figure 10 pour notre fractale. Le resultat aurait pu etre encore meilleur si nous avions pu coder de la global illumination.

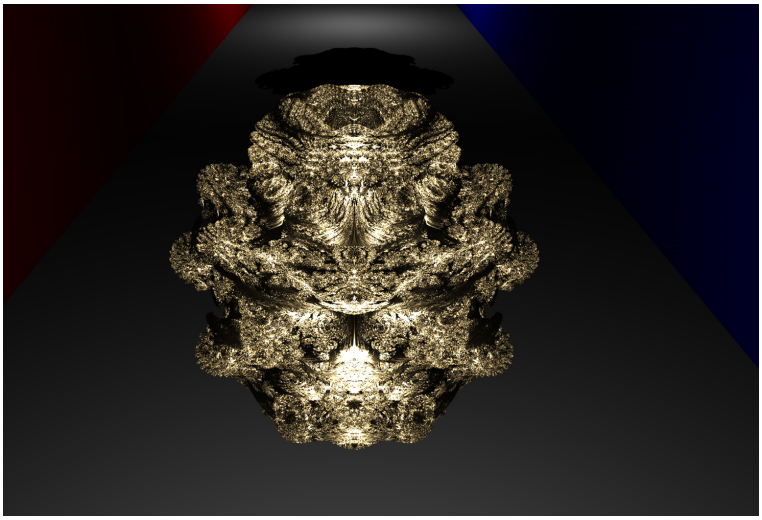


Figure 10: MandelBulb

10 Conclusion

Pour moi, ces TP ont été très intéressants, il permettait de passer plus de temps sur des notions vu en cours et de les implémenter. Je suis plutôt fière du travail que j'ai produit lors des TPs. Mais très déçu de ce que j'ai produit hors des tp, je reste extrêmement frustre, car j'ai été bloqué trop de temps, et j'ai eu beaucoup de mal à avancer. J'ai réellement voulu faire plein de fonctionnalité, mais je n'arrivai à rien, certainement le stress des autres projets à côté.. J'ai commencé à un implémenter un BVH utilisant le sah et une spot light. J'ai aussi voulu approfondi mes compétences en surface implicite, j'ai donc fait un Mandelbulb à l'aide de pas mal de ressource sur internet. Je me suis pas mal renseigné sur le texturing, la CSG, le depth of field, et Monte-Carlo afin de produire des caustique, mais sans succès .. Je penses à vouloir trop en faire, je me suis vite perdu..

References

[1] scratchpixel

[2] pbrt

[3] stackOverflow

[4] Cours de monsieur CLAUX

[5] Wikipedia

[6] shadertoy

[7] iquilezles.org

[8] https://raytracing.github.io/

[9] pbr-book.org