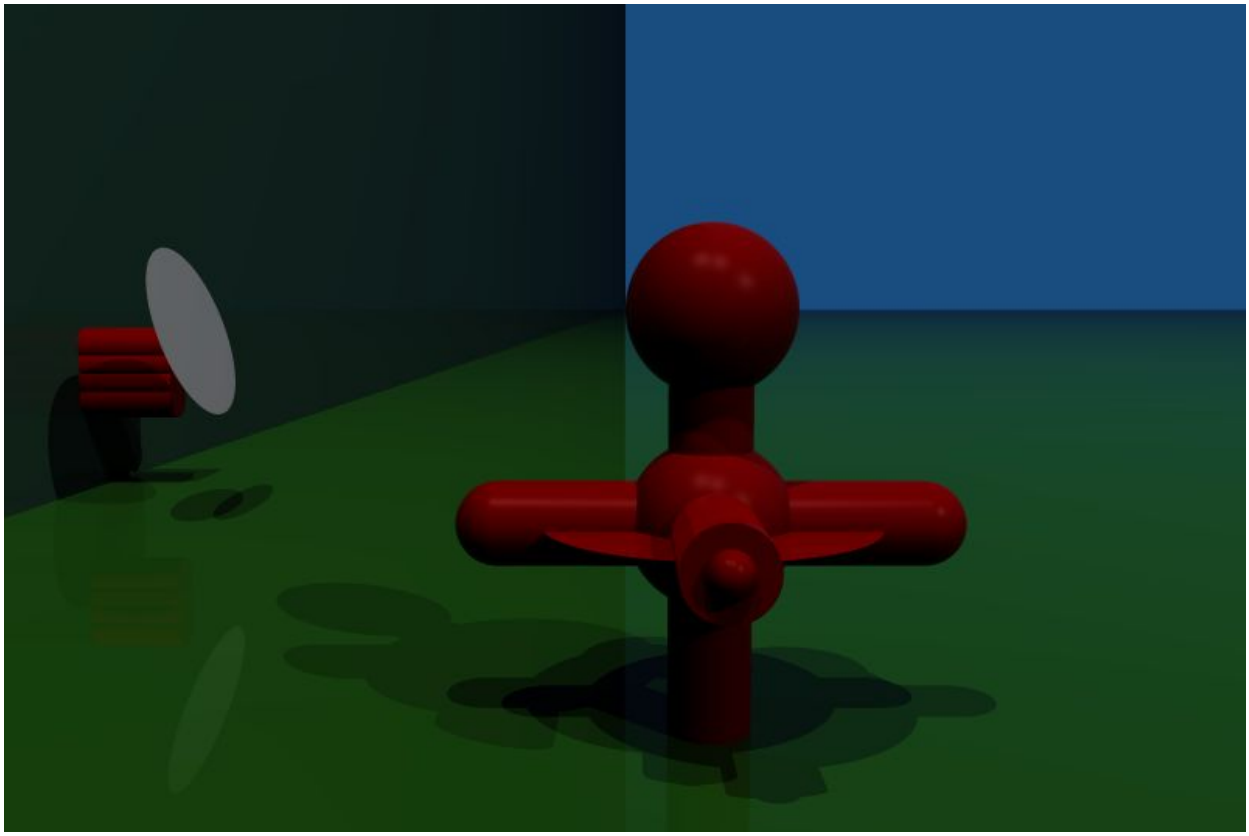


TP Mini-RayTracer

02.2018 - 16.03.2018

MMADI Anzilane

Vue d'ensemble

Créer des scènes avec différents objets 3D (triangle, sphère, plan, disque, etc...) grâce au synthese d'image en 3D.

mettre en place une structure d'accélération (kd-tree).

Objectifs

1. Implémenter un rendu par lancer de rayon (ray-tracing)

La génération de l'image de synthèse se fait en calculant l'intersection de rayons avec la surface des objets de la scène 3D, décrits par leur géométrie et leur matériau. À chaque point d'intersection, on fait une simulation d'éclairage dont le résultat est une couleur qui part dans la direction du rayon. (src : sujet tp)

Caractéristiques

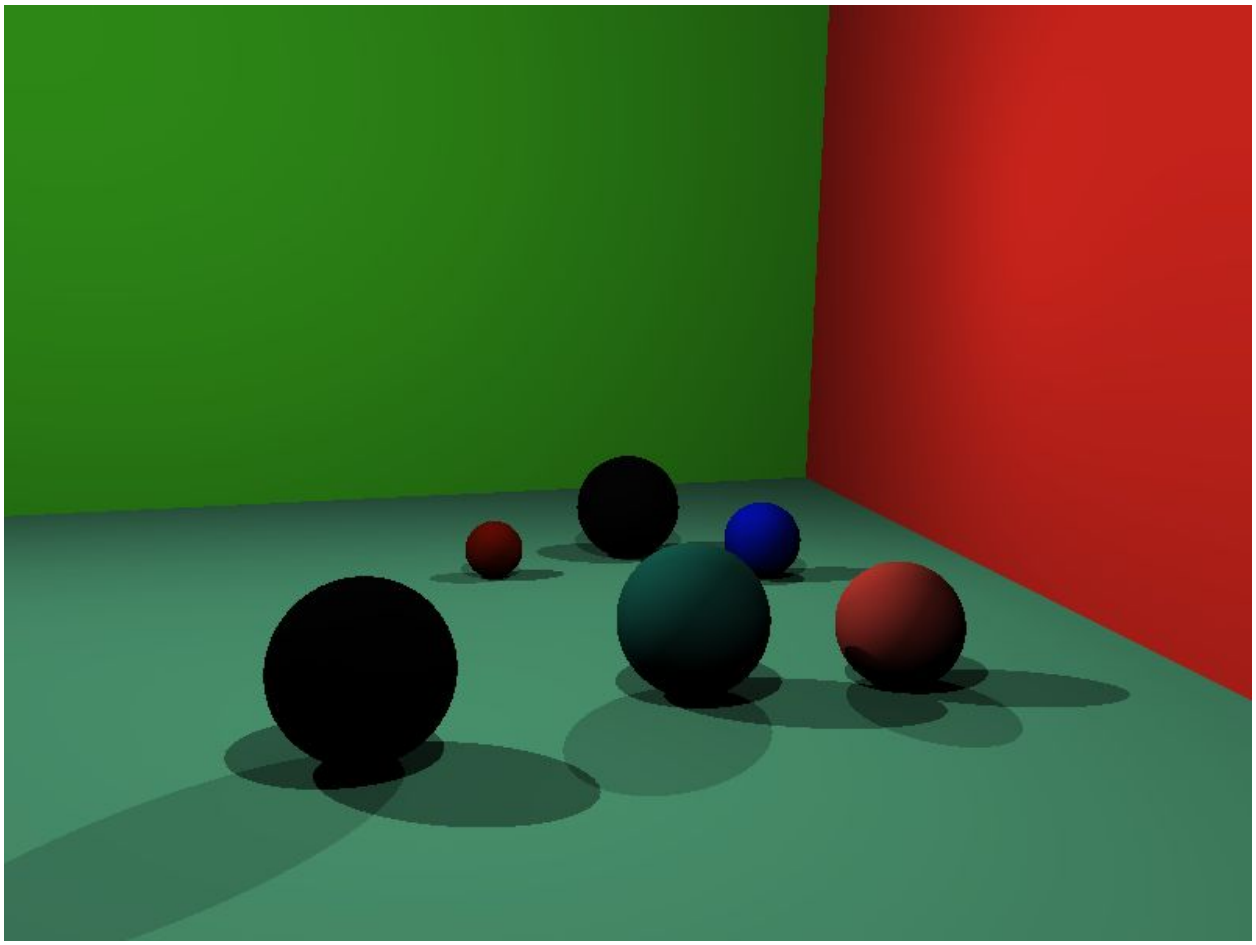
Une partie du code nous a été fournie, et implémentée en C/C++.
On a eu le choix du langage d'implémentation entre le C ou C++
Une bibliothèque de calcul matriciel et vectoriel nous a été fournie (GLM) ainsi que **lodepng** : une autre bibliothèque qui permet d'exporter des images au format PNG, ce qui nous permettra de visualiser les résultats de notre lancer de rayon.

Grandes étapes

I. Intersections et éclairage simplifier

Dans cette partie, nous allons mettre en place la structure de base du lancer de rayons avec un modèle simple d'éclairage. (src : sujet tp)

- **Calcul d'intersection**
  Mes scènes vont avoir cinq types d'objets (sphère, triangle, disque, plan, cylindre) ce qu'implique le calcul des cinq intersections différentes
- **Couleur et shading**
  cette section nous permet d'ajouter la couleur aux objets intersectés dans la scène, pour cela j'ai implémenté la fonction shade, qui sera modifié au fure et mesur que le projet avance.
- **Ombre**
  A ce stade, on procède à l'implémentation des ombres porté. Pour cela on modifie la fonction trace_ray avec ce mini code :



résultat après ajout des ombre

- for (size_t i=0; i<lightCount; i++) {
  // lgt = scene->lights[i];
  // l = normalize(lgt->position-p);
  // newRay.orig = p+acne_eps*(l);
  // newRay.dir = l;
  // newRay.tmin = 0; // le min est à 0
  // newRay.tmax = (lgt->position.x - p.x)/(newRay.dir).x;
  // lc = lgt->color;
  // Intersection interOmbre;
  // if (!intersectScene(scene, &newRay, &interOmbre)) {
  // ret += shade(n, v, l, lc, intersection.mat);
  // }
  // }
-

II. Vers un modèle d'éclairage plus réaliste

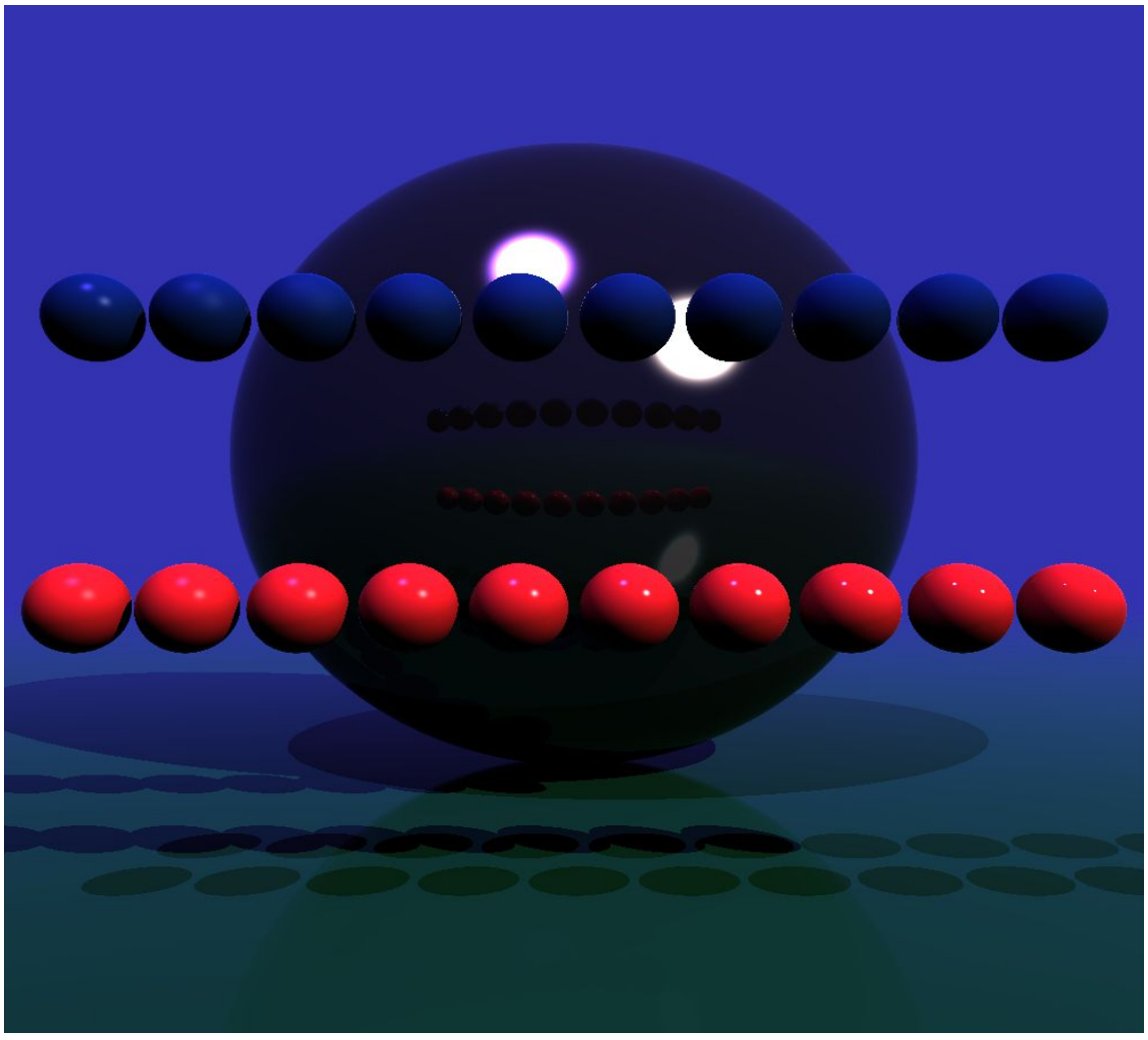
Dans cette partie on a amélioré la fonction shade pour donner un aspect un peu plus réaliste, et ce grâce au calcul du BSDF (Bidirectionnal Scattering Distribution Function). différentes fonctions sont développées.

- float RDM_Beckmann (float NdotH, float alpha); : **distribution de beckman** qui permet de calculer la distribution des micro facettes sur un matériau rigoureux.

- float RDM_Fresnel(float LdotH, float extIOR, float intIOR); : **Le Terme de Fresnel** : permet la représentation de quantité de lumière réfléchie.

Ces différentes fonctions plus l'atténuation, permettent le calcul du BSDF.

- **Réflexions :** On calcule la réflexion grâce à une récursion sur la fonction trace ray avec un nombre de rebond maximal fixé pour éviter une infinité

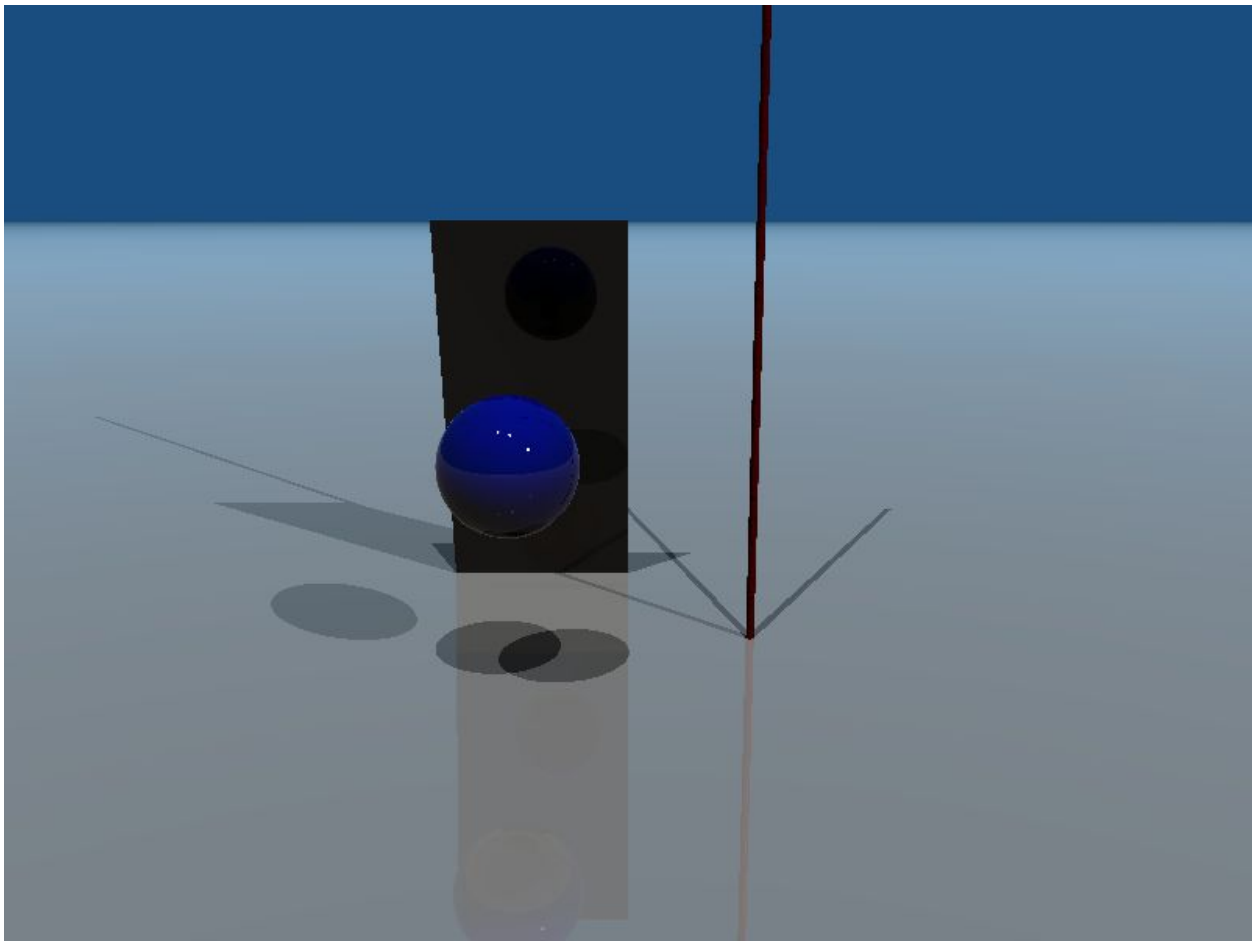


ave réflexion

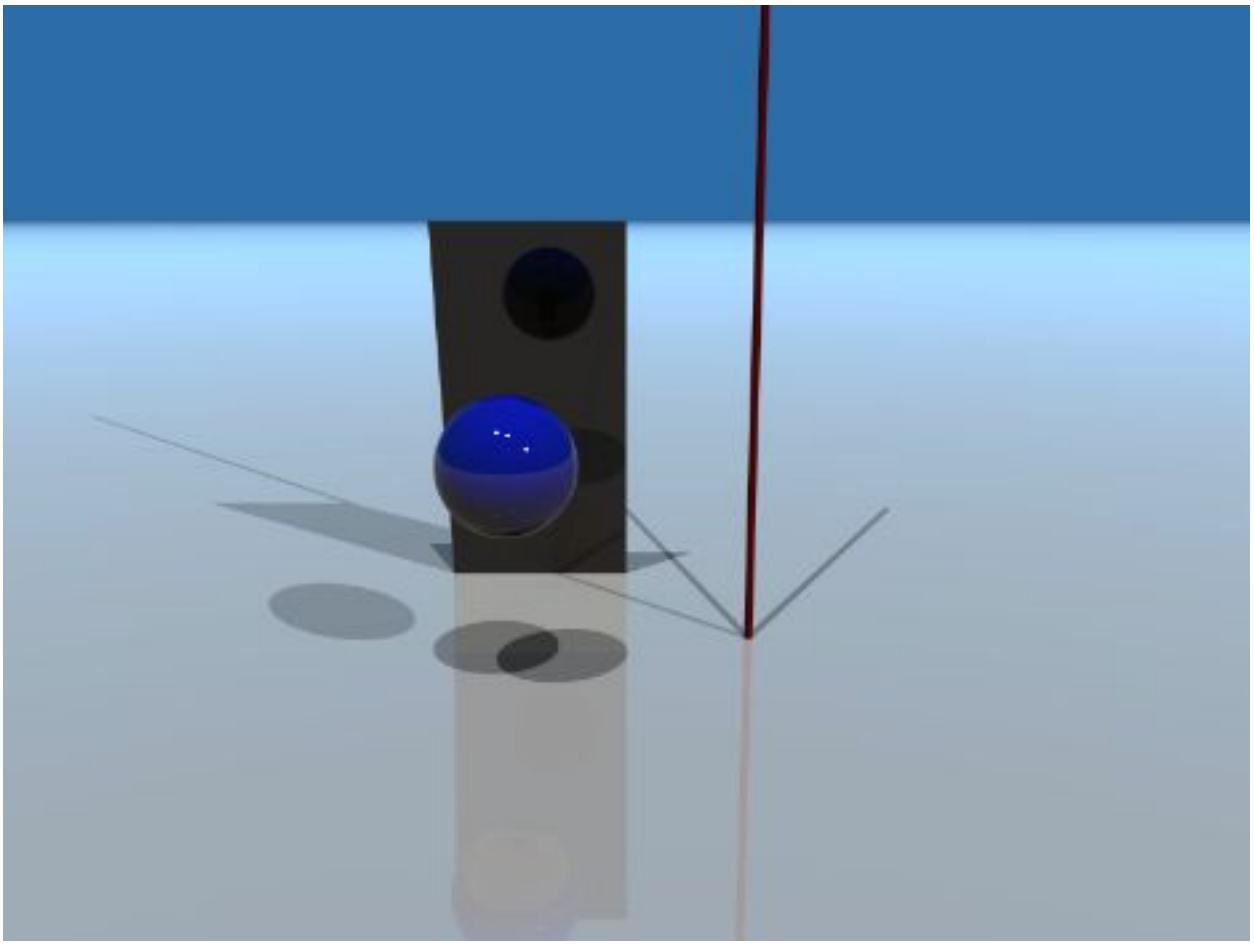
III. Fonctionnalité avancées

Dans cette partie, je me suis sentie un peu à l'aise avec le projet, je l'ai mieux compris, et je me suis permis d'implémenter quelque petite fonctionnalité en plus.

- **Antialiasing** : pour gérer l'effet escalier sur les contours des images. J'ai donc implémenté deux version.
  -Une en calculant les points de façon régulière sur l'interval -0.5 - 0.5
  -Et l'autre en calculant l'interval sur -0.5 - 0.5 et ensuite je trouve un point aléatoire sur cet interval. La différence entre ces deux méthodes se constate lorsque le nombre de rayon lancé est important, dans ce cas la seconde version est plus pertinente, si non la différence ne se voit pas.



sans antialiasing 1 rayon



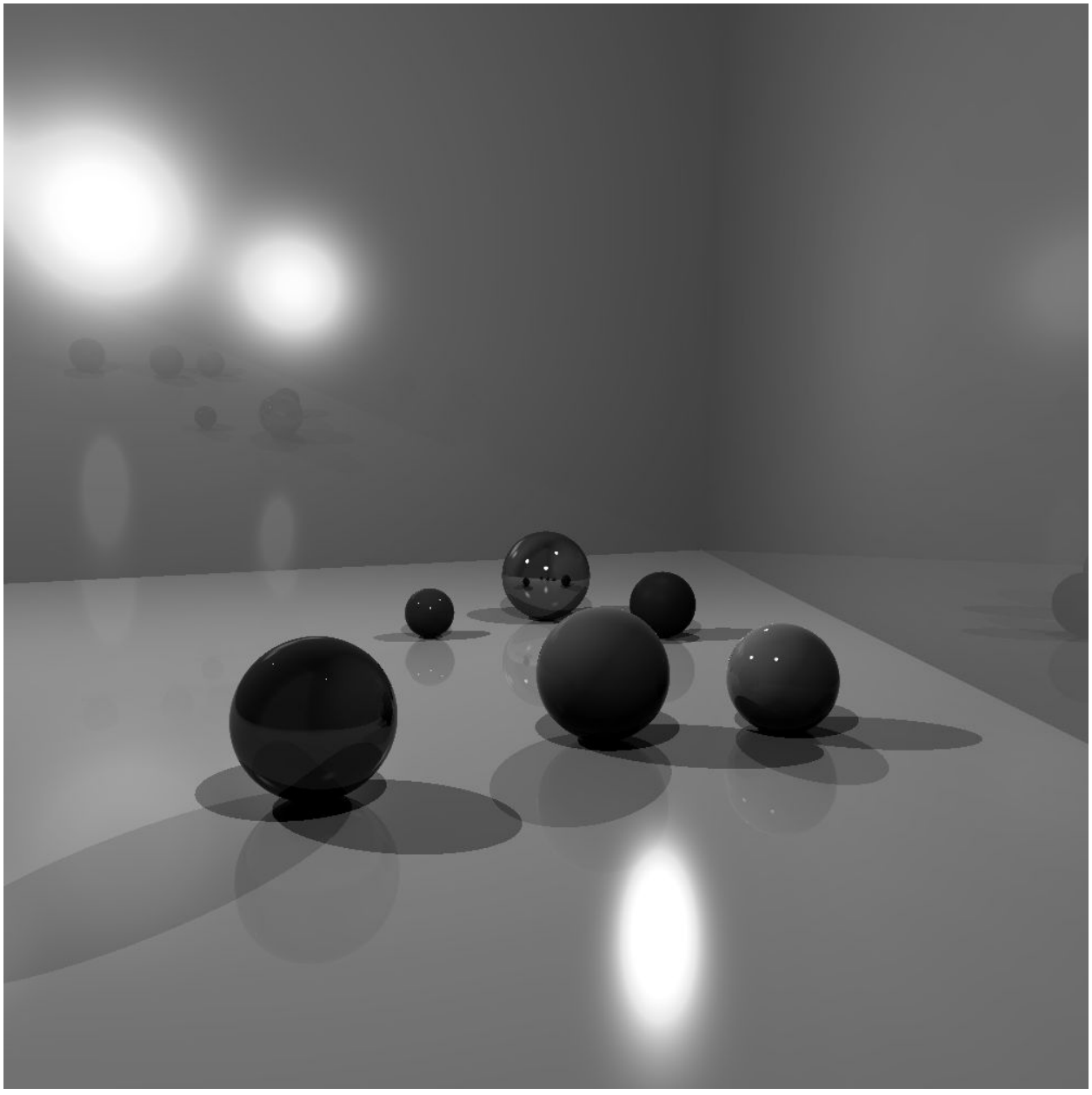
avec un antialiasing de 8 rayons lancés

- J'ai commencé l'implémentation du **Kd Tree** : la structure d'accélération, mais pour des bugs non encore résolue j'ai juste le code.

- **Autre Objets :**
  J'ai par la suite ajouter différents objets : comme le triangle, disque, cylindre (on ne peut l'orienté que sur les axe x = 0, y = 1, z = 2 : valeur possible pour l'orientation du cylindre).

- **Faites nous rêver**
  Je ne sais pas si cela vous fait rêver mais j'ai implémenté une fonction permettant de réaliser un maillage en triangle, j'ai d'ailleurs différentes figures que j'ai pris sur le net (3D model : src de mes fichier obj que j'ai modifié pour les lire plus simplement, car je n'utiliserai pas certaine fonctionnalité comme la texture).

- **TP Analyse d'image**
  J'ai appliqué à certaines images créées par la synthèse d'image, le tp d'analyse d'image. des résultat plutôt drôle



niveau de gris

IV. Difficulté Rencontré

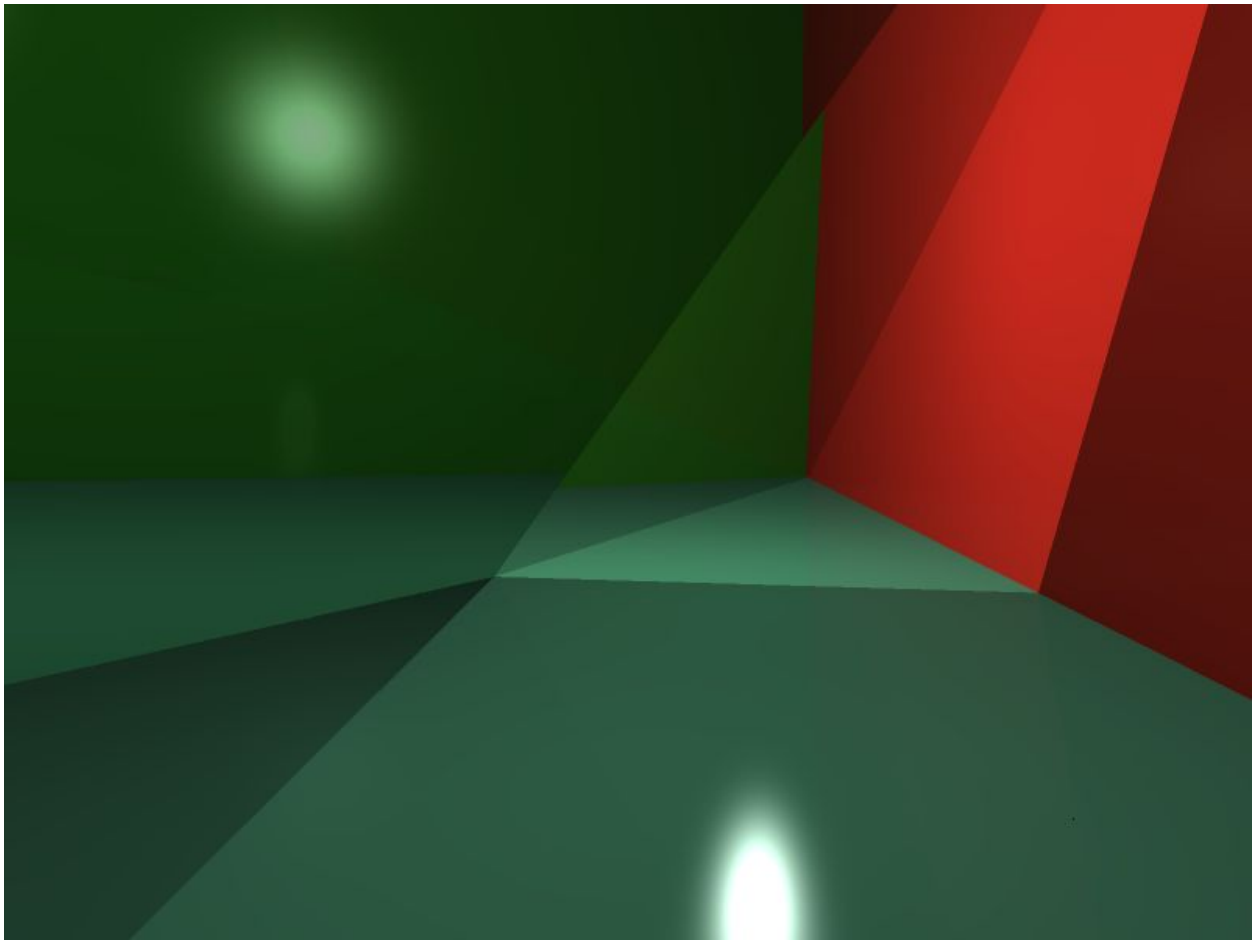
- La première difficulté que j'ai pu rencontré, fut me repérer dans les différentes scènes: savoir orienter la caméra, positionnée mes object correctement lors de la

création de nouvelle scène etc..

⇒ Pour palier à ce problème : au départ j'ai utilisé la sphère, car étant représentée par un seul point, c'était plus simple de la positionner. Par la suite j'ai implémenter le cylindre et la c'est devenu un peu plus simple car je n'avais qu'à représenter 3 cylindre m'indiquant l'orientation des x, y, z sur le scène et par la suite trouver le point 0.

- Second difficulté : la réflexion, lors de l'appel récursif, j'avais du mal avec le cas d'arrêt.

  ⇒ j'ai résolue le problème grâce à l'aide du professeur (thibault.lejemble)
- L'intersection Triangle m'a posé un petit problème sur le calcul de la normal.



bug sur l'intersection triangle

- Le kd tree, malgré les nombreux explication et document sur le net (je peux dire que j'ai bien saisis son sens et son fonctionnement), je n'arrivais toujours pas à

comprendre le fichier kdtree.cpp (car on peut dire que ce n'était pas du tout expliqué, il fallait presque tout deviner, qu'est ce qui représente quoi), quand j'ai compris il était un peu trop tard, mais je continue l'implémentation quand même (pour moi). Je vous transmets quand même ce que j'ai pu faire la dessus.

- L'un des problème majeur que j'ai pu rencontré c'est l'emprise que ce projet a eu sur moi. Au début j'étais septique, mais une fois en tp j'ai accroché en une semaine j'avais déjà fini la réflexion. Ensuite j'ai passé pas mal de temps à chercher les autres objets. J'ai toujours pas réussi à faire le cône.

  ⇒ J'ai quand même fait attention à faire les autre TP qu'on a eu. Mais j'avoue avoir passer pas mal de temps sur ce projet et j'en passe encore à titre personnel.

V. Résumé

Pour résumer, je dirais que j'ai beaucoup appris dans ce domaine grâce à ce petit projet, mais surtout pris énormément de plaisir à le faire et aujourd'hui je suis entrain de me demander, si je veux pas en faire mon métier.

Quelques fonction Personnel.

Dans le main vous pouvez trouver différentes fonctions implémentées pour insérer objets : comme un quadrilatère, un cylindre, une fonction d'insertion d'objet en maillage triangulaire...

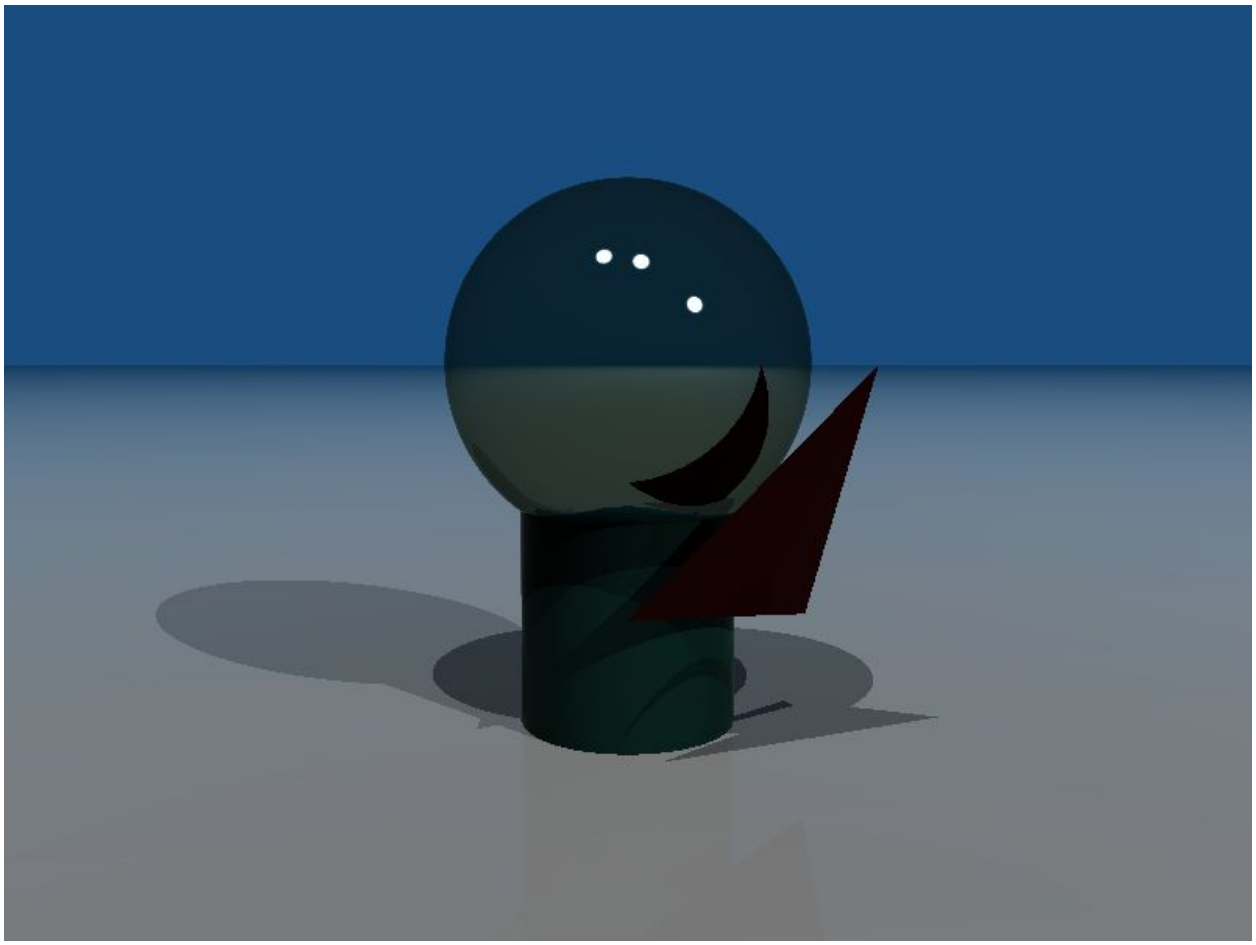
void insererUnObjetEnMaillage(Scene *scene, const char * filename, int nombreDeFace, int nombreDePoint, float miseAlEchelle, Material mat) : cette fonction insère des objets en maillage triangulé, les paramètres sont assez explicite information important : pour la variable nombreDePoint il faut ajouter 1 à la valeur sinon vous risqué d'avoir un segfault.
Je précise que le fichier de lecture pour générer les objets et modifier par moi et donc la fonction de lecture est adapter à ce fichier je vous en transmettrai un exemple.

void addCylinder(Scene * scene, float r, float h, point3 centre, int orientation, Material mat) : comme son nom l'indique, elle ajoute un cylindre en fonction de son rayon de sa hauteur, de sa position et de son orientation sur les axes x y z (0, 1, 2 respectivement).

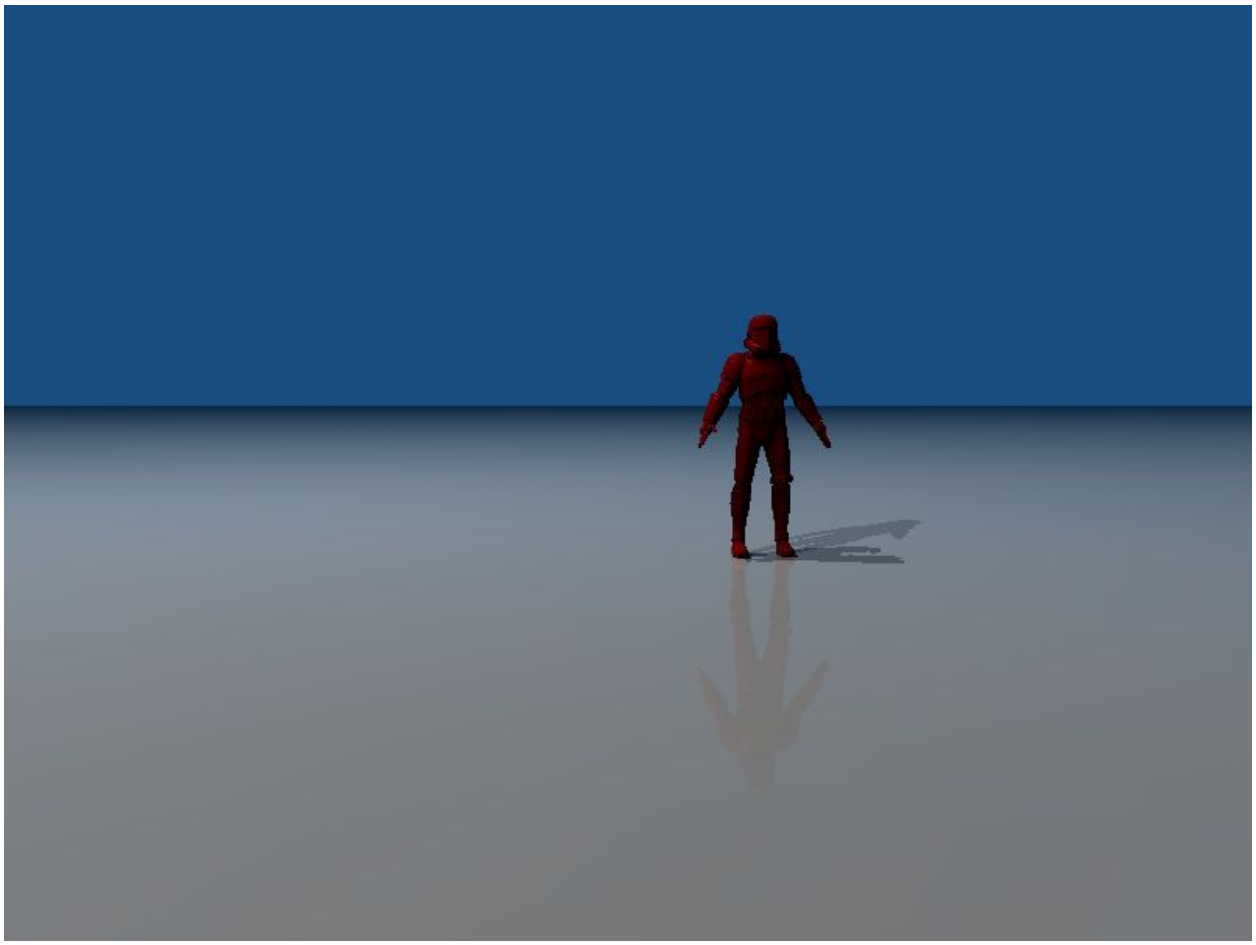
....

Quelques Images

loup lancé avec 2 rayon en antialiasing sans kdtree



le bonhomme du ciel



... le reste des image seront dans l'archive