MPIL(3I008)
Année 2016/2017
Mini-projet : lancer de rayons - partie 2

Objectifs

1. Chargement d'une scène
2. Plus d'objets
3. Parcours d'arbre
4. Plus de phénomènes physiques

Plusieurs questions ou exercices sont facultatifs mais permettent d'obtenir un bonus de points.

Exercice 1 : Chargement d'une scène

Au lieu de décrire vos scènes en OCaml, et d'avoir à recompiler à chaque changement de scène, il est plus pratique de définir ses scènes dans un fichier et de lancer le programme dessus.

Nous allons utiliser le format (très simple) suivant :

— Chaque ligne correspond à un élément de la scène
— Les éléments peuvent être une *caméra*, un *objet*, un *material*, une *lumière*.
— Chaque ligne commence par un mot clé qui indique l'élément, suivi de paramètres pour préciser cet élément.

Les éléments sont :

camera x0 y0 z0 xA yA zA alpha width height où $(x0, y0, z0)$ est l'origine de la caméra, $x1, yA, zA$ est le point de visée, alpha le champ de vision, width et height les dimensions de la fenêtre de rendu.

material name r g b ka kd ks alpha où name est le nom du *material*, (r, g, b) est sa couleur, (ka, kd, ks) les coefficients pour l'illumination ambiante, diffuse, et spéculaire, et *alpha*, l'exposant dans le modèle de Blinn-Phong.

sphere xC yC zC radius material où (xC, yC, zC) est le centre de la sphère, radius le rayon, material le nom du *material* à appliquer sur la sphère.

light x y z r g b i où (x, y, z) est la direction de la lumière distante, (r, g, b) est la couleur de la lumière, et i est son intensité.

Q.1.1 Un module OCaml vous est fourni pour parser de tels fichiers de description de scène à l'adresse `https://drive.google.com/file/d/0BzuP8lg-Dqt6R1JTSndzbnNrelE`. Adaptez le module à vos propres types et fonction pour manipuler les vecteurs et les éléments de la scène.

Faire en sorte que votre lanceur de rayon puisse être appelé en ligne de commande avec un fichier de description de scène en argument.

Q.1.2 Tester votre lanceur de rayon sur la description de scène suivante :

```
camera 0. 0. 0. 0. 0. 5. 2.0943951 700 500
material sphereMat 255 0 0 1. 0.6 0.7 10.
sphere 1. 1. 5. 0.4 sphereMat
light -10. -30. 20. 255 255 255 0.1
```

Q.1.3 *Facultatif* Écrire une fonction pour sauvegarder une scène dans un fichier dans le format décrit ci-dessus.

Exercice 2 : Plus d'objets

Q.2.1 Dans cette question, nous rajoutons les plans comme objets possibles à afficher (on rappelle qu'on n'a que les sphères pour l'instant). Un plan est défini par un point p_0 sur le plan et un vecteur normal [1] au plan $\vec{p}$. Écrire le type plane. Ainsi, tout point P sur le plan vérifie l'équation suivante :

$$\overrightarrow{p_0P} \cdot \vec{p} = 0$$

Intersection L'équation paramétrique d'un rayon (voir partie 1) est :

$$P = O + t\vec{v}$$

d'où $(O + t\vec{v} - p_0) \cdot \vec{p} = 0$ et donc :

$$t = \frac{(p_0 - O) \cdot \vec{p}}{\vec{v} \cdot \vec{p}}$$

Un rayon a donc une intersection avec le plan si le dénominateur est non nul, c'est-à-dire si $\vec{v} \cdot \vec{p} \neq 0$, en d'autres termes, si le rayon et le plan sont parallèles. Par ailleurs, il faut aussi vérifier que $t \geq 0$. Dans le cas contraire, cela signifie que le plan se trouve derrière la caméra.

Ajouter une fonction (ou une méthode) intersect pour le plan.

Q.2.2 Rajouter le triangle comme objet représentable dans notre lanceur de rayon. Le triangle est un objet qui est très utile quand il est agrégé avec d'autres triangles pour former des objets complexes. Un triangle est défini par trois points. Écrire son type.

Intersection Pour vérifier l'intersection entre un rayon et un triangle, nous allons utiliser l'algorithme de Möller-Trumbore. On suppose que le triangle a pour sommets, v_0, v_1, v_2. Le rayon a pour direction $\vec{d}$ et comme origine O.

On calcule le produit mixte :

$$m = (\vec{d} \wedge \overrightarrow{v_0v_2}) \cdot \overrightarrow{v_0v_1}$$

Si $m = 0$ alors il n'y a pas intersection.
Sinon, on calcule :

$$u = \frac{\overrightarrow{v_0O} \cdot (\vec{d} \wedge \overrightarrow{v_0v_2})}{m}$$

Si $u < 0$ ou $u > 1$ alors il n'y a pas intersection.
Sinon on calcule :

$$v = \frac{\vec{d} \cdot (\overrightarrow{v_0O} \wedge \overrightarrow{v_0v_1})}{m}$$

Si $v < 0$ ou $u + v > 1$, alors il n'y a pas intersection.
Sinon, il y a bien intersection et la distance à l'origine est :

$$t = \frac{\overrightarrow{v_0v_2} \cdot (\overrightarrow{v_0O} \wedge \overrightarrow{v_0v_1})}{m}$$

Écrire intersect pour un triangle.

Q.2.3 *Facultatif.* Les objets disponibles pour l'instant rendent difficile le rendu d'une scène complexe, avec des objets qui auraient des formes géométriques arbitraires, par

1. Perpendiculaire.

exemple issus d'un logiciel de modélisation 3D comme Blender. Rajouter une gestion basique du format *Wavefront Obj*. Une façon de faire est d'interfacer avec OCaml une bibliothèque C, *tinyobjloader-c* pour charger les .obj : `https://github.com/syoyo/tinyobjloader-c`

Une autre façon est de parser un sous-ensemble du format .obj, par exemple, seulement la définition des sommets et des faces. Le format .obj est un format textuel simple à parser, où chaque élément est défini ligne par ligne. Les faces d'un .obj sont triangulaires.

```
#Un commentaire
# Un sommet de cordonnées (2.3, 4.5, 10)
v 2.3 4.5 10
v 1.0 0.34 0.5
v 0.34 -0.3 -2.456
# Une face, composée des arêtes définies
# en premier, en deuxième, et en troisième
f 1 2 3
```

Vous devez pouvoir charger un sous-ensemble de ce format, écrire un type ou une classe pour le représenter, et ajouter une fonction (ou méthode) intersect qui vérifie l'intersection entre un tel objet et un rayon.

Q.2.4 *Facultatif.* Ajouter la gestion des plans, des triangles, et des Wavefront obj, dans notre format de description de scène.

Exercice 3 : Parcours d'arbre

Parmi les nombreuses optimisations existantes d'un lanceur de rayon, on trouve l'utilisation d'une structure visant à organiser les objets de la scène à intersecter. Nous allons appliquer pour cette question une version simple du kd-tree [2, 3, 4] pour optimiser le rendu d'une scène lorsque le nombre d'objet présent devient conséquent.

L'idée simple est illustrée sur l'image 1 et consiste à séparer l'espace d'une scène en k plans et ainsi entourer ses objets par des boites représentant les bornes min/max des coordonnées d'un objet. Ainsi, si un rayon n'intersecte pas cette boite alors le ou les objets contenus dans cette boite ne sont pas intersectés. Cette structure ajoute également une notion de distance et le premier objet ayant été intersecté est le plus proche de l'origine du rayon (les autres objets pouvant être intersectés après ne seront donc pas calculés).

Q.3.1 Avant de commencer, il faut définir quelques types et interfaces (.mli) pour pouvoir manipuler notre kd-tree sans contraindre les objets déjà définis dans votre projet. Remarquer qu'un kd−tree est un objet (ou *forme*)comme une sphère et doit donc présenter la même interface `objet`. L'écrire ou la compléter avec les fonctions :

- `intersect`, étant la fonction d'intersection codée dans la phase 1 du projet,
- `min`, retournant la valeur du point minimal de la forme
- `max`, retournant la valeur du point maximal de la forme

Ainsi, nous avons le point minimum et maximum d'une forme.

Q.3.2 Un kd-tree est un simple arbre binaire, son type est un objet et ses nœuds sont soit une feuille (une liste de objets), soit un nœud intermédiaire représenté par une boite limitant la zone contenue par ses deux sous arbres. Pour cela notre boite limitante doit avoir deux points (ses deux points extrêmes c1 et c2). Définissez les types `kd-tree` et `boitelimite`.

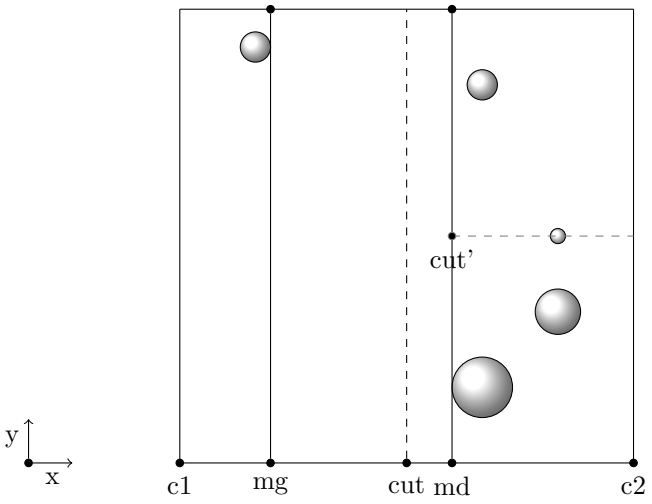

FIGURE 1 – Phase de construction d'un kd-tree (représentée par le premier point de découpe *cut* et les bornes du plan $c1 - c2$ projetées sur l'axe des x). La phase suivante est représentée uniquement par son point de découpe cut' et son médian.

Q.3.3 (construction) Pour construire un kd-tree à partir d'une scène (une liste de formes pour être précis), il faut considérer la notion de projection. En effet, chaque découpage est calculé sur un certain axe et changé pour l'étape suivante, ceci permet d'obtenir des plans alignés simplifiant les calculs et la structure de l'arbre. Pour ce projet nous allons projeter nos calculs sur la suite d'axes $x, y, z, x, y, z \ldots$ qui est la stratégie généralement utilisée. La construction d'un arbre ce fait donc suivant ces étapes :

- `decoupe`, pour un plan courant (c1-c2) le point médian est calculé selon l'axe courant (le point *cut* sur l'image 1)
- `répartition`, la liste des objets restants à trier est séparée en deux listes :
  - `lgauche`, la liste où le point minimum d'un objet est plus petit que le médian (*i.e.* les objets se trouvant à gauche du milieu du plan)
  - `ldroit`, la liste où le point maximum d'un objet est plus grand que le médian (*i.e.* les objets se trouvant à droite du milieu du plan)

  Il peut y avoir bien évidemment le même objet dans les deux listes si le médian coupe cet objet en deux, notez que cette étape calcule également `maxgauche` (mg) et `mindroit` (md) le point maximum des objets de gauche (sans que ce soit plus grand que le médian) et inversement le point minimum des objets de droite
- `boucle`, finalement la fonction se rappelle pour chaque côté si la profondeur k n'est pas atteinte et si la taille de la sous-liste de objets est supérieure à 1. L'appel incrémente la profondeur k, change l'axe de projection et s'applique sur une des sous listes avec la nouvelle boite limitante construit ainsi :
  - (c1, maxgauche) pour celle de gauche et
  - (mindroit, c2) pour celle de droite.

Q.3.4 (parcours) Pendant le lancement de rayon, le kd-tree va devoir être traversé selon l'origine du rayon et jusqu'à ce qu'une feuille de l'arbre soit rencontrée ou que la boite limite soit traversée. L'image 2 résume les différentes possibilités. Cette traversée à deux initialisions distinctes selon si l'origine

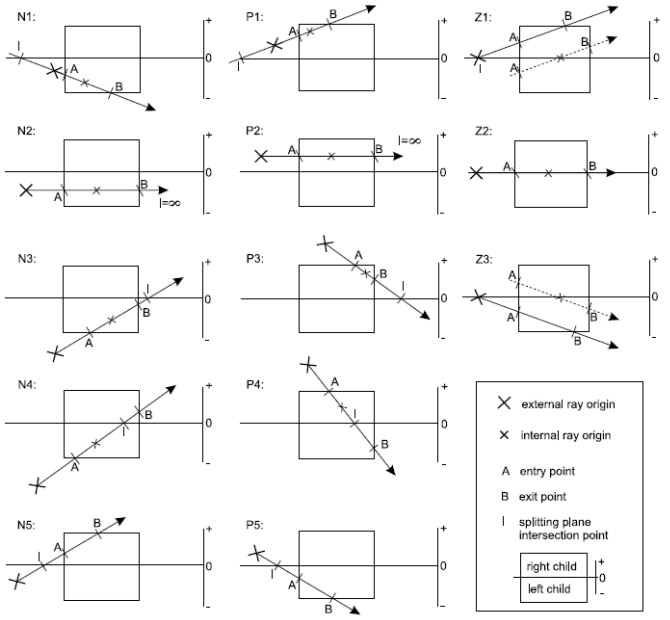

FIGURE 2 – Un rayon au travers d'une boite limitante.

du rayon est dans ou hors du plan. Les étapes d'une traversée sont les suivantes :

- le point d'entrée (A) est calculé et est soit l'origine (si le rayon part dans le plan) ou l'intersection du rayon et de la boite racine.
- selon si ce point d'entrée est plus petit ou plus grand que le médian d'une boite, le parcours entre dans les noeuds gauche ou droite du kd-tree. Ceci jusqu'à atteindre une feuille.
- Si cette feuille contient des objets alors l'intersection est calculée avec toutes ses objets.
- Si il y a une intersection (dans le plan courant) alors le parcours termine pour ce rayon et le reste de la fonction peut s'appliquer (illumination)
- Si il n'y a pas de objets, pas d'intersection ou que cette intersection n'est pas dans le plan courant, alors le point de sortie (B) est calculé.
- Le parcours continue avec comme point d'entrée le nouveau point B.
- Finalement si le point de point de sortie est calculé pour le plan racine alors le rayon n'intersecte aucun objet.

Q.3.5 Comparer le temp de rendu avec *kd-tree* et sans *kd-tree* pour la scène de la fin de la partie 1.

Exercice 4 : Plus de phénomènes physiques

Cette partie est *facultative*. Nous allons ajouter deux phénomènes physiques courants : la *réflexion* (miroir) et la *réfraction* (comme pour de l'eau).

Q.4.1 Chaque *matériau* possède un coefficient de réflexion ρ_r, qui est la quantité de lumière qu'il va renvoyer. Un matériau non réfléchissant a donc un coefficient valant 0. Augmenter le type material avec ρ_r.

Le rayon réfléchi $\vec{R}$ est renvoyé par rapport à la normale $\vec{N}$ au point incident, avec le même angle θ que le rayon incident $\vec{I}$, comme montré sur la Fig. 3.

Le rayon réfléchi $\vec{R}$ est donc donné par l'équation suivante :

$$\vec{R} = \vec{I} - 2(\vec{N} \cdot \vec{I})\vec{N}$$

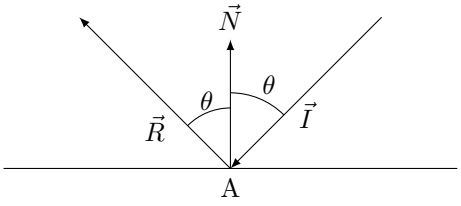

FIGURE 3 – Le rayon incident de direction $\vec{I}$ est réfléchi par rapport à la normale $\vec{N}$. Le rayon réfléchi est $\vec{R}$. Les angles incidents et réfléchis sont égaux.

Ce que l'on voit dans la surface réfléchissante dépend du point de vue : déplacez-vous devant un miroir ; vous ne verrez pas la même chose. On va donc envoyer un nouveau rayon récursivement ayant pour direction le rayon réfléchi $\vec{R}$, appelé aussi *rayon spéculaire*. La couleur du point incident est alors la couleur obtenue à partir de ce rayon réfléchi, multiplié par $\rho_r \leq 1$. En cas de deux miroirs face à face, le nombre d'appels récursifs est potentiellement infini.

A chaque réflexion, l'intensité de lumière réfléchie diminue, si $\rho_r < 1$. Dans ce cas, on peut stopper les appels récursifs quand la quantité est sous un certain ϵ. On peut aussi limiter à un nombre donné, 15 par exemple, le nombre d'appels récursifs. Il faut se rendre compte que plus on autorise des appels récursifs, plus le temps de rendu sera long.

Ajouter la réflexion à votre lanceur de rayons.

Q.4.2 Quand un rayon passe d'un milieu transparent à un autre, le rayon est dévié, comme montré sur la Fig. 4. La déviation dépend de *l'indice de réfraction* du milieu, η_r. Le verre a typiquement un indice de 1.5, l'air, de 1, et l'eau, de 1.3. Ajouter dans le type material l'indice de réfraction.

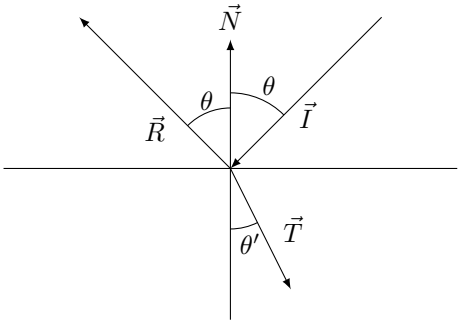

FIGURE 4 – Le rayon incident de direction $\vec{I}$ est réfléchi par rapport à la normale $\vec{N}$. Le rayon réfléchi est $\vec{R}$. Le rayon réfracté est $\vec{T}$. Les angles incidents et réfléchis sont égaux, alors que l'angle réfracté est θ' et suit les lois de Descartes.

Les angles incident et réfracté suivent la loi de Descartes :

$$\eta_i \sin\theta = \eta_r \sin\theta'$$

On en déduit $\vec{T}$. On pose $c_1 = \vec{N} \cdot \vec{I}$ et

$$c_1 = \sqrt{1 - (\frac{\eta_i}{\eta_r})^2 \sin^2(\theta)}$$

et on a :

$$\vec{T} = \frac{\eta_i}{\eta_r}\vec{I} - (\frac{\eta_i}{\eta_r}c_1 - c_2)\vec{N}$$

Si le terme sous la racine dans c_2 est négatif, on a une *réflexion totale*, et pas de *réfraction*.

Ajouter la réfraction au lanceur de rayons.

*

Références

[1] Pharr, Matt and Jakob, Wenzel and Humphreys, Greg, *Physically based rendering : From theory to implementation*. Morgan Kaufmann, 3rd Edition, 2016

[2] `https://en.wikipedia.org/wiki/K-d_tree`

[3] `https://www.scratchapixel.com/lessons/advanced-rendering/introduction-acceleration-structure/bounding-volume-hierarchy-BVH-part1`

[4] `https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html`