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

Objectifs

1. Principe du lancer de rayons
2. Affichage graphique ou sauvegarde d'image

Le lancer de rayons (*raytracing*) est une technique de rendu d'images de synthèses fondée sur les lois de l'optique. Elle consiste à définir une scène fictive formée d'un ensemble d'objets et de sources lumineuses. L'objectif est de synthétiser l'image que capturerait une caméra placée en un point de cette scène. Cette image est formée par l'ensemble des rayons qui se propagent dans la scène avant d'atteindre la caméra. En simulant le parcours inverse de ces rayons, il est possible de déterminer la valeur prise par chaque pixel de la caméra. Pour cela, il faut trouver les intersections et calculer les interactions des rayons simulés sortant de la caméra avec les différents objets et sources lumineuses de la scène.

Cette technique est une bonne approximation de l'équation de rendu (Eq. 1) qui exprime la luminance énergétique (*radiance*) L_0 émise en un point x de l'espace et dans une direction $\vec{w}$. Elle s'exprime comme la somme de la lumière émise L_e et de la lumière réfléchie qui est elle-même la somme des lumières L_i venues de toutes les directions, corrigées par la capacité de réflexion de la surface et l'angle incident. Elle permet donc d'obtenir des images particulièrement photoréalistes (comme celle de l'image 1), mais est coûteuse en temps de calcul, au contraire d'une méthode comme la rastérisation. C'est pourquoi le lancer de rayons est surtout utilisé pour les films en image de synthèse, alors que les applications qui ont des contraintes temps réelles, comme les jeux vidéos, utilisent plutôt la rastérisation. Le lancer de rayons permet de représenter plus facilement les ombres, les réflexions, les réfractions, les diffractions, et les diffusions de la lumière, ou les caustiques.

$$L_0(x,\vec{w}) = L_e(x,\vec{w}) + \int_\Omega f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}'\cdot\vec{n})d\vec{w}' \quad (1)$$

FIGURE 1 – Exemple wikipédien d'un rendu de lanceur de rayons.

Dans le cadre de ce projet, nous souhaitons créer un lanceur de rayons qui puisse gérer des scènes complexes en nombre d'objets. La première partie du projet consiste à créer une version de base qui gère des phénomènes physiques simples pour un petit nombre d'objets dans la scène, et affiche ou enregistre l'image de rendu.

Le projet pourra être programmé en OCaml, F# ou en Swift. Il devra être soigneusement documenté et testé. Une attention particulière sera accordée à l'architecture du code. Il faudra fournir un Makefile où `make` compilera le projet et `./raytracer-demo` (sans arguments) lancera une démo du programme. Vous pouvez supposer que nous avons installé *ocamlbuild*. Votre programme doit être multi-plateforme, ou au moins fonctionner sous Mac et Linux.

Exercice 1 : Manipulation de vecteurs

La modélisation de la scène à représenter nécessite la manipulation de points et de vecteurs dans un espace en trois dimensions.

Q.1.1 Définissez des types points et vecteurs. Vous pouvez utiliser des tableaux, ou des *records*.

Q.1.2 Définissez les opérateurs suivants entre vecteurs : addition, soustraction, multiplication par un scalaire, produit scalaire, produit vectoriel, norme et normalisation. N'hésitez pas à définir ces opérateurs comme opérateurs infixes pour les plus utilisés.

Exercice 2 : Lancer de rayons : visibilité

Le rendu d'une scène est calculé depuis un certain point de cette scène, appelé caméra. Pour commencer un rendu, le lanceur de rayons calcule les objets visibles depuis cette caméra. On dit qu'un objet est visible par un pixel de la caméra s'il n'y a aucun obstacle le long du segment qui relie l'objet au pixel. Un rayon est donc la donnée d'une origine et d'une direction. Les rayons envoyés depuis la caméra sont appelés *rayons primaires*.

Q.2.1 Définir le type d'un rayon, formé d'une origine O (le point depuis lequel le rayon est envoyé) et d'une direction $\vec{d}$, comme sur la Figure 2. La direction sera normalisée (norme de $\vec{d}$ égale à 1). Une équation paramétrique du rayon est donc $P = O + t\,\vec{d}$ où $t \geq 0$ et P est un point sur le rayon. Un rayon est donc une *demi-droite*.

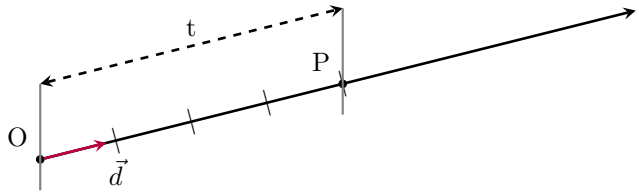

FIGURE 2 – Schéma d'un rayon d'origine O, de direction $\vec{d}$, avec un point P tel que $P = O + t\,\vec{d}$

Q.2.2 Dans cette question, nous définissons la caméra (l'appareil-photo, ou l'œil) qui va prendre une image de la scène 3D. Il existe de nombreuses variétés de caméras mais nous allons considérer un modèle simple : une caméra ponctuelle. Le passage du monde 3D de la scène au monde 2D de l'image avec une telle caméra est appelée *perspective linéaire*. La caméra est décrite entièrement par son origine, les dimensions de l'écran, le champ de vision (en radian, l'angle entre les côtés gauche et droit de la caméra) et le point de visée, c'est-à-dire le point situé au centre de l'écran. Définissez le type d'une telle caméra.

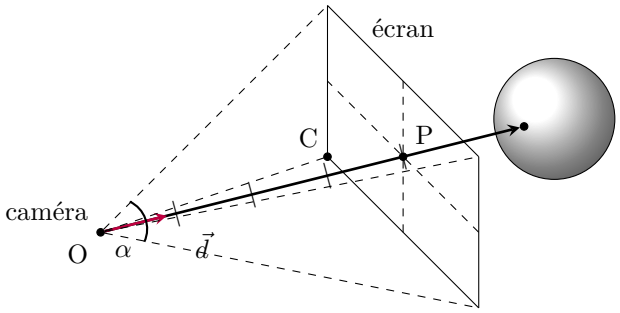

FIGURE 3 – Schéma d'une caméra avec un champ de vision α, un écran de rendu et une sphère. C est le point en bas à gauche de l'écran et P le point de visée de l'écran pour le rayon de direction $\vec{d}$.

Q.2.3 Chaque rayon est envoyé depuis la caméra et passe par le milieu de chacun des pixels de l'image. Il va donc falloir convertir des coordonnées de l'espace discrétisé et en deux dimensions de l'image, à celui du monde en 3 dimensions. Écrivez la fonction qui lance les rayons primaires, et qui prend une caméra en entrée. A ce stade, la fonction ne fait rien d'autre que de calculer successivement pour chaque pixel de l'image le rayon sortant. Ces rayons ne sont pour l'instant pas utilisés, mais la fonction sera complétée ultérieurement pour décrire leur manipulation.

Une façon de faire (voir Fig. 3) : si la caméra a pour origine O, pour point de visée P, comme champ de vision α et que l'on veut une image de dimensions (w, h), alors la direction de la caméra est $\overrightarrow{OP}$ et les vecteurs $\vec{u}, \vec{v}$ qui déterminent les directions de l'écran (si on considère une direction verticale arbitraire $\overrightarrow{\text{up}}$ de coordonnées $(0, 1, 0)$) sont :

$$\vec{u} = \overrightarrow{\text{up}} \wedge \overrightarrow{OP}$$
$$\vec{v} = \overrightarrow{OP} \wedge \vec{u}$$

Définissons quelques paramètres intermédiaires :

$$\text{image\_ratio} = \frac{h}{w}$$
$$\text{half\_width} = \text{image\_ratio} \times \tan(\frac{\alpha}{2})$$
$$\text{half\_height} = \tan(\frac{\alpha}{2})$$

Ensuite, nous calculons le coin en bas à gauche de l'écran C, et deux vecteurs qui permettront de parcourir l'écran $\vec{x}, \vec{y}$.

$$C = P - \text{half\_width} \cdot \vec{u} - \text{half\_height} \cdot \vec{v}$$
$$\vec{x} = \frac{2 \times \text{half\_width}}{w} \cdot \vec{u}$$
$$\vec{y} = \frac{2 \times \text{half\_height}}{h} \cdot v$$

On en déduit la direction $\overrightarrow{OQ}$ dans l'espace pour le point qui correspond aux coordonnées (x_e, y_e) sur l'écran :

$$Q = C + x_e \cdot \vec{x} + y_e \cdot \vec{y}$$

Q.2.4 Définir le type d'une sphère.

Q.2.5 Nous cherchons à savoir si un rayon intersecte une sphère présente dans la scène. L'équation implicite d'une sphère de centre $C = (x_c, y_c, z_c)$ est donnée par $(x-x_c)^2+(y-y_c)^2+(z-z_c)^2 = r^2$. Le rayon a pour équation paramétrique $P = O + t\ \vec{d}$ c'est-à-dire :

$$\begin{cases} x(t) = x_o + t \times x_d \\ y(t) = y_o + t \times y_d \\ z(t) = z_o + t \times z_d \end{cases}$$

Son intersection avec un rayon peut donc être déterminée en développant puis résolvant l'équation du second degré suivant le paramètre t :

$$(x(t) - x_c)^2 + (y(t) - y_c)^2 + (z(t) - z_c)^2 = r^2$$

Écrire une fonction `intersect` qui renvoie la distance à la caméra en cas d'intersection, ou bien $+\infty$ s'il n'y a pas d'intersection.

Q.2.6 Modifier la fonction qui lance les rayons de la caméra pour qu'elle prenne une liste de sphères en paramètres, et qu'elle utilise `intersect`. En cas de plusieurs intersections pour un même rayon, on ne gardera que l'intersection avec la sphère la plus proche.

Exercice 3 : Affichage

Q.3.1 Affichez à l'écran votre rendu. On affichera en blanc les pixels *visibles*, c'est-à-dire ceux pour lesquels leur rayon a rencontré un objet, et en noir ou en bleu ciel les autres (voir Fig. 4).

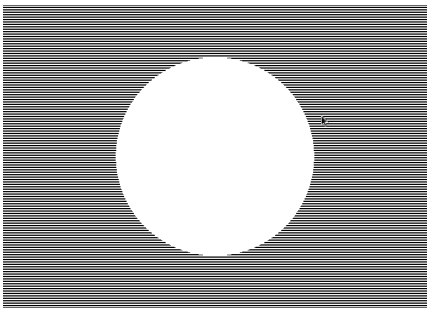

FIGURE 4 – Affichage d'une sphère de centre $(0, 0, 1)$ et de rayon 0.4 avec une caméra placée à l'origine, qui regarde vers le point $(0, 0, 5)$ avec un champ de vision de $\frac{8\pi}{3}$ et une image de taille 700×500. Le fond noir a été remplacé par des hachures pour sauver l'environnement...

En OCaml, on pourra utiliser la bibliothèque Graphics, qui est fourni par défaut. En F#, vous pouvez utiliser les composants graphiques .Net ou GTK# ou même faire de *l'interopérabilité* et faire l'interface graphique en C#. En Swift, vous pouvez utiliser des composants OS X ou bien interfacer avec une librairie C externe. Vous pouvez aussi générer une image sur le disque : un format simple, textuel, pour représenter des images, est le format *ppm*.

Exercice 4 : Lancer de rayons : illumination

Une fois la visibilité de l'objet décidée, il faut calculer la couleur du pixel, en propageant le rayon vers les autres objets et les sources de lumière. Ces nouveaux rayons sont appelés *rayons secondaires*. Ils peuvent être des rayons d'ombre, des rayons de réflexion, des rayons de réfraction etc. On commence ici par un modèle simple, le modèle de Blinn-Phong.

Le modèle de Blinn-Phong décompose la réflexion d'un point entre trois composantes (voir Fig. 5) :

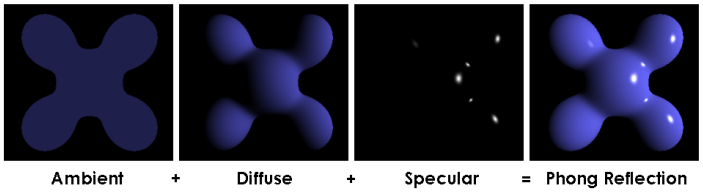

FIGURE 5 – Les trois composantes du modèle de Blinn-Phong (source : Wikipédia)

1. **composante ambiante** : I_a,
2. **composante diffuse** : I_d,
3. **composante spéculaire** : I_s.

La composante ambiante représente la luminosité moyenne qui règne dans la scène. La composante diffuse correspond à la réflexion de la lumière sur une surface irrégulière [1] et la composante spéculaire, à la réflexion sur une surface brillante : cela correspond aux taches de lumière sur un objet.

L'illumination totale en un point de la scène est la somme des trois contributions :

$$I = I_a + I_d + I_s$$

I_a (respectivement I_d et I_s) s'exprime en fonction d'un coefficient [2] noté $k_a \in [0, 1]$ (respectivement k_d et k_s). Les 3 composantes dépendent également d'un paramètre β qui correspond à la brillance du matériau [3].

Le but des questions suivantes et d'implémenter le calcul, détaillé dans la suite de l'énoncé, des trois composantes d'illumination.

Q.4.1 Définir un type pour représenter une couleur, puis un type pour représenter le matériau avec lequel est fait un objet. Un *matériau* sera composé d'une couleur (r, g, b), des trois coefficients k_a, k_d et k_s à valeurs dans $[0, 1]$ et de β. On peut déduire de ces valeurs les paramètres du modèle de Blinn-Phong par couleur. Par exemple, $k_d^{\text{rouge}} = k_d \times \frac{r}{255}$ si vous représentez les couleurs par des valeurs entre 0 et 255.

Q.4.2 Nous allons ici modéliser des sources primaires de lumières simples : des lumières distantes, comme le soleil. On considère que les rayons émis par ces sources sont tous envoyés dans la même direction. Ces sources ont aussi une couleur et une intensité qui est la même en tout point de l'espace. Définir le type d'une lumière distante qui contient donc une direction, une couleur et une intensité. Modifiez la fonction principale de lancer de rayons pour faire en sorte que l'on puisse rajouter une lumière dans la scène.

Vous pouvez aussi ajouter des sources primaires de lumière ponctuelles et anisotropes (c'est-à-dire qui diffusent la lumière dans toutes les directions). Pour celles-ci, l'intensité décroît en $4\pi d^2$ où d est la distance entre le point à illuminer et la source ponctuelle. Attention, choisissez une très grande intensité de départ, sinon, vous risquez d'obtenir une image toute noire.

Q.4.3 L'éclairage ambiant est calculé comme une contribution de toutes les lampes de la scène. Vous pouvez commencer par un modèle très simple : la composante ambiante en un point d'un objet est le produit de son coefficient k_a^{objet} par une constante i_a commune à toute la scène. Soit :

$$I_a = k_a^{\text{objet}} \times i_a$$

Attention, il faut bien garder à l'esprit qu'ici I_a est un vecteur dont chaque composante est associée à une couleur. $i_a = 0.2$ fonctionne bien.

Écrire une fonction `illuminate` qui calcule les trois composantes (r, g, b) d'un pixel en prenant en compte l'éclairage ambiant.

Q.4.4 Modifier la fonction `intersect` pour qu'elle renvoie aussi la normale à la sphère au point incident. Nous aurons besoin de la normale pour calculer les composantes diffuses et spéculaires. La figure 6 décrit les vecteurs nécessaires au calcul de l'illumination dans le modèle de Blinn-Phong, ce qui sera nécessaire également pour les questions suivantes.

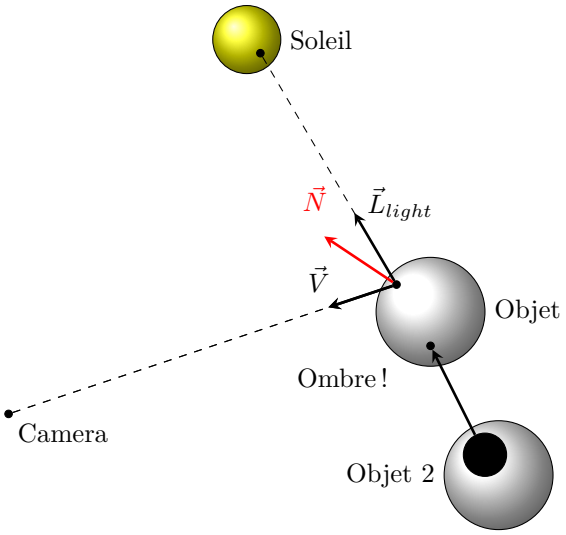

FIGURE 6 – Schéma d'une sphère avec les vecteurs $\vec{L}_{\text{light}}$, $\vec{N}$, $\vec{V}$ nécessaires pour le modèle de Blinn-Phong, ainsi que pour les rayons d'ombre. $\vec{H}$ a été omis pour des raisons esthétiques.

Q.4.5 L'éclairage diffus (ou lambertien) I_d est obtenu, dans le cas d'une source distante, selon la loi :

$$I_d = k_d \times (\vec{L}_{\text{light}} \cdot \vec{N}) \times i$$

où $\vec{L}_{\text{light}}$ est le vecteur normalisé du point d'intersection de la surface à la source de lumière, $\vec{N}$ la normale à la surface au point d'intersection, et i l'intensité de la lumière incidente. On rappelle que pour une source non distante, il faudrait prendre en compte la décroissance de l'intensité en $4\pi d^2$.

Modifiez la fonction `illuminate` pour qu'elle prenne aussi en compte l'éclairage diffus.

Q.4.6 La relation de Blinn-Phong décrit la composante spéculaire de la façon suivante :

$$I_s = k_s \times (\vec{H} \cdot \vec{N})^\beta \times i$$

avec $\vec{H} = \frac{\vec{L}_{\text{light}} + \vec{V}}{\|(\vec{L}_{\text{light}} + \vec{V})\|}$ et $\vec{V}$ la direction vers l'appareil-photo. $\vec{H}$ est appelé le *demi-vecteur*. Modifiez la fonction `illuminate` pour qu'elle prenne aussi en compte l'éclairage spéculaire.

Q.4.7 Des ombres apparaissent quand un obstacle se trouve sur le trajet d'une source de lumière. Réutilisez la fonction

1. On parle de *microfacettes*. La lumière se diffuse sur la sphère.
2. Le coefficient peut être rendu dépendant de la fréquence de la lumière, et donc de la couleur...
3. Plus β est grand, plus la tache de lumière sera petite.

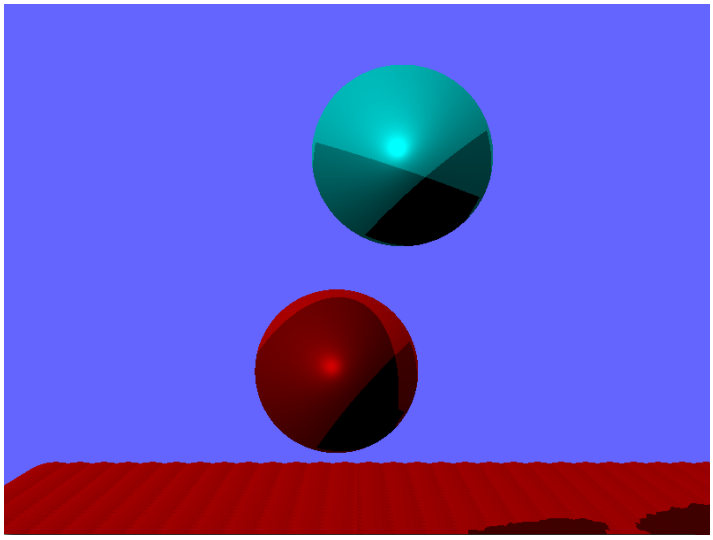

FIGURE 7 – Un rendu d'une image basique avec deux sphères et un carré de 28 sphères de côté. L'éclairage est donné par deux lampes distantes.

d'intersection entre un rayon et une sphère pour savoir s'il y a un obstacle. Le nouveau rayon part du point d'intersection courant et a pour direction la direction opposée à la propagation de la lumière. S'il y a intersection du rayon de lumière avec un obstacle, alors le point à illuminer sera noir. Sinon, nous utilisons le modèle de Blinn-Phong pour illuminer le point.

Exercice 5 : Scène démo

Q.5.1 Votre programme `./raytracer-demo` doit faire le rendu d'au moins deux sphères, avec une lampe, de façon à voir au moins une ombre d'une sphère sur l'autre. Un exemple de rendu se trouve en Figure 7.

Q.5.2 Écrivez une fonction permettant de faire le rendu d'un carré de sphères de rayons r accolées centré en un point donné en argument, avec $2n$ sphères de rayon r par côté. Vous pouvez l'intégrer à la scène de démo.

Facultatif – bonus

Q.5.3 Il s'agit ici de gérer plusieurs sources de lumières. Chaque source de lumière apporte linéairement son éclairage à la scène. Cela signifie qu'il suffit d'additionner les contributions de toutes les lumières au moment du calcul de l'illumination.

Vous devriez retrouver le même résultat en rendant plusieurs images séparément pour chaque lampe puis en superposant ces images, par exemple avec Photoshop ou The Gimp.

Q.5.4 Vous pouvez voir que les objets de la scène sont crénelés ; cela se voit encore mieux en zoomant sur votre rendu. Nous pouvons développer une forme d'anti-crénelage simple, par sur-échantillonnage. Pour chaque pixel de l'image, calculez les couleurs de $n > 1$ pixels puis moyennez.

Ce qui vous attend dans la partie 2 Testez votre carré de sphères avec 500 sphères de côté. Le rendu prend beaucoup de temps ! Pourquoi ? Dans la partie 2, nous verrons des techniques pour améliorer les performances, par exemple, l'utilisation de structures hiérarchiques, ou la parallélisation des calculs.

Nous améliorerons aussi le modèle physique avec les réflexions par exemple et nous ferons en sorte de représenter des scènes avec des objets plus complexes.

*

Références

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