ENS Rennes
L3 Informatique, parcours R & I
Cours de programmation, 1er semestre 2013–2014

—

Projet 3

Luc Bougé, Pierre Karpman*

5 novembre 2013

Présentation du devoir

Ce devoir est à faire par groupes de 2 ± 1. La soutenance de ce travail aura lieu le

mardi 26 novembre 2013à ***13h15****, sur le créneau du TD.*

Le rapport est à envoyer à pierre.karpman@gmail.com pour le

dimanche 1 décembre 2013à ***18h00****.*

Une pénalité d'un point par heure sera appliquée en cas de retard pour le rendu du rapport.

Modalités de la soutenance. La soutenance dure 12 minutes, suivies de 5 minutes de questions. Il est souhaité que le groupe utilise un support visuel permettant de rapidement résumer le travail effectué (comprenant typiquement des captures d'écrans de résultats d'exécutions). Les démonstrations de programmes en direct sont aussi appréciées (quoique non obligatoires), mais faites attention à bien gérer votre temps dans ce cas.

Contenu du rendu. Le rendu se fait sous la forme d'une archive contenant :

- Un rapport rédigé (en format .pdf) d'environ 5 pages expliquant et motivant les choix de conception (non triviaux) que vous avez fait, ainsi que les extensions que vous avez ajoutées au sujet de base (cf. §Évaluation).
- Les documents utilisés pour la soutenance.
- Le code source de vos programmes (si possible, proposez deux versions pour chaque programme : une version de base répondant strictement aux questions et une version complète avec toutes vos extensions).

*D'après un document original de Matthieu Dorier.

– Le fichier *Makefile* utilisé pour compiler le (les) programme(s).
– Un fichier *README* donnant les commandes nécessaires à la compilation du (des) programme(s) (et éventuellement des instructions d'utilisation).

L'archive sera nommée d'après la convention *Nom1Nom2...*, les noms de famille étant ordonnés alphabétiquement (par ex. *Calvin*, *CalvinHobbes*, *CalvinDerkinsHobbes*). Elle sera envoyée par courrier électronique, l'objet duquel utilisera la même convention que ci-dessus, préfixée de *Prog1P3* (par ex. *Prog1P3CalvinHobbes*).

Remarque. *La qualité de la rédaction est très importante. La notation prend en compte le soin, la structure et l'orthographe*[1] *du rapport. L'aspect « esthétique » de vos programmes est aussi important. Vous ferez donc attention à écrire un code commenté, lisible, bien indenté, au choix de vos noms de variables, etc.*

Évaluation

Le sujet de projet tel que décrit dans ce document consiste en le *minimum nécessaire* pour obtenir une note passable. Il vous est fortement conseillé de réaliser des extensions aux programmes demandés. Le sujet en donne quelques exemples, mais vous êtes libres d'en choisir d'autres. Prenez cependant garde à ne pas être trop ambitieux ; une extension modeste mais bien réalisée et fonctionnant correctement sera plus valorisée qu'une qui serait plus complexe mais implémentée de façon brouillonne et imparfaite (mais que ce conseil ne soit pas non plus pris comme un encouragement à la paresse) !

Aide

Vous pouvez poser toutes vos questions à l'adresse pierre.karpman@gmail.com.

1 Lancer de rayons

1.1 Introduction

Le lancer de rayons (*ray tracing*) est une technique de rendu 3D basée sur le principe du retour inverse des rayons. Elle consiste à lancer des rayons de lumière depuis l'objectif de la caméra, de détecter les objets touchés par ces rayons et de déterminer la couleur des pixels à l'écran en propageant des rayons vers les sources de lumière.

Cette technique a l'avantage d'utiliser une définition mathématique des objets plutôt qu'une représentation sous forme de polygones. En effet, une définition mathématique permet de calculer plus simplement le point d'intersection entre un rayon et un objet. De plus, comme elle imite un phénomène physique, le rendu est en général de meilleure qualité que les rendus utilisant des projections de polygones sur l'écran.

L'inconvénient de cette technique est sa complexité : pour chaque rayon lancé (autant que de pixels), il faut tester l'intersection avec tous les objets de la scène, puis propager des

1. Pensez par exemple à utiliser un correcteur orthographique tel qu'*aspell* si vous rédigez votre rapport avec LaTeX.

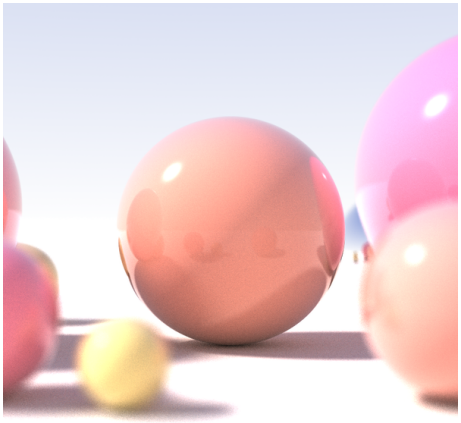

FIGURE 1 – Exemple d'image générée par lancer de rayons

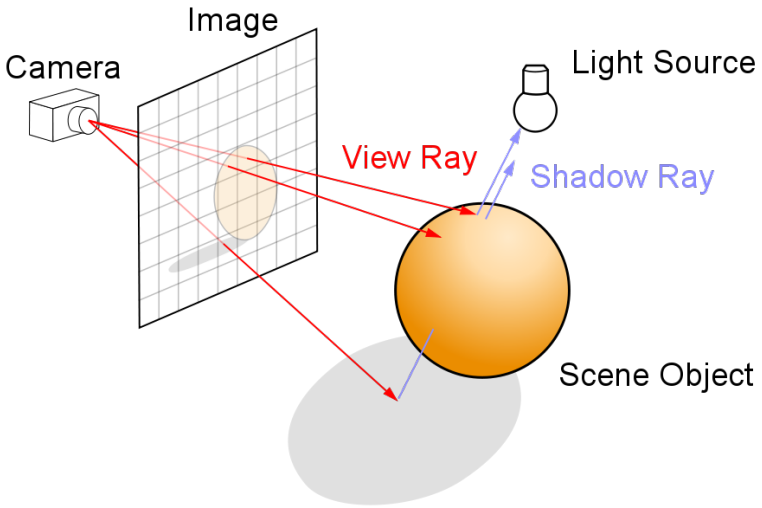

FIGURE 2 – Principe du lancer de rayons.

rayons vers toutes les sources lumineuses, ainsi que des rayons correspondant à la réflexion (pour les surfaces réfléchissantes) et la réfraction (pour les milieux transparents).

Le but de ce projet est de programmer un petit moteur de lancer de rayons en C++, qui ne gérera que des sphères pour l'instant. Ce projet sera la base du projet suivant : tâchez donc de programmer proprement et de comprendre ce que vous faites !

1.2 Algorithme et formules

L'algorithme du lancé de rayons fonctionne de la manière suivante (Figure 2) : pour tout pixel de coordonnées (x, y) de l'image à générer, on crée un rayon partant de l'objectif de la caméra et passant par ce pixel. On cherche alors le première objet intersecté par ce rayon. En termes mathématiques, cela revient à trouver d'abord l'ensemble des objets potentiellement intersectés par le rayon, puis de prendre le plus proche. Dans le cas particulier d'une sphère

de centre C : (x_C, y_C, z_C) et de rayon R, le rayon d'origine A : (x_A, y_A, z_A) et de vecteur directeur unitaire $\vec{u}$: $(x_{\vec{u}}, y_{\vec{u}}, z_{\vec{u}})$ intersecte la sphère si et seulement s'il existe une solution en t à l'équation suivante :

$$||\vec{AC} - t\vec{u}||^2 = R^2$$

Si ce polynôme n'a qu'une seule racine, alors le rayon est tangent à la sphère et on considérera qu'il ne l'intersecte pas. S'il a deux racines, la plus petite racine positive correspond au premier point d'intersection et t correspond à la distance entre l'origine du rayon et le point d'intersection (si les racines sont toutes les deux négatives, la sphère est "derrière" le rayon). Ce point d'intersection est alors I : $(x_A + t.x_{\vec{u}}, y_A + t.y_{\vec{u}}, z_A + t.z_{\vec{u}})$. On peut aisément trouver la sphère la plus proche (en triant les sphères intersectées par la distance du point d'intersection à l'origine du rayon), et calculer le point d'intersection de la plus proche.

Il faut maintenant propager ce rayon du point I vers la source de lumière, positionnée en un point L. Pour cela, on lance un rayon depuis le point I, en direction de L, et on vérifie les deux propriétés suivantes :

1. Le rayon n'intersecte aucune autre sphère (sinon cette sphère cache la lumière) ;
2. Le produit scalaire entre le vecteur dirigeant le rayon et la normale à la sphère au point d'intersection est strictement positif.

La couleur du pixel est alors calculée en fonction de la couleur de la sphère, de la couleur de la lumière et éventuellement d'autres paramètres comme le produit scalaire entre la normale à la sphère et les rayons, ou encore la distance entre le point d'intersection et la source de lumière. A vous de tester différentes méthodes pour calculer cette couleur.

2 Classes et fonctions fournies

Un certain nombre de classes et structures vous sont fournies. Vous pouvez les récupérer depuis le dépôt SVN[2]. Vous êtes invités à lire les commentaires de ces fichiers pour comprendre comment les utiliser. Il est interdit de modifier ces classes. Vous noterez que toutes ces classes sont définies dans le *namespace* rt (pour *"ray tracing"*).

rt::vector : définie dans *vector.hpp* et *vector.cpp*, cette classe représente un vecteur à trois dimensions sous la forme de ses trois composantes **double** x, y et z. Les opérations arithmétiques ont été redéfinies pour cette classe. Ainsi, il est possible d'additionner ou soustraire des vecteurs, de les multiplier par un scalaire **double** : `s*v` ou `v*s`, d'effectuer un produit vectoriel : `u^v`, ou encore un produit scalaire : `u|v`. Attention à ne pas confondre cette classe rt::vector avec la classe std::vector de la STL, le *namespace* est là pour faire la différence !

Les classes rt::color, rt::image et rt::screen ont déjà été vues lors des TP précédents.

3 Travail à faire

À l'aide des classes fournies et des formules données, réalisez un mini-moteur de lancer de rayons pouvant gérer plusieurs sphères et une ou même plusieurs sources de lumières.

2. Qui sera bientôt mis en place. En attendant, vous trouverez une archive sur *Piazza*.

Vous êtes libres de définir les classes et méthodes qui vous semblent judicieuses. On pourra par exemple définir une classe représentant un rayon, une source de lumière, une sphère, etc.

La présentation et le rapport devront mettre en évidence :

1. vos choix en terme de programmation C++ (choix des classes, des méthodes, visibilité de celles-ci, choix des passages par référence, par copie ou par pointeur, etc.) ;
2. les algorithmes utilisés avec leurs spécificités ;
3. les difficultés rencontrées, ainsi que les résultats obtenus.

Il est (comme toujours !) conseillé de procéder par étapes.

Étape 1. Placez une sphère dans la scène, et produisez une image en noir et blanc indiquant simplement si un rayon correspondant à un pixel a touché la sphère ou non. Cela permettra de vérifier que la sphère est bien placée par rapport à la caméra.

Étape 2. Ajoutez un premier calcul de la couleur uniquement en fonction de la couleur de la sphère.

Étape 3. Ajoutez la gestion d'une source de lumière. Revoyez le calcul de la couleur en prenant en compte la source de lumière.

La précision limité du type ***double*** *fait qu'un point n'est jamais vraiment* sur *une sphère. Il est soit légèrement à l'intérieur, soit légèrement à l'extérieur, ce qui peut provoquer des artefacts de rendu. A vous de trouver une manière de gérer ce problème, ô futurs chercheurs !*

4 Ouverture

Vous êtes libres d'améliorer votre moteur comme bon vous semble, cependant il vous est demandé de séparer deux ensembles de fichiers sources : les sources d'un moteur suivant strictement ce sujet, et les sources contenant vos améliorations. Le projet le mieux programmé (sans prise en compte des possibles améliorations), c'est-à-dire ayant fait les meilleurs choix de classes, proposant les implémentations les plus claires des algorithmes et donnant la meilleur documentation du code, servira de base pour tout le monde lors du projet suivant !