Rapport du projet C++ : Lancer de rayons

Mathieu MARI Xavier MONTILLET

1er décembre 2013

Table des matières

1 Introduction

Dans ce nouveau projet, nous nous sommes intéressé à la technique du lancer de rayons et nous l'avons implémenter en C++. Cette technique permet de générer des images 3D à partir de la déscription des objets présent dans la scène ainsi que de la position de la caméra. Elle a son utilité dans les logiciels d'images de synthèse, la création de jeux vidéos, la simulation numérique...

2 Implémentation en C++

2.1 Principe général

Afin de générer l'image 3D d'une scène, il faut deja définir quels sont les objets qui composent la scène, c'est dire définir quel est leur type (sphère, plan, ...), quelle est leur position, leur couleur, leur texture.... Ensuite, il faut choisir la posistion des sources lumineuses puis leur couleur, leur intensité...Enfin, il faut positionner la (ou les) caméra(s) qui observe(nt) la scène. Pour cela, on choisit un point qui sera la position de l'objectif, puis la position de l'écran sur lequel l'image sera projetée.

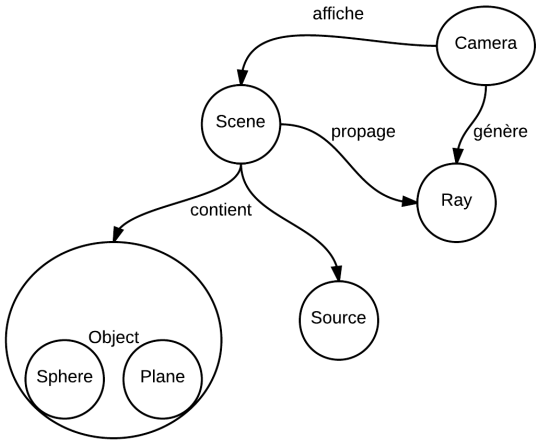
Fabriquer une image, c'est calculer pour chaque pixel sa couleur. Dans cette optique, le principe de la technique du lancer de rayons consiste à créer une demi-droite ayant pour origine l'objectif de la caméra et passant par le un pixel de l'écran. On regarde quel objet de la scène est intersecté en premier par la demi-droite. On calcule ensuite la couleur du pixel en regardant les sources qui éclairent le point et la couleur de l'objet. Il est également possible de prendre en compte les réflexions, réfractions etc. en lançant de nouveaux rayons à partir du point et en renvoyant une moyenne pondérée de la couleur de l'objet et des couleurs retournées par le rayon réfléchi, le rayon réfracté, etc.

2.2 Les différentes classes utilisées

Afin de représenter les différents éléments qui seront nécessaires à la technique du lancer de rayons et leur dépendance mutuelle, nous allons utiliser les classes C++.

2.2.1 Dépendances des classes

Le schéma de dépendance des classes est le suivant :



— La classe *Scene* est la classe principale. Elle contient des objets de la classe *Object* qui a deux sous-classes : *Sphere* et *Plane*.
— La classe *Scene* contient également des sources lumineuses de la classe *Source*.
— La caméra, représentée par la classe *Camera* est indépendante de la classe *Scene*. Nous aurions pu l'ajouter dans la *Scene* mais nous avons fait le choix de la garder indépendante parce que cela nous semblait plus logique et permet de rajouter simplement une deuxième *Camera*, par exemple pour afficher en 3D.

Toutes ces classes interagissent entre elles grâce à des rayons de la classe *Ray* : la *Camera* fabrique des rayons qui vont se propager dans la *Scene* afin de déterminer la couleur de chaque pixel.

D'autres classes nous sont utiles dans la réalisation de l'algorithme de lancer de rayons.

2.2.2 La classe *Light*

Cette classe permet de représenter une lumière. Elle possède trois attributs : une composante *red*, une composante *green*, et une composante *blue*. La différence avec la classe *Color* est que les composantes ne sont pas comprisent entre 0 et 255 mais entre 0 et $+\infty$ (non compris pour éviter des problèmes au niveau de la multiplication par un scalaire). Ceci permet d'additionner des lumières en additionnant simplement composante par composante.

Nous avons codé une fonction permettant de transformer une lumière en une couleur (l'image renvoyée à la fin ne comprend que des couleurs ayant des composantes comprises entre 0 et 255. Pour cela nous utilisons la fonction $x \to 255 \tanh(x)$.

2.2.3 Ls classes *ObjectTexture* et *PointTexture*

La couleur d'un pixel dépend de la texture de l'objet visé. En effet, l'effet visuel ne sera pas le même suivant si l'objet est brillant, mate, transparent.... La classe *PointTexture* permet donc de représenter ces différentes caractéristiques et d'en ajouter sans devoir redéfinir tous les constructeurs des classes héritant de *Object*. Elle ne contient pour l'instant que la couleur mais il est aisé de la modifier pour ajouter de nouvelles caractéristiques.

La classe *ObjectTexture* est une classe associée à un *Object* qui renvoie une *PointTexture* pour chaque *Point*. Nous en avons deux utilisations concrètes : *UniformObjectTexture* qui renvoie la même *PointTexture* pour tous les points et *RadialObjectTexture* pour lequel la couleur dépend de la distance à une origine.

2.2.4 Les classes auxiliaires

Les classes *Point*, *Vector* et *UnitVector* : Ces classes permettent de distinguer d'un point de vue logique des objets ayant la même représentation physique. Il n'est par exemple pas logique d'additionner deux points. Nous définissons donc des fonctions définissant les opérations entre la classe *Point* et la classe *vector*.

La classe *Option* : Lorsque nous cherchons le premier objet qu'un rayon va rencontrer, on demande à chaque objet si le rayon va l'atteindre et si oui, quelle distance le rayon doit parcourir. Le type option permet de demander directement la distance et tout en laissant le choix à l'objet de répondre que le rayon ne l'atteindra jamais.

3 Comment renvoyer la couleur d'un pixel ?

Afin de générer l'image finale, la caméra envoie pour chaque pixel un rayon (de la classe *Ray*) partant de l'objectif et passant par le pixel correspondant. Les programmes que nous avons codé consistent à associer à chacun de ces rayons une couleur.

3.1 Première étape : Trouver le premier objet de la scène intersécté par le rayon

Afin de touver le premier objet de la scène intersecté par le rayon, nous calculons pour chaque objet si le rayon l'intersecte puis parmi tous les objets intersectés nous prenons l'objet le plus proche de la caméra. Dans un deuxième temps nous renvoyons le point P (classe *Point*) d'intersection entre le rayon et l'objet.

3.2 Deuxième étape : Calcul de la couleur

A présent, il faut savoir si le point P est éclairé par les sources lumineuses. Pour cela pour chaque source S de la scène, nous regardons si le rayon reliant

P à S intersecte au moins un objet de la scène autre que l'objet courant. Pour chaque composante (r par exemple) nous calculons la lumière résultante par la formule :

$$r_{final} = \sum_{s \in \{\text{sources éclairant } P\}} r_s * \lambda(s) * \frac{r_{objet}}{255} \quad (1)$$

où $\lambda(s)$ est le cosinus entre la normale à l'objet en P et le vecteur $\overrightarrow{PS}$. On obtient ainsi une lumière que l'on transforme en couleur.

Remarque : Cette première version de moteur de lancer de rayon ne tient compte que de l'aspect mâte des objets, on ne considère ni la réflection ni la transparence des objets.

4 Ameliorations

A partir du moteur de lancer de rayons que nous avons codé, nous pouvons penser à diverses améliorations.

4.1 Améliorations visant à rendre l'image plus réaliste

— Ajouter des classes filles de la classe *Object*. Par exemple des cubes ou des surfaces planes.
— Rendre la classe Texture un peu plus exhaustive en prenant en compte la transparance, un indice de réfraction...
— Faire en sorte que *RadialObjectTexture* prenne un vecteur de couleurs plutôt que d'imposer le cycle rouve-vert-bleu.
— Permettre le texture mapping.
— Chercher à représenter des objets ayant une certaine épaisseur.
— Intégrer des phénomènes physiques tels que la diffraction, la réfraction...

4.2 Avancer vers un moteur plus fonctionnel...

— Uiliser deux caméras, générer deux images que l'on filtre l'un avec du rouge et l'autre avec du bleu par exemple, et observer grâce à des lunettes, une image en trois dimensions.
— Rajouter une dépendance temporelle des objets et regarder la scène évoluer.
— Déplacer la caméra dans la scène et visualiser de façon "fluide" les transformations, dans le but de faire des vidéos de synthèse par exemple.

5 Conclusion

Lors de ce projet sur la technique du lancer de rayons, nous avons compris à quel point les classes peuvent s'avérer être des outils puissants lors de l'implémentation de problèmes de ce type. En effet, la multiplicité des éléments d'un

moteur de lancer de rayon (scène, rayon, lumières, couleurs, objets, caméra, texture, etc) et leur forte intriquation mutuelle conduit à penser les programmes à travers les différentes dépendances des éléments entre eux et donc d'utiliser des classes. Ce projet à conduit à générer des images relativement réalistes, d'une scène contenant des sphères et des plans.

6 Annexes - Rendu

FIGURE 1 – Trois sphères (bleue, rouge et verte) éclairées par une unique lumière blanche

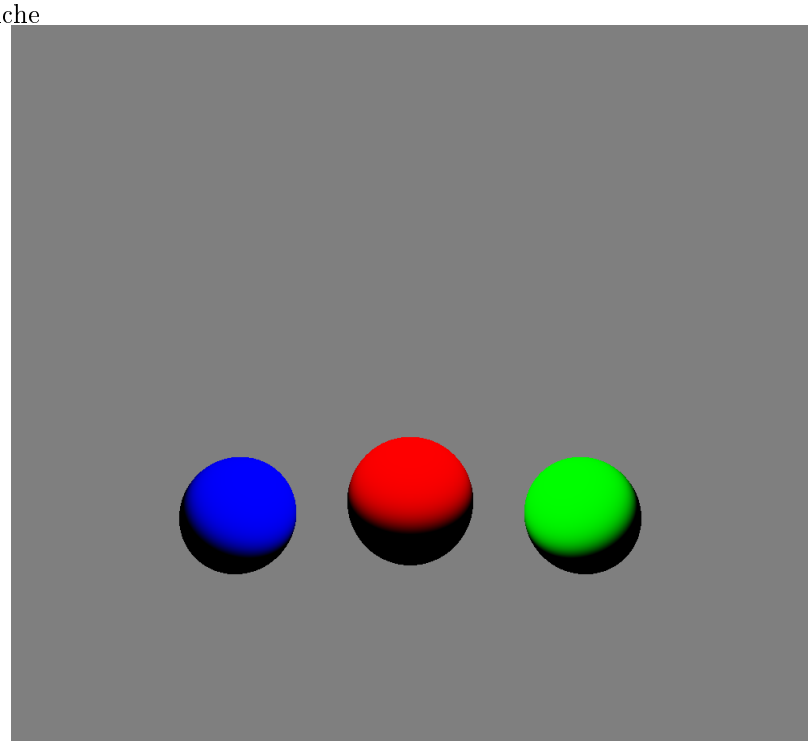


FIGURE 2 – Trois sphères noires et un plan blanc éclairées par cinq sources différentes

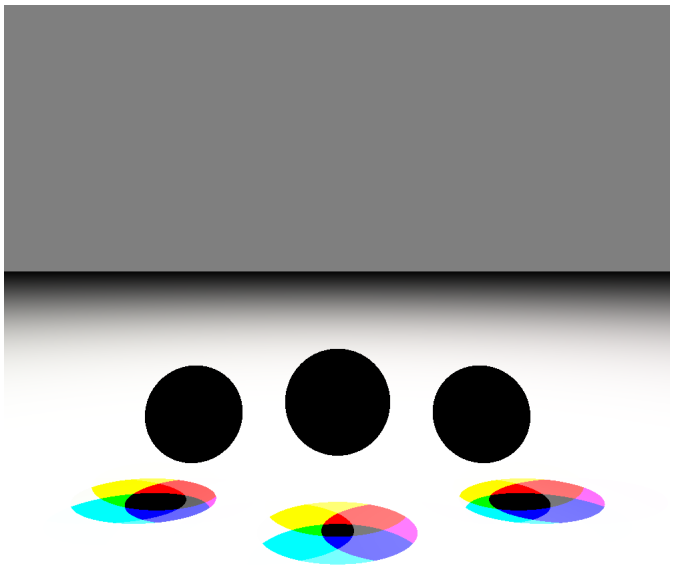


FIGURE 3 – Trois sphères noires et un plan de couleur radiale éclairées par cinq sources différentes