

# Outils de programmation et C++

## Projet : Lancer de rayons

### 1 Introduction

Le but de ce projet est de concevoir complètement un programme de synthèse d'images par lancer de rayons. Le lancer de rayon est une technique de production d'images de synthèse de haute qualité telle que la figure 1.

Le principe de cette technique est simple. Sur la base d'une scène décrite en 3D en positionnant divers objets et sources lumineuses ainsi qu'une "caméra", l'application des lois de l'optique géométrique permet de calculer l'image perçue par la caméra. Les moteurs les plus performants prennent également en compte des effets produits par l'optique ondulatoire tels que la diffraction ou la diffusion.

Ce programme ne sera pas optimisé, car le seul but est de comprendre certains mécanismes du langage *C++*.

### 2 Principe du lancer de rayon

#### 2.1 Description qualitative

Le lancer de rayons (*raytracing*) est une technique de rendu d'images de synthèses fondée sur les lois de l'optique géométrique. 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 (d'une partie seulement en pratique), il est possible de déterminer la couleur de chaque pixel de la caméra. Pour cela, il faut calculer les interactions des rayons simulés sortant de la caméra avec les différents objets et sources lumineuses de la scène (voir figure 1).

Une scène complète de lancer de rayon est formée d'une *caméra* (ou œil de l'observateur) associée à un *écran*, d'*objets* (des sphères dans notre cas), ainsi que d'une *source lumineuse ponctuelle* comme présenté dans la figure ??.



FIGURE 1 – Images produites par lancer de rayons.

L'algorithme du lancer de rayon se résume à ces étapes :

1. Pour chaque pixel de l'écran, calculer le rayon issu de la caméra et passant en son centre.
2. Calculer le point d'intersection entre ce rayon et le premier objet de la scène qu'il rencontre.
3. En déduire la couleur du rayon (et donc du pixel par lequel il passe) par application d'une formule donnée plus loin.

Il existe une grande variété de phénomènes lumineux que la technique du lancer de rayons peut modéliser. Nous nous limiterons à seulement deux d'entre eux dans ce projet<sup>1</sup> :

- La réflexion diffuse. C'est à dire la couleur qu'émet un objet dans toutes les directions lorsqu'il est éclairé par une autre source de lumière. Un objet sur lequel seule la réflexion diffuse existe est un objet mat.
- La réflexion spéculaire. C'est à dire la lumière qui est réfléchie à la surface de l'objet. Les miroirs sont l'exemple caractéristique de la réflexion spéculaire.

## 2.2 Points techniques importants

### 2.2.1 Calcul du point d'intersection entre rayon et sphère

Le calcul du point d'intersection entre un rayon et une sphère consiste à résoudre une équation polynomiale de degré 2. Cette équation s'obtient à partir de la représentation mathématique d'un rayon : une demi-droite caractérisée par son origine  $O$  (point depuis lequel il est lancé) et sa direction  $\vec{d}$ , tel que représenté sur la figure 2.

Une équation paramétrique du rayon est donc  $P = O + t\vec{d}$  où  $t \geq 0$  et  $P$  est un point sur le rayon, c'est à dire

$$\begin{cases} x(t) = x_0 + t \times x_d \\ y(t) = y_0 + t \times y_d \\ z(t) = z_0 + t \times z_d \end{cases} \quad (1)$$

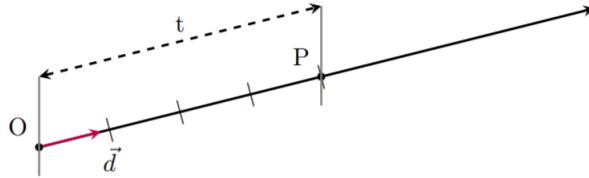


FIGURE 2 – Schéma d'un rayon d'origine  $O$ , de direction  $\vec{d}$ , avec un point  $P$  tel que  $P = O + t\vec{d}$

D'autre part, l'équation implicite d'une sphère de centre  $C = (x_c, y_c, z_c)$  et de rayon  $r$  est donnée par :

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

En substituant l'expression paramétrique du rayon dans l'équation implicite de la sphère, on obtient l'équation polynomiale en  $t$  de degré 2

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

Les solutions réelles positives de cette équation peuvent ensuite être injectées dans l'équation paramétrique du rayon pour obtenir les coordonnées des points d'intersections (0, 1 ou 2).

---

1. Plus en bonus

### 2.2.2 Calcul de l'illumination diffuse

Déterminer la couleur d'un point à la surface d'une sphère implique de déterminer dans un premier temps si ce point est éclairé par la source lumineuse de la scène. Pour ce faire, il suffit de déterminer si le rayon fictif partant de ce point à la surface de la sphère en direction de la source<sup>2</sup> rencontre un obstacle sur son chemin. Si oui, le point n'est pas éclairé par la source, sinon, il l'est. Cette étape réutilise la méthode de calcul entre rayon et sphère décrite ci-dessus.

Cette donnée est ensuite réutilisée dans une formule donnée plus loin dans le sujet.

### 2.2.3 Calcul du rayon réfléchi

La prise en compte de la réflexion spéculaire implique le calcul de rayons réfléchis. Le schéma 3 illustre ce principe simple. En notant  $O$  le point d'incidence du rayon lumineux,  $\vec{n}$  la normale à la surface unitaire sortante en  $O$  et  $\vec{r}$  le vecteur directeur *unitaire* du rayon, l'équation paramétrique du rayon réfléchi est :

$$P = O + t(\vec{r} - 2(\vec{r} \cdot \vec{n})\vec{n})$$

Le vecteur unitaire normal au point d'incidence  $O$  sur une sphère de centre  $C$  est

$$\vec{n} = \frac{\vec{CO}}{|CO|}$$

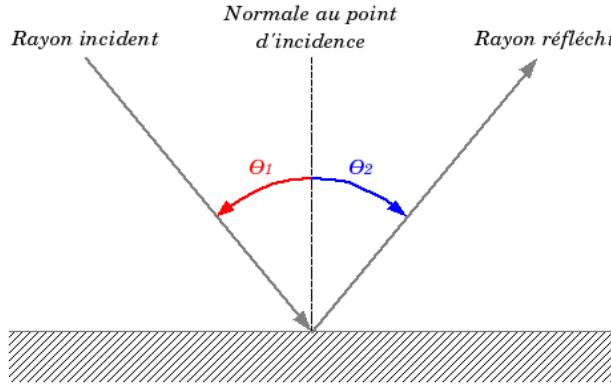


FIGURE 3 – Schéma de la réflexion d'un rayon lumineux ( $\theta_1 = \theta_2$ )

## 3 Gestion des fichiers d'entrée et de sortie

### 3.1 Parsing du fichier de scène

La scène 3D utilisée par le programme de lancer de rayon est décrite par l'utilisateur. Pour ce faire, le programme de lancer de rayons doit commencer par parser (lire et interpréter) un fichier texte dans lequel la scène est décrite.

Pour simplifier les choses, la scène ne contient qu'un ensemble de sphères, une source de lumière, une caméra et un écran. Un exemple de fichier de description de scène est le suivant :

```
# camera position (x = 100, y = 100, z = 0)
100 100 0

# screen position
# top left corner
90 110 30
```

2. Ce type de rayon fictif est appelé *shadow ray*

```

# top right corner
110 110 30

# bottom left corner
90 90 30

# screen horizontal resolution
400

# background color
80 80 100
# light source position (200, 300, 0) and color (245, 200, 200)
200 300 0 245 200 200

# three spheres
# center = (90, 90, 60) ; radius = 7 ; color = (250, 0, 0) ; reflexion = 0.3
sphere: 90 90 60 7 250 0 0 0.3
sphere: 100 110 60 7 0 250 0 0.3
sphere: 110 110 60 7 0 0 250 0.3

```

Voici les règles de rédaction du fichier texte :

- Une ligne débutant par # est un commentaire ;
- La première ligne non commentée indique la position de la caméra ;
- La ligne suivante indique la position du coin supérieur gauche de l'écran ;
- La ligne suivante indique la position du coin supérieur droit de l'écran ;
- La ligne suivante indique la position du coin inférieur gauche de l'écran ;
- La ligne suivante indique la résolution horizontale de l'écran (la résolution verticale se calcule à partir des coordonnées des coins de l'écran) ;
- La ligne suivante indique la couleur du fond ;
- La ligne suivante indique la position de la source lumineuse suivie de sa couleur RGB ;
- Les lignes suivantes indiquent la position des objets de la scène, i.e. des sphères dans notre cas. Elles débutent par **sphere**:. Elles sont suivies de la position du **centre** de la sphère, de son **rayon**, sa **couleur en RGB** et de son **indice de réflexion** entre 0 et 1.
- BONUS : Parmi les bonus proposés figure l'ajout d'autres formes. Préfixer leurs lignes d'un autre mot (ex : **triangle**:) avant d'écrire les caractéristiques que vous définirez.

Vous devez donc écrire un parser pour ce type de fichier qui générera une représentation de la scène dans une classe dédiée. La lecture de fichier est expliquée dans la documentation :

[http://en.cppreference.com/w/cpp/io/basic\\_ifstream](http://en.cppreference.com/w/cpp/io/basic_ifstream). Si cela ne suffit pas, *Google* regorge d'explications, surtout si on les cherche en anglais. Utilisez des conteneurs tels que les vecteurs de la *STL* pour stocker les sphères.

### 3.2 Écriture de l'image produite

L'image produite par votre programme de lancer de rayon doit être transmise à l'utilisateur. La technique la plus simple consiste à écrire l'image dans un fichier *ppm*. Les règles d'écriture d'un tel fichier sont disponibles partout sur internet et ici en particulier : [https://en.wikipedia.org/wiki/Netpbm\\_format](https://en.wikipedia.org/wiki/Netpbm_format). L'écriture de fichier est similaire à la lecture et est expliquée ici : [http://en.cppreference.com/w/cpp/io/basic\\_ofstream](http://en.cppreference.com/w/cpp/io/basic_ofstream). Là encore, *Google* peut fournir des explications plus simples.

Votre programme devra donc produire l'image *ppm* correspondant au fichier de scène passé en entrée.

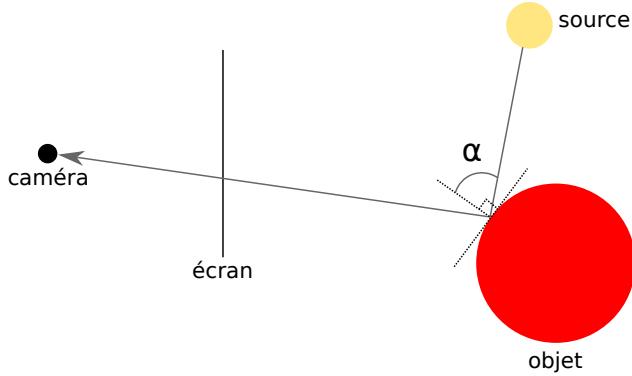


FIGURE 4 – Lancer de rayon avec sources primaires seulement.

## 4 calcul de l'image sans réflexion

La première étape consiste à calculer l'image sans prendre en compte la réflexion de la lumière sur les surfaces des objets de la scène. Pour ce faire, le point d'intersection des rayons issus de chaque pixel avec les objets de la scène doit être calculé. Si aucune intersection n'a été trouvée, la couleur du fond est utilisée. Sinon, l'intersection la plus proche est retenue. Il faut ensuite déterminer la couleur renvoyée par ce point d'intersection d'après la formule suivante :

$$C = E \times \cos(\alpha) \times \frac{C_i * C_s}{255}$$

où :

- $C$  est le vecteur RGB de la couleur du pixel ;
- $E$  vaut 1 si le point d'intersection entre le rayon et la surface est visible depuis la source lumineuse ;
- $\alpha$  est l'angle entre la normale à la surface sortante et le vecteur allant de l'intersection à la source lumineuse ;
- $C_i$  est le vecteur RGB de la couleur de la surface à l'intersection ;
- $C_s$  est le vecteur RGB de la couleur de la source lumineuse.
- L'étoile (\*) représente le produit terme à terme de deux vecteurs.

## 5 Calcul de l'image avec réflexion

La méthode de rendu utilisée jusqu'ici ne prend en compte que les sources de lumière primaires, c'est à dire les sources produisant leur propre lumière (ex : le soleil). Cependant, pour obtenir un rendu plus réaliste, il est important de considérer les sources secondaires, c'est à dire les sources retransmettant une partie de la lumière qu'elles ont reçue (ex : la lune).

La simulation complète des sources de lumière secondaires par lancer de rayon est extrêmement lourde. Les rayons secondaires proviennent en effet potentiellement de toutes les directions et il est impossible d'appliquer le principe de retour inverse de la lumière à ces derniers. Nous allons donc privilégier uniquement les rayons secondaires provenant de la réflexion spéculaire<sup>3</sup>.

Deux cas de figures se présentent alors comme illustré figure 5 :

- Le rayon secondaire est issu d'un autre objet par diffusion et réfléchi spéculairement au point de calcul de la couleur (rayon (2)) ;
- Le rayon secondaire est issu de la réflexion spéculaire d'un autre objet et diffusé au point de calcul (rayon (1)).

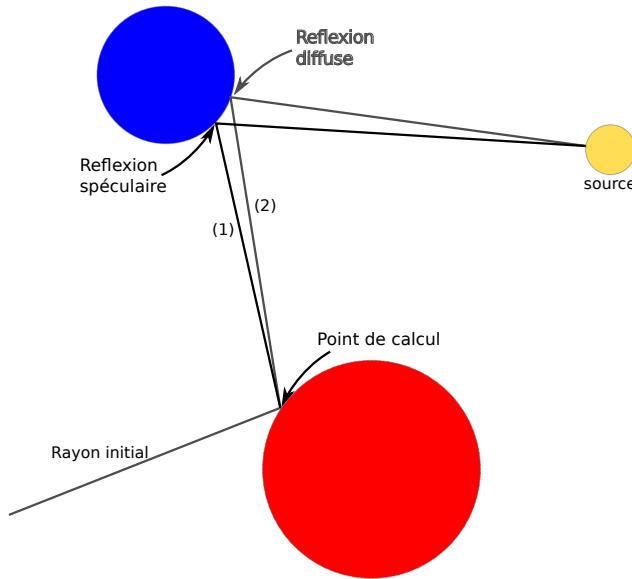


FIGURE 5 – Les deux types de réflexion spéculaire qui devraient être pris en compte.

Le second cas est complexe à calculer et implique la résolution d'équations polynomiales de degré important. Il peut même être impossible à résoudre analytiquement pour des formes plus complexes que des sphères. C'est pourquoi *nous nous limiterons au premier cas* qui reste simple à calculer à l'aide de la formule suivante :

$$C = E \times (1 - R) \times \cos(\alpha) \times \frac{C_i * C_s}{255} + R \times C_r$$

où :

- $C$  est le vecteur RGB de la couleur du pixel ;
- $E$  vaut 1 si le point d'intersection entre le rayon et la surface est visible depuis la source lumineuse ;
- $R$  est le coefficient de réflexion au point d'intersection ;
- $\alpha$  est l'angle entre la normale à la surface sortante et le vecteur allant de l'intersection à la source lumineuse ;
- $C_i$  est le vecteur RGB de la couleur de la surface à l'intersection ;
- $C_s$  est le vecteur RGB de la couleur de la source lumineuse.
- $C_r$  est le vecteur RGB de la couleur du rayon réfléchi spéculairement ;
- L'étoile (\*) représente le produit terme à terme de deux vecteurs.

La détermination de la couleur du rayon réfléchi spéculairement se fait en appliquant cette même formule récursivement en considérant le rayon réfléchi comme un rayon primaire. Attention à limiter le nombre maximum de réflexions car il n'est pas possible d'atteindre des niveaux de récursion aussi profond que souhaité.

## 6 Résumé des consignes

### 6.1 Fonctionnalités à implémenter

1. Parser le fichier d'entrée.
2. Proposer une implémentation pour chaque élément de la scène.

---

3. [https://fr.wikipedia.org/wiki/R%C3%A9flexion\\_\(optique\)#R.C3.A9flexion\\_sp.C3.A9culaire](https://fr.wikipedia.org/wiki/R%C3%A9flexion_(optique)#R.C3.A9flexion_sp.C3.A9culaire)

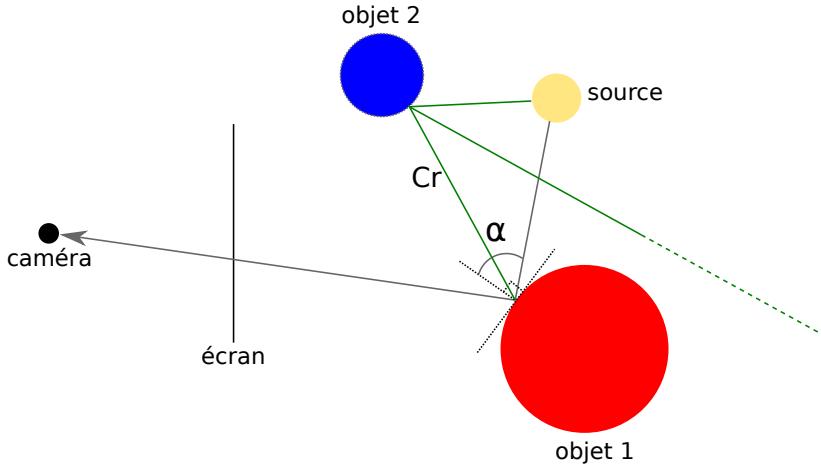


FIGURE 6 – Lancer de rayon avec rayon secondaires issus de la réflexion spéculaire.

3. Trouver le point d'intersection entre un rayon et les différents objets de la scène.
4. Générer des *tests unitaires* pour la fonction de calcul de l'intersection entre le rayon et la sphère.
5. Calculer la couleur d'un point d'intersection rayon/sphère **sans** réflexion.
6. Calculer la couleur d'un point d'intersection rayon/sphère **avec** réflexion.
7. Produire le fichier image en sortie après rendu.

## 6.2 Critères d'évaluation

Le projet est à rendre au plus tard au *31 Décembre 2017 à 23h59*. Un pénalité de **4 points** sera appliquée à chaque jour de retard. Le projet sera noté sur les points suivants :

- Le projet compile directement sous Linux (Code : :Blocks ou Makefile). Attention si utilisation de bibliothèques externes. (*8 point*)
- Le programme fournit l'image attendue sur notre scène de test ainsi que sur une scène de votre composition.
- Propreté du code : Indentation et documentation du code est *obligatoire*. (*7 points*)
- Un rapport de 5 pages en moyenne afin de documenter votre solution et l'approche abordée. (*5 points*)
  1. Architecture du code (diagramme de classe et diagramme d'activité).
  2. Si bonus effectué, explication.
  3. Autocritique de l'architecture adoptée (suggestions d'améliorations).
  4. Critique de la technique de lancer de rayon proposée.
  5. Résultats obtenus.
- Tout bonus rapportera entre 0,5 et 2 points suivant la difficulté et la qualité de mise en oeuvre de la solution.

## 7 Fonctionnalités bonus

Vous disposez maintenant d'un programme de lancer de rayon minimalisté mais il est possible de lui ajouter de nombreuses fonctionnalités. L'ajout de telles fonctionnalités pourra vous apporter un bonus sur votre note. Sachez cependant qu'une importance bien supérieure sera accordée à la qualité du travail fourni sur la partie obligatoire du projet. Les suggestions listées ici permettent avant tout d'approfondir votre pratique du C++ en gagnant quelques points au passage si la solution proposée est de bonne qualité.

Vous êtes libres d'apporter ou non les améliorations suggérées ici ou même d'en apporter d'autres de votre propre cru, pour peu qu'elles soient correctement documentées. La liste ci-dessous est classé par ordre globalement croissant de difficulté.

- Amélioration des performances par parallélisation OpenMp. Il s'agit d'une technique de programmation parallèle extrêmement puissante mais également simple à mettre en oeuvre. Très peu de code est à écrire mais à vous de trouver comment fonctionne OpenMP.
- Ajouter une luminosité ambiante pour relever la luminosité des zones non éclairées. Cette luminosité ambiante est une couleur constante que possède chaque surface et qui vient s'ajouter à celle calculée par la méthode implémenté jusqu'ici. Cet ajout peut cependant provoquer une saturation des couleurs, c'est à dire un dépassement de la valeur maximale des couleurs d'un pixel. On peut alors se contenter de limiter la valeur maximale à 255, ce qui provoquera une distorsion des couleurs. Cet effet peut cependant augmenter le réalisme en faisant apparaître des zones très lumineuses.
- Supporter des sources de lumière primaires multiples. Le problème de la saturation est ici aussi à traiter. Il n'est plus possible de seulement additionner les contribution des multiples sources lumineuses. Il n'est pas non plus possible de moyennner les contributions. Il faut alors trouver une loi permettant d'ajouter les contributions de chaque source de manière réaliste. L'utilisation du logarithme en combinaison d'une limitation par saturation est une possibilité.
- Ajout d'une forme de base supplémentaire. Le triangle est à privilégier puisqu'il permet ensuite d'approcher n'importe quelle forme.
- Ajouter la transparence aux objets. Cela implique d'ajouter à leurs caractéristiques un indice de transparence. La formule à utiliser devient alors :

$$C = E * (1 - R - T) * \cos(\alpha) * C_i \cdot C_s + R * C_r + T * C_t$$

où  $C_t$  est la couleur du rayon traversant. Et quite à ajouter la transparence, autant ajouter la réfraction à l'aide de la formule de Snell-Descartes :

$$n_1 \sin(i_1) = n_2 \sin(i_2)$$

où  $n_1$  et  $n_2$  sont les indices optiques des milieux de part et d'autre de la surface au point d'intersection du rayon lumineux et  $i_1$  et  $i_2$  sont les angles formés par les rayons incidents et refractés avec la normale à la surface.

- Implémentation de l'antialiasing par sur-échantillonage. Cela consiste à calculer une image dont la résolution est supérieure à celle demandée puis à effectuer une interpolation pour réduire sa taille à celle voulue.
- La gestion de la profondeur de champ par ajout d'un flou de mise au point. Cet effet de flou provient de divers facteurs ayant pour cause qu'un point lumineux est transposé en une tâche sur le capteur de l'appareil. Plus le point lumineux est éloigné du plan focal, c'est à dire le plan dans lequel l'image est nette, plus il devient flou. Une manière d'imiter ce phénomène approximativement est de pivoter l'ensemble caméra+écran autour du point de netteté afin d'obtenir plusieurs images légèrement décalées mais toutes centrées sur le même point. Il suffit ensuite de moyennner ces images. Cette technique n'est cependant que approximative puisqu'elle suppose que la zone de netteté est un point alors qu'elle se rapproche du plan dans la pratique.
- Remplacer les couleurs des surfaces par des textures provenant d'images. Cela implique d'ouvrir des images à l'aide de la SDL<sup>4</sup> puis de définir des règles de mappage de ces images vers les surfaces de objets<sup>5</sup>. Cela peut s'avérer plus simple sur des triangles que sur des sphères bien que des problèmes de continuité de texture à la jonction entre deux triangles surviennent.

---

4. Cf. exercice sur le flocon de Koch.

5. [https://en.wikipedia.org/wiki/Texture\\_mapping](https://en.wikipedia.org/wiki/Texture_mapping)