

Lancer de rayon

Laura GENEAU, Safidy RAKOTOBÉ

4 janvier 2026

Table des matières

1	Contexte	3
2	Mise en scène	3
3	Concepts physiques et mathématiques	3
3.1	Intersection d'un rayon et d'une shape	3
3.1.1	Intersection d'un rayon et d'une sphère	3
3.1.2	Intersection d'un rayon et d'un plan infini	4
3.2	Réflexion d'un rayon sur une surface	5
3.3	Calcul des ombres portées	6
4	Conception Logicielle : Diagramme UML	6
5	Conception Logicielle : Diagramme UML	6
6	Explication des méthodes et fonctions	7
7	Nos choix	7
8	Problèmes rencontrés et leur solution	8
8.1	Erreurs de calcul sur les flottants	8
8.1.1	Pour la réflexion	8
8.1.2	Pour l'ombre	9
9	Améliorations entreprises	11
9.1	La lumière	11
9.2	La caméra	12

1 Contexte

La synthèse d'images 3D repose principalement sur deux approches : la rasterisation et le lancer de rayon. Contrairement à la rasterisation qui projette des objets sur un plan, le lancer de rayon simule le trajet inverse de la lumière en suivant des rayons partant de l'œil du spectateur à travers chaque pixel de l'écran.

Ce projet vise à implémenter un algorithme de base permettant de calculer l'intersection de ces rayons avec des objets géométriques (sphères, cubes, quadrilatères) et de gérer les phénomènes d'illumination, d'ombre portée et de réflexion. La scène finale consiste en une boîte composée de cinq quadrilatères, contenant divers volumes et une source de lumière, offrant ainsi un rendu réaliste par simulation physique des rebonds lumineux.

2 Mise en scène

Nous avons une caméra orientée vers la face ouverte d'une boîte verte mate. Sur la face du haut de celle-ci, est disposée une source de lumière. Dans cette boîte nous avons ajouté une sphère rouge réfléchissante, un cube bleu réfléchissant et enfin un quad jaune mat.

3 Concepts physiques et mathématiques

3.1 Intersection d'un rayon et d'une shape

3.1.1 Intersection d'un rayon et d'une sphère

Le calcul d'intersection repose sur la résolution d'un système à deux équations :

- **Le Rayon** : Un point P appartenant au rayon est défini par :

$$P = P_0 + t\vec{d}$$

Où :

- P_0 est l'origine du rayon (`Vector3f`).
- \vec{d} est la direction du rayon (`Vector3f`).
- t est la distance parcourue depuis l'origine (`float`), avec $t > 0$.

- **La sphère** : Un point P appartient à la sphère si :

$$\|P - C\|^2 = R^2$$

Où :

- C est le centre de la sphère (**Vector3f**).
- R est le rayon de la sphère (**float**).

En substituant l'équation du rayon dans celle de la sphère, on obtient une équation du second degré : $at^2 + bt + c = 0$, où :

- $a = \vec{d} \cdot \vec{d}$
- $b = 2\vec{d} \cdot (P_0 - C)$
- $c = (P_0 \cdot P_0) + (C \cdot C) - 2(C \cdot P_0) - R^2$

Interprétation du discriminant $\Delta = b^2 - 4ac$:

- **Si $\Delta < 0$:** Aucune intersection réelle : le rayon ne touche pas la sphère.
- **Si $\Delta = 0$:** Le rayon est tangent à la sphère : une seule intersection.
- **Si $\Delta > 0$:** Le rayon traverse la sphère : deux points d'intersection. On choisit le point d'intersection le plus proche de la caméra.

3.1.2 Intersection d'un rayon et d'un plan infini

De même que pour la sphère, on a deux équations :

- **Le Rayon :** Un point P situé sur le rayon est défini par :

$$P = P_0 + t\vec{d}$$

- **Le Plan :** Un point P appartient au plan si le vecteur allant du centre de la surface vers P est perpendiculaire à la normale :

$$(P - \text{origine}) \cdot \vec{N} = 0$$

Où :

- *origine* est le centre de la surface (**Vector3f**).
- \vec{N} est le vecteur normal à la surface (**Vector3f**).

En substituant l'équation du rayon dans celle du plan, on cherche la valeur de t telle que :

$$(P_0 + t\vec{d} - \text{origine}) \cdot \vec{N} = 0$$

En développant, on obtient la distance d'intersection t :

$$t = \frac{(\text{origine} - P_0) \cdot \vec{N}}{\vec{d} \cdot \vec{N}}$$

Remarque : Si le dénominateur $\vec{d} \cdot \vec{N} = 0$, cela signifie que le rayon est parallèle au plan et qu'il n'y a pas d'intersection.

3.2 Réflexion d'un rayon sur une surface

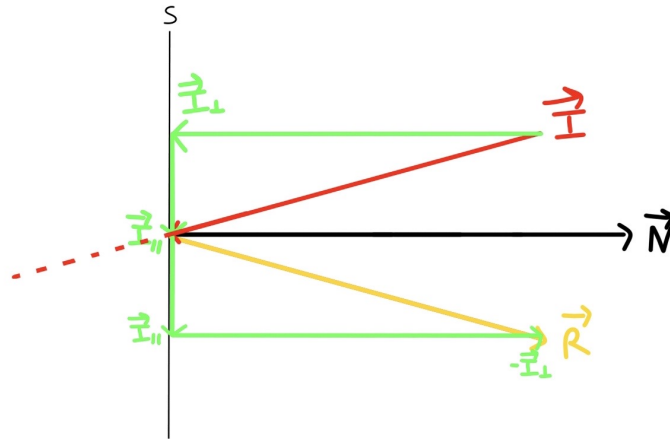


FIGURE 1 – Réflexion d'un rayon sur une surface avec un coefficient de réflexion (shininess) non nul

Légende du schéma :

- S : Surface d'impact
- \vec{I} : Rayon incident
- \vec{I}_\perp : Composante perpendiculaire du rayon incident
- \vec{I}_\parallel : Composante parallèle du rayon incident
- \vec{R} : Rayon réfléchi
- \vec{N} : Vecteur normal à la surface

Calculs :

$$\vec{I} = \vec{I}_\perp + \vec{I}_\parallel$$

$$\begin{aligned}
 \vec{R} &= \vec{I}_\parallel - \vec{I}_\perp \\
 &= \vec{I} - 2(\vec{I} \cdot \vec{N})\vec{N} \\
 &= \vec{I}_\perp + \vec{I}_\parallel - 2\vec{I}_\perp \\
 &= \vec{I} - 2\vec{I}_\perp \\
 &= \vec{I} - 2(\vec{I} \cdot \vec{N})\vec{N} \quad \text{car } \vec{I} \cdot \vec{N} \leq 0
 \end{aligned}$$

3.3 Calcul des ombres portées

Pour déterminer si un point d'impact P est situé dans l'ombre, on génère un **rayon d'ombre** (shadow ray) partant de P en direction de la source de lumière L .

- **Direction du rayon** : $\vec{d}_{ombre} = \frac{L-P}{\|L-P\|}$
- **Origine corrigée** : Pour éviter l'auto-intersection due aux erreurs de précision flottante, on décale l'origine : $P_{start} = P + \epsilon \vec{N}$ (où ϵ est une valeur très petite).

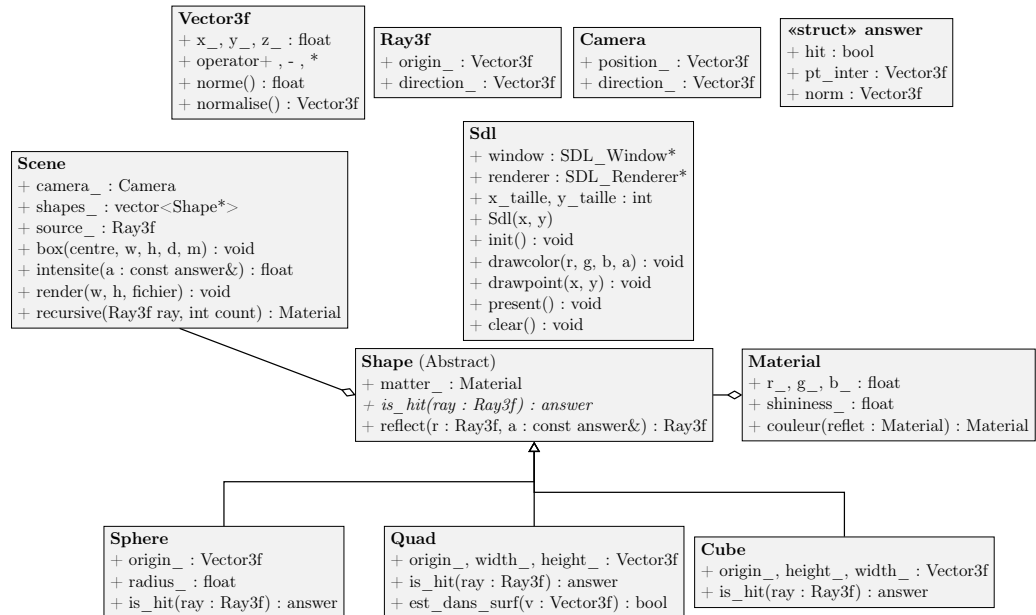
Un point est considéré comme **à l'ombre** si une intersection est détectée avec un objet de la scène à une distance $d_{obstacle \rightarrow Lumiere}$ telle que :

$$0 < d_{obstacle \rightarrow Lumiere} < d_{P \rightarrow Lumiere}$$

Si un obstacle est trouvé dans cet intervalle, l'intensité lumineuse du pixel est réduite à la valeur de la lumière ambiante. Dans le cas contraire, on applique le modèle d'illumination classique (produit scalaire entre la normale et le vecteur lumière).

4 Conception Logicielle : Diagramme UML

5 Conception Logicielle : Diagramme UML



Note : Les fonctions globales suivantes complètent le système :

- `draw_color(SDL_Renderer*, Material, float)` : Application de la couleur avec intensité.
- `sauvegarder_image(SDL_Renderer*, int, int, string)` : Capture BMP de l'écran.
- `prod_scal(v1,v2)` et `prod_vect(v1,v2)` : Opérations mathématiques sur les vecteurs.
- `egal(Vector3f v1, Vector3f v2)` : Test d'égalité sur des vecteurs ;
- `max(float a, float b)` : Maximum entre deux nombres flottants.

6 Explication des méthodes et fonctions

Présentes dans le fichier Doxyfile.

7 Nos choix

La méthode `is_hit` ne renvoie pas seulement un booléen indiquant si oui ou non il y a eu intersection entre un rayon et une shape, mais également :

- Les coordonnées du point d'intersection lorsqu'il est présent. (un vecteur nul est renvoyé s'il n'y a pas d'intersection).
- La normale à la surface au point d'intersection le plus proche de la caméra (un vecteur nul est renvoyé s'il n'y a pas d'intersection).

En effet, pour savoir s'il y a intersection entre la **shape** et le rayon, nous avons dû calculer ce point et vérifier s'il était contenu dans les contraintes spatiales de la forme. Ce type de retour, structuré via le type `answer`, nous a permis de ne pas avoir la méthode `reflect` virtuelle dans la classe `shape`. Ainsi, nous n'avons pas eu à coder trois méthodes différentes pour le cube, le quad et la sphère, mais bien une seule et même méthode commune à toutes les shapes.

Les arguments de type simple tels que `Ray` ou `Vector3f` ont été passés par valeur. Les arguments de type `answer` ont quant à eux été passés par référence car il s'agit d'une structure plus grosse qu'un simple `Vector3f` dont on ne souhaite pas faire la copie pour ne pas encombrer la mémoire.

Nous avons codé dans la méthode recursive à la fois le calcul de l'ombre et celui de la réflexion car toutes deux passent par la recherche

de la plus proche shape intersectée. Alors que la réflexion s'appelle récursivement, ce n'est pas le cas pour l'ombre d'où la présence d'un compteur qui permet non seulement de ne pas appeler la fonction récursive un nombre infini de fois (dans le cas où deux miroirs se feraient face par exemple) mais aussi de rajouter une condition sur ce compteur afin que seule la partie sur la réflexion soit récursive et que celle sur l'ombre ne soit appelée qu'une fois pour chaque rayon.

On aurait souhaité pouvoir initialiser par défaut la structure answer avec les valeurs correspondantes au cas où le rayon n'intersecte rien mais la version du C++ utilisé nous génère des warnings dans ce cas. Nous avons donc redéfini les valeurs de cette structure à chaque fois qu'il n'y avait pas d'intersection.

8 Problèmes rencontrés et leur solution

8.1 Erreurs de calcul sur les flottants

Comme nous faisons des calculs sur les flottants, il y a des arrondis qui faussent légèrement les calculs et donc le rendu final.

8.1.1 Pour la réflexion

Lorsque nous calculons le rayon réfléchi, il se peut qu'avec les erreurs de calculs engendrés par la manipulation de flottants, le point d'intersection entre le rayon et la shape, qui correspond au point d'origine du rayon réfléchi, se trouve à l'intérieur de cette même shape. Ce rayon, à l'intérieur de la shape repart donc vers l'extérieur de cette même shape avec un angle opposé au rayon incident. La shape étant fermée, la source de lumière n'éclaire pas l'intérieur de la shape qui est donc foncé d'où le fait que le rayon réfléchi soit foncé et d'où la présence de points noirs.

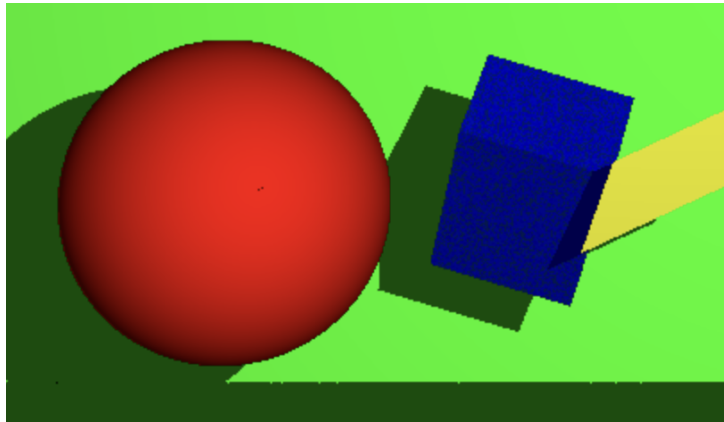


FIGURE 2 – Réflexion d'un rayon sans décalage de l'origine du rayon réfléchi

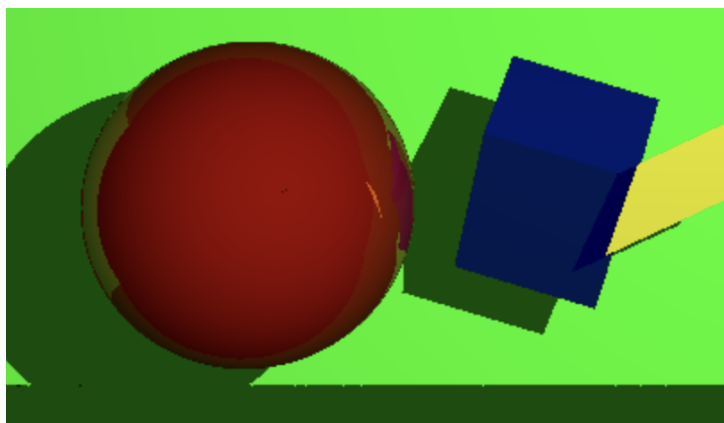


FIGURE 3 – Réflexion d'un rayon avec décalage de l'origine du rayon réfléchi

8.1.2 Pour l'ombre

De manière analogue à la réflexion, le calcul de l'ombre souffre des imprécisions des flottants. Lors du lancer du rayon vers la source lumineuse, si l'origine du rayon (le point d'intersection) n'est pas légèrement décalée le long de la normale, le rayon risque de s'auto-intersecter avec sa propre surface d'origine. L'algorithme détecte alors un obstacle entre le point et la lumière, ce qui crée des artefacts visuels appelés « shadow acne » (points noirs parasites) là où l'objet devrait être éclairé.

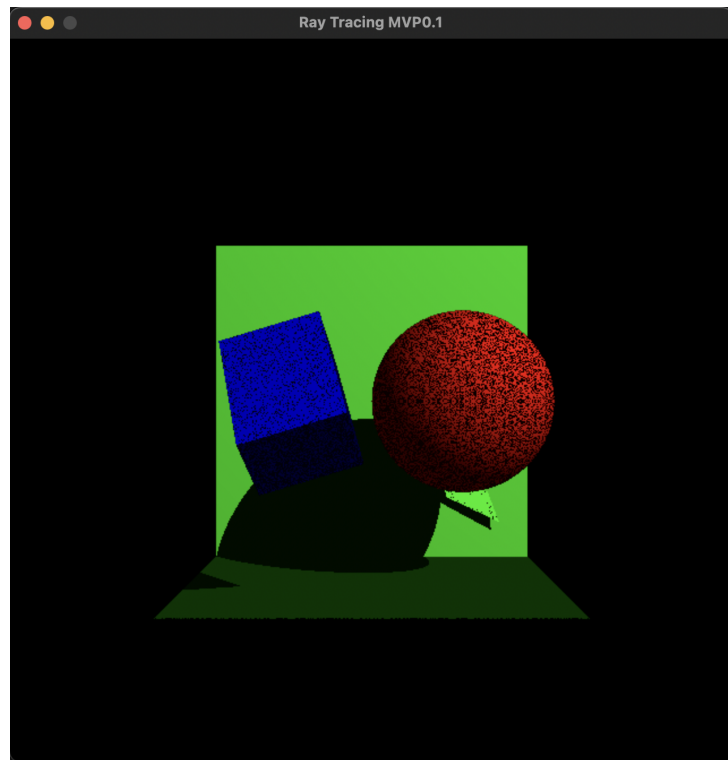


FIGURE 4 – L'objet s'auto intersecte par endroit

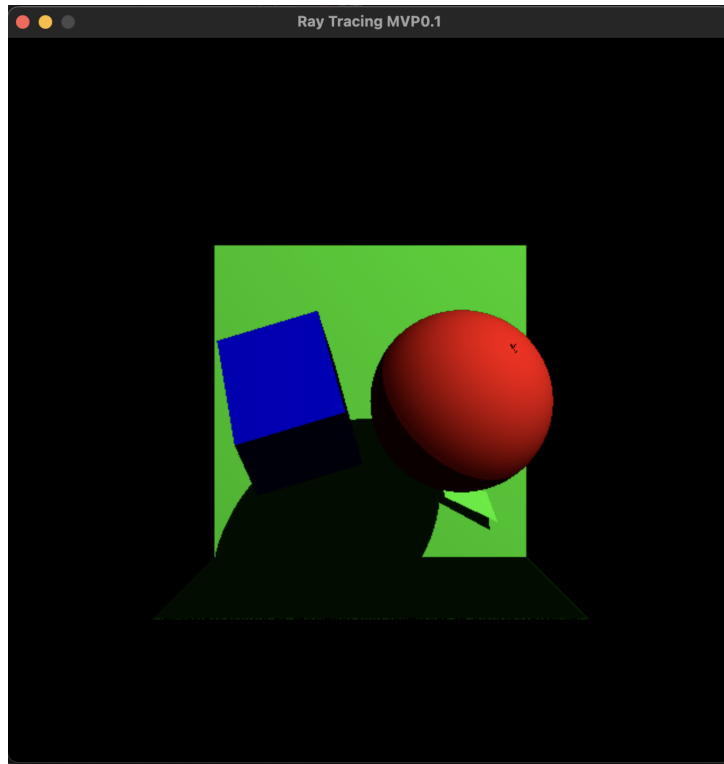


FIGURE 5 – Rayon décalé donc plus d’auto-intersection

9 Améliorations entreprises

Nous avons souhaité pouvoir déplacer la source de lumière et la caméra avec les touches du clavier. La lumière étant initialisée pour être localisée vers le centre de la face haute de la boîte et la caméra étant initialisée pour être

9.1 La lumière

- la touche **z** : pour monter la lumière.
- la touche **s** : pour descendre la lumière.
- la touche **q** : pour décaler la lumière à gauche.
- la touche **d** : pour décaler la lumière à droite.
- la touche **e** : pour éloigner la lumière de l'utilisateur (la décaler vers le fond de la boîte).
- la touche **w** : pour rapprocher la lumière de l'utilisateur.

9.2 La caméra

- *la flèche du haut* : pour monter la caméra.
- *la flèche du haut* : pour descendre la caméra.
- *la flèche de droite* : pour décaler la caméra vers la droite.
- *la flèche de gauche* : pour décaler la caméra vers la gauche.