

---

# Rendu 3D réaliste par lancé de rayons

## IS1260 - Sujet

Matthieu Garrigues,  
Vladimir Paun

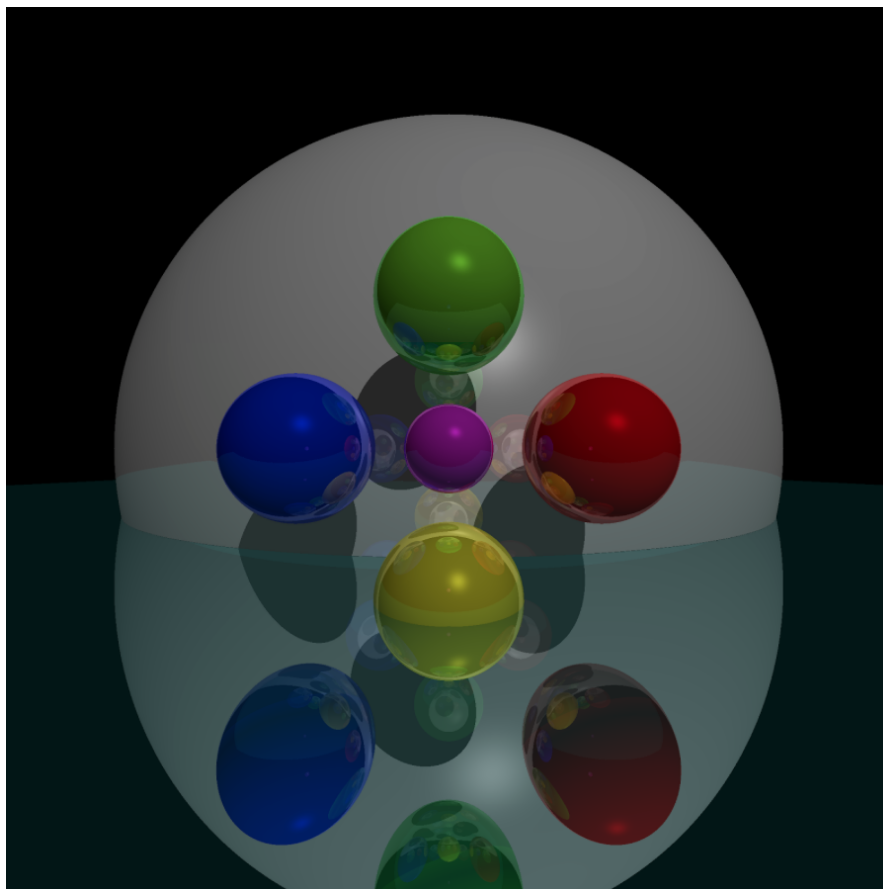
---

### Prérequis

- Algèbre linéaire
  - Connaissances de bases en Python3
  - Structures de données
  - Récursivité
- 

### Instructions générales

- En cas de problèmes, référez-vous au sujet, références bibliographiques, recherchez sur internet, copiez vos message d'erreurs dans un moteur de recherche, etc... Si malgré tout vous restez sans réponses, posez votre question au chargé de TP.
- Utilisez l'éditeur de texte de votre choix pour les exercices. Vous pouvez un éditeur de texte simple comme emacs, ou un environnement de développement plus complet comme spyder ou idle.
- **N'utilisez pas le copier-coller depuis ce PDF.** Il ne conserve pas les indentations python et introduira des erreurs de syntaxe difficiles à corriger.
- Documentez et commentez précisément votre code : décrivez l'objectif général de l'algorithme, expliquez les points-clefs de son implémentation, détaillez les conditions normales d'exécution du script (le fichier ".py") et exposez les cas extrêmes et les erreurs prévues.



# 1 Introduction

Le lancé de rayon est une technique de rendu 3D qui consiste à tracer le chemin de la lumière arrivant sur chacun des pixels de l'image rendue. Cette technique permet un réalisme de beaucoup plus haute qualité, au prix d'un temps de calcul beaucoup plus élevé. Elle est utilisée dans plusieurs industries comme par exemple le cinéma ou l'architecture.

Le but de ce projet est d'implémenter un lancé de rayon 3D capable de rendre des scènes 3D simples.

## 2 Tests

Pour vous aider à vérifier la validité de votre code, nous vous fournissons une série de tests. Téléchargez le fichier zip <http://ensta.fr/~garrigues/is1260/tests.zip> et placez le dossier qu'il contient dans le répertoire de votre projet.

## 3 Numpy

Numpy est une bibliothèque python définissant une série d'outils de calcul numérique. Nous allons utiliser dans ce projet ses opérateurs de calcul vectoriel. Le type de base de Numpy, la matrice, sera utilisé dans ce projet pour représenter les images, les coordonnées et les couleurs.

Quand cela sera nécessaire, vous importerez le module numpy ainsi que son module d'algèbre linéaire dans vos fichiers python.

```
import numpy as np
from numpy import linalg as la
```

Voici les différents outils dont vous aurez besoin pour implémenter le lancé de rayon:

```
# Crée une matrice de taille 100x300x3 initialisée à zéro
# (idéal pour stocker une image RGB).
img = np.zeros((100, 300, 3))

# Crée un vecteur 3D à partir d'une liste python
python_liste = [2,3,4]
numpy_vector = np.array(python_liste);

a = np.array([1,2,3])
b = np.array([4,5,6])

# Opérateurs
c = (a + b - 2 * a) / 23.3

# Produit scalaire
prod = a.dot(b)

# La norme, fonction du module linalg
n = la.norm(a)
```

## 4 Définition de la classe Camera

Dans les applications de rendu 3D, la caméra définit comment la scène est projetée sur l'image. Dans le cadre de ce projet, la caméra sera positionnée aux coordonnées  $[0, 0, 0]$  et visera le point  $[0, 0, 1]$ .

Dans le fichier `camera.py`, définissez la classe `Camera`. Son constructeur prendra en arguments, dans l'ordre suivant:

- `image_nrows`: le nombre de lignes de l'image rendue
- `image_ncols`: le nombre de colonnes de l'image rendue
- `focal_length`: la distance focale

```
class Camera
    def __init__(self, image_nrows, image_ncols, focale_length):
        # Fixme
```

Lancez le test `tests/01_camera.py` et assurez vous qu'aucune assertion n'ait échoué.

## 5 Création d'un rayon

Étant donné une caméra, exprimez la direction du rayon passant par le pixel aux coordonnées  $x, y$  de l'image rendue.

Pour effectuer le rendu de la scène, nous allons tracer les rayons partant de la caméra. Pour simplifier la projection, vous placerez la caméra aux coordonnées  $[0,0,0]$  et viserez le point  $[0,0,1]$  pour qu'il soit placé au centre de l'image.

**Attention, les lignes de l'image sont numérotées de haut en bas.**

Créez le module `raytracer.py`. Ajoutez-y la classe `Ray` avec son constructeur qui prendra dans l'ordre suivant les membres à initialiser:

- `starting_point`: Les coordonnées 3D de départ du rayon
- `direction`: Le vecteur 3D directeur du rayon

Implémentez la méthode `ray_at(row, col)` de la classe `Camera`. Les rayons retournés par la méthode devront respecter les conditions suivantes:

- Tout les rayons ont  $[0,0,0]$  pour `starting_point`.
- la direction  $[0,0, \text{distance\_focale}]$  est associée au pixel  $[\text{nrows} / 2, \text{ncols} / 2]$
- la direction  $[1,1, \text{distance\_focale}]$  est associée au pixel  $[0,0]$

La méthode prendra trois arguments:

- `self`: Comme pour toute méthode d'objet en python
- `row`: La position de la ligne
- `col`: La position de la colonne

```
def ray_at(self, row, col):
    # fixme
```

Lancez le test `tests/02_rayon.py` et assurez vous qu'aucune assertion n'ait échoué.

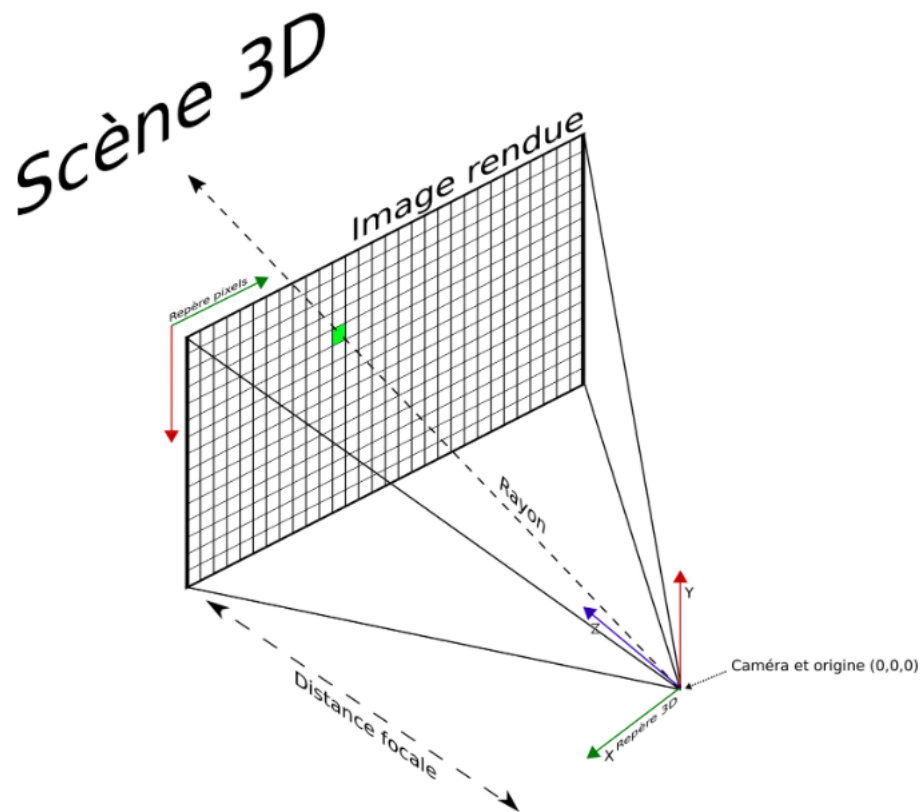


Figure 1: Projection d'un point de la scène sur la caméra.

## 6 La classe Sphere

Dans un premier temps, nous allons rendre une scène composée seulement de sphères. Créez le module `sphere.py` pour y ajouter la classe `Sphere`. Son constructeur prendra en argument et initialisera ces deux champs:

- `center`: Les coordonnées 3D du centre
- `rayon`: Le rayon

## 7 Intersection Rayon - Sphere

Étant donné un rayon et une sphère, calculez la coordonnée de la première intersection entre la sphère et le rayon. Calculez aussi la normale à la sphère en cette même coordonnée.

Étant donné l'équation d'un rayon et d'une sphère, vous allez calculer le premier point de contact entre la sphère et le rayon, si il existe.

Commencez par créer le module `intersection.py` et implémentez-y la classe `Intersection` qui contiendra toutes les informations relatives à une intersection. Son constructeur prendra en argument et initialisera les champs suivants:

- `position`: Les coordonnées 3D de l'intersection
- `normal`: Le vecteur 3D normal à l'intersection

- **object**: L'objet impliqué dans l'intersection

Une fois la classe Intersection terminée, vous allez ajouter la fonction `intersect` au module `intersection`. Dans la suite du TP, cette fonction gèrera plusieurs types d'objets, mais pour l'instant, seul le cas de la sphere est à implémenter.

Si il existe une intersection entre le rayon et la sphère, retournez une instance de la classe `intersection`. Sinon, retournez la valeur `None`.

```
def intersect(object, ray):
    if type(object) is Sphere:
        # Fixme
```

Si besoin, référez vous à Wikipedia pour l'équation de l'intersection ligne/sphère:

[https://en.wikipedia.org/wiki/Line-sphere\\_intersection](https://en.wikipedia.org/wiki/Line-sphere_intersection)

Assurez vous que tout les cas suivants retournent bien les intersections désirées:

- Un rayon démarrart en dehors de la sphère qui n'intersecte pas la sphère.
- Un rayon démarrart en dehors de la sphère et qui intersecte la sphère.
- Un rayon démarrart à l'intérieur de la sphère qui intersecte la sphère.

Attention : La normale de l'intersection avec un rayon provenant de l'extérieur de la sphère est différente de celle provoquée par un rayon provenant de l'intérieur.

Lancez le test `tests/03_sphere_intersection.py` et assurez vous qu'aucune assertion n'ait échoué.

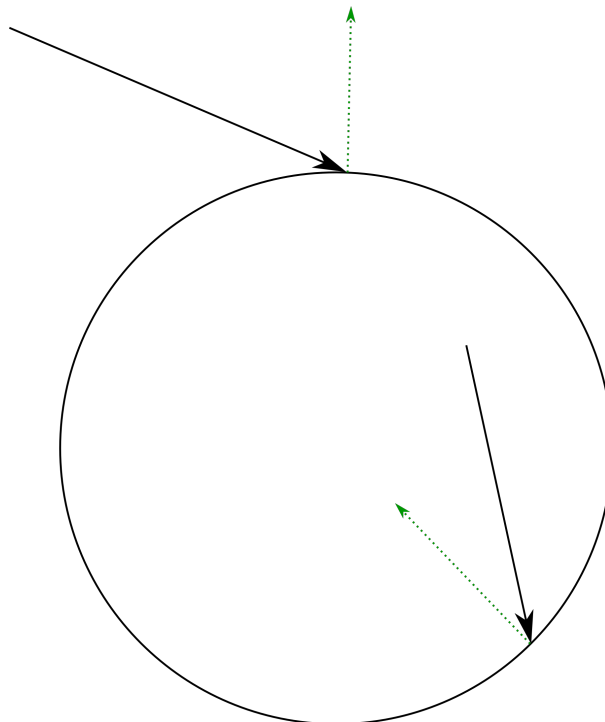


Figure 2: Coupe 2D d'une sphère montrant deux exemples d'intersection. Les normales sont représentées en vert.

## 8 Matériaux, lumière ambiante, diffuse et spéculaire

Étant donné les coordonnées d'un point d'une surface, sa normale, sa couleur et la configuration du matériel, calculer l'illumination du point. Vous allez utiliser le modèle de Phong qui simule la lumière ambiante, diffuse et spéculaire.

Dans ce modèle, la couleur émise par chaque surface est calculée en fonction de plusieurs facteurs:

- La couleur intrinsèque de l'objet
- Les propriétés du matériel (ou la matière) qui compose l'objet
- La normale de l'objet au point éclairé
- Les couleurs et positions des lumières qui éclairent l'objet
- La position de la caméra

Lisez rapidement la description du modèle de Phong sur Wikipedia: [https://en.wikipedia.org/wiki/Phong\\_reflection\\_model](https://en.wikipedia.org/wiki/Phong_reflection_model)

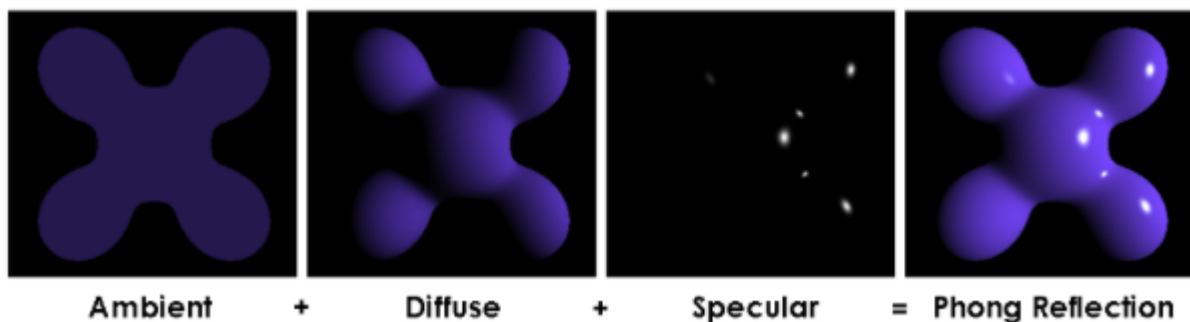


Figure 3: Illumination utilisant le modèle de Phong.

Avant d'implémenter ce modèle, nous allons définir les deux classes qui vont nous permettre de modéliser l'éclairage: Material et Spotlight.

### 8.1 La classe Material

Comme indiqué dans la description du modèle sur Wikipedia, implémentez dans le module `scene.py` la classe Material qui définira les propriétés de matière d'un objet. Le constructeur prendra en arguments et initialisera les membres suivants:

- `color`: La couleur intrinsèque de l'objet sous forme d'un vecteur 3D (rouge, vert, bleu) avec des coefficients compris entre 0 et 1
- `ambient`: Le coefficient de lumière ambiante compris entre 0 et 1
- `diffuse`: Le coefficient de lumière diffuse compris entre 0 et 1
- `specular`: Le coefficient de lumière spéculaire compris entre 0 et 1
- `shininess`: Le coefficient de brillance compris entre 0 et 1

- reflection : Le coefficient de réflexion compris entre 0 et 1

Le coefficient de réflexion ne concerne pas le modèle de Phong mais sera utilisé par l'implémentation des réflexions entre objets.

Testez le constructeur de la classe Material avec le test `tests/04_material.py`

### 8.1.1 Le matériel de la classe sphère

Ajoutez le champ `material` au constructeur de la classe `sphère`. Ce champ définira comment chaque instance de sphère réfléchira la lumière.

## 8.2 La classe Spotlight

Créez le module `light.py` et implémentez y la classe `Spotlight` avec un constructeur prenant en arguments les membres suivants:

- position: Les coordonnées 3D de la lumière ponctuelle
- color: La couleur de la lumière émise sous forme d'un vecteur 3D RGB.

## 8.3 Le modèle d'illumination de Phong

Dans le module `light.py`, définissez la fonction `phong_illuminate` qui prendra en argument:

- light: L'objet `Spotlight` éclairant l'objet
- position: les coordonnées 3D du point à éclairer
- normal: La normale de la surface à la position en question
- **object**: L'objet à éclairer (utilisez le champ `object.material` pour accéder aux propriétés du matériel
- viewer: La position de l'observateur

```
def phong_illuminate(light, position, normal, object, viewer):
    # Fixme
```

La fonction retournera la couleur RGB résultant des composantes diffuses et spéculaires de Phong. La composante ambiante sera calculée dans une autre fonction.

Si  $i$  est l'intensité calculée par le modèle de Phong,  $[L_r, L_g, L_b]$  la couleur de la lumière et  $[C_r, C_g, C_b]$  la couleur de la surface, `phong_illuminate` retournera la valeur suivante:

$$i \times [L_r \times C_r, L_g \times C_g, L_b \times C_b]$$

Si la surface n'est pas orientée vers la lumière, `phong_illuminate` retournera la couleur noire.

Implémentez maintenant la fonction `ambient_illuminate` qui calcule pour un objet donné la composante de couleur résultant de la lumière ambiante.

```
def ambient_illuminate(object):
    # Fixme
```

## 9 La classe Scene

Nous allons créer une classe `Scene` pour y stocker tout les objets et les lumières. Dans le module `scene.py`, ajoutez la classe `Scene`. Elle stockera deux membres: une liste d'objets nommée `objects` et une liste de lumières nommée `lights`. Implémentez les deux méthodes suivantes:

- `add_object(o)`: Ajoute l'objet `o` à la liste d'objets
- `add_light(l)`: Ajoute la lumière `l` à la liste de lumières

## 10 Rendu de la scène

Nous allons utiliser les fonctions et classes écrites précédemment pour calculer la couleur de chaque pixel de l'image et générer une première image.

Dans le module `raytracer.py`, commencez par implémenter la première version du tracé d'un rayon via la fonction `trace_ray(ray, scene, camera)`. Cette fonction prendra en argument, dans l'ordre suivant:

- `ray`: Le rayon à tracer
- `scene`: La scène
- `camera`: La caméra

Utilisez vos fonctions `ambient_illuminate`, `phong_illuminate` et `intersect` pour calculer la couleur de la première intersection rencontrée par le rayon `ray`. `trace_ray` retournera cette couleur si une intersection est trouvée et la couleur noire `[0, 0, 0]` sinon.

Attention: La lumière ambiante est à appliquer une seule fois alors que le modèle de Phong doit être appliqué pour chaque source de lumière.

Implémentez ensuite `raytracer.render`. Cette fonction reposera sur la méthode `camera.ray_at` et `trace_ray` pour renseigner chaque pixel de l'image de rendu. Elle prendra en argument:

- `camera`: Une instance de la classe `Camera`
- `scene`: Une instance de la classe `Scene`

Utilisez une matrice `numpy` à pour stocker l'image RGB de rendu. Initialisez la avec `np.zeros((nrows, ncols, 3))`. Retournez cette matrice à la fin de la fonction.

## 11 Rendu d'une scène simple

Créez un répertoire `scenes` dans votre dossier de projet. Ajoutez y un fichier `one_sphere.py` dans lequel vous allez initialiser et rendre une `Scene` contenant:

- Une sphère bleue de rayon 1 placée au coordonnées `[0, 0, 3]`
- Une lumière de type `Spotlight` et de couleur blanche aux coordonnées `[1, 1, 0]`



Générez une image de  $200 \times 200$  pixels en utilisant une distance focale de 1. Enregistrez le tableau de pixels retourné par la fonction `raytracer.render` dans le fichier `one_sphere.png` en utilisant la fonction `pyplot.imsave`. Référez vous à la documentation pour en savoir plus sur l'utilisation de cette fonction: [http://matplotlib.org/api/image\\_api.html?highlight=imsave#matplotlib.image.imsave](http://matplotlib.org/api/image_api.html?highlight=imsave#matplotlib.image.imsave)

Attention, étant donné que les modules python de votre raytracer se trouvent dans le dossier parent, vous devez indiquer à python que vous reposez sur des modules situés dans ce dernier:

```
import sys
sys.path.append( '.. ' )
```

## 12 Ombres

Jusqu'à maintenant, l'illumination des objets était implémentée sans la gestion des ombres. Dans un monde un peu plus réaliste, si un objet A est situé entre l'objet B et la lumière, B ne devrait pas être illuminé.

Dans le module `raytracer`, implémentez la fonction `compute_light(light, scene, intersection, viewer)` qui, pour une lumière, une scene, une intersection et un point de vue, retourne la couleur donnée par le modèle de phong via la fonction. `phong.illuminate` seulement si aucun objet n'est positionné entre la lumière et l'intersection. En cas d'occultation, la fonction retournera la couleur `[0,0,0]`. La fonction `intersect` vous permettra de tester simplement la présence d'une occultation.

Les arguments de `compute_light` seront:

- `light`: La lumière éclairant l'objet de type `Spotlight`.
- `scene`: La scène de type `Scene`.
- `intersection`: L'intersection à illuminer, de type `Intersection`.
- `viewer`: Les coordonnées 3D de l'observateur.

Faites reposer la fonction `trace_ray` sur `compute_light` au lieu de `phong.illuminate` pour l'illumination.

Dans le fichier `scenes/shadow.py`, codez le rendu de la scène suivante:

- Une sphère bleue de rayon 0.8 placée au coordonnées `[0, 0, 3]`
- Une sphère rouge de rayon 0.5 placée au coordonnées `[0.5, 0.5, 2]`
- Une lumière de type `Spotlight` et de couleur blanche aux coordonnées `[1, 1, 0]`

Vous devriez observer l'ombre de la sphère rouge projetée sur la sphère bleue.

## 13 Réflexions

Dans cette partie du projet, nous allons implémenter le lancé de rayon récursif. Cette méthode permet de simuler avec réalisme le phénomène de réflexion.

Au lieu de déterminer la couleur d'un pixel en fonction de la première intersection rencontrée, vous allez faire rebondir le rayon un certain nombre de fois et calculer sa couleur en fonction de tout les objets rencontrés.

La coefficient de réflexion du matériel indique la proportion de lumière réfléchié par la surface: 0 indique aucune réflexion alors que 1 correspond au miroir parfait. Si  $l_i$  est la couleur résultant de l'illumination de l'intersection  $i$  et  $r_i$  le coefficient de réflexion associé à la surface, la couleur prenant en compte les réflexions est la suivante:

$$color_i = l_i * (1 - r_i) + color_{i+1} * (r_i)$$

À chaque itération, vous calculerez le rayon résultant du rebond sur la surface. Sa direction peut être calculée avec la formule suivante: Si  $D_i$  est la direction du rayon à l'itération  $i$  et  $N_i$  la normale à l'intersection calculée, la direction du rayon suivant  $D_{i+1}$  est:

$$D_{i+1} = D_i - 2 * (D_i \cdot N_i) * N_i$$

Implémentez le lancé de rayon récursif dans la fonction `trace_ray` avec un nombre de rebonds maximum fixé à 20.

## 14 Parallélisation du rendu des lignes de l'image

Les processeurs d'aujourd'hui sont en général divisés en plusieurs coeurs chacun capable d'exécuter une suite d'instruction. Cela oblige, s'il veut utiliser toute les capacités d'un processeurs, le programmeur à diviser un problème en sous problèmes à distribuer aux différents coeurs.

Vous allez utiliser `multiprocess.Pool` pour distribuer le rendu des différentes lignes de votre image. Vous trouverez plus d'information sur cette fonction dans la documentation de python: <https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing.pool>

## 15 Classe Triangle

Dans le module `scene`, implémentez la classe `Triangle`. Son constructeur prendra en argument les 3 membres à initialiser:

- `v0`: les coordonnées 3d du premier vertex.
- `v1`: les coordonnées 3d du deuxième vertex.
- `v2`: les coordonnées 3d du troisième vertex.
- `material`: l'objet de type `Material` définissant les propriétés physiques du triangle.

## 16 Intersection Rayon - Triangle

Dans la fonction `intersect`, ajoutez le cas de l'intersection rayon - triangle dans le cas ou le type de l'objet est `Triangle`. Vous pouvez tester si `object` est un triangle avec le test suivant: `if type(object) is Triangle`.

## 17 Bibliographie

- [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))
- <http://virtuelium.free.fr>
- <https://www.itu.dk/courses/IM/Projects/Raytracer/RayNotes.pdf>
- <https://cs.stanford.edu/people/eroberts/courses/soco/projects/1997-98/ray-tracing/applications.html>
- <https://www.ics.uci.edu/~gopi/CS211B/RayTracing%20tutorial.pdf>
- <https://www.cs.cornell.edu/courses/cs4620/2012fa/lectures/36raytracing.pdf>
- [https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection\\_refraction.pdf](https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf)