# TP : Raytracing
# Practical Work: Raytracing

Jonathan Fabrizio
http://jo.fabrizio.free.fr/

## Objectif — Goal

L'objectif de ce T.P. est d'écrire un *raytracer* simple.

The goal of this practical work is to implement a simple raytracer.

## 1 Travail préliminaire — Preliminary work

Pour pouvoir réaliser notre programme nous devons disposer d'un certain nombre de fonctions/classes que nous décrivons dans cette première partie du T.P.. La liste des éléments à implémenter ici n'est qu'une recommandation et cette liste n'est pas exaustive. De même vous pouvez implémenter ces éléments avant de commencer ou au fur et à mesure des questions du T.P.. Dans l'ensemble, tachez de bien faire usage des `namespace`, de `git` et des commentaires.

In order to implement our raytracer, we need a set of functions/classes. We describe them in this first part of this practical work. The provided list of functions to implement is not exhaustive. You can implement these functions before starting or adding them when they are required. During this practical work, use `namespace`, `git` correctly and document you code properly.

### 1.1 Couleur, rayons, vecteurs... — Color, rays and vectors...

Définissez des types pour les couleurs. Définissez aussi des classes pour les points de l'espace et des vecteurs en 3 (ou 4) dimensions (Vous pouvez par exemple definir des `Point3` ou des `Point4` et des `Vector3` ou des `Vector4` selon que vous souhaitez vous placer dans un espace Euclidien ou un espace projectif).

Vous penserez à ajouter des méthodes pour faire des calculs sur ces éléments comme par exemple :

```
Vector3 Vector3::operator*(const float &l) const;
Vector3 Vector3::operator-(const Vector3 &v) const;
...
```

et vous penserez à chaque fois (pour ces classes mais aussi pour les classes à venir) à fournir des fonctions d'affichage comme :

```
std::ostream& operator<<(std::ostream &out, Vector3 &vect);
```

... très utiles pour le *débogage*... Ce n'est pas de l'informatique graphiques mais c'est une bonne pratique.

Define types for colors. Define also classes for points in space and vectors in 3 (or 4) dimensions. You can, for example define `Point3` or `Point4` and `Vector3` or `Vector4` depending on the space you want to use (Euclidean or protective space).

Add the following methods on these classes:

```
Vector3 Vector3::operator*(const float &l) const;
Vector3 Vector3::operator-(const Vector3 &v) const;
...
```

and always add (for these classes and the following classes)

```
std::ostream& operator<<(std::ostream &out, Vector3 &vect);
```

... useful for debugging... It is not computer graphics, but it is a good practice.

### 1.2 Image Image

Faites vous une classe Image rudimentaire contenant (au moins) une largeur, une hauteur et l'ensemble des pixels. Ajoutez une méthode permettant d'enregistrer l'image au format *ppm* - `http://netpbm.sourceforge.net/doc/ppm.html` ou dans le format qui vous arrange (c'est simplement pour faciliter la visualisation).

Implement a simple Image class with (at least) width, height and the pixels. Add a method to save your image in the *ppm* format - `http://netpbm.sourceforge.net/doc/ppm.html` or in the format of your choice (it is to simplify the display of the image).

## 2 La scène The scene

Faites une classe Scene. Cette classe doit comporter une collection d'objets, une collection de sources lumineuses et une caméra. La définition de chacun de ces éléments sera décrite dans les sous-sections suivantes.

Implement a class: Scene. This class embed a collection of objects (the objects of your scene), a collection of light sources and a virtual camera. The definition of all these elements will be provided in the following sections.

### 2.1 Les textures/les matériaux Textures and materials

Faites une classe abstraite Texture_Material avec (au moins) une méthode virtuelle pure qui, étant donné une position sur la surface, renvoie tous les éléments de la texture et du matériaux en ce point ($k_d$, $k_s$...). Faites une classe Uniform_Texture dérivant de la classe Texture_Material. Elle devra implémenter de manière finale la méthode virtuelle. Cette methode renvoie les même paramètres ($k_d$, $k_s$...) en tout les points de la surface.

Add an abstract class for Texture_Material with (at least) one pure virtual method which provide all parameters for the texture and the material ($k_d$, $k_s$...). This method take in argument, a position of a point on the surface. Add the class: Uniform_Texture (which derive from Texture_Material). The implementation of the method will provide constant parameters ($k_d$, $k_s$...) at every points of the surface.

### 2.2 Les objets Objects

Faites une classe abstraite pour les objets. Cette classe doit avoir un Texture_Material et doit contenir (au moins) 3 méthodes virtuelles pures :

- l'une pour déterminer si un rayon (une droite affine exprimée sous forme paramétrique) intersecte l'objet (en renvoyant le bon paramètre),
- une autre pour déterminer la normale à la surface en un point donné de la surface,
- et enfin une pour déterminer la texture en un point donné (qui délègue le travail au Texture_Material).

Faites une classe Sphere qui dérive de cette classe et qui implémente (au moins) ces trois méthodes virtuelles.

Implement an abstract class to manage objects. This class embed a `Texture_Material` and must held at least the 3 following pure virtual methods:

- the first to determine if a ray (a line expressed with a parametric equation) intersects the object (returning the correct value of the parameter in this case),
- the second to provide the normal of the surface at a specific position on the surface,
- and the last one to determine the texture at the given point on the surface (and delegate the process to `Texture_Material`)

Implement the class `Sphere` (derived from your object class) with the implementation of (at least) the 3 previous methods.

### 2.3 La lumière The light

Faites une classe abstraite `Light` qui code une lumière. Dérivez cette classe et écrivez la classe `Point_Light`.

Add an abstract class `Light` to store a light. Implement a class `Point_Light` (derived from `Light`).

### 2.4 La caméra The virtual camera

Définissez une classe Camera. La caméra est repérée dans l'espace par :

- son centre $C$,
- le point $P$ qu'elle regarde,
- un vecteur $\vec{up}$ qui donne la direction du haut.

A cela s'ajoute, une ouverture de champ $\alpha$ en $x$ (angle), une ouverture de champ $\beta$ en $y$ (angle) et un $z_{min}$ (la position du plan image).

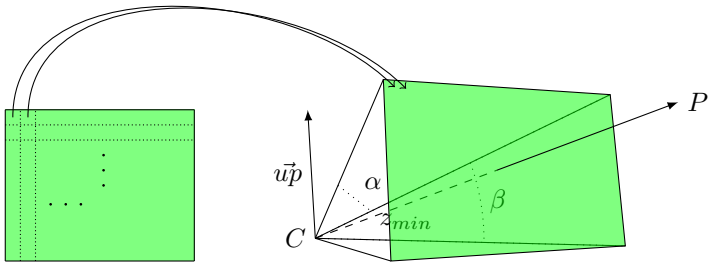
Avec toutes ces informations, il est possible de calculer la position du centre de chaque pixel dans l'espace et d'itérer sur ces pixels.

Define a class Camera. The virtual camera is defined by:

- the center $C$,
- the spotted point $P$,
- an $\vec{up}$ vector which provides the direction of the sky.

You have to add the following parameters: the field of view angle $\alpha$ in the $x$ direction and the the field of view angle $\beta$ in the $y$ direction, $z_{min}$ the position of the image plane.

With all these information, it is possible to compute the position/the center of all pixels in space (and to iterate over them).




## 3 Le moteur The engine

Ecrivez une fonction main qui instancie une scène et qui est prête à démarrer le moteur. Le moteur, en fin de calcul doit renvoyer une image et cette image sera sauvegardée.

Write your main function. This function must create a scene and start the engine. At the end, the engine return an image. This image must be saved.

## 3.1 *Raycasting* *Raycasting*

### 3.1.1 Étape 1 - Les points visibles Step 1 - Visible points

Ecrivez un premier algorithme qui parcours tous les pixels de l'image et qui, pour chaque pixel, lance un rayon et détermine l'objet rencontré le plus proche. Pour chaque point d'intersection, il doit renvoyer la couleur de l'objet sans se soucier de la lumière.

Write a first algorithm, which casts a ray, for each pixel of the image, and determine which object intersects this ray (the nearest one). For every intersection, the ray must get the object color back (without taking lighting into account).

### 3.1.2 Étape 2 - L'apport de la diffusion Step 2 - Diffusion part

Modifier votre algorithme pour évaluer la part diffuse. Vous devez donc, pour chaque point visible, itérer sur les sources (visible ou non) et calculer l'apport de la diffusion.

Modify your algorithm to evaluate the diffuse part of the lighting. You must then, for each visible point, iterate over each light sources (visible or not) and compute the diffuse part.

### 3.1.3 Étape 3 - L'apport de la spécularité Step 3 - Specular part

Modifier votre algorithme pour évaluer la part spéculaire. Vous devez donc, lorsque vous itérez sur les sources (visible ou non) ajouter l'apport spéculaire.

Modify your algorithm to evaluate the specular part of the lighting. When you iterate over each light sources (visible or not) you must compute and add the specular part to your previous computation.

### 3.1.4 Étape 4 - La lumière ambiante (Optionnel) Step 4 - Ambient light (Optional)

Ajoutez éventuellement un apport ambiant à vos calculs.

You may add ambient light to your scene (and then at every visible points).

## 3.2 *Raytracing*

### 3.2.1 Étape 5 - La réflexion

Modifier votre algorithme pour tenir compte de la réflexion. Relancez donc un rayon réfléchis et recommencez tous les calculs pour ce rayon. Ajoutez sa contribution dans le calcul de l'illumination d'origine.

Recommencez récursivement l'ensemble des traitements sur le rayon réfléchis et mettez une borne à votre nombre d'itérations (5 ou 6 par exemple).

Modify your algorithm to compute reflection. When computing the illumination to a given point of your surface, cast the reflected ray and compute the illumination on this ray. Add the contribution of this ray to your point.

Recompute recursively all the lighting computation onto this reflected ray. Set up a boundary on the number of allowed reflections (5 or 6 for example).

### 3.2.2 Étape 6 - Les ombres

Modifier votre algorithme pour gérer les ombres. Vous devez donc éliminer du calcul les sources qui sont masquées par un autre objet.

Modify your algorithm to manage shadows. You have to discard light sources which are not visible (occluded by another object) from the considered point over the surface.

# 4 Pour aller plus loin

Si vous en êtes arrivé là, c'est déjà pas mal ! Pour allez plus loin, vous pouvez ajouter des fonctionnalités comme l'anti-aliasing, les ombres douces ou les anaglyphes. Si vos classes d'origines sont bien pensées, les extensions ne devraient pas être trop difficiles à faire.

If you have reached this point, congratulations! To go further, you may add other functions, for example: anti-aliasing, soft shadows or anaglyphs. If your design is correct (class hierarchy...), these functions are easy to implement.