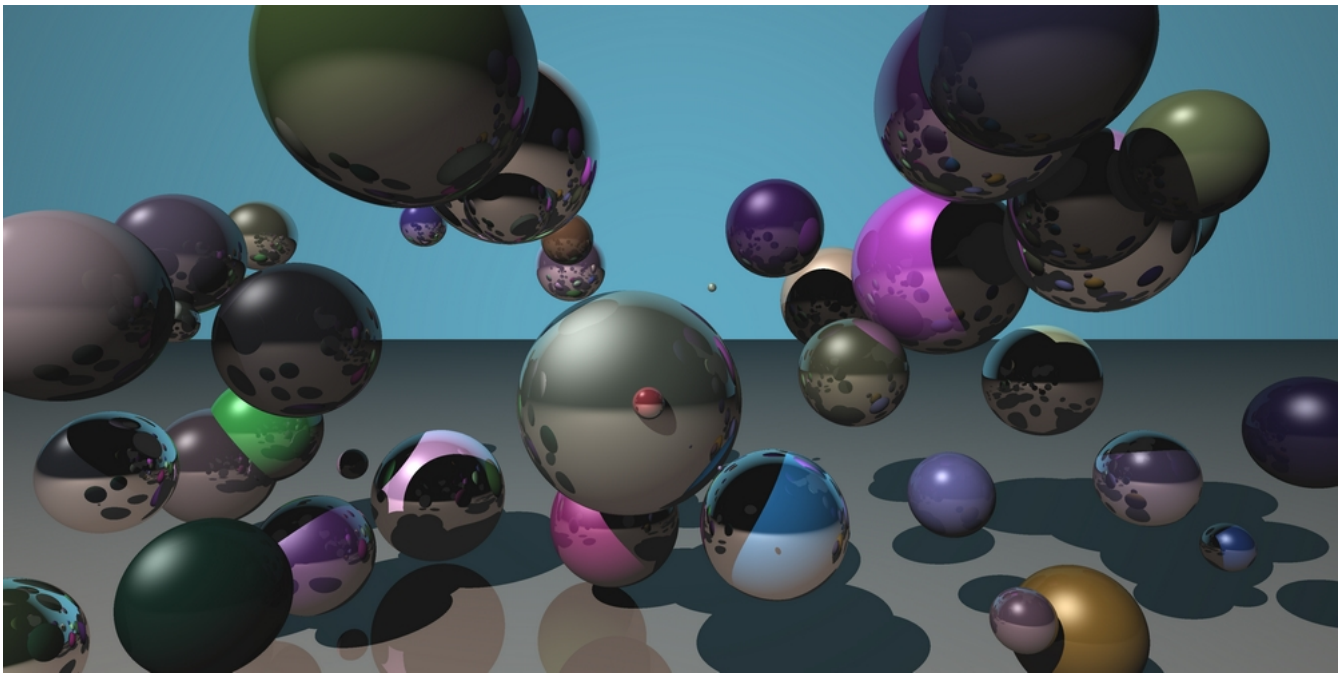
Raytracting

« Nous ne prétendons pas à réussir à représenter l'objet exactement tel qu'il apparaîtrait dans la réalité, avec sa texture, son ombre etc. Nous espérons seulement obtenir une image suffisamment proche de l'objet réel pour donner une certaine impression de réalisme. » Bui Tuong Phong



Sommaire

Préambule :

Le but de ce document est de présenter au lecteur les rudiments d'une technique de rendu graphique en trois dimensions appelée raytracing afin qu'il soit lui-même capable d'obtenir des images élémentaires comme celle en page de garde.

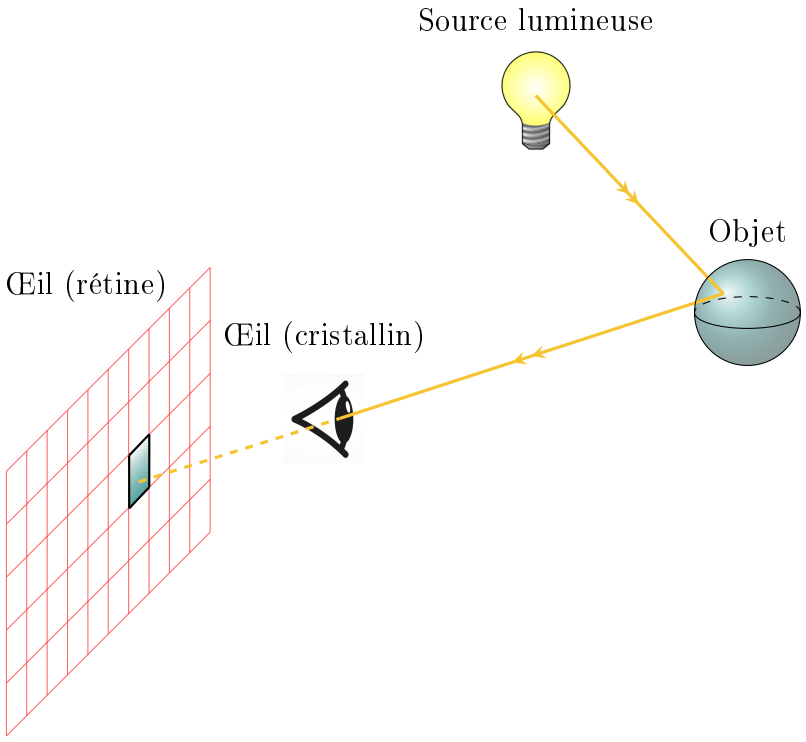
Pour cela les pré-requis mathématiques nécessaires sont la résolution d'équations du second degré, le produit scalaire dans l'espace ainsi qu'un minimum de motivation. Ces dernières notions interviendront pour appuyer des en raisonnant qualitativement sur les trajectoires que peut prendre la lumière depuis sa source pour atteindre un observateur donné.

Le modèle obtenu sera alors ensuite implémenté en langage python, pour faciliter cela il est indispensable de garder le fichier vecteur.py joint au document (voir **II.3**).

Les diverses idées, réflexions, calculs ou relations introduites ici seront motivées et expliquées de manière détailles, en revanche les notions algorithmiques basiques de boucles, instructions conditionnelles, fonctions, etc...seront considérées comme déjà acquises par le lecteur. Les quelques principes optiques appuyant les raisonnements faits sont rappelés en Appendice 1 mais leur méconnaissance n'entrave pas la bonne compréhension du présent document.

I Suivre la lumière

I.1 À l'œil nu :



Rappelons sommairement le fonctionnement de la vision humaine à partir du schéma ci-dessus : il y a autour de nous diverses sources émettant ou réfléchissant des rayons lumineux (Soleil, lampes, vitres, ...), ces derniers étant partiellement reflétés sur tout ce qui nous entoure jusqu'à ce que certains trouvent leur chemin jusqu'à l'œil. Ils sont alors captés par le cristallin et projetés au fond de l'œil, sur une paroi interne appelée la rétine. Cette dernière est constituée de cellules photosensibles qui transmettent l'information jusqu'au cerveau, lequel la traite de sorte à constituer les images que nos yeux perçoivent.

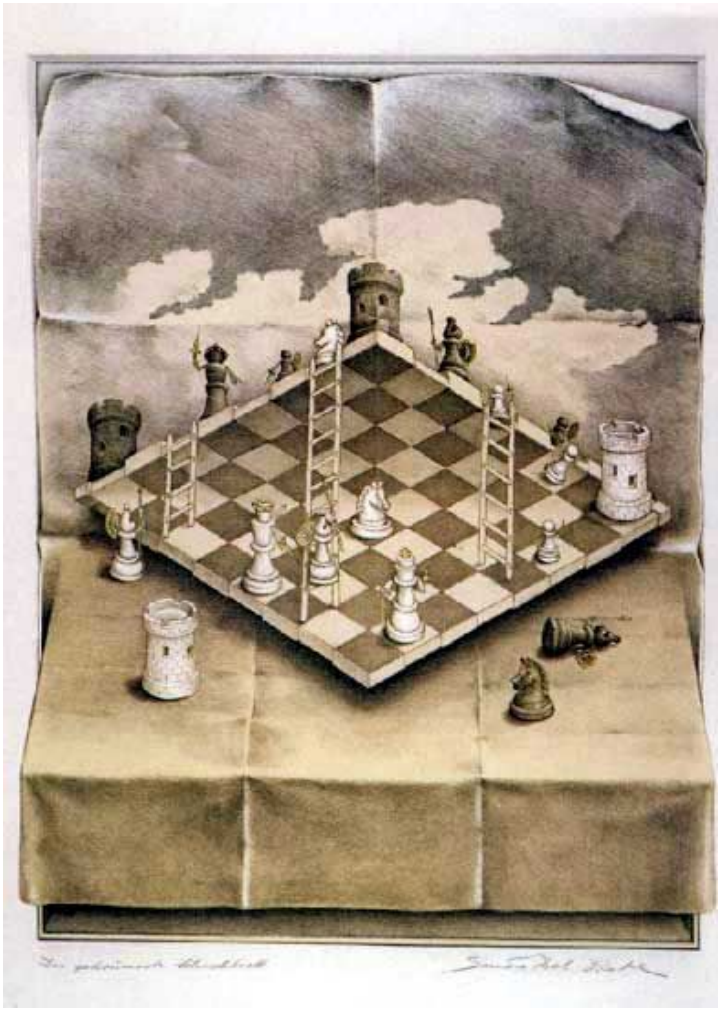
Ainsi l'image numérique que nous cherchons à produire peut-être assimilée à celle qui se forme sur notre rétine, le problème est donc le suivant : étant donné un observateur et une scène constituée de divers objets éclairés par une source de lumière, à quel point chaque objet est-il éclairé ? Autrement dit quelles sont les parties de la scène redirigeant au mieux les rayons émis par la source vers l'œil de l'observateur ?

Une image numérique est une grille de pixels, chacun d'entre eux étant doté une couleur. Le principe du « raytracing » est de retracer, pour chaque pixel, le cheminement du rayon lumineux qui l'a atteint afin de lui attribuer la couleur de l'objet qui l'a reflété.

Notre but sera donc de déterminer en fonction de cet objet quelle couleur attribuer à chaque pixel, mais aussi avec quelle intensité, selon sa position par rapport à la source de lumière. La clef pour se rapprocher d'une image réaliste sera d'imaginer une retranscription des différents jeux de lumière (réflexions, ombres, transparence, ...) en accord avec quelques postulats raisonnablement choisis.

Le résultat sera bien entendu une image en deux dimensions, si les couleurs sont suffisamment proches de celles d'une scène réelle similaire, notre cerveau fera le reste et conférera à la scène une illusion de profondeur ! À noter en effet que nous ne percevons pas directement en trois dimensions, c'est encore une fois le cerveau qui se charge de nous donner une sensation de profondeur à partir de différentes informations, et en particulier la luminosité, nous trichons donc à peine.

Ces réflexes d'interprétations que nous avons ont inspiré de nombreux artistes à qui l'on doit de fascinantes illusions d'optique. Il est en effet possible, en jouant sur l'image projetée sur la rétine de créer des visions que nous comprenons spontanément en relief bien qu'elles n'aient physiquement aucun sens, ou au contraire de donner lieu à des perceptions excessivement réalistes. Notre objectif est ici de se rapprocher de ce dernier cas en considérant des objets simples tels que des sphères.



« The Folded Chess Set » de Sandro Del-Prete.

« Art Imitating Life Imitating Art Imitating Life », Trompe l'œil mural de John Pugh.

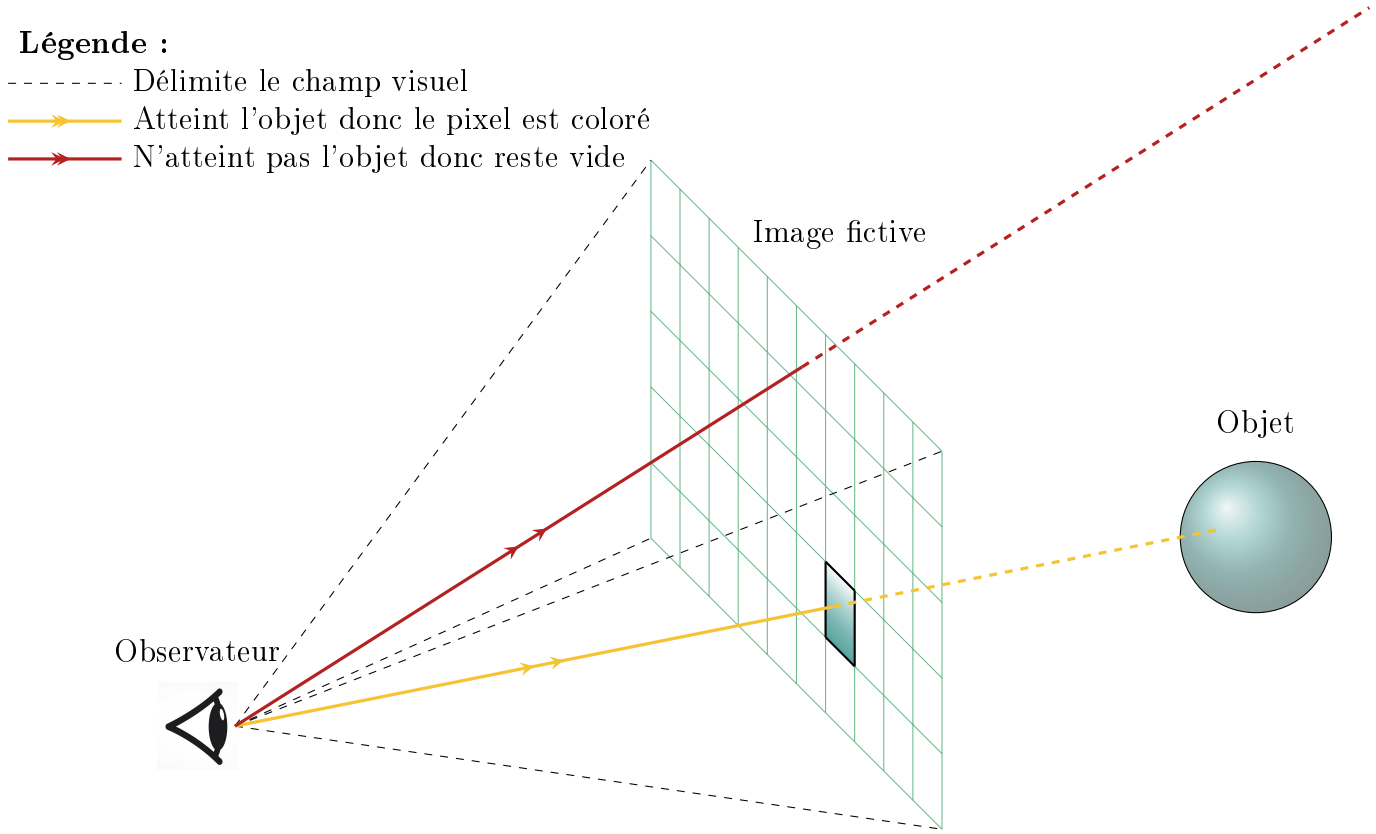
I.2 À l'envers

Le problème est posé : pour une source donnée il faut étudier les trajectoires des rayons lumineux émis dans toutes les différentes directions possibles afin de déterminer ceux qui seront reflétés de l'objet à la rétine.

Un peu...fastidieux n'est-ce pas ? Cela reviendrait en théorie à examiner une infinité de possibilités afin de colorer une grille tout ce qu'il y a de plus finie ce qui semble quelque peu excessif. Heureusement il y a une manière bien plus simple de voir les choses : il suffit de prendre le problème à l'envers.

En effet, imaginons que l'image que l'on cherche à remplir correspond directement au champ visuel de l'observateur plutôt qu'à sa rétine, il suffit alors de vérifier pour chaque pixel de cette grille fictive si la droite qui le relie à l'observateur intersecte un objet de la scène, le cas échéant on attribue sa couleur au pixel avec une teinte claire ou foncée selon sa position par rapport à la principale source lumineuse.

Tout ce passe donc comme si l'on cherchait à suivre la lumière en partant de l'observateur pour remonter jusqu'à la source plutôt que de le faire dans l'autre sens, mais les deux situations sont équivalentes d'après le principe du retour inverse de la lumière (voir Appendice 2).



Ainsi notre modélisation peut à présent se décomposer en deux étapes :

1. Pour un pixel donné, déterminer si le rayon lumineux correspondant atteint un objet et le cas échéant lui attribuer sa couleur de base.
2. Une fois la couleur du pixel déterminée, établir ensuite sa teinte selon la position du point de l'objet correspondant par rapport à la source lumineuse d'une manière qui reste à définir.

Nous traiterons le premier point dans la partie **II** mais n'aborderons que le second qu'en partie **III.**
Récapitulons avant cela les paramètres sur lesquels nous avons la main :

- La position de l'observateur que l'on définira comme étant l'origine du repère.
- Le champ visuel de l'observateur qui est en fait lié à la taille de l'image finale obtenue et à la distance entre ce dernier et l'image fictive.
- Les différents objets présents dans la scène que l'on veut représenter, définis par leurs positions, leurs formes ainsi que leurs couleurs ou autres caractéristiques.
- La position et la couleur de la (ou des) source(s) de lumière.

II Cadre, outils divers et résolution mathématique

Comme annoncé, nous nous efforcerons dans cette seconde partie de déterminer la couleur à attribuer à chaque pixel. Pour cela, il sera nécessaire de poser le problème de manière plus précise, de le résoudre mathématiquement et enfin de présenter les divers outils informatiques qui nous permettront d'implémenter notre solution et d'obtenir une première image.

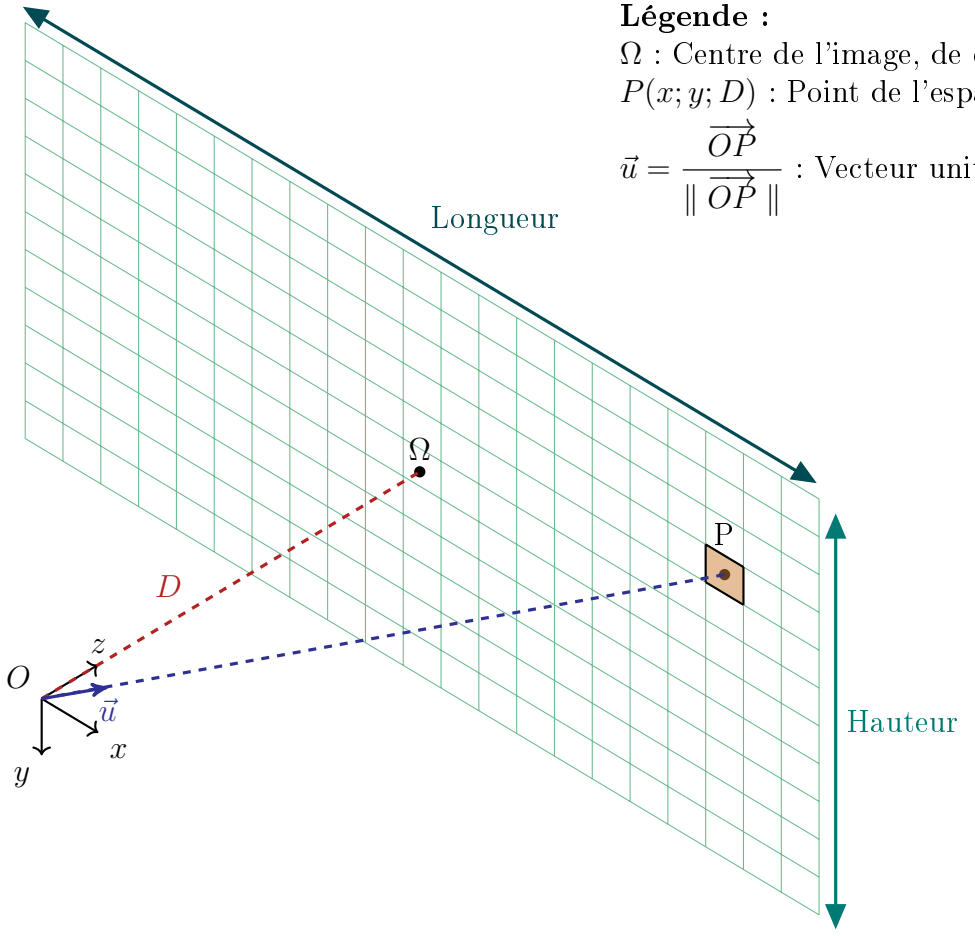
II.1 Des pixels dans l'espace

L'enjeu est ici de déterminer une relation entre l'observateur et les pixels de la grille fictive.

Dans toute la suite on assimilera l'observateur à un point O situé à l'origine d'un repère orthonormé de l'espace représenté ci-dessous.

On suppose également que l'observateur regarde dans la direction $[Oz)$, on placera ainsi notre image fictive à une distance $D > 0$ en suivant cet axe de sorte que celle-ci soit incluse dans le plan $z = D$.

Ainsi la longueur de l'image, sa largeur et la valeur de D sont des paramètres qui pourront être ajustés selon le résultat que l'on cherche à produire.

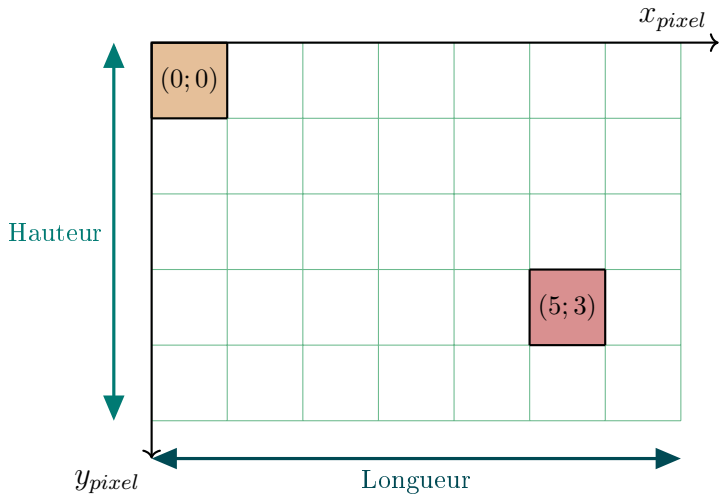


Légende :
Ω : Centre de l'image, de coordonnées $(0;0;D)$ dans notre repère.
$P(x;y;D)$: Point de l'espace associé à un pixel donné.
$\vec{u} = \dfrac{\overrightarrow{OP}}{\| \overrightarrow{OP} \|}$: Vecteur unitaire suivant la direction $[OP)$.

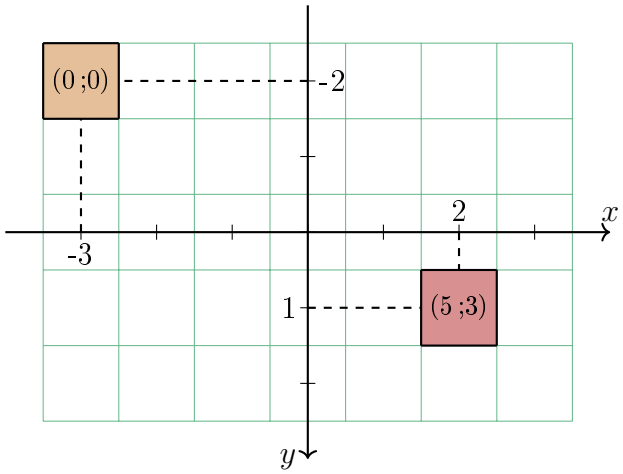
Sur une image numérique, les pixels sont repérés selon leur colonne, puis leur ligne, ainsi le pixel de coordonnées $(0;0)$ est celui se situant en haut à gauche de l'image.

Plus généralement le pixel $(x;y)$ est à l'intersection entre la colonne x et la colonne y. Il faudra donc prendre soin de distinguer les coordonnées des pixels au sein même de l'image numérique de celles de leur position fictive dans l'espace et d'établir une correspondance entre les deux.

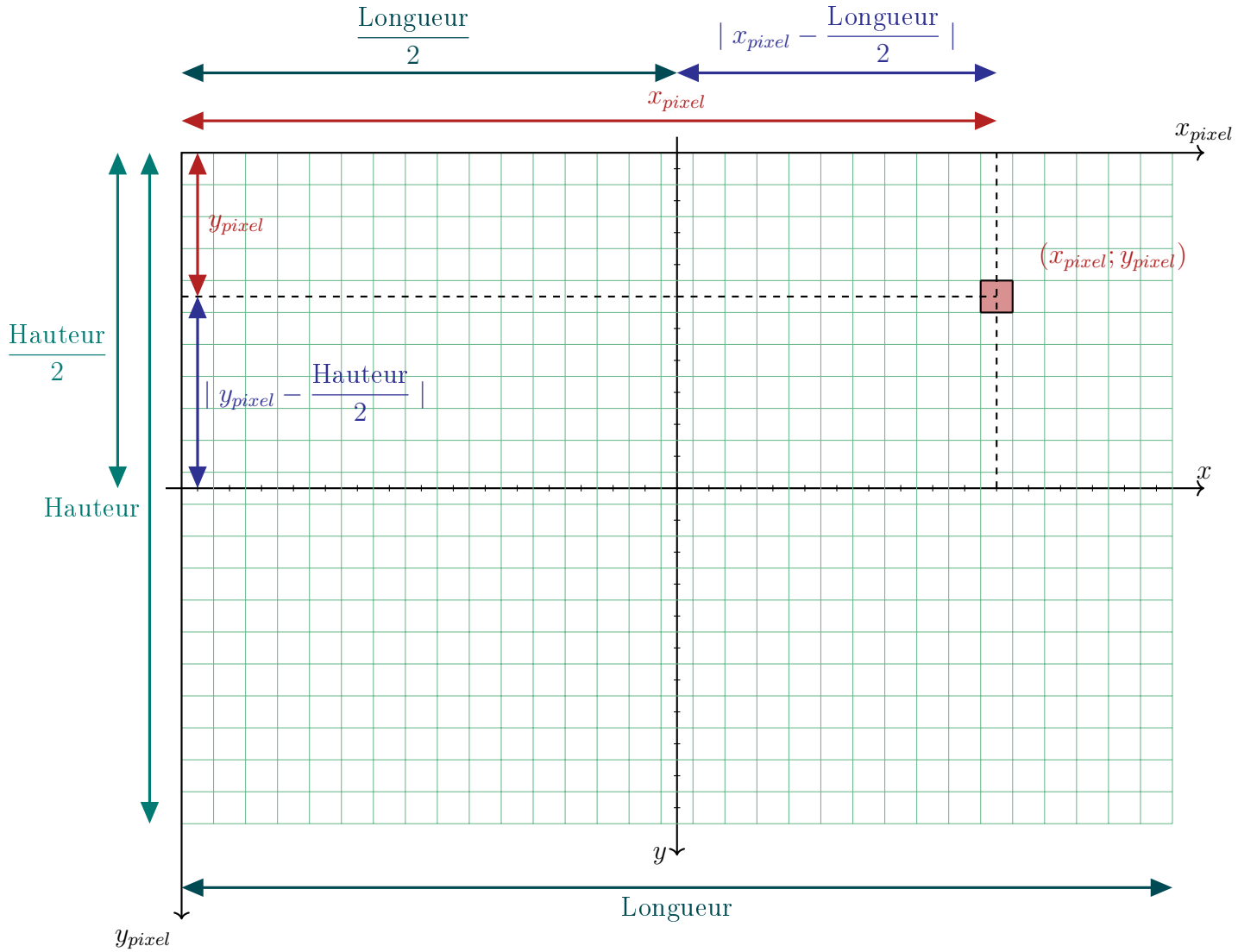
Coordonnées image numérique :



Coordonnées dans le repère :



Pour passer de l'un à l'autre il suffit ainsi d'effectuer une translation de vecteur $\vec{t}\begin{pmatrix}\frac{\text{Longueur}}{2}\\ \frac{\text{Hauteur}}{2}\end{pmatrix}$, la figure suivante résume le changement de coordonnées de l'image numérique au repère :



Nous pouvons alors définir la fonction suivante permettant d'obtenir les coordonnées dans l'espace d'un pixel donné : $(x_{pixel}; y_{pixel}) \mapsto \left(x_{pixel} - \frac{\text{Longueur}}{2}; y_{pixel} - \frac{\text{Hauteur}}{2}; D\right)$.

Ce qui correspond à la fonction suivante sur python, prenant en entrée un pixel et renvoyant le vecteur correspondant :

```
# renvoie le vecteur associe au pixel (x;y)
def direction(x, y) :
    return vecteur(x - longueur / 2, y - hauteur / 2, D)
```

II.2 Une question d'intersection

Nous avons à présent à faire à un problème mathématique bien plus classique ; étant donné :

- Un observateur situé en $O(0;0;0)$.
- Un pixel situé en $(x;y)$ sur l'image assimilable au point $P\left(x-\dfrac{\text{Longueur}}{2};y-\dfrac{\text{Hauteur}}{2};D\right)$.
- Une sphère $\mathscr{S}$ de centre Ω et de rayon r.

L'intersection entre la droite (OP) et $\mathscr{S}$ existe-t-elle ? Si oui à quelle(s) condition(s) ?

Soit M un point de l'espace, M est situé sur la droite (OP) de vecteur directeur $\overrightarrow{OP}$ si et seulement si $\overrightarrow{OM}$ est colinéaire à $\overrightarrow{OP}$. Autrement dit, si et seulement si il existe un réel t tel que : $\overrightarrow{OM}=t\overrightarrow{OP}$.

Nous utiliserons par la suite la définition suivante pour caractériser la sphère :

Définition (Sphère).

On appelle sphère de centre Ω et de rayon r l'ensemble des points de l'espace distants de r par rapport à Ω.
Autrement dit il s'agit de l'ensemble des points M tels que $\Omega M=r$.

On a donc par définition de $\mathscr{S}$:

$M\in\mathscr{S}\Leftrightarrow \Omega M=r$

$M\in\mathscr{S}\Leftrightarrow \Omega M^2=r^2$ (car ΩM et r sont des longueurs, donc positives)

$M\in\mathscr{S}\Leftrightarrow \overrightarrow{\Omega M}\cdot\overrightarrow{\Omega M}=r^2$ ($\overrightarrow{u}^2=\vec{u}\cdot\vec{u}$)

$M\in\mathscr{S}\Leftrightarrow (\overrightarrow{\Omega O}+\overrightarrow{OM})\cdot(\overrightarrow{\Omega O}+\overrightarrow{OM})=r^2$ (Par la relation de Chasles)

$M\in\mathscr{S}\Leftrightarrow \overrightarrow{\Omega O}\cdot\overrightarrow{\Omega O}+2\overrightarrow{\Omega O}\cdot\overrightarrow{OM}+\overrightarrow{OM}\cdot\overrightarrow{OM}=r^2$

$M\in\mathscr{S}\Leftrightarrow \Omega O^2+2\overrightarrow{\Omega O}\cdot\overrightarrow{OM}+OM^2=r^2$

Ainsi : $M\in(OP)\cap\mathscr{S}\Leftrightarrow$ Il existe $t\in\mathbb{R}$ tel que : $\overrightarrow{OM}=t\overrightarrow{OP}$ et $\Omega O^2+2\overrightarrow{\Omega O}\cdot(t\overrightarrow{OP})+(tOP)^2=r^2$

$M\in(OP)\cap\mathscr{S}\Leftrightarrow$ Il existe $t\in\mathbb{R}$ tel que : $\Omega O^2+2\overrightarrow{\Omega O}\cdot(t\overrightarrow{OP})+(tOP)^2=r^2$

$M\in(OP)\cap\mathscr{S}\Leftrightarrow$ Il existe $t\in\mathbb{R}$ tel que : $\Omega O^2+2t\overrightarrow{\Omega O}\cdot\overrightarrow{OP}+t^2OP^2=r^2$

$M\in(OP)\cap\mathscr{S}\Leftrightarrow$ Il existe $t\in\mathbb{R}$ tel que : $(OP^2)t^2+(2\overrightarrow{\Omega O}\cdot\overrightarrow{OP})t+\Omega O^2-r^2=0$

Notre problème est donc équivalent à la recherche de solution(s) à l'équation du second degré en t : $at^2+bt+c=0$, où $a=OP^2$, $b=2\overrightarrow{\Omega O}\cdot\overrightarrow{OP}$ et $c=\Omega O^2-r^2$.

Remarquons que l'on a $a=OP^2\neq 0$ car $D>0$, il s'agit donc bien d'une équation du second degré.

À noter que a, b et c sont parfaitement calculables à partir des données dont nous disposons puisque les coordonnées de chacun des points sont connues, donc celles des vecteurs également, desquelles nous pouvons aisément déduire les différentes longueurs ainsi que la valeur du produit scalaire puisque le repère est orthonormé.

Le raisonnement précédent reste valable en prenant $\vec{u}=\dfrac{\overrightarrow{OP}}{\|\overrightarrow{OP}\|}$ comme vecteur directeur de la droite (OP) : il suit bien la même direction et le même sens et présente l'avantage d'être unitaire.

En effet : $\|\vec{u}\|=\left\|\dfrac{\overrightarrow{OP}}{\|\overrightarrow{OP}\|}\right\|=\dfrac{\|\overrightarrow{OP}\|}{\|\overrightarrow{OP}\|}=1$.

L'équation devient alors, en cherchant $\overrightarrow{OM} = t\vec{u}$ au lieu de $t\overrightarrow{OP}$:

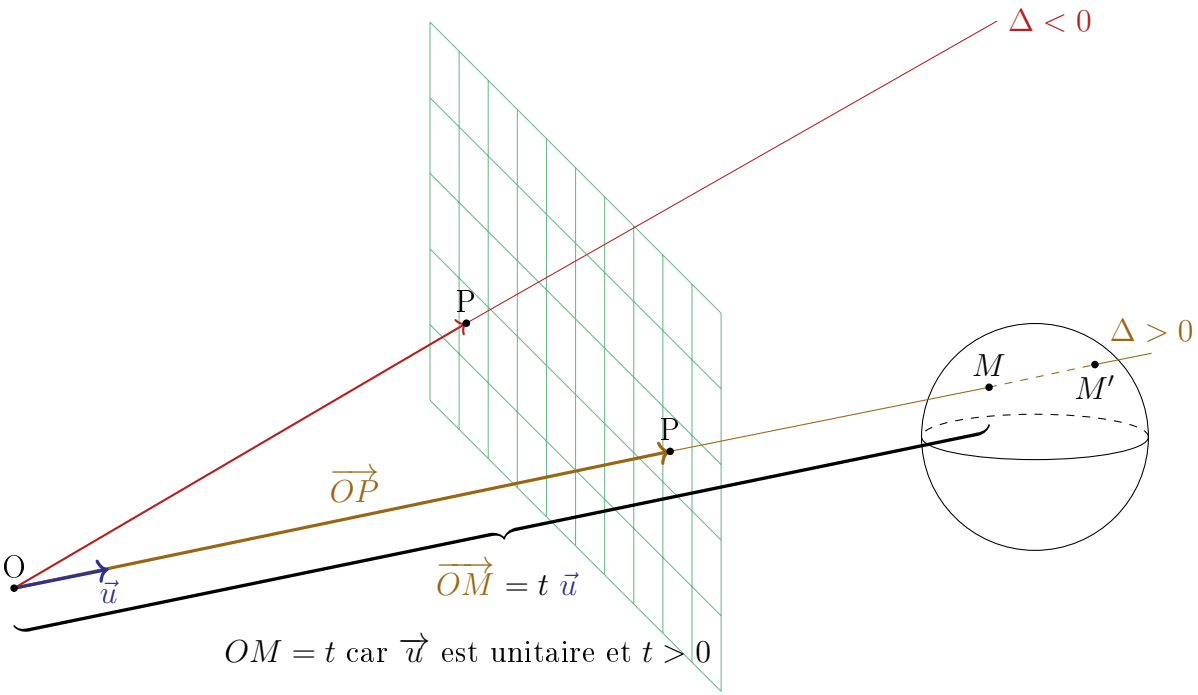
$$\| \vec{u} \| t^2 + 2\overrightarrow{\Omega M} \cdot \vec{u}t + \Omega O^2 - r^2 = 0 \Leftrightarrow t^2 + 2\overrightarrow{\Omega M} \cdot \vec{u}t + \Omega O^2 - r^2 = 0$$

Cela simplifie légèrement les coefficients puisque l'on obtient : $a = 1$, $b = 2\overrightarrow{\Omega M} \cdot \vec{u}$ et $c = \Omega O^2 - r^2$.

On sait quoiqu'il en soit déterminer le nombre exact de solutions d'une telle équation en calculant le discriminant $\Delta = b^2 - 4ac$ et en examinant son signe.

Plusieurs cas de figure se présentent alors :

- Si $\Delta < 0$, l'équation n'admet pas de solution et l'intersection est vide.
- Si $\Delta = 0$, l'équation admet une unique solution, la droite et la sphère admettent un unique point d'intersection, ainsi la droite est tangente à la sphère.
- Si $\Delta > 0$, l'équation admet deux solutions, il y a donc deux points d'intersection (la droite rentre dans la sphère et en ressort).



Attention même dans le cas où il y a existence de solutions, il faut de plus vérifier ici qu'elles sont bien positives. Une valeur de t négative signifierait que la sphère est située derrière l'observateur, il ne peut donc pas la voir.

Ainsi dans le cas où il y a deux racines de signes opposés, nous retiendrons seulement celle qui est positive. De plus si l'on trouve deux racines positives seule celle correspondant au point entrant, donc la plus petite, sera retenue car il s'agit du point de la sphère aperçu par l'observateur.

De ce fait dans un but strictement pratique nous modifions légèrement notre raisonnement dans la fonction **intersection** qui suit. À l'aide des commandes **min** et **max**, on fait en sorte que la fonction renvoie ainsi 0 dans les cas où il n'y a pas d'intersection visible. On néglige d'ailleurs ainsi délibérément le cas où l'intersection est confondue avec l'observateur.

```
def intersection(u, OC, r):
        a = 1
        b = - 2 * scalaire(u, OC)
        c = norme(OC)**2 - r**2
        delta = b**2 - 4 * a * c
        if delta == 0:
            return max(- b / (2 * a),0)
        elif delta > 0 :
            x1 = (- b - racine(delta)) / (2 * a)
            x2 = (- b + racine(delta)) / (2 * a)
```

```
            if min(x1, x2) >= 0 :
                return min(x1, x2)
            else :
                return max(0, x1, x2)
        else:
            return 0
```

II.3 Quelques fonctions en python

La fonction intersection vient d'être définie, il est cependant nécessaire pour qu'elle puisse être utilisée de clarifier certains points, en particulier les modules de python que nous allons utiliser, comment définir des points et des vecteurs de l'espace ainsi que les fonctions **scalaire** et **norme** mentionnées précédemment.

- Nous définirons la plupart des outils dont nous aurons besoin, ainsi les seuls pré-requis pour ce programme sont la fonction **racine** du module math, **numpy** pour gérer les images numériques sous forme de tableaux, **imsave** pour sauvegarder l'image finale ainsi que la classe prédéfinie nommée **vecteur** afin de pouvoir manipuler les vecteurs à l'aide des opérations mathématiques usuelles (voir Appendice 2 pour plus de détails à ce sujet). **Attention il est primordial que vecteur.py soit dans le même dossier que le programme que nous allons écrire.**

```
from math import sqrt as racine
import numpy as np
from matplotlib.image import imsave
from vecteur import *
```

- Pour définir un vecteur $\vec{u}\begin{pmatrix}x\\y\\z\end{pmatrix}$, nous utiliserons la classe vecteur importée précédemment de la manière suivante : $u = \text{vecteur}(x, y, z)$
  La classe vecteur permet également de récupérer facilement chacune des coordonnées de $\vec{u}$ puisque $u.x$ renvoie x, $u.y$ renvoie y et $u.z$ renvoie z.
- Les coordonnées d'un point M seront assimilées à celle du vecteur $\overrightarrow{OM}$.
- Les couleurs seront définies à l'aide de triplets (R, B, V) que l'on définira sur python également comme des vecteurs afin de réaliser des opérations avec.
- Comme l'étude se déroule en repère orthonormé, nous définirons le produit scalaire de $\vec{u}\begin{pmatrix}x\\y\\z\end{pmatrix}$ et $\vec{v}\begin{pmatrix}x'\\y'\\z'\end{pmatrix}$ à l'aide des coordonnées : $\vec{u}\cdot\vec{v} = xx' + yy' + zz'$.

```
def scalaire(u, v) :
    return u.x * v.x + u.y * v.y + u.z * v.z
```

- Il est alors aisé de définir la norme d'un vecteur $\vec{u}$ comme étant la racine carrée de $\vec{u}\cdot\vec{u} = \| \vec{u} \|^2$.

```
# definit la norme du vecteur u
def norme(u) :
    return racine(scalaire(u,u))
```

- Ainsi que nous l'avons vu, pour rendre un vecteur non nul unitaire, il suffit de le diviser par sa norme, ce qui revient à diviser chacune de ses coordonnées comme le fait cette fonction :

```
# transforme un vecteur en un vecteur de norme 1
# suivant la meme direction dans le meme sens
def unitaire(u) :
    return vecteur(u.x / norme(u), u.y / norme(u), u.z / norme(u))
```

III Un objet à la fois . . .

Puisque la résolution mathématique du problème a été effectuée, nous pouvons dans un premier temps en effectuer l'implémentation en python.

Cela nous fournira un premier résultat que nous nous efforcerons par la suite d'améliorer en cherchant, comme annoncé en **I.**, à prendre en compte une source de lumière de sorte à .

III.1 Intersection et coloration

Une fois que les fonctions précédentes ont été définies, il ne reste plus qu'à choisir la taille de l'image et une valeur pour D, puis à créer l'image correspondante.

```
longueur = 1000
hauteur = 500
D = 500

image = np.zeros((hauteur, longueur, 3), dtype = np.uint8)
image.fill(50) # remplit prealablement avec la couleur (100,100,100) pour davantage de
    visibilite
```

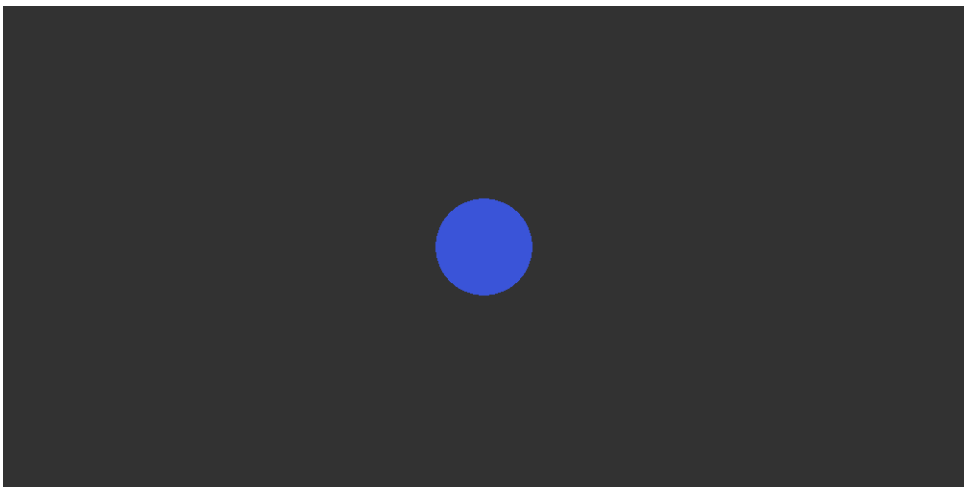
Ce à quoi l'on rajoute les caractéristiques de la sphère que l'on veut représenter. Il s'agira par exemple ici de la sphère de centre $(2; -1; 10)$ et de rayon 1 à laquelle on attribuera la couleur $(56, 84, 216)$:

```
centre = vecteur(0, 0, 10)
rayon = 1
couleur = vecteur(58, 84, 216)
```

On définit alors la fonction **coloration** prenant l'image vide en entrée. Elle est constituée de deux boucles parcourant les pixels de l'image et attribuant une couleur à chaque pixel en fonction de si le rayon émis intersecte la sphère choisie ou pas.

```
def coloration(image) :
    for x in range(longueur):
        for y in range(hauteur) :
            u = unitaire(direction(x, y))
            if intersection(u, centre, rayon) != 0 :
                image[y, x] = couleur.rgb()
    return image

image = coloration(image)
imsave("cercle.png", image)
```

La commande couleur**.rgb()** permet de convertir le vecteur couleur au format nécessaire. Elle est définie dans vecteur.py et l'explication de son fonctionnement est ici peu pertinent (voir éventuellement Appendice 2). On obtient alors l'image suivante, pour l'instant peu réaliste mais qui constitue tout de même notre tout premier pas en ce sens.



La sphère est ainsi retranscrite sur notre image par un cercle bleu qu'il va nous falloir nuancer en fonction de la position de la source de lumière, ce qui sera l'objet de la partie suivante.

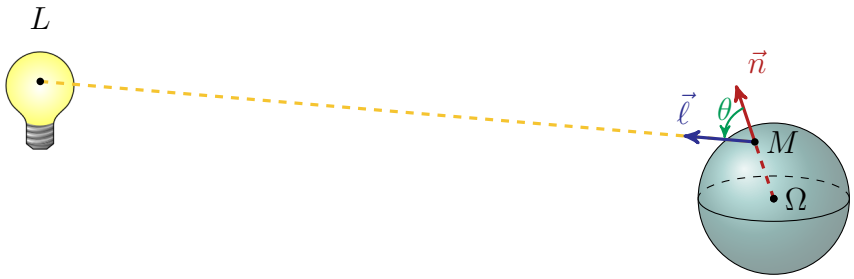
III.2 Que la lumière soit

Partons ici d'une remarque simple : un point situé sur une surface sera d'autant plus « éclairé » que la direction du vecteur le reliant à la source lumineuse est proche de celle de la normale.

Le fait que la luminosité soit plus élevée aux alentours de midi en est un exemple connu et concret, en effet le soleil étant notre source de lumière principale cela signifie tout simplement qu'il s'agit du moment de la journée où ses rayons nous parviennent perpendiculairement.

Il devient alors nécessaire d'introduire les notations suivantes :

- M : point en lequel on cherche à déterminer la couleur suivant la position d'une source de lumière L.
- $\vec{n}$: vecteur normal unitaire (dirigé côté source) en M.
- $\vec{\ell} = \dfrac{\overrightarrow{ML}}{\| \overrightarrow{ML} \|}$: vecteur unitaire de M vers L.
- $\theta = (\vec{n}, \vec{\ell})$: angle entre les deux vecteurs, plus il est grand et moins ils sont « superposés ».

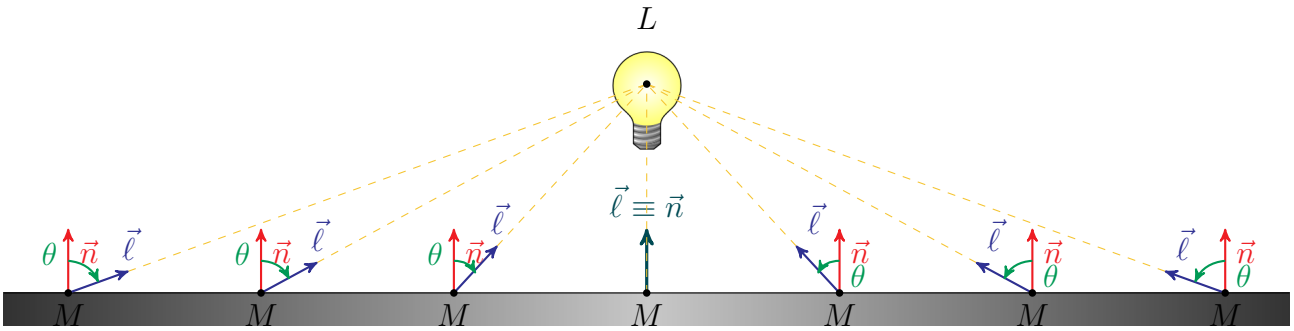


D'après la remarque précédente, déterminer la coloration reviendrait à déterminer l'angle θ. Il s'agit d'un problème qu'il est tout à fait possible de résoudre, cela dit il existe une manière moins fastidieuse de procéder. En effet, considérons le produit scalaire des vecteurs $\vec{n}$ et $\vec{\ell}$:

$$\vec{n} \cdot \vec{\ell} = \| \vec{n} \| \times \| \vec{\ell} \| \times \cos(\vec{n}, \vec{\ell}) = \cos(\theta)$$

$\vec{n} \cdot \vec{\ell}$ dépend donc ici uniquement de θ, de sorte que plus la direction et le sens de $\vec{\ell}$ est proche de $\vec{n}$ et plus $\vec{n} \cdot \vec{\ell}$ est proche de 1 et réciproquement.

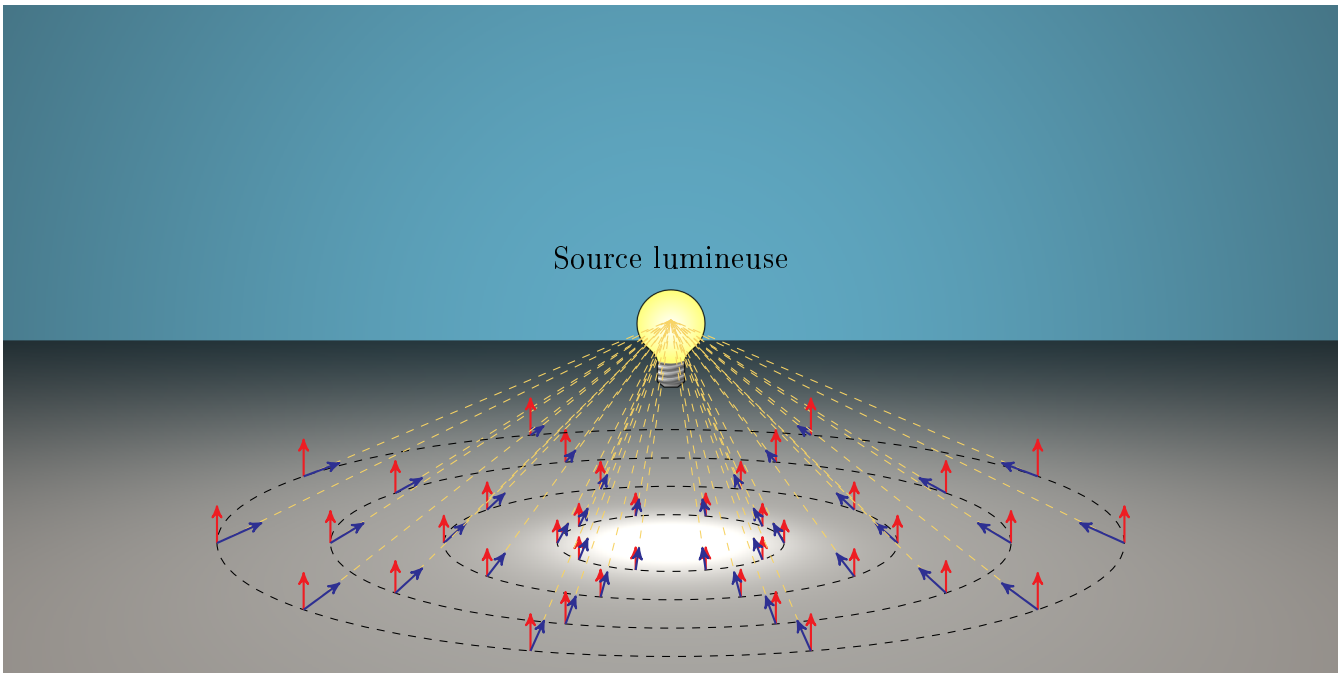
Inversement plus θ s'approche d'un angle droit et plus $\vec{n} \cdot \vec{\ell}$ est proche de 0, ce que l'on peut par ailleurs visualiser plus facilement en deux dimensions avec le schéma suivant :



θ	$-\frac{\pi}{2}$	0	$\frac{\pi}{2}$
$\vec{n} \cdot \vec{\ell}$	0	$\nearrow 1 \searrow$	0

Comme on peut le voir, $\vec{n} \cdot \vec{\ell}$ varie entre 0 et 1, c'est donc le candidat idéal pour faire office de « coefficient chromatique » en définissant : $couleur_{finale} = (\vec{n} \cdot \vec{\ell}) \times couleur_{objet}$.

Ainsi lorsque $\vec{\ell} = \vec{n}$, on retranscrit la totalité de la couleur de l'objet mais plus l'angle entre les deux vecteurs est grand et plus la proportion retranscrite est faible. Ci-après un aperçu permettant de visualiser ce principe en dimension trois sur une surface plane :



L'objectif se précise mais le problème n'est pas encore tout à fait résolu : il est clair à présent qu'il nous faut pouvoir calculer $\vec{n} \cdot \vec{\ell}$ sans passer par θ, ce qui implique de connaître les coordonnées des deux vecteurs unitaires dans l'espace. C'est cependant chose aisée puisque $\vec{\ell} = \dfrac{\overrightarrow{ML}}{\| \overrightarrow{ML} \|}$ et dans le cas de la sphère $\vec{n} = \dfrac{\overrightarrow{\Omega M}}{\| \overrightarrow{\Omega M} \|}$, de plus les coordonnées des points Ω et L sont connues puisque c'est nous qui choisissons où les placer.

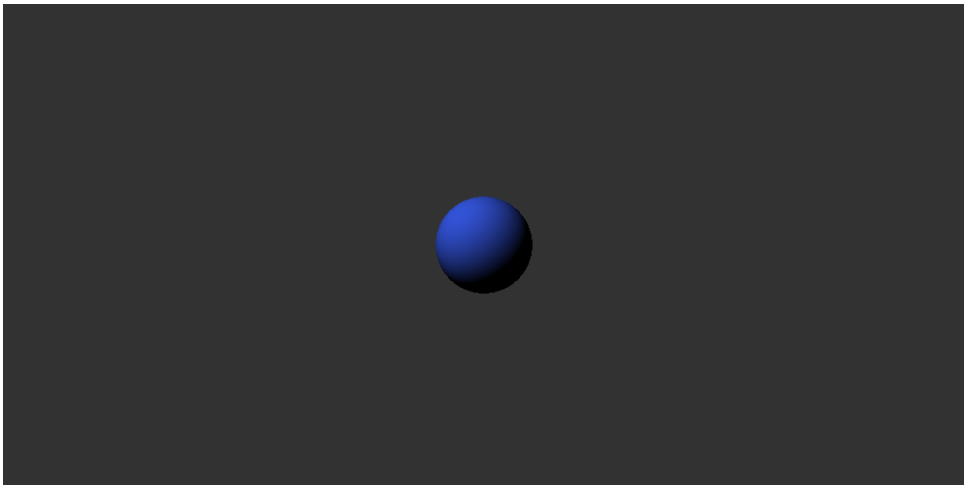
Par ailleurs nous obtenons les coordonnées de M dans le cas de la sphère grâce au raisonnement présenté en **II.2** et à la fonction **intersection**.

Remarquons également que $\vec{n} \cdot \vec{\ell}$ reste positif tant que $\vec{n}$ est orienté « côte source », dans le cas contraire nous pouvons considérer que le point n'est pas éclairé, on utilisera donc plutôt $\mathbf{max}(\vec{n} \cdot \vec{\ell}, 0)$ dans le programme pour s'épargner des valeurs négatives pour les couleurs.

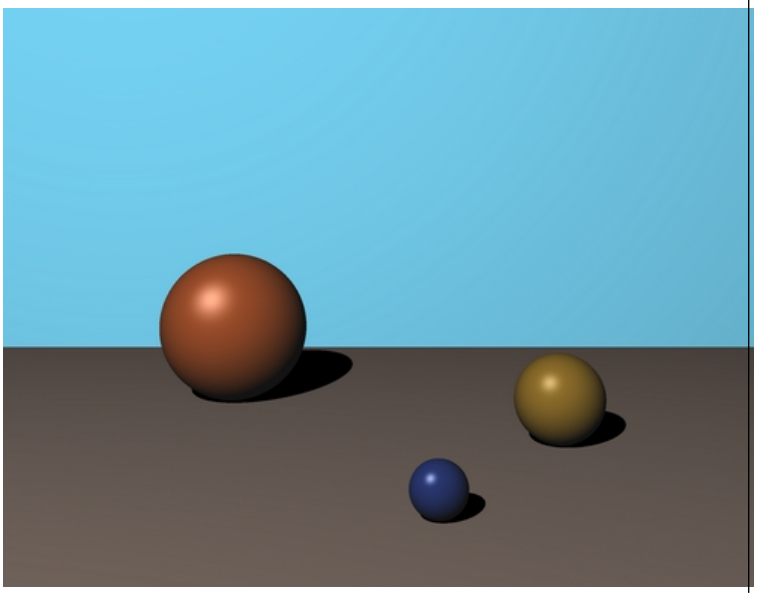
Il ne reste alors plus qu'à placer la source de lumière et à modifier la fonction **coloration** suivant ce qui a été dit précédemment.

```
# source
lumiere = vecteur(-20,-25,-15)

def coloration(image) :
    for x in range(longueur):
        for y in range(hauteur) :
            u = unitaire(direction(x, y))
            if intersection(u, centre, rayon) != 0 :
                OM = intersection(u, centre, rayon) * u
                ML = lumiere - OM
                l = unitaire(ML)
                n = unitaire(OM - centre)
                couleur = couleur_sphere * max(scalaire(n, l),0)
                image[y, x] = couleur.rgb()
    return image

image = coloration(image)
imsave("couleur.png", image)
```

Ce qui permet de produire l'image suivante déjà beaucoup plus convaincante :



III.3 Et la lumière fut



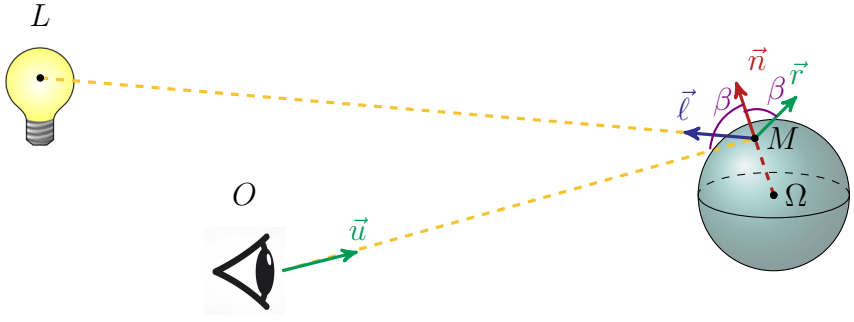
Comme on peut l'observer sur cette image certaines zones d'un objet présentent parfois un aspect brillant du fait que la lumière qui s'y reflète depuis la source atteint directement l'œil.

Afin de retranscrire cet effet il est nécessaire d'ajouter à la couleur une valeur supplémentaire appelée « composante spéculaire » que nous définirons selon le modèle d'ombrage de Phong (voir Appendice 3) explicité ci-après.

L'idée est ici, de manière analogue à la partie précédente, de mesurer l'écart angulaire entre le rayon lumineux issu de l'utilisateur réfléchi sur la sphère et le vecteur $\vec{\ell}$. Plus les deux sont proches et plus la lumière de la source a tendance a être redirigée vers l'observateur.

Il nous faut pour cela déterminer :

- $\vec{u} = \dfrac{\overrightarrow{OM}}{\| \overrightarrow{OM} \|}$ vecteur unitaire issu de l'origine vers M où M est le point de l'objet considéré.
- $\vec{r}$: vecteur unitaire de direction celle que suivrait la lumière si la surface était parfaitement réfléchissante. Ainsi l'angle formé avec la normale par le vecteur représentant le rayon incident est le même que celui formé entre la normale et le rayon réfléchi (voir Appendice 1 sur la réflexion de la lumière).
- $\beta = (\vec{r}, \vec{n}) = (\vec{n}, \vec{u})$ mentionné au point précédent.



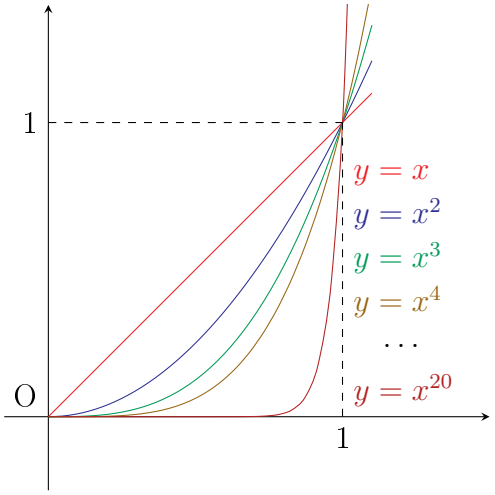
En reprenant les notations précédentes et le raisonnement présenté en **II.2**, on obtient :

$$\textit{composante speculaire} = \textit{couleur}_{source} \times (\vec{r} \cdot \vec{\ell})$$

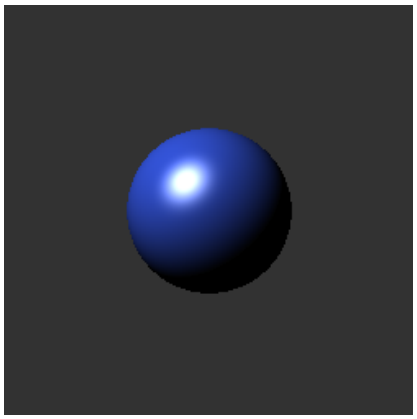
Le modèle de Phong incorpore de plus un paramètre α dépendant de l'objet considéré, plus sa valeur est grande et moins la zone brillante sera étendue :

$$\textit{composante speculaire} = \textit{couleur}_{source} \times (\vec{r} \cdot \vec{\ell})^{\alpha}$$

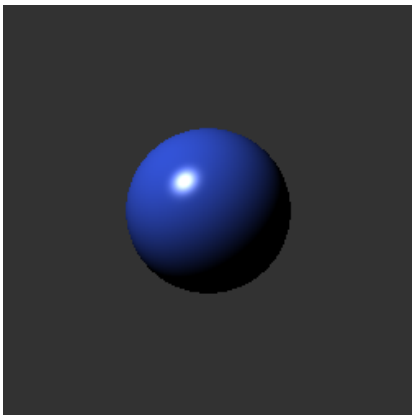
Pour mieux comprendre le rôle de α il est intéressant d'examiner les courbes de représentatives des fonctions puissances sur l'intervalle $[0;1]$ représentées ci-contre.



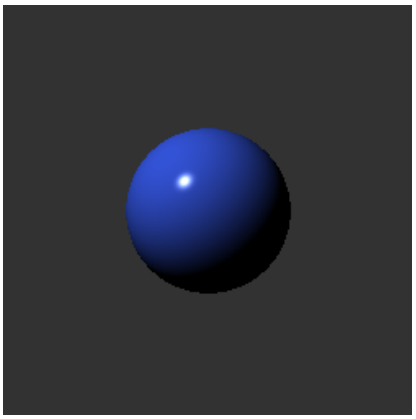
En effet, nous pouvons ainsi constater que plus la puissance est grande et plus les valeurs des images des antécédents proches de 0 sont négligeables. Ce qui signifie dans le cas de notre modèle que plus la valeur de α choisie pour l'objet représenté est grande est plus la zone blanche autour du point reflétant directement le rayon jusqu'à l'observateur sera restreinte.



$\alpha = 10$



$\alpha = 30$

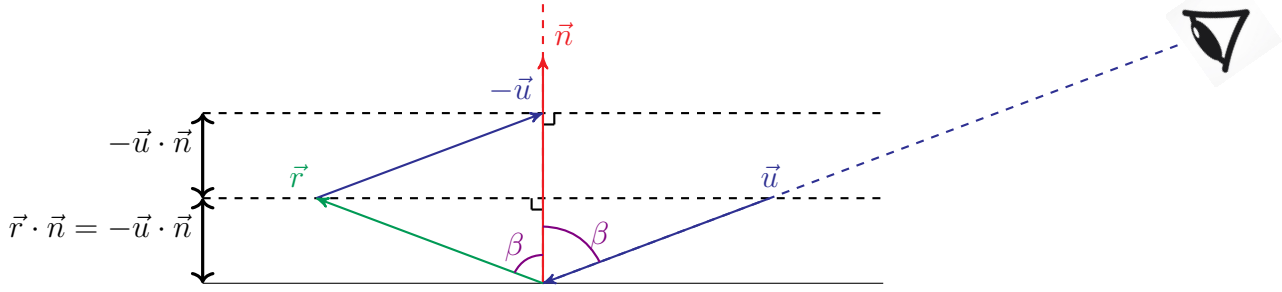


$\alpha = 100$

Proposition.

$$\vec{r} = \vec{u} - 2(\vec{u} \cdot \vec{n})\vec{n}$$

Preuve:



Tout d'abord remarquons que : $\vec{r} = \vec{u} - 2(\vec{u} \cdot \vec{n})\vec{n} \Leftrightarrow \vec{r} - \vec{u} = -2(\vec{u} \cdot \vec{n})\vec{n}$
Il suffit alors de justifier que $\vec{r} - \vec{u}$ est colinéaire à $\vec{n}$ et de composante $-2\vec{n} \cdot \vec{u}$ selon $\vec{n}$.

Soit $\vec{m}$ un vecteur unitaire coplanaire à $\vec{u}$ et $\vec{n}$ et orthogonal à $\vec{m}$. $(\vec{m}, \vec{n})$ forme alors une base du plan considéré.

Soit alors $\begin{pmatrix} x \\ y \end{pmatrix}$ les coordonnées du vecteur $\vec{u}$ dans cette base, par définition de $\vec{r}$ on a $\vec{r}\begin{pmatrix} x \\ -y \end{pmatrix}$.

Ainsi on a bien $\vec{r} - \vec{u} = \begin{pmatrix} 0 \\ -2y \end{pmatrix}$ donc $\vec{r} - \vec{u}$ est bien colinéaire à $\vec{n}$.

De plus $\vec{u} \cdot \vec{n} = y$, la composante de $\vec{r} - \vec{u}$ selon $\vec{n}$ est donc bien $-2\vec{u} \cdot \vec{n}$ et ainsi $\vec{r} - \vec{u} = -2(\vec{u} \cdot \vec{n})\vec{n}$

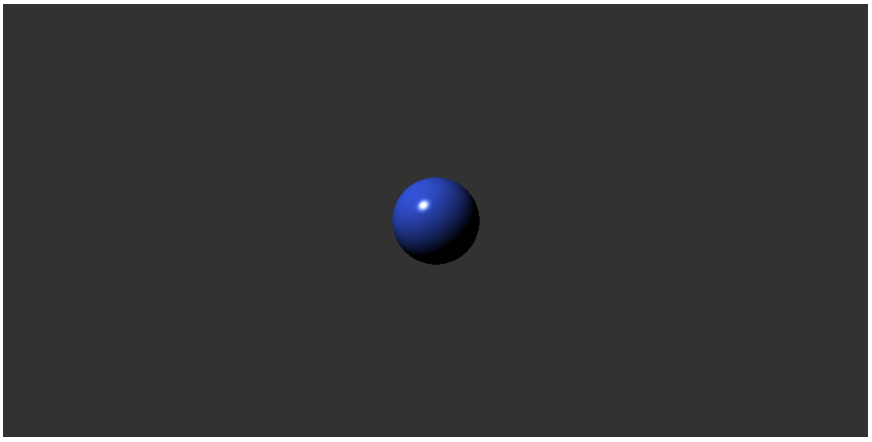
Après avoir rajouté les variable alpha et couleur lumiere, définissons alors la fonction **speculaire** qui calcule, à l'aide de la formule précédente, la composante spéculaire.

```
rayon = 1
centre = vecteur(0, 0, 10)
couleur_sphere = vecteur(52, 84, 216)
alpha = 50

# source
lumiere = vecteur(-20,-25,-15)
couleur_lumiere = vecteur(255,255,255) # lumiere blanche

def speculaire(u,n,l):
    if scalaire(l,n) > 0 :
        r = unitaire(u - 2 * scalaire(n, u) * n)
        speculaire = couleur_lumiere * scalaire(l, r)**alpha
    else :
        speculaire = vecteur(0,0,0)
    return speculaire
```

Il suffit ensuite de rajouter cette composante dans la fonction **coloration** :

```
    for x in range(longueur):
        for y in range(hauteur) :
            u = unitaire(direction(x, y))
            if intersection(u, centre, rayon) != 0 :
                OM = intersection(u, centre, rayon) * u
                ML = lumiere - OM
                l = unitaire(ML)
                n = unitaire(OM - centre)
                spec = speculaire(u,n,l)
                couleur = couleur_sphere * max(scalaire(n, l),0) + spec
                image[y, x] = couleur.rgb()
    return image

image = coloration(image)
imsave("raytracing.png", image)
```

On obtient alors ainsi l'image suivante :



À suivre. . .

IV Et de deux !

IV.1 Ombres

IV.2 Reflets

IV.3 Transparence

V Une sphère, et pourquoi pas un plan ?

VI Appendices

VI.1 Principe de Fermat et conséquences

Propagation rectiligne de la lumière, Retour inverse de la lumière,Réflexion,Réfraction.

VI.2 Définir une classe en python

Vecteur, sphère, plan

VI.3 Bui Tuong Phong

VI.4 Animation : orbite d'une planète ?

VI.5 Documentation