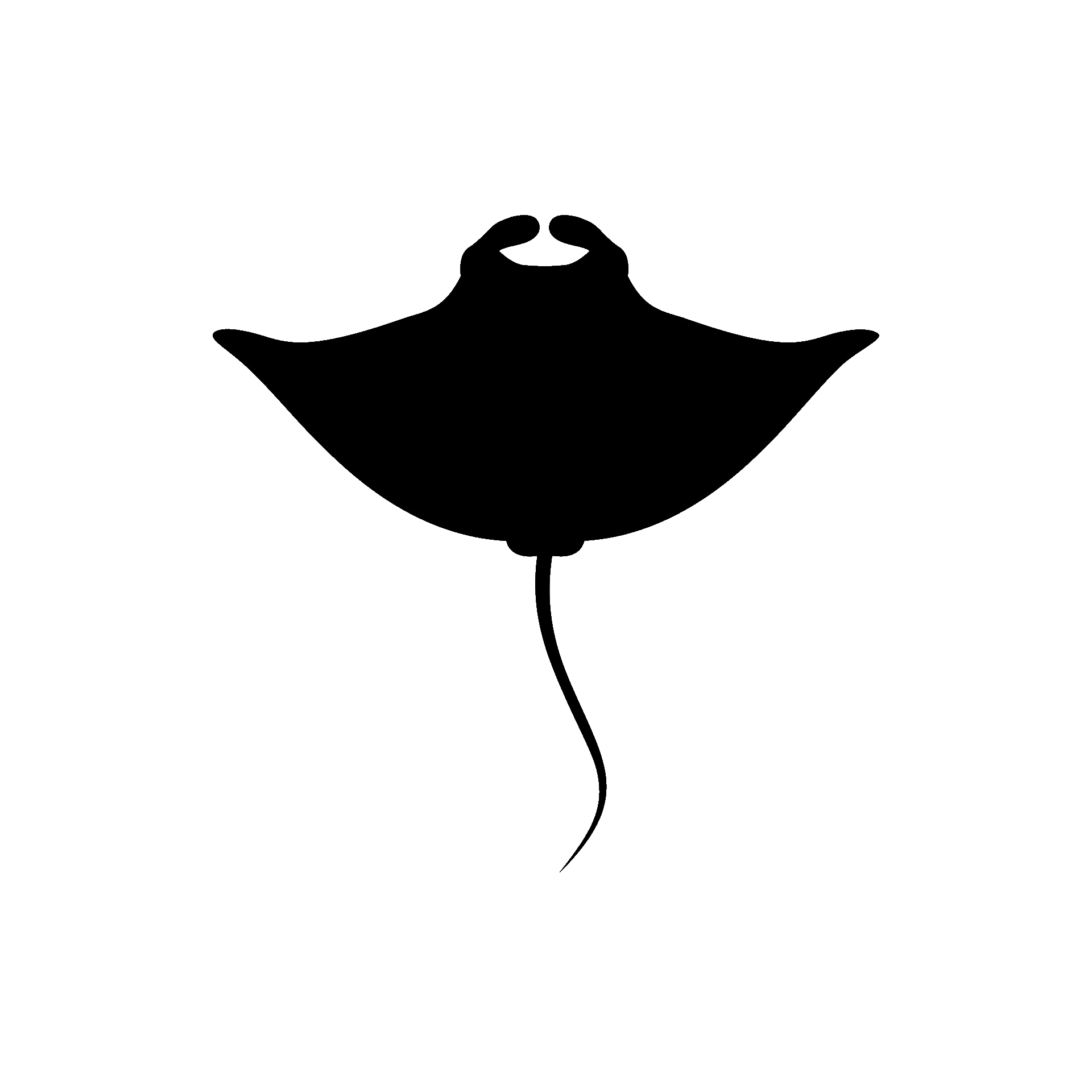
Projet Ray-Tracer

Analyse des fonctionnalités du moteur

#### Par Laurent Louis Lauret



Ray

tracer

Table des matières

[Analyse du calcul du rendu 3](#_Toc130851641)

[Modélisation du champ de vision (R.1) 3](#_Toc130851642)

[Lancer de rayon (R.2) 5](#_Toc130851643)

[Calcul d’intersection R.3 6](#_Toc130851644)

[Phong Shading R.4.1 7](#_Toc130851645)

[Réflexion R.4.2 8](#_Toc130851646)

[Multi-Processing R.4.5 9](#_Toc130851647)

[Transparence R.4.3 11](#_Toc130851648)

[Occlusion ambiante R.4.4 14](#_Toc130851649)

# Analyse du calcul du rendu

## Modélisation du champ de vision (R.1)

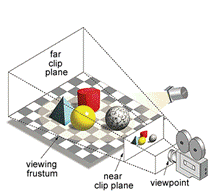
Le champ de vision (Near plane) est le plan sur lequel l’image rendu sera projetée, il s’agit donc d’un plan spatial qui correspond à la grille de pixels de l’image finale. On peut alors définir une coordonnée pour chaque pixel :

Figure  : limites du champ de vision

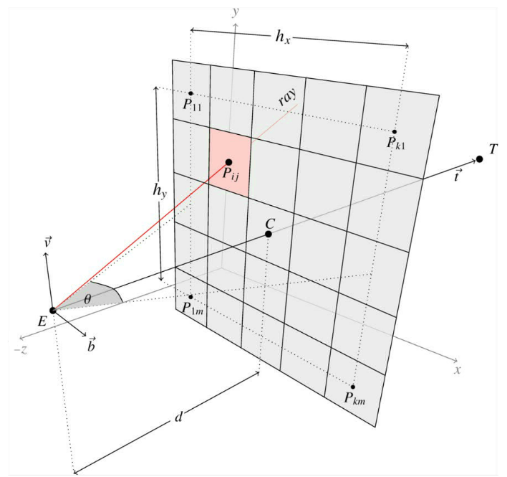
Sur la figure 2, on peut constater que la distance focale(d), la largeur du Near plane, ainsi que l’angle d’ouverture (θ) sont intimement liés :

Figure : définition du Near Plane

En se basant sur des règles de trigonométrie, on peut déduire que :

* **Hx** = 2 \* d \* tan(θ/2)
* **Hy** = **Hx** \* Ratio
* Ratio étant donné par la résolution par exemple si on a du 800x600 : le ratio est 800/600 ce qui donne 1,3333

Le but est de déterminer les coordonnés des milieux de chaque case, en utilisant des vecteurs.

Le premier vecteur est celui qui est donné par la caméra (direction) et les autres sont obtenus par des multiplications vectorielles, car le plan est toujours orthogonal à la direction de la caméra (voir figure 3). Il faut donc déterminer des vecteurs pour se déplacer horizontalement et verticalement :

* **VecZ** (normalisé) = (C-E) / d
* **VecX**(normalisé) = **VecZ** x Y\_unit\_vector
* **VecY**(normalisé) = **VecX** x **VecZ**

Une fois les vecteurs trouvés il suffit de déplacer E avec les vecteurs associés aux bonnes longueurs : on trouve P1 en partant de E :

* C = E + **VecZ** \* d
* P{1, 3} = C + **VecX** \* (**Hx**/2)
* P{1, 1} = P{1, 3} + **VecY** \* (**Hy**/2)

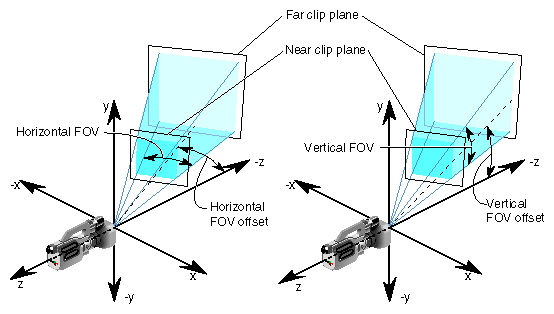


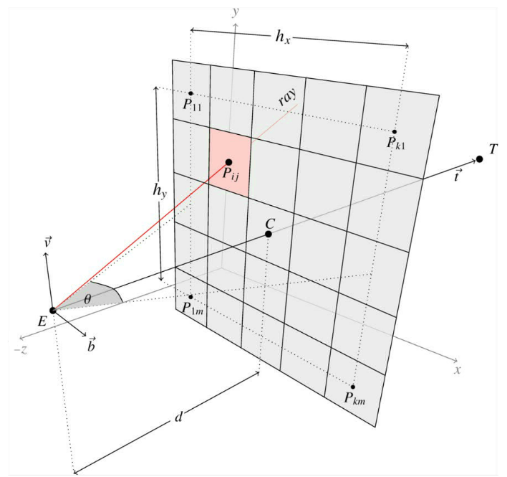
Figure : direction de la caméra

**Remarque** : Pour une résolution donnée les points sur le Near plane seront plus ou moins espacés, selon la distance focale et l’angle d’ouverture. Cela peut mener à un rendu dont la qualité peut sembler mauvaise, sans compter les phénomènes de déformation. Il convient donc de choisir des valeurs cohérentes.

Aussi dans l’exemple précédent, on considère que les limites du champ de vision sont données par les point P{1, 1}; P{k, 1}; P{1, m}; P{k, m}. Cependant, dans le code, nous déterminons les limites du plan, puis nous nous déplaçons vers le point P1,1.

## Lancer de rayon (R.2)

Le lancer de rayon consiste à créer des segments ou demi-droites dont l’origine est la position de la caméra, passant à travers les points du plan, qui représentent les pixels finalement.

****Une fois le point P1,1 déterminé, il suffira de définir des vecteurs unitaires pour se déplacer en largeur et en longueur.

* **VecRight**(normalisé)= (P{k, 1} – P{1, 1}) / **Hx**
* **VecDown**(normalisé) = (P{1, m} – p{1, 1}) / **Hy**

Une fois les vecteurs de déplacement obtenus, on peut définir le pas pour trouver le prochain point :

* **step\_x** = **Hx** / NbrPoints\_x
* **step\_y** = **Hy** / NbrPoints\_y

À partir d’ici, on peut effectuer un balayage sur tous les points, répétant les déplacements à partir de P1,1 :

Figure 3 : balayage du plan

* P{k, 1} = P{k-1, 1} + **VecRight** \* **step\_x**
* P{1, m} = P{1, m-1} + **VecDown** \* **step\_y**

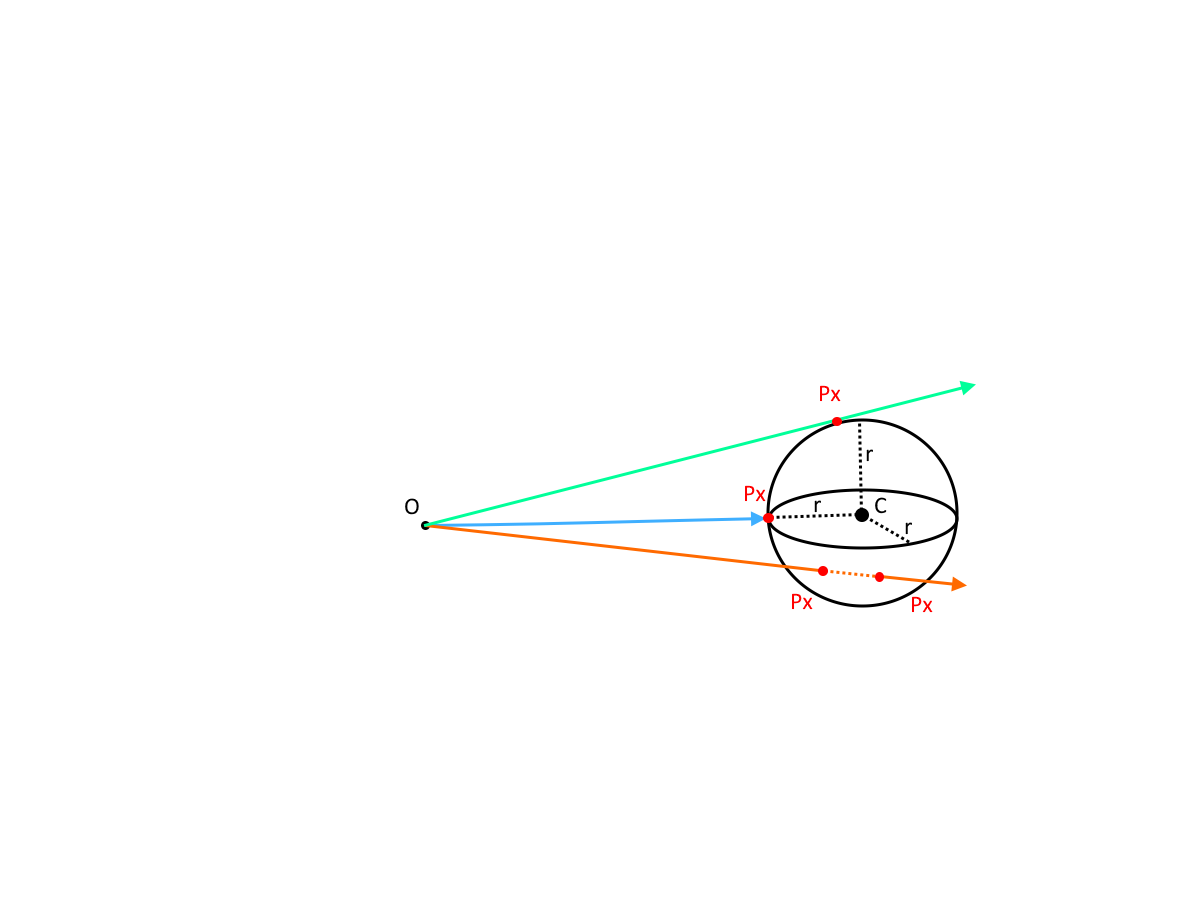
Une image contenant texte

Description générée automatiquementApproche algorithmique du balayage:

## Calcul d’intersection R.3

Lorsqu’un rayon est lancé, on doit déterminer si celui-ci intersecte une ou plusieurs géométries de la scène. Il faut donc une façon de déterminer cette intersection. Cependant, il y a autant de différentes méthodes d’intersection que de géométries. Étant donné que le Cube et le plan sont déjà supportés dans la librairie Geometry3D, il ne reste que la sphère qui repose sur un modèle mathématique.

Intersection à la sphère : Si on a une sphère dont le **centre** est **C** (point3D) et un **rayon r**, alors on peut dire que tout **Px** se trouve sur la sphère si la **distance** entre **Px** et **C** correspond au rayon **r** : norme(**Px** – **C**) = **r**. ou encore

Ici Px peut être remplacé par l’expression d’un rayon qui aurait pour direction d, et de norme x : Px = O + d \* x

Une image contenant texte

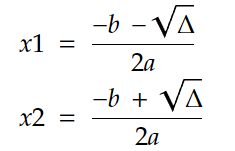
Description générée automatiquementFinalement l’équation devient :

En se basant sur le fait que la norme d’un vecteur au carré est égale au produit scalaire de ce vecteur par lui-même, on peut obtenir une équation quadratique, que l’on doit résoudre pour obtenir les points d’intersections existants.

Une image contenant texte, orange

Description générée automatiquementOn peut alors passer par le calcul du discriminant.

Il n’y aura une intersection que si le discriminant est positif.

 S’il y a deux solution positive, celle qui sera la plus proche du point d’origine sera retenue.

## Phong Shading R.4.1

Une fois l’intersection vérifiée, on peut procéder au calcul de l’illumination, pour laquelle on utilise le modèle de Phong. Ce modèle repose sur la combinaison de 3 éléments : la couleur ambiante, diffuse, et spéculaire.

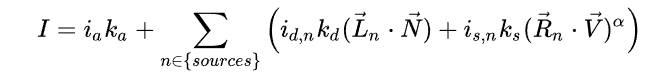
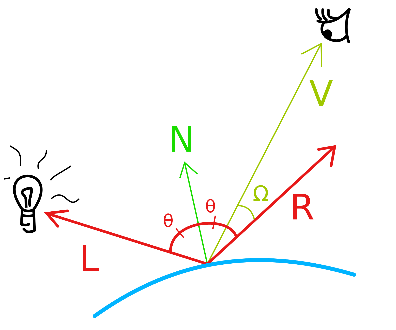
* **Ambiante** : est constante, propre à la géométrie en absence de lumière.
* **Diffuse** : varie en fonction de la direction de la lumière par rapport à la géométrie
* **Spéculaire** : couleur qui serait renvoyée en cas de parfaite réflexion de la lumière vers la caméra

Figure : formule de l'illumination selon le modèle de Phong (Wikipédia)

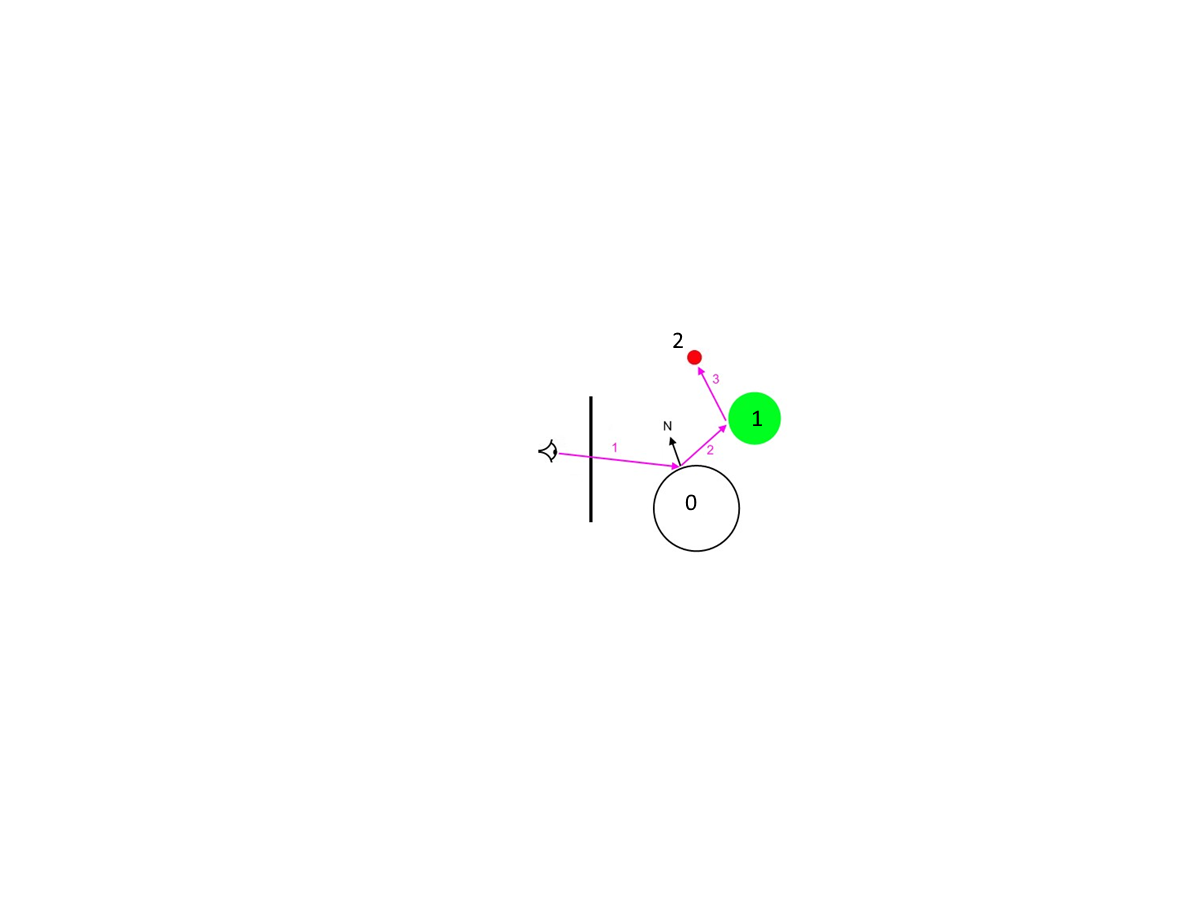
Une image contenant horloge, montre

Description générée automatiquementLa fig.4 montre la formule pour calculer l’illumination d’une surface pour une à plusieurs sources de lumières. Si on considère **un seul point de lumière** et que les **paramètres de la** **lumière ne varient pas**, on peut **simplifier** le modèle :

* I étant l’illumination totale
* Ka étant le coefficient ambiant de la surface/géométrie (généralement 0.3)
* Kd étant le coefficient diffus de la surface/géométrie (généralement 0.7)
* Ks étant le coefficient spéculaire de la surface/géométrie (généralement 1)
* N, la normal au point d’intersection de la surface/géométrie
* L, vecteur unitaire partant du point d’intersection vers la source
* V, vecteur unitaire partant du point d’intersection vers la caméra
* Alpha : coefficient de la surface qui va définir la taille du point spéculaire en relation avec le matériel (doux ou rugueux) (brillance)
* R, vecteur unitaire parfaitement réfléchi qu’on peut déterminer de la façon suivante :

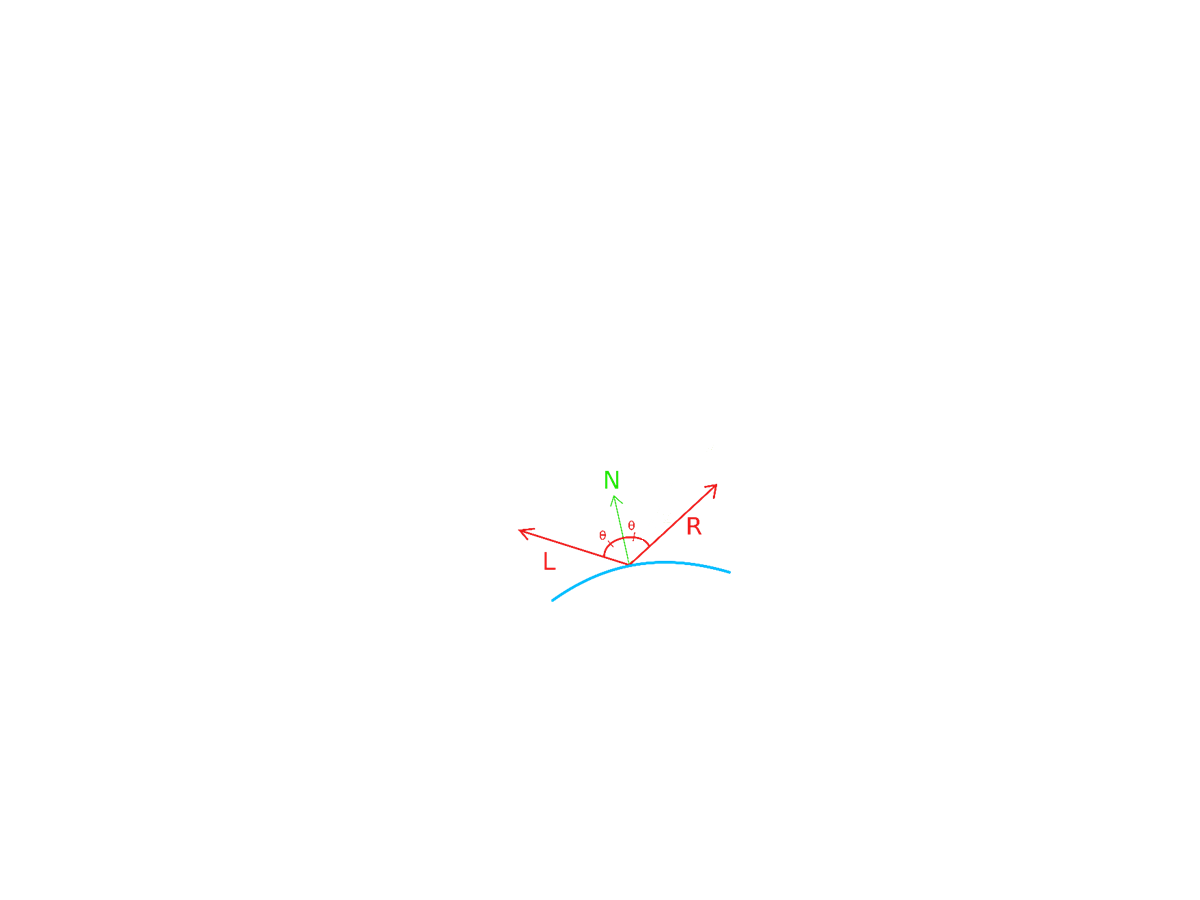
## Réflexion R.4.2

Pour implémenter la réflectivité, nous devons d’abord assumer que les surfaces peuvent avoir un coefficient de réflexion, allant de 0 à 1, 0 étant une surface non réflective et 1, complètement réflective.

Une image contenant texte, tableau blanc

Description générée automatiquementLe phénomène de réflexion implique que l’illumination renvoyée par l’objet n’est plus seulement la sienne, mais également celle de son environnement selon son coefficient de réflexion, autrement dit :

* I étant l’illumination totale
* In, illumination de la géométrie n, au point d’intersection
* Rn, coefficient de réflexion de la géométrie n

Il reste à déterminer comment obtenir le rayon réfléchi. On peut reprendre la formule donnée dans le Phong Shading lorsqu’on parlait de rayon **parfaitement réfléchit**.

La réflexion peut donc se traiter récursivement, c’est-à-dire qui si le rayon incident (**-L**) intersecte une surface réfléchissante, on doit lancer un **rayon secondaire** en partant de la **surface réfléchissante**, dans la **direction de R**. Et si ce dernier intersecte une **autre géométrie réfléchissante**, on recommence le processus, avec **le nouveau rayon** incident et un **nouveau rayon réfléchit**.

Finalement, le traitement récursif ne s’arrête que si la surface actuelle n’est pas réfléchissante, ou encore que le nombre de rebonds ait dépassé le maximum.

Une image contenant texte

Description générée automatiquementApproche algorithmique :

## Multi-Processing R.4.5

Lorsqu’il s’agit d’implémenter le traitement **concurrent**, ou **parallèle**, les solutions sont le **multithread**, ou bien le **multiprocessing**. Ces solutions présentent des similitudes mais ne fonctionnent pas de la même manière. De manière générale, on peut avoir plusieurs threads au sein d’une application, et les threads partagent leur mémoire, cependant l’application n’occupe qu’un seul processus. Alors que dans le cas du multiprocessing, l’application effectue son traitement sur plusieurs cœurs du processeur. Autrement dit, il y a un **thread principal par processus**.

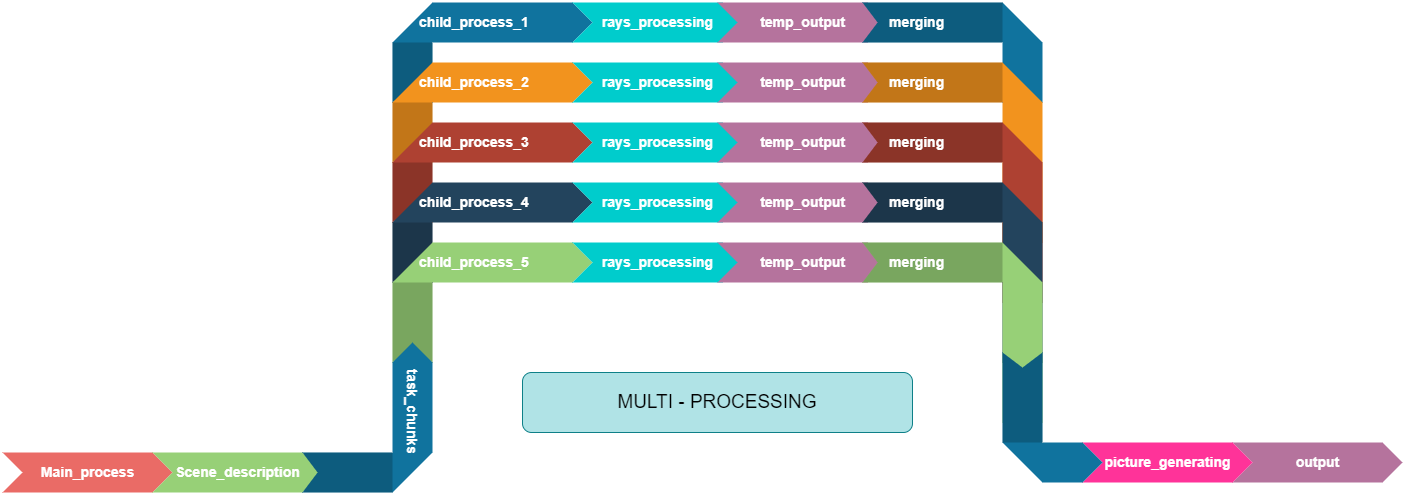
Python n’a pas réellement de **sécurité** au niveau des **threads**, et c’est pourquoi le **GIL** (Global Interpreter Lock) a été implémenté. Il impacte grandement le multithreading dans la mesure ou il n’y a **qu’un seul thread** qui peut prendre le contrôle de l’interpréteur Python à la fois, et de ce fait, il ne s’agit pas réellement de traitement parallèle.

Dans mon application j’ai décidé de faire du **multiprocessing**, mais celui-ci présente des **contraintes**, telles que la **mémoire n’est pas partagée** entre les processus. Pour retourner des **résultats**, il faut mettre en place des Queues ou des Pipes qui sont en fait des **canaux de communication** entre les processus.

Une image contenant texte

Description générée automatiquementImplémentation :

En utilisant un **seul** processus, le traitement de **chaque rayon** se fait de manière **séquentielle**, c’est-à-dire qu’on itère sur tout les rayons, et on traite toutes les intersections les unes après les autres. Avec les processeurs récents, l’utilisation du CPU est très **faible**, même si la **vitesse est élevée**. Pour accélérer le traitement, il faudrait donc utiliser pleinement le processeur, en se servant de tous les cœurs du CPU.



En utilisant une approche **multiprocessus**, il y a quelques **étapes supplémentaires**. Il faut, en effet **découper** le traitement de la scène en morceaux, pour en faire des **traitements séparés**. C’est uniquement lorsque les processus enfants seront **terminés** qu’on pourra **réassembler** le résultat de chaque processus enfant, pour pouvoir **regénérer** l’image au complet.

Pour découper le traitement, on peut diviser le champ de vision sur la hauteur, afin de faciliter le réassemblage. On se retrouve alors avec un nombre réduit de rayons à traiter au niveau de chaque processus. Une fois le traitement effectué chaque processus crée un fichier temporaire sur le disque dur. Il suffira donc d’importer ces fichiers en mémoire grâce à une structure adaptée qui permettra de concaténer chaque morceau.

Une image contenant diagramme

Description générée automatiquement

## Transparence R.4.3

Il est difficile de parler de transparence sans parler de réfraction. En effet jusqu’ici, les géométries étaient considérées opaques, donc les rayons ne traversaient pas les géométries, au pire ils étaient réfléchis. Cependant, lorsque les objets deviennent transparents, les rayons les traversants peuvent être réfractés selon un certain indice.

Une image contenant texte

Description générée automatiquement

Une image contenant diagramme, schématique

Description générée automatiquementPour aborder le rendu de la transparence, nous allons commencer par la transparence sans réfraction (r =1), et ensuite voir ce qui se passe lorsque la réfraction entre en jeu.

Transparence :

Comme on peut le voir sur la fig.5, lorsque les rayons intersectent la boule transparente, ils continuent dans la même direction. Il faut donc aller chercher l’illumination de la **prochaine** géométrie qui se trouve dans la **même direction**. Par exemple au point **S1**, on va chercher **l’illumination de S2**, et en **S3** on celle de **S4**.

Figure : Transparence sans changement de milieu

Une image contenant graphique

Description générée automatiquementSi on s’arrêtait là on ne verrait pas de différence dans l’image. Pour avoir le rendu de la transparence il faut **ajouter** à la lumière trouvée en **arrière de la sphère**, sa **propre illumination** (on peut également le faire avec du Phong shading mais en utilisant des coefficients très bas).

Si la surface de la sphère est réfléchissante, il faut également ajouter la lumière réfléchie par exemple au point S1 ou S3, comme vu précédemment.

Comme on peut le voir sur la fig.6, on a donné une légère teinte à la boule, afin de mettre en avant le calcul de la lumière pour un objet transparent sans réfraction.

Figure : aperçu de la transparence

Réfraction :

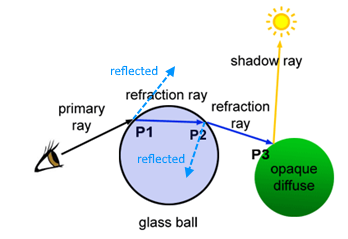
Lorsqu’il y a réfraction, la **direction** du rayon **change** autant de fois qu’on **change de milieu**. L’**angle** de réfraction dépend des **indices des milieux respectifs** ainsi que l’angle que fait le **rayon incident** par rapport à la **normale** au point d’incidence. Cependant, la réfraction n’est pas totale, car une partie du rayon incident est réfléchit.

Figure : trajet de la lumière lors d'une réfraction

Le calcul des différents vecteurs nous est donné par la loi de Snell-Descartes (Fig.8):

* Une image contenant texte, lettre

  Description générée automatiquementn représente la normale
* I représente la direction du rayon incident
* Teta1 l’angle entre le rayon incident et la normale
* Teta2 l’angle entre le rayon réfracté et la normale

La note nous indique qu’il y a une inversion à faire selon si le rayon incident pénètre la sphère ou qu’il en ressort.

Figure : lois de Snell-Descartes

Une image contenant texte

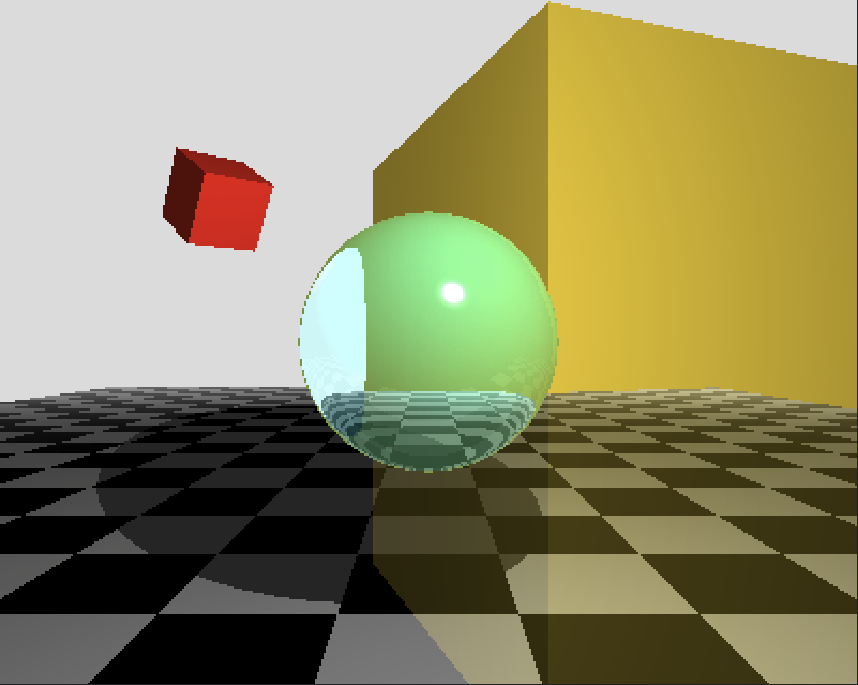
Description générée automatiquementUn **cas spécifique** peut se produire lorsque le rayon est **réfracté en sortie** de la sphère (en allant vers P3). En effet lorsqu’on passe d’un milieu dont l’**indice est supérieur à celui de destination**, une **réflexion totale** peut se produire, il n’y a alors aucun rayon réfracté. Ce cas se produit lorsque :

Dans ce cas, il faut se servir du **rayon reflété** pour pouvoir créer un autre rayon **réfracté à la sortie** de la sphère.

En ce qui concerne le calcul de la luminosité, elle est identique à celle de la transparence sans réfraction, mis à part le fait, qu’on doit lancer des rayons secondaires afin de déterminer le point de sortie de la géométrie qu’on traverse pis enfin suivre la direction du rayon réfracté pour aller chercher la l’illumination des objets qui intersectent ce rayon.

Remarque : Pour éviter une explosion du nombre de rayon reflétés à l’intérieure d’une géométrie, dans mon application, je me limite à une réflexion, ce qui signifie que si le rayon ne peut être réfracté après celui-ci, je poursuis dans la direction du rayon reflété.

Une image contenant noir

Description générée automatiquementUne image contenant graphique

Description générée automatiquement

Figure : réfraction faible

Figure : apparition de la réflexion totale

Figure : réfraction forte

## Occlusion ambiante R.4.4

Lorsque deux surfaces entrent en contact, comme un coin de la scène par exemple, il se produit un phénomène **d’occlusion**, c’est-à-dire que la **lumière ne peut pas** atteindre les points où les surfaces entre en contact. Pour en faire le rendu, il faut vérifier pour chaque point de la scène, si on est en mesure d’identifier des intersections aux **alentours** de ces points (fig.12).

Dans la plupart des moteurs l’occlusion est précalculée. Cependant, elle peut être réalisée à même le ray-casting.

Une image contenant graphique

Description générée automatiquementPour ce faire, il suffit de déterminer préalablement si le rayon primaire a touché une géométrie. Si le point est déjà dans une ombre, on peut se passer de l’occlusion. Si la surface est éclairée on peut alors déterminer s’il y a une occlusion.

Figure : détection d'occlusions

Notion de quantité d’occlusion

Une image contenant graphique

Description générée automatiquementPour pouvoir en faire le rendu, il est nécessaire de pouvoir **mesurer** l’occlusion. Le principe est assez simple. On lance des **rayons secondaires** à partir du point d’intersection suivant **plusieurs directions**, puis on conserve ceux qui **intersectent** d’autres surfaces sur une **courte distance**. Ce faisant on est en mesure de déterminer la proximité des autres surfaces.

Finalement, s’il n’y a **aucune** intersection, la luminosité ne sera **pas affectée**. Si le nombre d’intersection est **élevé**, cela signifie que ce point ne peut être atteint par la lumière, il sera donc **fortement terni**. En comptant le **nombre** d’intersections, il devient possible de **moduler** **la quantité d’occlusion**.

Une image contenant texte

Description générée automatiquementAfin de **limiter** le nombre de rayon à lancer, on peut se baser sur la **normale** au point d’intersection, pour déterminer les rayons dans un **champs de 180 degrés** autour du point d’intersection.

Figure : vecteurs d'occlusions

* **Ir**, couleur résultante
* **I0**, illumination au hit point

En modifiant le ratio d’occlusion à la puissance 0.6, on a un rendu plus contrasté pour de faibles valeurs de ratio. Le ratio avoisinant les 50% au maximum, pour deux surfaces qui entrent en collision, cette modification permet de mieux distinguer l’occlusion.

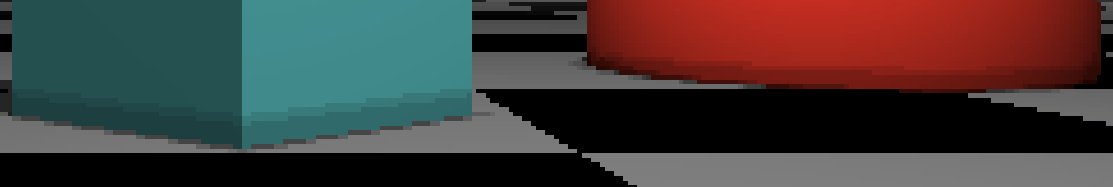


Figure : occlusion basée sur le nombre de rayons

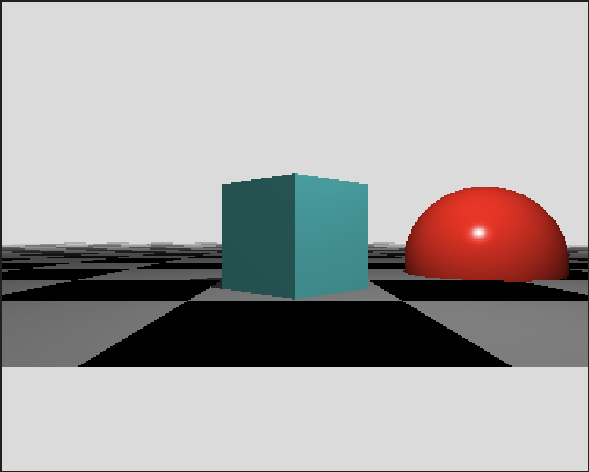


Figure : Phong (1m57)

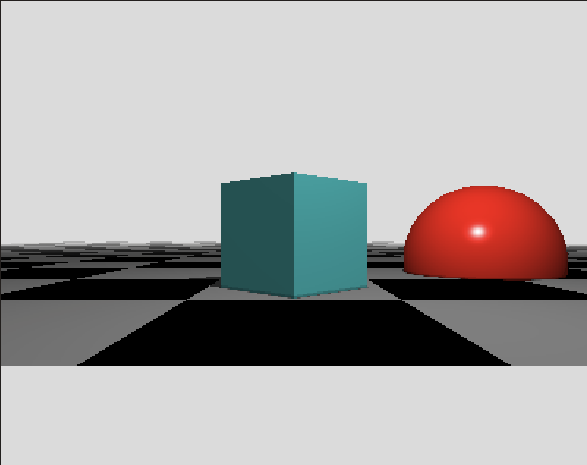


Figure : Phong + occlusion 1-x (9min11) (9 rays)

Figure : Phong + occlusion 1-x^0.6 (9min32) (9 rays)

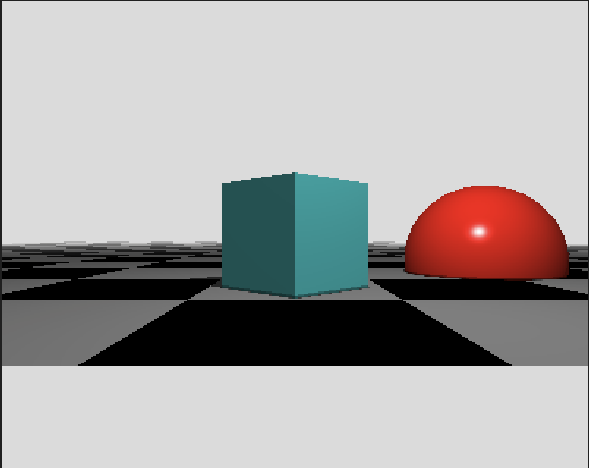
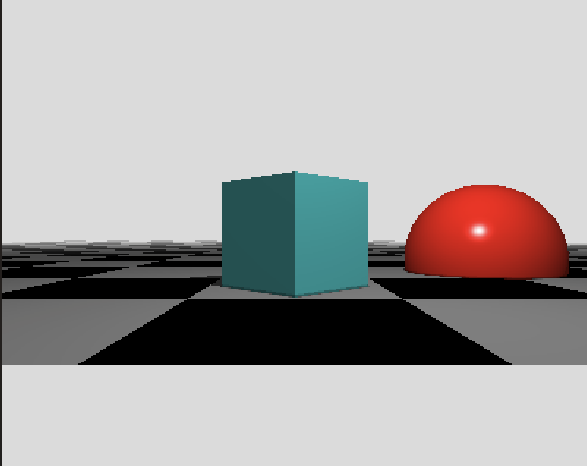


Figure : Phong + occlusion (12m27) (13 rays)

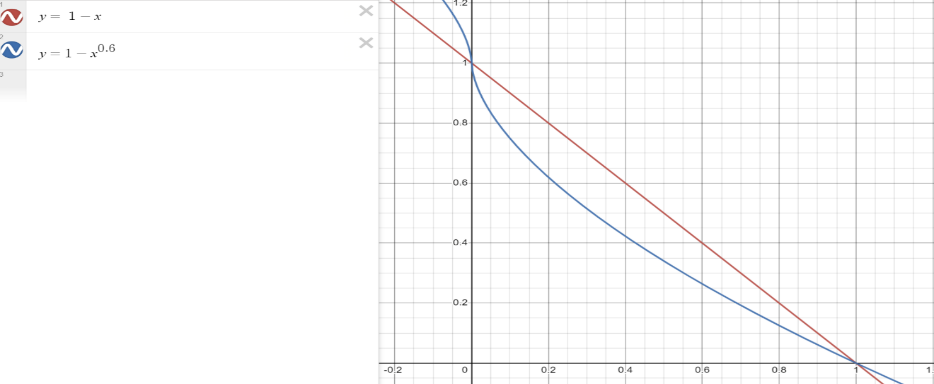


Figure : ajustement de la courbe

Occlusion adaptive :

Il s’agit de faire le rendu de l’occlusion en ayant un **dégradé** et non de façon abrupte, en se **basant sur la distance** d’intersection des **rayons d’occlusions**. Pour l’implémenter, en premier lieu il faut **pondérer les rayons**, car certains rayons ne donneront pas la même information concernant l’occlusion (voir fig.13). Par exemple, on peut décider que :

* Les rayons vert, bleus et la normale sont prioritaires : 50 % (10% chaque)
* Les autres rayons comptent pour 50%, ce qui donne 6,25% par rayon
* Une image contenant texte

  Description générée automatiquementLa distance des rayons évolue entre 0 (100% d’occlusion) et une distance max (0% d’occlusion).

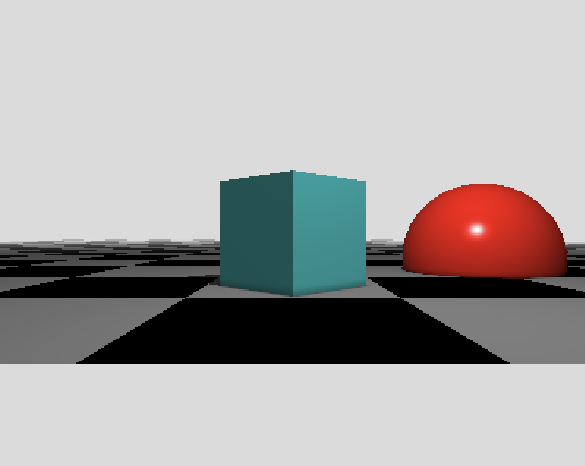


Figure : occlusion adaptive (dmax = 0.5)

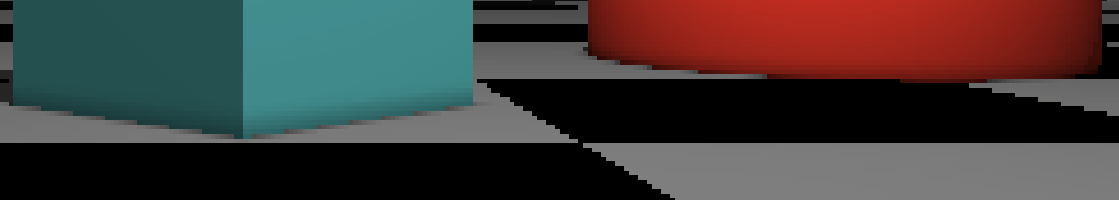


Figure : occlusion adaptive

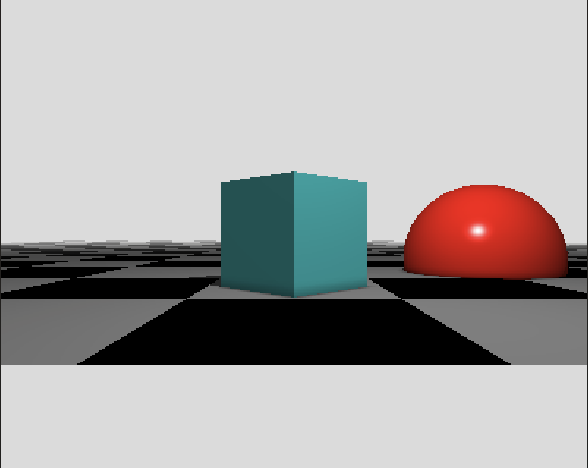


Figure : occlusion adaptive (dmax = 0.7)