Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
M4102Cin - Synthèse d'images 2

1. illumination

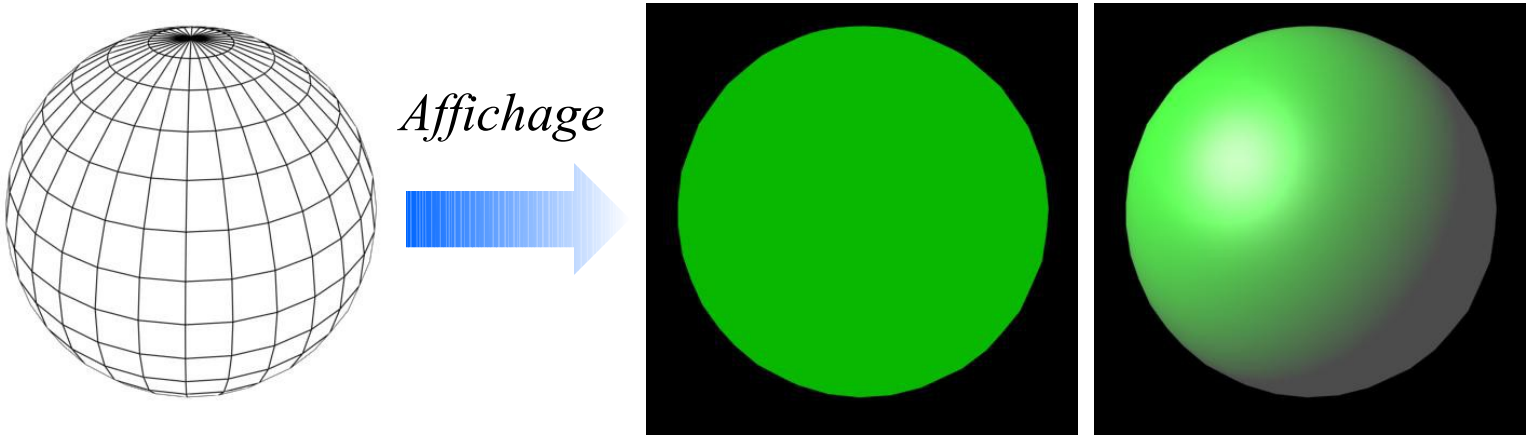
DUT INFO 2ème année 2019-2020

Sébastien THON

IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

1. Problématique

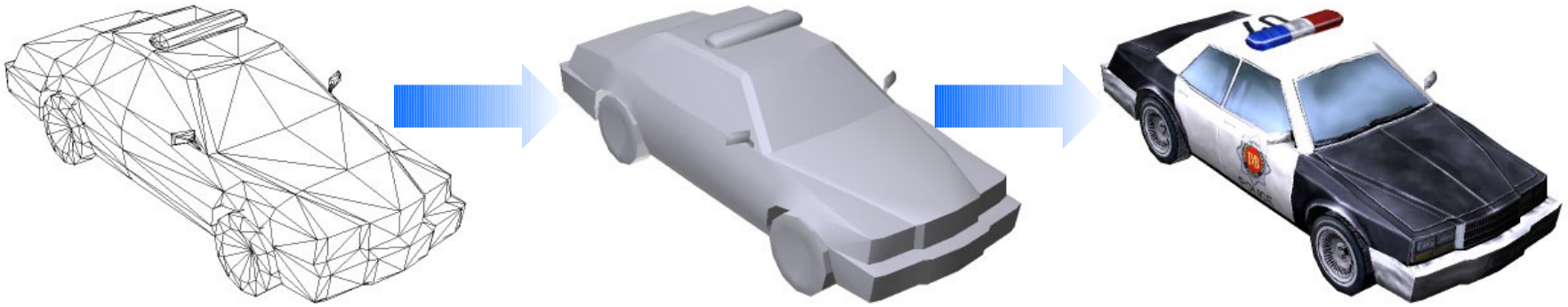
Afficher un objet 3D en utilisant seulement sa couleur ne donne pas de résultat réaliste.



Objet *Couleur seule* *Prise en compte de la lumière*

→ Il faut prendre en compte les interactions de la lumière avec les surfaces des objets (= illumination ou « *shading* »).

Après avoir défini un objet géométriquement (ex: liste de triangles), il faut l'afficher en tenant compte des sources de lumières, des propriétés de réflexion de sa surface, de sa texture, etc.



Objet 3D

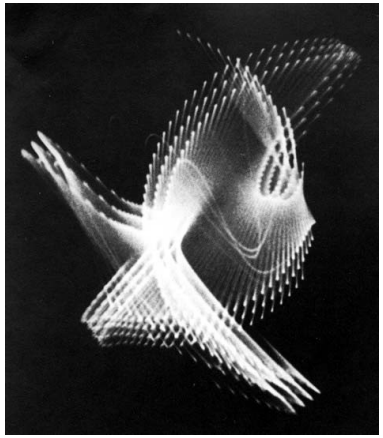
Prise en compte de la lumière

Application de texture

➔ **« Rendu »** (réaliste ou non)

Le réalisme des images de synthèse s'est amélioré au cours du temps grâce à plusieurs facteurs :

- Images de plus en plus précises (résolution, nombre de couleurs).
- Puissance de calcul plus élevée.
- Hardware dédié et programmable (GPU).
- Algorithmes plus sophistiqués.



1950

1963

1981

2016

Les images de synthèse atteignent aujourd'hui un réalisme photographique.

De nombreux phénomènes de la réalité doivent être pris en compte :

modélisation des sources de lumière, réflexion de la lumière par la surface, réfraction, ombres, etc.

Les images de synthèse ont généralement deux utilisations :

- 3D précalculée

Ex: images fixes, film d'animation, effets spéciaux, …

Produit avec un logiciel de synthèse d'images (3D Studio Max, Blender, Maya, Lightwave, …).
L'ordinateur peut passer plusieurs minutes ou heures pour calculer une image. On peut obtenir des résultats très réalistes (beaucoup de polygones, modélisation précise de la lumière, etc.).

- 3D temps réel

Ex: simulateur, jeu, visualisation scientifique, …

Produit avec une librairie graphique (OpenGL, DirectX, …).
Il faut calculer suffisamment d'images par seconde (FPS: *Frames Per Second*), au moins une vingtaine. Le nombre de FPS dépend de la machine. On sacrifie le réalisme visuel pour produire rapidement des images (compromis nécessaire).

Évolution de la 3D temps réel :

Aux débuts :

Images 3D tracées par le **CPU** (Central *Processing Unit*) uniquement.

→Monopolise la puissance du CPU.

→Laisse peu de temps pour d'autres traitements.

Depuis 2001 :

Hardware spécialisé travaillant en parallèle avec le CPU : cartes graphiques dotées d'un **GPU** (*Graphics Processing Unit*) programmable au moyen de **shaders**

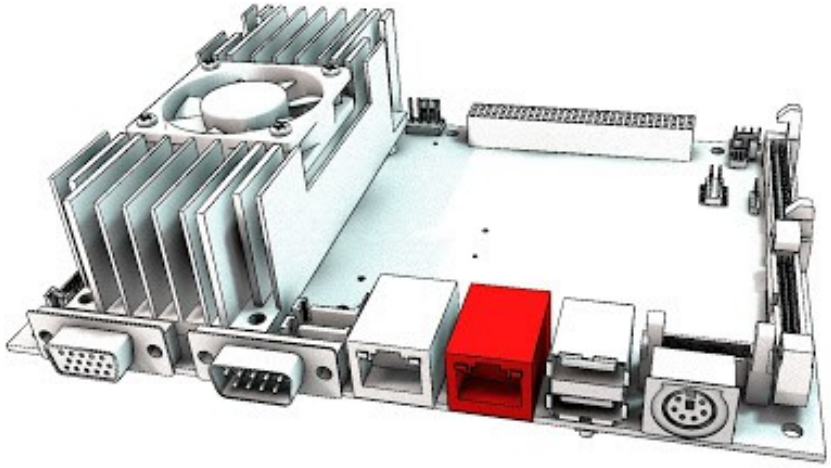
→ Affichage 3D plus rapide, plus performant.

→ Libère le CPU pour d'autres traitements.

NPR (Non Photorealistic Rendering)

On ne cherche pas toujours un rendu photoréaliste.

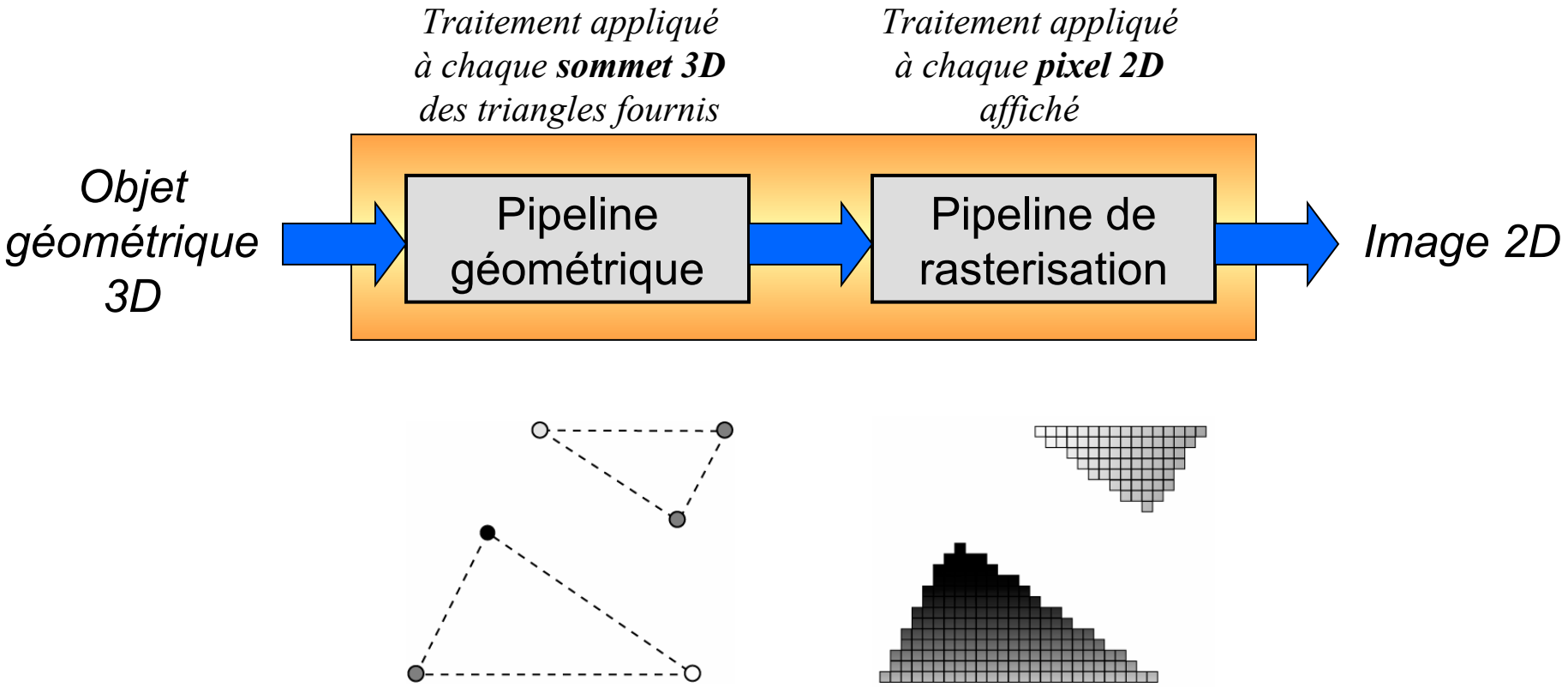
Rendu “cartoon” pour le jeu ou le dessin animé, schématique pour de l'illustration technique, effets artistiques, etc.



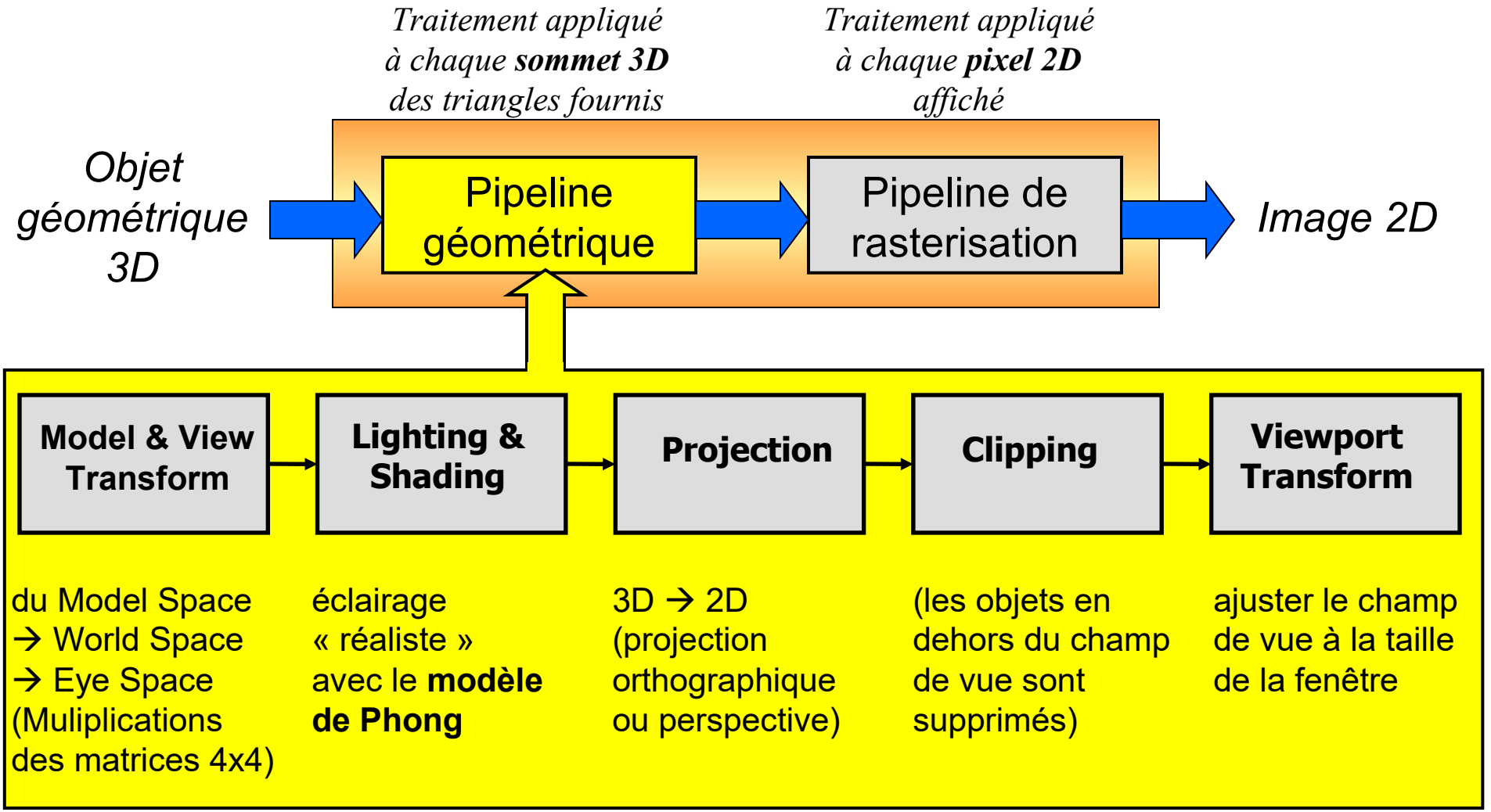
On peut écrire des shaders qui vont produire un affichage complexe (ombrage, reflets, contours, hachures, ...).

2. Pipeline graphique

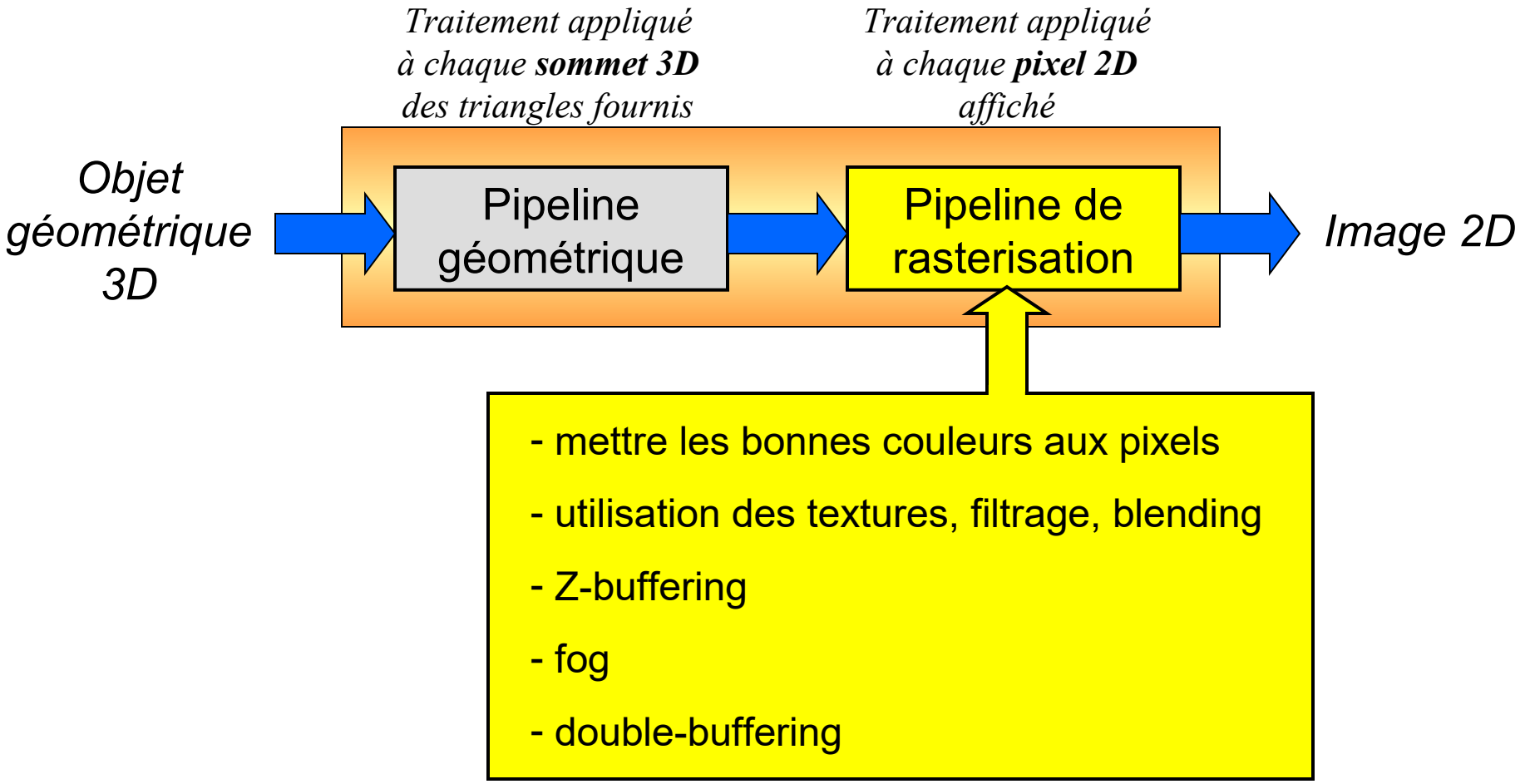
= suite des traitements appliqués à un objet (ou un ensemble d'objets) géométrique(s) dans le but d'en obtenir le rendu à l'écran, sous la forme d'une image 2D.



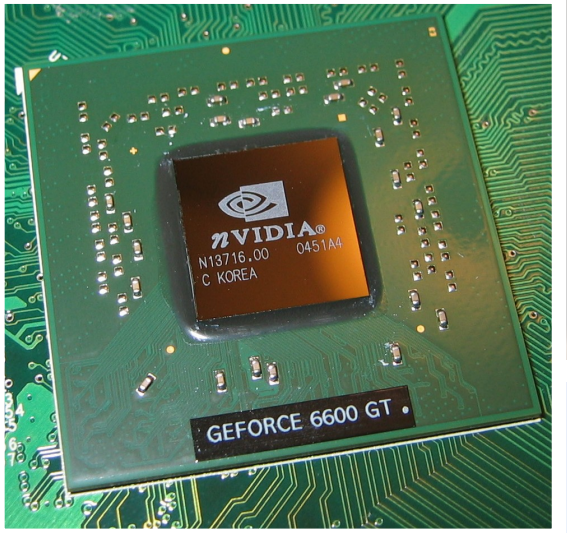
Pipeline graphique

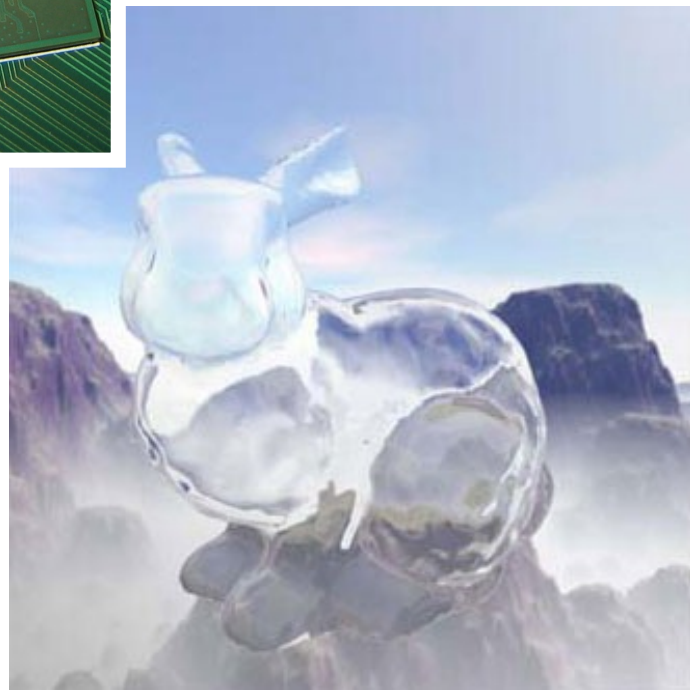


Pipeline graphique



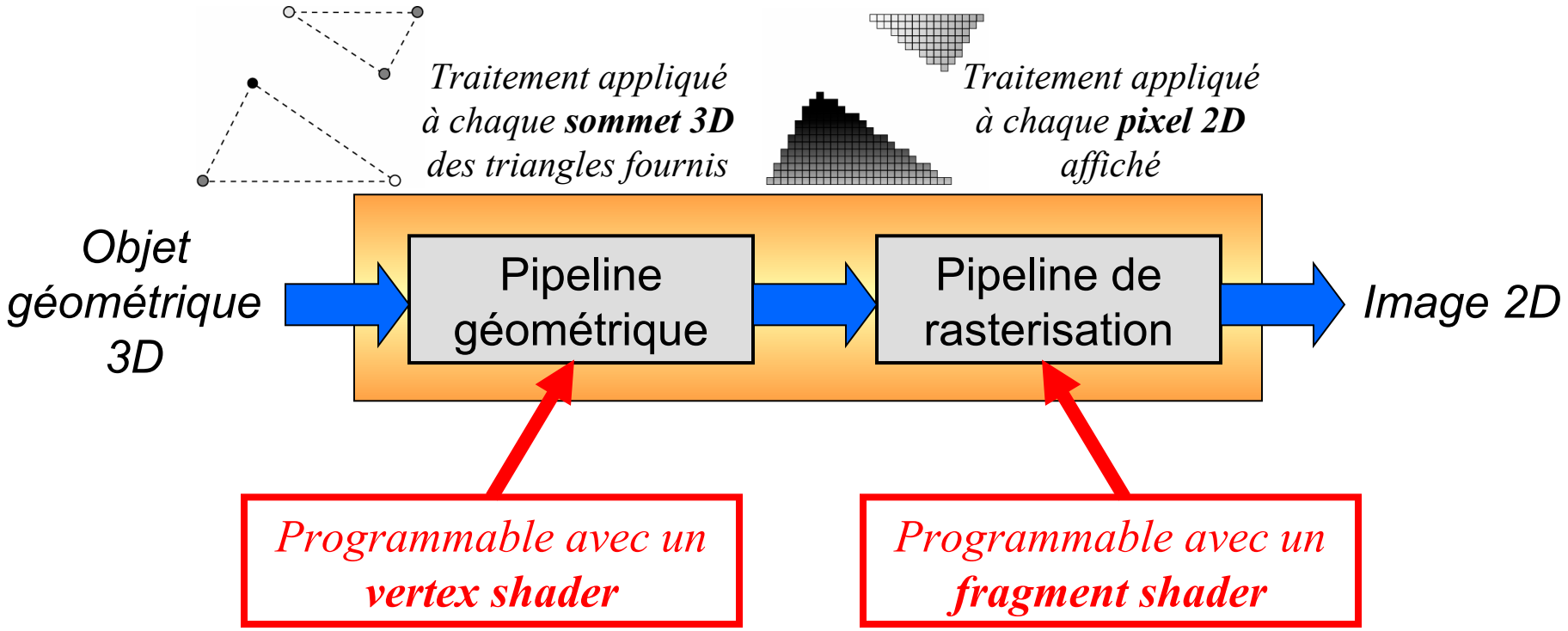
Ce pipeline peut être reprogrammé grâce au **GPU** des cartes graphiques, ce qui permet d'obtenir des images très sophistiquées en temps réel, réalistes ou pas.





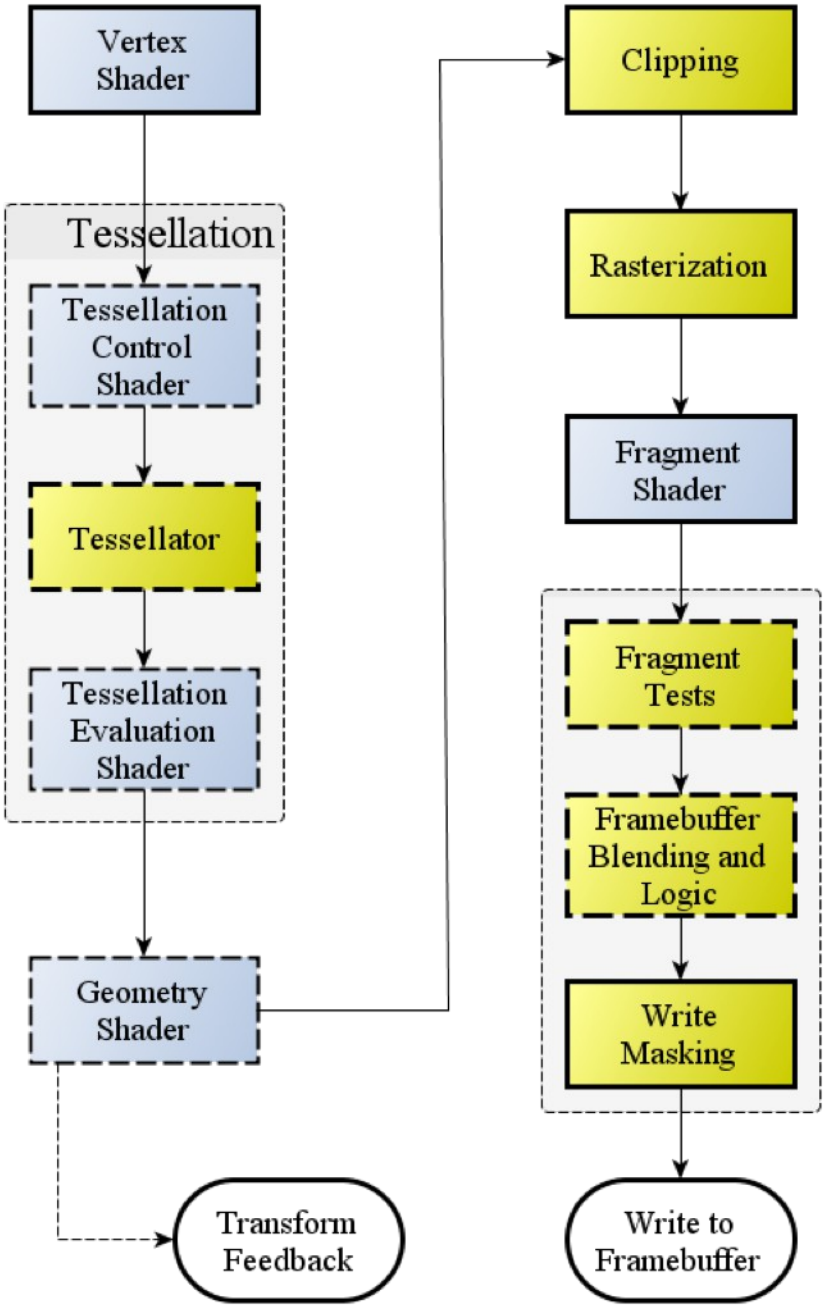
Les programmes écrits pour le GPU afin de reprogrammer le pipeline sont appelés des **shaders**, dont il existe plusieurs sortes :

- Les **vertex shaders** permettent d'appliquer des effets géométriques au niveau des sommets des objets 3D.
- Les **fragments shaders** (aussi appelés **pixel shaders**) permettent d'appliquer des effets visuels au niveau des pixels.



D'autres shaders sont optionnels :

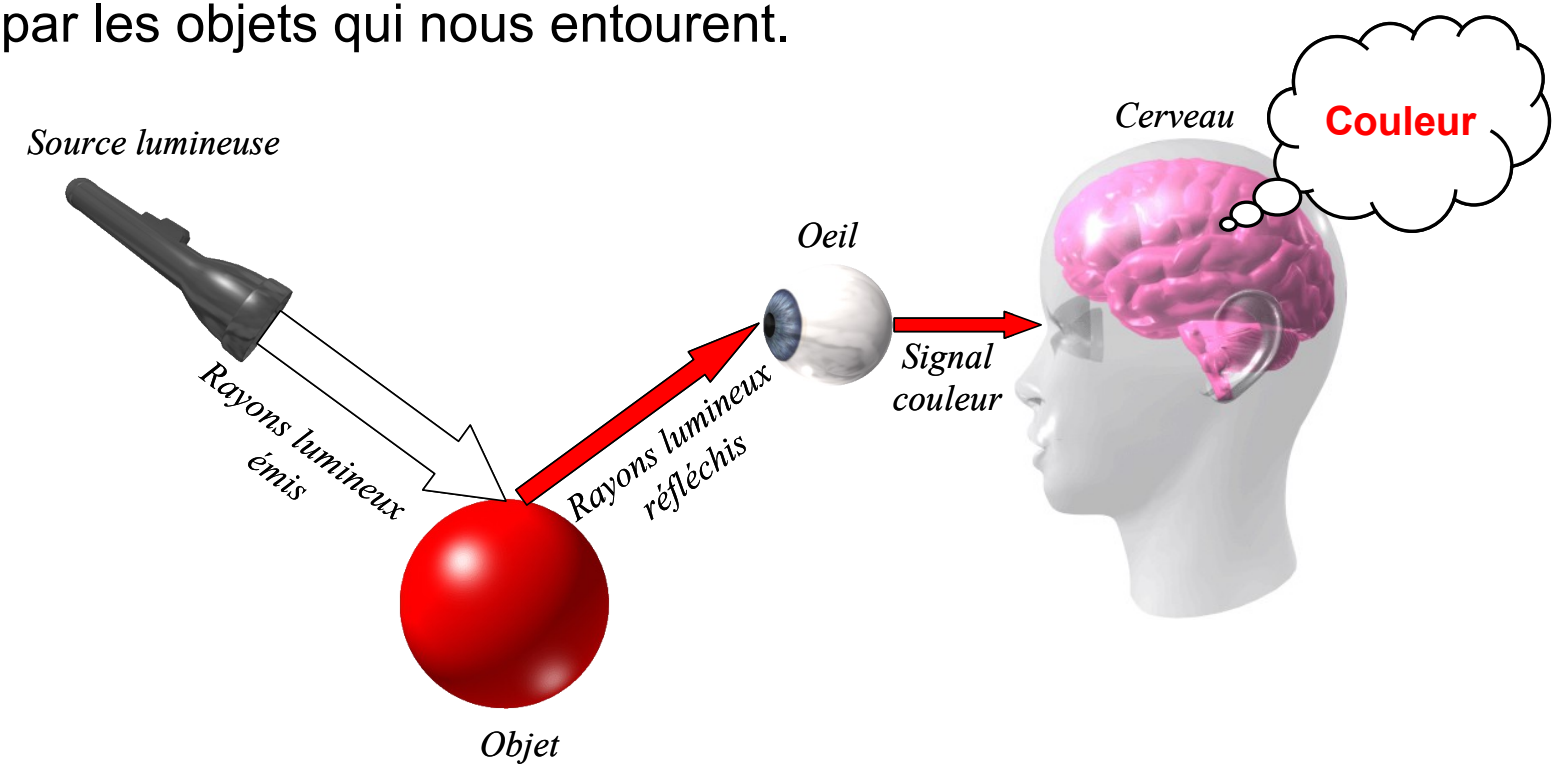
- **Tessellation control & evaluation shaders** : subdivision de la géométrie.
- **Geometry shader** : permet de créer de nouvelles primitives géométriques (points, lignes, triangles) à partir des données reçues du vertex shader.
- **Compute shader** : utilisation générale.



Réflexion de la lumière

La lumière se propage en ligne droite et tout objet opaque lui fait obstacle. Comme le son, la lumière peut rebondir sur une surface : ce phénomène est appelé **réflexion**.

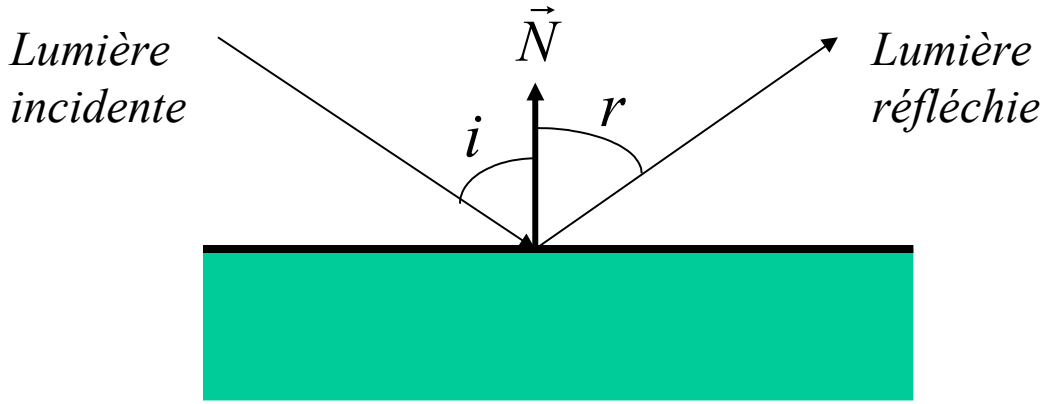
La majeure partie de la lumière qui atteint nos yeux a été réfléchie par les objets qui nous entourent.



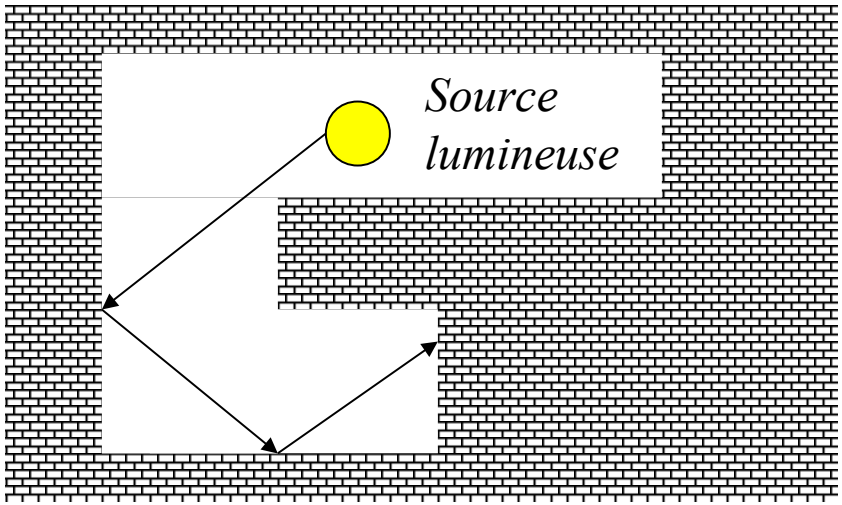
Loi de réflexion

Selon la loi de **Descartes – Snell**, la lumière incidente en un point d'une surface selon un certain angle par rapport à la normale de la surface en ce point est réfléchie selon le même angle.

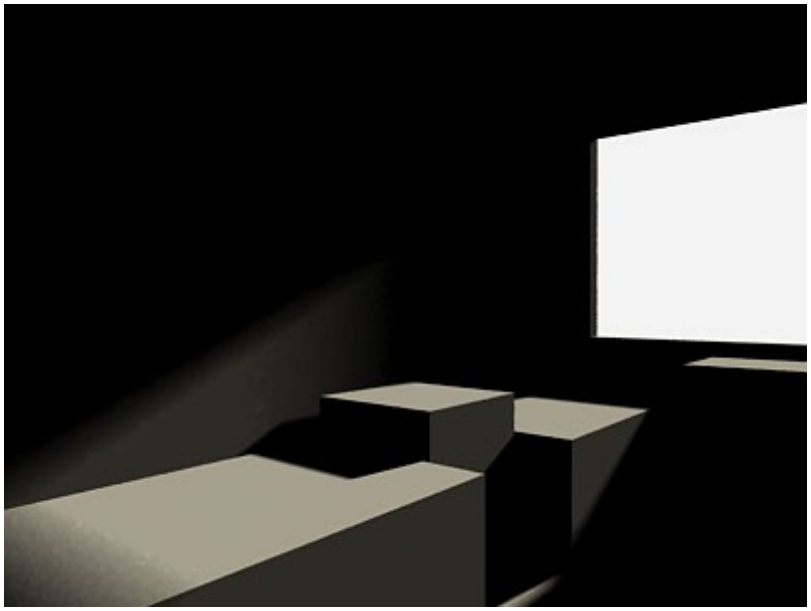
$$i = r$$



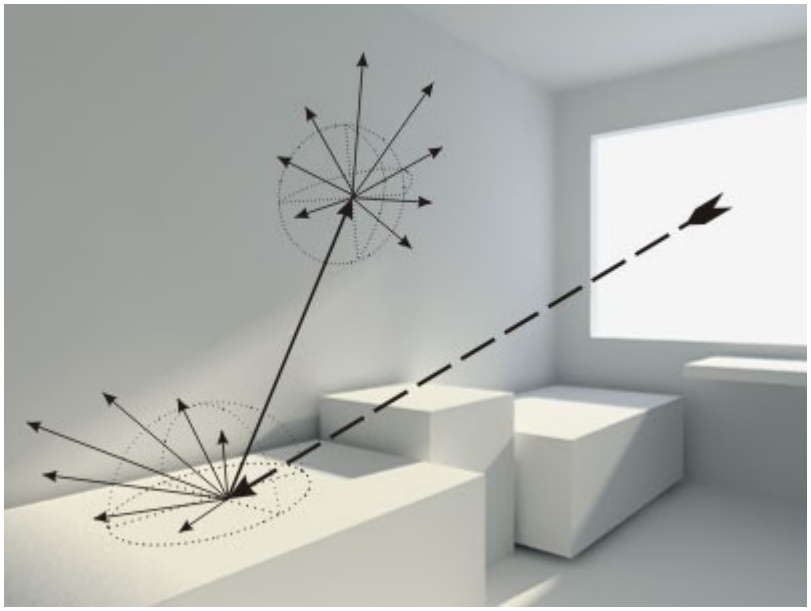
La lumière peut être réfléchie successivement par plusieurs surfaces et ainsi parvenir jusque dans des zones qui ne sont pas directement éclairées par les sources de lumière.



Dans un algorithme de synthèse d'image, si les rebonds successifs de la lumière ne sont pas pris en compte, les parties de la scène qui ne sont pas directement éclairées seront noires.



Illumination locale



Illumination globale

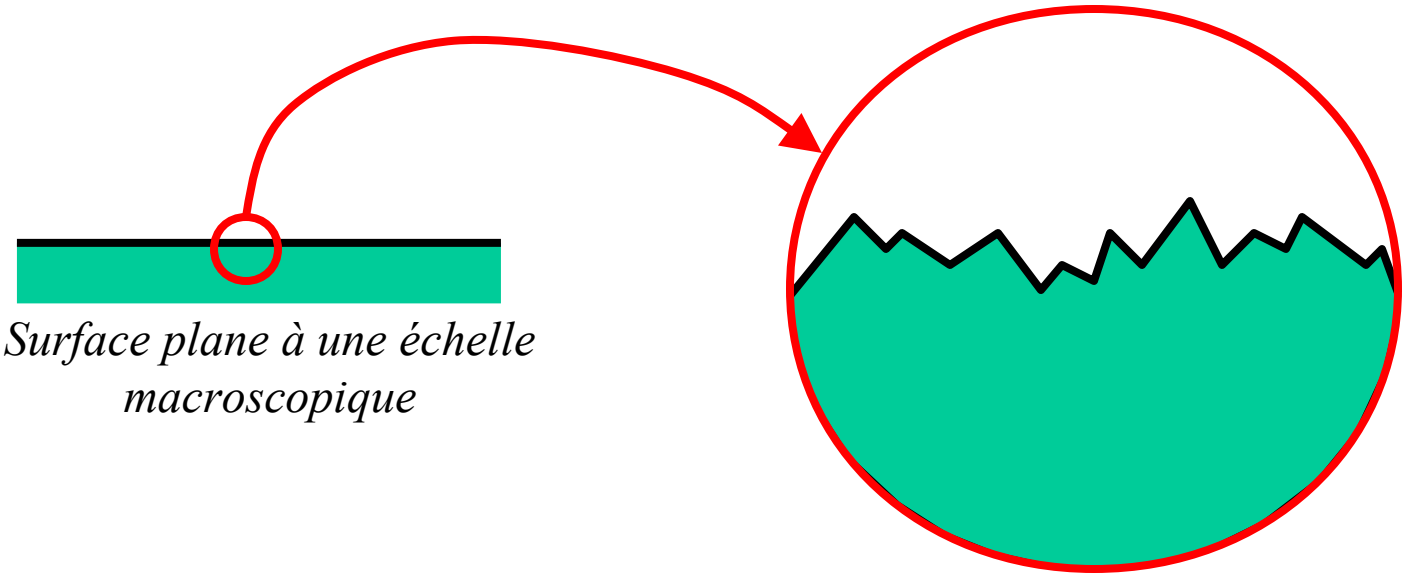
En synthèse d'image, ce mécanisme est simulé par diverses techniques d'**illumination globale** comme la *radiosité*, le *path tracing...*

OpenGL ne simule pas ces phénomènes, c'est une technique d'**illumination locale**. On imite l'effet de cet éclairage indirect avec la composante ambiante de la lumière et des matériaux.

Réflexion de la lumière dans la réalité

La lumière incidente sur une surface n'est réfléchie comme dans la loi de Descartes que dans le cas de miroirs parfaits.

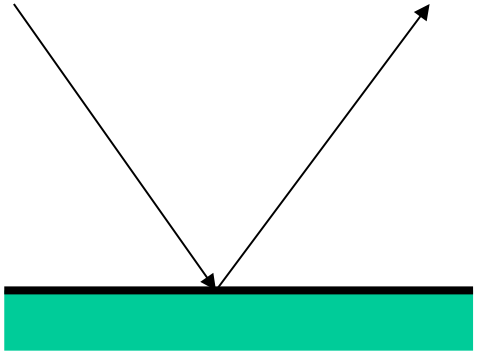
En pratique, les surfaces réelles ont des défauts à un niveau microscopique.



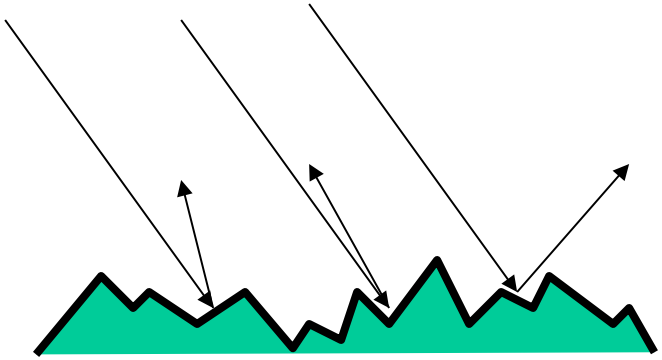
Surface plane à une échelle macroscopique

Agrandissement à une échelle microscopique montrant les imperfections de la surface.

→ La lumière n'est pas réfléchie dans une direction unique mais dans un **ensemble de directions** dépendant des propriétés microscopiques de la surface.

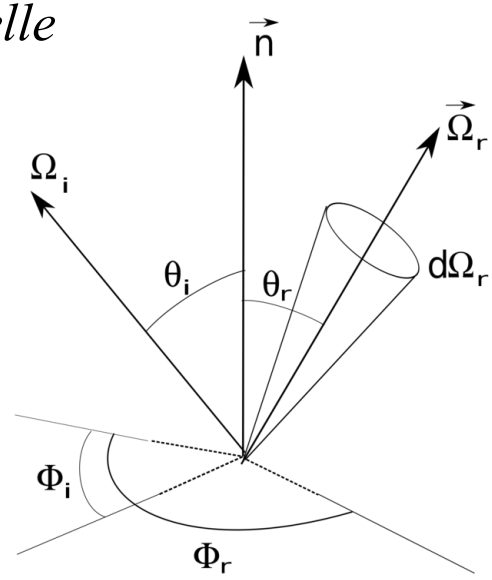


Miroir parfait théorique



Surface imparfaite réelle

La fonction mathématique qui donne les directions de réflexion de la lumière en fonction de la lumière incidente est appelée une **BRDF** (*Bidirectional Reflectance Distribution Function*).



Il existe de nombreuses BRDF : Lambert, Phong, Blinn, Ward, Cook-Torrance, Oren-Nayar, etc.

Elles sont plus ou moins réalistes ou permettent de représenter certaines surfaces particulières (tissu, cuir, matériaux anisotropiques, etc.)

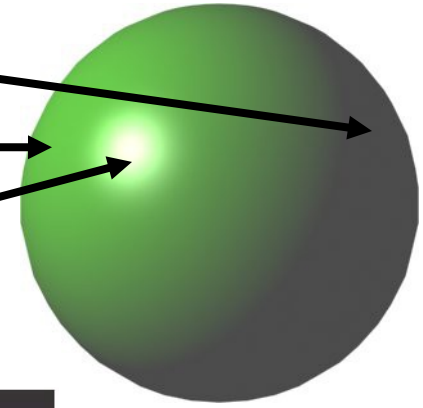
3. Calcul d'illumination locale

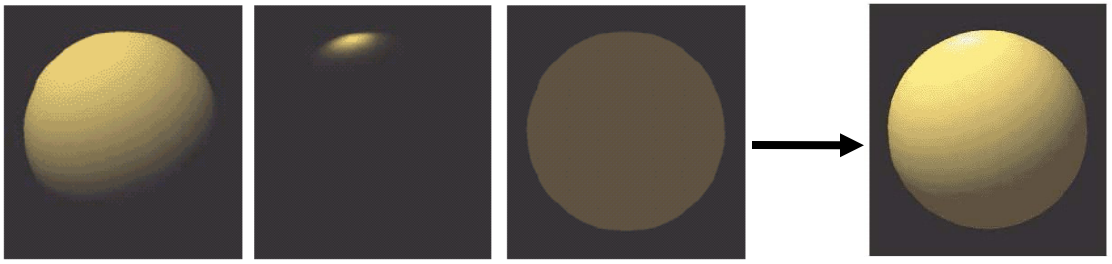
On va voir la théorie du **modèle d'illumination de Phong**, très utilisé par de nombreux systèmes de synthèse d'image, dont OpenGL. Les objets sont vus parce qu'ils réfléchissent la lumière. En tout point de l'objet, la couleur est donnée par la somme de :

Réflexion **ambiante** I_a

Réflexion **diffuse** I_d

Réflexion **spéculaire** I_s





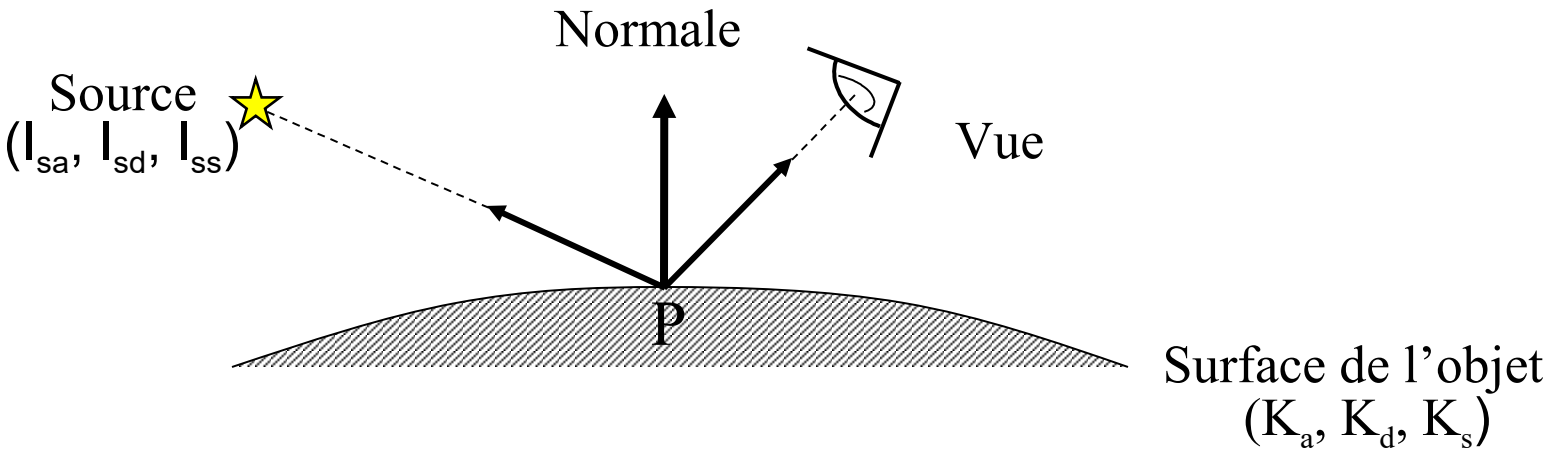
diffus + spéculaire + ambiant *Image finale*

Couleur affichée d'un pixel : $I = I_a + I_d + I_s$

3.1 Calcul de l'illumination

Le calcul de l'illumination (→la couleur) en un point P d'une surface nécessite :

- Le vecteur normal à la surface
- Le vecteur de direction de visualisation
- Le vecteur de direction de la source lumineuse I_{sa}, I_{sd}, I_{ss} : couleurs (RVB) **ambiante**, **diffuse** et **spéculaire** de la source lumineuse.
- K_a, K_d, K_s : couleurs (RVB) **ambiante**, **diffuse** et **spéculaire** du matériau de l'objet.



3.1.1 Réflexion ambiante

La couleur ambiante d'un objet ne dépend que du coefficient de réflexion ambiante K_a de l'objet, pas de sa position par rapport à la lumière.

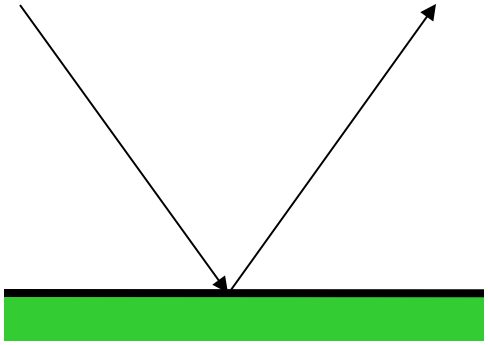
$$I_a = I_{sa} \cdot K_a$$

I_a : intensité de la lumière ambiante réfléchie

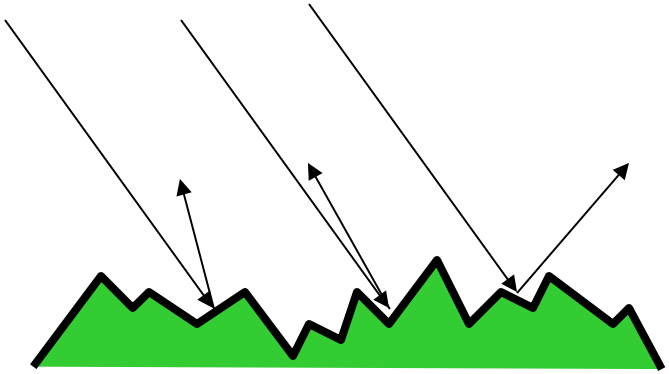
I_{sa} : intensité de la lumière ambiante

K_a : coefficient de réflexion ambiante de l'objet

La lumière n’est pas réfléchie dans une direction unique, avec un angle de réflexion = angle d'incidence (loi de Descartes) mais dans un **ensemble de directions** dépendant des propriétés microscopiques de la surface.



Miroir parfait théorique

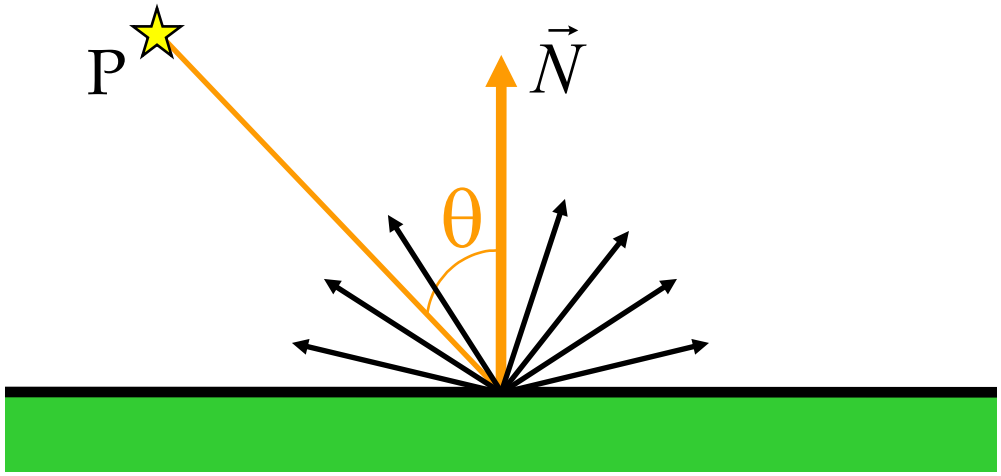


Surface imparfaite réelle

→ Ces directions sont réparties selon une composante **diffuse** et une composante **spéculaire**, que l’on va ajouter à la composante ambiante pour donner plus de relief à l’objet affiché.

3.1.2 Réflexion diffuse

- Matériaux mats (craie, plâtre, etc.)
- La lumière de la source est réfléchie par l'objet dans toutes les directions (réflexion isotrope).
- La couleur de l'objet est indépendante de la position de l'observateur.
- Ne dépend que de l'angle θ entre la direction de la source et la normale, et du coefficient de réflexion diffuse K_d de l'objet (loi de Lambert)



Loi de Lambert :

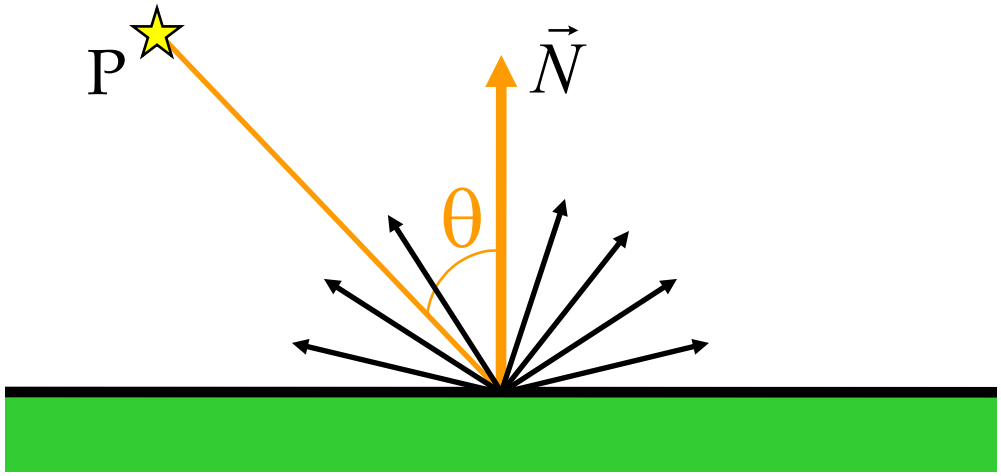
$$I_d = I_{sd} \,.\, K_d \,.\, \cos\theta$$

I_d : intensité de la lumière diffuse réfléchie

I_{sd} : intensité de la lumière diffuse

K_d : coefficient de réflexion diffuse du matériau

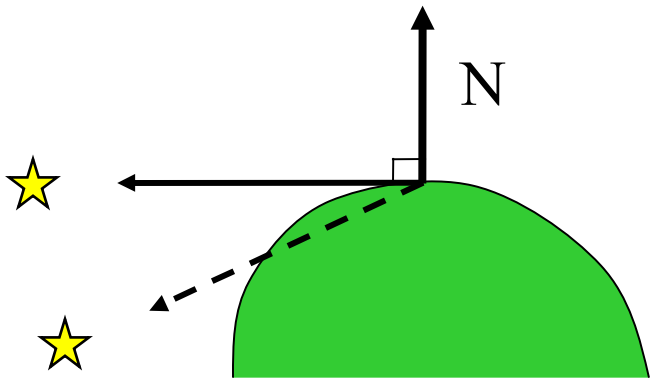
θ : angle entre la source de lumière et la normale



Notes :

$$I_d = I_{sd} \cdot K_d \cdot \cos \theta$$

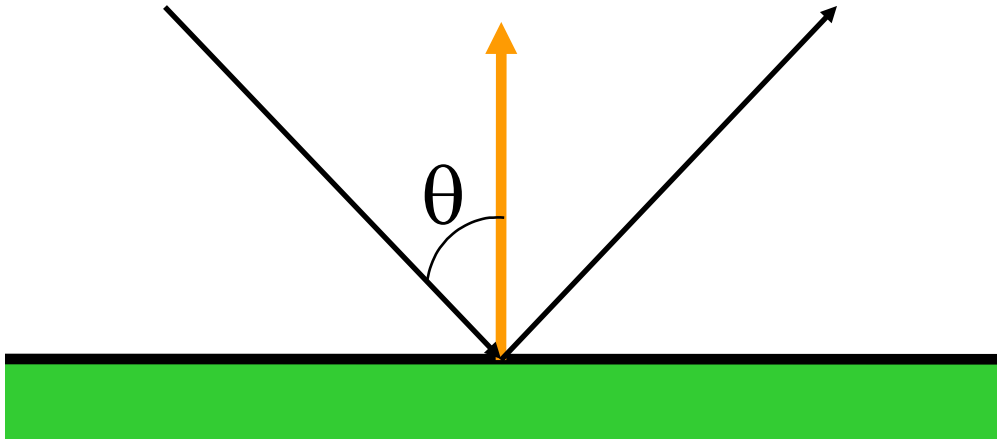
- L'intensité diffuse est maximale pour $\theta = 0$ (source de lumière à la verticale de la surface)
- Elle est nulle pour un éclairage rasant ($\theta=90°$)
- Si $\theta > 90°$ alors le point n'est pas visible par la source de lumière.



3.1.3 Réflexion spéculaire

- Permet d'obtenir des reflets
- **Miroir parfait** → Loi de Descartes :

La lumière qui atteint un objet est réfléchie dans la direction faisant le même angle avec la normale

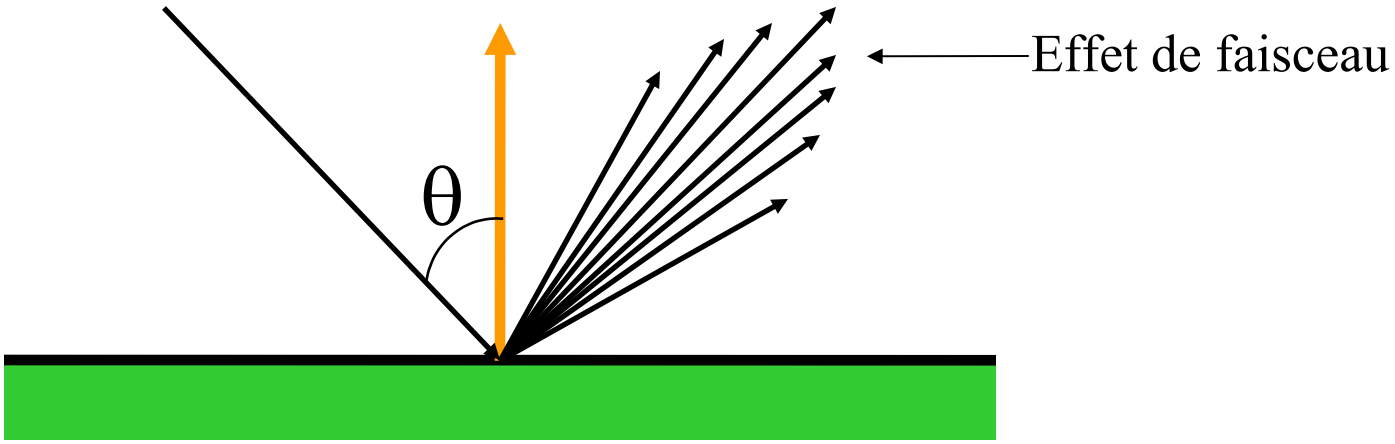


En réalité, les surfaces ne sont pas des miroirs parfaits.

→ Réflexion spéculaire : miroir imparfait

→ La lumière est réfléchie principalement dans la direction de réflexion miroir parfaite

→ L'intensité de la lumière réfléchie diminue lorsqu'on s'éloigne de cette direction parfaite.



Réflexion spéculaire dans le modèle de Phong

$$I_s = I_{ss} \,.\, K_s \,.\, (\cos \alpha)^n$$

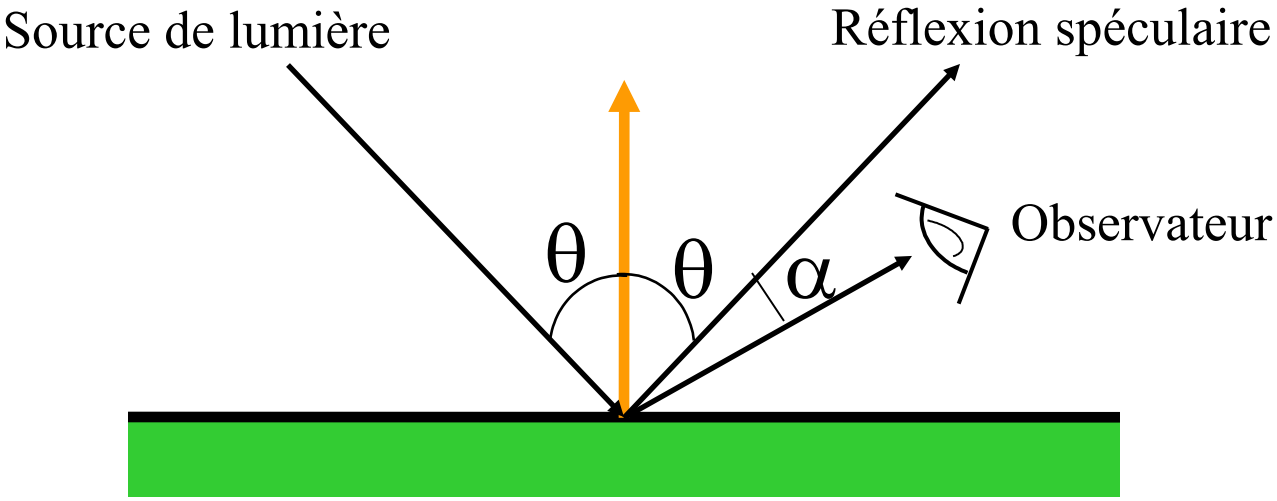
I_s : intensité de la lumière spéculaire réfléchie

I_{ss} : intensité de la lumière spéculaire de la source

K_s : coefficient de réflexion spéculaire du matériau

α : angle entre les directions de réflexion et de vue

n : coefficient de brillance



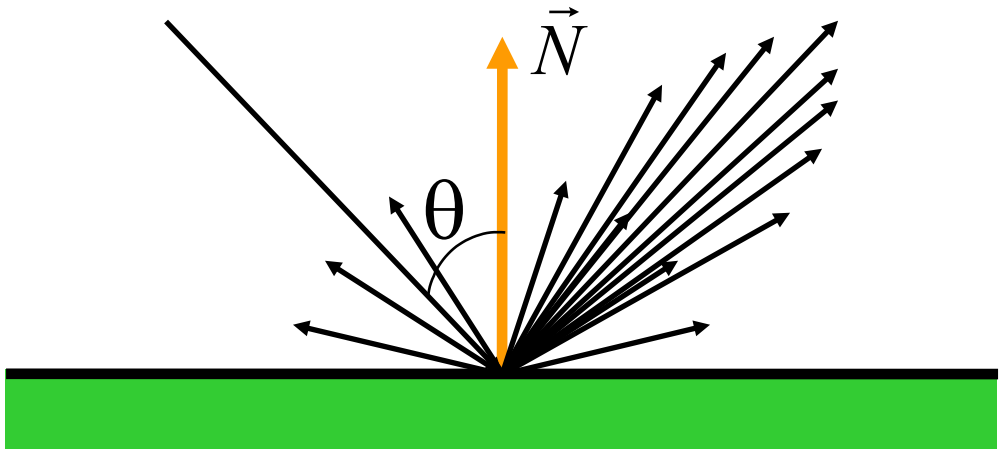
3.2 Modèle de Phong final

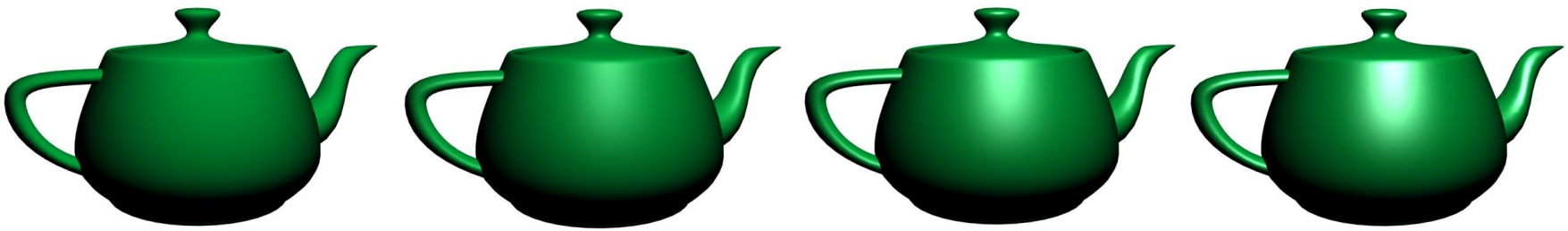
- C'est la somme des réflexions **ambiante**, **diffuse** et **spéculaire** :

$$I = I_a + I_d + I_s$$

➔ $I = I_{sa} \cdot K_a + I_{sd} \cdot K_d \cdot \cos\theta + I_{ss} \cdot K_s \cdot (\cos\alpha)^n$

- Les proportions de réflexion diffuse et spéculaire dépendent du matériau. Certains sont plus diffus (craie, papier, etc.) que spéculaires (métal, verre, etc.)





Diffus

(« mat »)

Spéculaire

(« brillant »)

Réflexion finale

Calcul final de la couleur

On additionne l'intensité lumineuse de chacune des composantes de la couleur.

Dans le système RVB, on ajoute les **intensités rouge, verte et bleue**.

On définit pour chacune de ces 3 composantes :

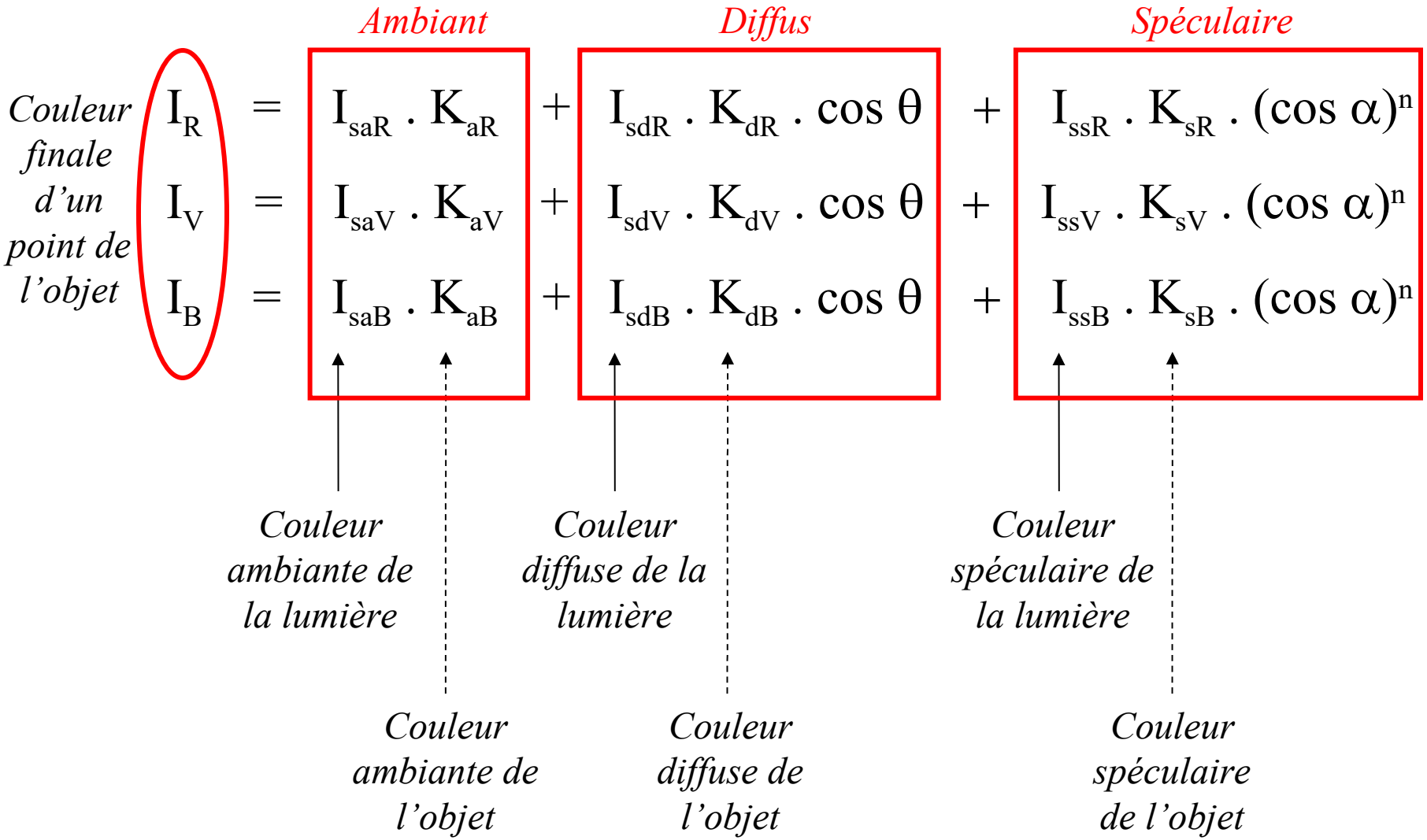
- Les caractéristiques des sources de lumière :

$$I_{saR}, I_{saV}, I_{saB} \;;\; I_{sdR}, I_{sdV}, I_{sdB} \;;\; I_{ssR}, I_{ssV}, I_{ssB}$$

- Les caractéristiques des matériaux :

$$K_{aR}, K_{aV}, K_{aB} \;;\; K_{dR}, K_{dV}, K_{dB} \;;\; K_{sR}, K_{sV}, K_{sB}$$

Les intensités lumineuses pour chacune des 3 composantes R,V,B s'obtiennent donc par :



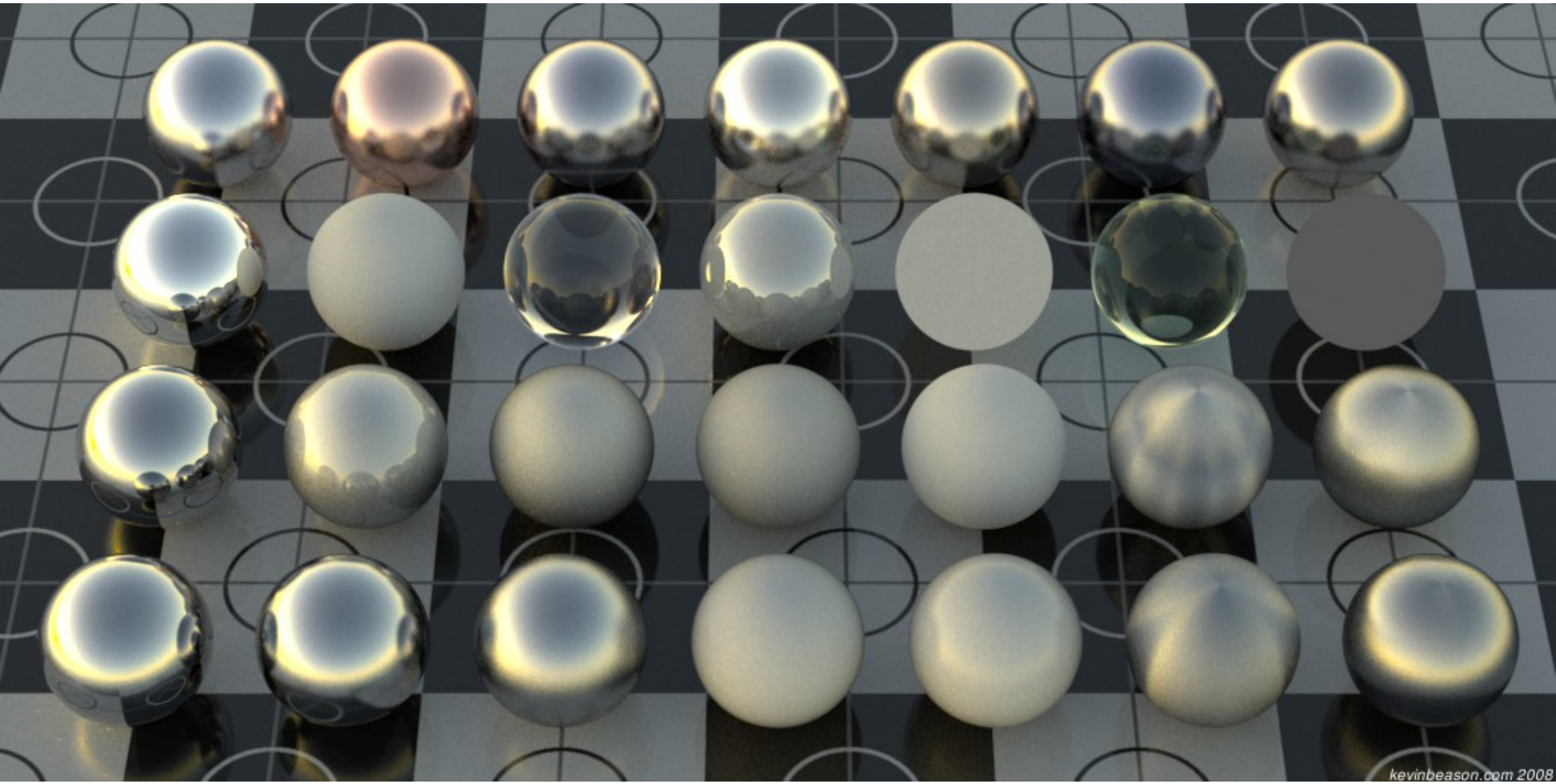
Avantages du modèle de Phong :

- Très pratique (simple à utiliser, résultats intéressants)
- Rapide à calculer

Désavantages :

- Pas de sens physique
- Pas de lien avec les propriétés du matériau (rugosité, …)

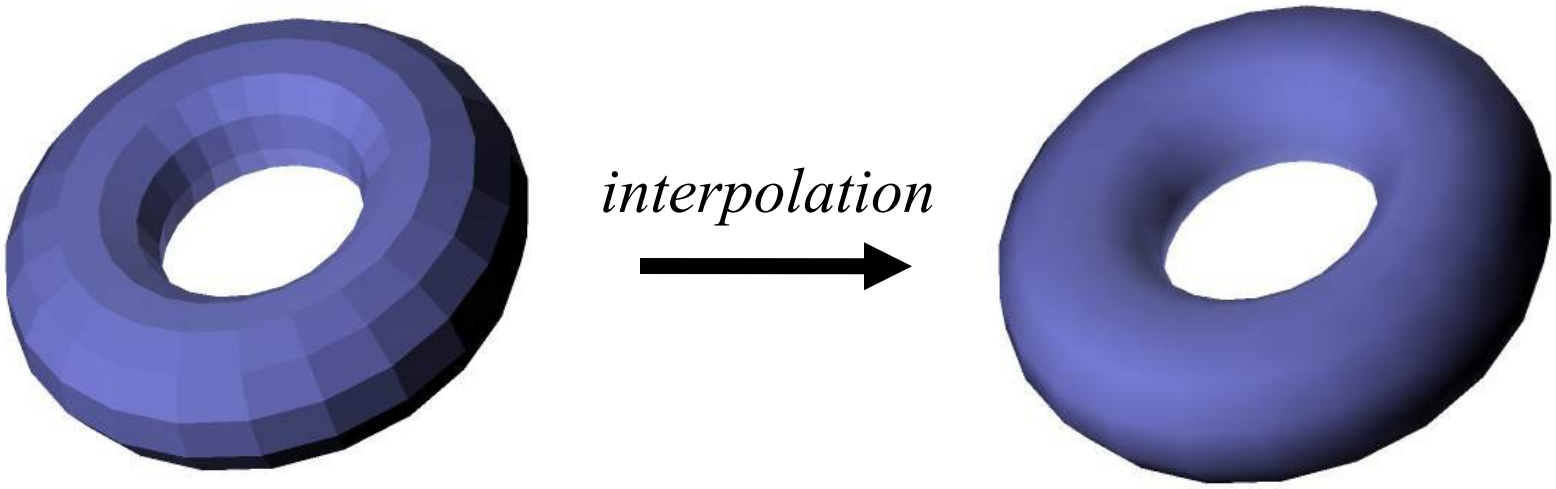
Plutôt qu'une simple illumination de Phong, on peut utiliser d'autres modèles de BRDF plus réalistes mais plus complexes (Cook-Torrance, etc.) en reprogrammant le pipeline avec des **shaders** et ainsi simuler l'apparence de matériaux plus complexes (*voir cours sur les shaders*).



4. Utilisation de l'illumination sur un objet 3D

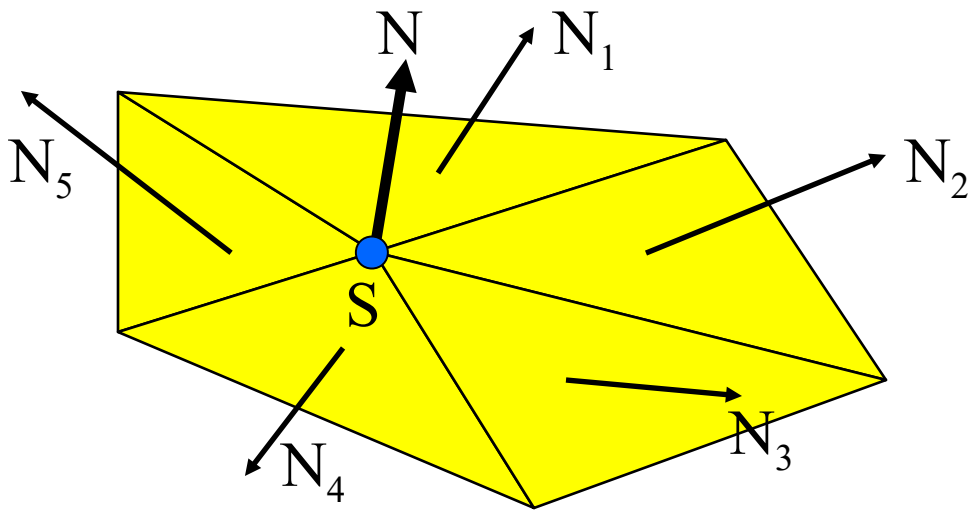
Problème si on utilise uniquement la normale d'un polygone : l'illumination sera la même en chacun de ses pixels

→ affichage « plat » (= « *flat shading* »)



→ Solution : calculer une normale pour chaque sommet d'un polygone, on obtient ainsi une couleur différente pour chacun d'entre eux, puis interpoler ces couleurs.

Normale d'un sommet d'un polygone = moyenne des normales des polygones ayant ce sommet en commun.



Normale au sommet S :

$$N = (N_1 + N_2 + N_3 + N_4 + N_5) \ / \ 5$$

4.1 Interpolation de l'illumination

But : calculer une couleur pour chaque pixel visible à l'écran de l'objet 3D qu'on affiche.

- Lissage de **Gouraud** : interpolation de couleurs

Calculer pour chaque sommet du polygone une couleur au moyen d'un modèle d'illumination (Phong, …), puis interpoler les **couleurs** des sommets pour calculer la couleur de chaque pixel du polygone.

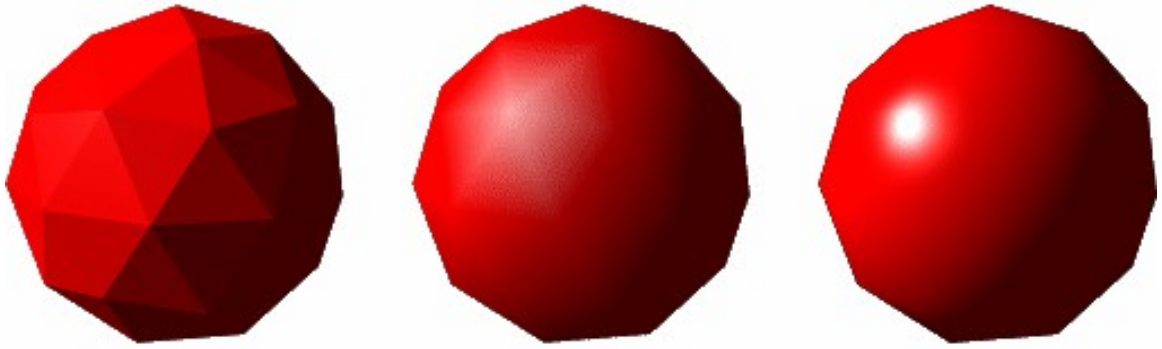
- Lissage de **Phong** : interpolation de normales

Pour chaque pixel, on interpole les normales des sommets, puis on calcule l'illumination pour chaque pixel avec la normale interpolée.

Note : Attention, ne pas confondre « Modèle d'illumination de Phong » et « Lissage de Phong ».

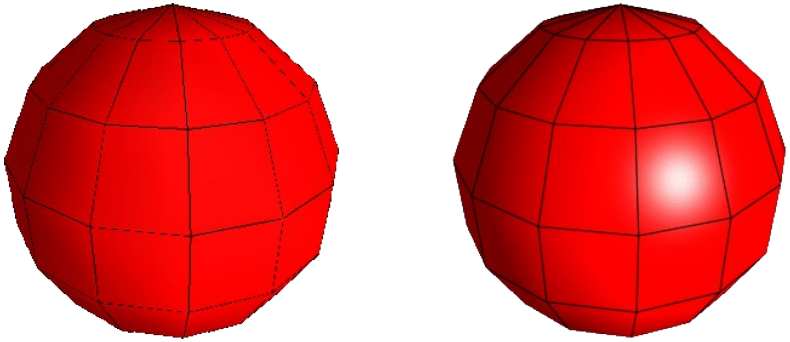
Bilan

- Lissage de Phong plus lent que celui de Gouraud (calcul de l'illumination en chaque pixel, au lieu de chaque sommet), mais nettement plus beau.



Flat *Gouraud* *Phong*

- Permet de calculer les effets spéculaires contenus dans une facette, contrairement au lissage de Gouraud.



Gouraud *Phong*

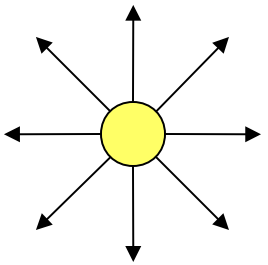
4.2 Sources lumineuses en OpenGL

Toute source lumineuse est définie par 3 vecteurs de 4 composantes (rouge, vert, bleu, alpha) :

- **lumière ambiante** (valeur par défaut <0, 0, 0, 1>)
- **lumière diffuse** (valeur par défaut <1, 1, 1, 1>)
- **lumière spéculaire** (valeur par défaut <1, 1, 1, 1>)

3 types de sources lumineuses sont supportés :

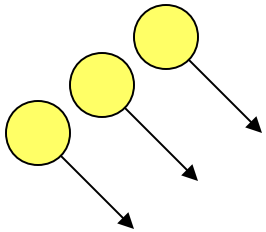
1) Source ponctuelle



La lumière vient d'un **point** spécifique.
Coordonnées homogènes d'un **point** :

```
GLfloat position[] = { 10.0, 20.0, 5.0, 1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

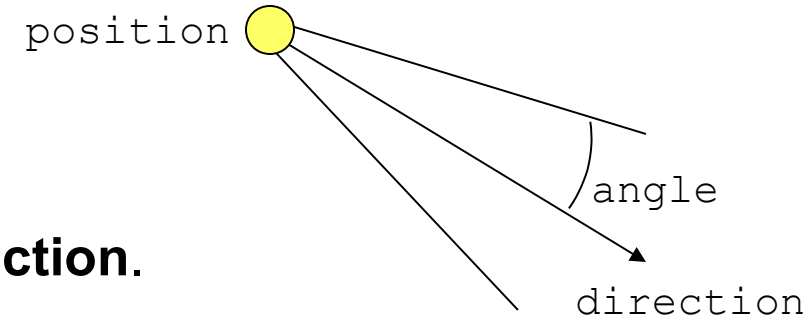
2) Source directionnelle



La lumière vient d'une **direction** spécifique.
Coordonnées homogènes d'un **vecteur** :

```
GLfloat direction[] = { 3.0, 5.0, 2.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, direction);
```

3) Spot



La lumière vient d'un **point** spécifique,
avec une intensité qui dépend de la **direction**.

– **Position** : emplacement de la source :

```
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

– **Direction** : axe central de la lumière :

```
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction);
```

– **Angle** : largeur du rayon :

```
glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

– **Puissance** : atténuation de la lumière aux bords du cône :

```
glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, 1.0);
```

Utilisation :

```
// Valeurs de couleur (ici, lumière rouge)
GLfloat Light0Amb[4] = {0.5f, 0.0f, 0.0f, 1.0f};
GLfloat Light0Dif[4] = {1.0f, 0.0f, 0.0f, 1.0f};
GLfloat Light0Spec[4]= {0.2f, 0.2f, 0.2f, 1.0f};

// Valeur de position (ici, source ponctuelle)
GLfloat Light0Pos[4] = {5.0f, 20.0f, 10.0f, 1.0f};
```

```
// Fixe les paramètres de couleur de la lumière 0
glLightfv(GL_LIGHT0, GL_AMBIENT, Light0Amb);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Light0Dif);
glLightfv(GL_LIGHT0, GL_SPECULAR, Light0Spec);

// Fixe la position de la lumière 0
glLightfv(GL_LIGHT0, GL_POSITION, Light0Pos);

// Active l'éclairage
glEnable(GL_LIGHTING);

// Active la lumière 0
glEnable(GL_LIGHT0);
```

Note :

Le nombre de lumières dans OpenGL est limité. Il est au minimum de 8, le nombre max étant donné par la constante **GL_MAX_LIGHTS**.

`glEnable(GL_LIGHTING)` permet d'activer l'éclairage de manière générale.

`glEnable(GL_LIGHT0)` permet d'activer l'éclairage de la 1ère source de lumière.

On peut désactiver l'éclairage :

- Général : **`glDisable(GL_LIGHTING)`**
- D'une source particulière : **`glDisable(GL_LIGHT0)`**

4.3 Matériaux en OpenGL

Tout matériau est défini par 3 vecteurs de 4 composantes (rouge, vert, bleu, alpha) :

- **coefficient de réflexion ambiant** (valeur par défaut <0, 0, 0, 1>)

- **coefficient de réflexion diffus** (valeur par défaut <1, 1, 1, 1>)

- **coefficient de réflexion spéculaire** (valeur par défaut <1, 1, 1, 1>)

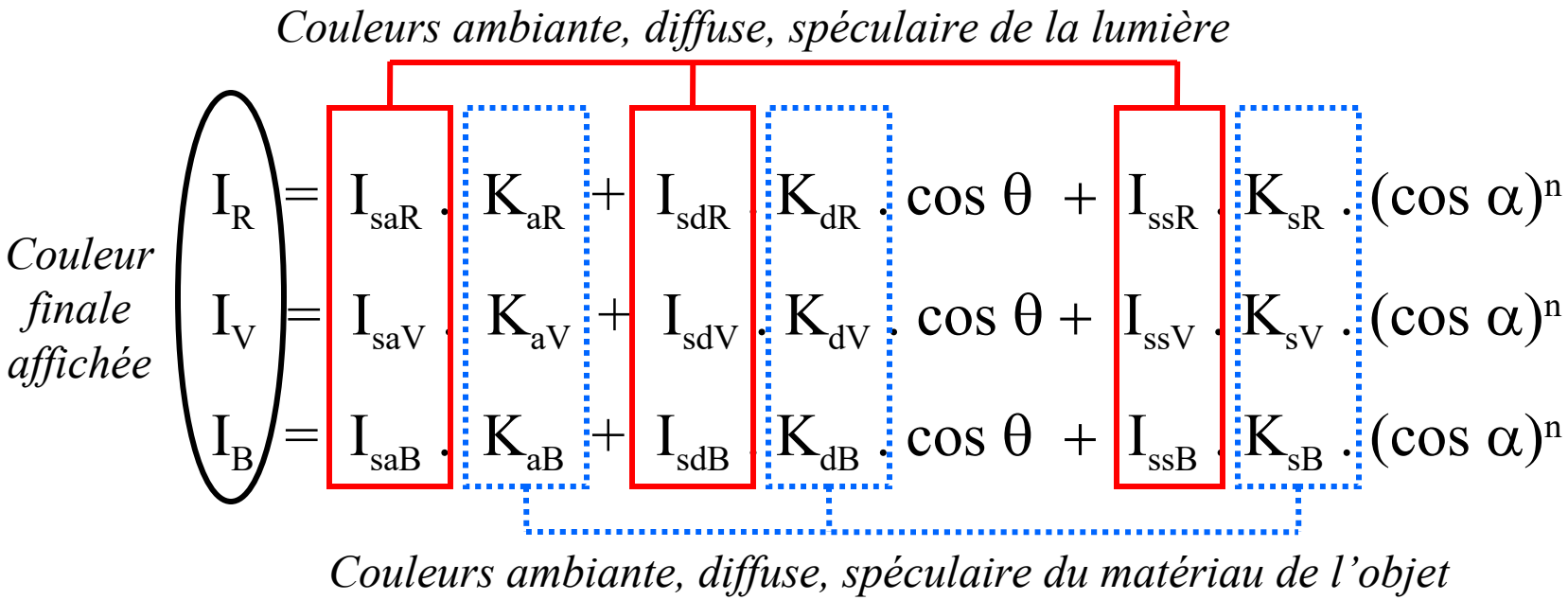
- Ainsi que le **coefficient de brillance** n du $(\cos \alpha)^n$ de la réflexion spéculaire.

Utilisation :

```
GLfloat MatAmbient[4] = {0.0f, 0.5f, 0.0f, 1.0f};
GLfloat MatDiffuse[4] = {0.0f, 0.8f, 0.0f, 1.0f};
GLfloat MatSpecular[4] = {0.2f,0.2f, 0.2f, 1.0f};
GLfloat MatShininess[] = { 5.0F };

glMaterialfv(GL_FRONT, GL_AMBIENT , MatAmbient);
glMaterialfv(GL_FRONT, GL_DIFFUSE , MatDiffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, MatSpecular);
glMaterialfv(GL_FRONT, GL_SHININESS, MatShininess);
```

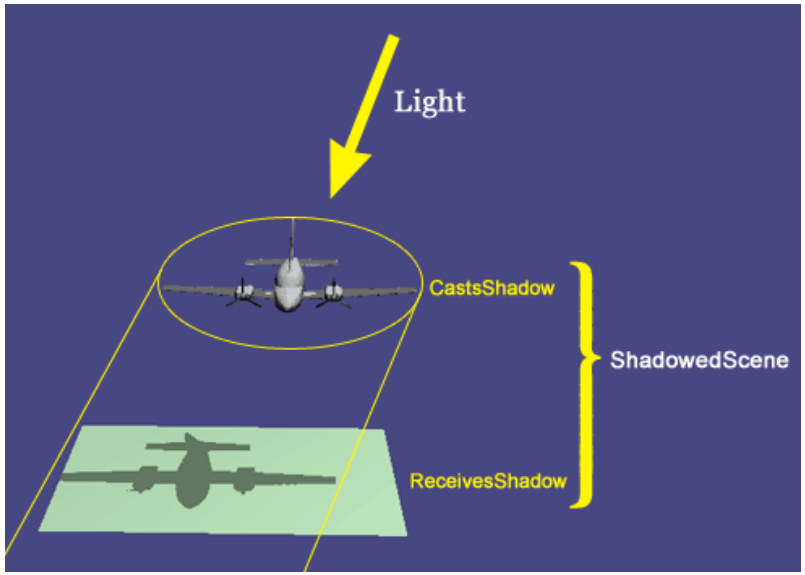
BILAN

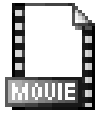
Étant donné les couleurs ambiante, diffuse, spéculaire de la lumière, ainsi que les composantes ambiante, diffuse, spéculaire du matériau d'un objet, la couleur finale apparaissant à l'écran de cet objet sera calculée grâce à l'équation du modèle de Phong que nous avons vu:



4.4. Ombres

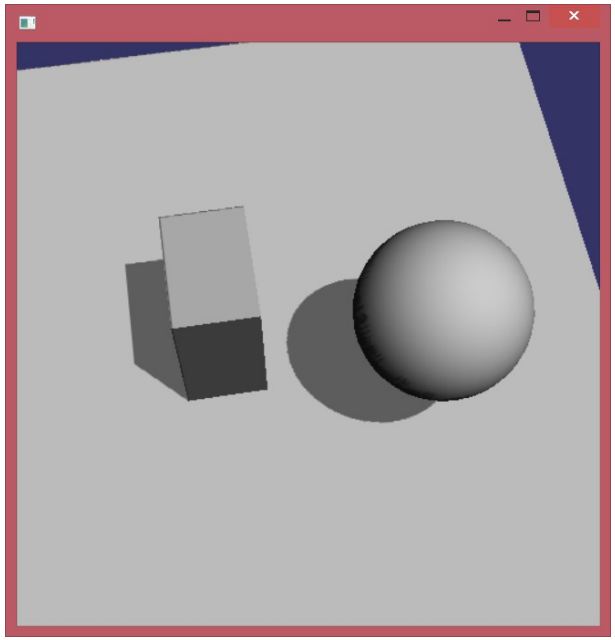
De base, OpenGL ne gère que l'ombrage des faces mais pas les ombres portées, c'est-à-dire les ombres projetées par des objets sur d'autres objets (ou sur eux-mêmes) en fonction de la position de la lumière.

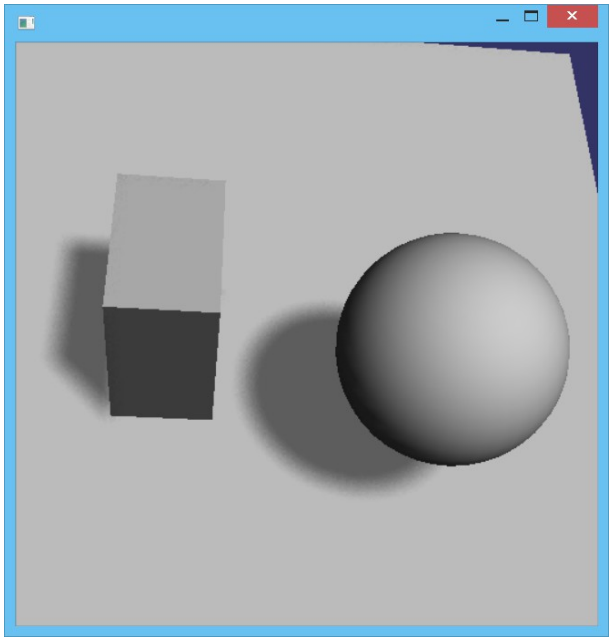




On peut les rajouter avec des techniques comme les Shadow maps, les Soft shadows, ...

https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping





4.5 Brouillard

OpenGL permet d'appliquer très facilement des effets de brouillard.

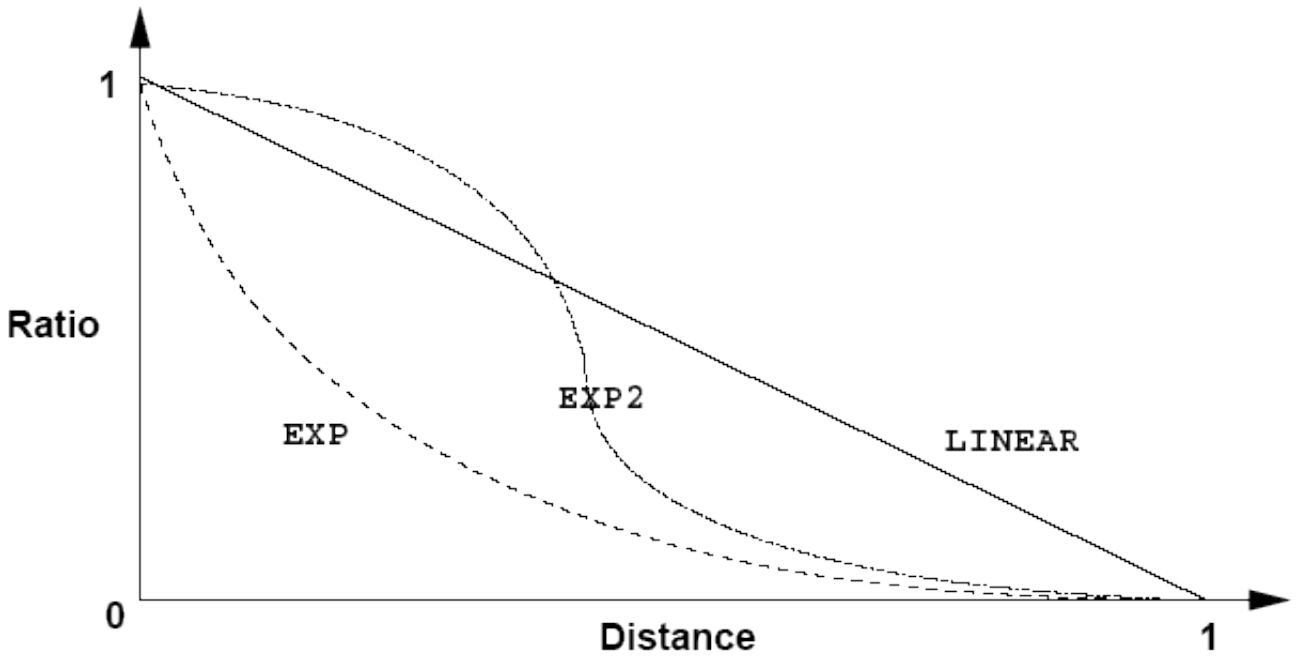
Scène sans brouillard

Scène avec brouillard

La couleur du brouillard est mélangée à celle des objets en fonction de la distance selon 3 modes :

- atténuation linéaire
- atténuation exponentielle
- atténuation exponentielle 2

$$\texttt{GL\_LINEAR} : f = \frac{end - z}{end - start}$$
$$\texttt{GL\_EXP} : f = e^{-(density.z)}$$
$$\texttt{GL\_EXP2} : f = e^{-(density.z)^2}$$



4.4.1 Atténuation linéaire

On fournit la distance de début et de fin du brouillard. Entre les deux, la couleur du brouillard est mélangée à celle des objets linéairement en fonction de la distance.

```
GLfloat fogColor[4]= {0.4f,0.4f,0.4f,0.0f};

glFogf(GL_FOG_MODE, GL_LINEAR);

glFogf(GL_FOG_START, 100);    // défaut : 0.0f

glFogf(GL_FOG_END, 800);      // défaut : 1.0f

glFogfv(GL_FOG_COLOR, fogColor);

glEnable(GL_FOG);
```

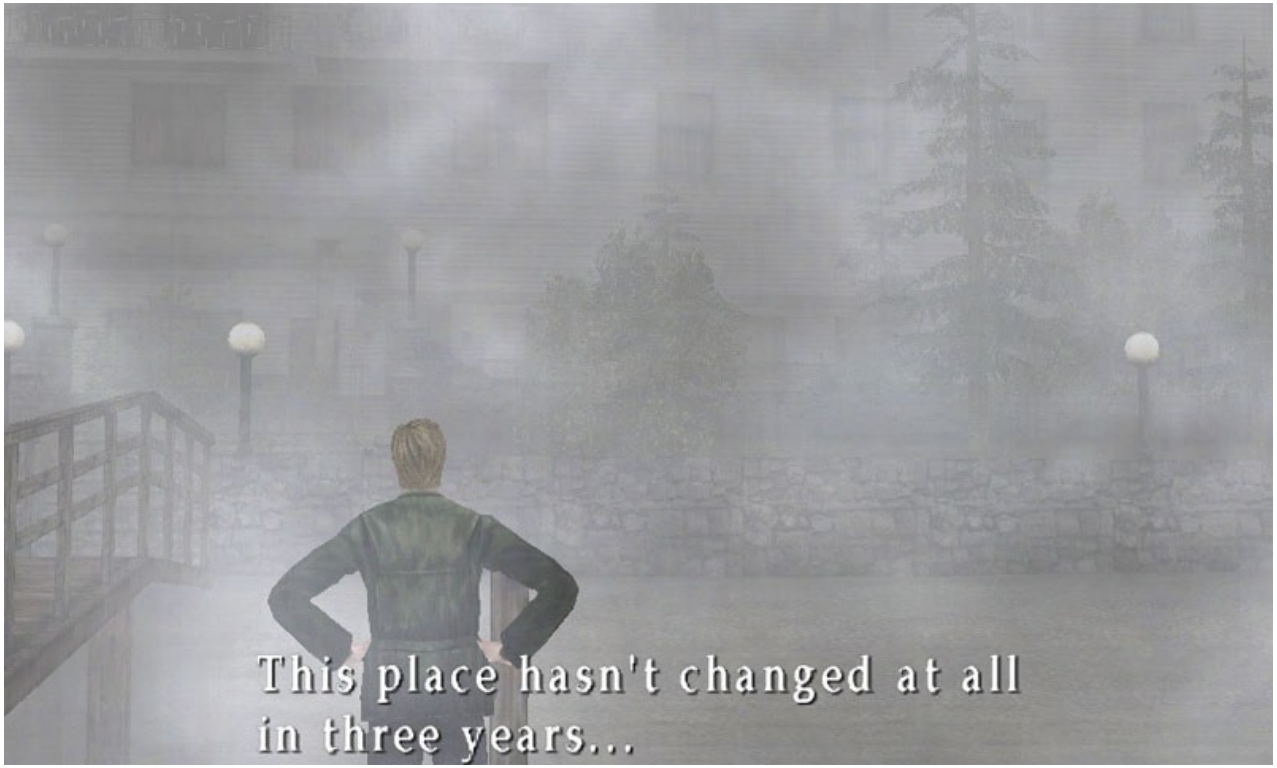
4.4.2 atténuation exponentielle

On fournit la densité du brouillard. La couleur du brouillard est mélangée à celle des objets exponentiellement en fonction de la distance.

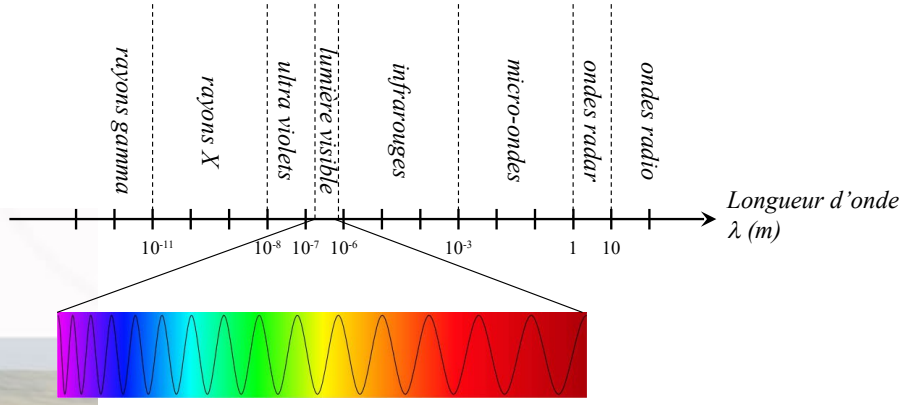
```
GLfloat fogColor[4]= {0.4f,0.4f,0.4f,0.0f};

glFogf(GL_FOG_MODE, GL_EXP);  // ou GL_EXP2

glFogf(GL_FOG_DENSITY, 2.0f); // défaut : 1.0f

glFogfv(GL_FOG_COLOR, fogColor);

glEnable(GL_FOG);
```

Intérêt du brouillard

- Permet de reproduire un phénomène naturel.
- Permet de donner une ambiance (angoissante, mystérieuse, etc.).



- Permet de donner plus d'effet de profondeur en simulant le « bleu atmosphérique » : atténuation de la lumière dans l'air due aux gouttelettes d'eau en suspension, impuretés, etc.



Ce sont d'abord les longueurs d'ondes les plus longues qui sont atténuées (rouge, etc.) et c'est le bleu qui va le plus loin.

- Le brouillard peut aussi être utilisé pour simuler l'atténuation de la lumière sous l'eau. Utiliser un brouillard de couleur bleue pour reproduire le fait que les longueurs d'onde correspondant au bleu sont absorbées en dernier.

- Un effet de brouillard peut aussi être utilisé pour limiter la distance de vue et ne pas être obligé d'afficher les objets trop éloignés (car invisibles, dans le brouillard)

  → réduction du nombre d'objets à afficher, donc **accélération de l'affichage**.