Synthèse d'images

Illumination

Gilles Gesquière

Cours s'appuie sur
- les supports de cours de S. Thon (LSIS, Université Aix Marseille)
- les supports de thèse de F. Graglia (Université Aix-Marseille)

0- Introduction

- Capacité des ordinateurs à afficher des images
  - intérêt pour la CAO, synthèse d'images, visualisation scientifique, multimédia, jeux, …
- Amélioration des images au cours du temps
  - Images de plus en plus précises (résolution, nombre de couleurs).
  - Puissance de calcul plus élevée.
  - Algorithmes plus sophistiqués.

1950 1963 1981 2001

0- Introduction

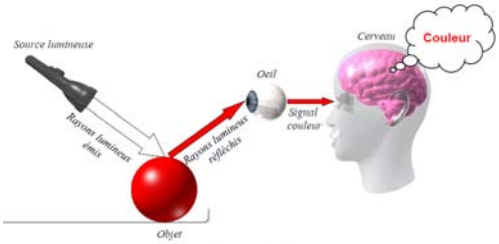
- Image de synthèse proche du réalisme du monde réel

0- Introduction

- De nombreux phénomènes de la réalité sont pris en compte :
  - réflexion de la lumière
  - Réfraction
  - Ombres
  - Effets de la matière
  - …

0- Introduction

Ce réalisme est dû à une modélisation plus ou moins fidèle des interactions entre la lumière et les objets, ainsi que la perception de cette lumière par le système visuel humain.

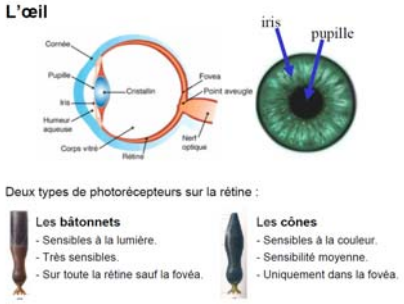


Principe de la perception visuelle humaine

0- Introduction

- Il n'y a pas de couleur sans lumière : la source lumineuse éclairant un objet est la première cause de sensation de la couleur de l'objet observé. Le matériau dont est composé l'objet réfléchit ou transmet les rayons lumineux émis par la source qui sont alors captés par l'oeil. Ces rayons forment le *stimulus de couleur*.
- En fait, un matériau n'a pas de couleur intrinsèque mais transforme les propriétés de la lumière. L'oeil transforme le stimulus reçu en un *signal couleur* qu'il transmet au cerveau. Ce dernier identifie la couleur de l'objet observé et, au delà, interprète l'objet.

0- Introduction



0- Introduction

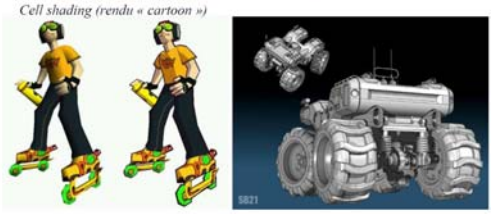
- Les images de synthèse ont généralement deux utilisations :
  - **3D précalculée**
    - Ex: images fixes, film d'animation, effets spéciaux, …
    - Produit avec un logiciel de synthèse d'images (3D Studio Max, Maya, Lightwave, …).
    - L'ordinateur peut passer plusieurs minutes ou heures pour calculer 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).

0- Introduction

Remarque:

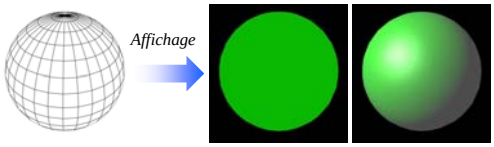
on ne cherche pas toujours le réalisme (NPR : *Non Photorealistic Rendering*) : effet artistique, illustration technique, etc.

Cell shading (rendu « cartoon »)



1. Problématique

Utiliser seulement la couleur de l'objet ne donne pas de résultat réaliste.



→ Il faut prendre en compte les interactions de la lumière avec les surfaces des objets.

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, ...

Objet 3D — *Prise en compte de la lumière* — *Application de texture*

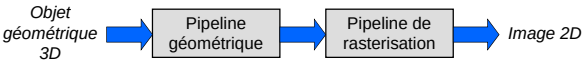
➔ **« Rendu »** (réaliste ou non)

2. Pipeline graphique

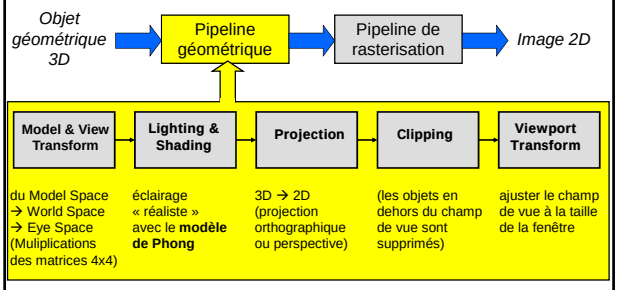
Suite des traitements appliqués à un objet (ou un ensemble d'objets) géométrique(s) dans le but d'en obtenir le rendu.

En entrée :
un objet géométrique 3D.

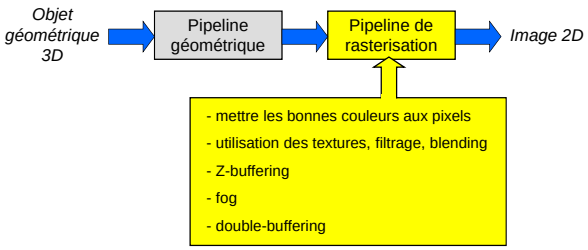
En sortie :
une image 2D.



Pipeline graphique



Pipeline graphique

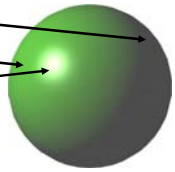


3. Calcul d'illumination locale

On va voir la théorie du modèle d'illumination de Phong, utilisé par OpenGL.

Les objets sont vus parce qu'ils réfléchissent la lumière.

- Réflexion ambiante
- Réflexion diffuse
- Réflexion spéculaire



→ Permet de calculer la couleur d'un objet en tout point.

3. Calcul d'illumination locale

Dans ce qui suit, on parlera d'« intensité » de la lumière, notée I.

→ I = niveaux de gris.

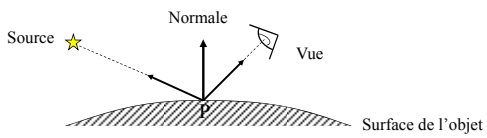
→ Pour obtenir une couleur, on calculera une valeur d'intensité pour R,V,B.

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

3.1 Ingrédients géométriques pour le 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



→ On peut calculer les 3 éléments constitutifs de la couleur d'un objet, résultant de la lumière que réfléchit sa surface :

- Réflexion **ambiante**
- Réflexion **diffuse**
- Réflexion **spéculaire**

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.

$$I_a = I_{sa} . K_a$$

I_a : intensité de la lumière ambiante réfléchie

I_{sa} : intensité de la lumière ambiante

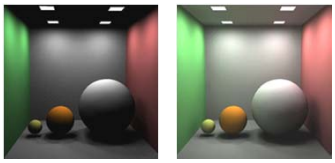
K_a : coefficient de réflexion ambiante de l'objet

On augmente K_a

Affichage d'un objet dont on ne prend en compte que la réflexion ambiante pour le calcul de la couleur

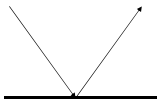
Modèle très primitif :

- Pas de sens physique
- La forme des objets est invisible
- Permet de masquer partiellement les problèmes dus à l'illumination locale (qui nécessitent une illumination globale pour être résolus)

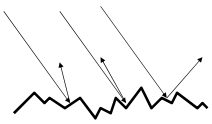


Images obtenues par illumination globale. Pour l'illumination locale, la composante ambiante permet de simuler une couleur en l'absence d'éclairage direct (figure de droite).

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

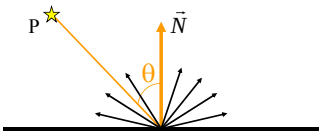


Surface imparfaite réelle

→ 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.
- 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 :

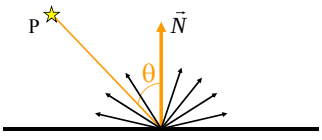
$$I_d = I_{sd} . K_d . \cos \theta$$

I_d : intensité de la lumière diffuse réfléchie

I_{sd} : intensité de la lumière diffuse

K_d : coefficient de réflexion diffuse du matériau

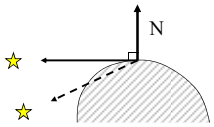
θ : angle entre la source de lumière et la normale



Notes :

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

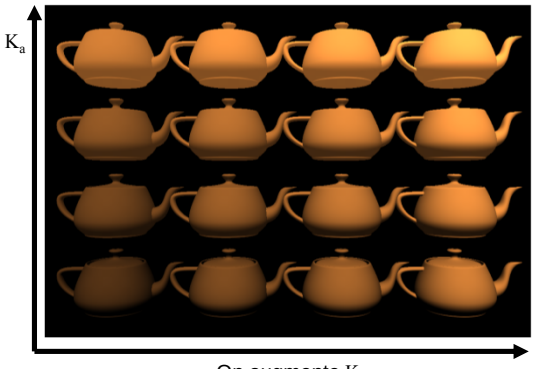
- L'intensité diffuse est maximale pour $\theta = 0$ (source de lumière à la verticale de la surface)
- Elle est nulle pour un éclairage rasant (θ=90°)
- Si $\theta > 90°$ alors le point n'est pas visible par la source de lumière.



Réflexion diffuse seule

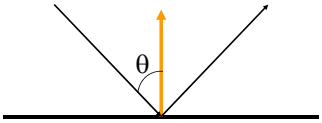
On augmente K_d (avec $K_a = 0$)

Réflexion diffuse + ambiante



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

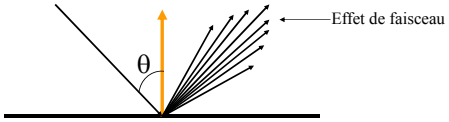


En réalité, les surfaces ne sont pas des miroirs parfaits.

→ Réflexion spéculaire : miroir imparfait

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

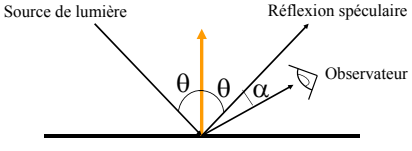
→ L'intensité de la lumière réfléchie diminue lorsqu'on s'éloigne de cette direction parfaite.



Réflexion spéculaire dans le modèle de Phong

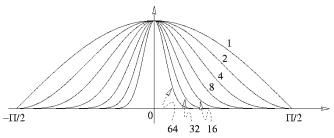
$$I_s = I_{ss} \cdot K_s \cdot (\cos\alpha)^n$$

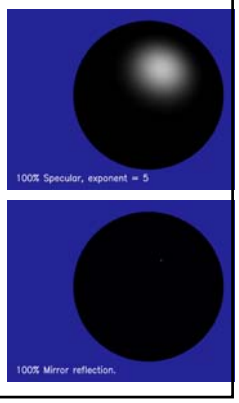
I_s : intensité de la lumière spéculaire réfléchie
I_{ss} : intensité de la lumière spéculaire de la source
K_s : coefficient de réflexion spéculaire du matériau
α : angle entre les directions de réflexion et de vue



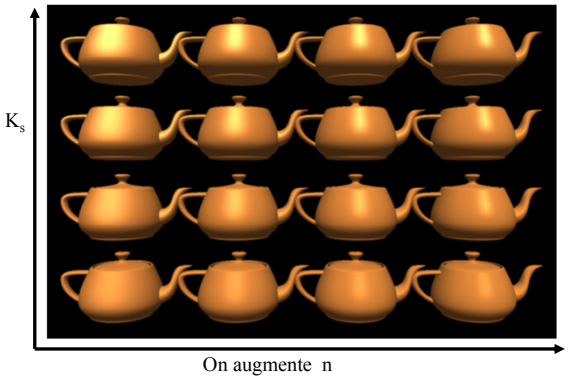
$I_s = I_{ss} . K_s . (\cos \alpha)^n$

Plus n est grand et plus on est proche d'un miroir parfait.



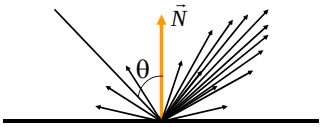


Pour obtenir un objet qui réfléchit ce qui l'entoure, il faut utiliser d'autres techniques (environment mapping, shaders).



3.2 Modèle de Phong final

- Dans la réalité, la lumière réfléchie par une surface est constituée par la somme de la réflexion diffuse et de la réflexion spéculaire.
- 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

Equation de Phong finale

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

$$I = I_a + I_d + I_s$$

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

- Si on a plusieurs sources lumineuses : somme des intensités.

→ Modèle utilisé par OpenGL, DirectX, …

Calcul 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 :

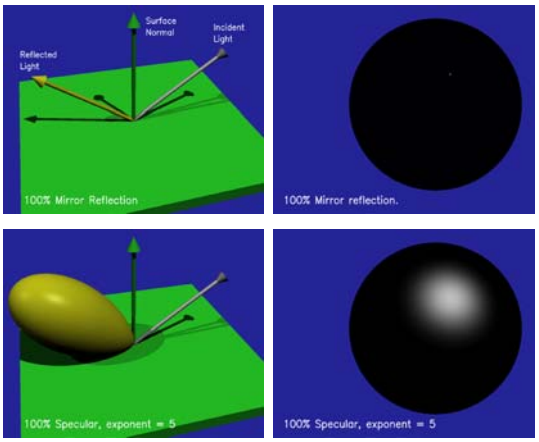
I_{saR} , I_{saV} , I_{saB} ; I_{sdR} , I_{sdV} , I_{sdB} ; I_{ssR} , I_{ssV} , I_{ssB}

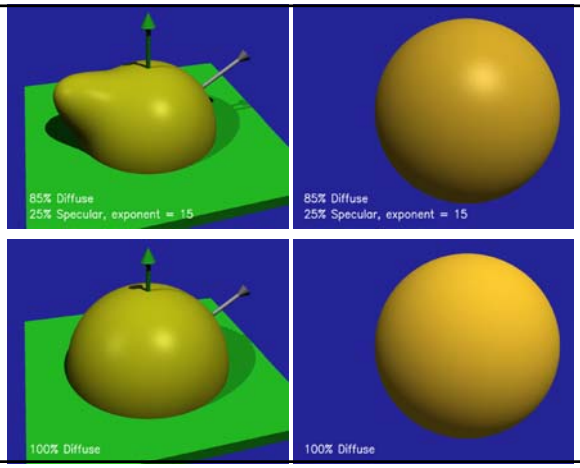
- Les caractéristiques des matériaux :

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 :

$$\begin{cases} I_R = I_{saR} \cdot K_{aR} + I_{sdR} \cdot K_{dR} \cdot \cos\theta + I_{ssR} \cdot K_{sR} \cdot (\cos\alpha)^n \\ I_V = I_{saV} \cdot K_{aV} + I_{sdV} \cdot K_{dV} \cdot \cos\theta + I_{ssV} \cdot K_{sV} \cdot (\cos\alpha)^n \\ I_B = I_{saB} \cdot K_{aB} + I_{sdB} \cdot K_{dB} \cdot \cos\theta + I_{ssB} \cdot K_{sB} \cdot (\cos\alpha)^n \end{cases}$$





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, …)

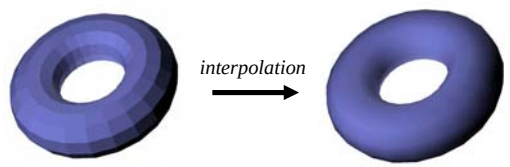
❑ Il existe des modèles d'illumination plus réalistes mais plus complexes (Cook-Torrance, etc.)

❑ Pipeline programmable : on peut programmer son propre modèle d'illumination.

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

Problème si on calcule la même illumination pour tout un polygone de l'objet

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



→ Solution : calculer l'illumination pour chaque sommet des polygones, puis interpoler l'illumination d'un sommet à un autre.

4.1 Interpolation de l'illumination

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

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

4.1.1 Interpolation de Gouraud

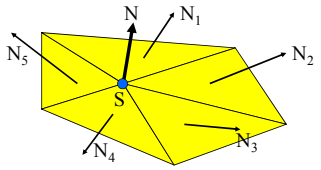
Pour chaque polygone à afficher :

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

→ Pour calculer la couleur en un sommet, on a besoin d'une normale en ce point :

- Si la surface de départ est analytiquement connue (ex: sphère, cylindre, cône, …), on peut calculer la normale en un point de cette surface.
- Si la surface de départ est un maillage polygonal ?

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

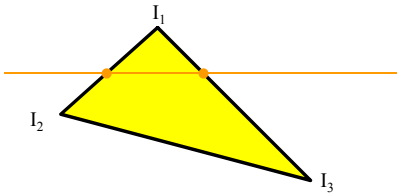


Normale au sommet S :

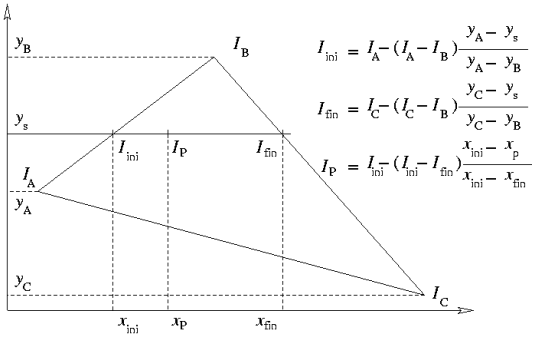
$$N = (N_1 + N_2 + N_3 + N_4 + N_5) \ / \ 5$$

Quand on a une normale pour chaque sommet d'un polygone :

- Calculer une couleur pour chaque sommet (avec un modèle d'illumination)
- Sur une arête, interpoler les couleurs entre 2 sommets
- Sur une ligne de remplissage («*scanline*») du polygone, interpoler les couleurs entre 2 arêtes.



Calcul d'interpolation par la méthode de Gouraud :

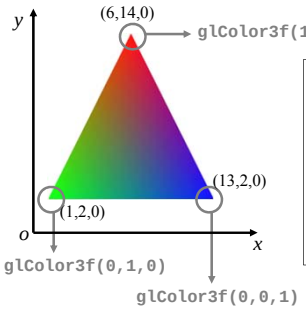


$$I_{ini} = I_A - (I_A - I_B)\frac{y_A - y_s}{y_A - y_B}$$

$$I_{fin} = I_C - (I_C - I_B)\frac{y_C - y_s}{y_C - y_B}$$

$$I_P = I_{ini} - (I_{ini} - I_{fin})\frac{x_{ini} - x_p}{x_{ini} - x_{fin}}$$

OpenGL interpole les couleurs données pour les trois sommets d'un triangle pour calculer la couleur de tous les autres points du triangle.



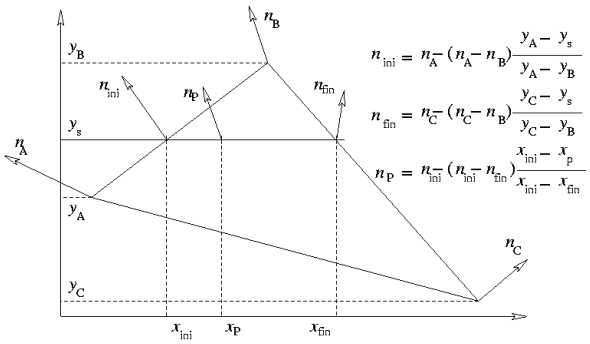

```
glBegin(GL_TRIANGLES);
   glColor3f (1, 0, 0);
   glVertex3f (6, 14, 0);
   glColor3f (0, 1, 0);
   glVertex3f (1, 2, 0);
   glColor3f (0, 0, 1);
   glVertex3f (13, 2, 0);
glEnd();
```

4.1.2 Interpolation de Phong

Principe :

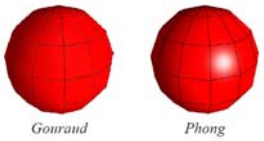
- Au lieu d'interpoler les couleurs, on interpole sur les arêtes les **normales** entre 2 sommets.
- Sur une ligne de remplissage («*scanline*») du polygone, on interpole les normales entre 2 arêtes.
- On recalcule l'illumination pour chaque pixel le long de chaque scanline.

Calcul d'interpolation par la méthode de Phong :



- 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.

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



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>)

Il existe plusieurs types de sources lumineuses :

1. Ponctuelle

Définie par une **position** : 4 coordonnées homogènes (*x*, *y*, *z*, 1)

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

Glfloat position[] = { 1.0, 1.0, 1.0, 1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, position);

2) Source directionnelle

Définie par une **direction** : 4 coordonnées homogènes (*x*, *y*, *z*, 0).

La lumière vient d'une **direction** spécifique.

Coordonnées homogènes d'un **vecteur** :

Glfloat direction[] = { 1.0, 1.0, 1.0, 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
GLfloat Light0Dif[4] = {1.0f, 0.2f, 0.2f, 1.0f};
GLfloat Light0Spec[4]= {1.0f, 0.2f, 0.2f, 1.0f};
GLfloat Light0Amb[4] = {0.5f, 0.5f, 0.5f, 1.0f};

// Valeur de position (source ponctuelle)
GLfloat Light0Pos[4] = {0.0f, 0.0f, 20.0f, 1.0f};
```

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

// 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 4 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 :

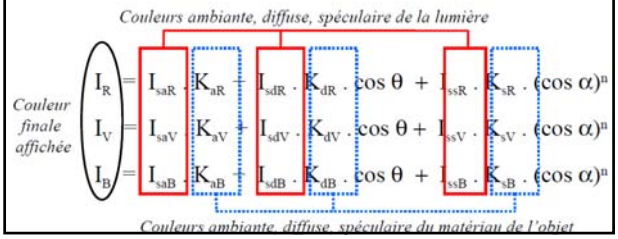
```
GLfloat MatSpec[4] = {1.0f, 1.0f, 1.0f, 1.0f};
glMaterialfv(GL_FRONT, GL_SPECULAR, MatSpec);
```

Même chose pour les autres coefficients, désignés par les constantes **GL_DIFFUSE**, **GL_AMBIENT** et **GL_EMISSION**.

```
GLfloat  MatShininess[] = { 5.0F };
glMaterialfv(GL_FRONT, GL_SHININESS, MatShininess);
```

BILAN

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



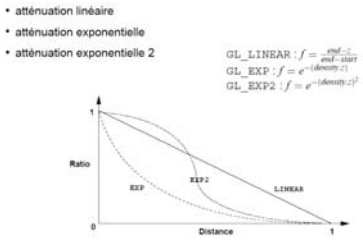
4.3 Brouillard

OpenGL permet d'appliquer un effet de brouillard selon deux modes :

- atténuation exponentielle (par défaut)
- atténuation linéaire

Scène sans brouillard

Scène avec brouillard



4.3.1 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);
```

4.3.2 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);
```

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.

- Le brouillard peut aussi être utilisé pour simuler l'atténuation de la lumière sous l'eau.

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)

- Limitation de la distance de vue → réduction du nombre d'objets à afficher, donc **accélération de l'affichage**.