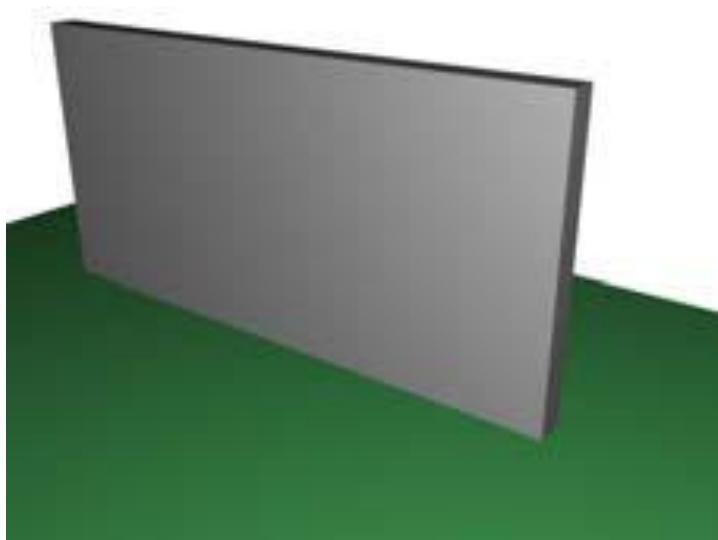


Modélisation et Programmation 3D

Modélisation avancée



Cours du 20/04/2015

roseline.beniere@c4w.com

Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Plan

- **Introduction**
- **Lumières**
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- **Textures**
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Introduction

- Rendu réaliste des objets 3D :



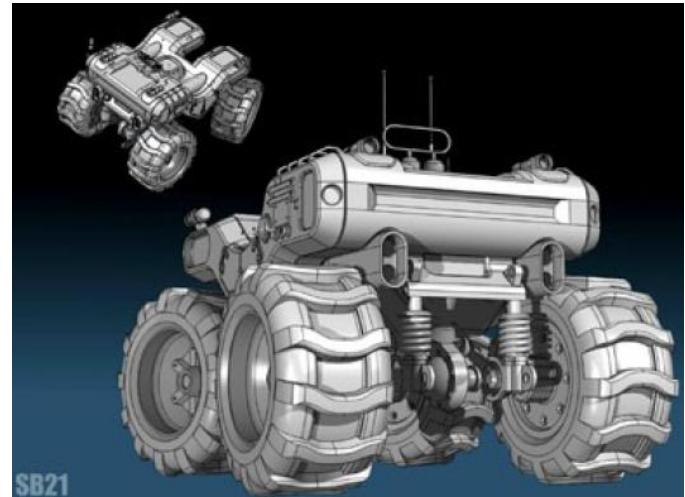
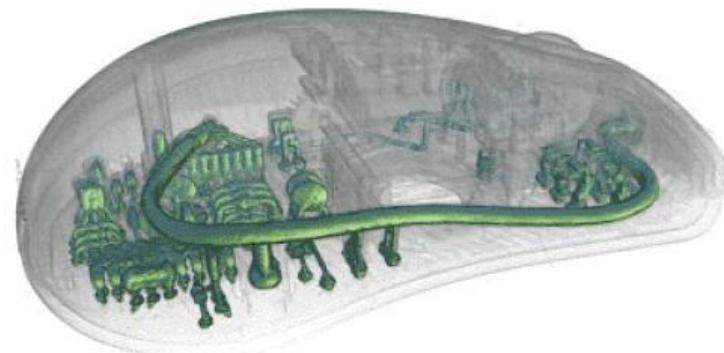
→ Eclairage réaliste.
→ Texture sur les objets

Introduction

- Pas toujours un rendu réaliste :

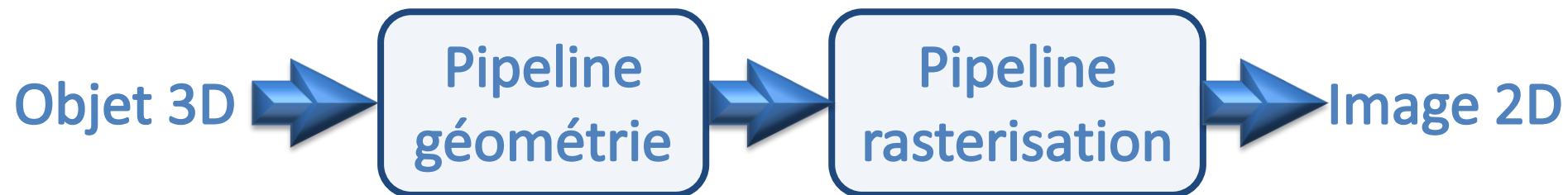


- Effet artistique
- Illustration technique
- Rendu “cartoon”

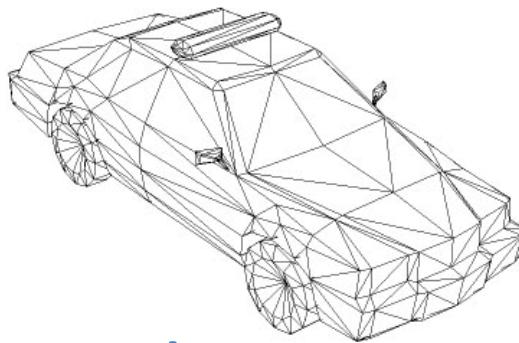


Introduction

- Pipeline graphique :



- Exemple :



Objet 3D



Prise en compte
de la lumière



Application
de la texture

Introduction

- Deux utilisations des images de synthèse :
 - 3D temps réel
 - simulateur, jeux, visualisation spécifique ...
 - produit par des librairies graphiques
 - nécessité de calculer un nombre important d'image par minute (FPS) => sacrifice du réalisme
 - 3D pré-calculée
 - images fixes, film d'animation, effets spéciaux ...
 - produit par des logiciels de synthèse d'image
 - plusieurs minutes (ou heures) peuvent être nécessaires pour calculer ces images

Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

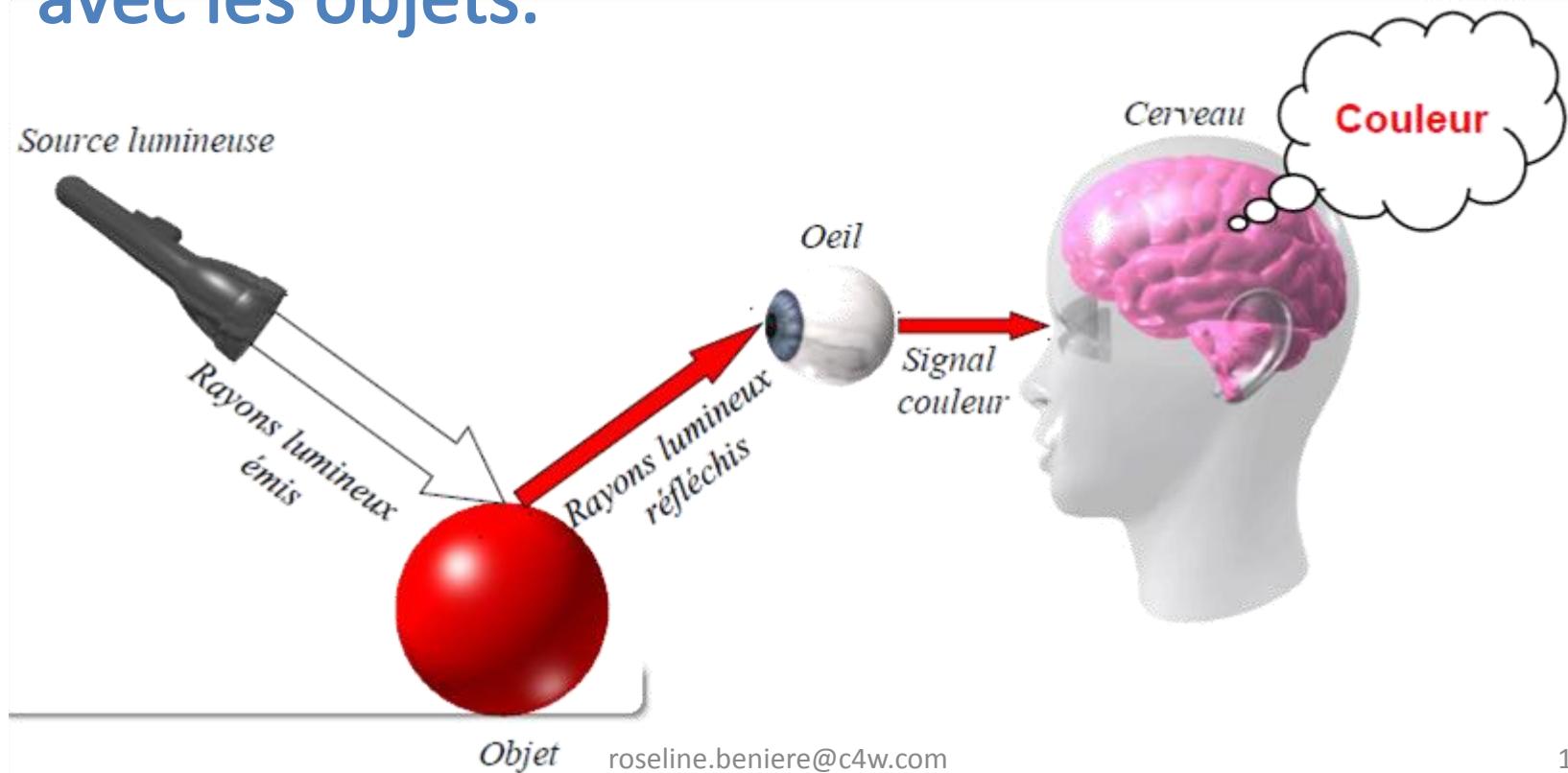
Lumières

- Recréée des phénomènes de la réalité :
 - réflexion de la lumière,
 - réfraction,
 - ombres,
 - effets de matière
 - ...



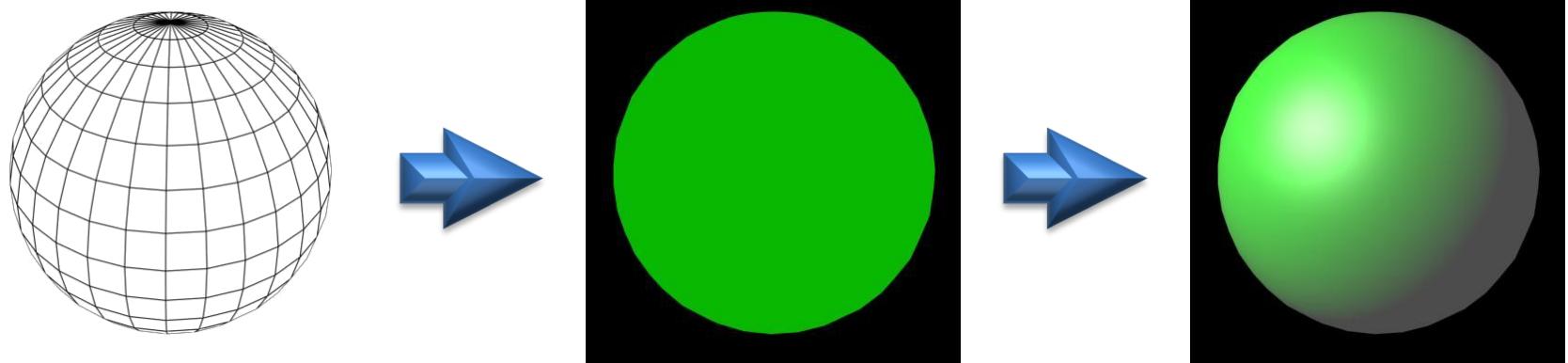
Lumières

- Réalisme dû à la perception par le système visuel humain de l'interaction entre la lumière avec les objets.



Lumières

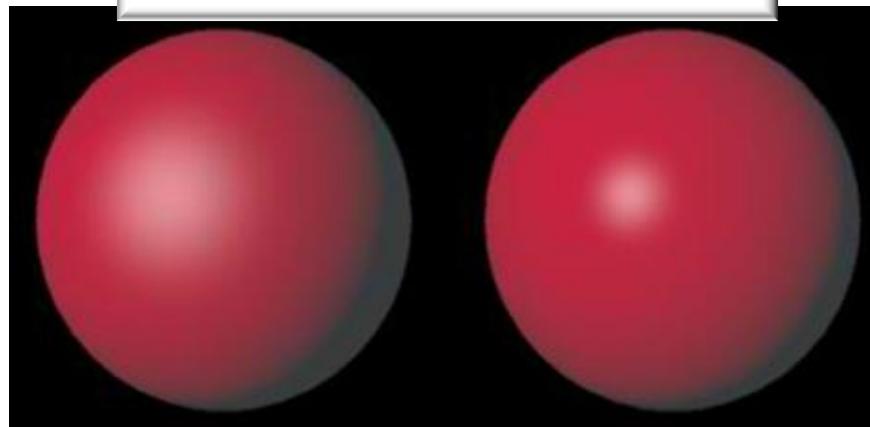
- Il n'y a pas de vraie couleur ou de vrai rendu 3D sans lumière.
- La manière dont l'objet “réfléchit” la lumière, fait que l’œil et le cerveau “reconstruisent la 3D”.



Lumières

- Des objets de forme identique mais de “matériaux différents” n’interagiront pas de la même façon avec la lumière.
- Chaque matériau transforme les propriétés de la lumière.

Matériaux différents

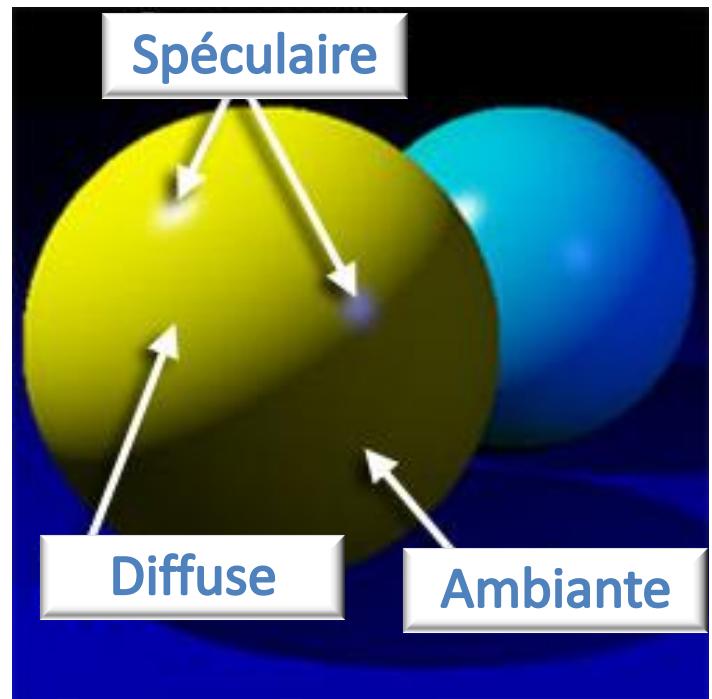


Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

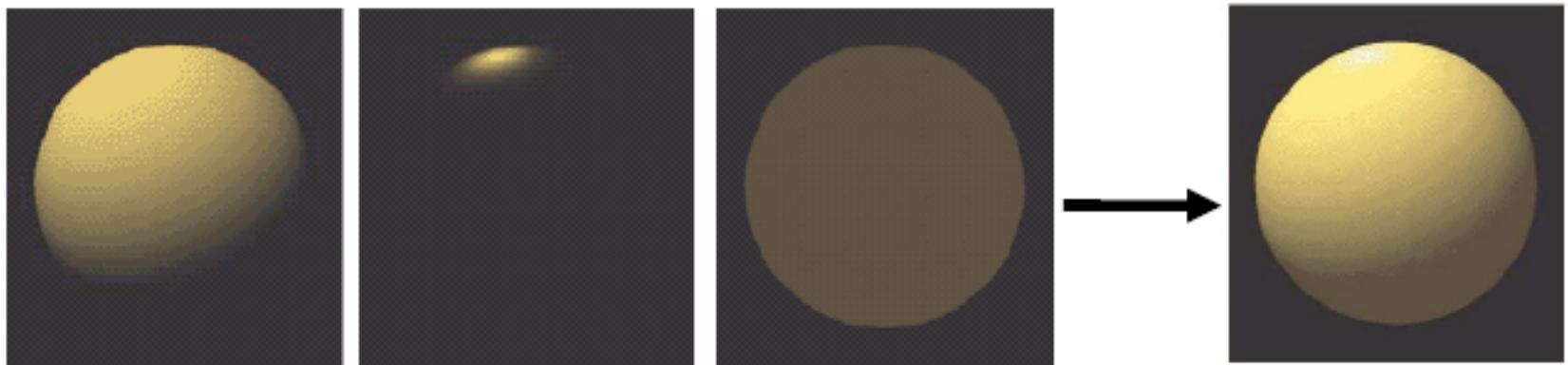
Illumination locale

- En OpenGL, utilisation du modèle d'illumination de Phong.
- Théorie : les objets sont vus parce qu'ils réfléchissent la lumière
 - réflexion ambiante
 - réflexion diffuse
 - réflexion spéculaire
 - permet de calculer la couleur en tout point



Illumination locale

- Pour obtenir une couleur, on étudie l'intensité de la lumière, noté I
 - $I = \text{niveau de gris}$
 - on calcule une valeur d'intensité pour R, V, B



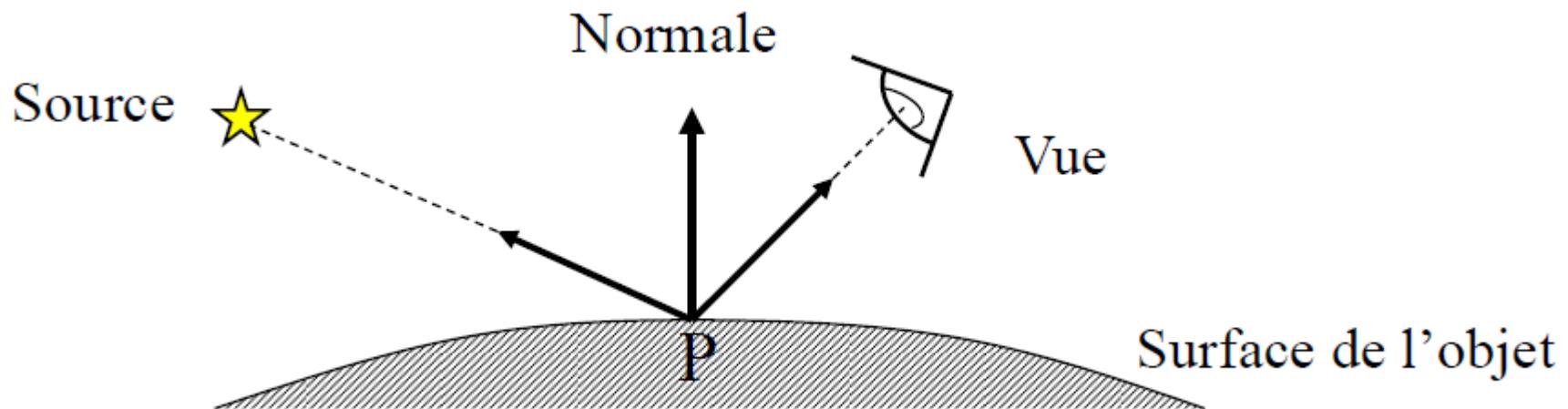
diffus + spéculaire + ambiant

Image finale

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

Illumination locale

- “Ingrédients” géométriques pour le calcul de l’illumination
 - le calcul en chaque point nécessite :
 - le vecteur normal à la surface
 - le vecteur de direction de visualisation
 - le vecteur de direction de la source lumineuse



Illumination locale

- **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

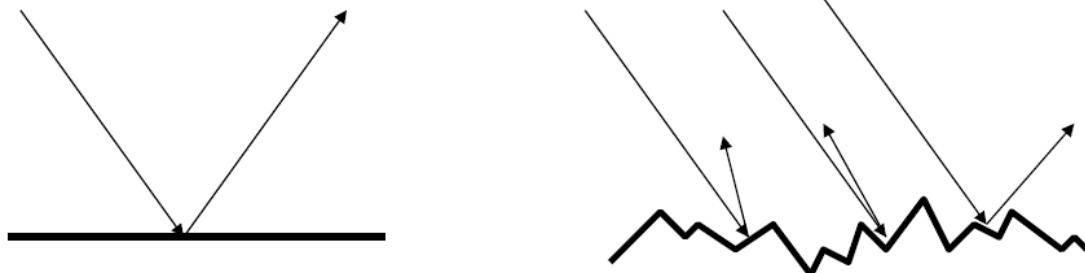
Illumination locale

- Réflexion ambiante :
 - affichage d'un objet dont on ne prend en compte que la réflexion ambiante pour le calcul de la couleur



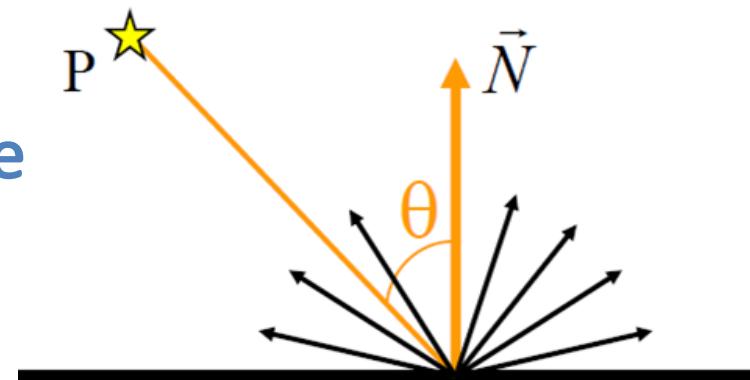
On augmente K_a

Illumination locale

- Réflexion **diffuse** et **spéculaire** :
 - la lumière n'est pas réfléchie dans une direction unique mais dans un ensemble de directions dépendant des propriétés microscopiques de la surface
 - 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.
- 
- The diagram consists of two parts. On the left, labeled "Miroir parfait théorique", a horizontal line represents a mirror reflecting a single, sharp ray back towards the source. On the right, labeled "Surface imparfaite réelle", a horizontal line represents a surface covered in small peaks and valleys. Multiple rays are shown hitting the surface at different points, reflecting in various directions, which illustrates the diffuse nature of the reflection from a rough surface.

Illumination locale

- Réflexion diffuse :
 - 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)

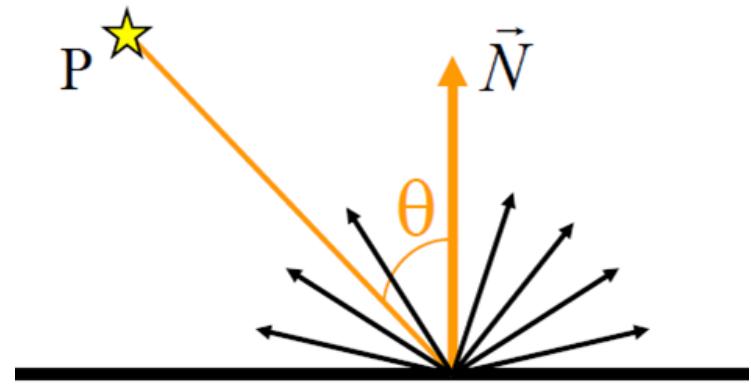


Illumination locale

- Réflexion diffuse :

- loi de Lambert

$$Id = Isd \cdot Kd \cdot \cos(\theta)$$



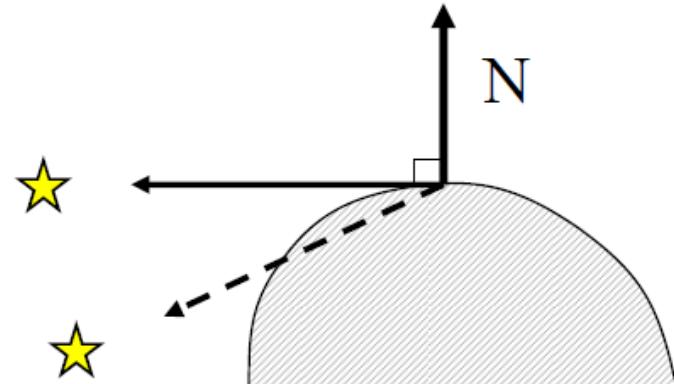
- Id : intensité de la lumière diffuse réfléchie
 - Isd : intensité de la lumière diffuse
 - Kd : coefficient de réflexion diffuse du matériau
 - θ : angle entre la source de lumière et la normale

Illumination locale

- Réflexion diffuse :

- loi de Lambert

$$Id = Isd \cdot Kd \cdot \cos(\theta)$$



- l'intensité diffuse est maximale pour $\theta = 0^\circ$ (source de lumière à la verticale de la surface)
 - elle est nulle pour un éclairage rasant $\theta = 90^\circ$
 - si $\theta > 90^\circ$ alors le point n'est pas visible par la source de lumière

Illumination locale

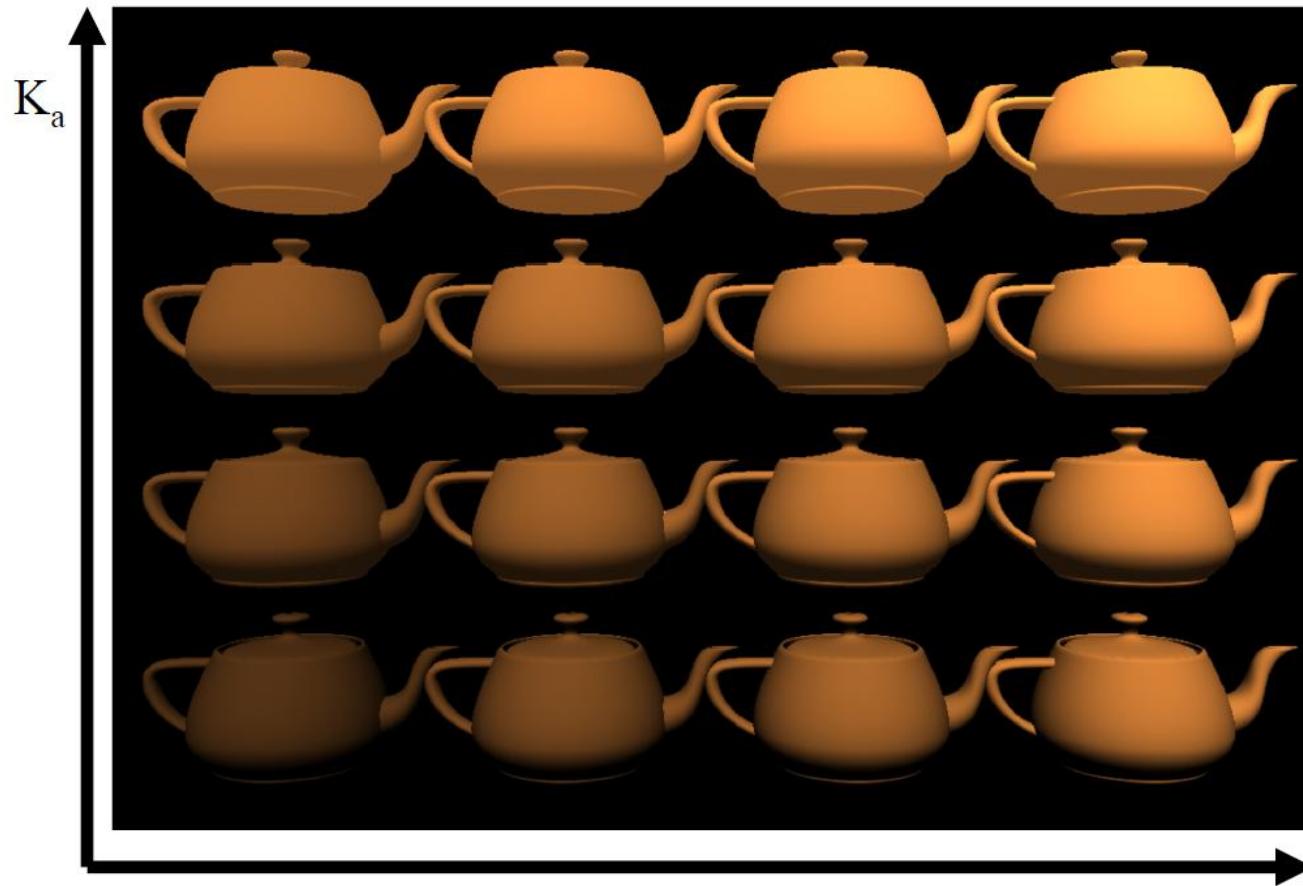
- Réflexion **diffuse** seule :



On augmente K_d (avec $K_a = 0$)

Illumination locale

- Réflexion diffuse + ambiante :



On augmente K_d

roseline.beniere@c4w.com

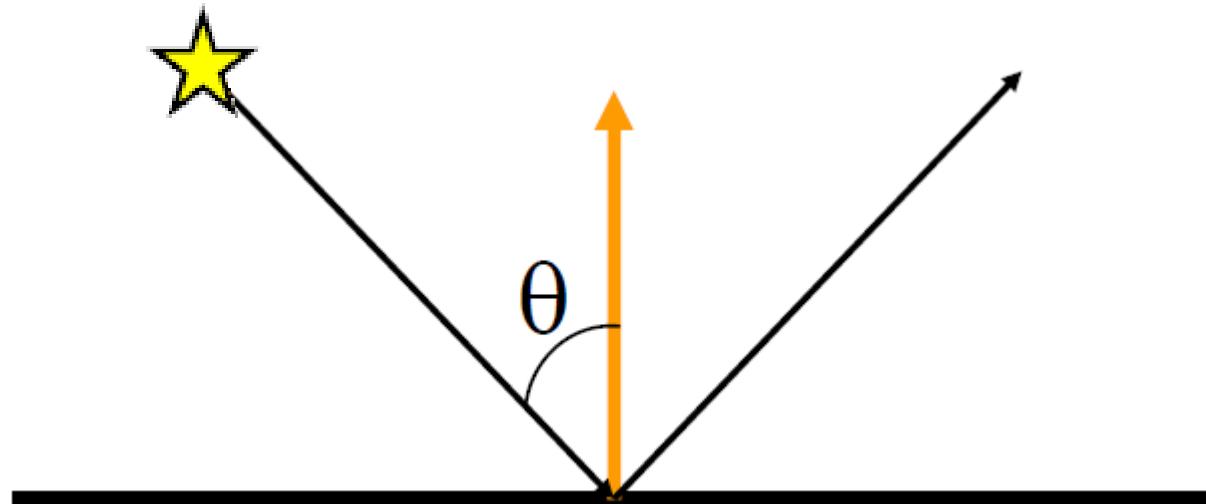
Illumination locale

- Réflexion spéculaire :

- permet d'obtenir des reflets

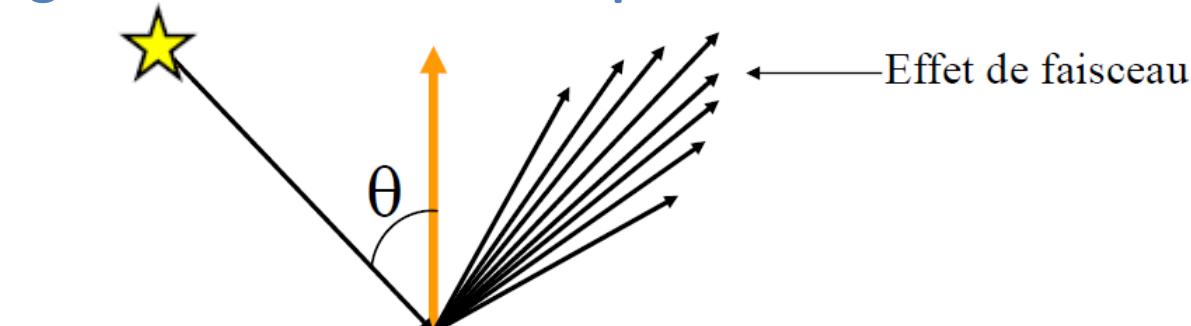
- miroir parfait => Loi de Descartes

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



Illumination locale

- Réflexion spéculaire :
 - en réalité, les surfaces ne sont jamais 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.

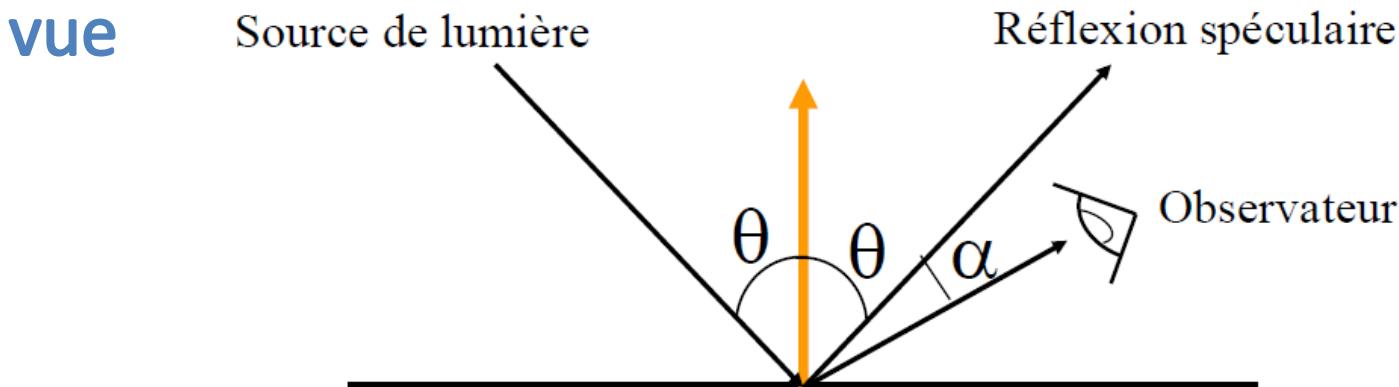


Illumination locale

- Réflexion spéculaire dans le modèle de Phong :

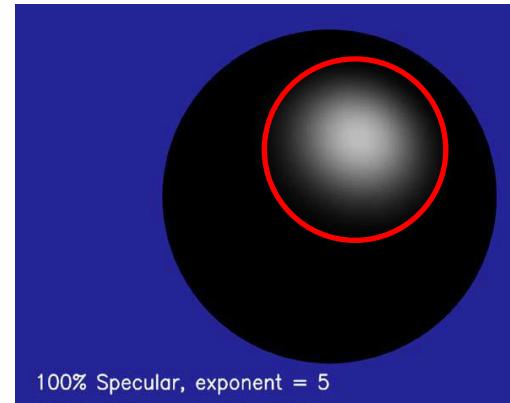
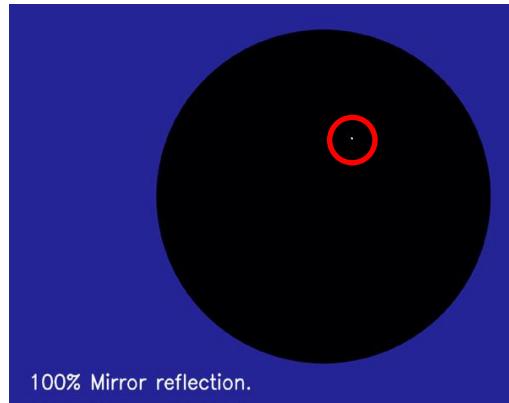
$$I_s = I_{ss} \cdot K_s \cdot \cos(\alpha)^n$$

- 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 la vue



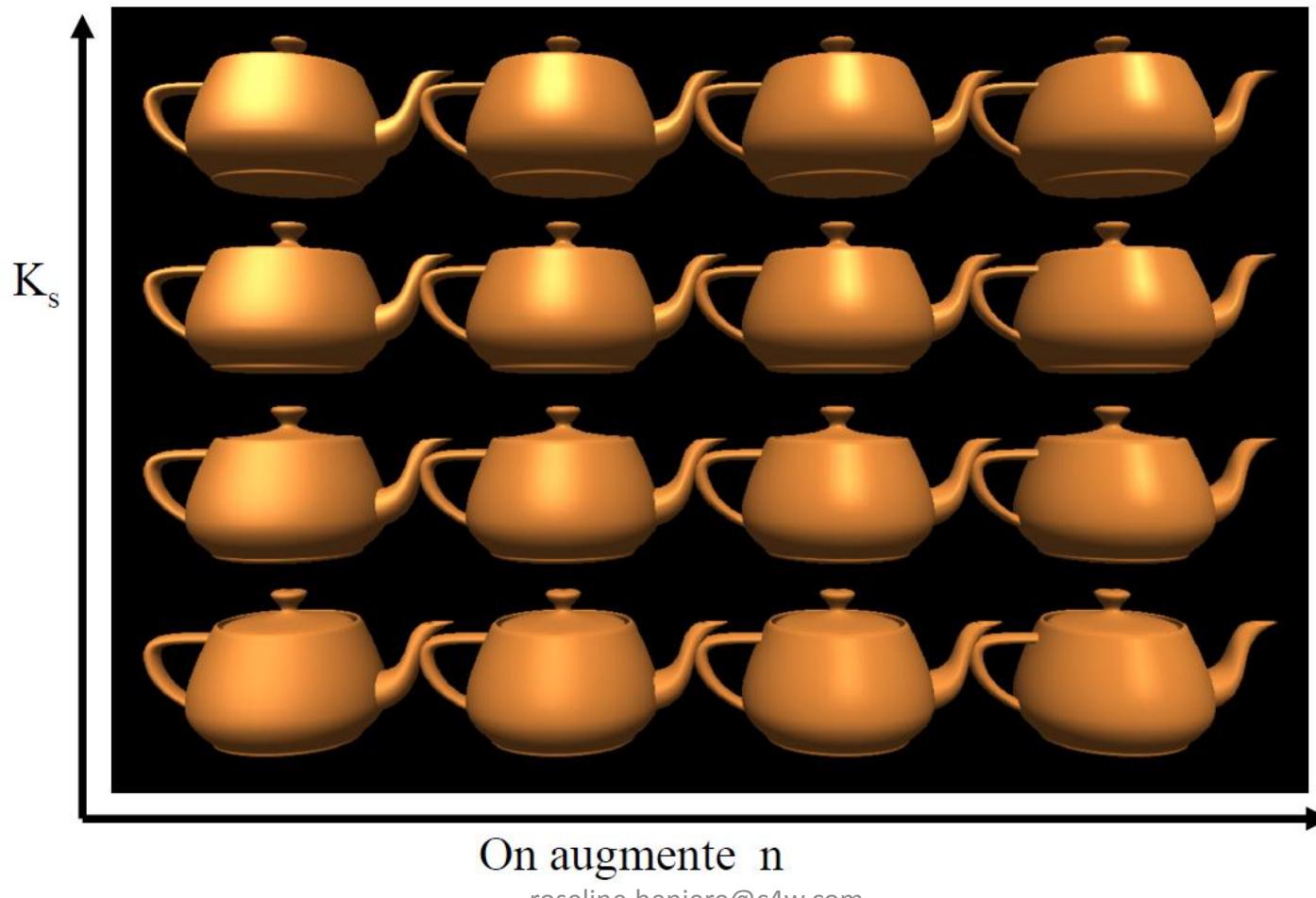
Illumination locale

- Réflexion spéculaire :
$$I_s = I_{ss} \cdot K_s \cdot \cos(\alpha)^n$$
 - 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'autre techniques (environnement mapping, shaders)



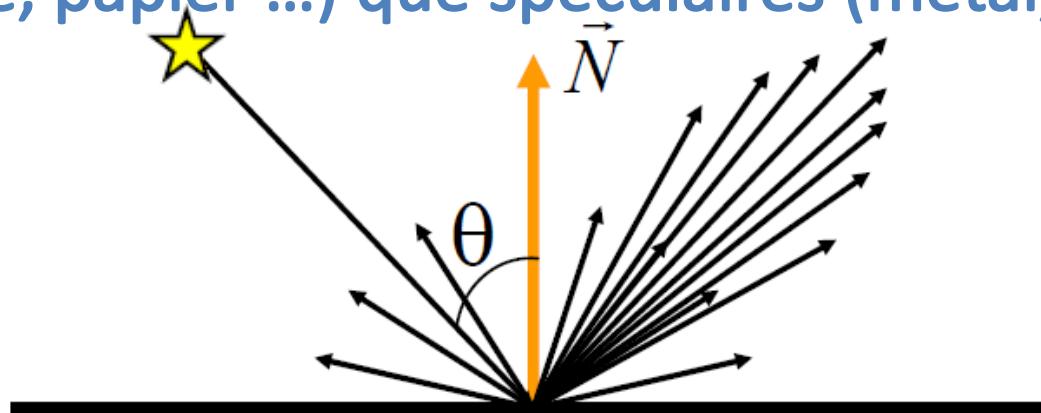
Illumination locale

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



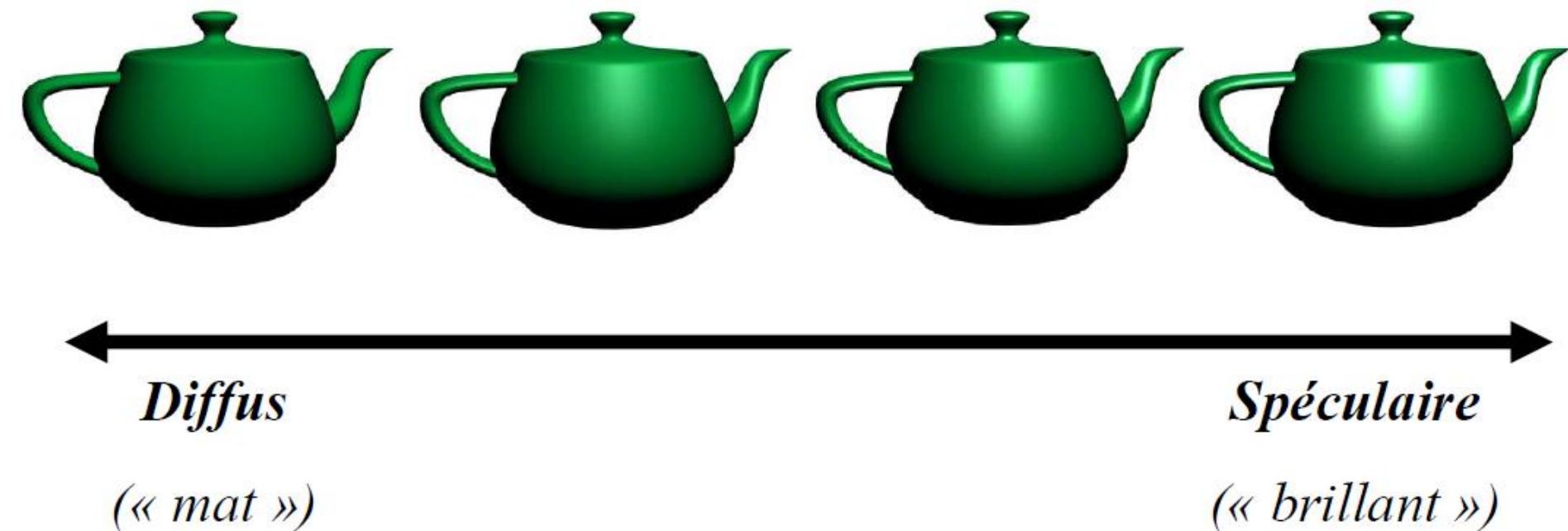
Illumination locale

- 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 ...) que spéculaires (métal, verre ...)



Illumination locale

- Réflexion finale



Illumination locale

- Equation de Phong finale
 - égale à la somme des réflexions ambiante, diffuse et spéculaire :

$$I = I_a + I_d + I_s$$

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

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

Illumination locale

- 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 composante :
 - les caractéristiques des sources de lumière
IsaR, IsaV, IsaB ; IsdR, IsdV, IsdB ; IssR, IssV, IssB
 - les caractéristiques des matériaux
KaR, KaV, KaB ; KdR, KdV, KdB ; KsR, KsV, KsB

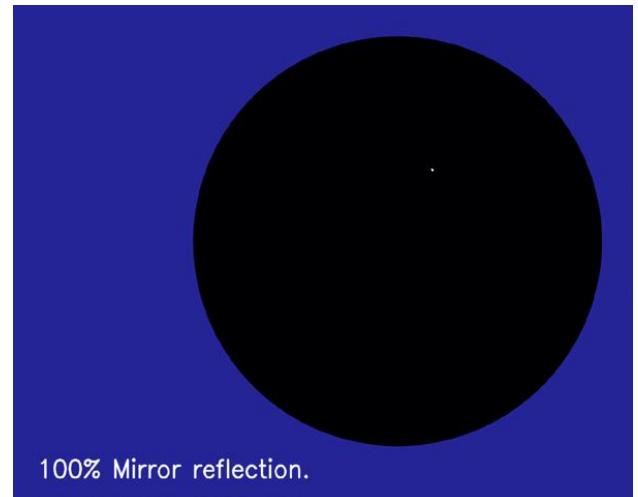
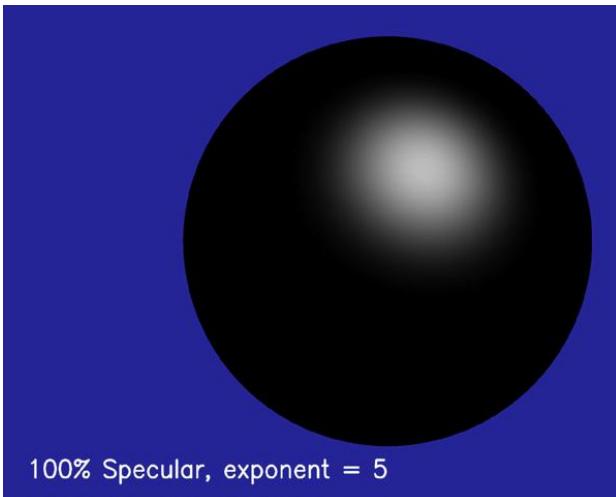
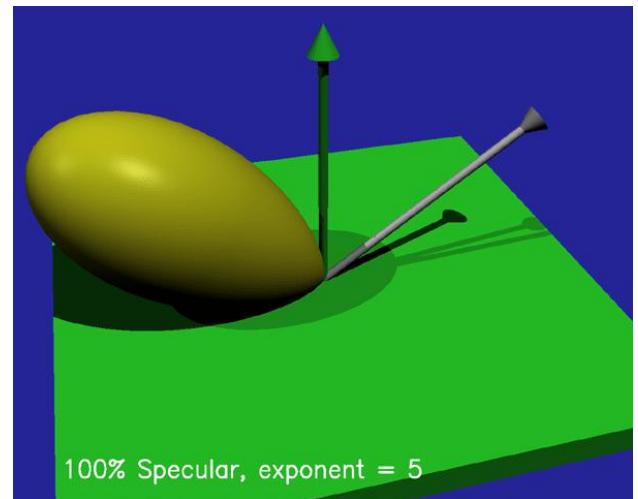
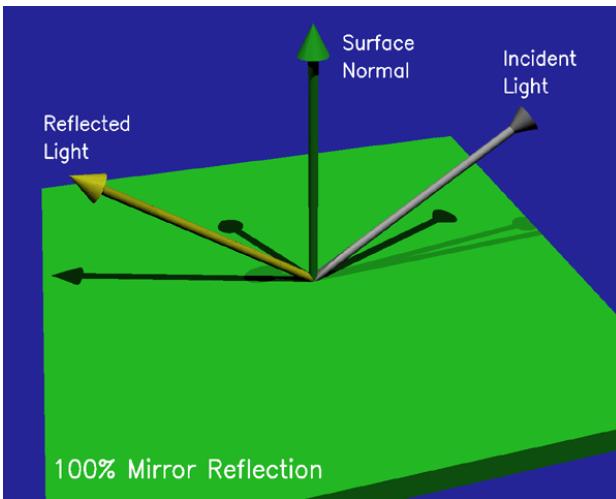
Illumination locale

- Calcul de la couleur
 - les intensités lumineuses pour chacune des 3 composantes R, V, B s'obtiennent donc ainsi :

$$\begin{aligned} IR &= IsaR \cdot KaR + IsdR \cdot KdR \cdot \cos(\theta) + IssR \cdot KsR \cdot \cos(\alpha)^n \\ IV &= IsaV \cdot KaV + IsdV \cdot KdV \cdot \cos(\theta) + IssV \cdot KsV \cdot \cos(\alpha)^n \\ IB &= IsaB \cdot KaB + IsdB \cdot KdB \cdot \cos(\theta) + IssB \cdot KsB \cdot \cos(\alpha)^n \end{aligned}$$

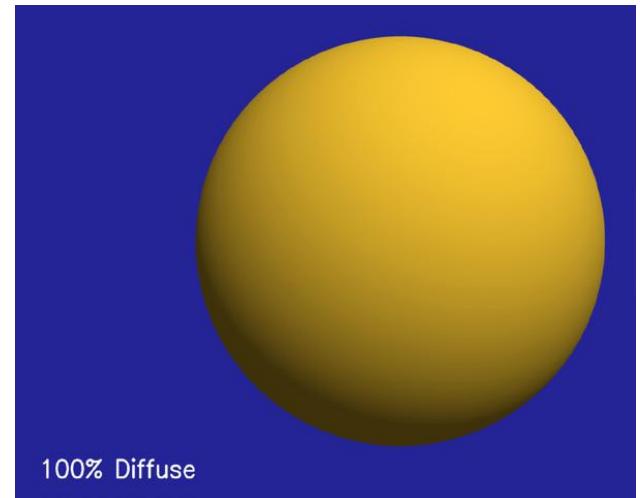
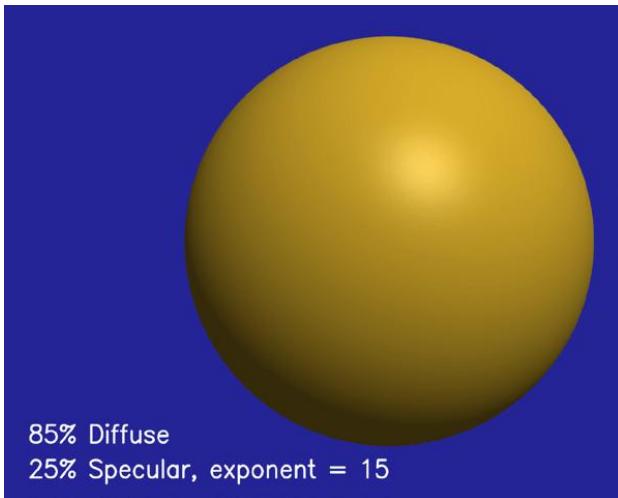
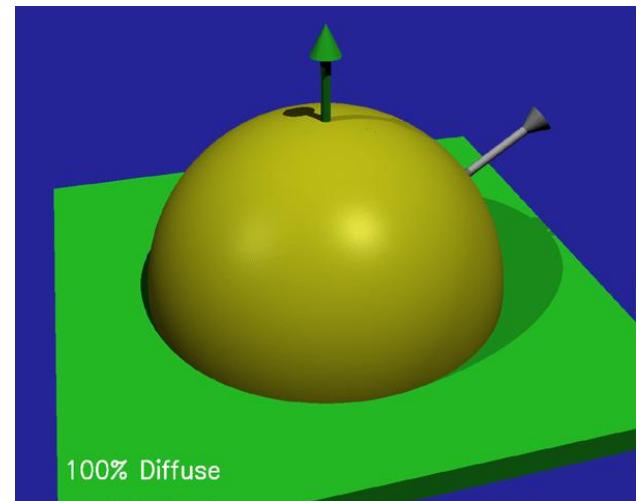
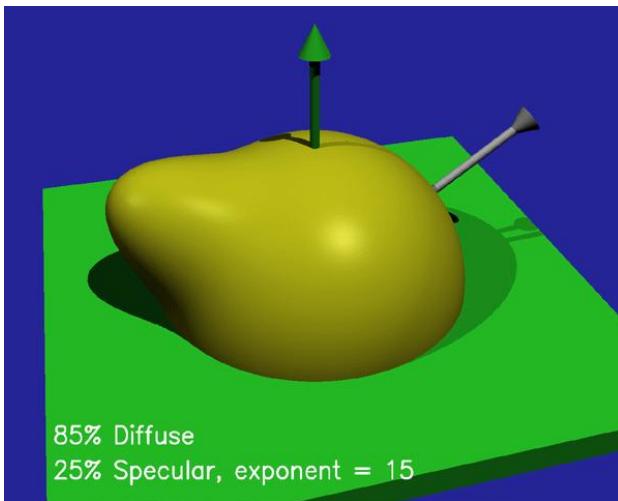
Illumination locale

- Résultats :



Illumination locale

- Résultats :



Illumination locale

- 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 ...)

Illumination locale

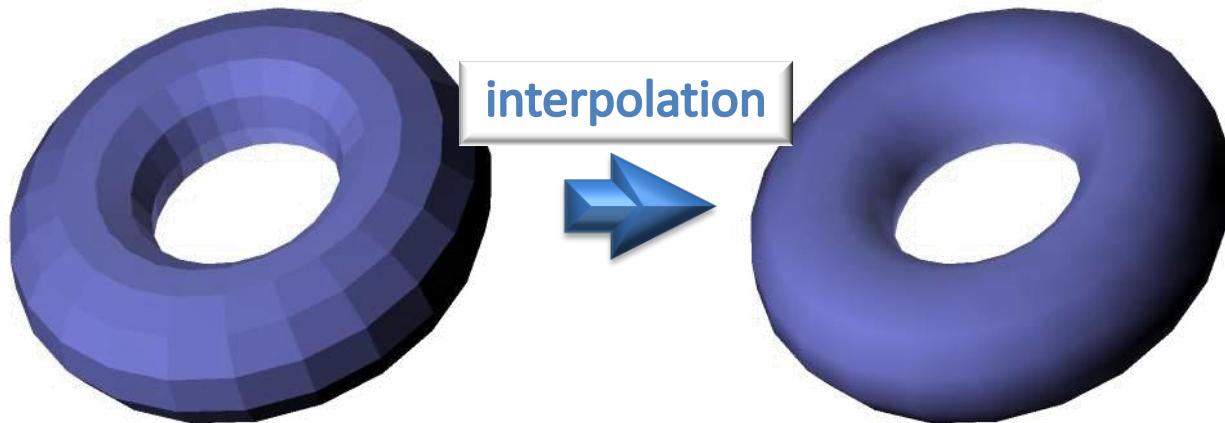
- Il existe des modèles d'illumination plus réalistes mais plus complexes (Cook-Torrance ...) et moins rapide
- Le pipeline géométrie est quelque chose de programmable : chacun peut donc programmer son propre modèle d'illumination.

Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Illumination d'un objet 3D

- Problème si on calcule la même illumination pour tout un polygone (élément du maillage) :
 - affichage ``plat'' (*flat shading*)



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

Illumination d'un objet 3D

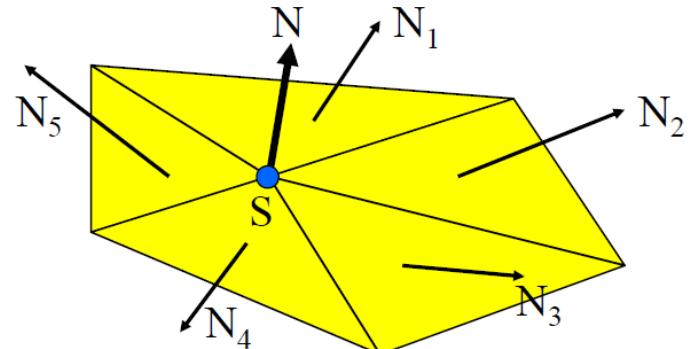
- Interpolation de l'illumination
 - But : calculer une couleur pour chaque point visible à l'écran de l'objet 3D qu'on affiche.
 - Lissage de **Gouraud** : interpolation de **couleurs**
 - Lissage de **Phong** : interpolation de **normales**

Illumination d'un objet 3D

- Interpolation de Gouraud
 - pour chaque polygone à afficher :
 - calculer pour chaque sommet du polygone une couleur au moyen d'un modèle d'illumination (Phong ...)
 - interpoler les **couleurs** des sommets pour calculer la couleur de chaque pixel du polygone.

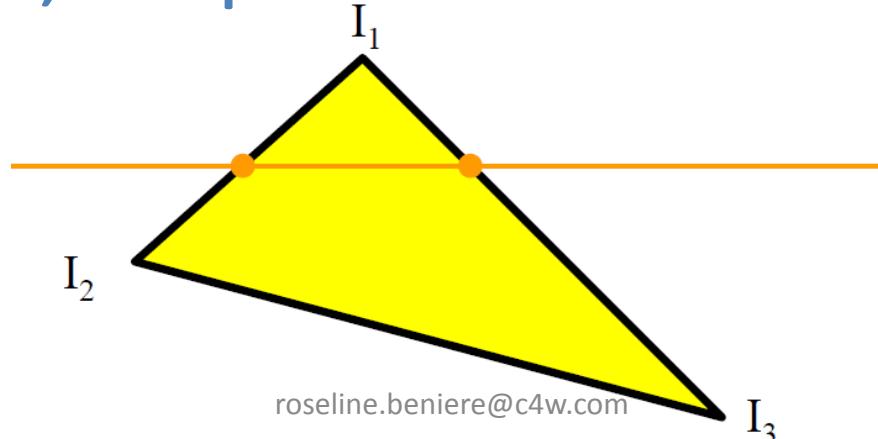
Illumination d'un objet 3D

- Interpolation de l'illumination
 - 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 : une sphère, un cylindre ...), on peut calculer la normale en un point de cette surface.
 - si la surface de départ est un maillage polygonal, comment faire ?? ➔ interpoler les normales



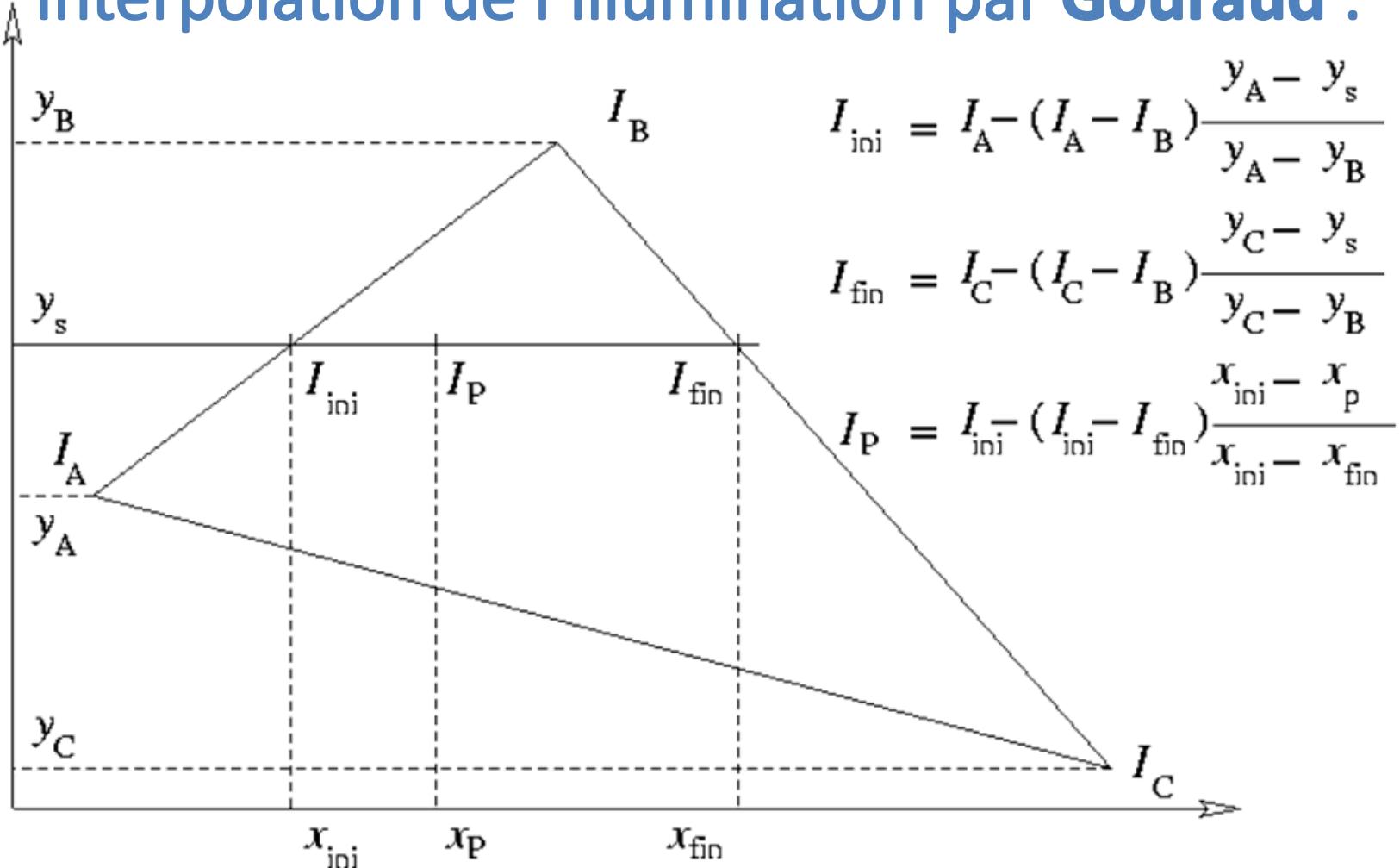
Illumination d'un objet 3D

- Interpolation de l'illumination
 - pour chaque sommet on calcule une couleur avec un modèle d'illumination
 - sur une arête, interpoler les couleurs entre les 2 sommets
 - sur une ligne de remplissage (*scanline*) du polygone, interpoler les couleurs entre 2 arêtes



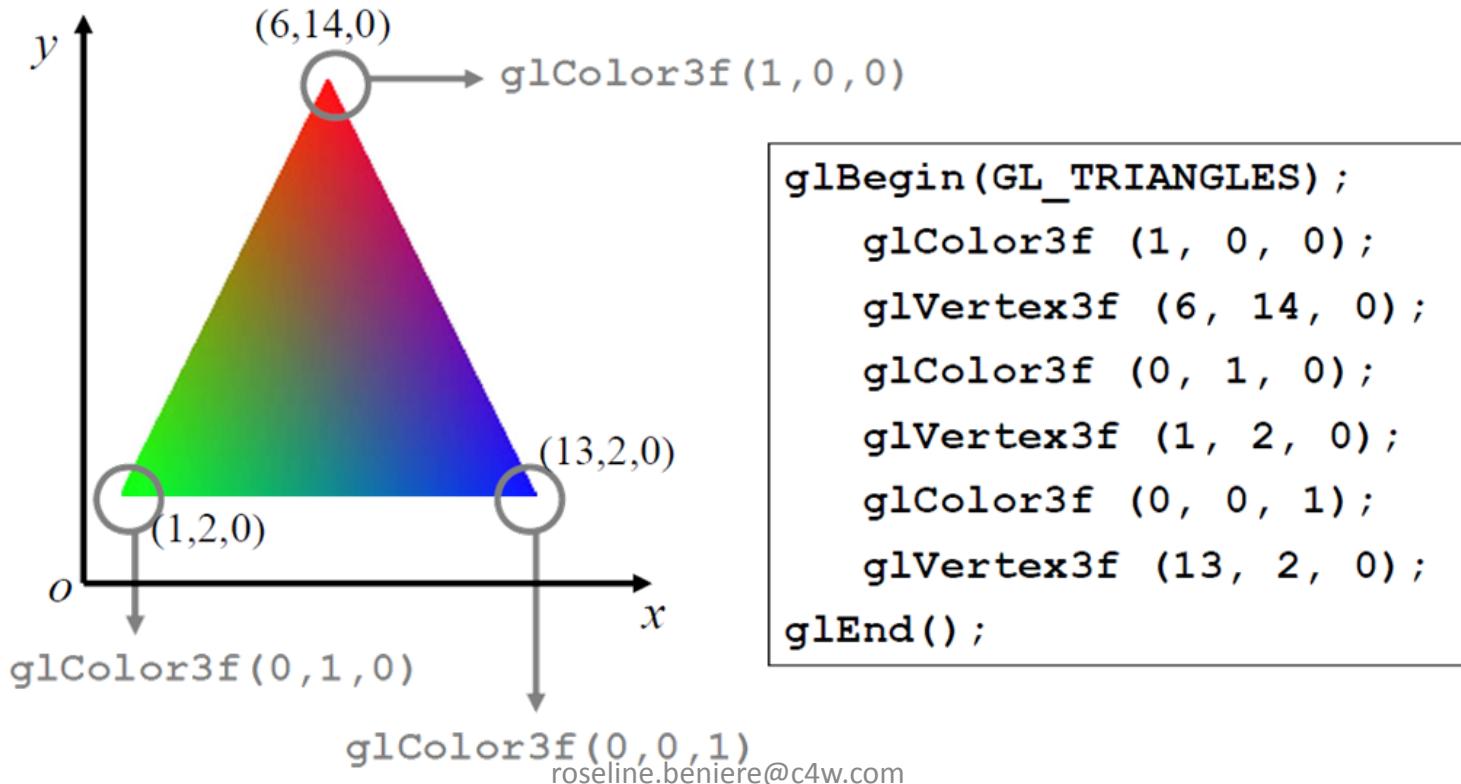
Illumination d'un objet 3D

- Interpolation de l'illumination par Gouraud :



Illumination d'un objet 3D

- OpenGL interpolate les couleurs données pour les 3 sommets d'un triangle, pour calculer la couleur de tous les autres points du triangle :

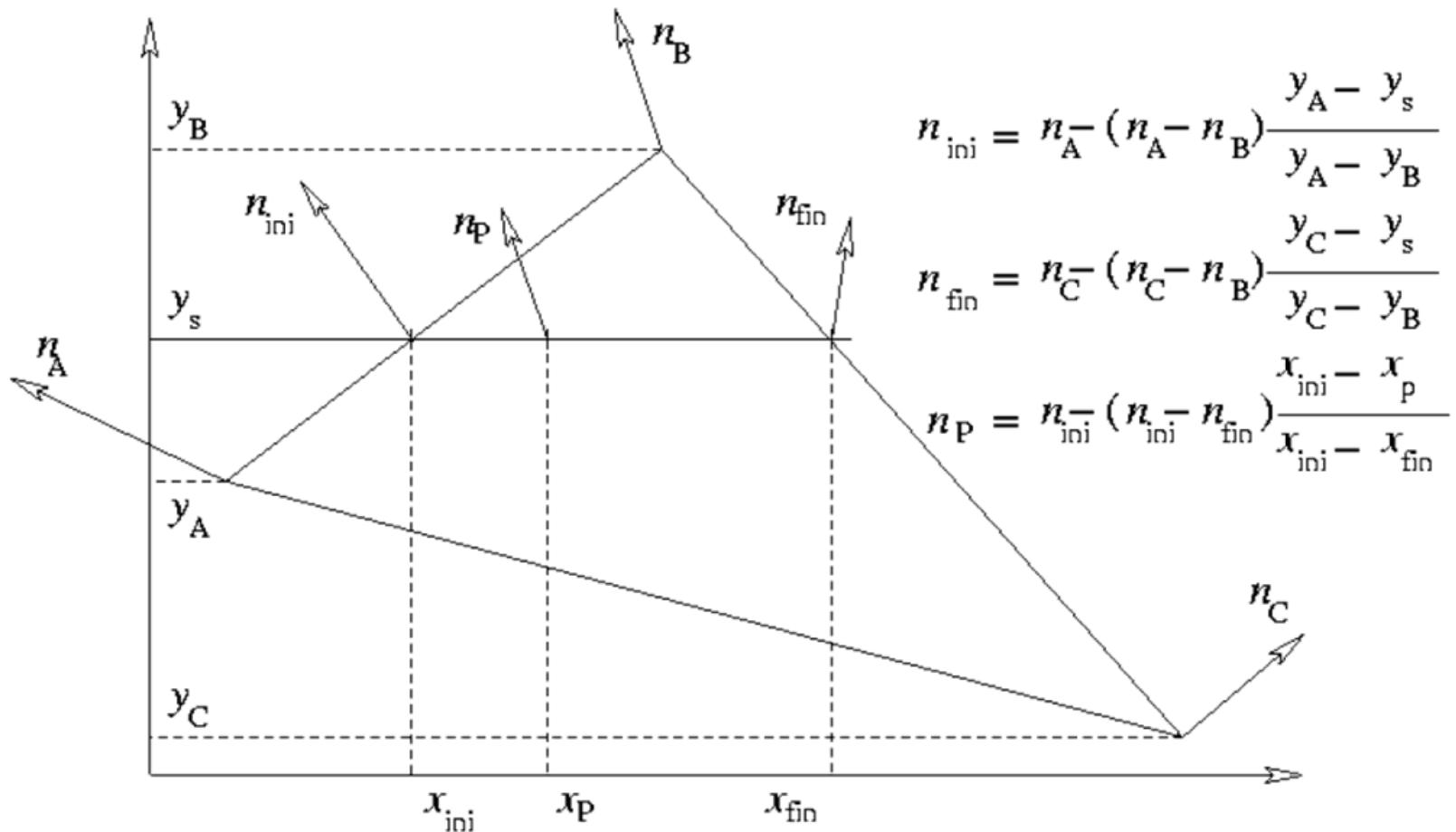


Illumination d'un objet 3D

- Interpolation de Phong
 - 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

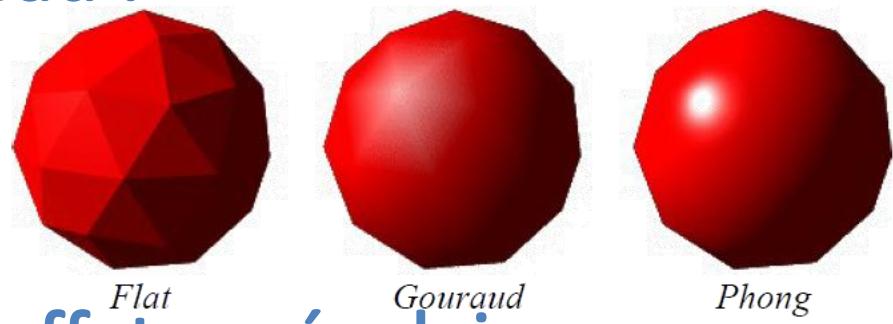
Illumination d'un objet 3D

- Interpolation de l'illumination par Phong :

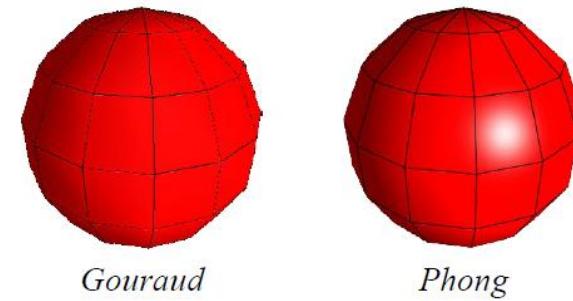


Illumination d'un objet 3D

- Lissage de Phong plus lent que celui de Gouraud (plus de calcul d'illumination) mais plus nettement plus beau :



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



Plan

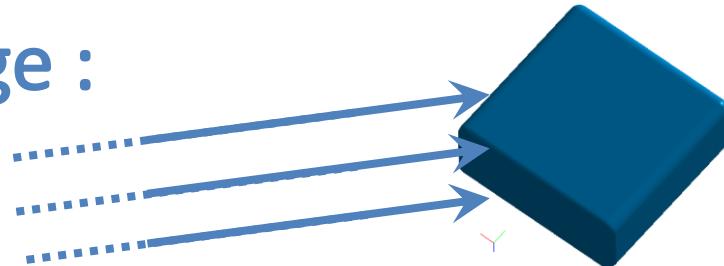
- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Lumière en OpenGL

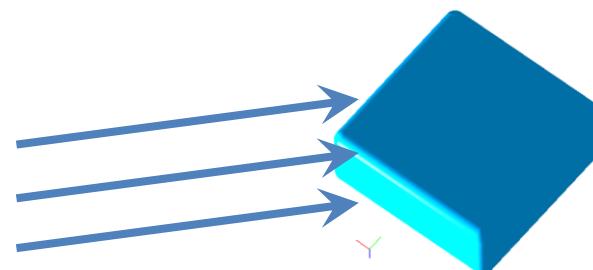
- Plusieurs type d'éclairage :

- source directionnelle

-> pas d'atténuation,

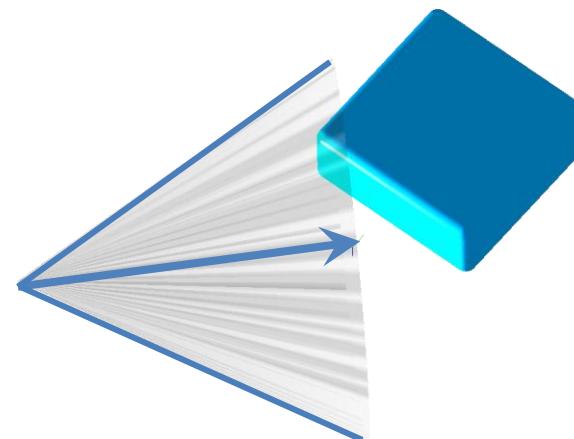


- source ponctuelle



- source projecteur

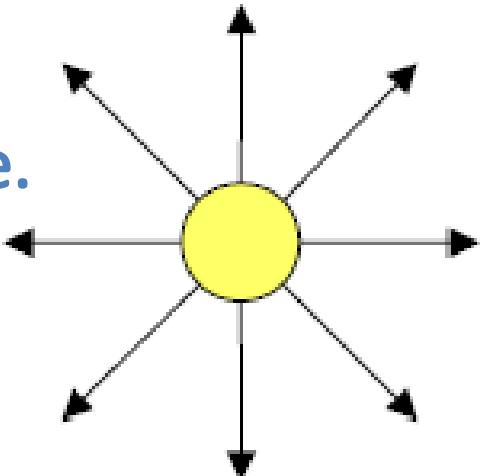
-> avec un angle



Lumière en OpenGL

- Source ponctuelle

- définie par une position : 4 coordonnées homogènes ($x, y, z, 1$)
- la lumière vient d'un point spécifique.
Coordonnées homogène d'un point :

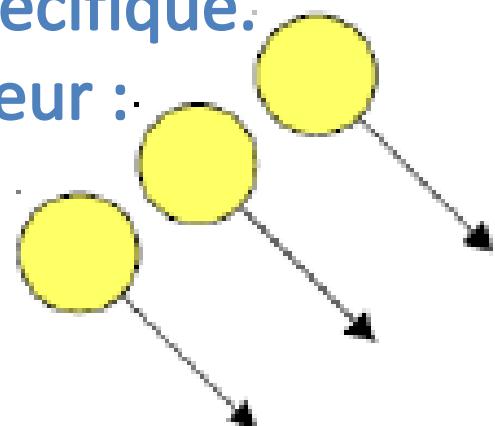


```
GLfloat position[] = { 1.0, 1.0, 1.0, 1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Lumière en OpenGL

- Source directionnelle

- définie par une direction : 4 coordonnées homogènes ($x, y, z, 0$)
- la lumière vient d'une direction spécifique.
Coordonnées homogène d'un vecteur :

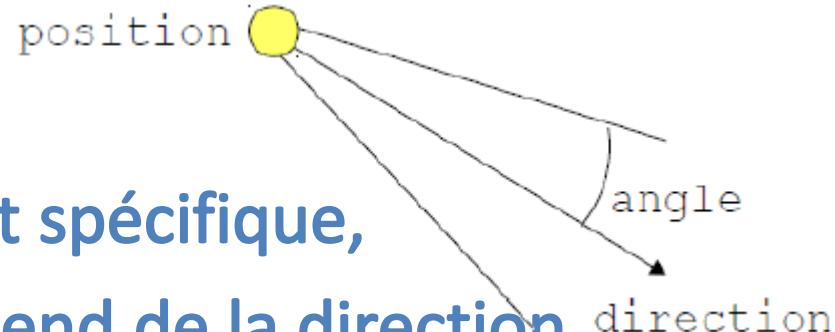


```
GLfloat direction[] = { 1.0, 1.0, 1.0, 0.0 };  
glLightfv(GL_LIGHT0, GL_POSITION, direction);
```

Lumière en OpenGL

- Source “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);
```

Lumière en OpenGL

- 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};
```

Lumière en OpenGL

- Utilisation :

```
// 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);
```

Lumière en OpenGL

- En OpenGL, il y a 8 lumières minimum
- Le nombre maximum de lumière est donné par la constante **GL_MAX_LIGHTS**
- On peut soit activé (ou désactivé) l'éclairage de manière générale:
 - `glEnable(GL_LIGHTING)` / `glDisable(GL_LIGHTING)`
- On peut soit activé (ou désactivé) une lumière particulière :
 - `glEnable(GL_LIGHT0)` / `glDisable(GL_LIGHT0)`

Lumière en OpenGL

- Matériaux en OpenGL :
 - Tout matériau est défini par 4 vecteurs de 4 composantes (rouge, vert, bleu, alpha) :
 - **coefficients de réflexion ambiant** (valeur par défaut $<0, 0, 0, 1>$)
 - **coefficients de réflexion diffus** (valeur par défaut $<1, 1, 1, 1>$)
 - **coefficients 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

Lumière en OpenGL

- 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);
```

Lumière en OpenGL

- Bilan :

Etant donné les couleurs ambiante, diffuse, spéculaire de la lumière, ainsi que les composantes ambiante, diffus et 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 :

The diagram illustrates the Phong lighting model equation. It shows the calculation of three final colors (I_R , I_V , I_B) based on the ambient, diffuse, and specular components of both the light source and the object's material.

Couleurs ambiante, diffuse, spéculaire de la lumière

Couleur finale affichée

Couleurs ambiante, diffuse, spéculaire du matériau de l'objet

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

Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

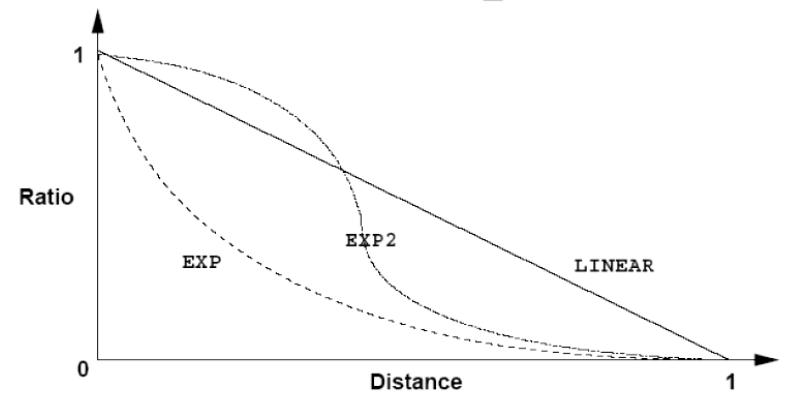
Brouillard

- OpenGL permet d'appliquer un effet de brouillard selon deux mode :
 - atténuation exponentielle (par défaut)
 - atténuation linéaire



- atténuation linéaire
- atténuation exponentielle
- atténuation exponentielle 2

$$\begin{aligned} \text{GL_LINEAR} &: f = \frac{\text{end}-z}{\text{end}-\text{start}} \\ \text{GL_EXP} &: f = e^{-(\text{density}.z)} \\ \text{GL_EXP2} &: f = e^{-(\text{density}.z)^2} \end{aligned}$$



Brouillard

- 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);
```

Brouillard

- 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);
```

Brouillard

- Intérêt du brouillard :
 - permet de reproduire un phénomène naturel
 - permet de donner une ambiance (angoissante, mystérieuse ...)



Brouillard

- Intérêt du brouillard :
 - 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 ...



Brouillard

- Intérêt du brouillard :
 - il peut aussi être utilisé pour simuler l'atténuation de la lumière sous l'eau.

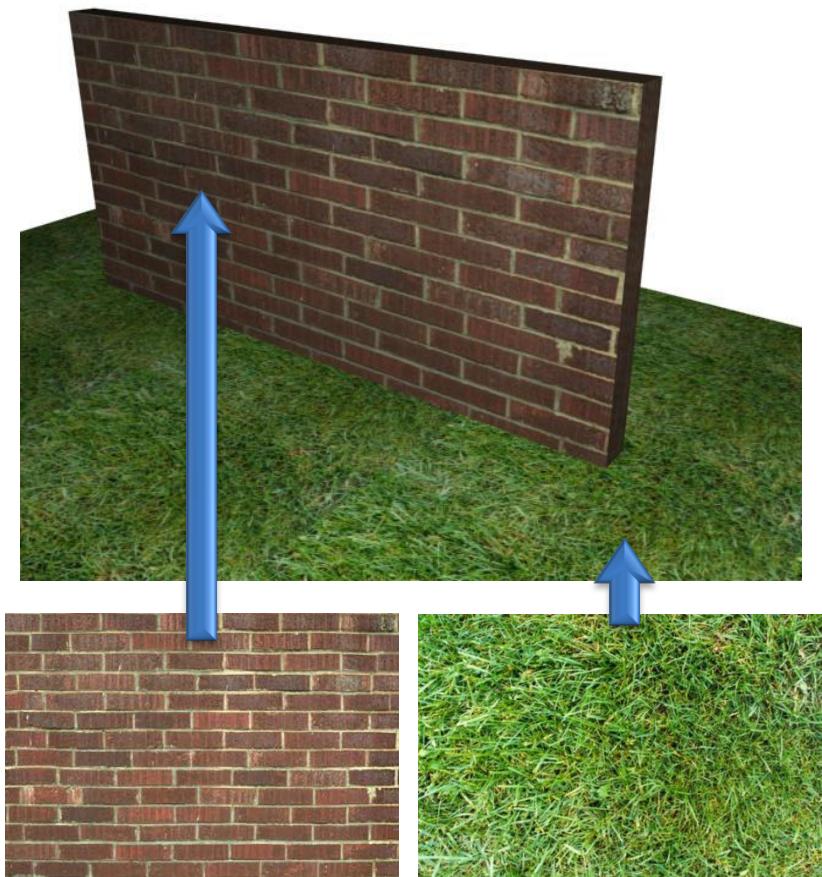
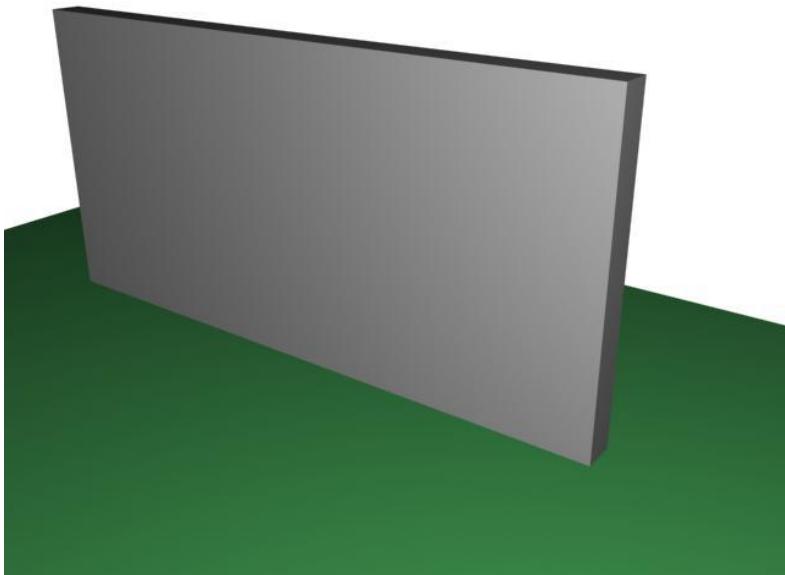


Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

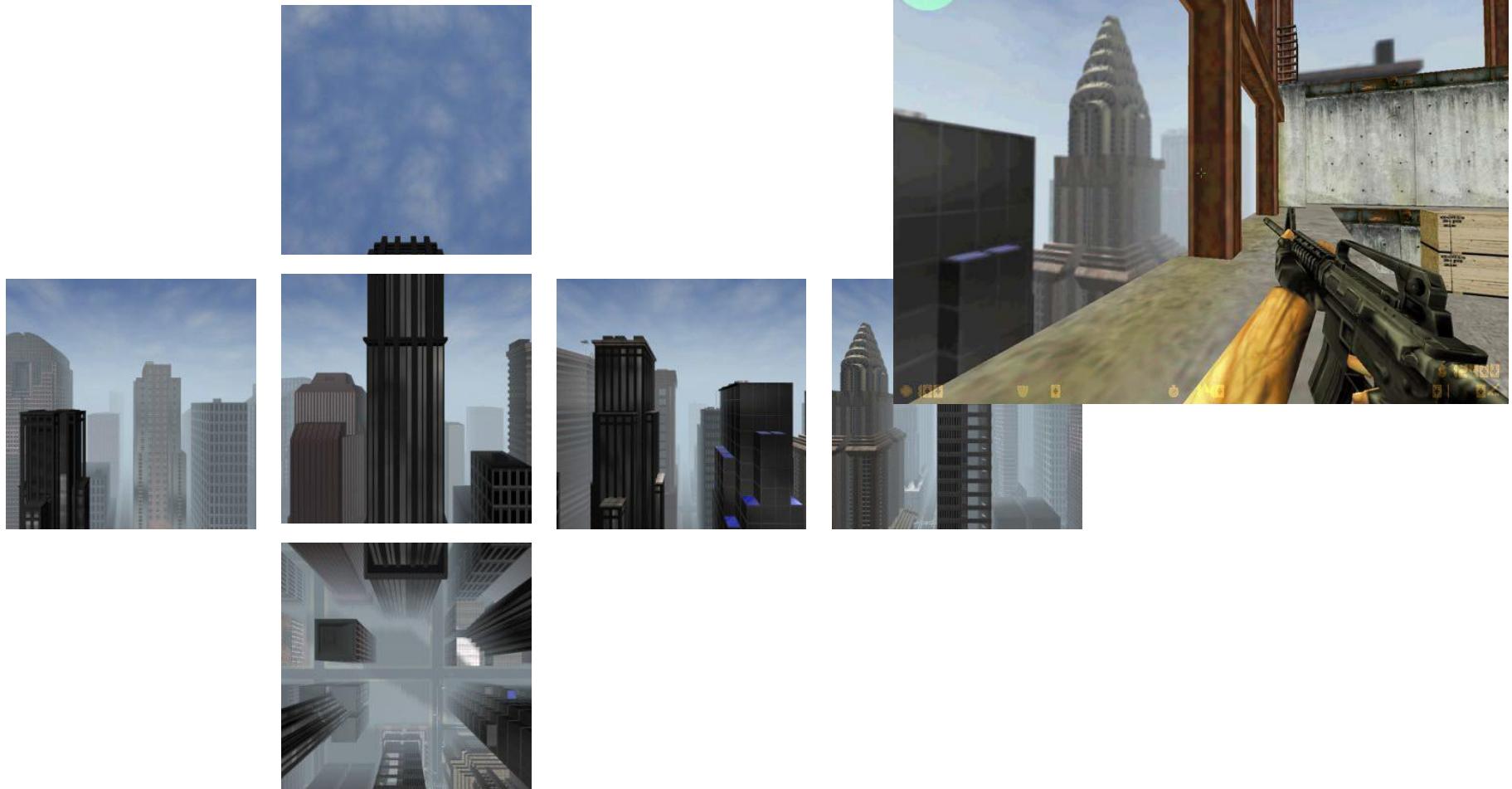
Placage de texture

- Malgré le calcul de la lumière, le rendu n'est pas toujours suffisant :



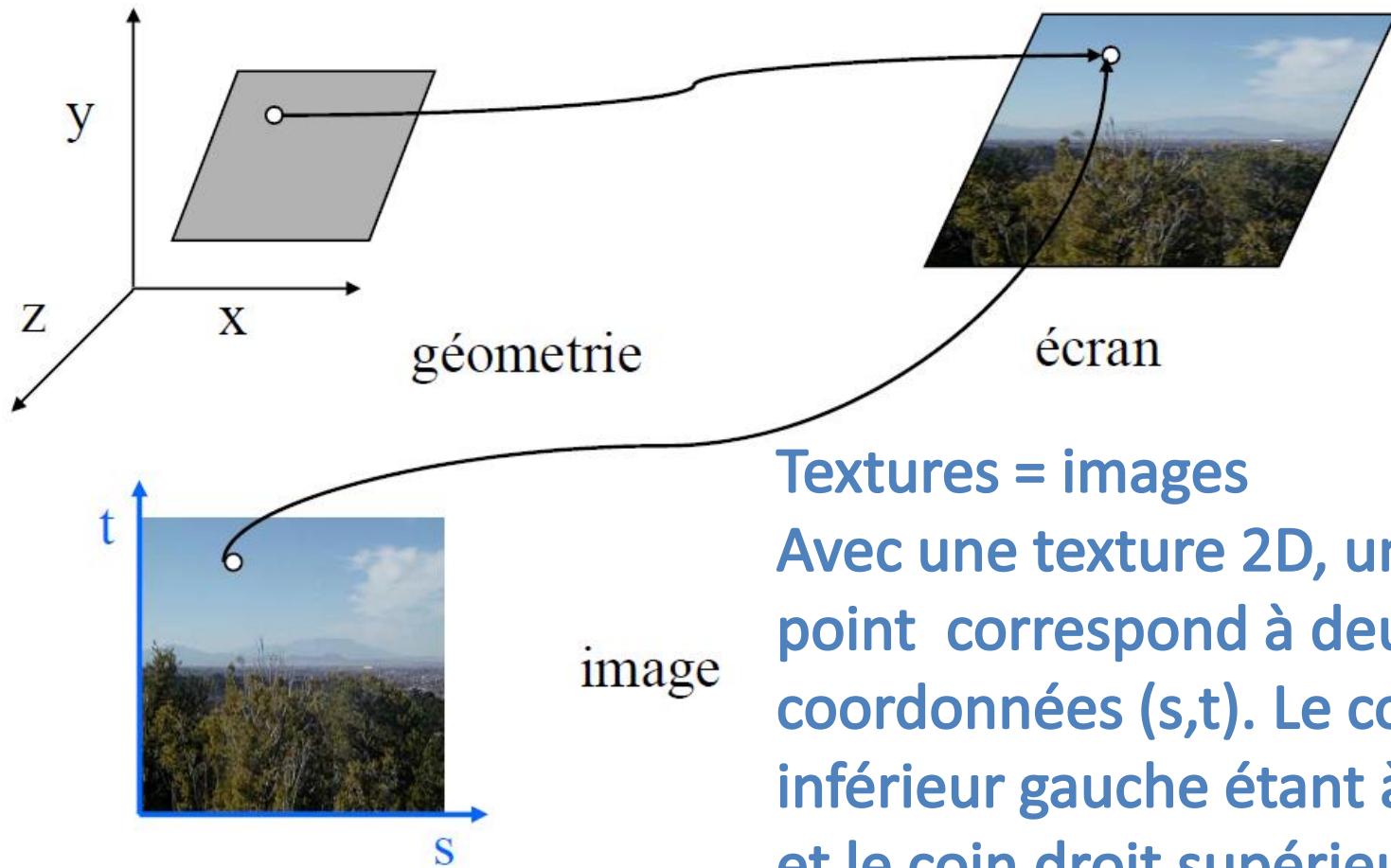
Placage de texture

- Exemple de la sky box :



Placage de texture

- Principe du placage :



Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Textures en OpenGL

- Pour utiliser les textures en OpenGL :
 - Spécifier la texture
 - lire ou générer une image
 - en faire une texture
 - activer le plaquage de texture
 - Assigner les coordonnées de texture aux points de l'objet 3D
 - Spécifier les paramètres de textures
 - Wrapping, filtering ...

Textures en OpenGL

- Spécifier la texture :
 - Lire ou générer une image

```
BYTE    *img;  
  
int     largeur, hauteur;  
  
GLuint texture;  
  
  
glGenTextures(1, &texture);  
  
img = load_tga( "image.tga", &largeur, &hauteur );
```

`glGenTextures` : permet de générer n indices de texture, qui seront stockés dans le tableau passé en paramètres

Textures en OpenGL

- Spécifier la texture :
 - En faire une texture

```
glBindTexture(GL_TEXTURE_2D, texture);  
  
glTexImage2D( GL_TEXTURE_2D, 0, 3,  
              largeur, hauteur,  
              0, GL_RGB, GL_UNSIGNED_BYTE, img);
```

Lorsqu'on charge une image pour en faire une texture, il faut ensuite la transférer dans la RAM vidéo.

Textures en OpenGL

- Spécifier la texture :

- En faire une texture

```
glTexImage2D( target, level, components, w, h,  
              border, format, type, *texels );
```

target : GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D

Level : 0 sans mip-mapping

components : nombre d'éléments par texel

w, h : dimensions de la texture (puissances de 2)

border : bordure supplémentaire autour de l'image

format : GL_RGB, GL_RGBA, ...

type : type des éléments des texels

texels : tableau de texels

Textures en OpenGL

- Spécifier la texture :
 - Activer le placage de texture

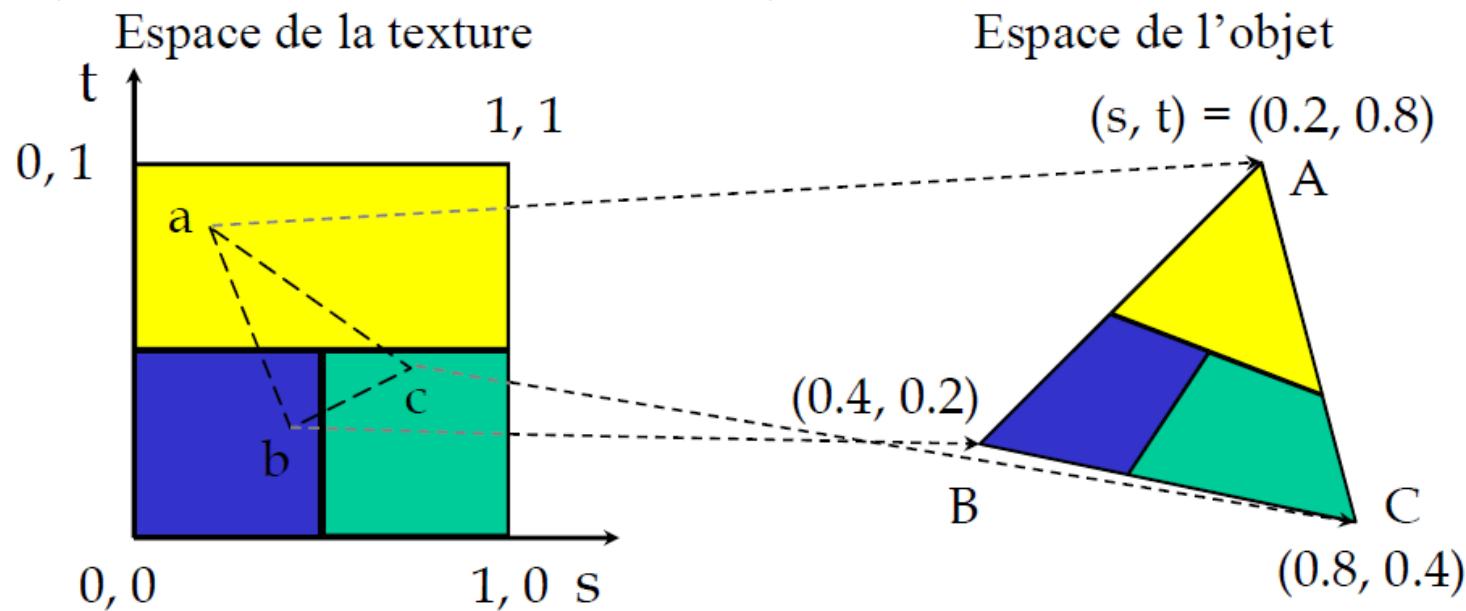
```
glEnable(GL_TEXTURE_2D);
```

- Ou désactiver le placage :

```
glDisable(GL_TEXTURE_2D);
```

Textures en OpenGL

- Assigner les coordonnées de texture aux points de l'objet 3D :
 - pour plaquer une texture sur un objet géométrique, fournir les coordonnées de texture (normalisés entre 0 et 1)



Textures en OpenGL

- Assigner les coordonnées de texture aux points de l'objet 3D :

```
glBindTexture(GL_TEXTURE_2D, texture);  
  
glBegin(GL_TRIANGLES);  
  
    glTexCoord2f(0.0f,0.0f);  
  
    glVertex3f(4.0f, 5.0f, 0.0f);  
  
    glTexCoord2f(1.0f,0.0f);  
  
    glVertex3f(10.0f, 5.0f, 0.0f);  
  
    glTexCoord2f(0.0f,1.0f);  
  
    glVertex3f(4.0f, 12.0f, 0.0f);  
  
glEnd();
```

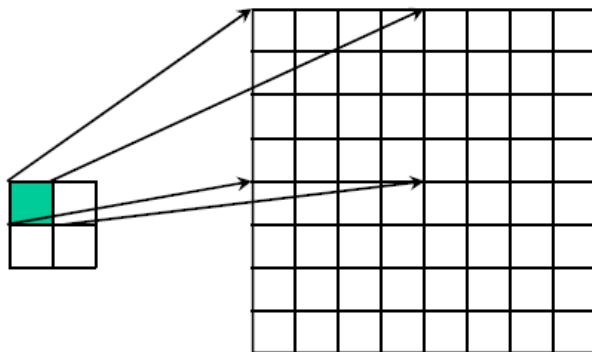
Textures en OpenGL

- Spécifier les paramètres de textures :
 - Modes de filtrage
 - réduction, agrandissement
 - Mip-mapping
 - Modes de bouclage
 - répéter, tronquer
 - Fonctions de textures
 - comment mélanger la couleur d'un objet avec sa texture

Textures en OpenGL

- Modes de filtrage

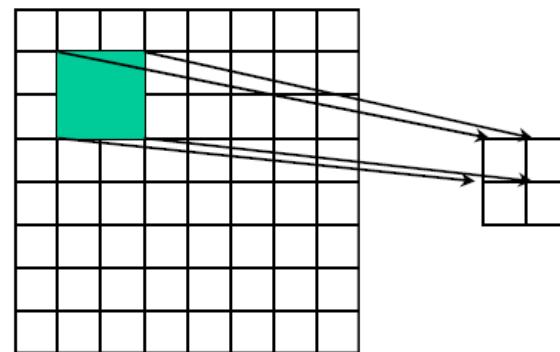
➤ réduction, agrandissement : les textures et les objets n'ont pas souvent la même taille. OpenGL définit des filtres indiquant comment agrandir ou réduire



Texture

Agrandissement

`GL_TEXTURE_MAG_FILTER`



Texture

Réduction

`GL_TEXTURE_MIN_FILTER`

Textures en OpenGL

- Modes de filtrage

```
glTexParameterI(target, type, mode);
```

Avec :

```
target : GL_TEXTURE_2D, ...  
type   : GL_TEXTURE_MIN_FILTER,  
        GL_TEXTURE_MAG_FILTER  
mode   : GL_NEAREST, GL_LINEAR
```

GL_NEAREST = couleur du pixel donnée par celle du texel le plus proche.

GL_LINEAR = couleur du pixel calculé par interpolation linéaire de texels les plus proches.

Textures en OpenGL

- Modes de filtrage
 - Les différents résultats



GL_NEAREST



GL_LINEAR

Textures en OpenGL

- Modes de filtrage

➤ exemple :

```
glBindTexture(GL_TEXTURE_2D, texture);  
  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER,  
                GL_LINEAR);  
  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER,  
                GL_LINEAR);
```

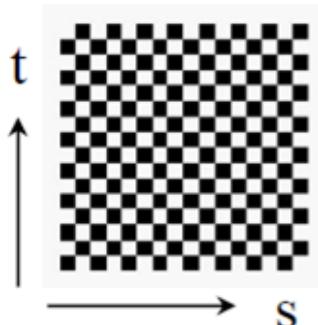
Textures en OpenGL

- Modes de bouclage (Wrap) :
 - ce mode indique ce qui doit se produire si une coordonée de texture sort de l'intervale [0,1]
 - deux solutions :
 - répéter (repeat)
 - tronquer (clamp)

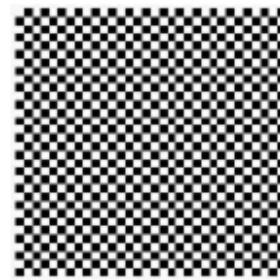
Textures en OpenGL

- Modes de bouclage (Wrap) : répéter
 - si le mode `GL_REPEAT` est utilisé, pour les coordonnées <0 ou >1, la partie entière est ignorée et seule la partie décimale est utilisée.

```
glTexParameteri( GL_TEXTURE_2D,  
                  GL_TEXTURE_WRAP_S,  
                  GL_REPEAT );
```



texture

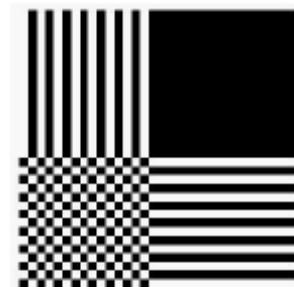
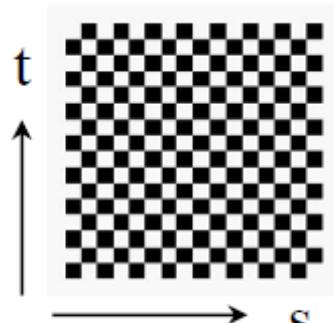


`GL_REPEAT`

Textures en OpenGL

- Modes de bouclage (Wrap) : **tronquer**
 - si le mode **GL_CLAMP** est utilisé, la valeur de la texture aux extrêmes (**<0 ou >1**) est utilisée.

```
glTexParameteri( GL_TEXTURE_2D,  
                  GL_TEXTURE_WRAP_S,  
                  GL_CLAMP );
```



GL_CLAMP
texture

Textures en OpenGL

- Fonctions de textures :
 - Contrôle la manière selon laquelle la texture est mélangée à la couleur de l'objet

```
glTexEnvf( GL_TEXTURE_ENV,  
           GL_TEXTURE_ENV_MODE,  
           param );
```

`param` peut prendre l'une des trois valeurs suivantes :

- `GL_DECAL` : remplace la couleur par le texel.
- `GL_MODULATE` : multiplie le texel par la couleur.
- `GL_BLEND` : mélange le texel, la couleur et `env_color`.

Textures en OpenGL

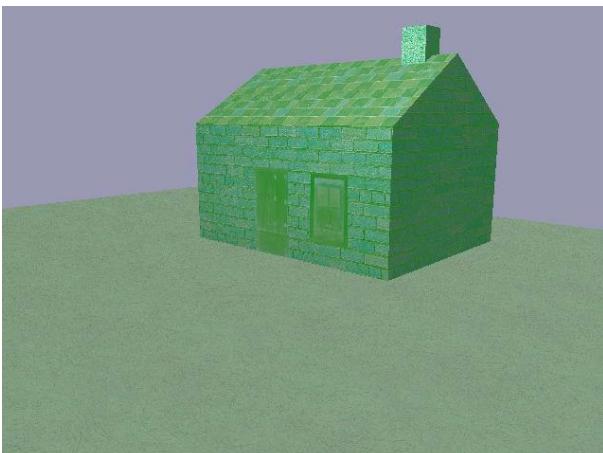
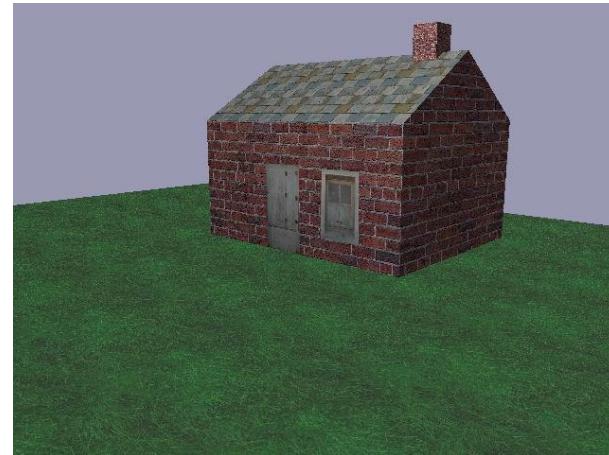
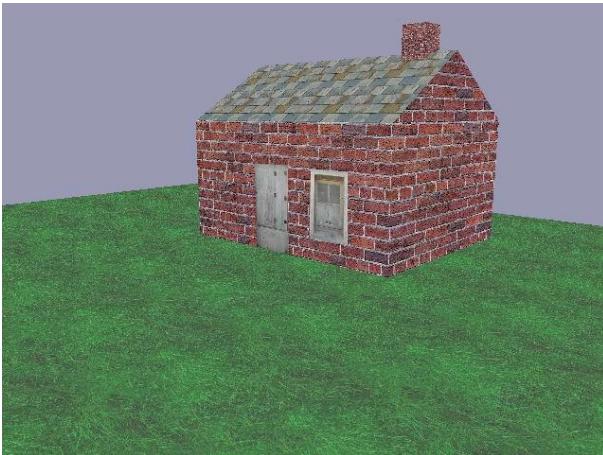
- Fonctions de textures :
 - définition de la couleur `env_color` à utiliser avec la fonction précédente dans le cas où
`param = GL_BLEND`

```
glTexEnvfv( GL_TEXTURE_ENV,  
            GL_TEXTURE_ENV_COLOR ,  
            env_color );
```

`env_color` est un tableau de 4 float représentant la couleur

Textures en OpenGL

- Fonctions de textures :
 - exemple de résultats



GL_BLEND
avec :
env_color [] = {0.0, 0.5, 0.0, 1.0}

Textures en OpenGL

- Exemple complet :

```
// Déclarations de variables

BYTE    *img;
int     largeur, hauteur;
GLuint texture;

// Fin des déclarations de variables
```

Textures en OpenGL

- Exemple complet :

```
// Création d'une texture
// - lecture d'une image
// - chargement en mémoire vidéo
// - réglage des paramètres de la texture

glGenTextures(1, &texture);

img = load_tga( "image.tga", &largeur, &hauteur );
if( img != NULL )
{
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexImage2D( GL_TEXTURE_2D, 0, 3,
                  largeur, hauteur,
                  0, GL_RGB, GL_UNSIGNED_BYTE, img );
    delete[] img;
}
```

Textures en OpenGL

- Exemple complet :

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER,  
                GL_LINEAR);  
  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER,  
                GL_LINEAR);  
  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_WRAP_S,  
                GL_REPEAT);  
  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_WRAP_T,  
                GL_REPEAT);
```

Textures en OpenGL

- Exemple complet :

```
glTexEnvf(GL_TEXTURE_ENV,  
          GL_TEXTURE_ENV_MODE,  
          GL_MODULATE);
```

```
// Fin de la création d'une texture
```

Textures en OpenGL

- Exemple complet :

```
// Utilisation d'une texture

// Si le mode "texture" avait été désactivé,
// on l'active :
glEnable(GL_TEXTURE_2D);

glBegin(GL_TRIANGLES);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(4.0f, 5.0f, 0.0f);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(10.0f, 5.0f, 0.0f);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(4.0f, 12.0f, 0.0f);
glEnd();

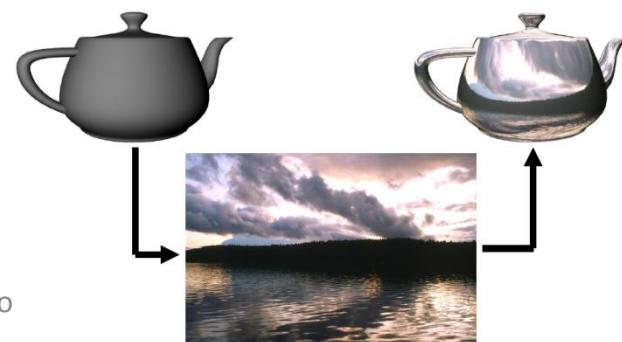
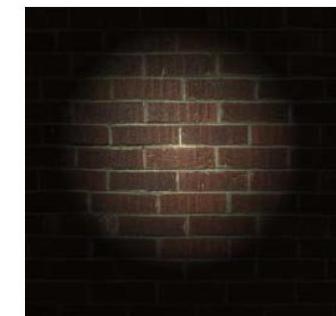
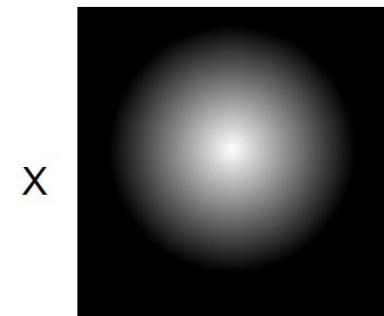
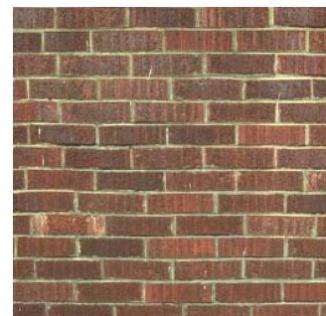
// Fin de l'utilisation d'une texture
```

Plan

- Introduction
- Lumières
 - Illumination locale
 - Illumination d'un objet 3D
 - Lumières en OpenGL
 - Brouillard
- Textures
 - Placage de textures
 - Textures en OpenGL
 - Fonctions avancées

Fonctions avancées

- Il y a plusieurs fonctions OpenGL avancées :
 - Fonctions de correction de textures
 - Gestion de textures en mémoire
 - Alpha-Blending
 - Multitexturing
 - Bump mapping
 - Environnement mapping



Conclusion

- Afin d'améliorer le rendu d'image 3D, il faut utiliser soit
- les lumières :
 - se composent de 3 composantes : ambiant, spéculaire et diffus
 - permettent de calculer les couleurs de chaque pixel
- les textures :
 - plaquer une image 2D sur un objet 3D

FIN

Partiel
lundi 18/05
de 8h30 à 10h30

Pour récupérer les cours et le TD/TP:

<http://www.lirmm.fr/~beniere/Enseignements.php>

Sources

- Cours utilisés pour ce support :
 - Gilles Gesquière (Gamagora, LIRIS, Lyon)
 - Sébastien Thon (LSIS, Université Aix-Marseille)
 - F. Graglia (Université Aix-Marseille)