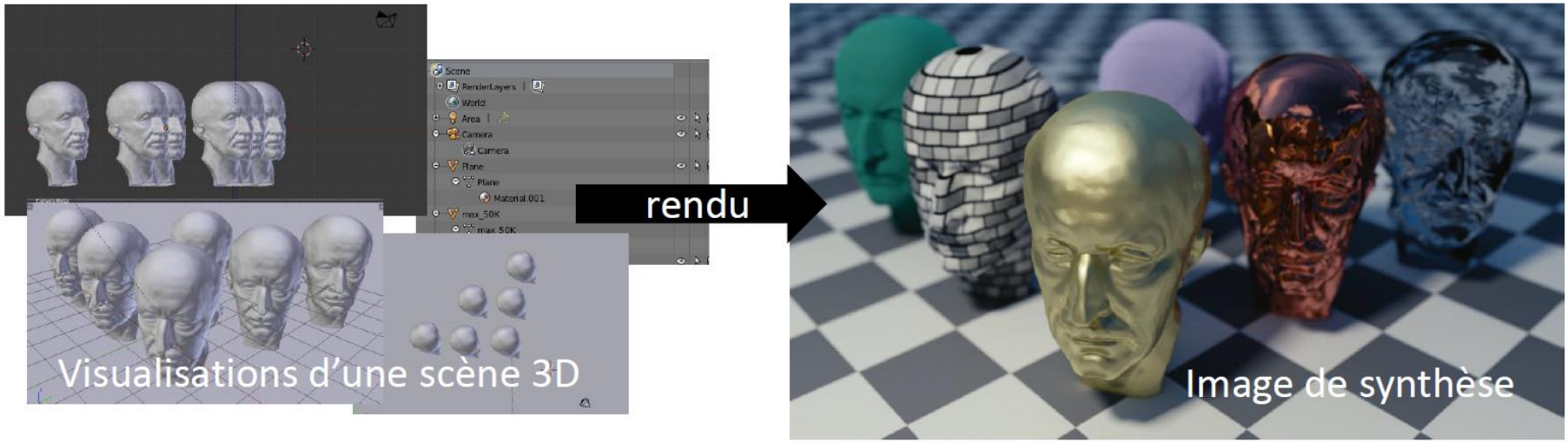
Rendu avancé

Master IMAGINA

Plan du cours

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

Un processus de simulation



- Rendu = Synthèse d'Image
- Génération d'une image numérique à partir d'une scène 3D
- Réaliste (physiquement plausible) ou expressif

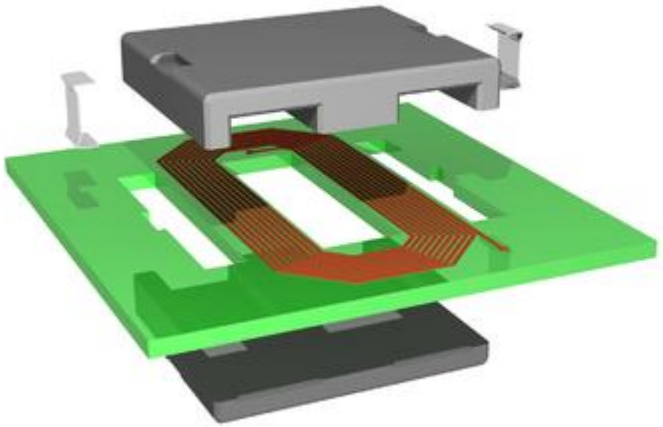
Rendu réaliste

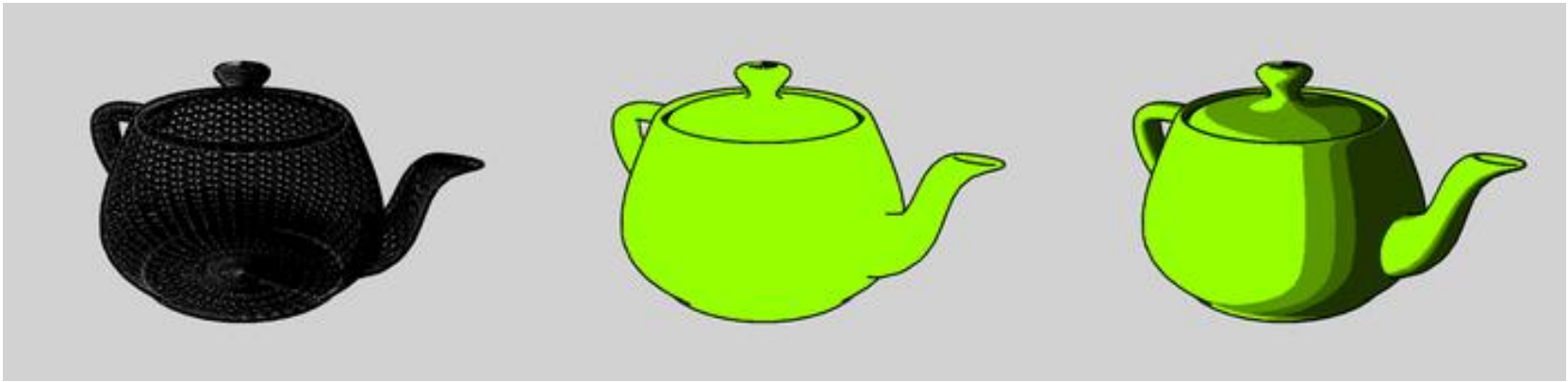
Simulation physique de l’éclairage

Book of the Dead (Unity Demo Studio, 2018)

Rendu expressif

- Effet artistique
- Illustration technique
- Rendu « cartoon »





Types de rendu

- Rendu temps réel

AC - Unity

Fortnite, Epic Games

- Rendu précalculé

Saya (Yuka et Teruyuki Ishikawa)

Ralph, Disney

Rendu pré-calculé

- Images fixes, film d'animation, effets spéciaux …
- Produit par des logiciels de synthèse d'image (CGI)
- Plusieurs minutes (ou heures) pour une image

Rendu temps réel

- Pour les systèmes interactifs
  - simulateur, jeux, visualisation spécifique …
- Produit par des librairies graphiques
- Nécessité de calculé un nombre important d'image par minute (FPS) →sacrifice du réalisme

Red Dead Redemption 2

Jeu

Cinématique

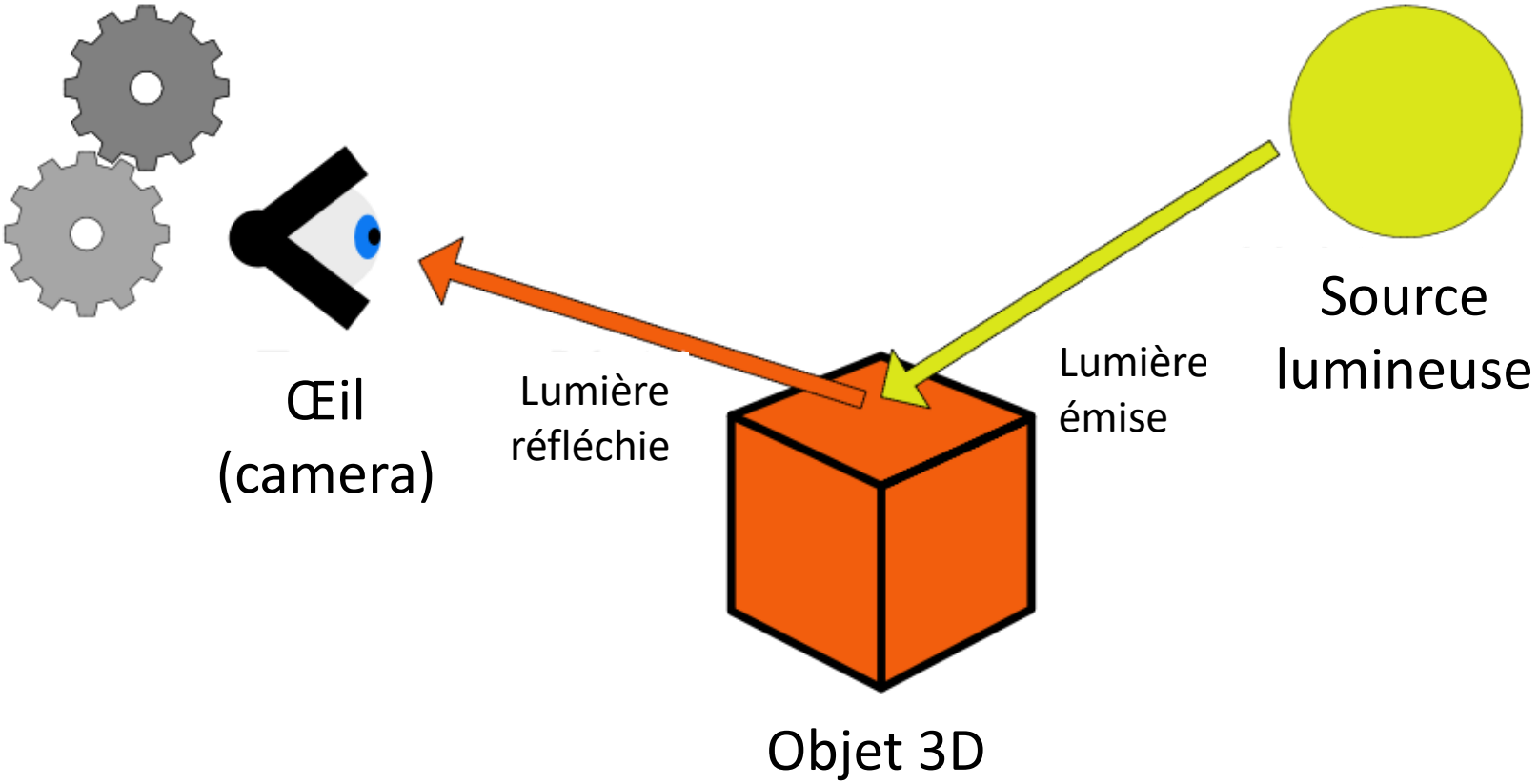
Lumières

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

Unreal engine

Blender

Lumières


Œil
(camera)
Lumière
réfléchie
Lumière
émise
Source
lumineuse
Objet 3D

Lumières

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

- Pas de couleurs ou de rendu 3D sans lumière.

- La manière dont l'objet « réfléchit» la lumière, fait que l'œil et le cerveau « reconstruisent la 3D ».

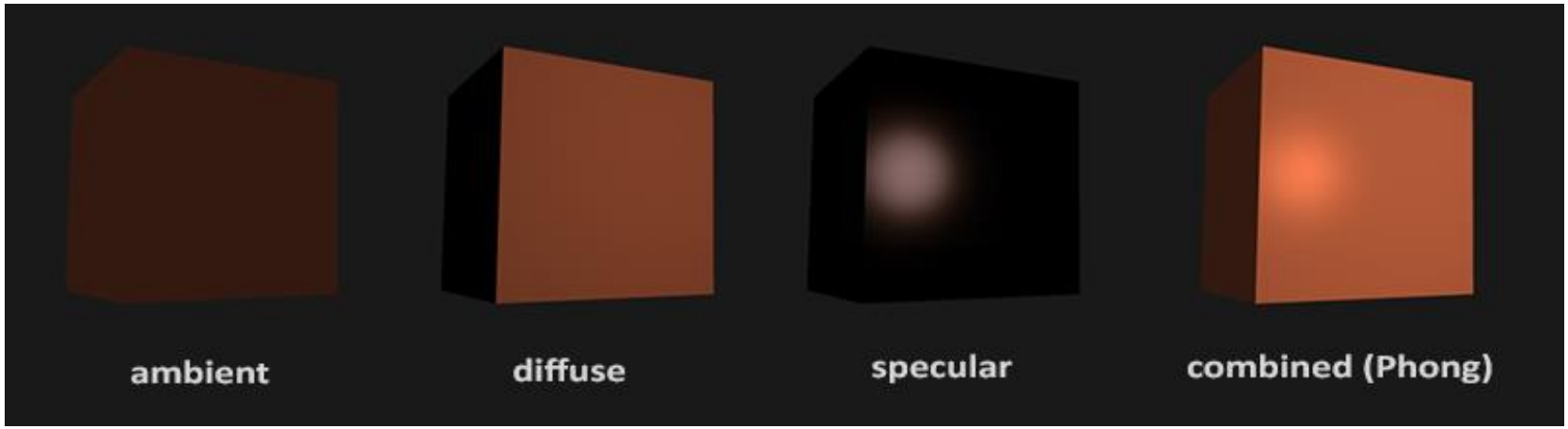
Lumières

- Les matériaux (propriétés physiques) changent les interactions avec la lumière

→ l'aspect visuel de l'objet

Illumination locale

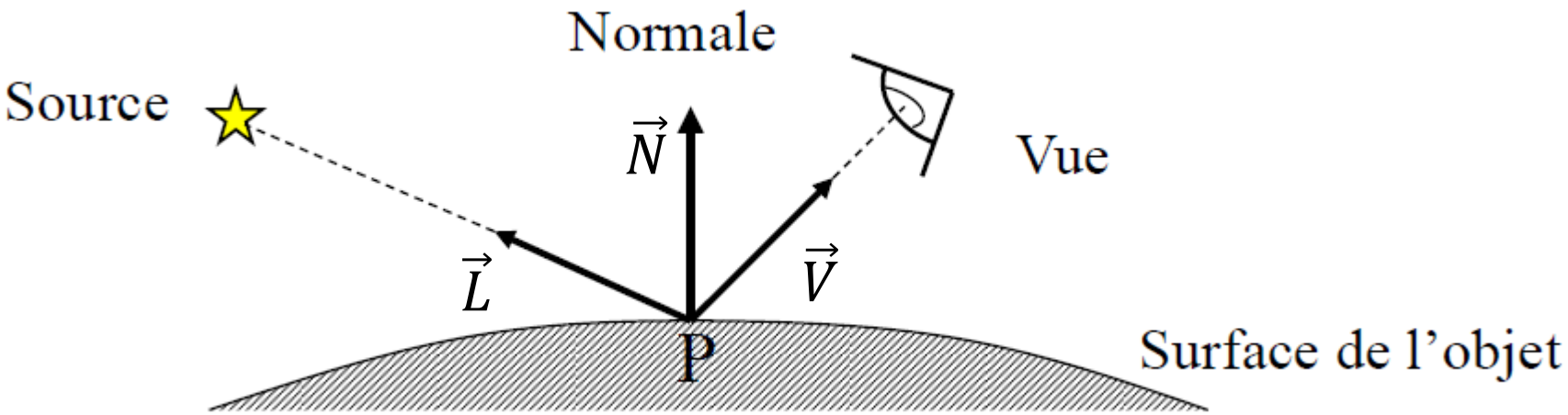
- Théorie : les objets sont vus parce qu'ils réfléchissent la lumière
  - réflexion ambiante
  - réflexion diffuse
  - réflexion spéculaire



En OpenGL : Phong = ambiante + diffuse + speculaire

Ingrédients géométriques

- Pour chaque point P de la surface :
  - Vecteur normal $\vec{N}$
  - Vecteur de direction de vue (camera) $\vec{V}$
  - Vecteur de direction de la source lumineuse $\vec{L}$



Réflexion ambiante

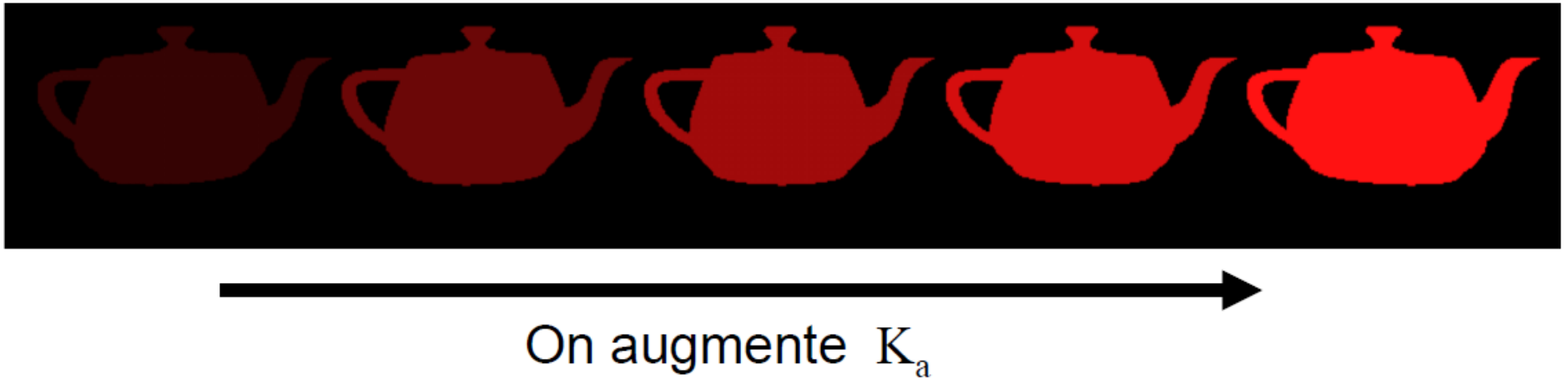
- Parasites provenant d'autre chose que la source considérée
  - lumière réfléchie par d'autres points
  - supposée égale en tout point de l'espace

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

- I_a : intensité de la lumière ambiante réfléchie
- I_{sa} : intensité de la lumière ambiante
- $K_a \in [0,1]$: coeff. de réflexion ambiante de l'objet

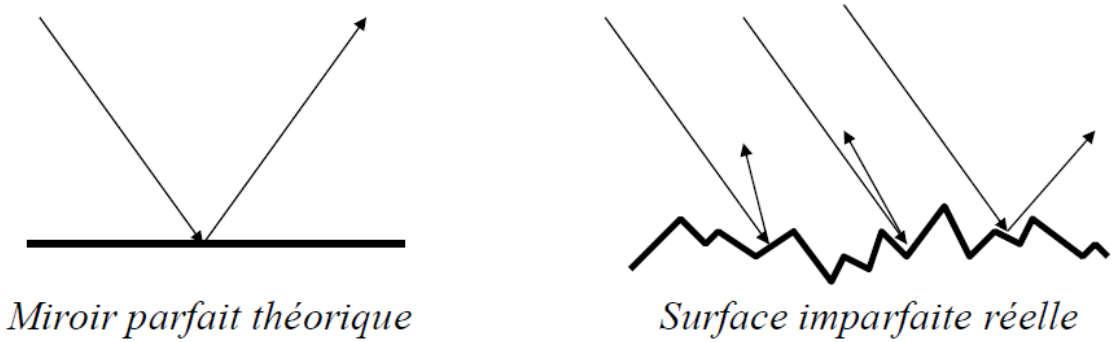
Réflexion ambiante

- Couleur ambiante d'un objet ne dépend que du coefficient de réflexion ambiante Ka de l'objet, pas de sa position par rapport à la lumière



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



Miroir parfait théorique *Surface imparfaite réelle*

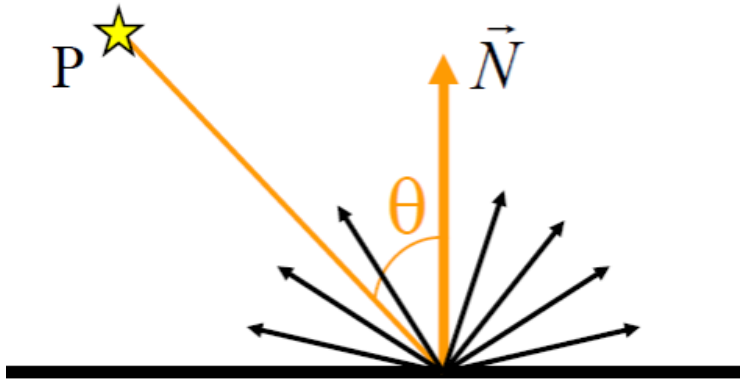
- Directions réparties selon une composante **diffuse** et une composante **spéculaire**, ajoutées à la composante ambiante pour donner plus de relief à l'objet.

Réflexion diffuse

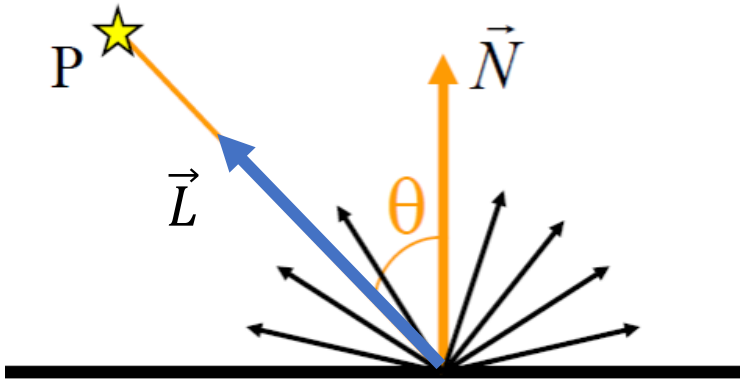
- Lumière réfléchie dans toutes les directions

→ indépendante de la position de l'observateur

- La couleur de l'objet dépend :
  - de l'angle θ entre la direction de la source et la normale
  - du coefficient de réflexion diffuse K_d de l'objet



Réflexion diffuse



- Loi de Lambert

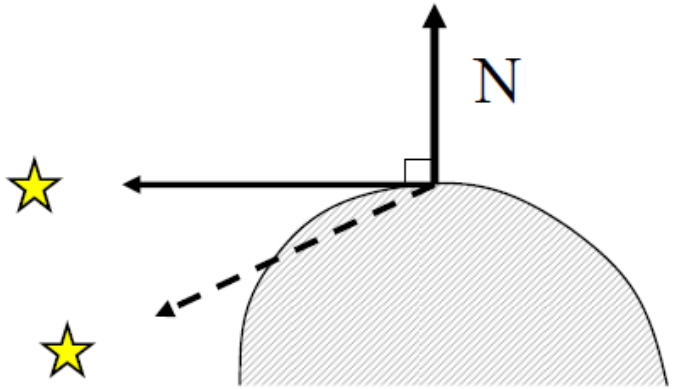
$$I_d = I_{sd} * K_d * \cos\theta$$

- I_d : intensité de la lumière diffuse réfléchie
- I_{sd} : intensité de la lumière diffuse
- $K_d \in [0,1]$: coeff. de réflexion diffuse du matériau
- θ : angle entre la source de lumière et la normale

- On peut également écrire :

$$I_d = I_{sd} * K_d * (\vec{L}.\vec{N})$$

Réflexion diffuse



- Loi de Lambert

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

- Maximale pour $\theta = 0°$ (source de lumière à la verticale de la surface, au zénith)
- Nulle pour un éclairage rasant $\theta = 90°$
- Si $\theta = 90°$ alors le point n'est pas visible par la source de lumière

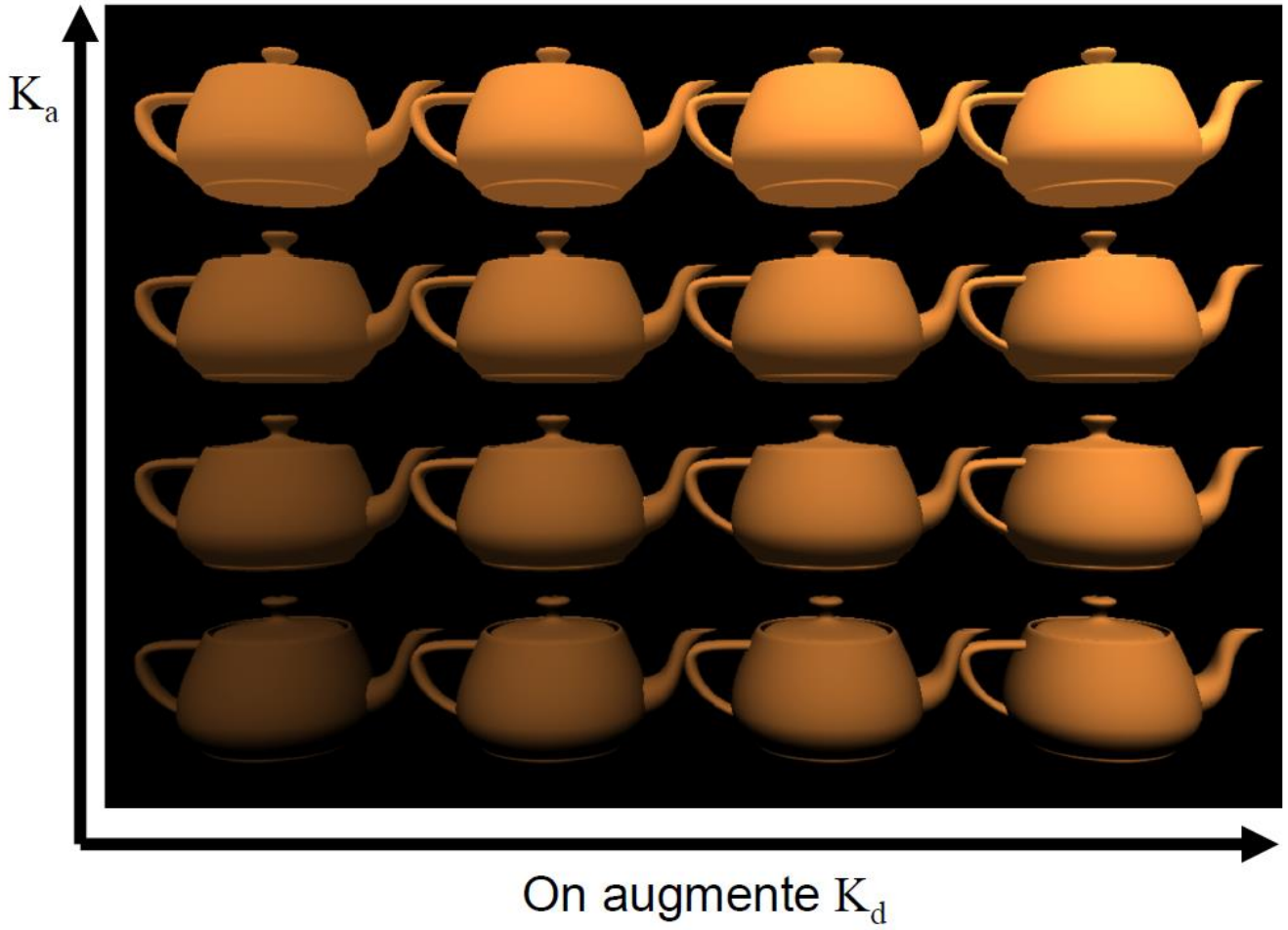
Réflexion diffuse

- Seule

On augmente K_d (avec $K_a = 0$)

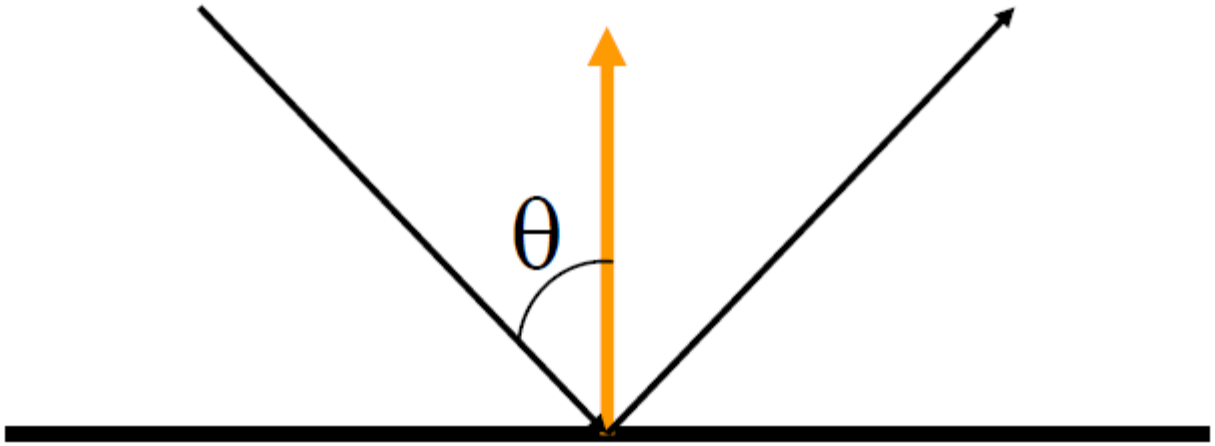
Réflexion diffuse

- Diffuse + ambiante



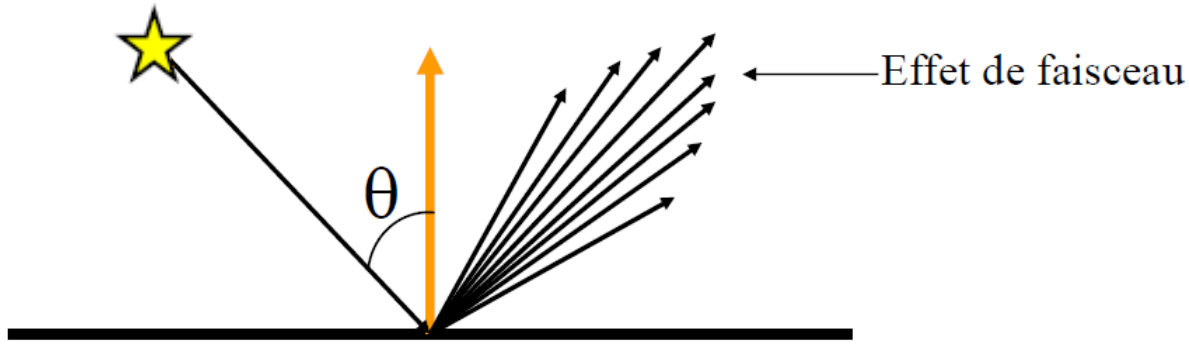
Réflexion spéculaire

- 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



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.

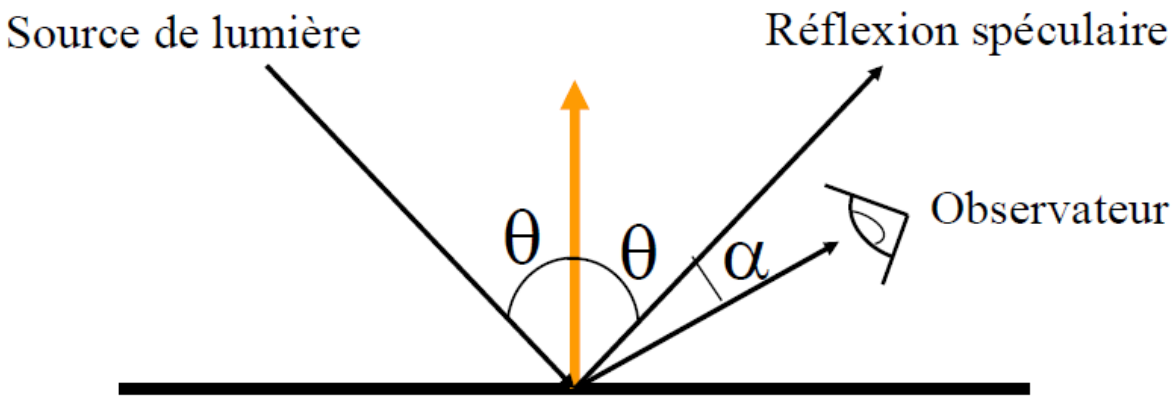


Réflexion spéculaire

- 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
- $K_s \in [0,1]$: coeff. de réflexion spéculaire du matériau
- α : angle entre les directions de réflexion et de la vue



Réflexion spéculaire

- On peut également écrire

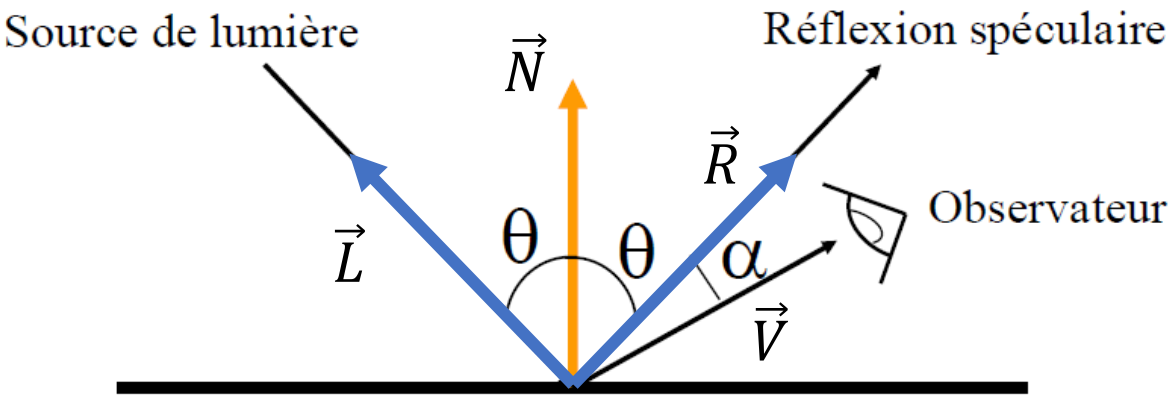
$$I_s = I_{ss} * K_s * \left(\vec{R}.\vec{V}\right)^n$$

avec

$\vec{R}$: direction de réflexion de la lumière sur un miroir

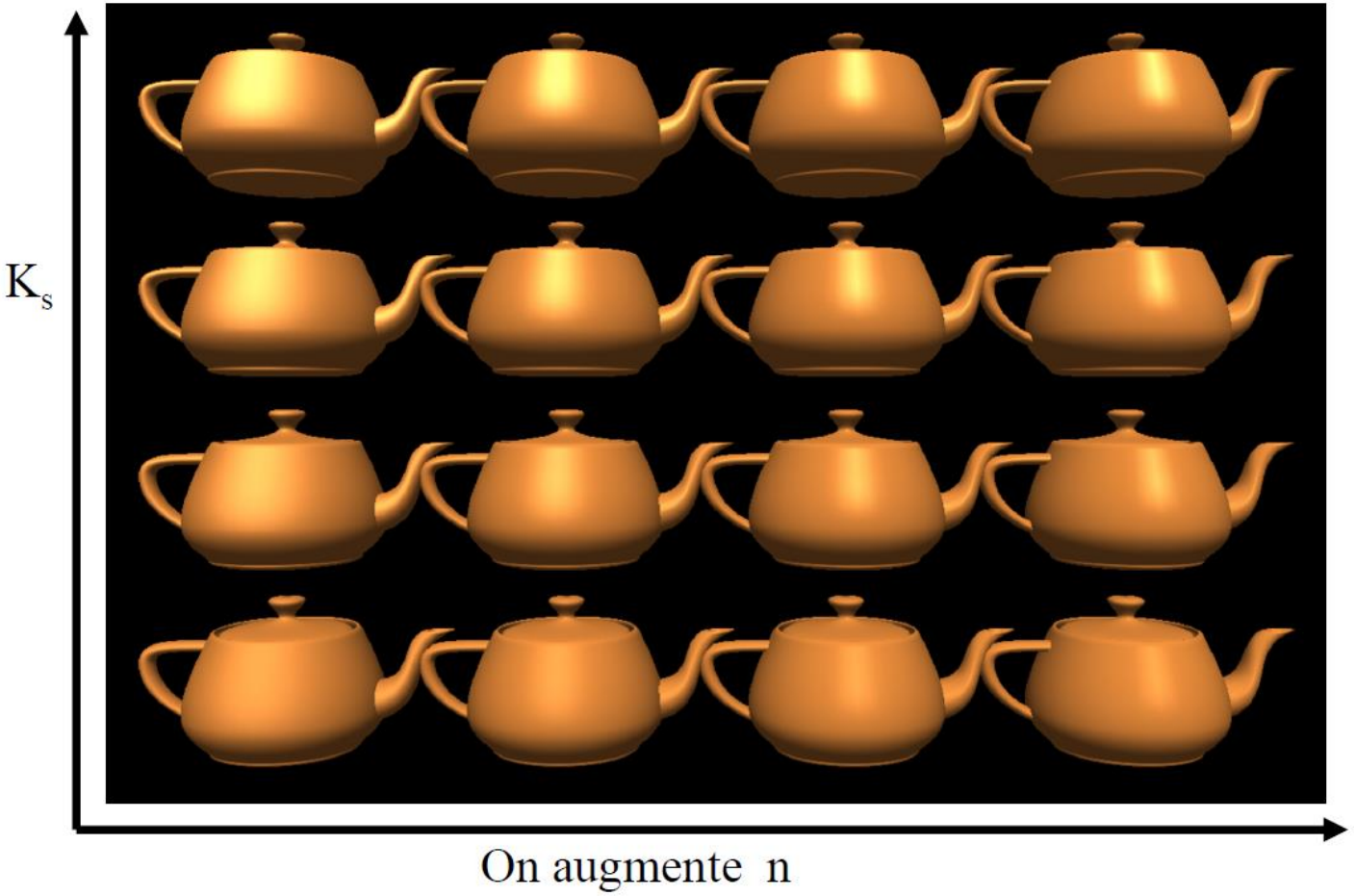
et

$$\vec{R} = 2\left(\vec{N}.\vec{L}\right)\vec{N} - \vec{L} = 2\cos\theta\ \vec{N} - \vec{L}$$



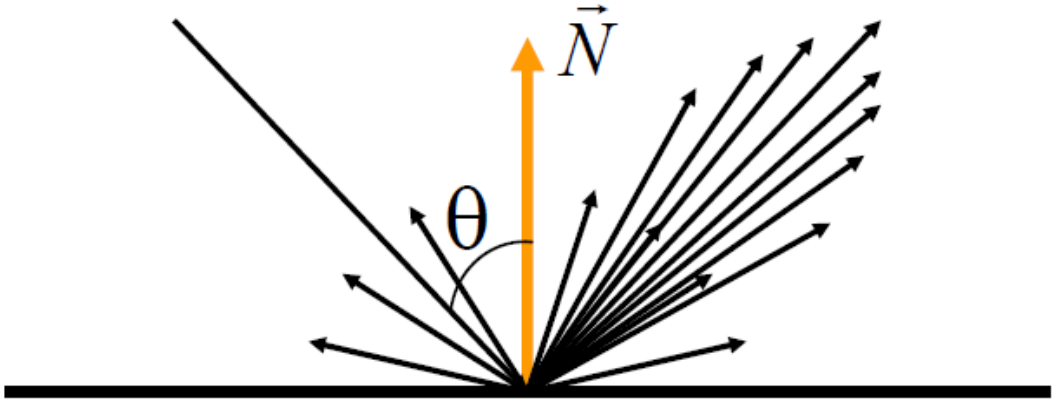
Réflexion spéculaire

$$I_s = I_{ss} * K_s * \cos \alpha^n$$

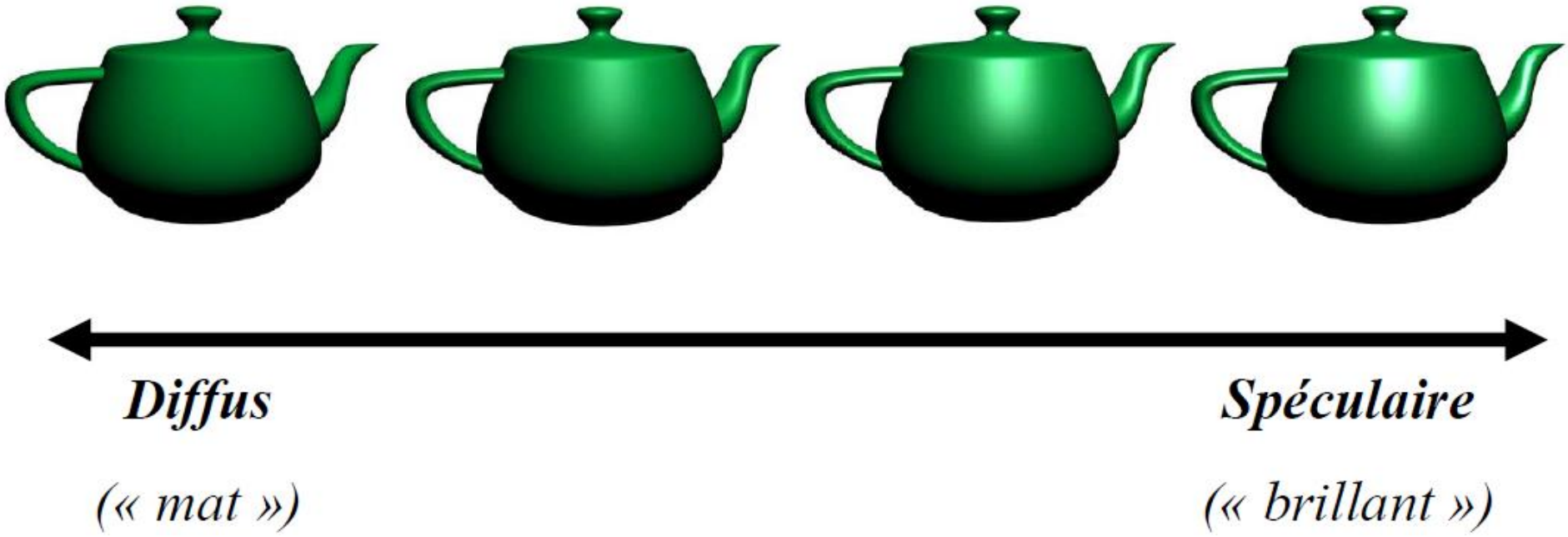


Modèle de Phong

- Dans la réalité, lumière réfléchie par une surface
  - réflexion diffuse + réflexion spéculaire
- Proportions de réflexion diffuse et spéculaire dépendent du matériau :
  - plus diffus (craie, papier …)
  - que spéculaires (métal, verre …)



Réflexion finale



Equation de Phong

- Egale à la somme des réflexions ambiante, diffuse et spéculaire :

$$I = I_a + I_d + I_s$$

- Plusieurs sources lumineuses : somme des intensités

$$I = I_{sa}K_a + \sum_{l \in \{sources\}} I_{ld}K_d(\overrightarrow{L_l}.\vec{N}) + I_{ls}K_s(\overrightarrow{R_l}.\vec{V})^n$$

Calcul de la couleur

- Addition l'intensité lumineuse de chacune des composantes de la couleur.
- RVB : intensités rouge, verte et bleue

→On définit pour chacune de ces 3 composantes

- les caractéristiques des sources de lumière
  $(I_{saR}, I_{saG}, I_{saB})$; $(I_{sdR}, I_{sdG}, I_{sdB})$; $(I_{ssR}, I_{ssG}, I_{ssB})$

- les caractéristiques des matériaux
  (K_{aR}, K_{aG}, K_{aB}); (K_{dR}, K_{dG}, K_{dB}); (K_{sR}, K_{sG}, K_{sB})

Calcul de la couleur

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

$$I_R = I_{saR}K_{aR} + I_{sdR}K_{dR}\cos\theta + I_{ssR}K_{sR}\cos\alpha^n$$
$$I_G = I_{saG}K_{aG} + I_{sdG}K_{dG}\cos\theta + I_{ssG}K_{sG}\cos\alpha^n$$
$$I_B = I_{saB}K_{aB} + I_{sdB}K_{dB}\cos\theta + I_{ssB}K_{sB}\cos\alpha^n$$

Ou

$$I_R = I_{saR}K_{aR} + I_{ldR}K_{dR}(\overrightarrow{L_l}.\vec{N}) + I_{lsR}K_{sR}(\overrightarrow{R_l}.\vec{V})^n$$
$$I_G = I_{saG}K_{aG} + I_{ldG}K_{dG}(\overrightarrow{L_l}.\vec{N}) + I_{lsG}K_{sG}(\overrightarrow{R_l}.\vec{V})^n$$
$$I_B = I_{saB}K_{aB} + I_{ldB}K_{dB}(\overrightarrow{L_l}.\vec{N}) + I_{lsB}K_{sB}(\overrightarrow{R_l}.\vec{V})^n$$

Modèle de Phong

- Avantages
  - très pratique (simple à utiliser, résultats intéressants)
  - rapide à calculer
- Désavantages
  - pas de sens physique
  - pas de lien avec les propriétés du matériau (rugosité …)

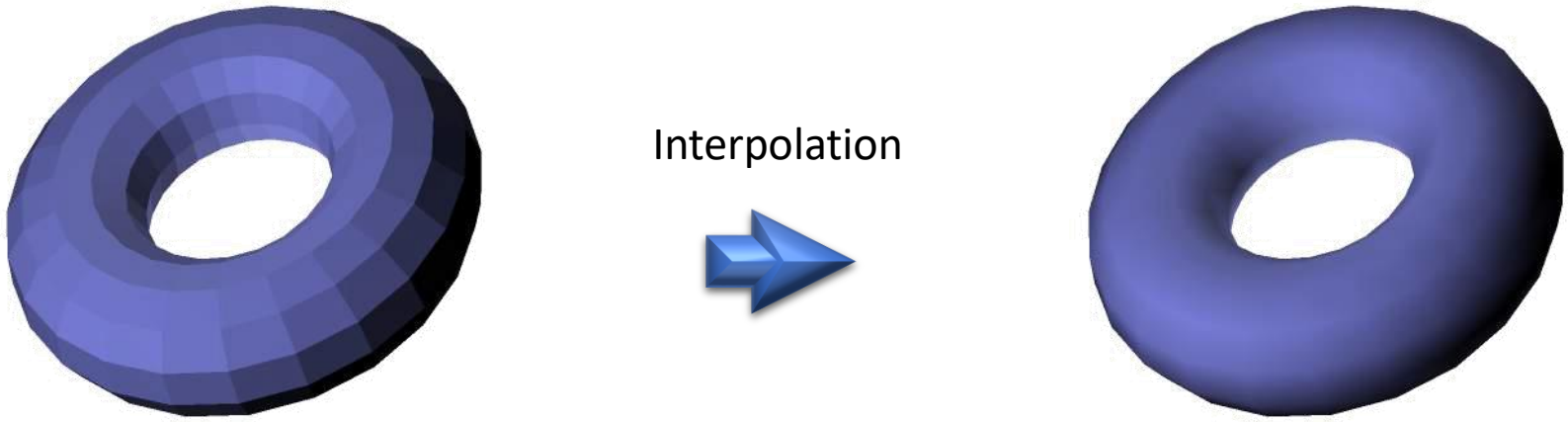
Autres modèles d’éclairages

- Plus efficaces ou réalistes
  - Cook-Torrance (1982)
  - Oren-nayar (1994)
  - Minnaert
  - Subsurface scattering (SSS)
  - BRDF
  - PBR modèle
  - Illumination globale

- Pipeline graphique programmable

Illumination d'un objet 3D

- 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

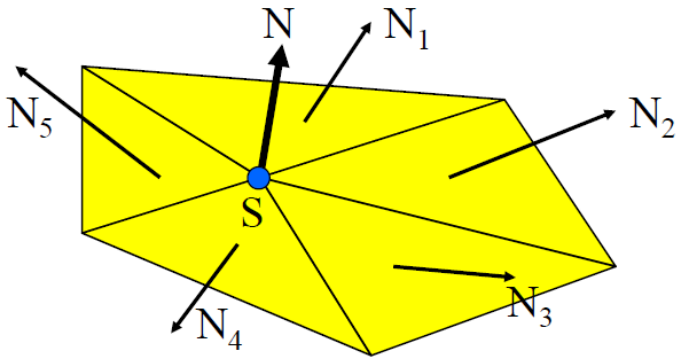
- But : calculer une couleur pour chaque point visible à l’écran de l’objet 3D qu’on affiche.
  - Lissage de **Gouraud** : interpolation de **couleurs**
  - Lissage de **Phong** : interpolation de **normales**

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.

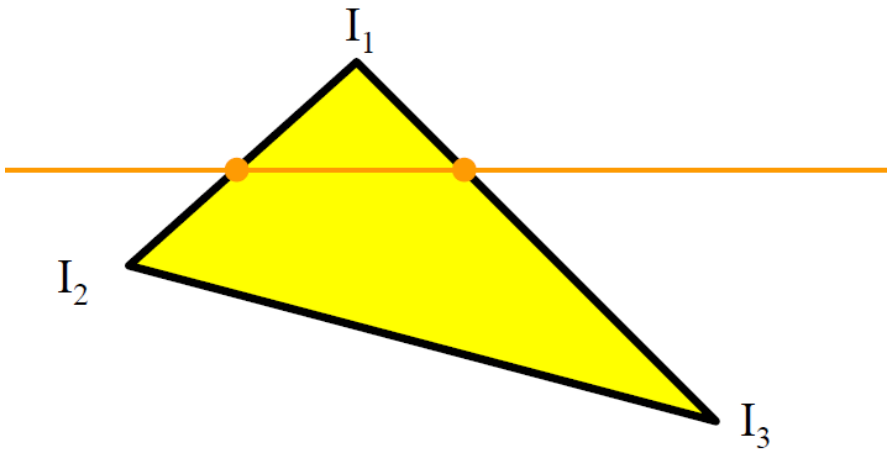
Interpolation de l'illumination

- Pour calculer la couleur en un sommet, on a besoin d'une normale en ce point :
  - surface analytiquement connue (ex : une sphère, un cylindre …) → calcul direct
  - surface de départ est un maillage polygonal, comment faire ?? → interpoler les normales de face



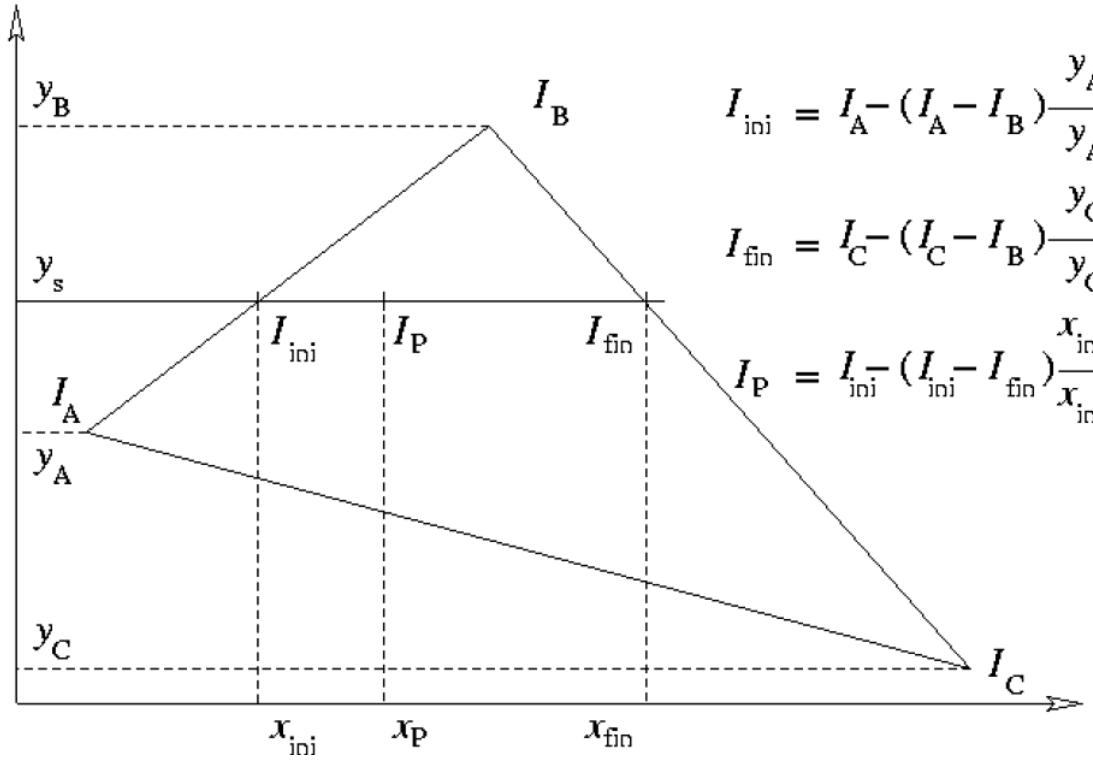
Interpolation de l'illumination

- 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** :



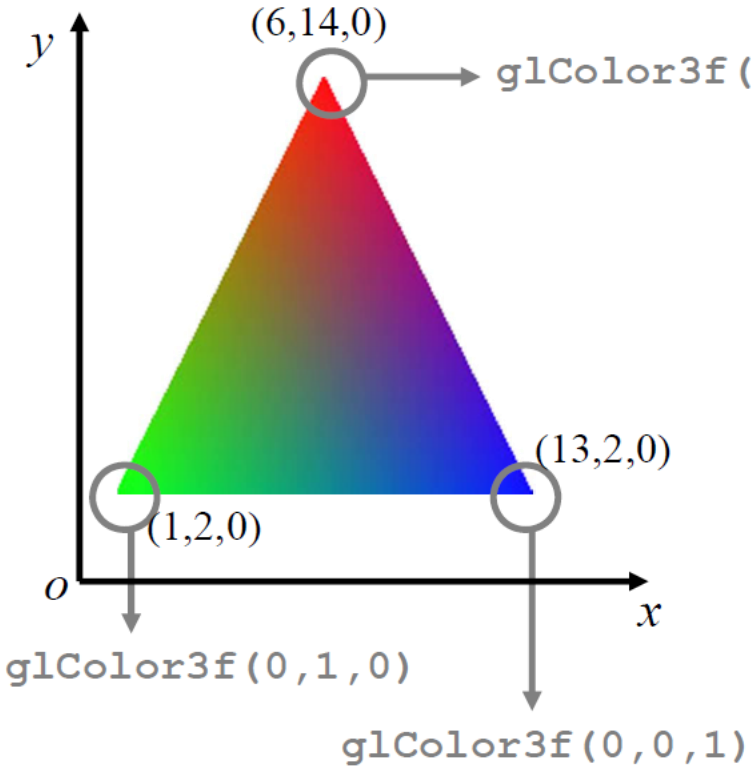
$$I_{ini} = I_A - (I_A - I_B)\frac{y_A - y_s}{y_A - y_B}$$

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

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

En OpenGL

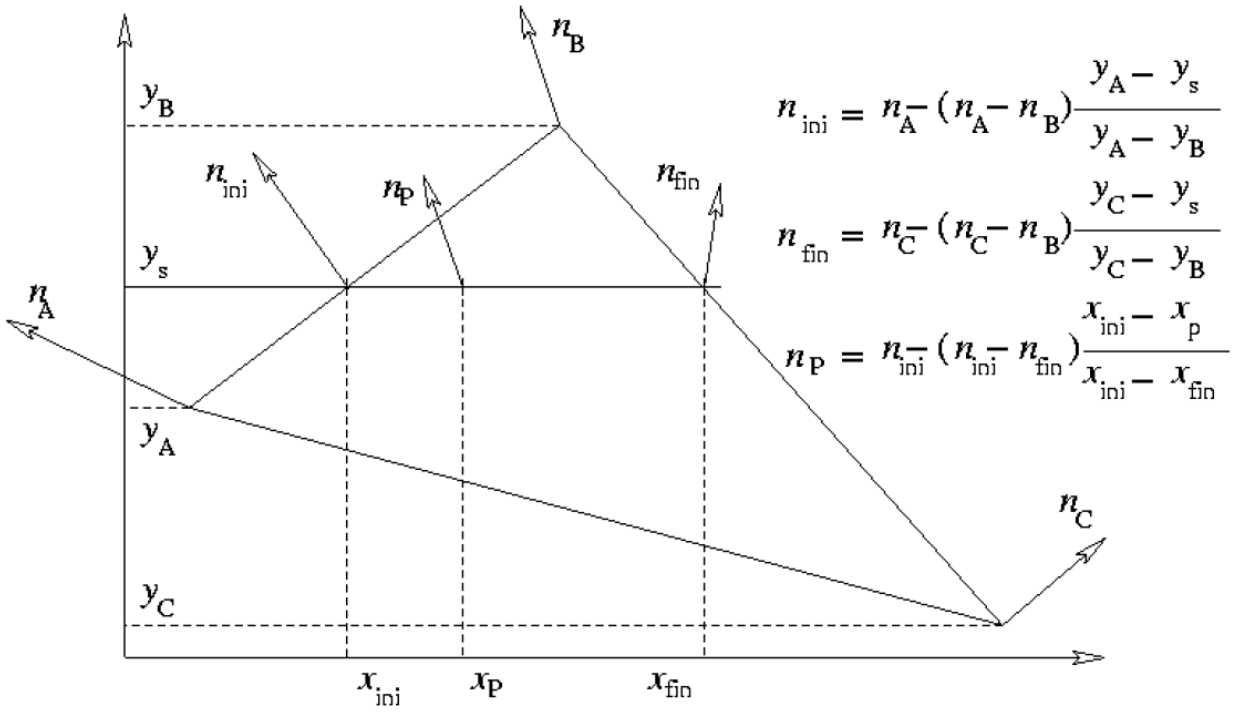
- Interpolation des couleurs données pour les 3 sommets d'un triangle, pour calculer la couleur de tous les autres points du triangle :




```
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();
```

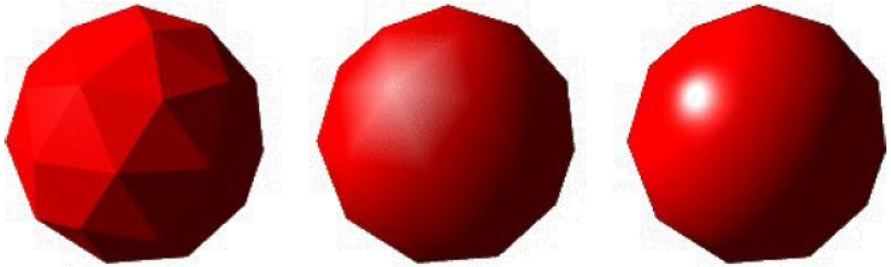
Illumination d'un objet 3D

- Interpolation de l'illumination par Phong :

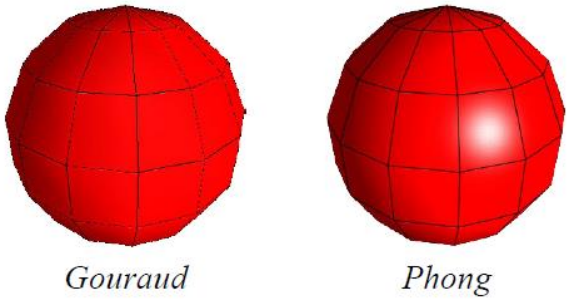


Illumination d’un objet 3D

- 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

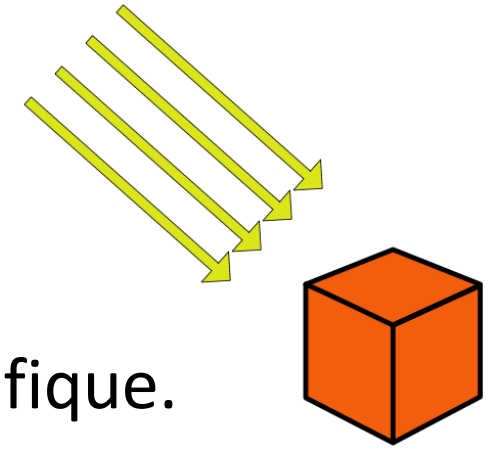


Lumières en OpenGL

- Plusieurs type d’éclairage :
  - source directionnelle
  - source ponctuelle
  - source projecteur

Lumière directionnelle

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



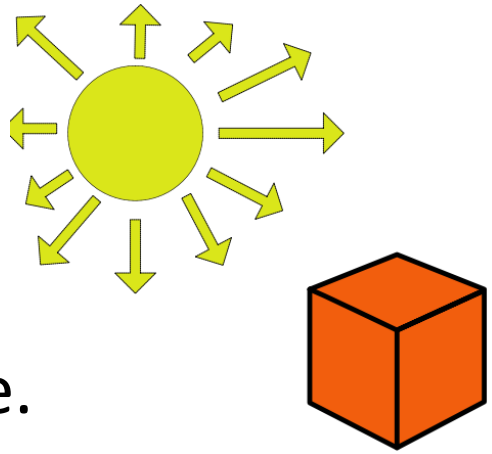
- La lumière vient d'une direction spécifique.

Coordonnées homogène d'un vecteur

```
float position[] = {0.0, -1, 0.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Source ponctuelle

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



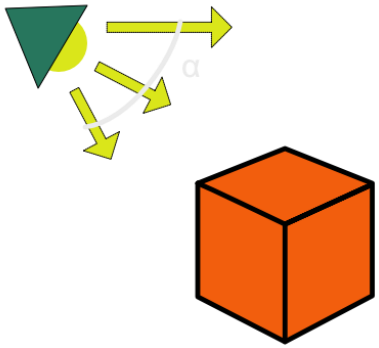
- La lumière vient d'un point spécifique.

Coordonnées homogène d'un point

```
float position[] = {0.0, -1, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Lumière projecteur = « spot »

- La lumière vient d'un point spécifique,
  - intensité dépendant de la direction



- Position : emplacement de la source
- Direction : axe central de la lumière
- Angle : largeur du rayon

```
float position[] = {0.0, 10.0, 0.0, 1.0};
float direction[] = {1.0, -1.0, 0.5};
float angle = 45.0f;
glLightfv(GL_LIGHT0, GL_POSITION, position);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction);
glLightf(GL_LIGHT0, GL_CUTOFF, angle);
```

Exemple

```
// 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);
```

Lumières 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)
```

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>)

- Coefficient de brillance n du cos(α) de la réflexion spéculaire

Exemple

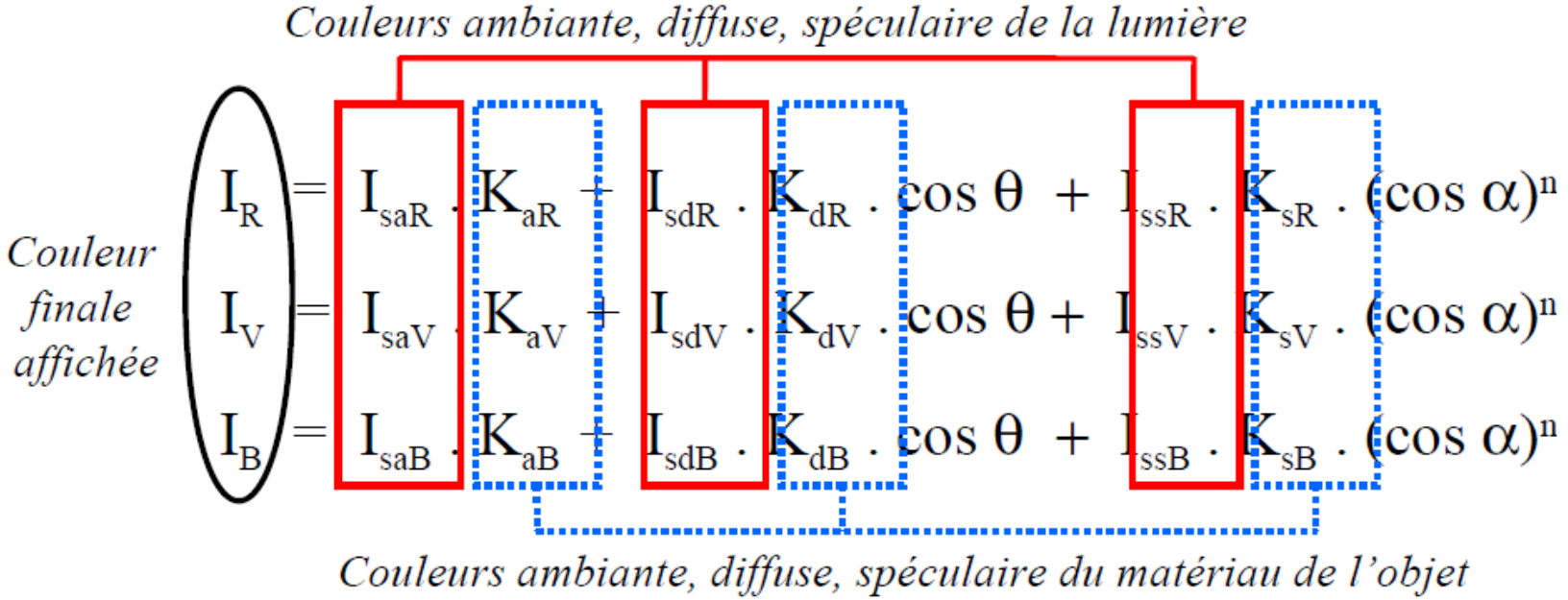
```
GLfloat MatSpec[4] = {1.0f, 1.0f, 1.0f, 1.0f};
glMaterialfv(GL_FRONT, GL_SPECULAR, MatSpec);
```

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

Etant donné les couleurs ambiante, diffuse, spéculaire de la lumière, les composantes du matériau d'un objet, la couleur finale sera calculée grâce à l'équation du **modèle de Phong** :



Brouillard

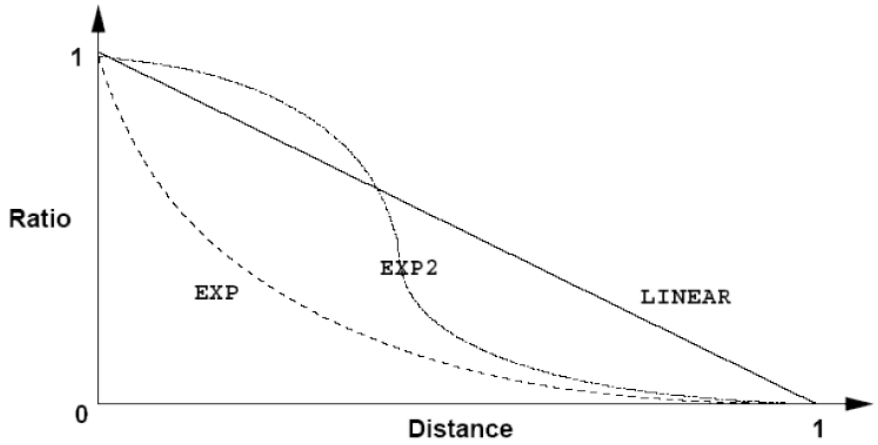
- OpenGL permet d'appliquer un effet de brouillard selon deux modes :
  - atténuation exponentielle (par défaut)
  - atténuation linéaire

- atténuation linéaire
- atténuation exponentielle
- atténuation exponentielle 2

$\texttt{GL\_LINEAR} : f = \frac{end - z}{end - start}$

$\texttt{GL\_EXP} : f = e^{-(density.z)}$

$\texttt{GL\_EXP2} : f = e^{-(density.z)^2}$



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);
```

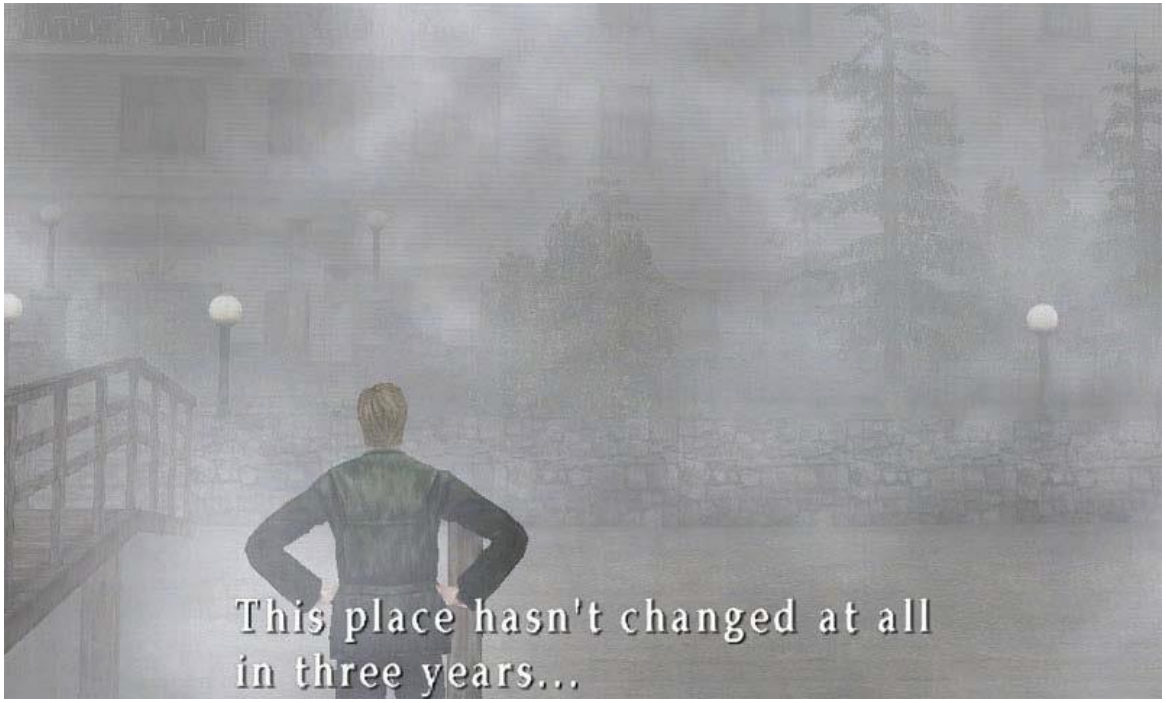
Brouillard

- 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);
```

Intérêt du brouillard

- Reproduction d’un phénomène naturel
- Donner une ambiance (angoissante, mystérieuse …)

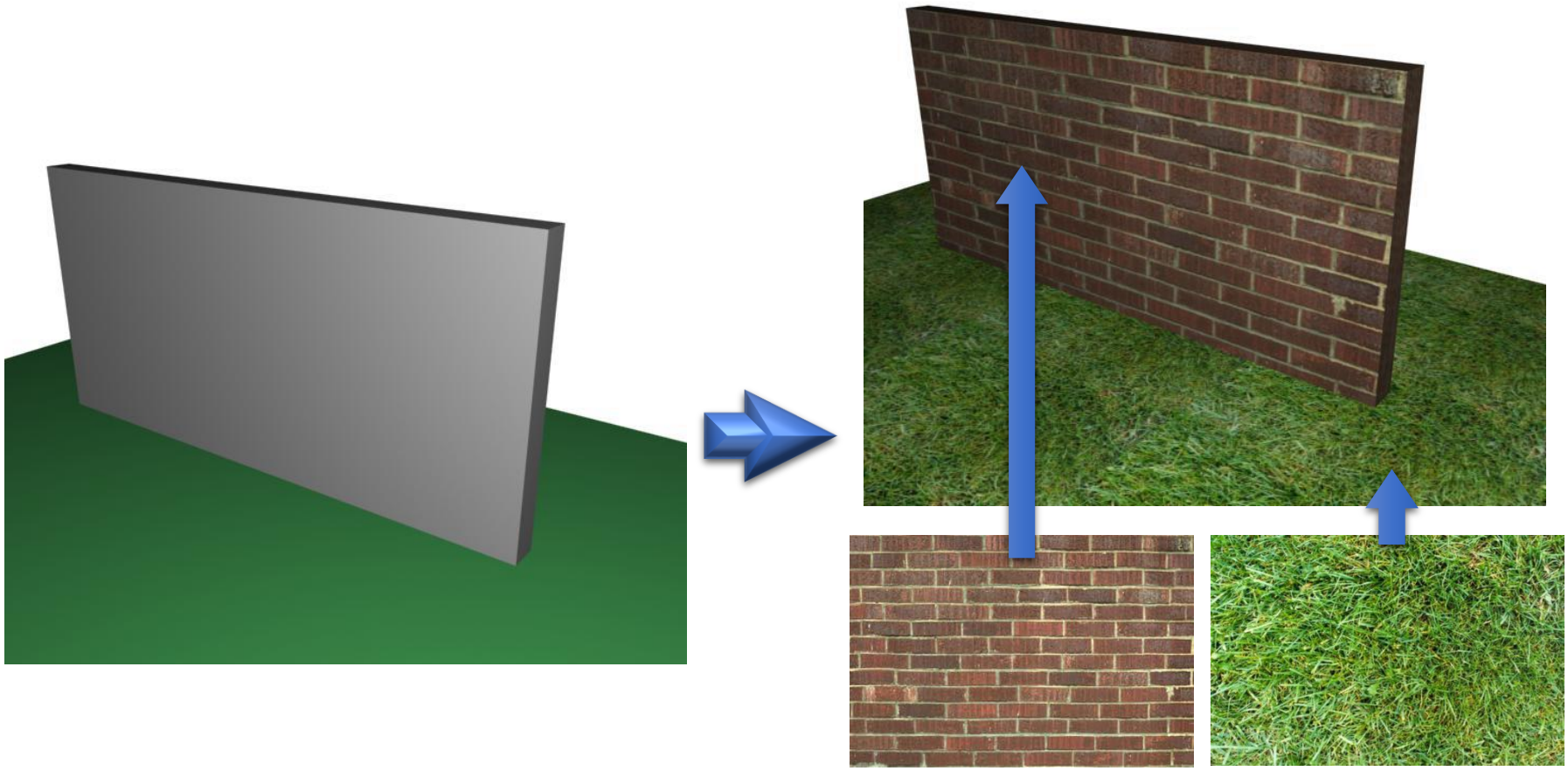


Intérêt du brouillard

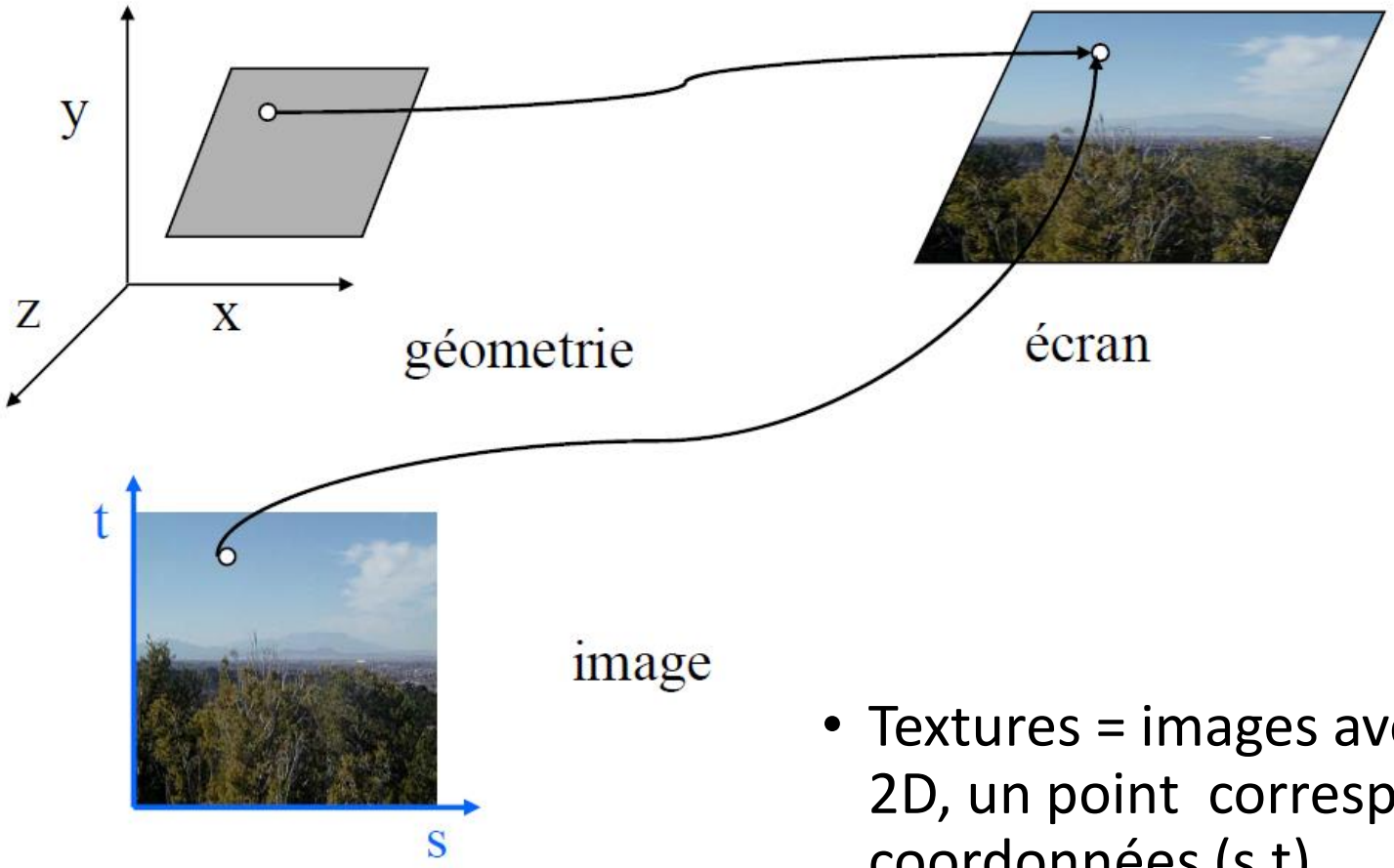
- Effet de profondeur en simulant le « bleu atmosphérique » : atténuation de la lumière dans l'air due aux gouttelettes d'eau en suspension, impuretés …
- Peut aussi être utilisé pour simuler l'atténuation de la lumière sous l'eau.

Placage de texture

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



Principe



- Textures = images avec une texture 2D, un point correspond à deux coordonnées (s,t).
- Le coin inférieur gauche étant à (0,0) et le coin droit supérieur (1,1)

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

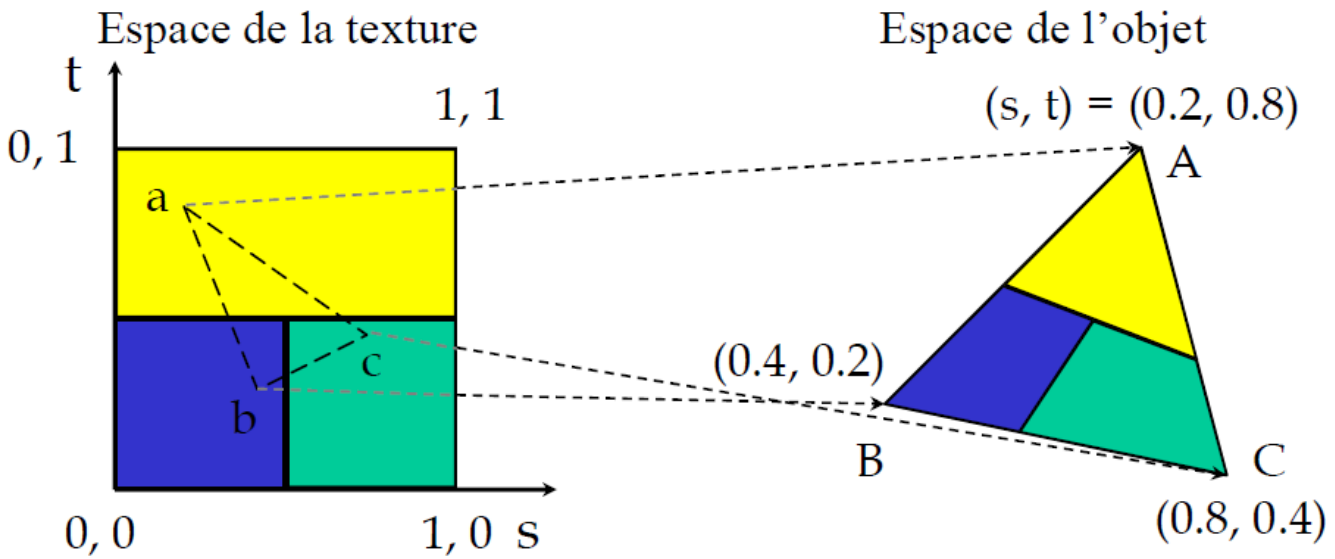
```
glEnable(GL_TEXTURE_2D);
```

  - 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 :

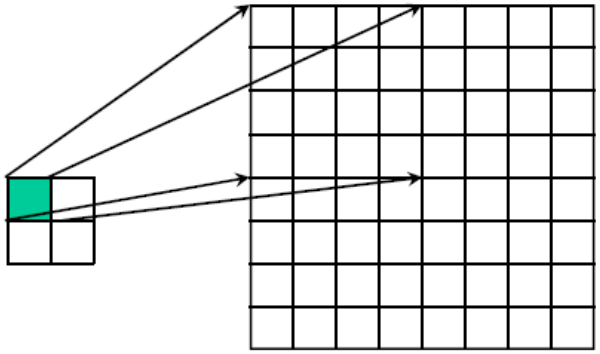
```
glBindTexture(GL_TEXTURE_2D, texture);

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

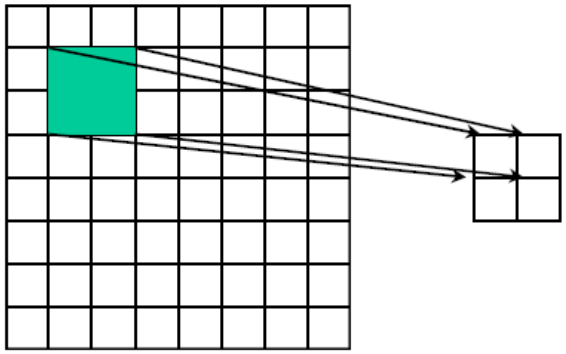
Textures en OpenGL

- Modes de filtrage
  - réduction, agrandissement : les textures et les objets n'ont pas souvent la même taille. OpenGL défini des filtres indiquant comment agrandir ou réduire



Agrandissement

GL_TEXTURE_MAG_FILTER



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
```

Textures en OpenGL

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.

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

Textures en OpenGL

- Modes de bouclage (Wrap) :
  - ce mode indique ce qui doit se produire si une coordonnée de texture sort de l'intervalle [0,1]

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

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 :

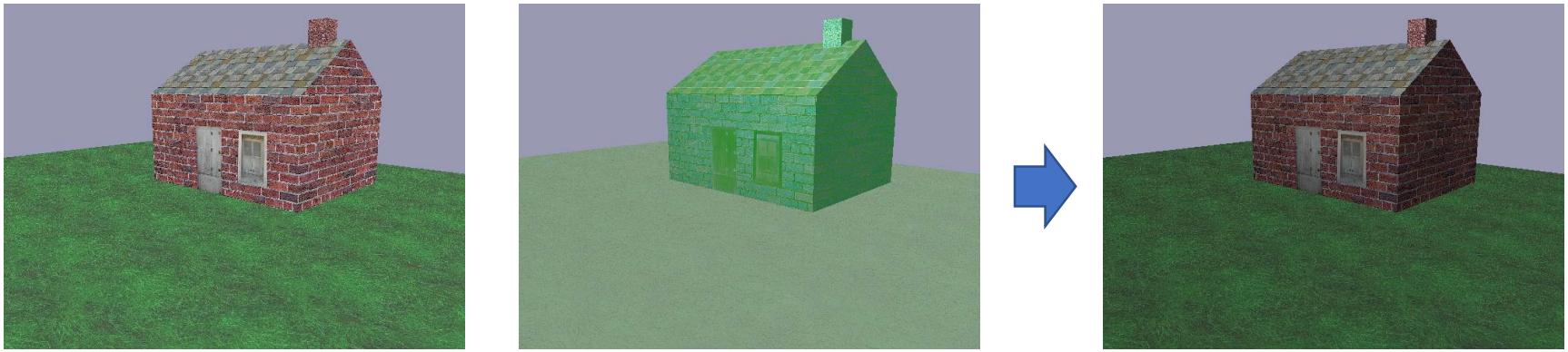
- **GL_DECAL** : remplace la couleur par le texel.
- **GL_MODULATE** : multiplie le texel par la couleur.
- **GL_BLEND** : mélange le texel, la couleur et **env_color**.

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 : tableau de 4 float représentant la couleur de base



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

Exemple complet

```
// Déclarations de variables

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

// Fin des déclarations de variables

// 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;
}
```

Exemple complet

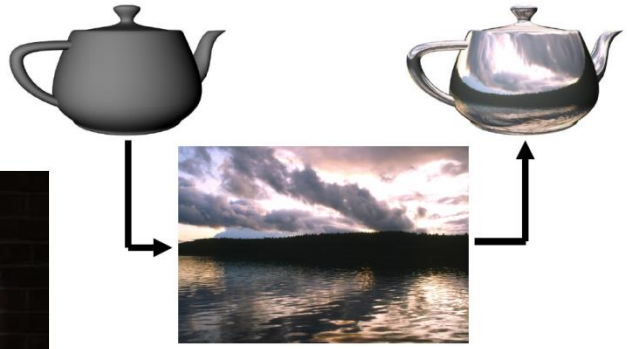
```
// Déclarations de variables

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);

glTexEnvf(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_MODULATE);

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

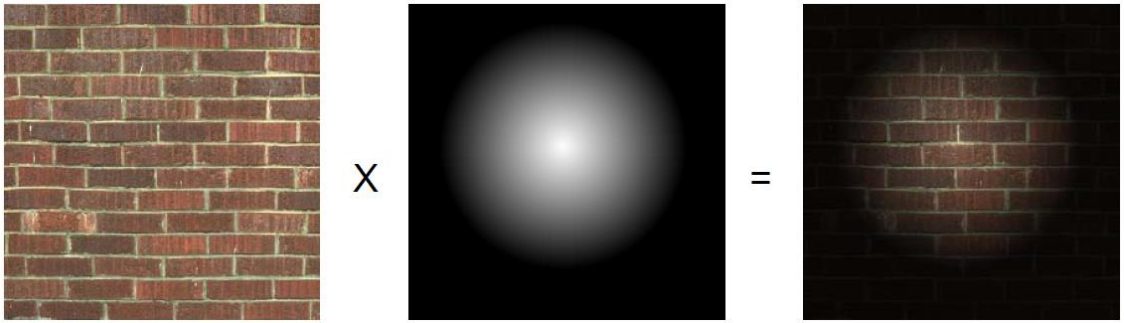
Exemple complet

```
glTexParameteri(GL_TEXTURE_2D,
// 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
```

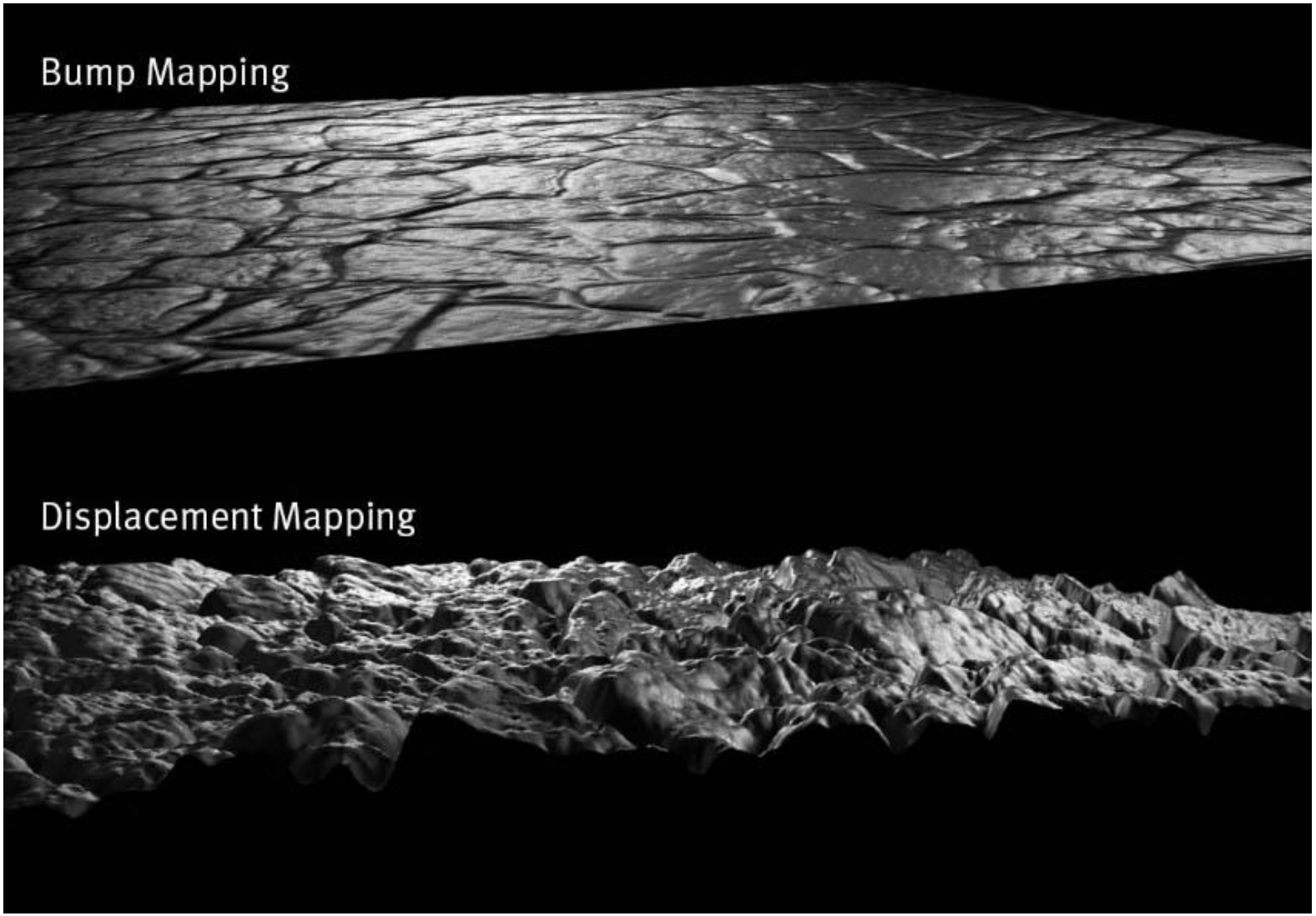
Fonctions avancées

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

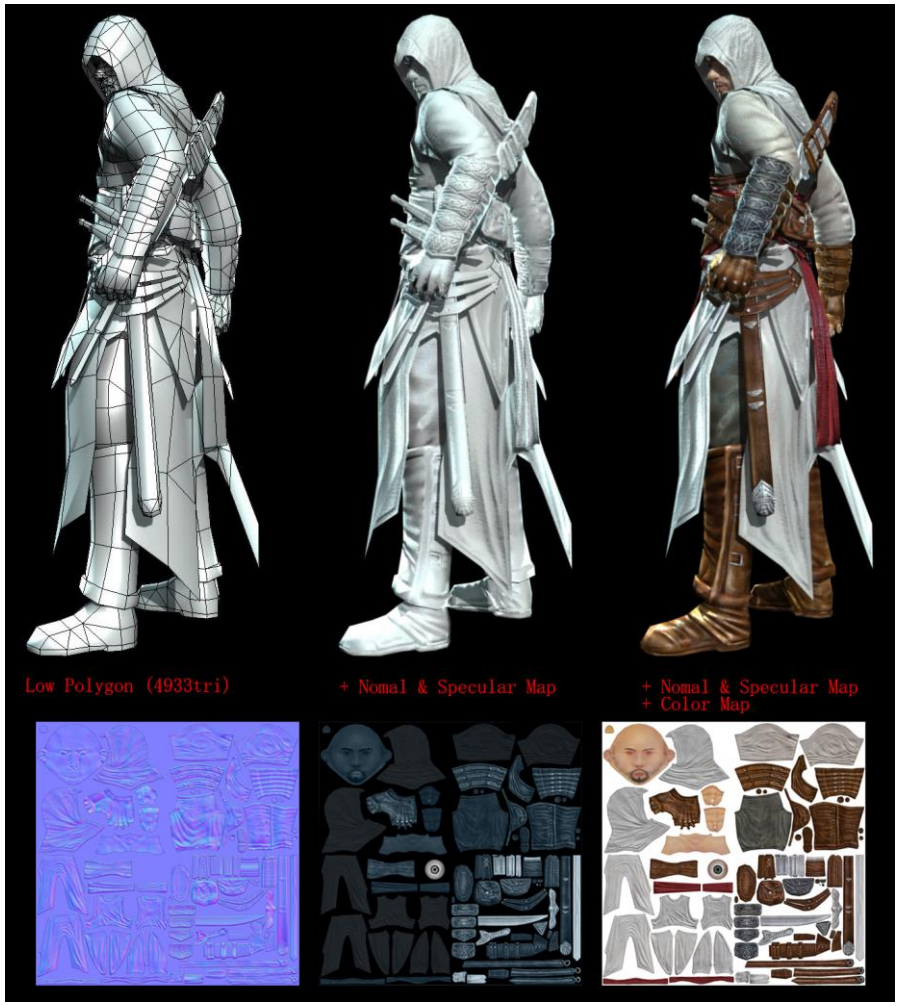




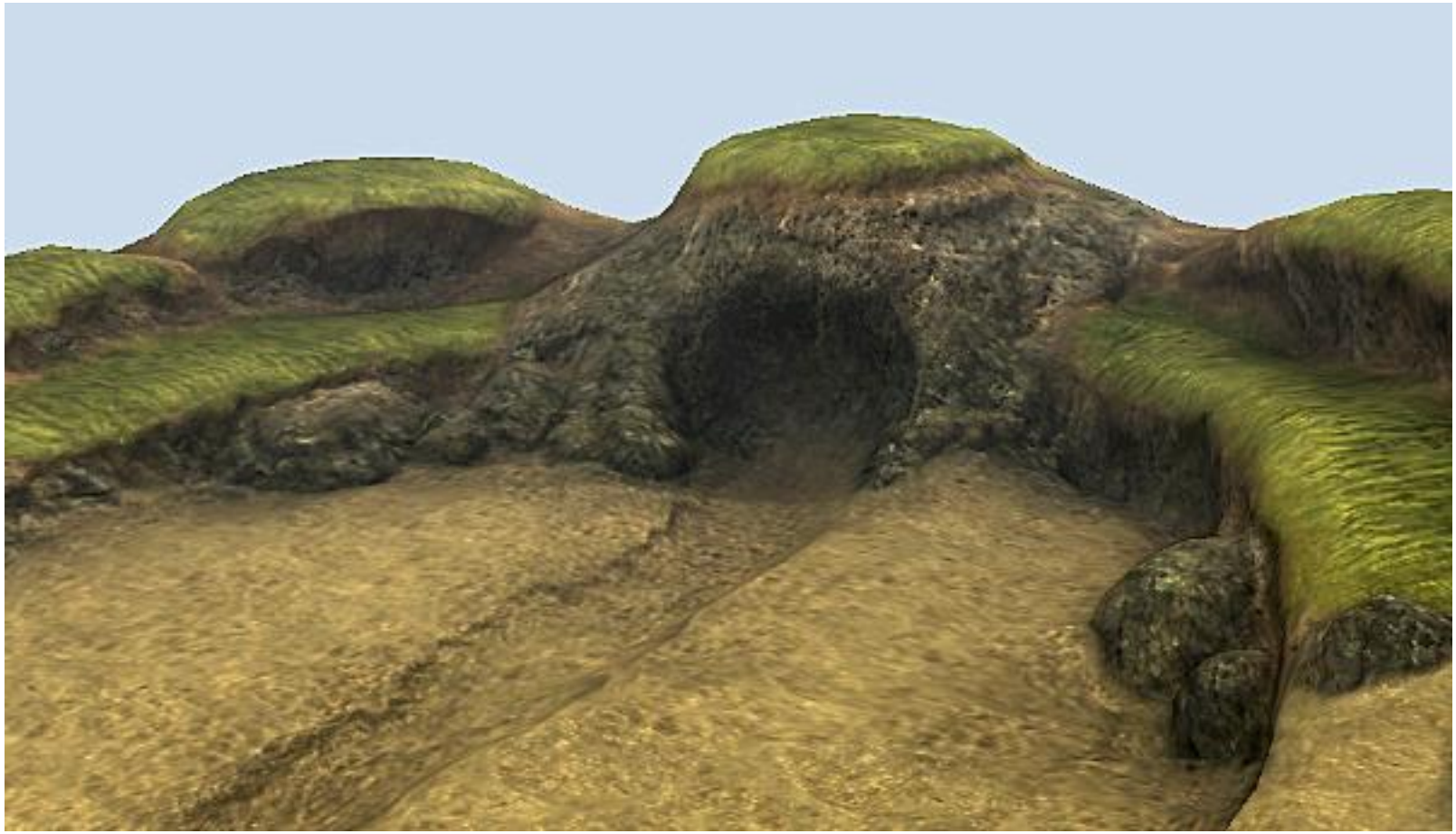
Fonctions avancées



Normal mapping



Multitexturing



Environnement mapping

Conclusion

Afin d'améliorer le rendu d'image 3D, il faut utiliser

- les lumières :
  - définies par de 3 composantes : ambiante, spéculaire et diffus
  - permettant de calculer les couleurs de chaque pixel

Et/ou

- les textures :
  - plaquer une image 2D sur un objet 3D

Sources

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