VII.
Rendu réaliste

1. Modèle physique d'éclairage
2. Eclairage sous OpenGL
3. Rendu lisse/facettes
4. Brouillard
5. Texture

Préliminaire :

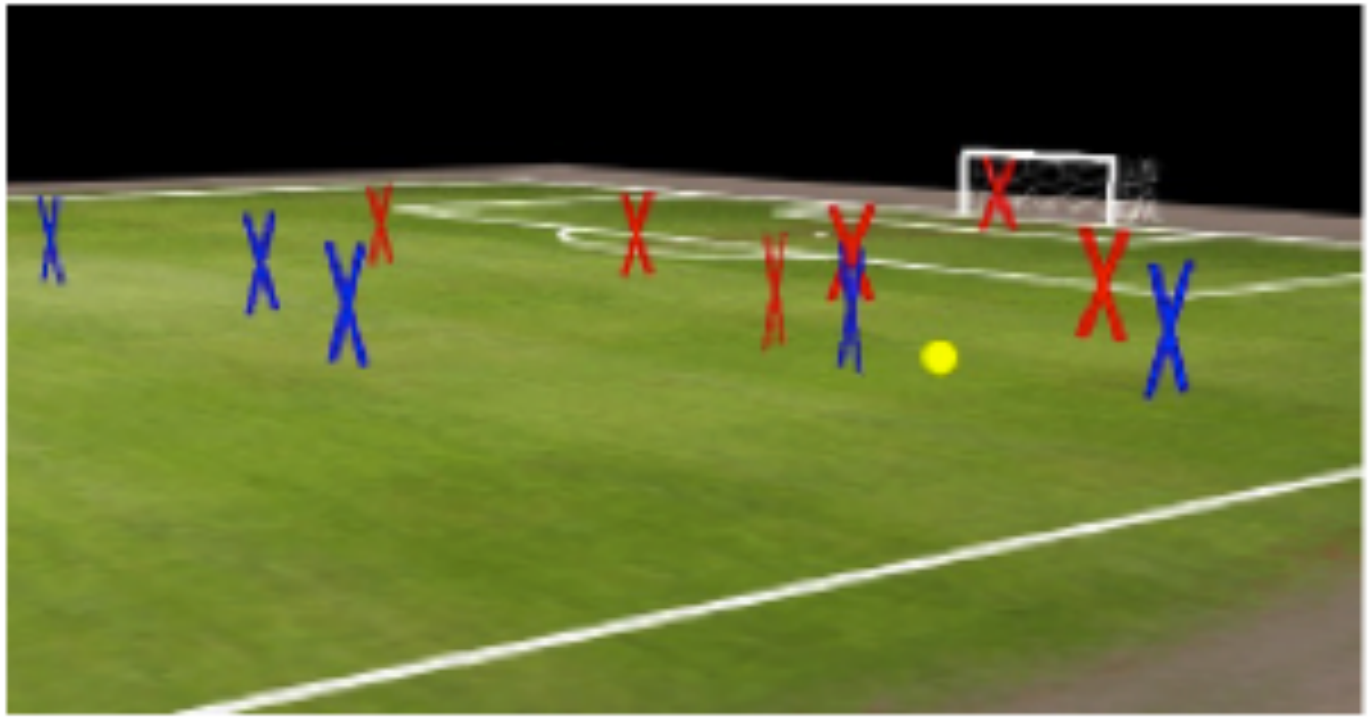
3D ->2D : perte d'information.

Utiliser l'inconscient pour restituer l'information perdue:

- vue en perspective,
- **éclairage des surfaces,**
- **effet de brouillard,**
- *flou dans le mouvement (non abordé dans ce cours)*
- ...

Quelques remarques sur la communication H/M :

- une image simplifiée peut être plus réussie qu'une « photographie »
- la réalité peut être intentionnellement altérée, voire faussée, pour améliorer le transfert d'information



1. Modèle physique d'éclairage

- lumière visible : entre 400 et 700nm.
- composée de fréquences élémentaires (séparable par un prisme)
- issue de :
  - sources actives (lampe, soleil, …),
  - sources passives qui restitue une partie de la lumière reçue.

La perception de la « couleur » d'un objet dépend :

- de la distribution des longueurs d'onde de la source lumineuse,
- des propriétés physiques du matériau
- de l'état de surface
- de la configuration géométrique sources-objet-observateur

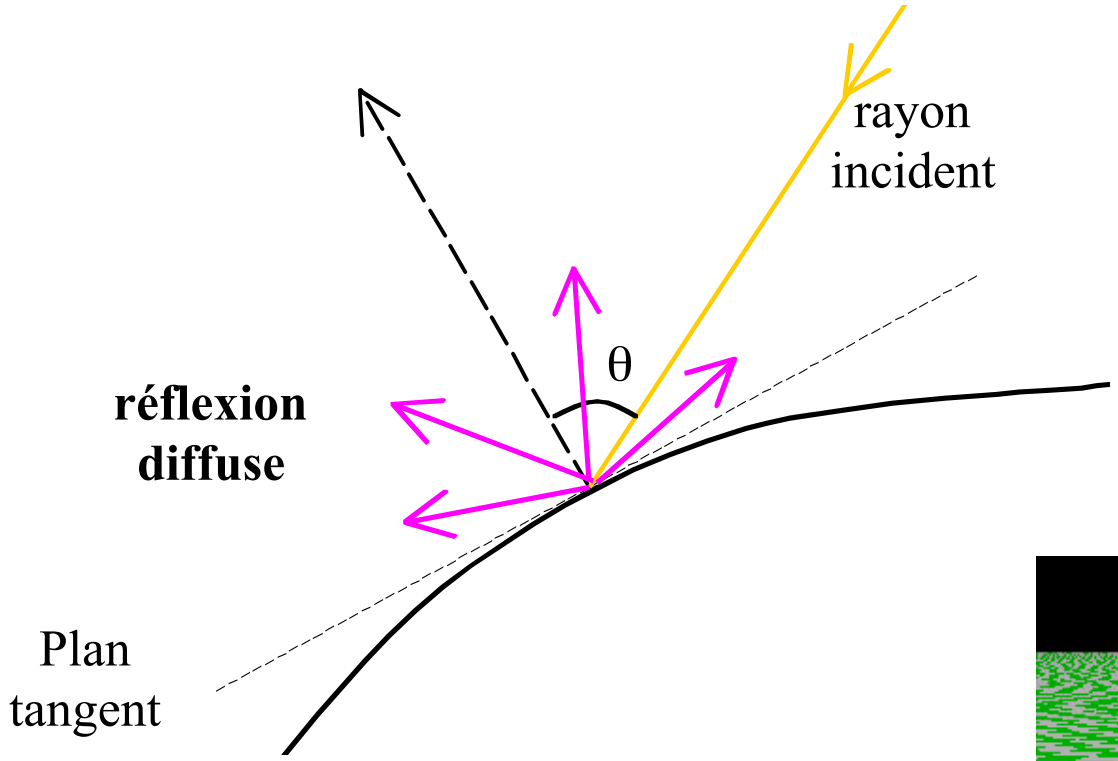
Dans une scène éclairée, on peut distinguer 3 "types" de réflexion :

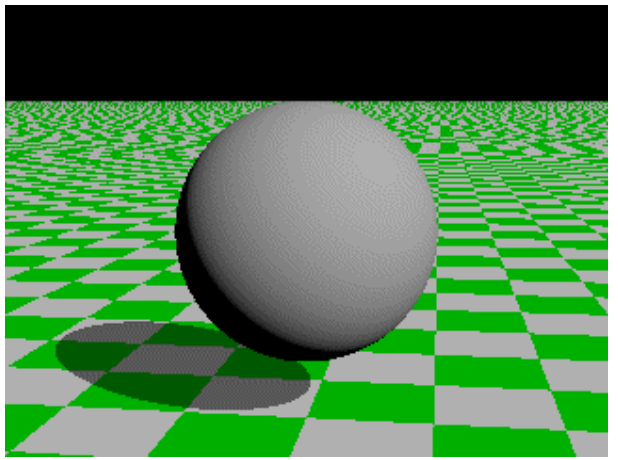
- Lumière **diffuse**
- Lumière **spéculaire**
- Lumière **ambiante**

La lumière reçue par un observateur peut-être vue comme une composition de ces trois types de lumières

Lumière diffuse :

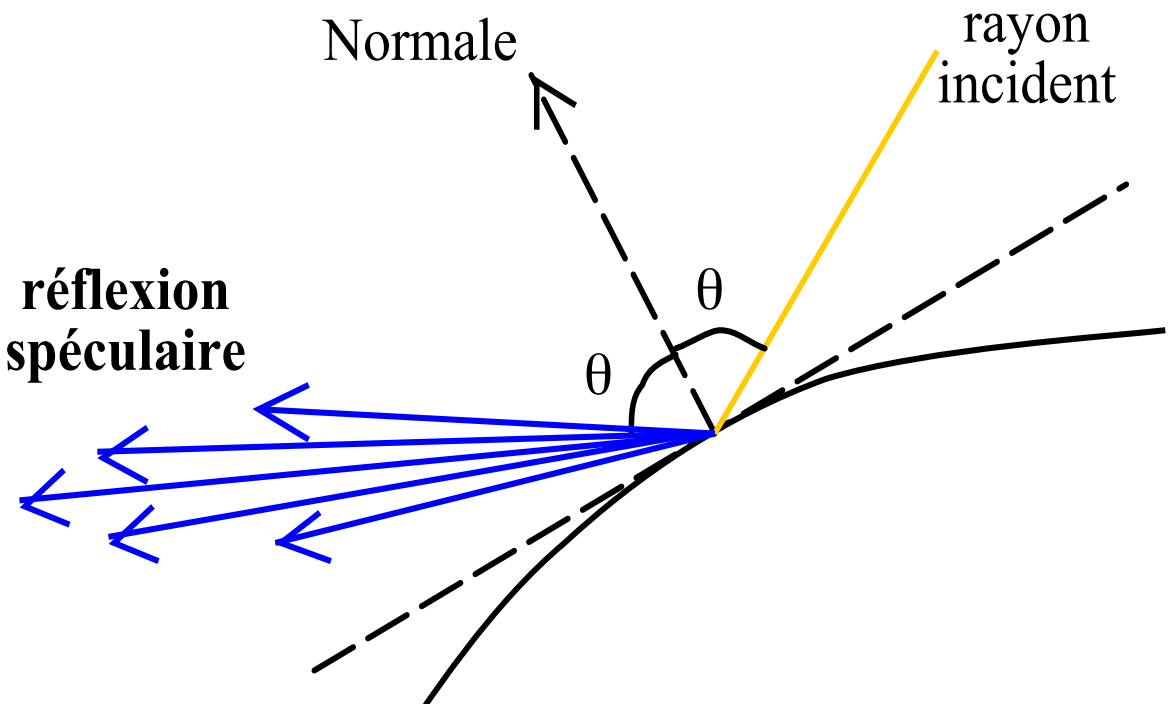
Une partie du rayon incident est réfléchi dans toutes les directions

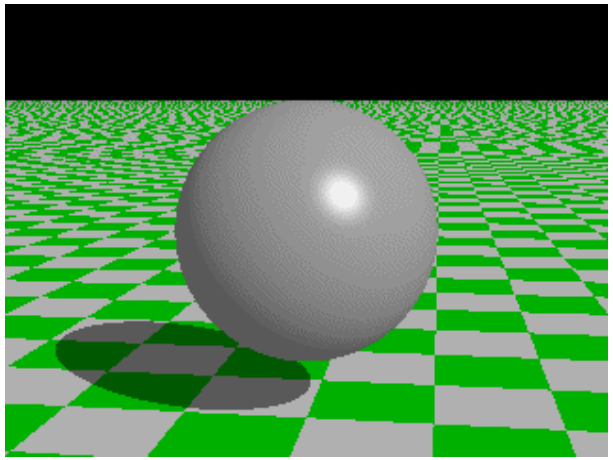


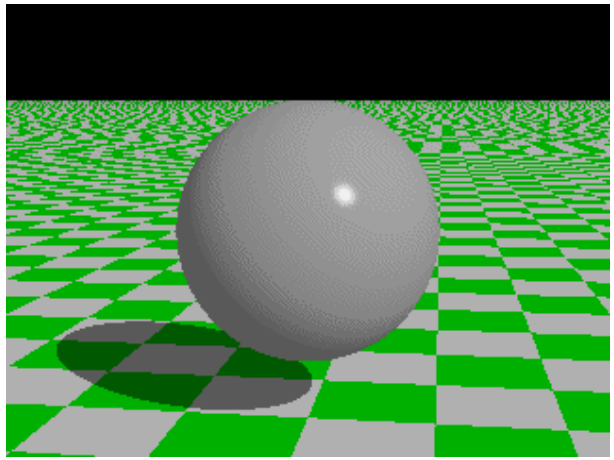


Lumière spéculaire :

Une partie du rayon incident est réfléchi suivant la loi de la normale dans un cône plus ou moins étroit (reflet plus ou moins large)

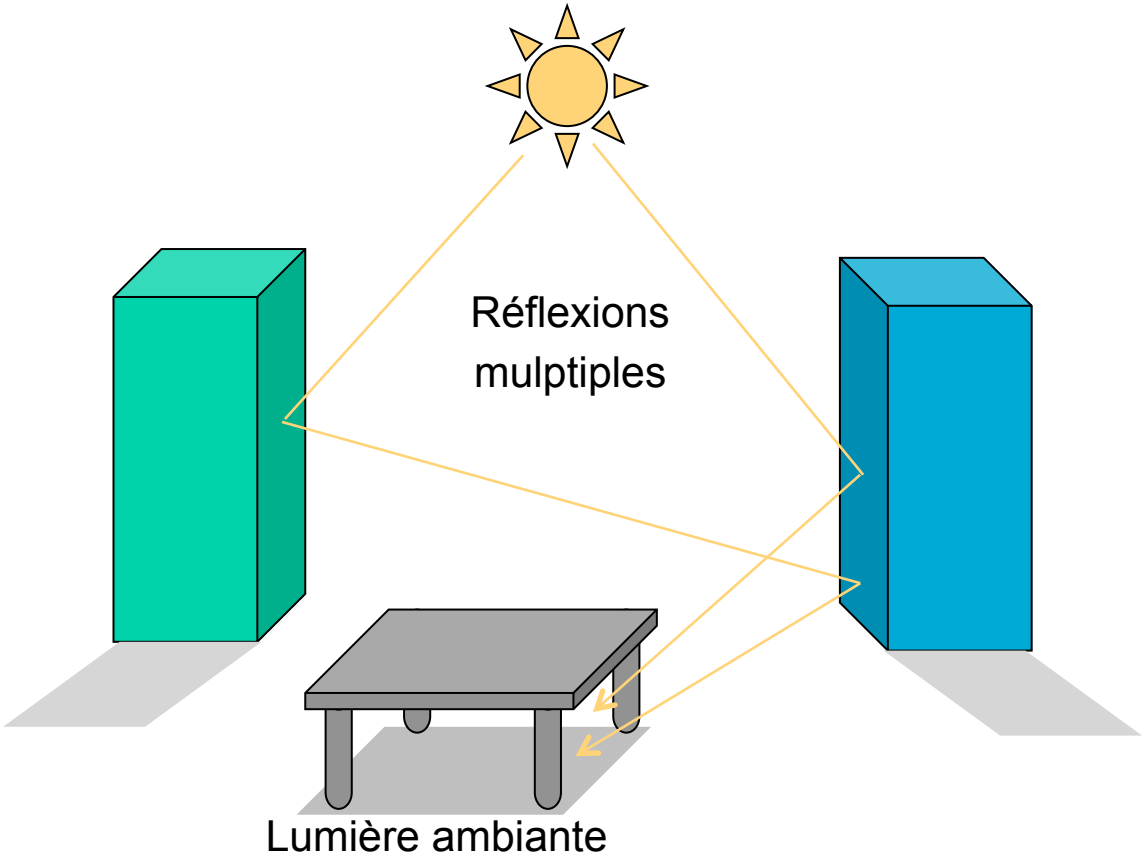




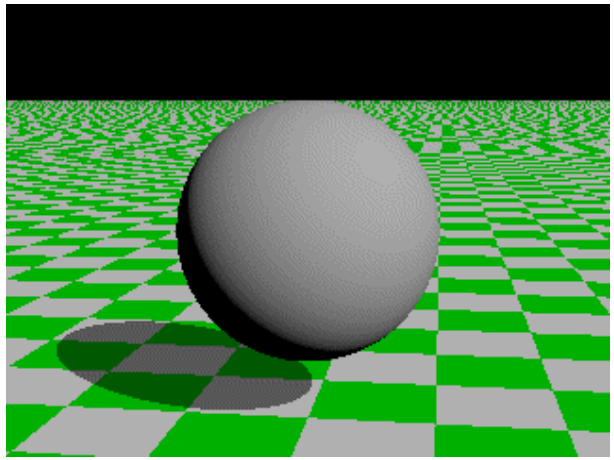


Lumière ambiante :

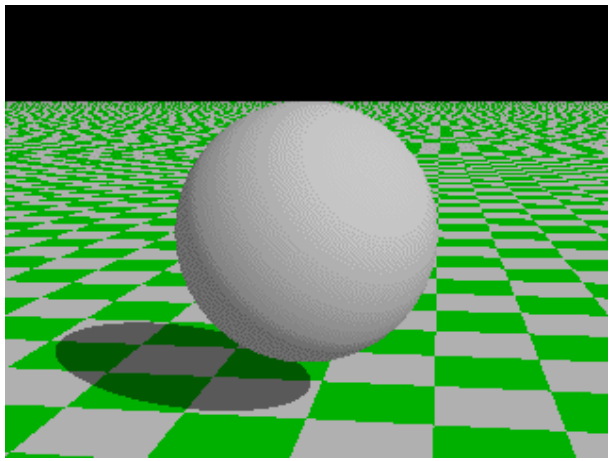
Elle arrive de toutes les directions suite à des réflexions multiples dans une scène.
Elle définie quantité de lumière reçue dans les zones non éclairée en direct



Sans lumière ambiante



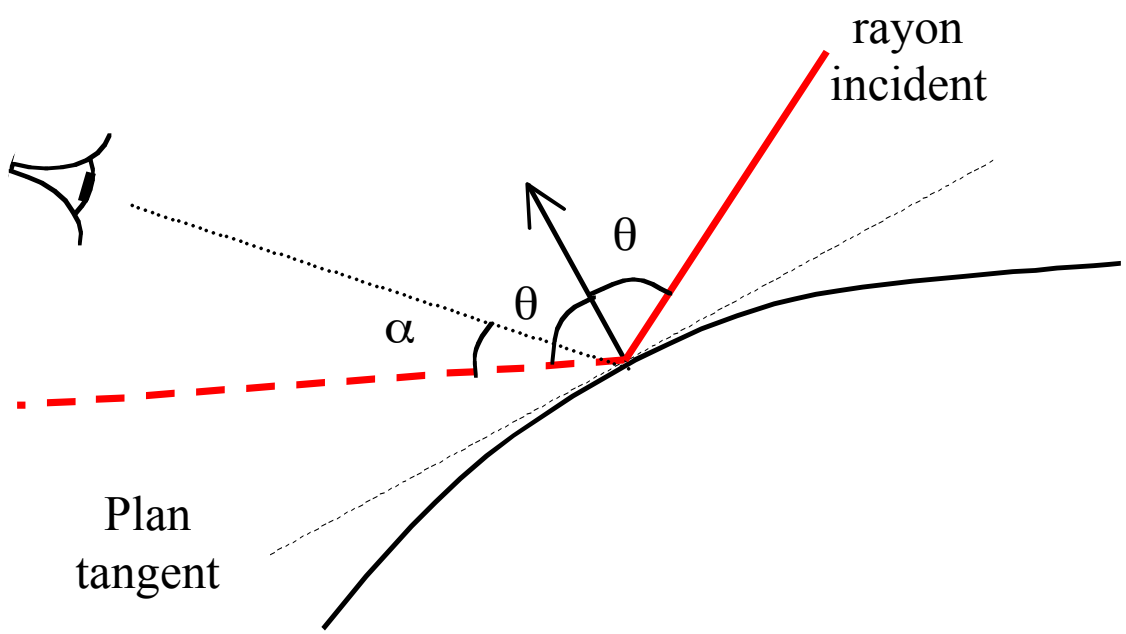
avec lumière ambiante



Modèle physique simplifié d'éclairage sous openGL :

- intensité ambiante (L_a)
- réflexion diffuse
- réflexion spéculaire

$$L = L_a + k_d \cos\theta + k_s \cos^n \alpha$$



Modèles d'illumination :

Gouraud

- calcul du rendu de chaque sommet
- interpolation du rendu des pixels pour le remplissage

➔ pauvre mais peu coûteux

Phong

- interpolation des normales pour chaque point projeté sur un pixel
- calcul du rendu de ces points

➔ meilleur rendu mais plus coûteux

Lancer de Rayon, radiosité

➔ produit les ombres

➔ très réaliste mais très coûteux

…

2. Scène éclairée sous OpenGL *(Gouraud)*

Il faut :

1 - Définir l'éclairage dans la scène :

- activer le mode éclairage
- définir les propriétés des « lumières » (au plus 8)
- positionner les lumières
- allumer/éteindre les lumières

2 - Définir les propriétés optiques de l'objet :

- rendu lisse où à facettes
- définir les faces éclairées des primitives (surface ou volume)
- définir le matériau (propriétés optiques)

3 - Définir une normale avant le tracé d'une facette

2.1 Eclairage de la scène :

On passe du modèle couleur « simple » (*glColor3f(r, v, b)*)

au modèle d'éclairage avec :

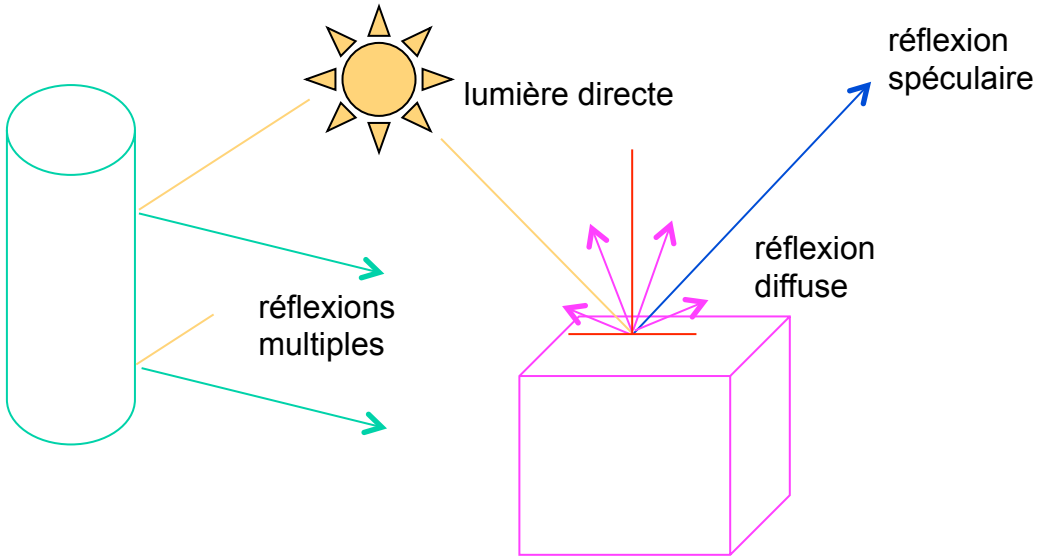
```
glEnable(GL_LIGHTING)
glDisable(GL_LIGHTING)
```

Jusqu'à 8 sources lumineuses :

- noms prédéfinis : **GL_LIGHT0, ..., GL_LIGHT7**
- les effets lumineux de plusieurs sources s'additionnent.
- pour allumer/éteindre une lumière :

```
glEnable(GL_LIGHT0)
glDisable(GL_LIGHT0)
```

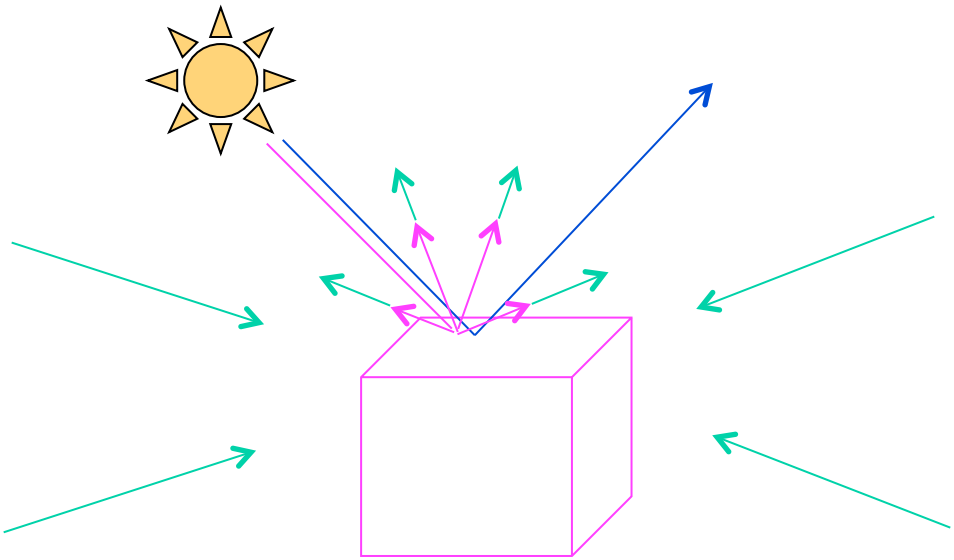
Dans la vrai vie :



Sous OpenGL :

Séparation ambiant - diffus - spéculaire

- pour chaque source
- et sur les matériaux



Caractéristiques d'une source lumineuse :

Décomposition de la source : ambiant - diffus - spéculaire

`glLightfv(`*`n°lumière,attribut,vecteur de float`*`)`

Définir par exemple :

```
GLfloat position [4] = {0.0, 10.0, 10.0, 1.0};
GLfloat Lambiant [4] = {0.2, 0.2, 0.2, 1.0};
GLfloat Lblanche [4] = {1.0, 1.0, 1.0, 1.0};
```

Éxécuter au moins une fois :

```
glLightfv (GL_LIGHT0, GL_AMBIENT, Lambiant);
glLightfv (GL_LIGHT0, GL_DIFFUSE, Lblanche);
glLightfv (GL_LIGHT0, GL_SPECULAR, Lblanche);

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

Remarques :

Une source lumineuse est assimilée à un objet

➔ elle subit les transformations géométriques (**GL_MODELVIEW**).

➔ on peut l'attacher à la scène :

```
glLoadIdentity();
gluLookAt(…);
glLightfv (GL_LIGHT0, GL_POSITION, position);
```

Séparation à la source de la lumière diffuse et spéculaire

➔ Exemple peu « physique » :

- une source qui ne génère pas de reflet,
- une source qui ne génère que des reflets,
- une source rouge qui génère des reflets bleus,
- une source qui ne contrôle que la lumière ambiante.

2.2 Propriétés optiques de l'objet

Orientation :

- face avant : sens trigonométrique des sommets
- plan tangent : la normale au sommet

Comme en mode `glColor`, les calculs de rendu s'effectuent soit :

- sur les faces avants **`GL_FRONT`**
- sur les faces arrières **`GL_BACK`**
- sur les faces avants et arrières **`GL_FRONT_AND_BACK`**

Le matériau :

On n'utilise plus `glColor3f(r, v, b )`

Un objet restitue une partie de la lumière reçue en fonction :

- du matériau (cuivre, argent, …)
- de l'état de surface (brillant, mat, …)

Propriétés optiques du matériau : séparation des 3 types de lumière

- **ambiant**
- **diffusion**
- **réflexion spéculaire** avec son cône de réflexion
- ***émission*** *(objet "brillant")*

Il faut définir le matériau avant le tracé des primitives :

```
glMaterialfv(face, attribut, vecteur de float)
```

Pour un matériau, on déclare par exemple :

```
GLfloat Lnoire [4] = {0.0, 0.0, 0.0, 1.0};
GLfloat matAmbiant [4] = {0.07, 0.04, 0.02, 1.0};
GLfloat matDiffuse [4] = {0.71, 0.42, 0.18, 1.0};
GLfloat matSpecular [4] = {0.39, 0.27, 0.16, 1.0};
GLfloat matShininess [1] = {25.0};
```

Ces propriétés sont attribuées aux faces avant/arrière des polygones avec

GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

```
glMaterialfv(GL_FRONT, GL_EMISSION, Lnoire);
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbiant);
```

luminosité uniforme

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse);
```

dépendant de la position relative lumière/ surface

```
glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);
```

dépendant des positions relatives lumière/surface/observateur

```
glMaterialfv(GL_FRONT, GL_SHININESS,matShininess);
```

taille du reflet (≈ état de surface du matériau)

Quelques matériaux

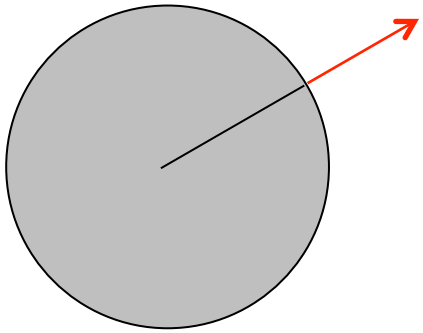
Matériau	ambiante	diffuse	spéculaire	Exposant : α
Plastique noir	0.0 0.0 0.0	0.01 0.01 0.01	0.5 0.5 0.5	32
Cuivre	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974
Bronze	0.2125 0.1275 0.054	0.714 0.4284 0.18144	0.393548 0.271906 0.166721	25.6
Chrome	0.25 0.25 0.25	0.4 0.4 0.4	0.774597 0.774597 0.774597	76.8
Cuivre rouge	0.19125 0.0735 0.0225	0.7038 0.27048 0.0828	0.256777 0.137622 0.086014	12.8
Or	0.24725 0.1995 0.0745	0.75164 0.60648 0.22648	0.628281 0.555802 0.366065	51.2
Étain	0.10588 0.058824 0.113725	0.427451 0.470588 0.541176	0.3333 0.3333 0.521569	9.84615
Argent	0.19225 0.19225 0.19225	0.50754 0.50754 0.50754	0.508273 0.508273 0.508273	51.2
Argent brillant	0.23125 0.23125 0.23125	0.2775 0.2775 0.2775	0.773911 0.773911 0.773911	89.6

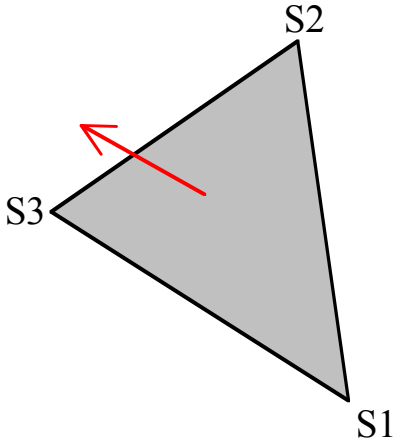
2.3 La normale

Le rendu d'une surface éclairée dépend de sa normale

La normale peut être obtenue :

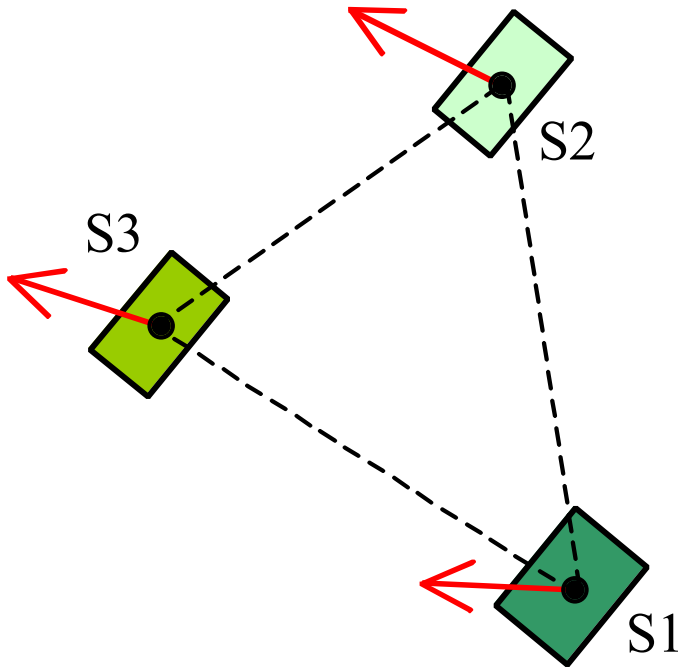
- par connaissances géométriques (sphère, cylindre, cube, …)
- par calcul : $u \wedge v = [(y_u z_v - z_u y_v),\ -(x_u z_v - z_u x_v),\ (x_u y_v - y_u x_v)]$





En pratique sous openGL :

- un sommet est considéré comme un petit élément de surface
- la normale est définie en chaque sommet => calcul du rendu
- le rendu d'une primitive est extrapolé du rendu des sommets qui la définissent



La normale est spécifiée avant le tracé des sommets avec :

`glNormal3f(x, y, z)` ou encore **`glNormal3fv(N)`**

Tous les sommets qui suivent seront affectés de cette normale

```
glBegin(GL_TRIANGLES);
  glNormal3f(0., 0., 1.) ;
  glVertex3f(0., 0., 0.);
  glVertex3f(5., 0., 0.);
  glVertex3f(2.5, 5., 0.);
glEnd();
```

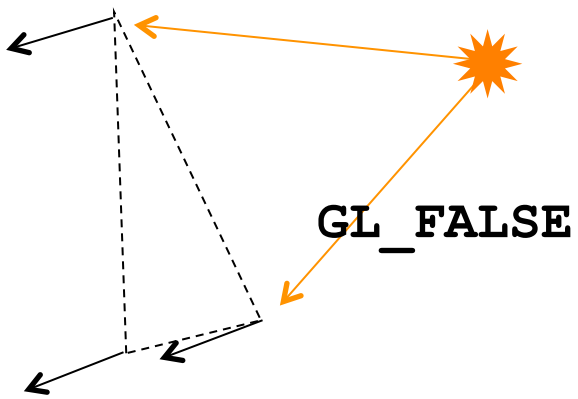
Remarque : openGL nécessite des normales **normées**

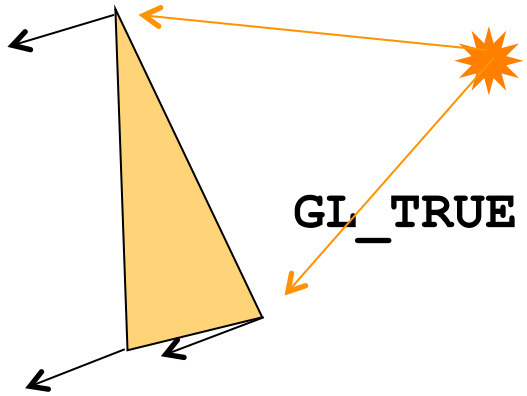
- peut être géré par OpenGL : `glEnable(GL_NORMALIZE)`
- indispensable avec `glScalef()`

Pour rendre visible les deux côtés d'une face, il faut activer :

`glLightModeli(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE) ;`

Cette option sera désactivée pour le tracé d'objets "pleins »
(gain de temps)





2.4 Résumé

Au moins une fois

```
glEnable(GL_LIGHTING);
```
activation du mode éclairage

```
glEnable(GL_LIGHT0);
```
allumage de la lumière 0

Caractérisques de la lumière 0 :

```
glLightfv (GL_LIGHT0, GL_AMBIENT, Lambiant);
glLightfv (GL_LIGHT0, GL_DIFFUSE, Lblanche);
glLightfv (GL_LIGHT0, GL_SPECULAR, Lblanche);
```

Facultatif

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE) ;
```
```
glEnable(GL_NORMALIZE);
```
normalisation auto. fortement conseillée

Désignation de la surface non visible et activation :

```
glCullFace(GL_BACK);
glEnable(GL_CULL_FACE);
```
ou
```
glDisable(GL_CULL_FACE);
```

Dans la construction de la scène,

- **Positionner la lumière au moins une fois :**

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

- *au début si c'est une position absolue*
- *juste après* `gluLookAt(...)` *si elle est attachée à la scène*

- **Définir le matériau avant le tracé d'un objet**

```
glMaterialfv(GL_FRONT, GL_EMISSION, Lnoire);
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbiant);
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);
glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
glMaterialfv(GL_FRONT, GL_SHININESS, matShin);
```

Au moins une normale pour chaque facette

```
glBegin(GL_TRIANGLES);
  glNormal3fv(N1) ; /* normale pour (S11,S12,S13) */
  glVertex3fv(S11);
  glVertex3fv(S12);
  glVertex3fv(S13);

  glNormal3fv(N2) ; /* normale pour (S21,S22,S23) */
  glVertex3fv(S21);
  glVertex3fv(S22);
  glVertex3fv(S23);
  ...
glEnd();
```

3. Rendu lisse/facettes

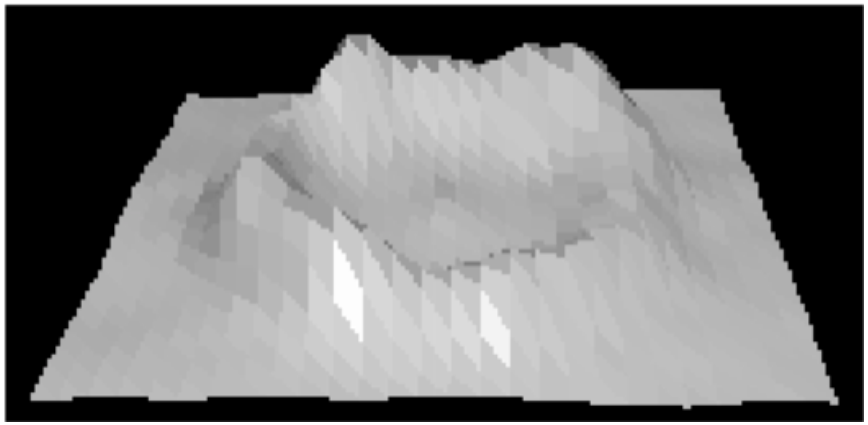
Surface gauche : approximée par des données polygonales

➔ rendu à facettes

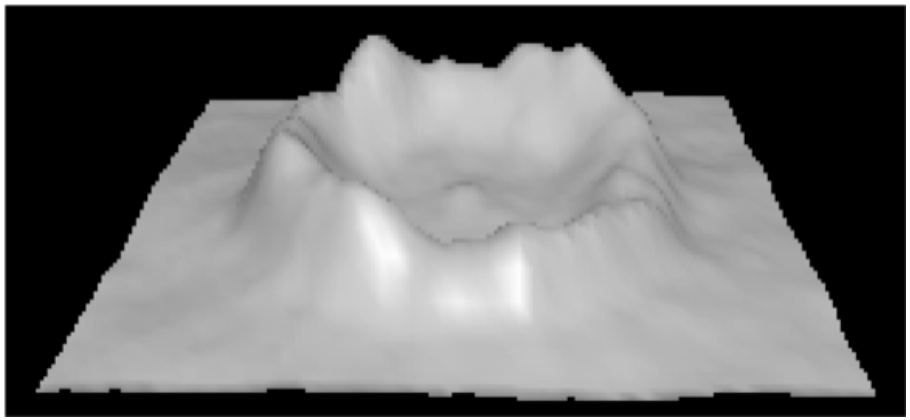
OpenGL permet de donner un aspect « lisse »

(extrapolation linéaire du rendu entre les sommets)

`glShadeModel(GL_FLAT)`



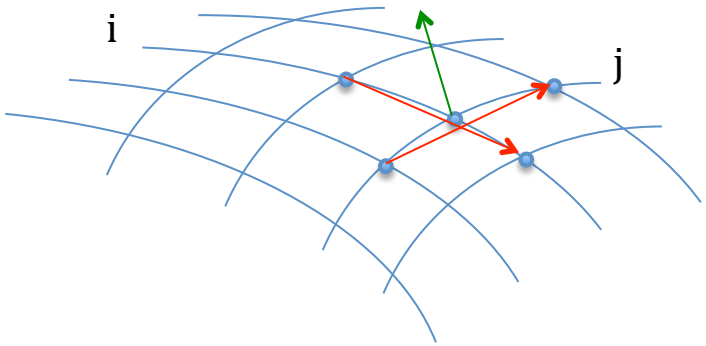
`glShadeModel(GL_SMOOTH)`



Méthode pour tracer une surface gauche

Surface gauche : maillage de points

- sommets stockés dans une matrice S de taille n×m
- normales définies en chaque sommet
- précalculées et stockées dans une matrice N n×m



Approximation de la normale en un sommet pour une surface dérivable

Tracé d'une surface lisse :

```
glShadeModel(GL_SMOOTH);
glBegin(GL_TRIANGLES);
  for (i=0 ; i<n-1 ; i++)
   for (j=0 ; j<m-1 ; j++)
   {/* dessin d'une maille (i,j) */
    glNormal3fv(N[i][j]);     glVertex3fv(S[i][j]);
    glNormal3fv(N[i][j+1]);   glVertex3fv(S[i][j+1]);
    glNormal3fv(N[i+1][j+1]); glVertex3fv(S[i+1][j+1]);

    glNormal3fv(N[i][j]);     glVertex3fv(S[i][j]);
    glNormal3fv(N[i+1][j+1]); glVertex3fv(S[i+1][j+1]);
    glNormal3fv(N[i+1][j]);   glVertex3fv(S[i+1][j]);
   }
glEnd();
```

4. Le brouillard

1. définir la couleur d'absorption
2. définir la fonction d'absorption (distance au point de vue)
3. activer/désactiver le brouillard pour le tracé

Activation : **`glEnable(GL_FOG)`**
Désactivation : **`glDisable(GL_FOG)`**

Caractéristiques du brouillard : **`glFogf(`***`type, paramètre`***`)`**

- couleur d'absorption : *Sirocco*

```
GLfloat fogColor[4] = {0.5, 0.5, 0.3, 1.};
glFogfv(GL_FOG_COLOR, fogColor) ;
```

on utilisera généralement la même couleur pour le fond de la scène :

```
glClearColor(0.5, 0.5, 0.3, 1. ) ;
```

- profil de la fonction brouillard

```
glFogi(GL_FOG_MODE, GL_LIN) ;
```

- extrémités de la fonction brouillard

```
glFogf(GL_FOG_START, 1.) ;
glFogf(GL_FOG_END, 5.) ;
```

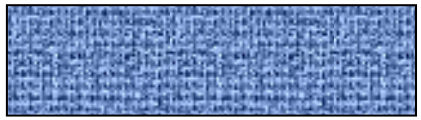
- coefficient de « cintrage » pour **GL_EXP2**

```
glFogf(GL_FOG_DENSITY, 0.35) ;
```

5. Un aperçu sur le placage de texture

5.1 Principe

Certains objets ont des aspects hétérogènes
qui sont difficiles à reproduire sur une surface :

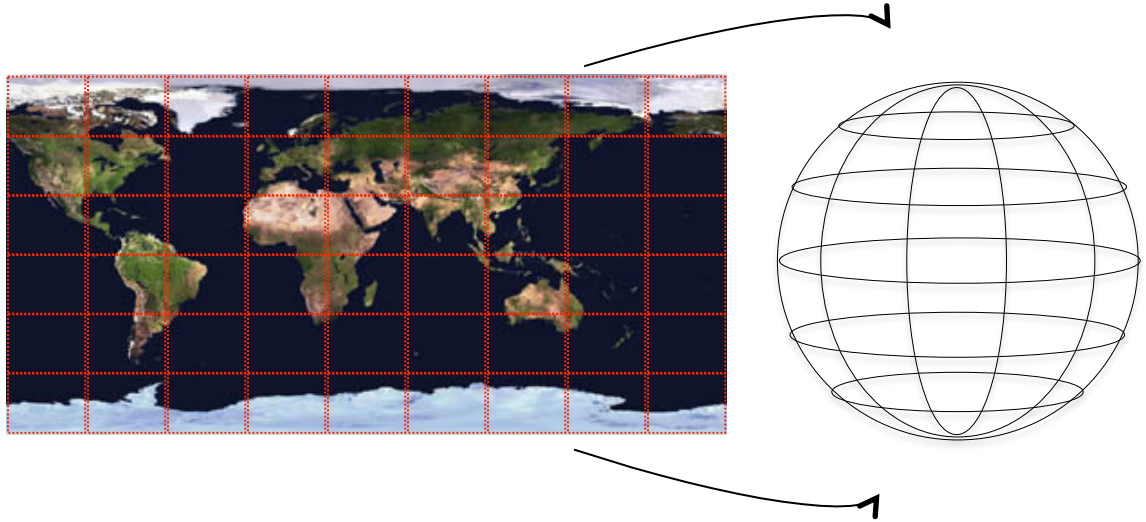
- bois 
- marbre 
- tissu 
- façade d'immeuble
- …

Principe du placage de texture :

- disposer d'une image 2D représentant la texture de cet objet
- sélectionner un polygone à N sommets dans cette image
- le mettre en correspondance avec N sommets de l'objet 3D

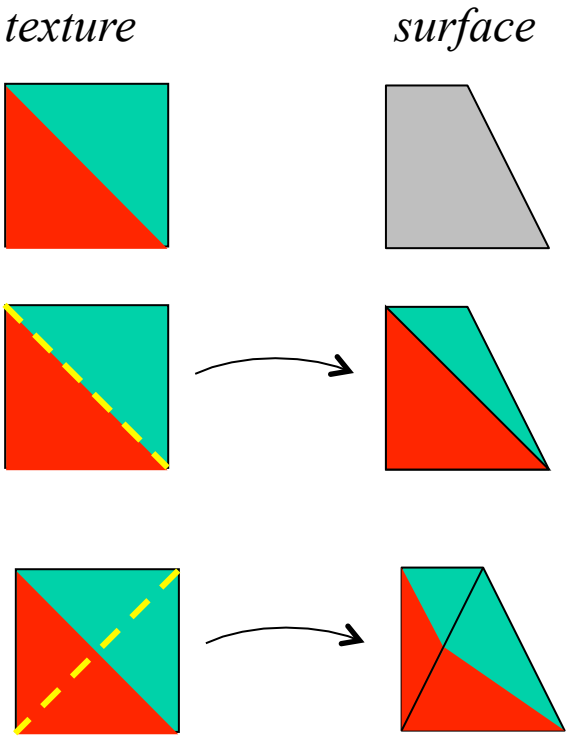
Remarque :

- la mise en correspondance peut nécessiter une déformation
- c'est un des aspects délicats à gérer pour obtenir le résultat attendu



En particulier :

- openGL fait une décomposition arbitraire des polygones en triangles
- il y a plusieurs décompositions possibles...
- qui influent sur le résultat du placage de texture



Placage de texture

sur deux décompositions

d'un même quadrilatère

➔ *On préfèrera une description par triangles*

5.2 Lire une image

Une texture peut être :

- générée par calcul
- Issue d'une image existante (photo, …)

Format d'un fichier ppm :

- entête au format texte
- image stockée ligne par ligne
- pixel RGB représenté par 3 octets

```
P6
# commentaires
hauteur largeur
max_value
rgbrgbrgb...
```

Structure de données :

```
GLubyte *image;  /* unsigned char */
image = malloc(sizeof(char)*3*hauteur*largeur) ;
```

5.3 Définir une texture 2D sous openGL

(*) 1 fois

(**) à chaque changement

Réserver plusieurs textures (2 dans cet exemple) :

(*) `GLint textures[2];`

(*) `glGenTextures(2,textures);`

Sélectionner une texture (en vue d'un chargement ou d'une utilisation)

(**) `glBindTexture(GL_TEXTURE_2D,textures[0]);`

Charger une texture (mémoire centrale -> carte graphique) :

(**) `gluBuild2DMipmaps(GL_TEXTURE_2D, 3, largeur, hauteur,`
`GL_RGB, GL_UNSIGNED_BYTE,`*`image`*`);`

Précaution (transfert par défaut = paquets de mots machine) :

```
glPixelStorei(GL_UNPACK_ALIGNMENT,1);//paquets d'1 octet
```

Coordonnées texture (image) :

5.4 Tracé avec placage

Activer/désactiver le mode texture avant le tracé :

```
glEnable(GL_TEXTURE_2D)
glDisable(GL_TEXTURE_2D)
```

```
glBegin(GL_POLYGON);
   glTexCoord2f(0.0, 0.0); glVertex3fv(S1);
   glTexCoord2f(1.0, 0.0); glVertex3fv(S2);
   glTexCoord2f(1.0, 1.0); glVertex3fv(S3);
   glTexCoord2f(0.0, 1.0); glVertex3fv(S4);
glEnd();
```

5.5 Quelques précisions sur le mode de placage

`glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode)`

`GL_REPLACE` : placage brut

`GL_MODULATE` : pondéré par une couleur/matériau éclairé

`GL_BLEND` : mélangé avec une couleur/matériau éclairé

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
glColor3f(1, 0, 0); // les composantes vertes et bleues seront éliminées
glBegin(GL_POLYGON);
      glTexCoord2f(0.0, 0.0); glVertex3fv(S1);
      glTexCoord2f(1.0, 0.0); glVertex3fv(S2);
      glTexCoord2f(1.0, 1.0); glVertex3fv(S3);
      glTexCoord2f(0.0, 1.0); glVertex3fv(S4);
glEnd();
```

Texture éclairée : utilisez un matériau blanc avec l'option `GL_MODULATE`

```
glTexCoord2f(x,y) ; glNormal3fv(N1); glVertex3fv(S1);
```

5.6 Répétition du motif (coordonnées texture hors de [0, 1])

`glTexParameteri(GL_TEXTURE_2D, où, type) ;`

type de répétition :

`GL_CLAMP` : répétition du bord (ligne ou colonne)

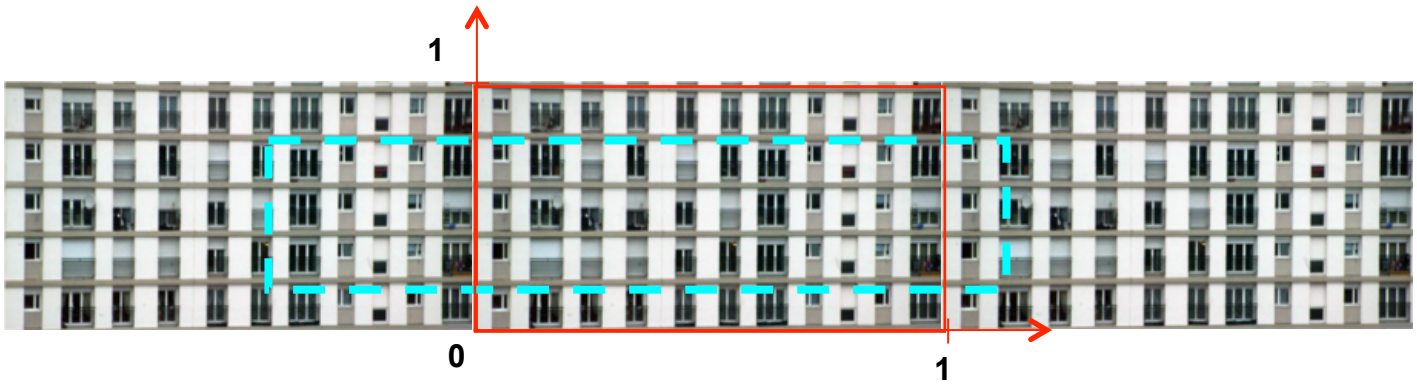
`GL_REPEAT` : reprise du motif (briques…)

où ? :

`GL_TEXTURE_WRAP_T` : dessus/dessous

`GL_TEXTURE_WRAP_S` : gauche/droite

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



5.7 Changer le repère texture

```
glMatrixMode(GL_TEXTURE);
  glLoadIdentity();
  glTranslatef(-0.3, 0, 0);
  glRotatef(45, 0, 0, 1);  /* dans le plan texture */
glMatrixMode(GL_MODELVIEW);
```