Introduction à OpenGL

Option I.I.M.

Julien Tierny
julien.tierny@lifl.fr

15 novembre 2007

Menu

- Cours 1 :
  - Généralités ;
  - Machine à états et rendu projectif ;
- Cours 2 :
  - Modèle d'éclairement ;
  - Placage de texture ;
- Cours 3 :
  - Techniques avancées de rendu projectif.

1. Généralités

OpenQuoi ?

- *GL* : Graphics Library ;
- Interface logicielle (API) pour le rendu 3D interactif (projectif) ;
- Convention d'instructions pour l'utilisation de la carte graphique ;
  - **Standard ouvert !** (aujourd'hui, version 2.1)
- Multi-plateforme (Linux, MacOS, Windows, Wii, DS, PSP, PS3) ;
- Interface entre le programme et le pilote du GPU ;
  - **Bas niveau !**
- **Bibles :**
  - The OpenGL Red Book ;
  - The OpenGL Blue Book ;
  - ...plus Green, Alpha et Orange.

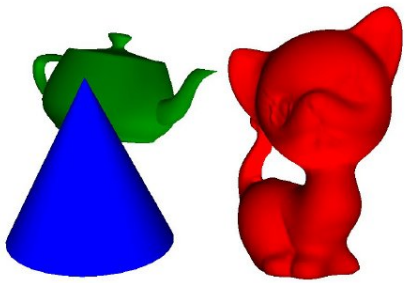
Contexte

- **Historique :**
  - Initialement conçue par Silicon Graphics Inc (IRIS-GL 1990) ;
  - Standardisée par un consortium industriel (OpenGL Architecture Review Board) en 1992 ;
  - Gestion transférée au Khronos Group depuis 2006 ;
- **Alternatives :**
  - Direct3D : Microsoft (Windows, XBox) ;
  - Fahrenheit : Microsoft, SGI, HP (abandonnée en 1999) ;
  - OpenRT (GNU-RT) ;

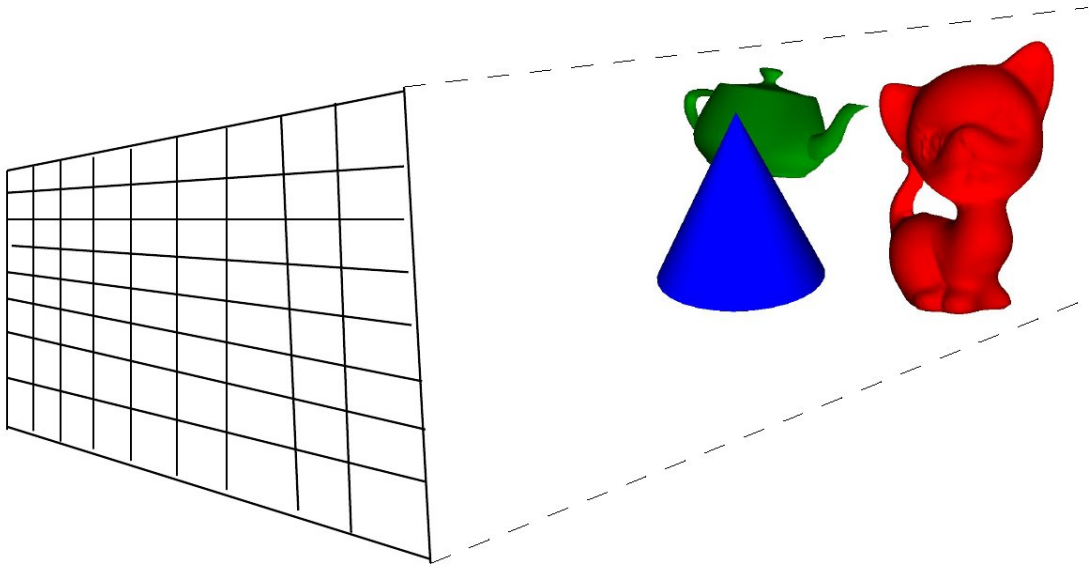
En bref

- **OpenGL, c'est :**
  - Une API en C de 250 instructions ;
- **Ce que fait OpenGL :**
  - Tracé de scènes tridimensionnelles (?) ;
- **Ce que ne fait pas OpenGL :**
  - Fenêtrage et interactions ;
  - Abstraction de scène (*"3D programming for humans"*) ;
- **Boîtes à outils :**
  - Fenêtrage : GLU, GLUT, **SDL** ;
  - Abstraction de scène : OpenInventor (SGI, Coin3D), Java3D ;
  - Bindings pour de **très nombreux** langages.

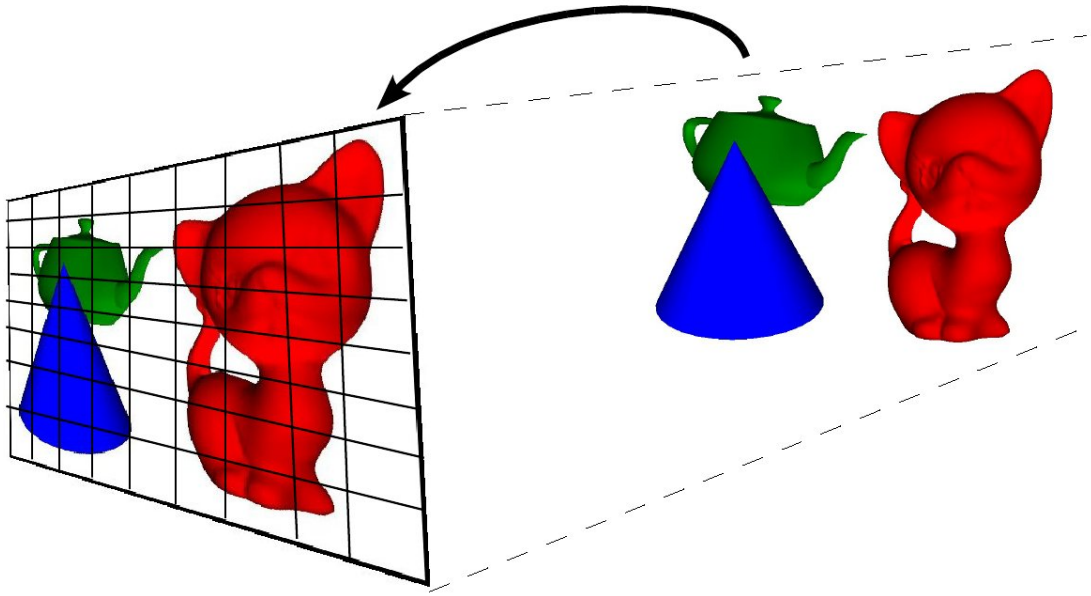
Comment ça marche : le *Pipeline* OpenGL



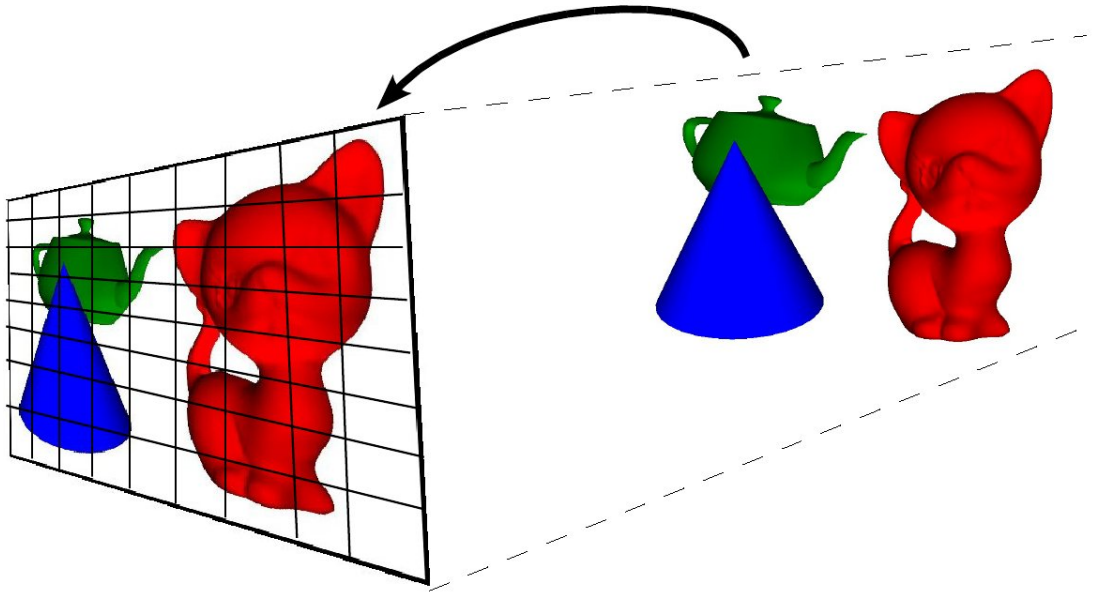
Comment ça marche : le *Pipeline* OpenGL



Comment ça marche : le *Pipeline* OpenGL

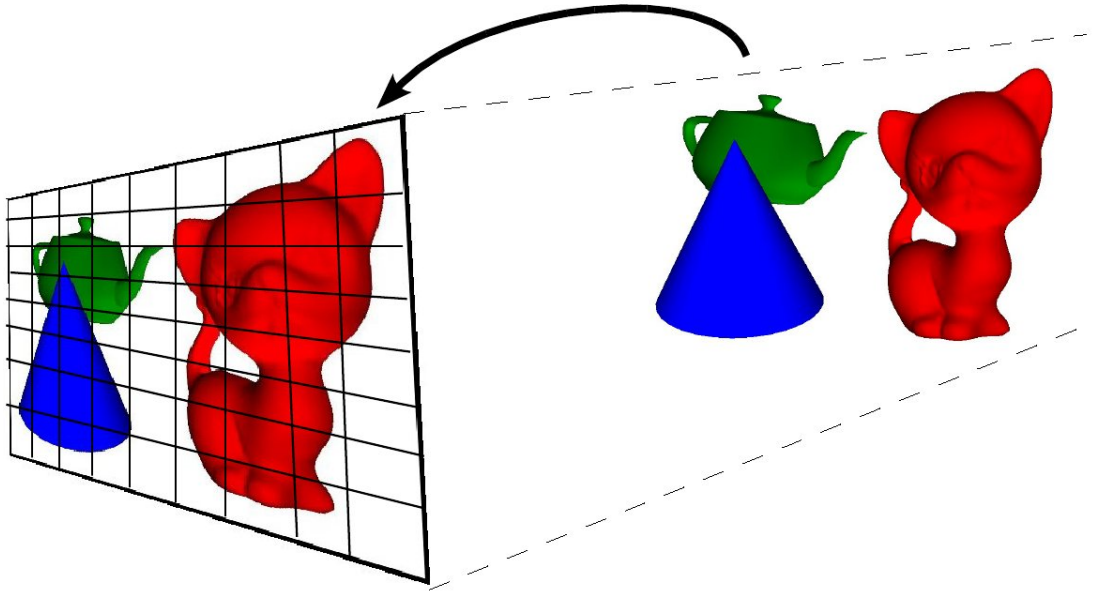


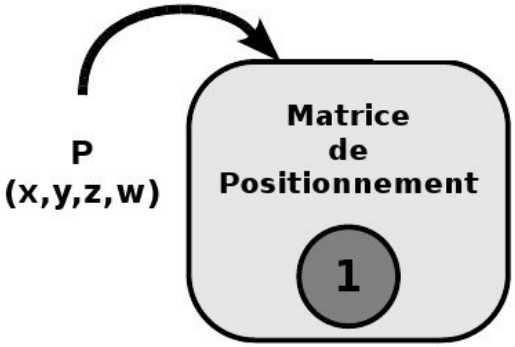
Comment ça marche : le *Pipeline* OpenGL



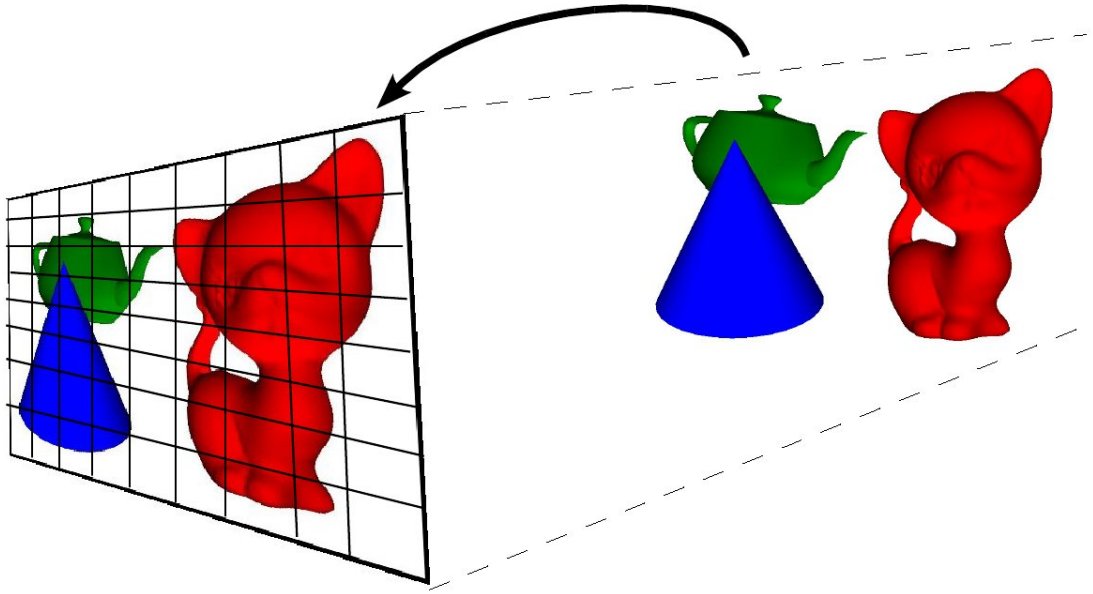
P
(x,y,z,w)

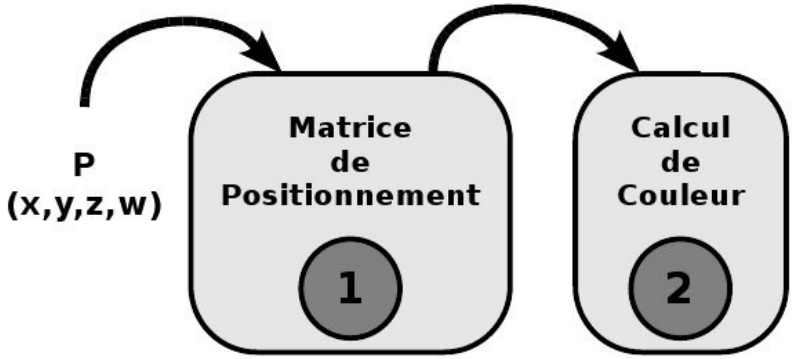
Comment ça marche : le *Pipeline* OpenGL



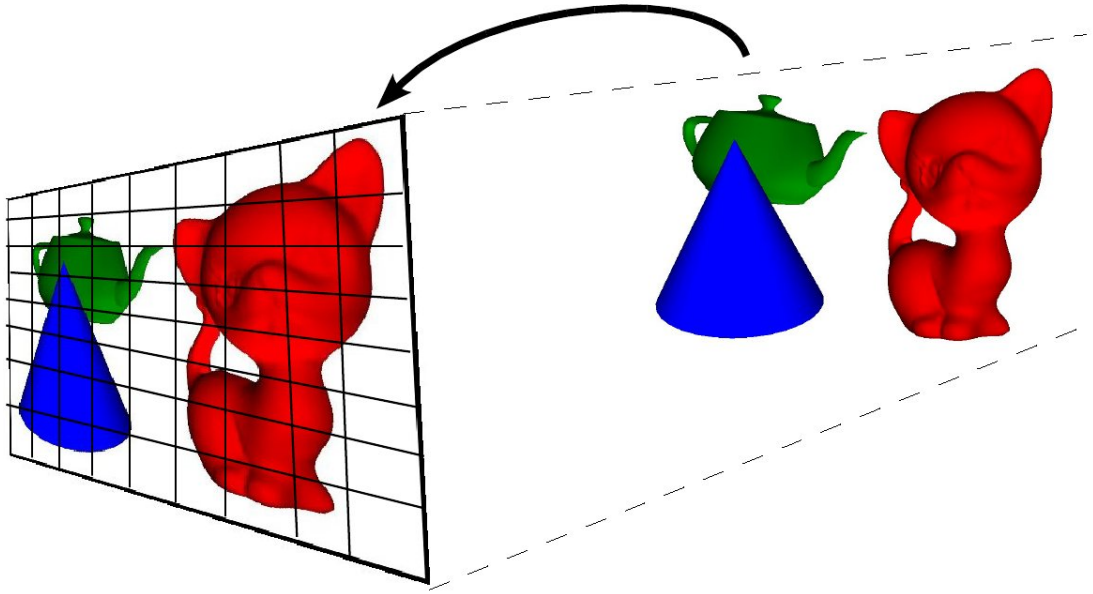


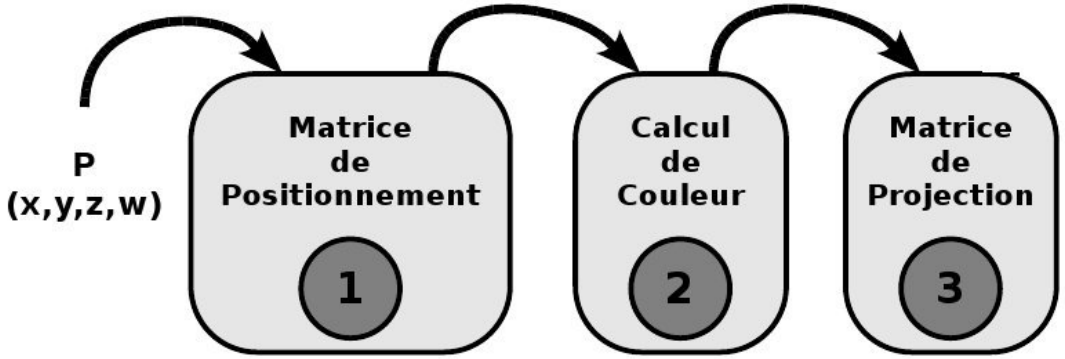
Comment ça marche : le *Pipeline* OpenGL



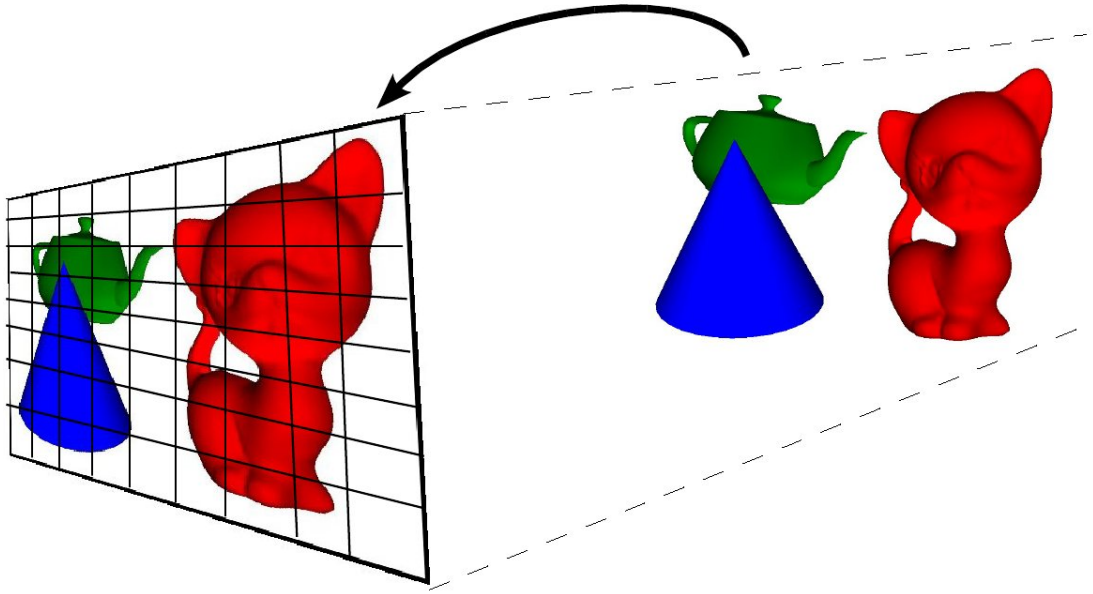


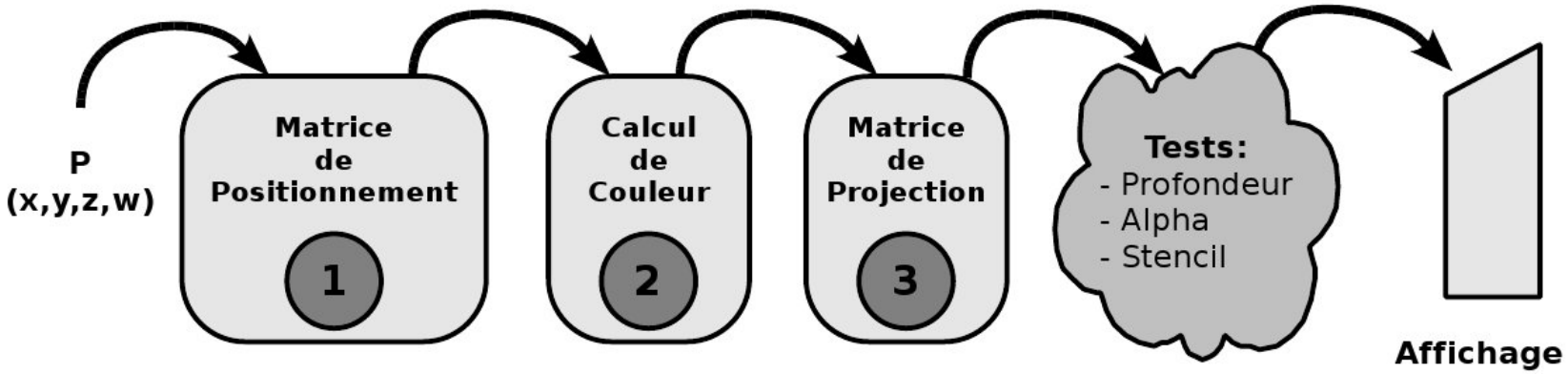
Comment ça marche : le *Pipeline* OpenGL





Comment ça marche : le *Pipeline* OpenGL





2. Machine à états et rendu projectif

Philosophie

- OpenGL : abstraction du GPU ;
- Machine à états (buffers, switches) ;
- Toutes les instructions OpenGL reposent sur :
  - L'activation et la désactivation d'états : `glEnable(GL_LIGHTING)` ;
  - La modification d'état : `glColor3f(1,0,0)` ;
  - L'empilement et la restauration de contexte : `glPushAttrib()` ;
- Constantes d'accès aux états ;
- Instructions de modification d'états ;
- Tracé des scènes *frame* par *frame* (en général double *buffer*) ;
- **Attention à la chronologie !**

Syntaxe des instructions

- Convention de l'API :
  - Préfixe en `gl` : `glPushMatrix() ;`
  - Suffixe explicitant le nombre et le type des arguments :
    - Exemple : `glColor4f(1,0,0,0.5) ;`
    - `glVertex[2,3,4][sidf][v] ;`
    - `[2,3,4]` : Nombre d'arguments ;
    - `[sidf]` : Type des arguments ;
    - `[v]` : Passage par valeur ou par adresse ;

Pour commencer : le tracé de primitives

- Instructions élémentaires pour le tracé de surfaces ;
- Exemple :

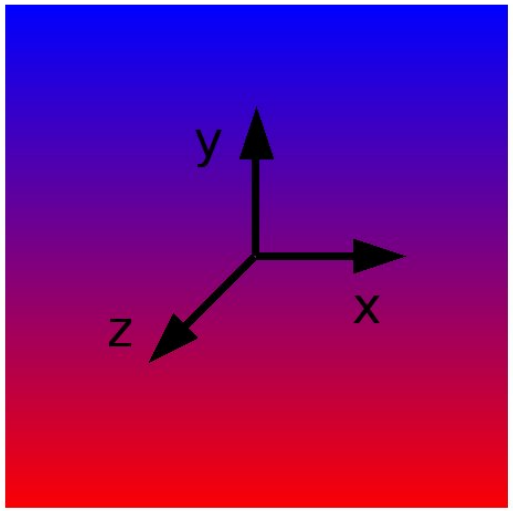
```
glBegin(GL_POLYGON) ;
  glColor3f(1,0,0) ;
  glVertex3f(-2,-2,0) ;
  glVertex3f(2,-2,0) ;
  glColor3f(0,0,1) ;
  glVertex3f(2,2,0) ;
  glVertex3f(-2,2,0) ;
glEnd() ;
```

- **Attention au sens !** (trigonométrique par défaut, *culling*)
- Entre le `glBegin` et le `glEnd` : uniquement les instructions relatives aux positions, couleurs, normales et textures ;

Pour commencer : le tracé de primitives

- Instructions élémentaires pour le tracé de surfaces ;
- Exemple :

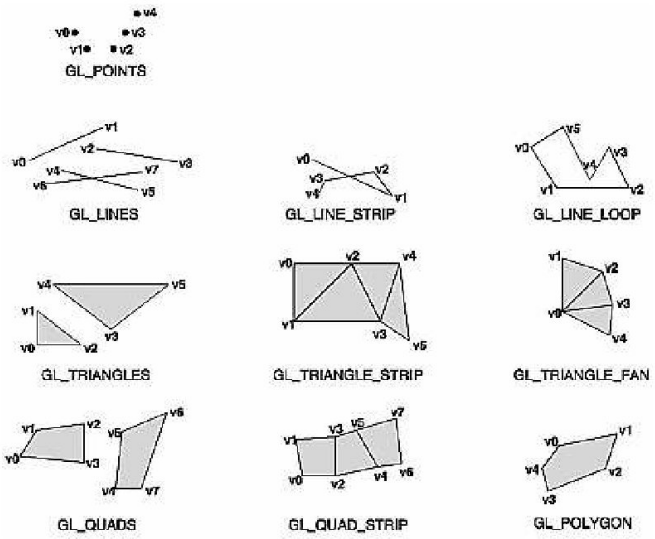
```
glBegin(GL_POLYGON) ;
  glColor3f(1,0,0) ;
  glVertex3f(-2,-2,0) ;
  glVertex3f(2,-2,0) ;
  glColor3f(0,0,1) ;
  glVertex3f(2,2,0) ;
  glVertex3f(-2,2,0) ;
glEnd() ;
```



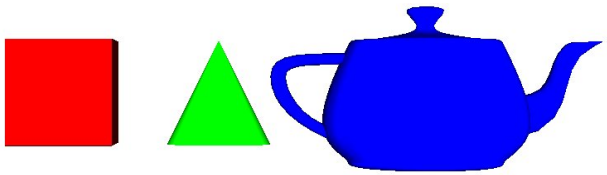
- **Attention au sens !** (trigonométrique par défaut, *culling*)
- Entre le `glBegin` et le `glEnd` : uniquement les instructions relatives aux positions, couleurs, normales et textures ;

Tracé de primitives

- Utile pour tracer :
  - des formes simples ;
  - des surfaces implicites ;
  - des maillages de polygones ;
- Primitives existantes :
  - `GL_POINTS`
  - `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
  - `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`
  - `GL_QUADS`, `GL_QUAD_STRIP`, `GL_POLYGON`
- Pour aller plus vite : primitives GLUT
  - `glutSolidSphere`, `glutSolidCone`, `glutSolidCube`, ...
  - `glutSolidTeaPot`



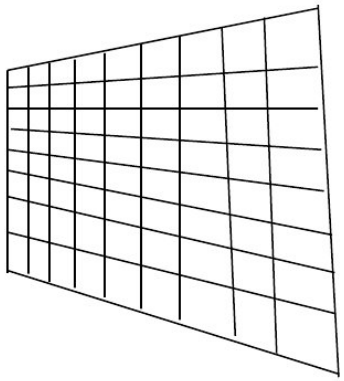
(source : OpenGL Red Book)



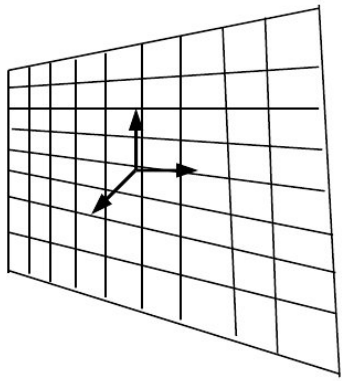
Positionnement

- Modification du repère courant (`GL_MODELVIEW`) avant le tracé ;
- Changement d'état de `GL_MODELVIEW` ;
- Déplacement par composition de produits matriciels ;
  - $M \times N \times L$: `glLoadMatrix(M) ; glMultMatrix(N) ; glMultMatrix(L) ;`
  - Rotation : `glRotatef(alpha, x, y, z) ;`
  - Translation : `glTranslatef(x,y,z) ;`
  - Mise à l'échelle : `glScalef(x,y,z) ;`
- **Attention à la chronologie !**
- Positionnement hiérarchique : sauvegarde des états de `GL_MODELVIEW` :
  - `glPushMatrix() ; glPopMatrix() ;`

Exemple de positionnement

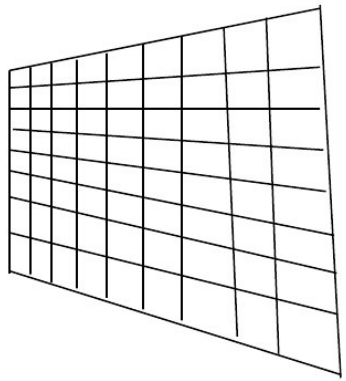


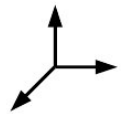
Exemple de positionnement



Exemple de positionnement

```
glTranslatef(0,0,-15) ;
```

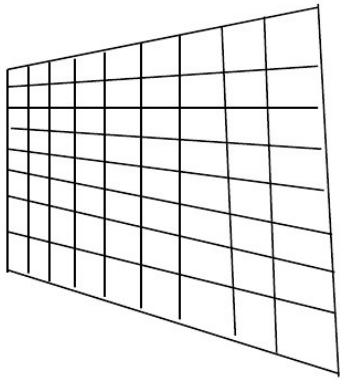


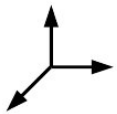


Exemple de positionnement

```
glTranslatef(0,0,-15) ;

glPushMatrix() ;
```

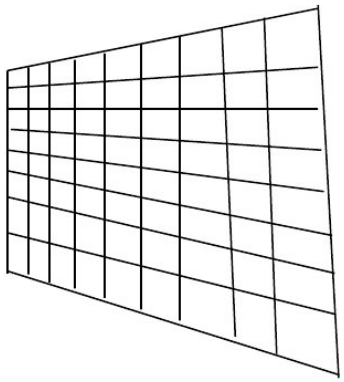


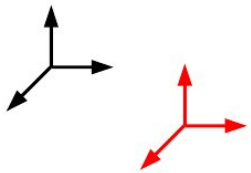


Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
```

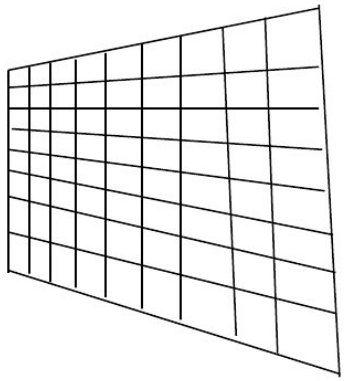


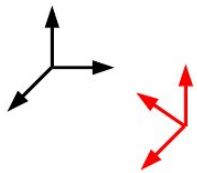


Exemple de positionnement

```
glTranslatef(0,0,-15) ;

glPushMatrix() ;
   glTranslatef(2,-2,-5) ;
   glRotatef(-100,0,1,0) ;
```

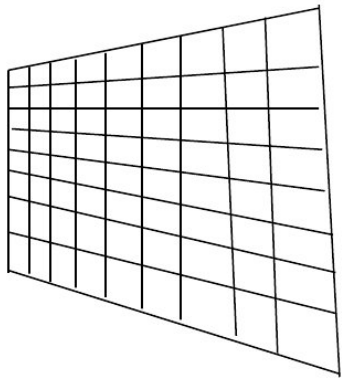


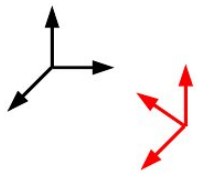


Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
```

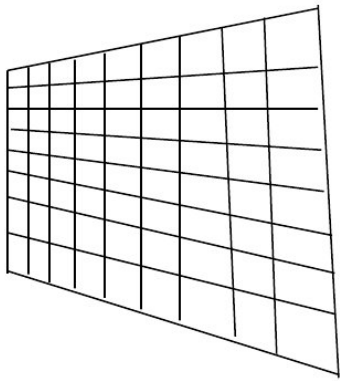


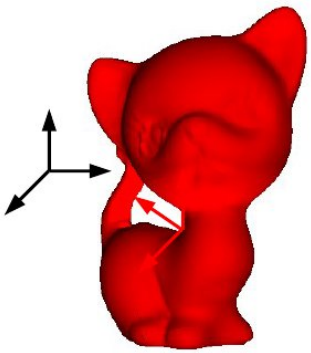


Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
   obj2polygons("kitten.obj");
```

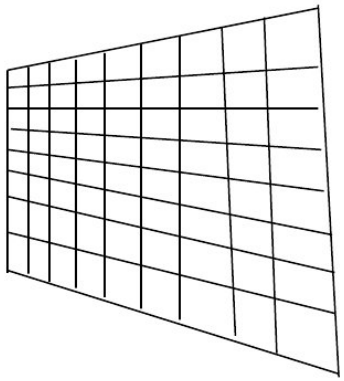


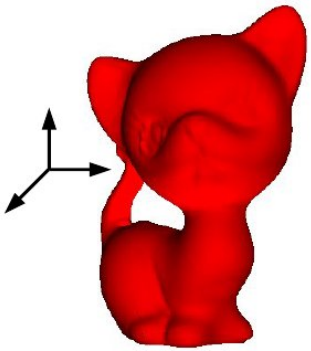


Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
   obj2polygons("kitten.obj");
glPopMatrix();
```



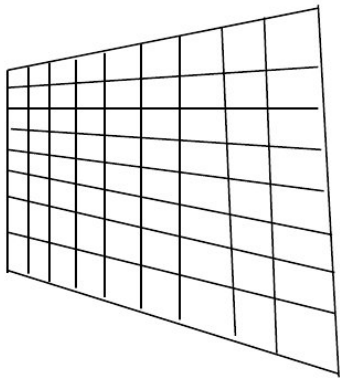


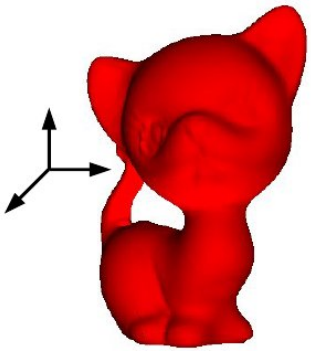
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
   obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
```



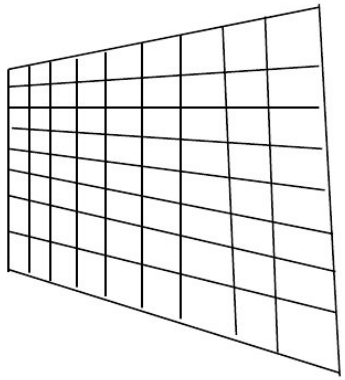


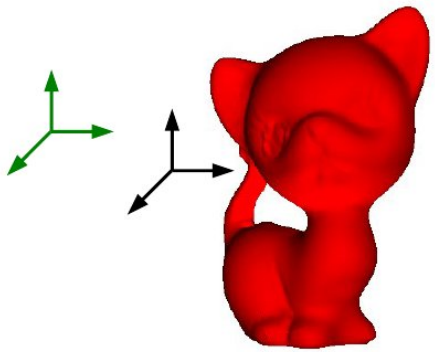
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
   obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
   glTranslatef(-1,1,-5);
```



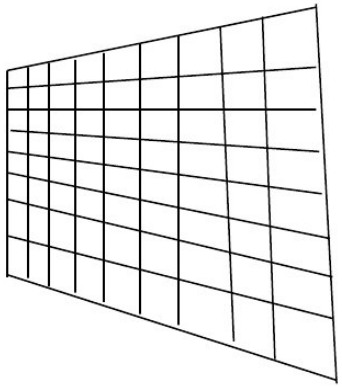


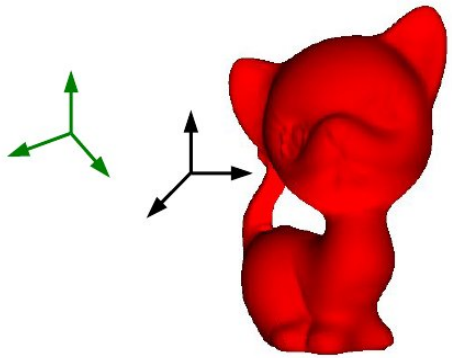
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
```



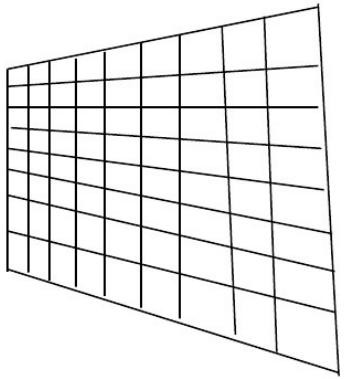


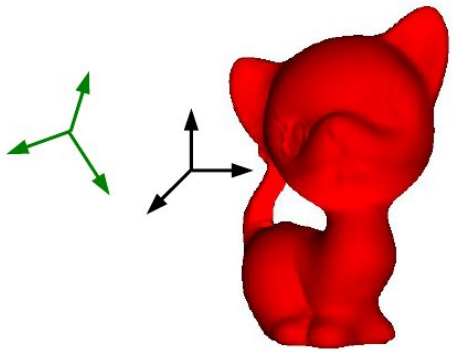
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
```



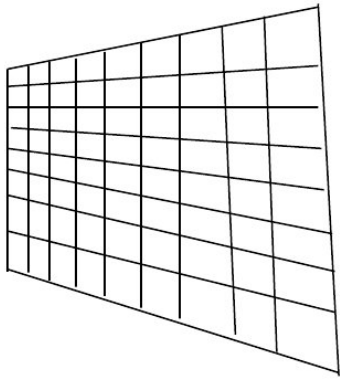


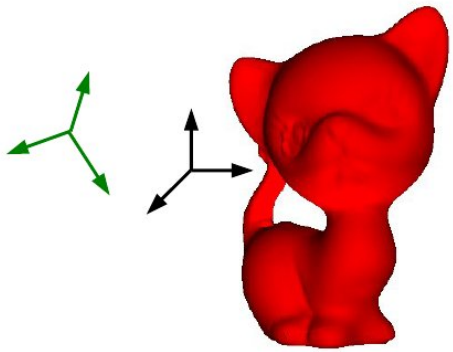
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
```



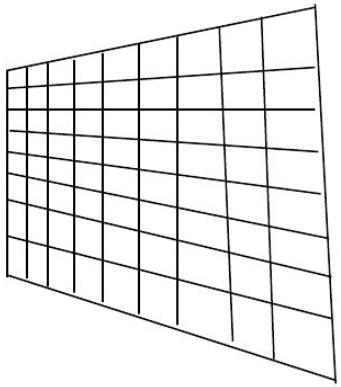


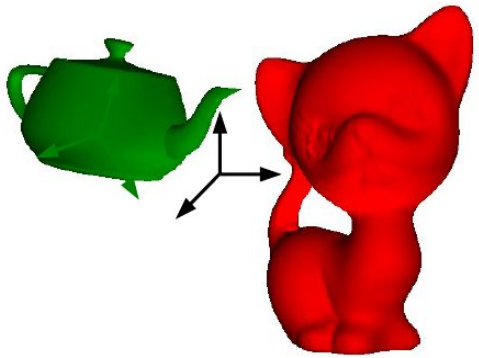
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
```



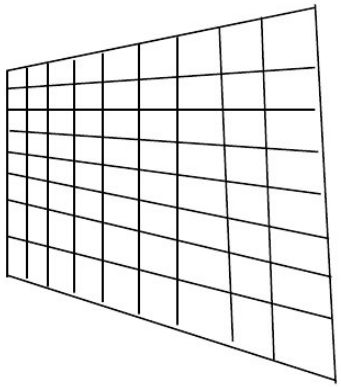


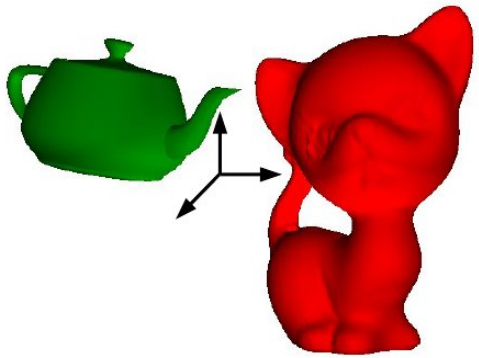
Exemple de positionnement

```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();
```





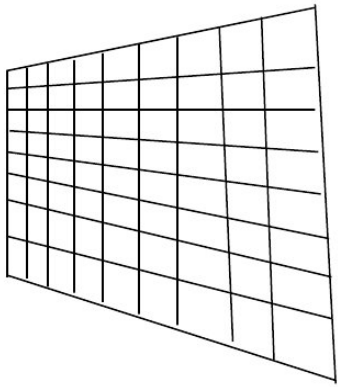
Exemple de positionnement

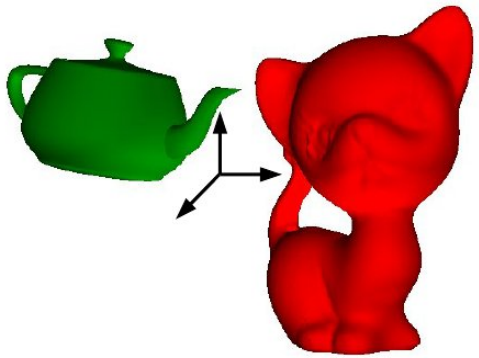
```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
```





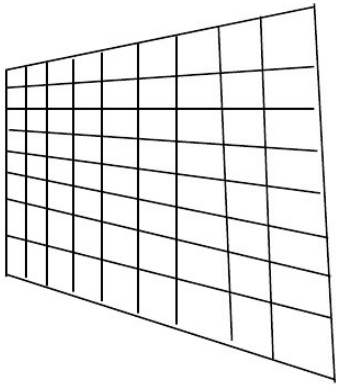
Exemple de positionnement

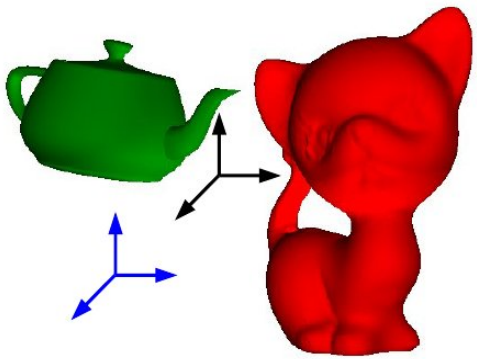
```
glTranslatef(0,0,-15);

glPushMatrix();
   glTranslatef(2,-2,-5);
   glRotatef(-100,0,1,0);
   glColor3f(0.8, 0, 0);
   obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
   glTranslatef(-1,1,-5);
   glRotatef(-30,0,1,0);
   glRotatef(-10,0,0,1);
   glColor3f(0, 0.4, 0);
   glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
   glTranslatef(-0.5,-0.5,4);
```





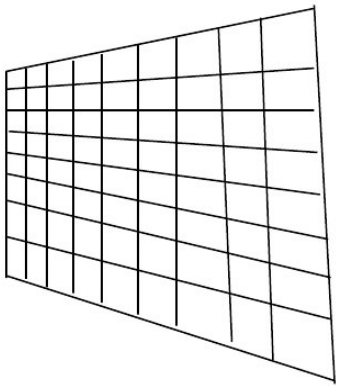
Exemple de positionnement

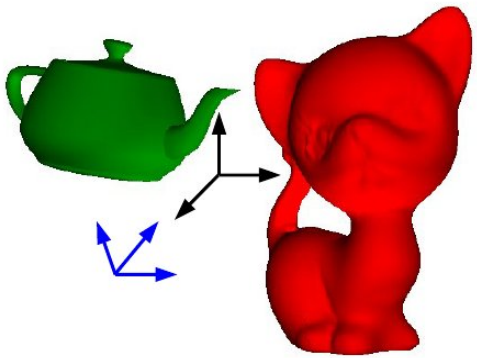
```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
    glTranslatef(-0.5,-0.5,4);
    glRotatef(-70,1,0,0);
```





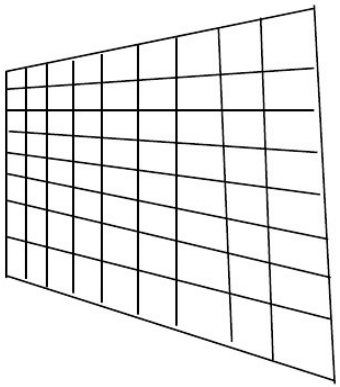
Exemple de positionnement

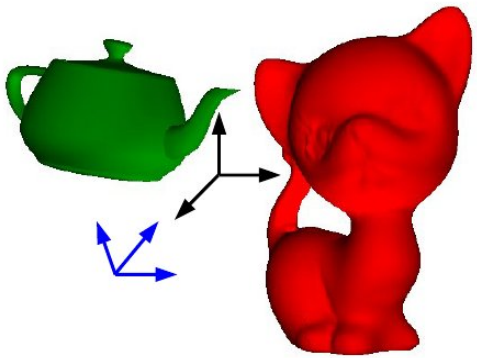
```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
    glTranslatef(-0.5,-0.5,4);
    glRotatef(-70,1,0,0);
    glColor3f(0,0,1);
```





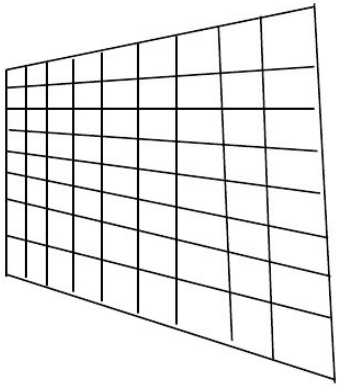
Exemple de positionnement

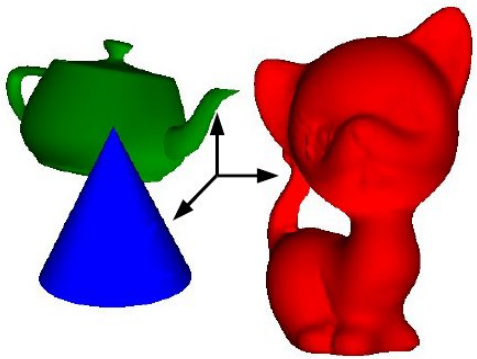
```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
    glTranslatef(-0.5,-0.5,4);
    glRotatef(-70,1,0,0);
    glColor3f(0,0,1);
    glutSolidCone(0.5,1,30,30);
```





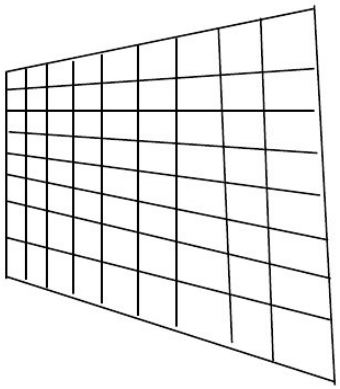
Exemple de positionnement

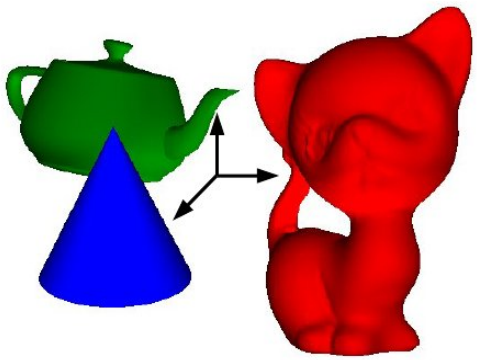
```
glTranslatef(0,0,-15);

glPushMatrix();
    glTranslatef(2,-2,-5);
    glRotatef(-100,0,1,0);
    glColor3f(0.8, 0, 0);
    obj2polygons("kitten.obj");
glPopMatrix();

glPushMatrix();
    glTranslatef(-1,1,-5);
    glRotatef(-30,0,1,0);
    glRotatef(-10,0,0,1);
    glColor3f(0, 0.4, 0);
    glutSolidTeapot(1);
glPopMatrix();

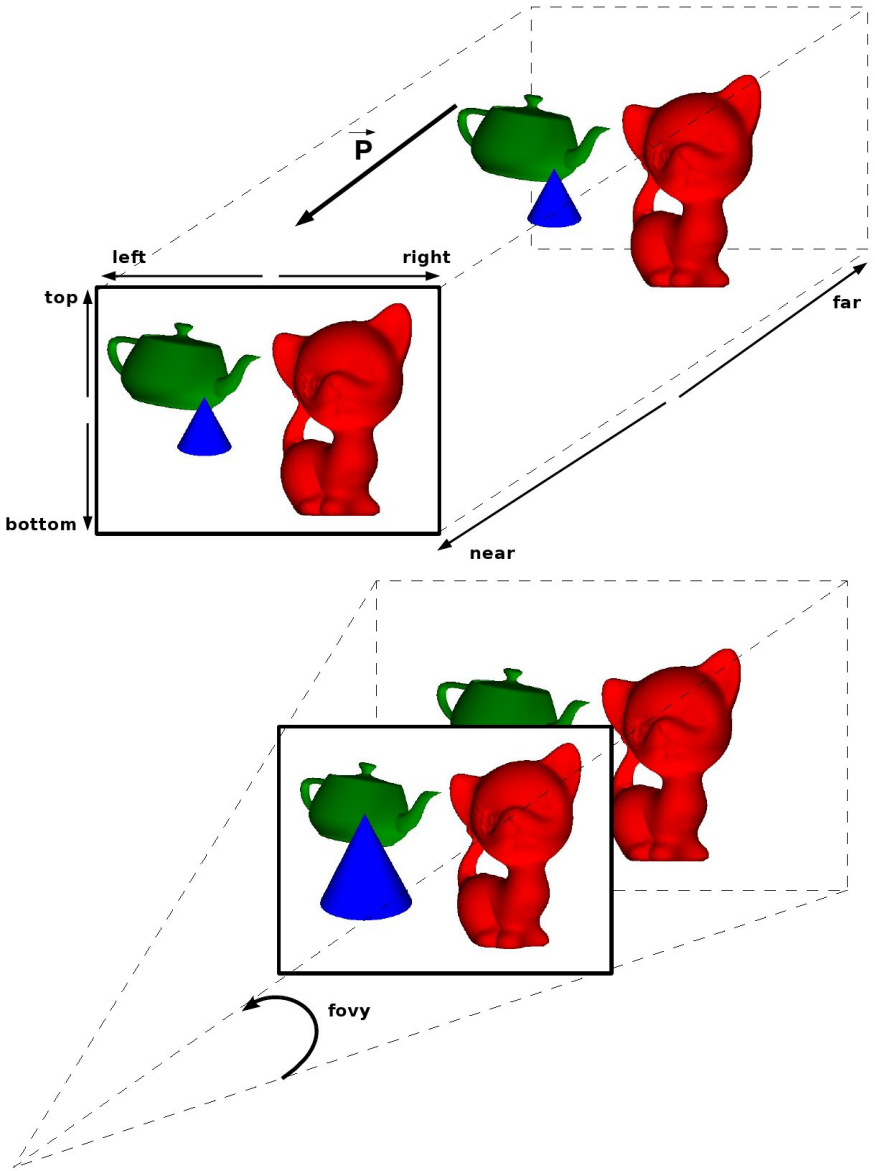
glPushMatrix();
    glTranslatef(-0.5,-0.5,4);
    glRotatef(-70,1,0,0);
    glColor3f(0,0,1);
    glutSolidCone(0.5,1,30,30);
glPopMatrix();
```





Projection à l'écran

- Définition d'un volume de vision ;
- Matrice de projection dans un plan ;
- **Attention :** modification de `GL_PROJECTION` ;
- Projection orthogonale :
  - "Supprime" la coordonée z ;
  - Conserve les parallélismes ;
  - `glOrtho(left, right, bottom, top, near, far)` ;
- Projection perspective :
  - Vue plus naturelle ;
  - `gluPerspective(fovy, ratio, near, far)` ;

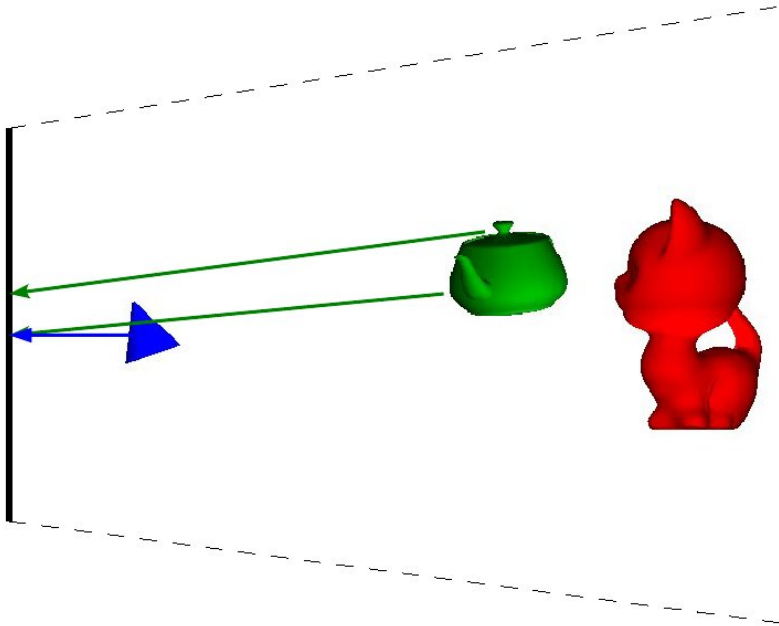


Les buffers

- Problèmes inhérents au rendu projectif :
  - Super-position d'objets ;
  - Objets transparents ; etc...
- L'écran est matérialisé par 3 buffers :
  - Le buffer de **couleur** :
    - À chaque pixel (x_i, y_i) est associée une couleur (r, g, b, a) ;
  - Le buffer de **profondeur** :
    - À chaque pixel (x_i, y_i) est associée une profondeur z.
  - Le buffer du **stencil** :
    - À chaque pixel (x_i, y_i) est associée une valeur entière.
- **Fragment** : ensemble des attributs associés à un pixel ;
- Un fragment *tracé* n'est effectivement *affiché* que s'il réussit les tests de **profondeurs**, **d'alpha** et du **stencil** ;

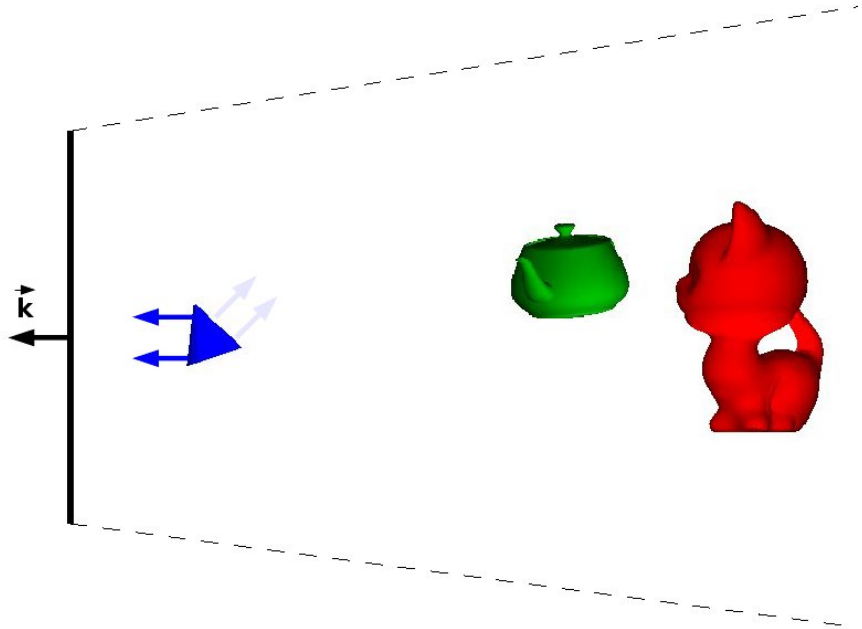
Élimination des parties cachées

- Algorithme du *Z-Buffer* :
  - Initialisation du buffer de profondeur à l'infini ;
  - Pour tous les polygones compris dans le volume de vision :
    - Calcul des fragments cibles f_c ;
    - Si $z(f_s) < z(f_c)$, $f_c \leftarrow f_s$;



Optimisation dite du *back-face culling*

- Idées :
  - Considérer uniquement les polygones tournés vers l'observateur ;
  - Réduit le temps de traitement par 2 (si maillages uniformes) ;
  - Pour chaque polygone inclus dans le volume de vision :
    - Calcul de sa normale $\vec{n}$;
    - si $\vec{n}.\vec{k} < 0$, le polygone est exclu du pipeline.



- Activation : `glEnable(GL_CULL_FACE) ;`
- cf. démo ;

Du code !

- Un main typique :

```
int main(int argc, char **argv){
  /* Initialisation de GLUT */
  glutInit(&argc,argv) ;
  glutInitWindowSize(1024,1024) ;
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH | GLUT_STENCIL) ;
  glutCreateWindow("Démo") ;

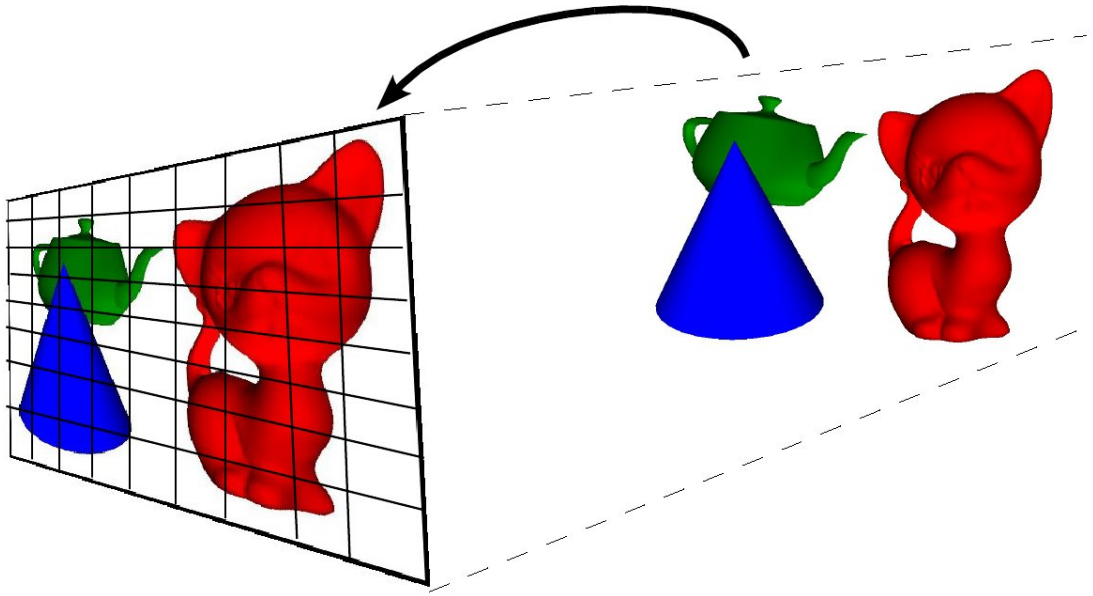
  /* Callbacks essentiels GLUT */

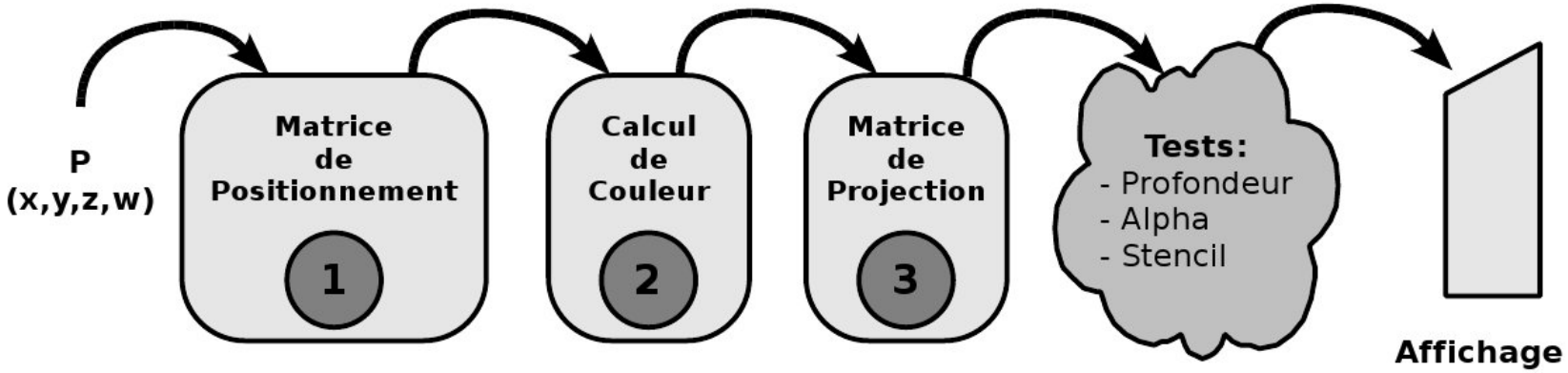
  /* Dans cette fonction est invoquée la projection */
  glutReshapeFunc(myReshape) ;
  /* Fonction de tracé (appelée frame par frame) */
  glutDisplayFunc(myDraw) ;
  /* Fonction réservée à la cinématique de la scène */
  glutIdleFunc(myIdle) ;

  glutMainLoop() ;
  return 0 ;
}
```

- cf. démo ;

Pour récapituler

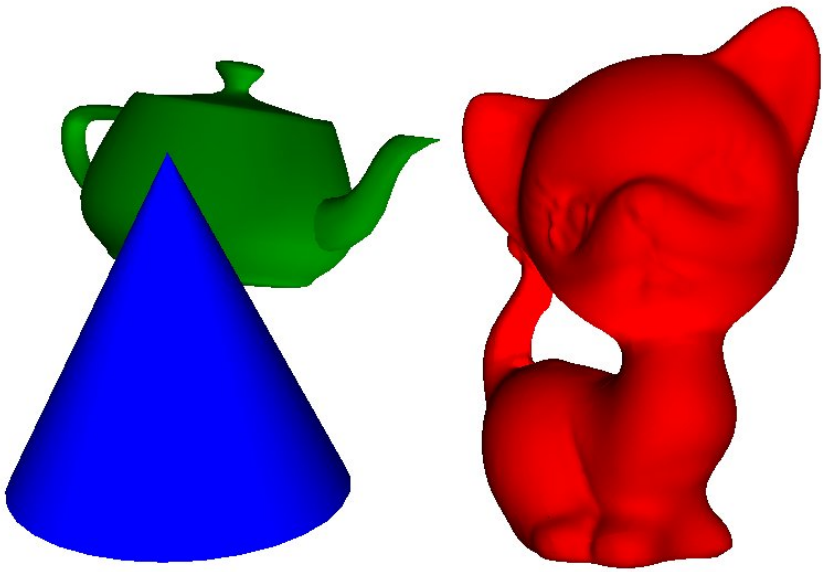




3. Modèle d'éclairement

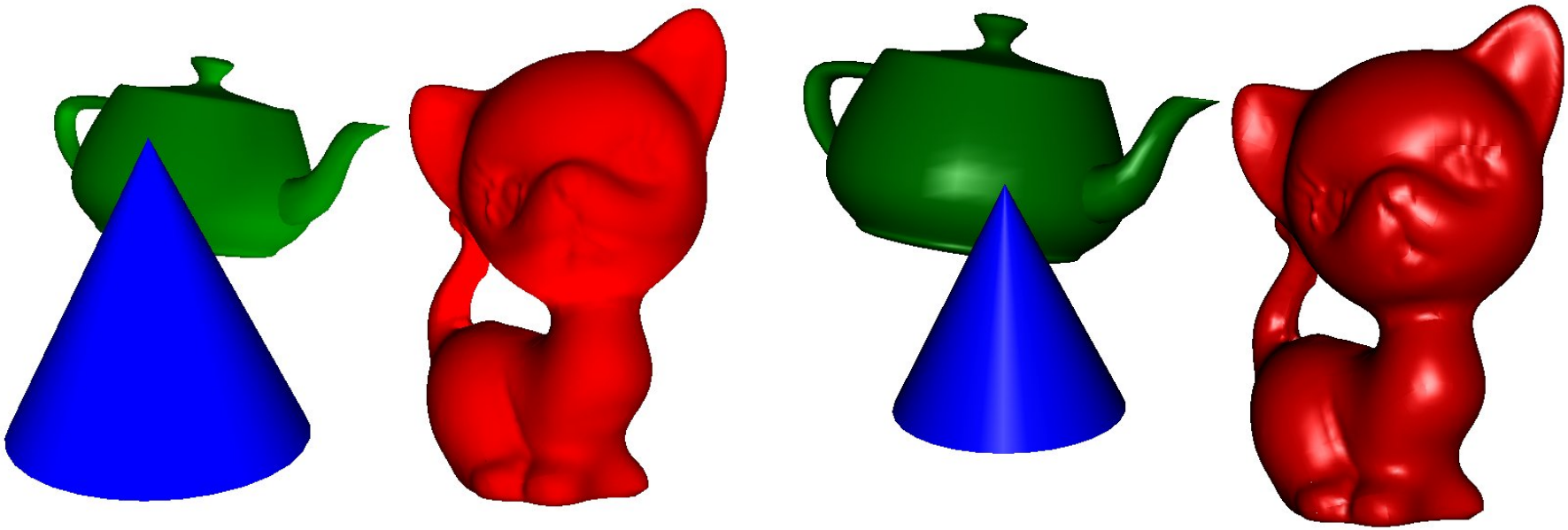
Pourquoi ?

- Ajouter du réalisme au rendu ;
- Synthétiser (grossièrement, localement) les interactions lumineuses ;



Pourquoi ?

- Ajouter du réalisme au rendu ;
- Synthétiser (grossièrement, localement) les interactions lumineuses ;

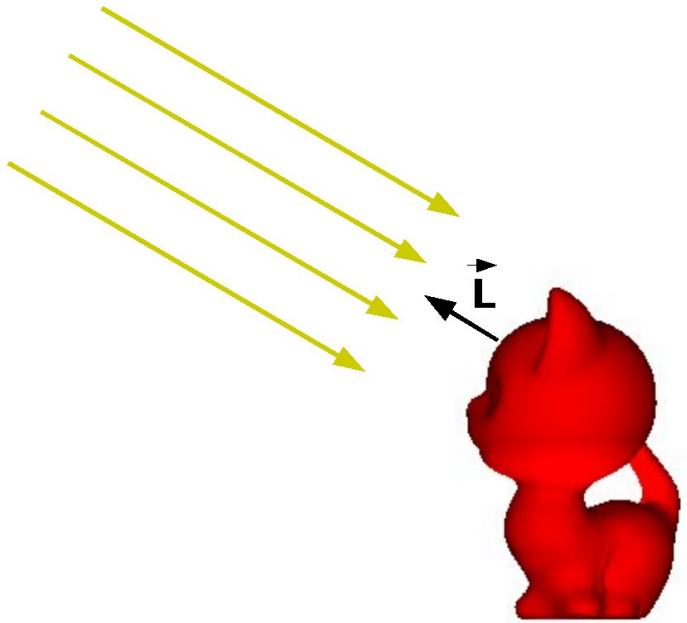


Comment ?

- Formulation empirique de Phong (**locale !**) :
  - Contribution ambiante (couleur intrinsèque de l'objet) ;
  - Contribution diffuse (couleur diffusée par l'objet) ;
  - Contribution spéculaire (couleur réfléchie par l'objet) ;
- Données nécessaires :
  - Caractéristiques des matériaux ;
  - Caractéristiques des lumières ;
- En OpenGL :
  - Calcul **sur les sommets** : `glShadeModel( {GL_FLAT , GL_SMOOTH}) ;`
  - Pour *allumer la lumière* : `glEnable(GL_LIGHTING) ;`
  - 8 sources lumineuses simultanées maximum ;
  - Deux types de sources : directionnelles et ponctuelles ;

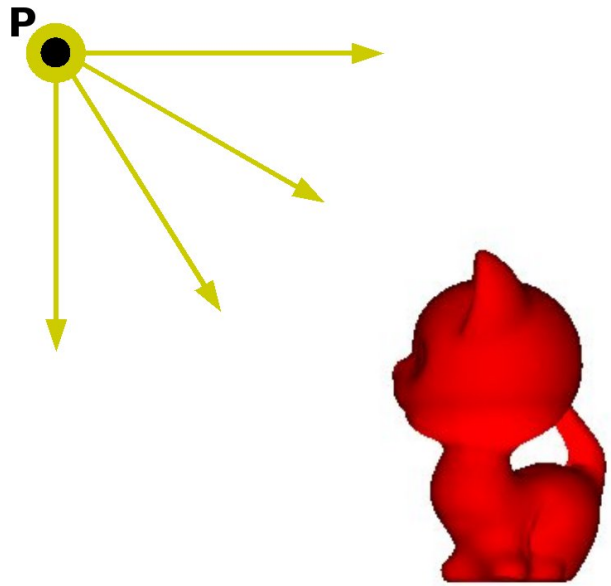
Sources directionnelles

- Lumière issue d'une direction donnée ;
- en OpenGL :
```
GLfloat dir[4] = {0, 1, 1, 0};
glLightfv(GL_LIGHT0, GL_POSITION, dir);
```
- `dir` : "d'où vient la lumière" $(\vec{L})$;
- `w = 0` $\Longleftrightarrow$ vecteur ;
- `dir` subit la matrice `GL_MODELVIEW` !
- Activation de la lumière :
```
glEnable(GL_LIGHT0);
```



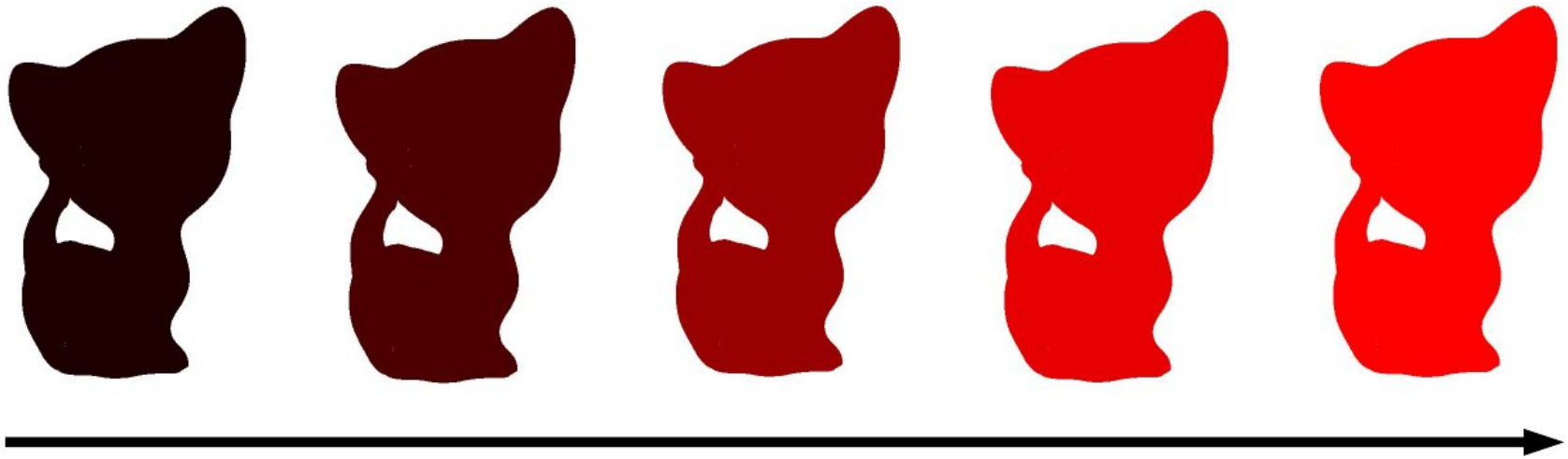
Sources ponctuelles

- Lumière omnidirectionnelle issue d'un point donné ;
- en OpenGL :
```
GLfloat pos[4] = {0, 1, 1, 1};
glLightfv(GL_LIGHT1, GL_POSITION, pos);
```
- `w = 1` $\Longleftrightarrow$ point ;
- Activation de la lumière :
```
glEnable(GL_LIGHT1);
```
- Pour information : il existe aussi un type de source "spot" ;



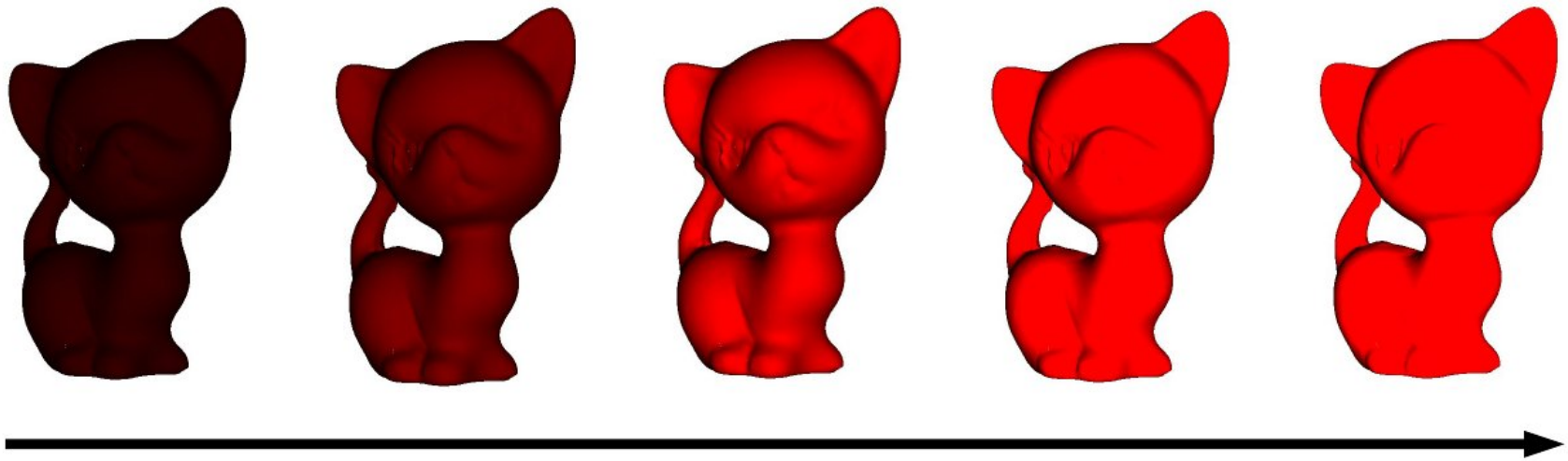
Caractéristiques d'éclairement ambiant

- Éclairement uniforme sur l'objet (couleur propre) ;
- En OpenGL, faibles coefficients (0.1) ;
- $C_A(P) = k_a \times I_a$
- `glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, float Ka[4]);`
- `glLightfv(GL_LIGHT0, GL_AMBIENT, float Ia[4]);`



Caractéristiques d'éclairement diffus

- Couleur diffusée par l'objet ;
- Dépend de l'angle d'incidence des rayons :
  - Rayon incident orthogonal à la surface : diffusion maximale ;
  - Rayon incident tangent à la surface : diffusion nulle ;
- $C_D(P) = \begin{cases} 0, & \vec{N}.\vec{L} < 0 \\ I_d \times K_d \times cos(\vec{N}, \vec{L}), & \vec{N}.\vec{L} \geq 0 \end{cases}$
- `glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, float Kd[4]);`
- `glLightfv(GL_LIGHT0, GL_DIFFUSE, float Id[4]);`



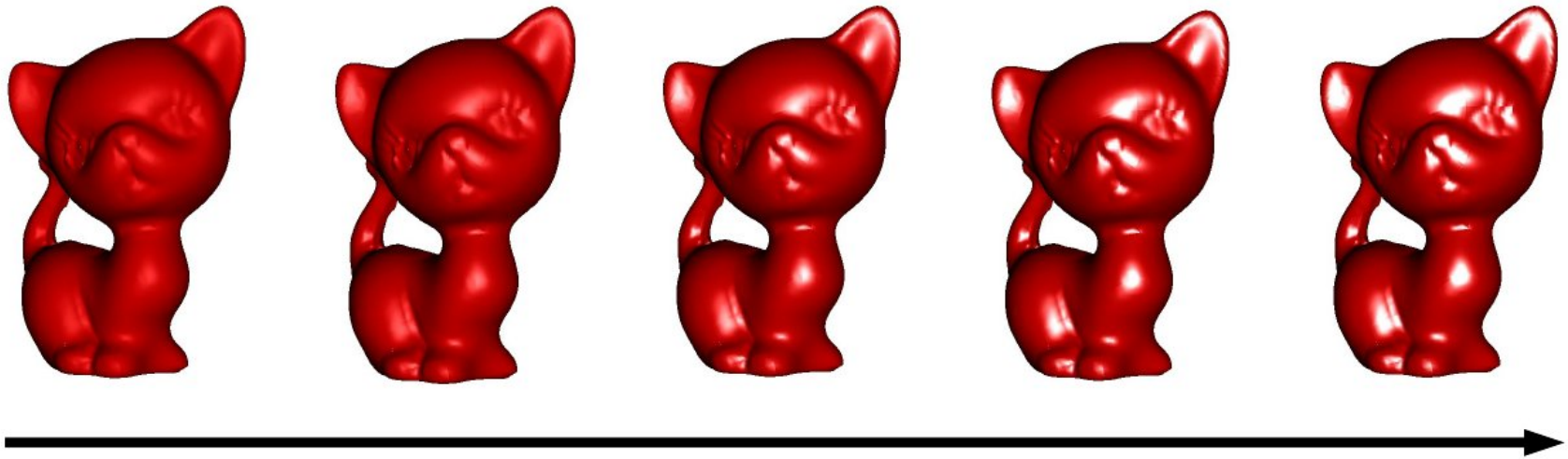
Rappels sur les normales

- Les normales ne sont pas calculées par OpenGL !
- Il faut les spécifier (flexibilité) ;
- Elles interviennent également dans le processus de **lissage** ;
- Elles sont définies lors du tracé des polygones :
  - Soit sommet par sommet ;
  - Soit par ensemble de sommet (machine à état) ;
- Exemple :

```
glBegin(GL_POLYGON) ;
   glNormal3f(0,0.1,0.9) ;
   glVertex3f(-2,-2,0) ;
   glVertex3f(2,-2,0) ;
   glNormal3f(0,-0.1,0.9) ;
   glVertex3f(2,2,0) ;
   glVertex3f(-2,2,0) ;
glEnd() ;
```

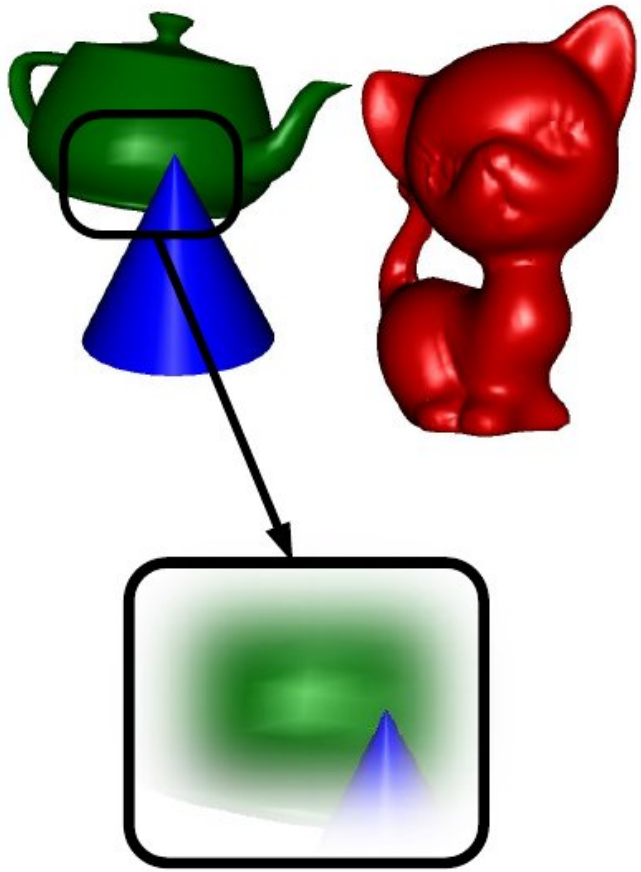
Caractéristiques d'éclairement spéculaire

- Couleur réfléchie par l'objet ;
- $\vec{R}$: vecteur du rayon réfléchi (rotation de 180° de $\vec{L}$ autour de $\vec{N}$) ;
- $\vec{V}$: vecteur de vision (visant l'observateur) ;
- Réflexion maximale quand $\vec{R}$ et $\vec{V}$ sont colinéaires ;
- $C_S(P) = I_s \times K_s \times cos(\vec{V}, \vec{R})^{Sh}$
- `glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, float Ks[4]);`
- `glLightfv(GL_LIGHT0, GL_SPECULAR, float Ia[4]);`
- Sh (brillance) : `glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, int Sh);`



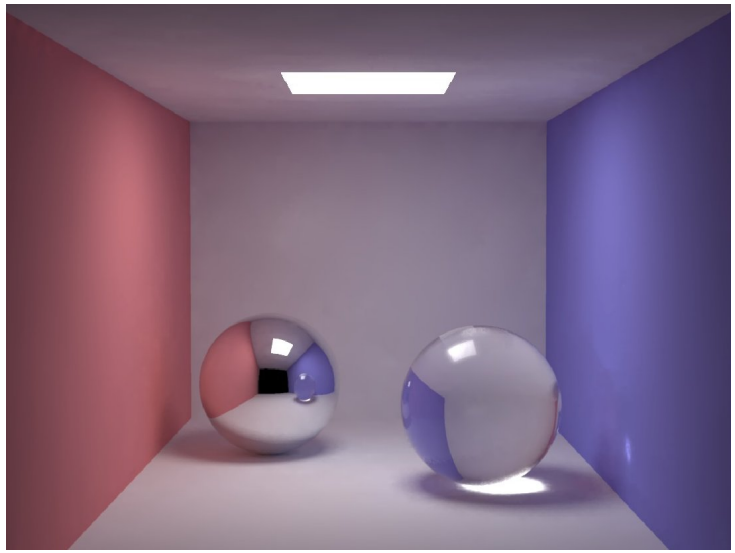
Remarques sur le modèle d'éclairement d'OpenGL

- Lorsque l'éclairement est actif, les *glColor3f* n'influent plus.
- Couleur finale : cumul des contributions des sources lumineuses ;
- **Modèle local surfacique :**
  - Dépend de l'échantillonnage de la surface ;
  - Pas de phénomène globaux :
    - Ombres et pénombres ;
    - Échange chromatique (radiosité) ;
  - Pas de modélisation des propriétés photométriques à **l'intérieur** des objets :
    - Matériaux semi-transparents ;
  - Bref... sommaire ;



Remarques sur le modèle d'éclairement d'OpenGL

- Lorsque l'éclairement est actif, les *glColor3f* n'influent plus.
- Couleur finale : cumul des contributions des sources lumineuses ;
- **Modèle local surfacique :**
  - Dépend de l'échantillonnage de la surface ;
  - Pas de phénomène globaux :
    - Ombres et pénombres ;
    - Échange chromatique (radiosité) ;
  - Pas de modélisation des propriétés photométriques à **l'intérieur** des objets :
    - Matériaux semi-transparents ;
  - Bref... sommaire ;



(Jensen et al., Siggraph 2007)

Remarques sur le modèle d'éclairement d'OpenGL

- Lorsque l'éclairement est actif, les *glColor3f* n'influent plus.
- Couleur finale : cumul des contributions des sources lumineuses ;
- **Modèle local surfacique :**
  - Dépend de l'échantillonnage de la surface ;
  - Pas de phénomène globaux :
    - Ombres et pénombres ;
    - Échange chromatique (radiosité) ;
  - Pas de modélisation des propriétés photométriques à **l'intérieur** des objets :
    - Matériaux semi-transparents ;
  - Bref... sommaire ;

(Jensen et al., Siggraph 2001)

Eye candy : le *bump-mapping*

- Enrichir le réalisme des objets ;
- Perturbation locale du calcul d'éclairement ;
- Pas de modification de la géométrie ;
- Modification des normales ;
- cf. démo ;

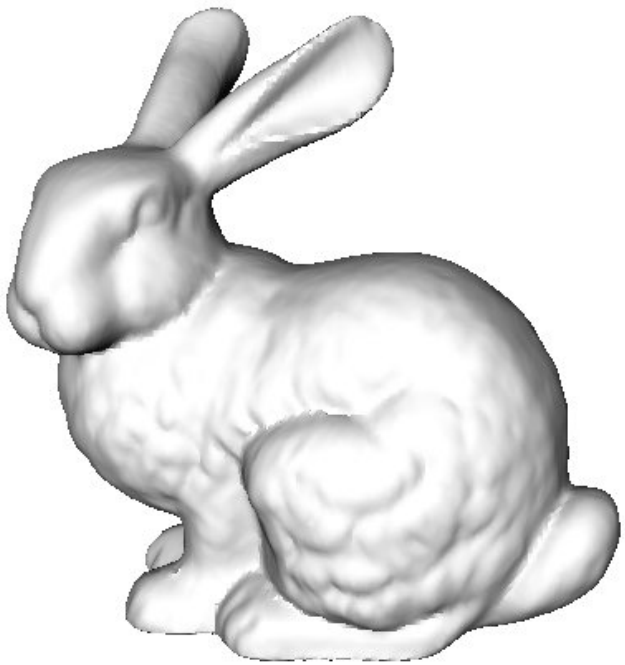
Pour récapituler

- L'éclairement en OpenGL :
  - Définir les sources lumineuses (8 au maximum) :
    - Type (point ou vecteur) ;
    - Caractéristiques ambiantes, diffuses et spéculaires ;
  - Les activer !
  - Définir les caractéristiques des matériaux (ambiantes, diffuses et spéculaires) lors du tracé ;

4. Placage de textures

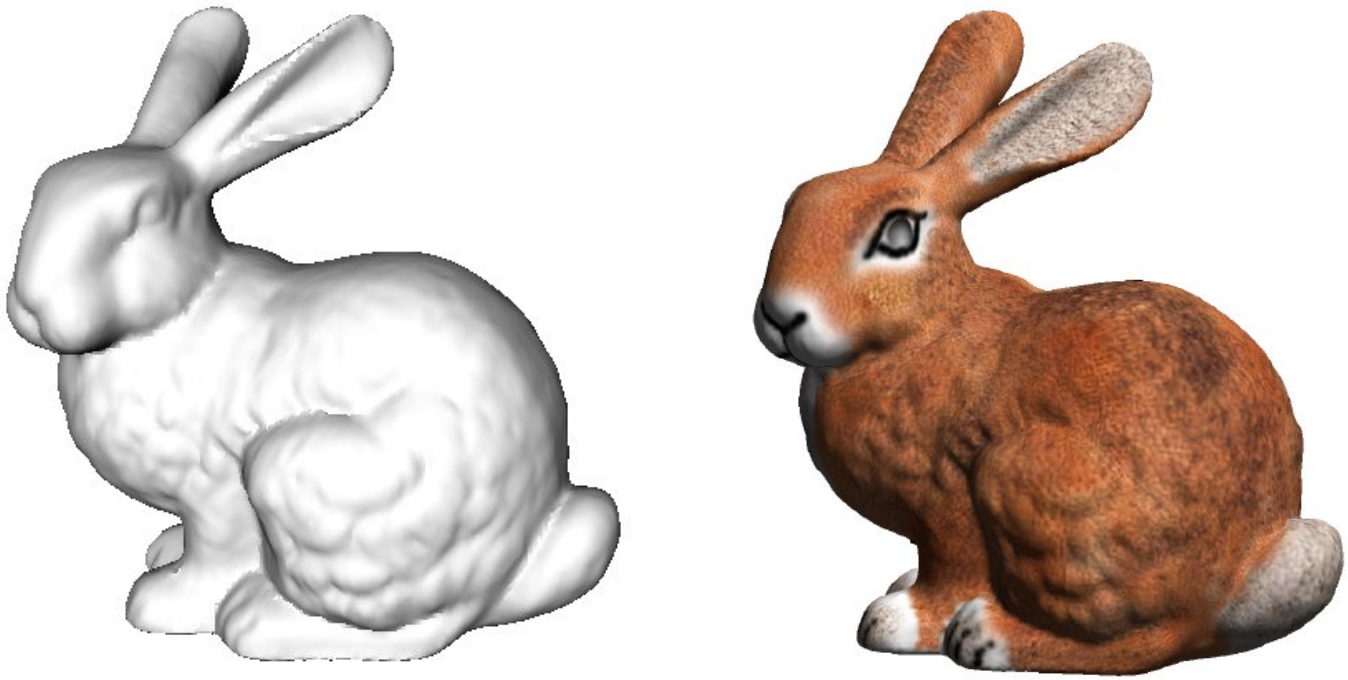
Objectifs

- Accroître le réalisme (bis) ;
- Ajouter du détail sur des géométries simples (murs, plafonds, etc.) ;



Objectifs

- Accroître le réalisme (bis) ;
- Ajouter du détail sur des géométries simples (murs, plafonds, etc.) ;



(Lévy et al., Siggraph 2002)

Comment ?

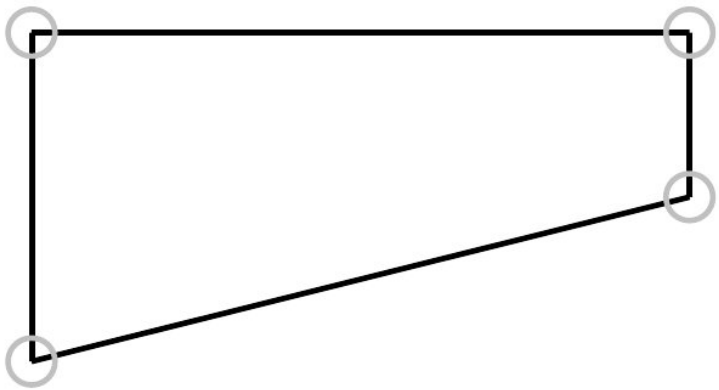
- En *collant* une image sur la surface ;
- Problème dit de paramétrisation :
  - $\phi : S \rightarrow \mathbb{R}^2$
  - Correspondance entre les points de la surface et les points de l'image ;
- Espace texture :
  - Système de coordonnées : $s \in [0, 1]$, $t \in [0, 1]$;
  - Origine du repère : en bas à gauche de l'image ;
  - Élément de texture : *texel* ;
- Paramétrisations :
  - Quelques paramétrisations simples fournies par OpenGL ;
  - ... à la main ;

Paramétrisation sommet par sommet

- Association *texel-vertex* pendant le tracé du polygone :

```
glBegin(GL_POlYGON);
   glTexCoord2f(0,1);
   glVertex3f(0,0,0);
   glTexCoord2f(1,1);
   glVertex3f(2,0.5,0);
   glTexCoord2f(1,0);
   glVertex3f(2,1,0);
   glTexCoord2f(0,0);
   glVertex3f(0,1,0);
glEnd();
```

- si $s \notin [0,1]$, $t \notin [0,1]$: répétition dans l'espace des textures ;

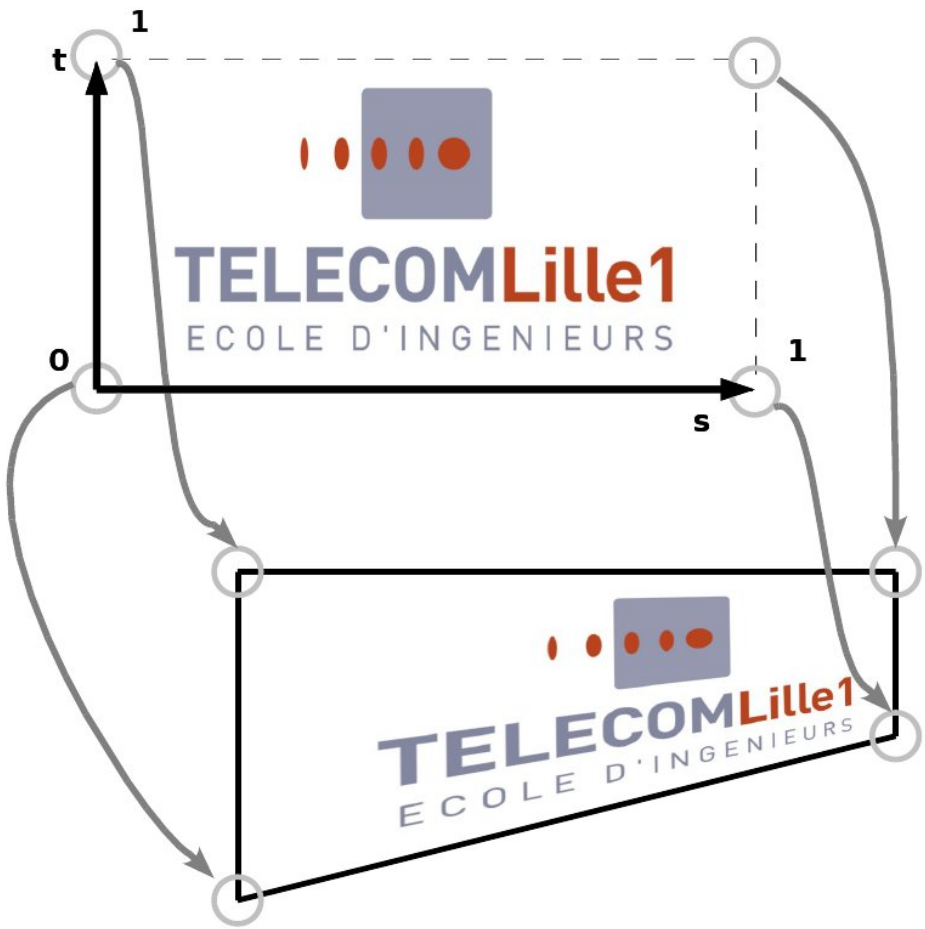


Paramétrisation sommet par sommet

- Association *texel-vertex* pendant le tracé du polygone :

```
glBegin(GL_POlYGON) ;
   glTexCoord2f(0,1) ;
   glVertex3f(0,0,0) ;
   glTexCoord2f(1,1) ;
   glVertex3f(2,0.5,0) ;
   glTexCoord2f(1,0) ;
   glVertex3f(2,1,0) ;
   glTexCoord2f(0,0) ;
   glVertex3f(0,1,0) ;
glEnd() ;
```

- si $s \notin [0,1]$, $t \notin [0,1]$: répétition dans l'espace des textures ;



Dans le code

- Chargement des images en mémoire (**taille en puissance de 2 !**) :

```
glGenTextures(1, &tex_id);          /* Génération de l'identifiant */
glBindTexture(GL_TEXTURE_2D, tex_id);   /* Sélection de la texture */
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexImage2D(
   GL_TEXTURE_2D,    /* Type image classique */
   0,    /* Niveau de mip-map (cf. cours de synthèse d'image) */
   3,    /* Taille mémoire d'un pixel */
   width, height,    /* Dimension de l'image */
   0,    /* Bords et recollements */
   GL_RGB, GL_UNSIGNED_BYTE,
   buffer);    /* Adresse du buffer image */
/* Anti-aliasing (cf. cours de synthèse d'image) */
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
 /* Texture et calcul d'éclairement */
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

- Activation du placage de texture (pendant le tracé) :

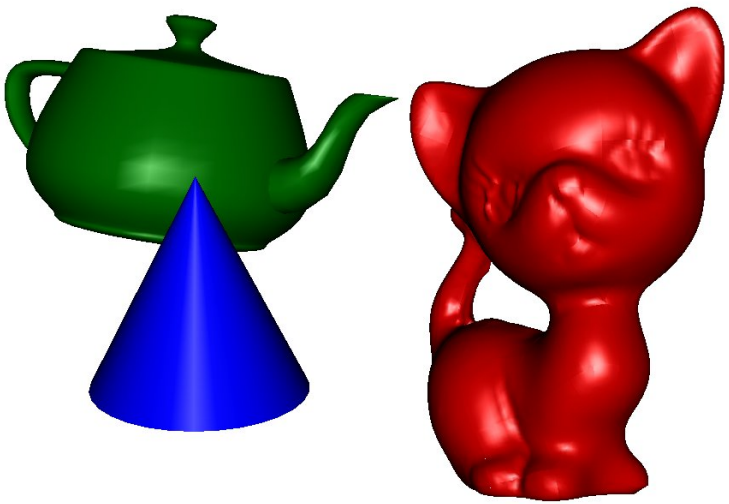
```
glBindTexture(GL_TEXTURE_2D, tex_id);

glEnable(GL_TEXTURE_2D);
```

5. Techniques avancées de rendu projectif

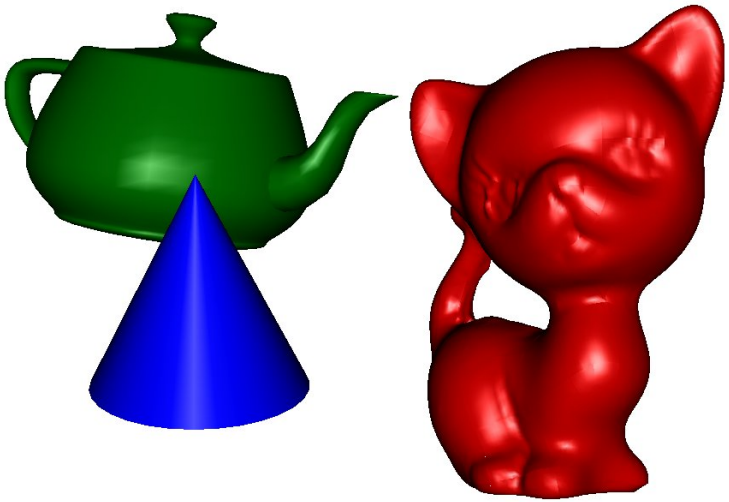
Objectifs

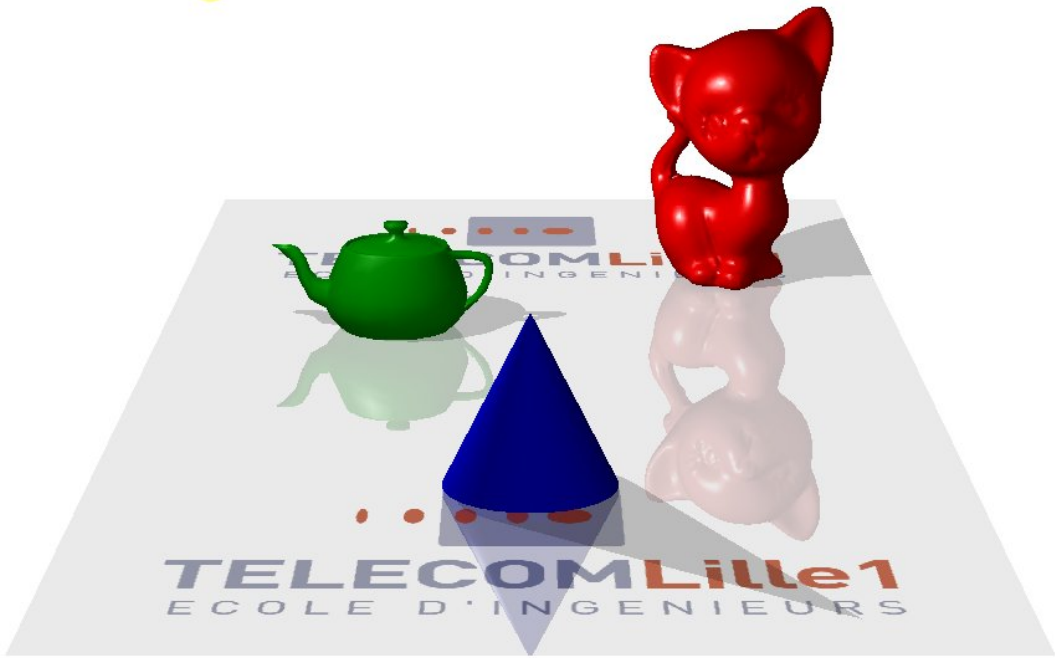
- Accroître le réalisme (ter) ;
- Effets *non réalisables* (en théorie) en rendu projectif :
  - Matériaux transparents ;
  - Phénomènes de réflexion (planaire) ;
  - Ombres projetées, etc...



Objectifs

- Accroître le réalisme (ter) ;
- Effets *non réalisables* (en théorie) en rendu projectif :
  - Matériaux transparents ;
  - Phénomènes de réflexion (planaire) ;
  - Ombres projetées, etc...





Transparence

- Technique dite de l'*alpha blending* (`glEnable(GL_BLEND)`) ;
- Mélange de couleur calculé au niveau pixel !
- Canal alpha : opacité du matériau ($\alpha \in [0, 1]$) ;
- $f_c \leftarrow \beta_s(\alpha_s) \times f_s + \beta_c(\alpha_c) \times f_c$;
- $\beta(\alpha)$: fonction de mélange :
  - `glBlendFunc(`β_s`,` β_c`)` ;
  - En général : $\beta_s = \alpha_s$ et $\beta_c = 1 - \alpha_s$;
  - `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)` ;
- **Problème !**
  - Mélange avec la couleur précédemment enregistrée dans le buffer ;
  - Les objets placés *derrière* les objets transparents doivent donc être tracés en premier !

Test alpha

- Tracé en deux passes :
  - Objets opaques d'abord ;
  - Objets transparents ensuite (**par ordre de profondeur !**) ;

- ```
  glEnable(GL_ALPHA_TEST) ;
  glAlphaFunc(GL_EQUAL, 1.0) ;
  tracer_scene() ;
  glAlphaFunc(GL_LESS, 1.0) ;
  tracer_scene() ;
  ```

- Dans le pipeline :

  Si le test alpha réussit alors
  Si le test de profondeur réussit alors
  Mise à Jour : Profondeur & Couleur
  ```

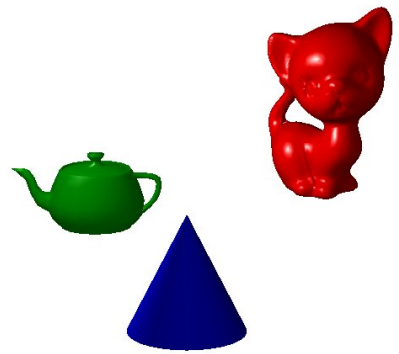
Test alpha

- Tracé en deux passes :
  - Objets opaques d'abord ;
  - Objets transparents ensuite (**par ordre de profondeur !**) ;

- ```
  glEnable(GL_ALPHA_TEST);
  glAlphaFunc(GL_EQUAL, 1.0);
  tracer_scene();
  glAlphaFunc(GL_LESS, 1.0);
  tracer_scene();
  ```

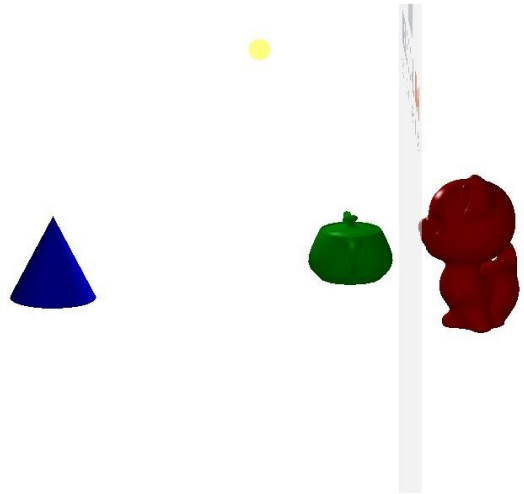
- Dans le pipeline :

  Si le test alpha réussit alors
  Si le test de profondeur réussit alors
  Mise à Jour : Profondeur & Couleur


  ```

Test alpha

- Tracé en deux passes :
  - Objets opaques d'abord ;
  - Objets transparents ensuite (**par ordre de profondeur !**) ;



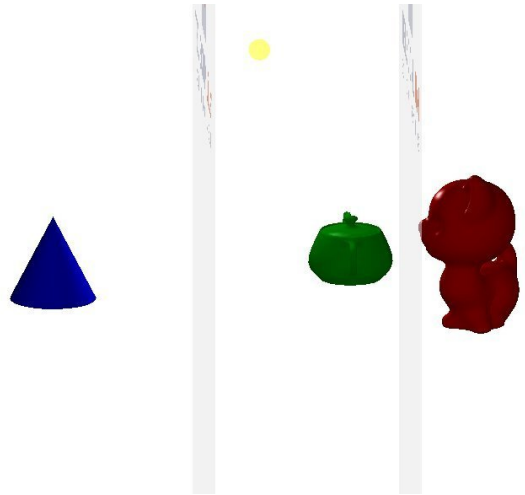
- ```
  glEnable(GL_ALPHA_TEST);
  glAlphaFunc(GL_EQUAL, 1.0);
  tracer_scene();
  glAlphaFunc(GL_LESS, 1.0);
  tracer_scene();
  ```

- Dans le pipeline :

  Si le test alpha réussit alors
  Si le test de profondeur réussit alors
  Mise à Jour : Profondeur & Couleur
  ```

Test alpha

- Tracé en deux passes :
  - Objets opaques d'abord ;
  - Objets transparents ensuite (**par ordre de profondeur !**) ;



```
glEnable(GL_ALPHA_TEST) ;
glAlphaFunc(GL_EQUAL, 1.0) ;
tracer_scene() ;
glAlphaFunc(GL_LESS, 1.0) ;
tracer_scene() ;
```

- Dans le pipeline :

  Si le test alpha réussit alors
  Si le test de profondeur réussit alors
  Mise à Jour : Profondeur & Couleur

Réflexion planaire

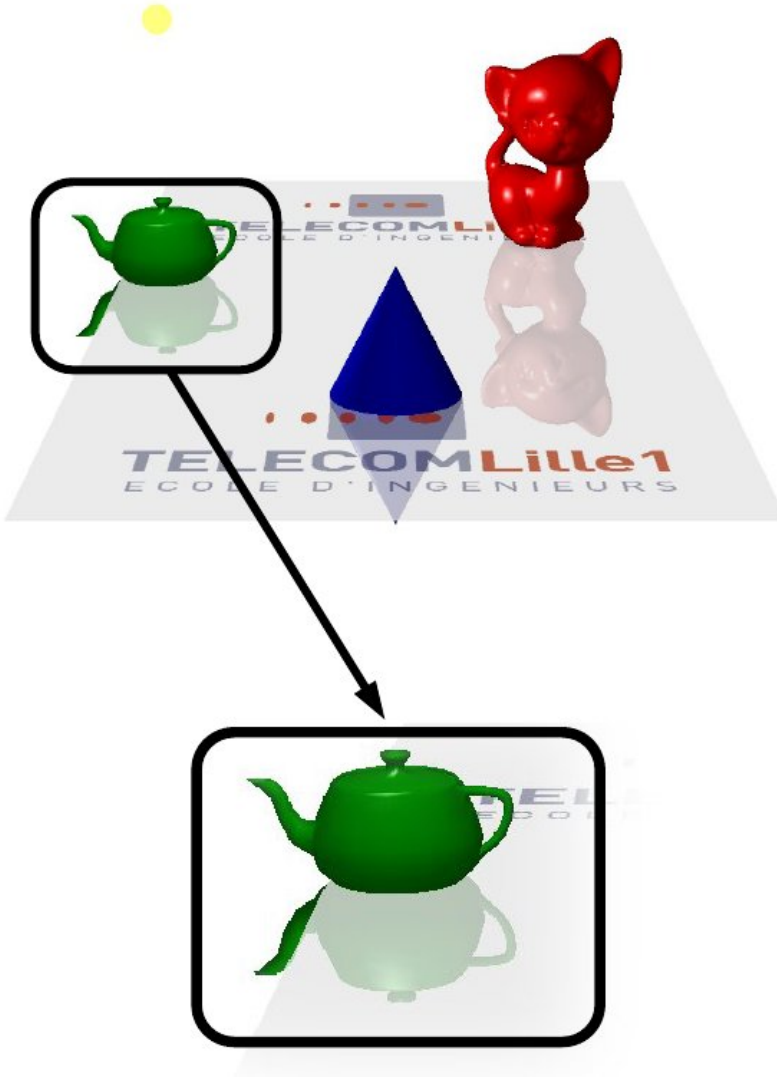
- **Astuce :** Utilisation de la transparence ;
- Tracé du symétrique **complet** de la scène :
  - Objets ;
  - ... **et sources lumineuses !**
  - En OpenGL (plan $\{0, x, z\}$) :

    ```
    glScale(1, -1, 1) ;
    ```

- Ordre du tracé :
  - Objets opaques (originaux et symétriques) ;
  - Miroir plan (en transparence) ;
- **Problème !**
  - Restreindre l'affichage du symétrique à la surface du plan !
  - Utilisation du *stencil buffer* (masque) ;

Réflexion planaire

- **Astuce :** Utilisation de la transparence ;
- Tracé du symétrique **complet** de la scène :
  - Objets ;
  - ... **et sources lumineuses !**
  - En OpenGL (plan $\{0, x, z\}$) :
    `glScale(1, -1, 1);`
- Ordre du tracé :
  - Objets opaques (originaux et symétriques) ;
  - Miroir plan (en transparence) ;
- **Problème !**
  - Restreindre l'affichage du symétrique à la surface du plan !
  - Utilisation du *stencil buffer* (masque) ;



Stencil buffer

- Mécanisme au niveau pixel permettant de conditionner finement l'affichage ;
- Buffer qui associe à chaque pixel (x_i, y_i) une valeur (un octet) ;
- Systèmes de tests sur ces valeurs avant mise à jour d'un fragment ;
- En OpenGL :
  - Activation : `glEnable(GL_STENCIL_TEST)` ;
  - Initialisation (à chaque frame) : `glClear(GL_STENCIL_BUFFER_BIT)` ;
- Dans le pipeline :

```
Si le test alpha réussit alors
  Si le test stencil réussit alors
    Si le test de profondeur réussit alors
      Mise à Jour : Profondeur, Couleur, Stencil (3)
    Sinon Mise à Jour : Stencil (2)
  Sinon Mise à Jour : Stencil (1)
```

Test du stencil

- Spécification du test
  - `glStencilFunc(<Op>, <Ref>, <Masq>)` ;
  - Le test stencil réussit si `(<Ref> & <Masq>)` appliqué à l'opérateur `Op` réussit ;
  - Opérateurs : `GL_ALWAYS`, `GL_NEVER`, `GL_EQUAL`, `GL_LESS`, `GL_GREATER`, etc...
  - Exemple :
    - `glStencilFunc(GL_EQUAL, 1, 0xFF)` ;
    - $\Longleftrightarrow$ le test stencil réussit si le stencil du pixel courant vaut 1 ;
- Spécification des mises à jour (1), (2) & (3)
  - `glStencilOp(<si le stencil échoue> (1),`
    `<si le stencil réussit et la profondeur échoue (2)>,`
    `<si le stencil et la profondeur réussissent (3)>)` ;
  - Opérations : `GL_KEEP`, `GL_REPLACE`, `GL_ZERO`, `GL_INCR`, `GL_DECR`, , etc...
- Très nombreuses possibilités !

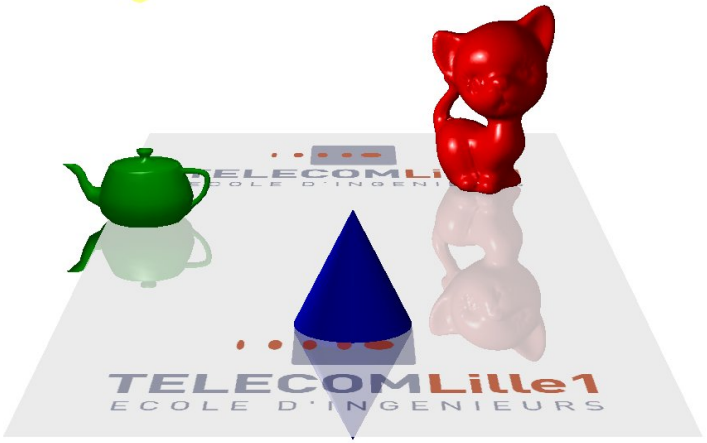
Réflexion planaire avec stencil

- **Astuce :** marquer les pixels *où* afficher les symétriques (masquage) ;
- Marquer les pixels correspondant au miroir plan ;
- Afficher les symétriques sur les pixels marqués uniquement ;
- **Problèmes !**
  - Le plan doit être tracé **avant** les symétriques pour *marquer* le stencil ;
  - Or le plan doit être tracé après les symétriques pour pouvoir mélanger les couleurs !
- **Solution** :
  - Tracer une première fois le plan en ne modifiant **que le stencil buffer** ;
  - Puis tracer la scène, avec test sur le stencil pour la symétrique ;
  - **Attention à l'ordre !**

Détail de la solution (1/2)

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT
        | GL_STENCIL_BUFFER_BIT)
/* Désactivation de la mise à jour C & P */
glDepthMask(GL_FALSE);
glColorMask(GL_FALSE,
        GL_FALSE,GL_FALSE,GL_FALSE);

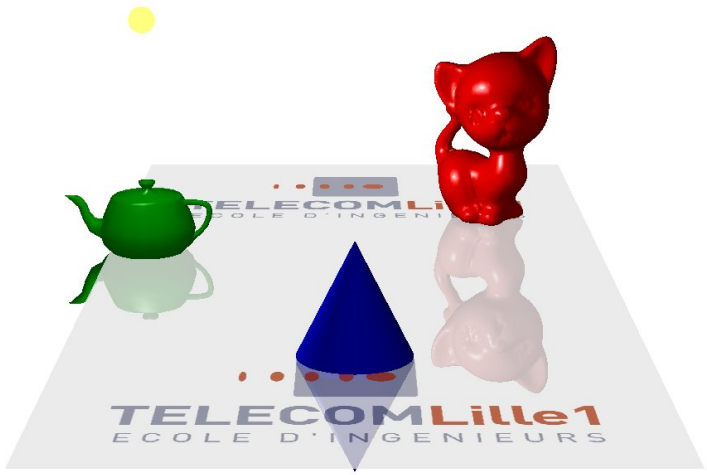
glEnable(GL_STENCIL_TEST);
/* Met à 1 tous les fragments tracés */
glStencilFunc(GL_ALWAYS, 1, 0xFF);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
tracer_plan();
```



Détail de la solution (2/2)

```
/* Activation de la mise à jour C & P */
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glStencilFunc(GL_EQUAL, 1, 0xFF);
glPushMatrix();
   glScalef(1, -1, 1);
   tracer_scene();
glPopMatrix();
glDisable(GL_STENCIL_TEST);
tracer_scene();

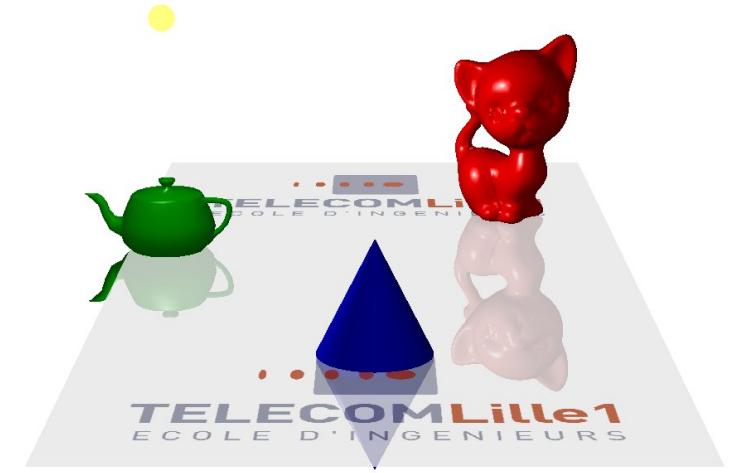
/* Affichage réel du plan */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);
tracer_plan();
glDisable(GL_BLEND);
```

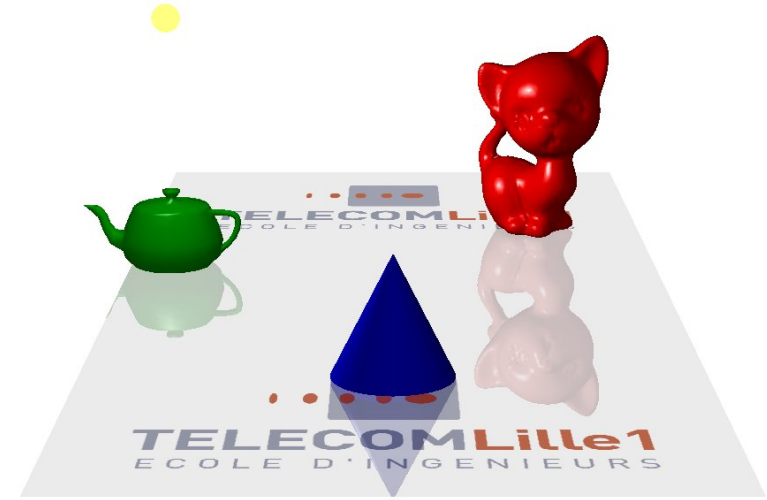


Détail de la solution (2/2)

```
/* Activation de la mise à jour C & P */
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glStencilFunc(GL_EQUAL, 1, 0xFF);
glPushMatrix();
   glScalef(1, -1, 1);
   tracer_scene();
glPopMatrix();
glDisable(GL_STENCIL_TEST);
tracer_scene();

/* Affichage réel du plan */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);
tracer_plan();
glDisable(GL_BLEND);
```



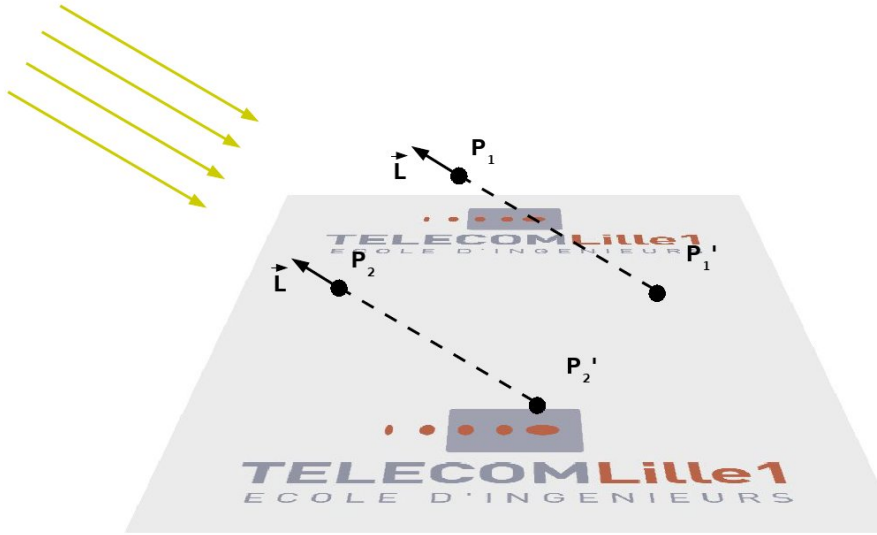


Ombres Projetées

- **Astuce** : considérer l'ombre comme un objet (transparent) ;
- O : matrice de projection des objets sur le plan ;
- Appliquer O et tracer chaque objet (en version aplatie) ;
- Deux cas à traiter :
  - Source lumineuse directionnelle ;
  - Source lumineuse ponctuelle ;

Matrice de projection - Source directionnelle

- $P' = P - \lambda_d \vec{L}$
- Équation d'un plan :
  $ax' + by' + cz' + d = 0$
- $\Longleftrightarrow \lambda_d = \frac{aP_x + bP_y + cP_z + d}{aL_x + bL_y + cL_z}$
- $\Longleftrightarrow$ Matrice de projection :

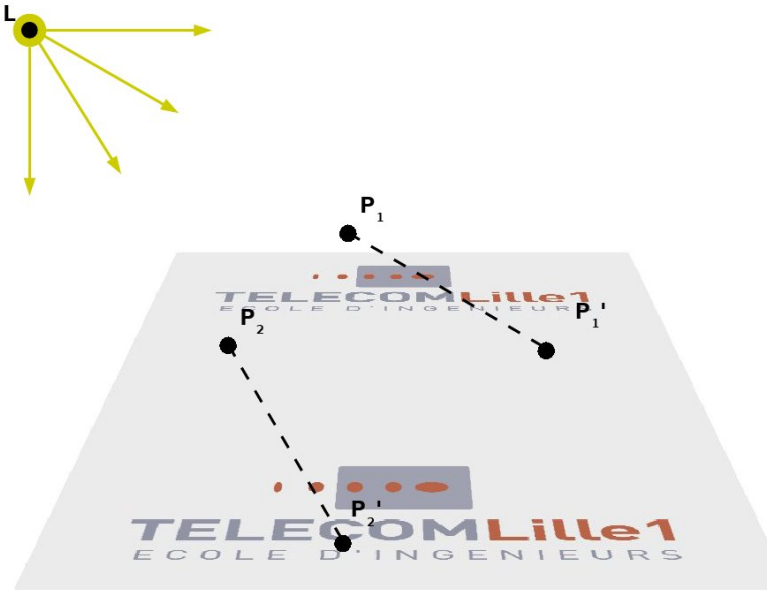


$$\begin{pmatrix} bL_y + cL_z & -bL_x & -cL_x & -dL_x \\ -aL_y & aL_x + cL_z & -cL_y & -dL_y \\ -aL_z & -bL_z & aL_x + bL_y & -dL_z \\ 0 & 0 & 0 & aL_x + bL_y + cL_z \end{pmatrix}$$

Matrice de projection - Source ponctuelle

- $P' = P - \lambda_p \overrightarrow{PL}$
- Équation d'un plan :
  $ax' + by' + cz' + d = 0$
- $\Longleftrightarrow \lambda_p = \frac{aP_x + bP_y + cP_z + d}{a(P_x - L_x) + b(P_y - L_y) + c(P_z - L_z)}$
- $\Longleftrightarrow$ Matrice de projection :

$$\begin{pmatrix} bL_y + cL_z + d & -bL_x & -cL_x & -dL_x \\ -aL_y & aL_x + cL_z + d & -cL_y & -dL_y \\ -aL_z & -bL_z & aL_x + bL_y + d & -dL_z \\ -a & -b & -c & aL_x + bL_y + cL_z \end{pmatrix}$$



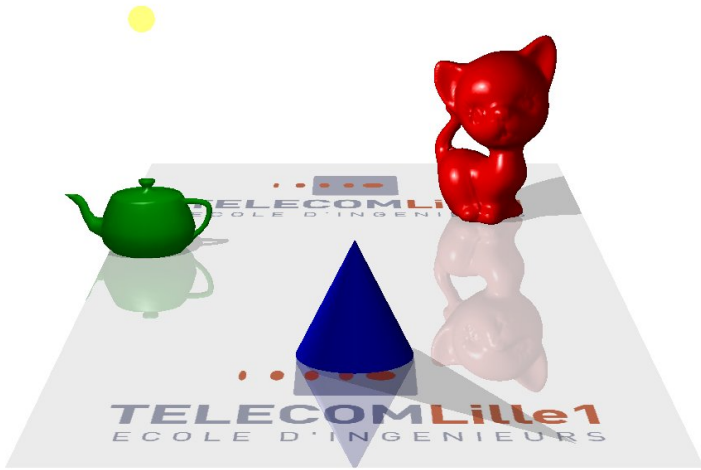
Détail de la solution (cf. démo)

```
 /* Après l'affichage réel du plan */

/* Légère translation (Z-fighting) */
glTranslatef(0, 0.02, 0);
/* Projection */
glMultMatrix(O);
/* Teinte de l'ombre */
glDisable(GL_LIGHTING);
glColor4f(0.01,0.01,0.01,0.2);

/* Affichage sur les pixels marqués seulement*/
glEnable(GL_STENCIL_TEST);
/* Une seule mise à jour par fragment !*/
glStencilOp(GL_KEEP, GL_KEEP, GL_ZERO);
/* Mélange de couleur avec le plan */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
   GL_ONE_MINUS_SRC_ALPHA);
tracer_scene();

/* Restauration des paramètres de tracé */
glEnable(GL_LIGHTING);
glDisable(GL_BLEND);
glDisable(GL_STENCIL_TEST);
```



Pour récapituler

- *Alpha-blending* : mélange de couleur au niveau pixel ;
- *Stencil buffer* : mécanisme pour conditionner finement l'affichage d'un fragment ;
- Exemples d'utilisation : miroir plan et ombres projetées ;
- Techniques récurrentes :
  - *Marquage* seul du stencil (masquage) ;
  - Calcul d'une matrice de projection ;
- Démarche similaire pour d'autres effets avancés (shadow volumes, etc.) ;

Conclusion

- OpenGL : API pour le rendu projectif interactif ;
- Limitations inhérentes au principe de rendu projectif ;
- **Machine à états !**
- Effets avancés : utilisation de l'alpha et du stencil ;
- Possibilités étendues avec les *shaders* ;
- OpenGL ne fait **que** du rendu !
- ... le plus dur reste à faire !