Applications multimédias
Modélisation et animation 3D
Opérations sur les sommets et les pixels

Frank Singhoff

Bureau C-202

Université de Brest, France

Lab-STICC

singhoff@univ-brest.fr

Introduction (1)

- **Modèle :** ensemble d'objets organisés pour réprésenter une scène devant être affichée. Chaque modèle est un ensemble de points (x,y,z).

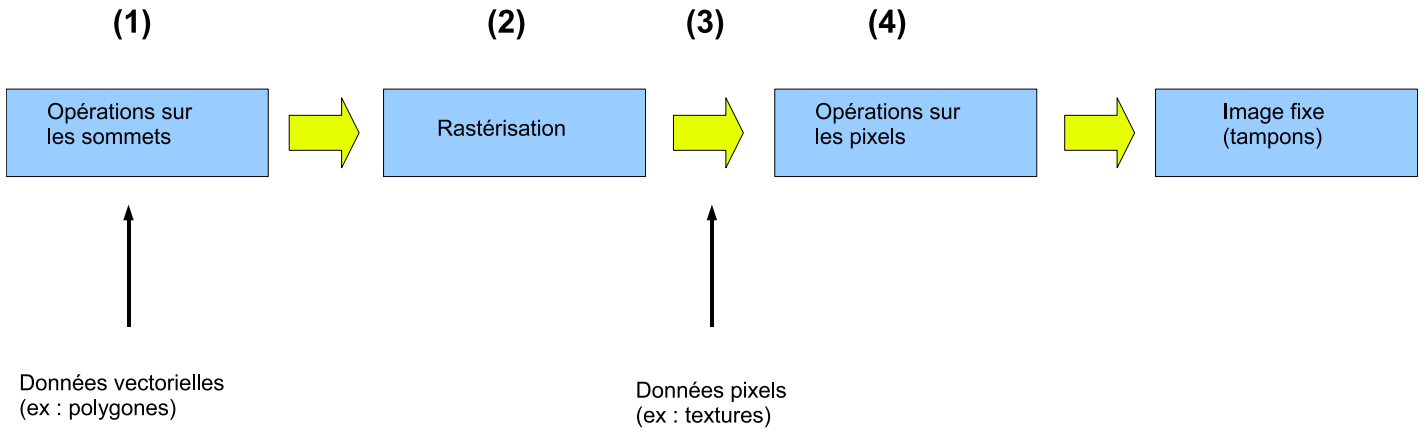
- **Processus/algorithme de synthèse/restitution :** analyse du modèle pour en obtenir une image fixe.

- **Image fixe :** résultat du processus de synthèse (de restitution). Ensemble de tampons.

- **Animation :** succession d'images fixes.

Introduction (2)

- **Processus de restitution :**



1. Traduction des sommets en matrices puis transformation de projection/perspective/cadrage, mise en couleur et calcul d'éclairage des polygones, ...
2. Traduction des données vectorielles en pixels.
3. Puis fusion avec les données "pixels".
4. Tests de profondeur, stencil, alpha, dithering (correction de dégradé), effet brouillard, ...

Sommaire

Opérations sur les sommets (1)

- **Principe du déplacement d'un objet :**
  - Soit un sommet $v = (x, y, z, 1)$ et un vecteur de translation $(tx, ty, tz, 1)$, les nouvelles coordonnées $v' = (x', y', z', 1)$ sont

$$x' = x + tx;$$
$$y' = y + ty;$$
$$z' = z + tz;$$

  - On cherche la matrice M, tel que :

$$v' = M.v$$

ou encore :

$$M = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Opérations sur les sommets (2)

- **Transformation :** opération sur les sommets afin de réaliser un effet graphique particulier (ex : déplacement d'un objet).

- **Etapes d'une transformation (ex : translation) :**

1. Soit un vecteur v contenant les coordonnées d'un sommet (x,y,z). On mémorise tous les points du modèle de cette façon.
2. Générer une **matrice de transformation** M 4x4. Cette matrice M mémorise le déplacement.
3. Sauver la matrice M dans la **matrice active**. Sauvegarde effectuée par la multiplication de M par la matrice active.
4. Multiplier le vecteur de chaque sommet du modèle par la matrice active. Si v est un sommet du modèle et Ma, la matrice active, alors $v' = Ma.v$ est le sommet transformé.
5. Afficher la scène avec ces nouveaux sommets v' ... l'objet est déplacé !

Opérations sur les sommets (3)

- **En pratique, on déplace le système de coordonnées et non les objets :**
  - Matrice active initiale, correspondant à un système de coordonnées initial:

$$Ma = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Vecteur de translation (tx, ty, tz) et sa matrice de transformation :

$$M = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Matrice active après translation, correspondant au système de coordonnées translaté :

$$Ma = Ma \cdot M = M$$

Opérations sur les sommets (4)

- **Cas des transformations multiples, si l'on souhaite effectuer n transformations :**

1. Générer autant de matrices M que de transformations.
2. Sauvegarder toutes les matrices générées en les multipliant les unes à la suite des autres avec la matrice active.
3. Multiplier le vecteur de chaque sommet du modèle par la matrice active. Si v est un sommet du modèle et Ma, la matrice active, alors $v' = Ma.v$ est le sommet transformé.
4. Afficher la scène avec ces nouveaux sommets v' ... l'objet est déplacé !

Opérations sur les sommets (5)

- **Exemple de deux transformations successives (rotation puis translation) :**

1. Soit la matrice $M1$ générée pour la rotation.
2. Soit la matrice $M2$ générée pour la translation.
3. Soit la matrice active Ma.
4. Application de la rotation : $Ma = Ma \cdot M1$
5. Puis application de la translation : $Ma = Ma \cdot M2$
6. Puis, pour chaque sommet v=(x,y,z), appliquer les deux transformations par:

$$v' = Ma \cdot v$$

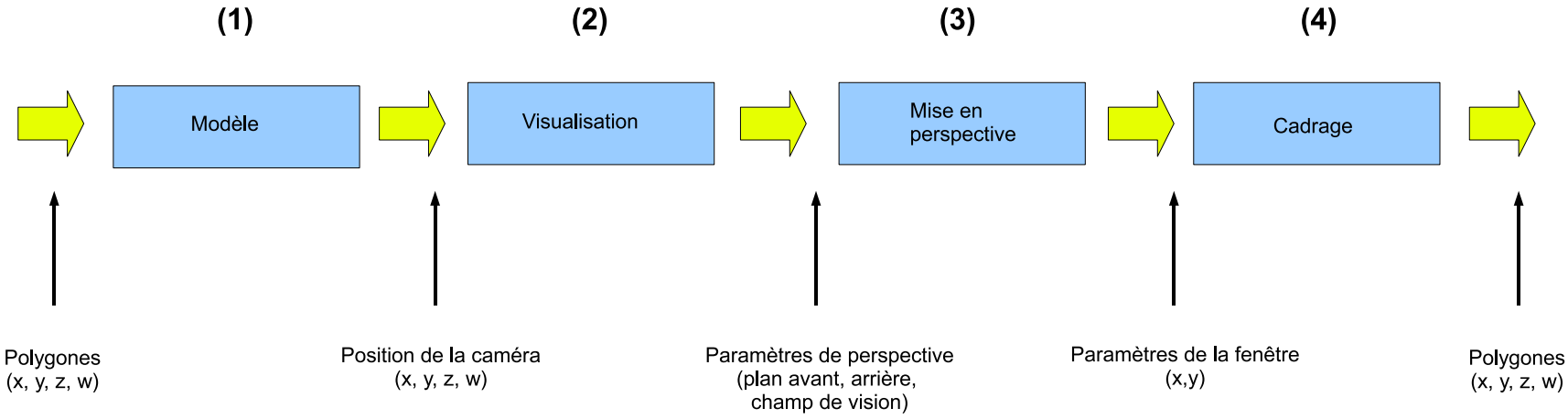
7. Afficher la scène avec les nouveaux sommets v' ... l'objet est déplacé !

Opérations sur les sommets (6)

- **Pourquoi des matrices 4x4 ?**
  - Notion de coordonnées homogènes : soit $v = (x, y, z)$ des coordonnées cartésiennes, les coordonnées homogènes de v sont $v' = (x', y', z', w)$ avec w, le dénominateur commun de x,y,z tel que $w >= 0$. (x, y, z) et $(x'/w, y'/w, z'/w)$ désignent donc le même sommet.

- **Pourquoi les coordonnées homogènes ?**
  - Permettre l'implantation des transformations par algèbre linéaire.
  - Moins de flottants = + rapide et + précision.
  - Moins de divisions = plus de stabilité numérique. Exemple : $(x1, y1, z1) -> (x1/3, y1/3, z1/3)$ devient $(x1, y1, z1, 1) -> (x1, y1, z1, 1 * 3)$.

Opérations sur les sommets (7)



- Quels types de transformation ?

1. **Transformation de modèle** : composition de la scène (taille et position des objets) + animation (déplacement des objets).
2. **Transformation de visualisation** : position de l'oeil/caméra.
3. **Transformation de projection** : forme du volume à visionner ainsi que les objets inclus dans ce volume
4. **Transformation de cadrage** : taille de la fenêtre de visualisation.

Transformations de modèle (1)

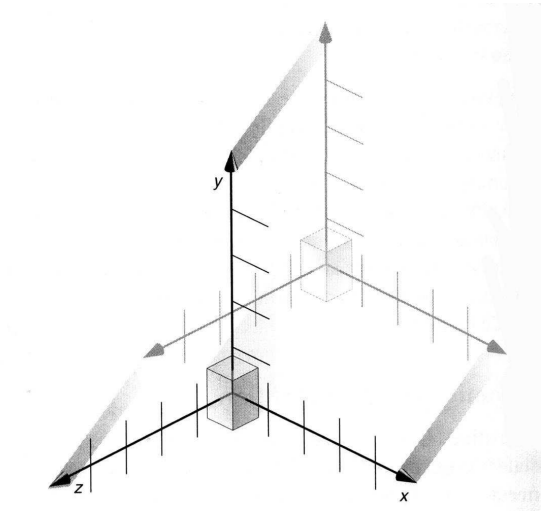
- Transformation qui consiste à déplacer un objet. Le déplacement d'un objet est en fait une modification **du système de coordonnées de la scène**.

- Types de modification sur le système de coordonnées :

1. La translation ($glTranslate$),
2. La rotation ($glRotate$),
3. La mise à l'échelle ($glScale$).

Transformations de modèle (2)

- **Translation :** gènére une matrice M de translation puis multiplie cette matrice avec la matrice active.



- **Primitive :**

```
glTranslatef(GLfloat tx, GLfloat ty, GLfloat tz);
```

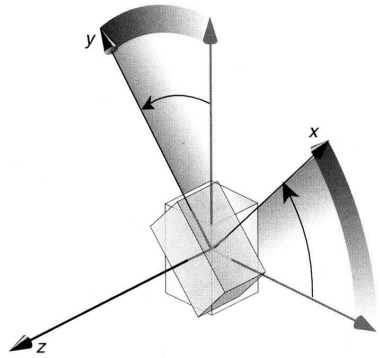
Applique sur le système de coordonnées un déplacement selon le vecteur de translation (tx, ty, tz).

- **Exemple :** déplacement sur l'axe des x.

```
glTranslatef(10, 0, 0);
```

Transformations de modèle (3)

- **Rotation :** gènére une matrice M de rotation puis multiplie cette matrice avec la matrice active.



- **Primitive :**

```
glRotatef(GLfloat angle, rx, ry, rz);
```

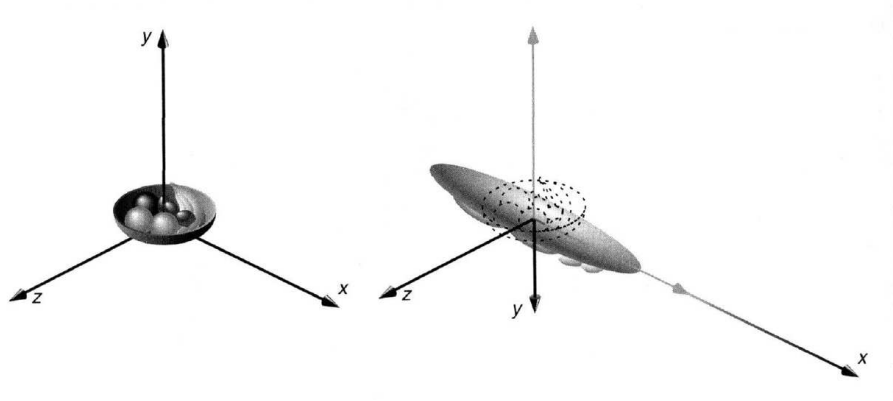
Applique sur le système de coordonnées une rotation d'angle $angle$ sur les axes x, y et/ou z.

- **Exemple :** rotation de 45 degrés sur l'axe des z.

```
glRotatef(45, 0, 0, 1);
```

Transformations de modèle (4)

- **Mise à l'échelle :** gènére une matrice M de mise à l'échelle puis multiplie cette matrice avec la matrice active.



- **Primitive :**

```
glScalef(GLfloat sx, GLfloat sy, GLfloat sz);
```

Elargit, rétrécit ou réfléchit le système de coordonnées sur les axes x, y et/ou z. Elargit sur x/y/z si sx/sy/sz>1. Rétrécit si sx/sy/sz<1. Réfléchit si sx/sy/sz=-1.

- **Exemple :** double la taille du système de coordonnées.

```
glScalef(2,2,2);
```

Transformations de modèle (5)

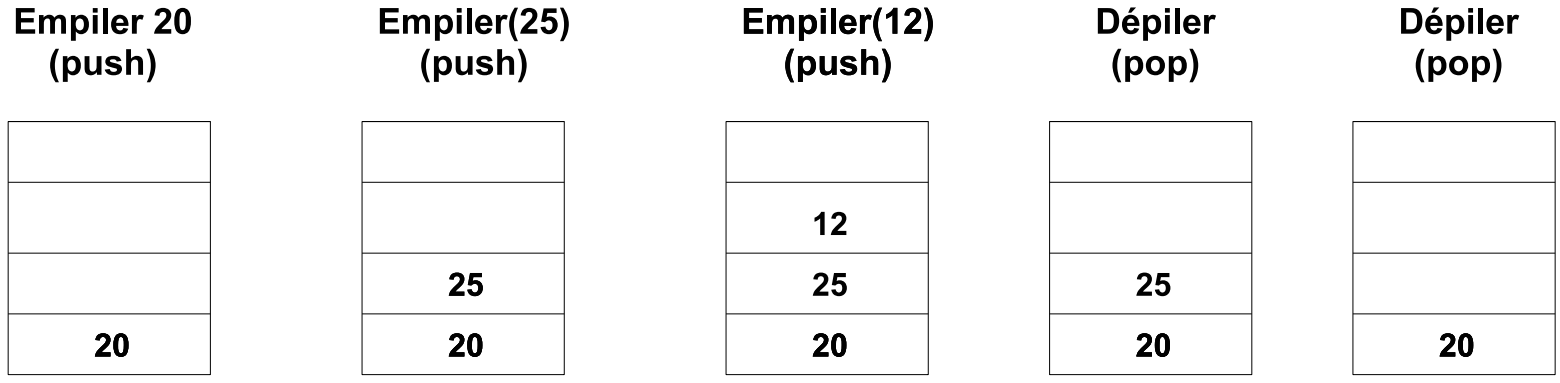
- **Exemple :**

```
float xRotation = 0.0;

void display(void) {
   glClear (GL_COLOR_BUFFER_BIT);
   glClear (GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

   xRotation = xRotation+0.1;
   glRotatef(xRotation, 1, 0, 0);
   glBegin(GL_QUADS);
      glVertex3f (0, 0, 0);
      glVertex3f (1, 0, 0);
      glVertex3f (1, 1, 0);
      glVertex3f (0, 1, 0);
   glEnd();

   glutSwapBuffers();
}
```

Transformations de modèle (6)

- La matrice active peut être sauvegardée dans une pile de matrice.

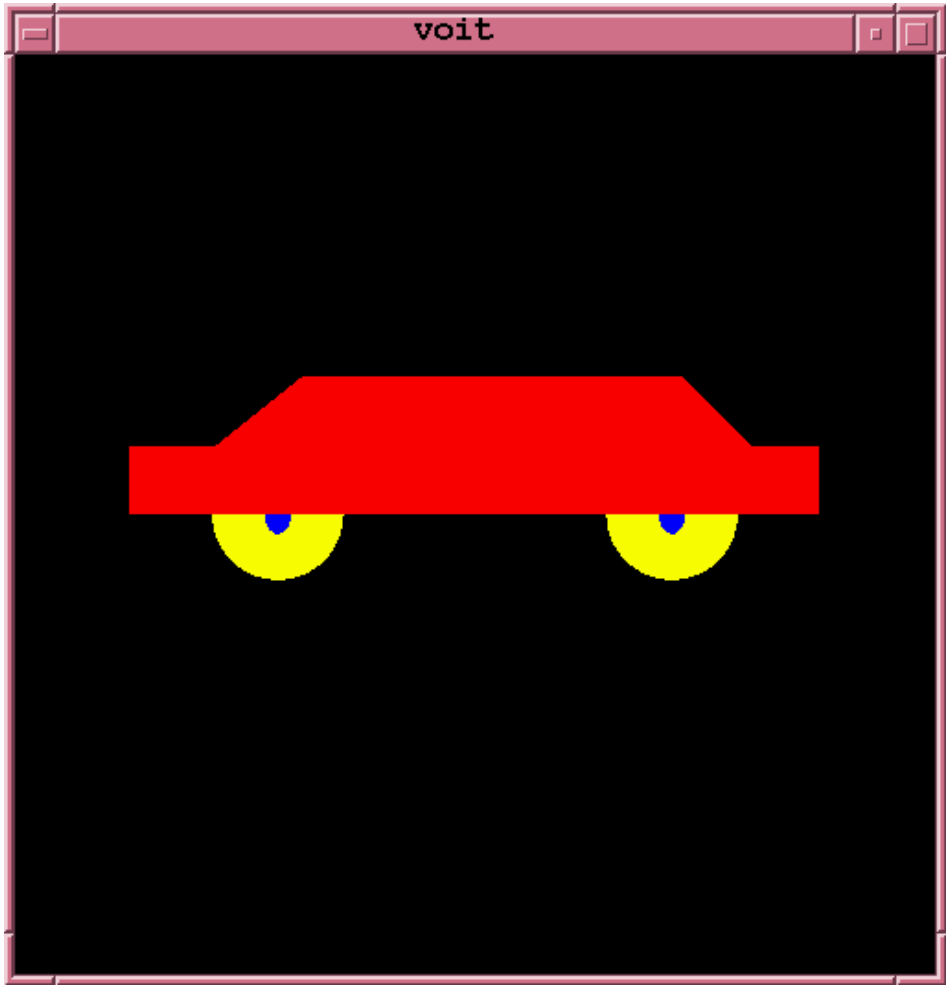
- **Pile :** tableau dont l'ajout et la suppression sont gérés comme une pile d'assiettes.



- **Primitives :**
  - $glLoadIdentity$: initialise la matrice active.
  - $glPushMatrix$: empile la matrice active courante.
  - $glPopMatrix$: dépile et écrase la matrice active.

Transformations de modèle (7)

- Modélisation **hiérarchique** de scène :



Transformations de modèle (8)

- **Dessiner une roue :**

```
// Cercle centré sur l'origine
void cercle(float r, float g, float b)  {
   glColor3f (r, g, b);
   glBegin(GL_POLYGON);
   for (i = 0; i <= 60; i++) {
            glVertex3f(x[i], y[i], 0);
   }
   glEnd();
}

void dessine_roue() {
   cercle(1.0, 1.0, 0.0);
   glPushMatrix();
   glTranslatef(0,0,.001);
   glScalef(.2, .3, .2);
   cercle(0, 0, 1);
   glPopMatrix();
}
```

Transformations de modèle (9)

- **Dessiner la voiture :**

```
void voiture()
{
   glColor3f (1, 0, 0);
   glBegin(GL_POLYGON);
      glVertex3f (0, 0, 0);
      glVertex3f (0, 1, 0);
      glVertex3f (2, 1, 0);
      glVertex3f (2.5, 2, 0);
      glVertex3f (8, 2, 0);
      glVertex3f (9, 1, 0);
      glVertex3f (10, 1, 0);
      glVertex3f (10, 0, 0);
   glEnd();
}
```

Transformations de modèle (10)

- **Fonction d'affichage :**

```
void display(void) {
   glLoadIdentity ();
   glClear (GL_COLOR_BUFFER_BIT);
   glClear (GL_DEPTH_BUFFER_BIT);

   glPushMatrix();
   glTranslatef(-5,0,0);
   voiture();
   glPopMatrix();

   glPushMatrix();
   glTranslatef(-3,0,-0.5);
   dessine_roue();
   glPopMatrix();
   glTranslatef(3,0,-0.5);
   dessine_roue();
   glutSwapBuffers();
}
```

Transformations de modèle (11)

- Il faut terminer la fonction d'affichage sur une pile de modélisation vide.
- Affichage de la taille d'une pile :

```
void etat()
{
        int val;
        glGetIntegerv(GL_MODELVIEW_STACK_DEPTH,&val);
        printf("taille pile = %d\n", val);
}
```

- Ne pas dépasser la taille maximale des piles de modélisation/visualisation (état $GL\_MAX\_MODELVIEW\_STACK\_DEPTH$) et de projection (état $GL\_MAX\_PROJECTION\_STACK\_DEPTH$).

Transformations de visualisation (1)

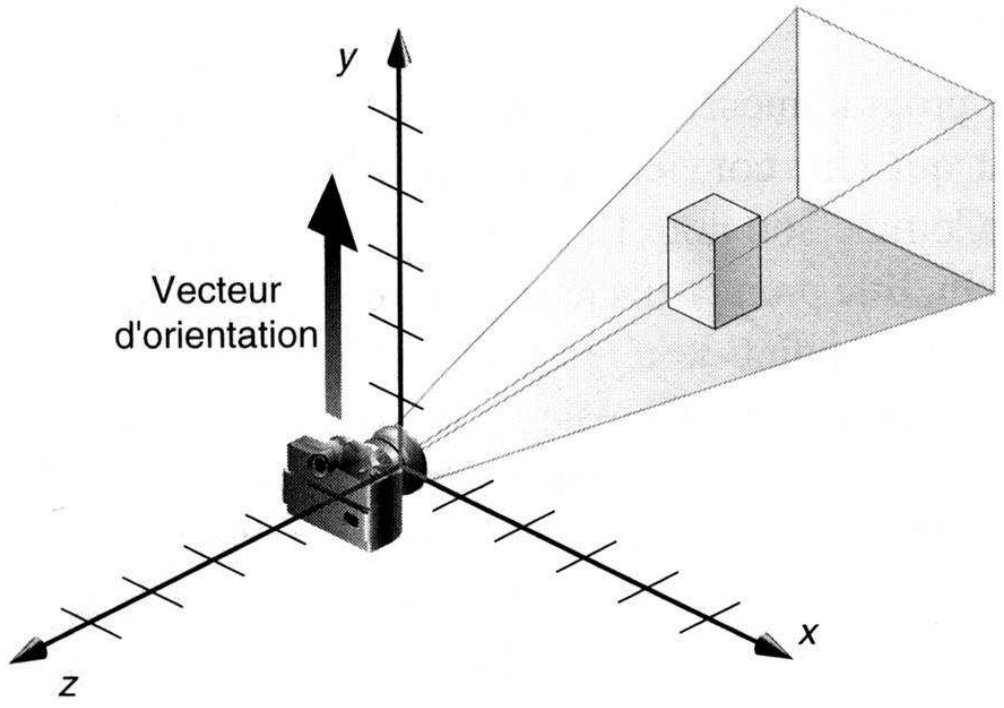
- Transformation qui consiste à définir la position et l'orientation de la caméra.

- Primitive :

```
gluLookAt(ex,ey,ez,   rx,ry,rz,   ox,oy,oz);
```

  - (ex, ey, ez) désigne la position de la caméra.
  - (rx, ry, rz) désigne le cadrage : c'est à dire le centre de la scène.
  - (ox, oy, oz) désigne l'orientation de la caméra vers le haut du volume visionné.
  - $gluLookAt$ génère une matrice M appliquant un positionnement de la caméra, puis multiplie la matrice active avec M.

Transformations de visualisation (2)

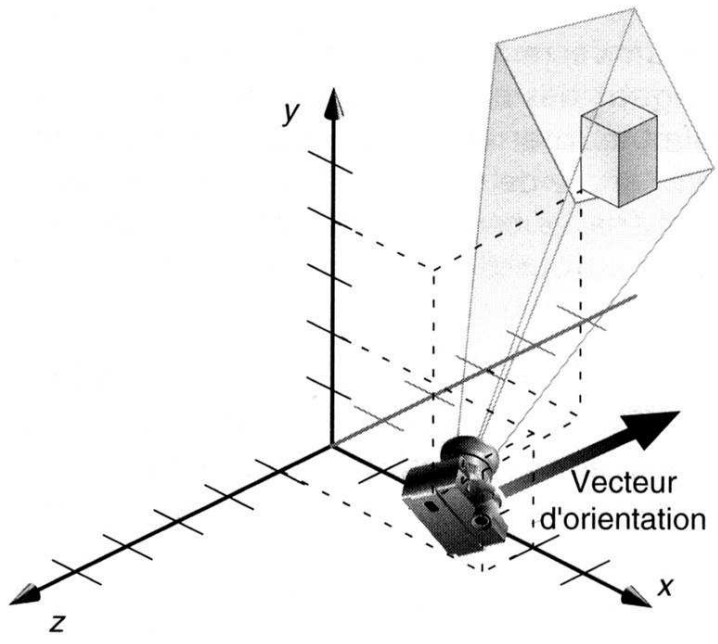


- Position par défaut :

```
gluLookAt(0,0,0,   0,0,-1,   0,1,0);
```

- Attention : la caméra est orientée vers l'axe négatif des z ; on voit donc le dos de la caméra.

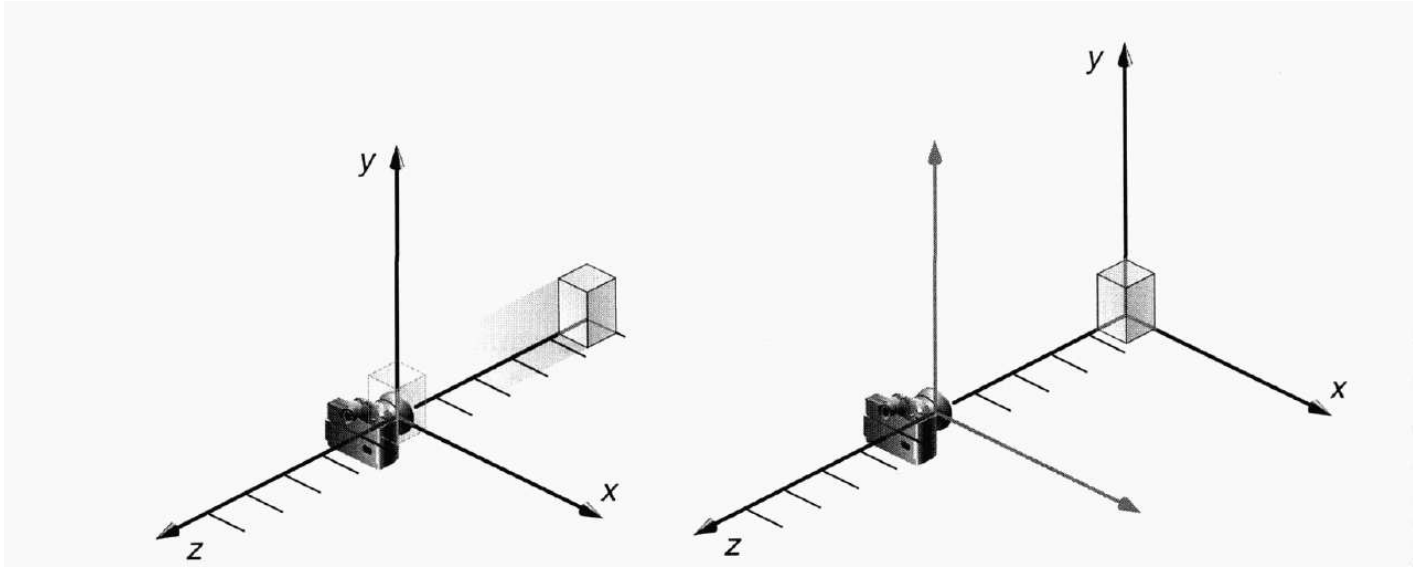
Transformations de visualisation (3)



- Exemple : la caméra est placée en (4,2,1). L'objectif est tourné vers l'objet et donc vers (2,4,-3). Un vecteur d'orientation a été choisi (2,2,-1) afin d'orienter le point de vue sur un angle de 45 degrés.

```
gluLookAt(4,2,1,      2,4,-3,      2, 2,-1);
```

Transformations de visualisation (4)



- Est ce l'objet qui s'éloigne ou la caméra ?
- Transformation de visualisation et de modèle constituent en fait, les mêmes opérations sur les sommets, d'où le partage de la même matrice.
- $gluLookAt$ n'est en fait qu'une combinaison de rotations et de translations.
- Attention : une seule transformation de visualisation doit être réalisée, et généralement avant les transformations de modélisation.

Transformation de projection (1)

- Transformation qui consiste à définir
  1. Quel est le volume visionné (clipping).
  2. Comment ce volume doit être projeté sur un plan.

- Comme pour les transformations de modélisation/visualisation, une projection, c'est un calcul matriciel.

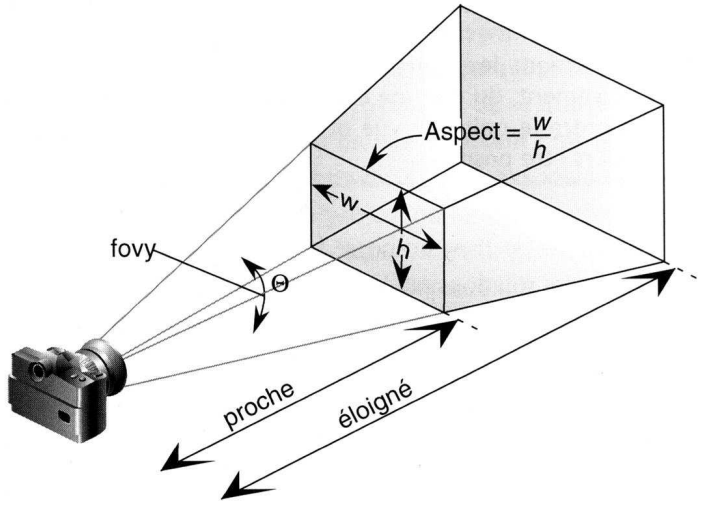
- Utilise une matrice différente : la matrice de projection.

- Opération sur la matrice de projection :

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
 ... opération de projection ...
glMatrixMode(GL_MODELVIEW);
```

- Projection par perspective conique versus perspective cavalière.

Transformation de projection (2)

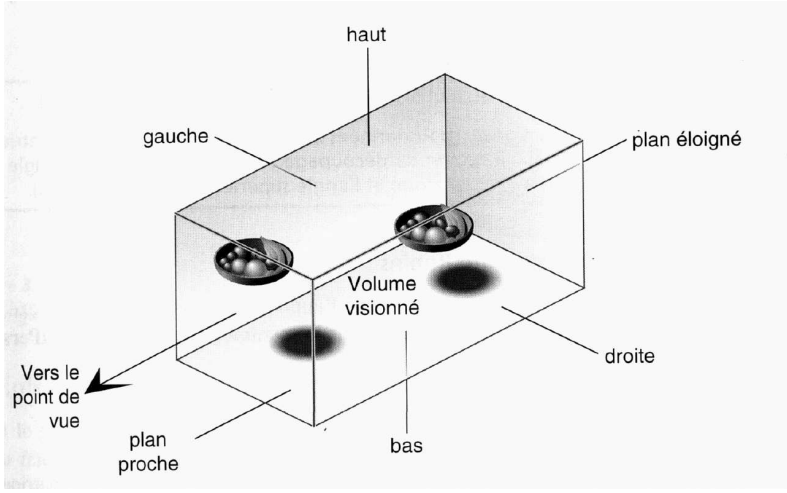


- **Perspective conique :** plus l'objet est éloigné et plus il est petit ; le volume est un frustum (tronc de pyramide).
- Deux primitives selon que le frustum soit symétrique avec un axe aligné sur l'axe des z ($gluPerspective$) ou non ($glFrustum$) :

```
gluPerspective(fovy, aspect, near, far);
```

$aspect$ est le ratio largeur/hauteur ; $near$ et far définissent les plans de clipping. $fovy$ est l'angle de vision dans le plan yz ($0 \leq fovy \leq 180$) ;

Transformation de projection (3)



- **Perspective cavalière :** chaque objet a une taille identique quelque soit sa profondeur. Peu utilisé.
- Primitive :

```
glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble near, GLdouble far);
```

$(left, bottom, near)$ et $(right, top, far)$ sont les plans de clipping. Parallèle à l'axe des z.

- Cas particulier de la perspective cavalière en 2D : $gluOrtho2D$.

Transformation de cadrage (1)

- Transformation qui consiste à définir la zone rectangulaire de la fenêtre ou sera restitué le dessin.

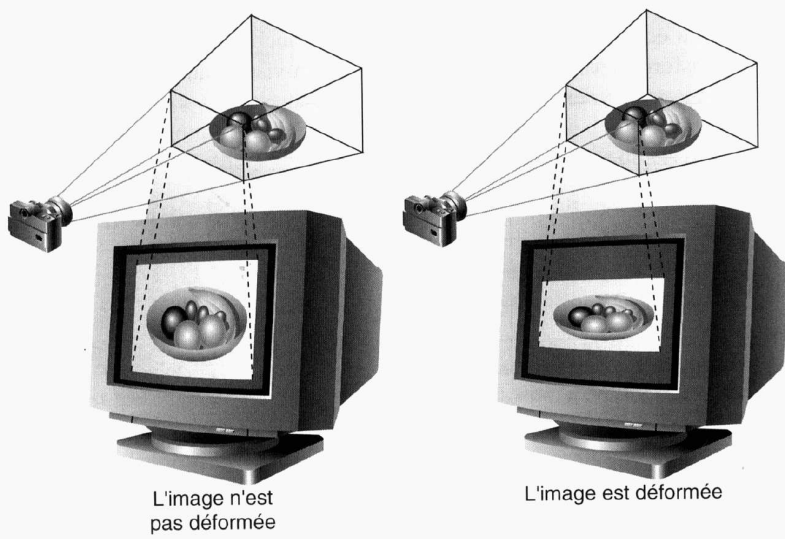
- Primitive :

```
glViewport(GLint x, GLint y, GLint w, GLint h);
```

(x, y) spécifie l'angle inférieur gauche du rectangle de cadrage et w et h la taille du rectangle de cadrage.

- Une fenêtre peut contenir plusieurs rectangles de cadrage.

Transformation de cadrage (2)



- Transformation de projection et de cadrage doivent être cohérentes pour éviter une déformation de l'image.
- Image déformée :

```
gluPerspective(angle, 1.0, proche, eloigne);
glViewport(0, 0, 400, 200);
```

- Image non déformée :

```
gluPerspective(angle, 2.0, proche, eloigne);
glViewport(0, 0, 400, 200);
```

Sommaire

1. Introduction.
2. Opérations sur les sommets.
3. Opérations sur les pixels.
4. Ce qu'il faut retenir.

Opérations sur les pixels (1)

- **Objets** = polyèdres = ensemble de facettes/polygones. Polygone = ensemble de sommets (x,y,z). Face avant et arrière.

- **Un polygone peut être :**
  - Dessiné en mode filaire (fonction $glPolygonMode$).
  - Rempli par un dégradé de couleur (une couleur peut être spécifiée par sommet).
  - Rempli par une couleur unique (cf. fonction $glColor3f$).
  - Rempli par une texture.
  - Accompagné d'un éclairage.

- Quelle "couleur" pour chaque face de chaque polygone ?

Opérations sur les pixels (2)

- Opérations sur les pixels:

1. Application de textures.
2. Calcul de l'éclairage.

Application de textures (1)

- **Texture =** tableau rectangulaire de données contenant, par exemple, des données chromatiques (valeurs appelées des $texels$). Ensemble de pixels.

- **Utilisation :**
  - Accélère la construction d'une scène (ex : mur).
  - Améliore la qualité d'une scène (moins de régularité).
  - Nombreux effets graphiques associés (ex : ombrage).
  - Application des transformations géométriques.

- Textures 1D, 2D voire 3D. On suppose ici, des textures 2D.
- Texture bitmap (ex : données scannées) ou procédurale.

Application de textures (2)

- **Méthode générale pour appliquer une texture sur un polygone :**

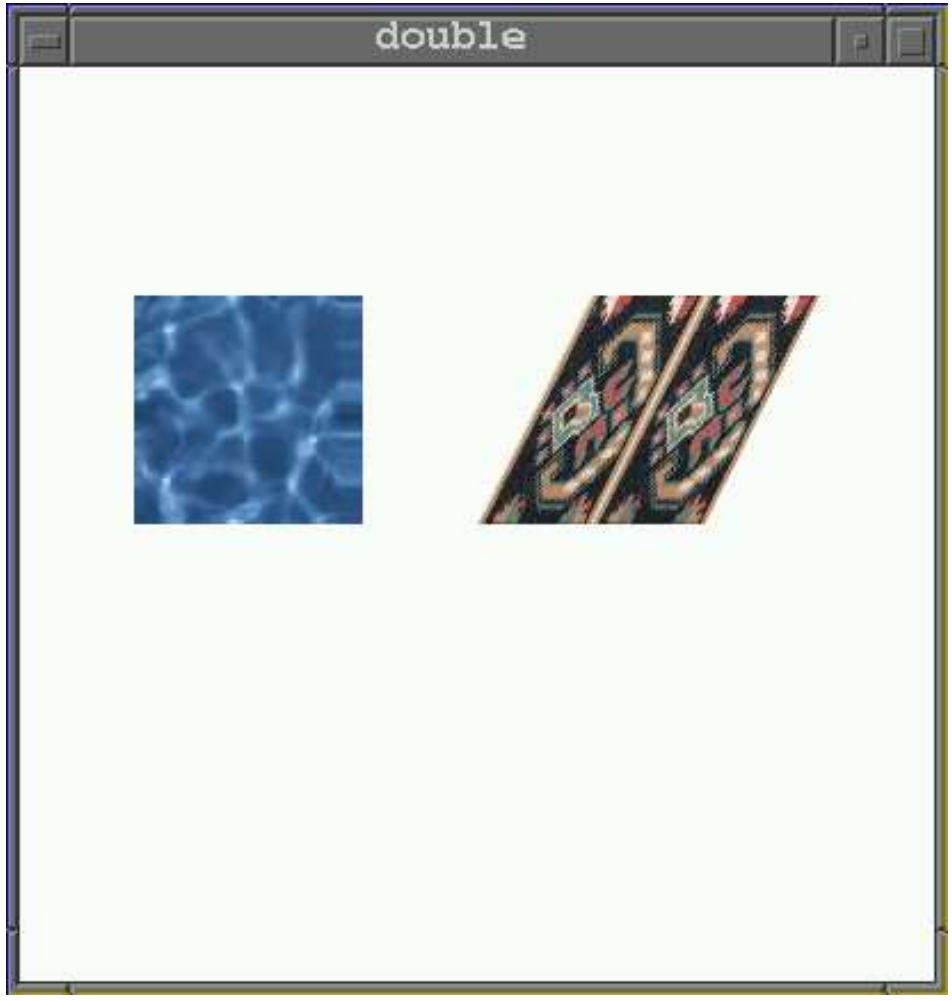
1. Activer les textures.
2. Définir un ou des objets de texture (définition des texels + comportement lors de l'application de la texture sur le polygone).
3. Lors de la description de scène :
   - Spécification de la texture courante + description géométrique de la scène.
   - Spécification des coordonnées de textures : indique les parties de la texture à "plaquer" sur le polygone.
   - Spécification de la combinaison chromatique (couleur polygone, texture, éclairage).

Application de textures (3)

- **Primitives :**
  - $glGenTextures$: allocation d'un ou plusieurs objets de texture.
  - $glTexImage2D$: création d'une texture à partir d'une zone de données.
  - $glBindTexture$: positionner une texture comme la texture courante.
  - $glTexParameteri$: paramétrage de la texture courante (méthode de placage).
  - $glTexEnvf$: paramétrage de la combinaison chromatique.

Application de textures (4)

- Structure d’un programme type :



Application de textures (5)

- Structure d'un programme type :

```
GLuint texName[2];

void init(void)
{
   ...
   glGenTextures(2, texName);

   glBindTexture(GL_TEXTURE_2D, texName[0]);
   glTexImage2D(...);
   glTexParameteri(...);

   glBindTexture(GL_TEXTURE_2D, texName[1]);
   glTexImage2D(...);
   glTexParameteri(...);

   glEnable(GL_TEXTURE_2D);
}
```

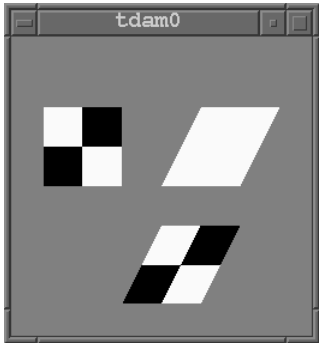
Application de textures (6)

```
void display(void)
{
   ...
   glTexEnvf(...);
   ...
   glBindTexture(GL_TEXTURE_2D, texName[1]);
   glBegin(GL_POLYGON);
   glTexCoord2f(...); glVertex3f(0.0, 0.0, 0.0);
   glTexCoord2f(...); glVertex3f(1.0, 0.0, 0.0);
   glTexCoord2f(...); glVertex3f(1.5, 1.0, 0.0);
   glTexCoord2f(...); glVertex3f(0.5, 1.0, 0.0);
   glEnd();
   ...
   glBindTexture(GL_TEXTURE_2D, texName[0]);
   glBegin(GL_POLYGON);
   glTexCoord2f(...); glVertex3f(1.5, 1.0, 0.0);
   ...
   glEnd();
}
```

Coordonnées de textures (1)

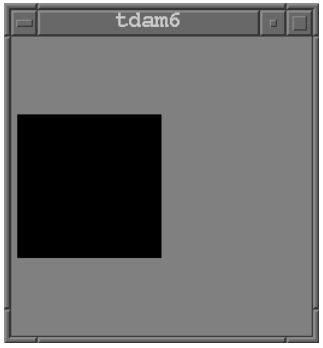
- Comment indiquer où et quels sont les texels d'une texture qui doivent être plaqués sur un polygone ?
- Les texels peuvent être référencés selon leurs coordonnées grâce à la primitive $glTexCoord$.
- Exemple des textures à 2 dimensions (S,T) :
  - Axe S = abscisse. Axe T = ordonnée.
  - (0,0) désigne le coin bas/gauche.
  - (0,1) désigne le coin haut/gauche.
  - (1,1) désigne le coin haut/droit.
  - (1,0) désigne le coin bas/droit.
  - Toutes les coordonnées dans l'intervalle]0,1[permettent d'extraire une partie de la texture.
  - Toutes les coordonnées hors de l'intervalle [0,1] permettent de resserrer la texture.

Coordonnées de textures (2)



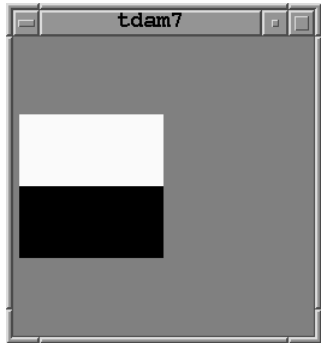

```
glBindTexture(GL_TEXTURE_2D, texName[0]);
glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);
   glTexCoord2f(0.0, 1.0); glVertex3f(1,2,0);
   glTexCoord2f(1.0, 1.0); glVertex3f(3,2,0);
   glTexCoord2f(1.0, 0.0); glVertex3f(2,0,0);
glEnd();
```

Coordonnées de textures (3)



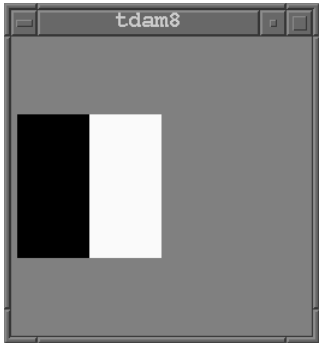

```
glBindTexture(GL_TEXTURE_2D, texName[0]);
glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);
   glTexCoord2f(0.0, 0.5); glVertex3f(0,1,0);
   glTexCoord2f(0.5, 0.5); glVertex3f(1,1,0);
   glTexCoord2f(0.5, 0.0); glVertex3f(1,0,0);
glEnd();
```

Coordonnées de textures (4)




```
glBindTexture(GL_TEXTURE_2D, texName[0]);
glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);
   glTexCoord2f(0.0, 1.0); glVertex3f(0,1,0);
   glTexCoord2f(0.5, 1.0); glVertex3f(1,1,0);
   glTexCoord2f(0.5, 0.0); glVertex3f(1,0,0);
glEnd();
```

Coordonnées de textures (5)




```
glBindTexture(GL_TEXTURE_2D, texName[0]);
glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);
   glTexCoord2f(0.0, 0.5); glVertex3f(0,1,0);
   glTexCoord2f(1.0, 0.5); glVertex3f(1,1,0);
   glTexCoord2f(1.0, 0.0); glVertex3f(1,0,0);
glEnd();
```

Paramétrage de textures (1)

- Que faire quand le polygone n'est pas un quadrilatère ?
- Comment rétrécir une texture ?
- Comment combiner les informations chromatiques d'une texture avec la couleur du polygone, de l'éclairage ?

$\Longrightarrow$ Deux paramètrages possibles :

1. Paramétrage d'objet de texture (technique de placage avec la primitive $glTexParameteri$).
2. Paramétrage lors de la combinaison chromatique avec la primitive $glTexEnvf$.

Paramétrage de textures (2)

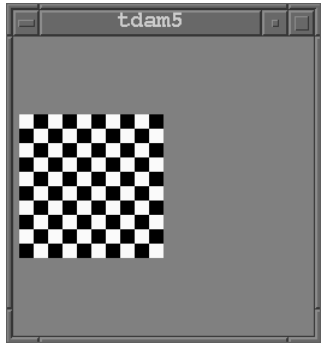
- **Paramétrer le placage, c'est :**

1. Définir comment on effectue un rétrécissement de texture ($GL\_TEXTURE\_WRAP\_S$, $GL\_TEXTURE\_WRAP\_T$, lorsque les coordonnées sont au delà de la plage [0,1]) :
   - Par répétition de texels (commande GL_REPEAT).
   - Par extrapolation de texels (commande GL_CLAMP).

2. Définir comment on adapte la forme de la texture à la géométrie du polygone ($GL\_TEXTURE\_MIN\_FILTER$, $GL\_TEXTURE\_MAG\_FILTER$).

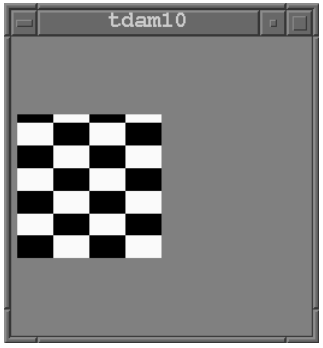
$\Longrightarrow$ utilisation de la primitive $glTexParameteri$.

Paramétrage de textures (3)



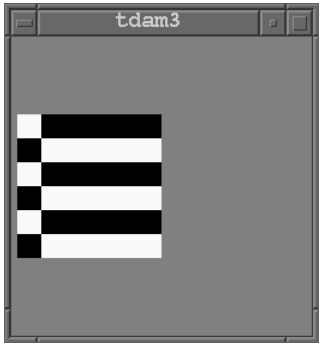

```
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 5.0); glVertex3f(0, 1, 0);
glTexCoord2f(5.0, 5.0); glVertex3f(1, 1, 0);
glTexCoord2f(5.0, 0.0); glVertex3f(1, 0, 0);
glEnd();
```

Paramétrage de textures (4)



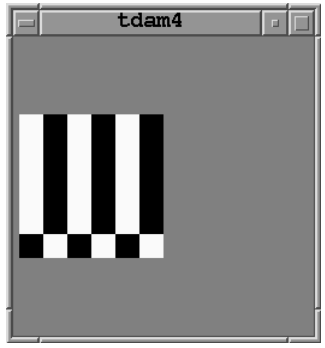

```
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.2); glVertex3f(0, 1, 0);
glTexCoord2f(2.0, 3.2); glVertex3f(1, 1, 0);
glTexCoord2f(2.0, 0.0); glVertex3f(1, 0, 0);
glEnd();
```

Paramétrage de textures (5)



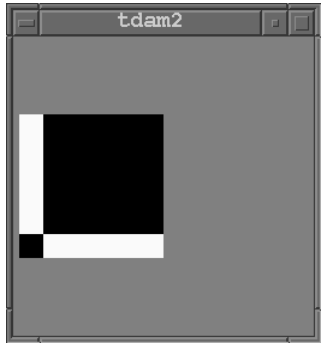

```
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);
glEnd();
```

Paramétrage de textures (6)




```
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_CLAMP);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);
glEnd();
```

Paramétrage de textures (7)

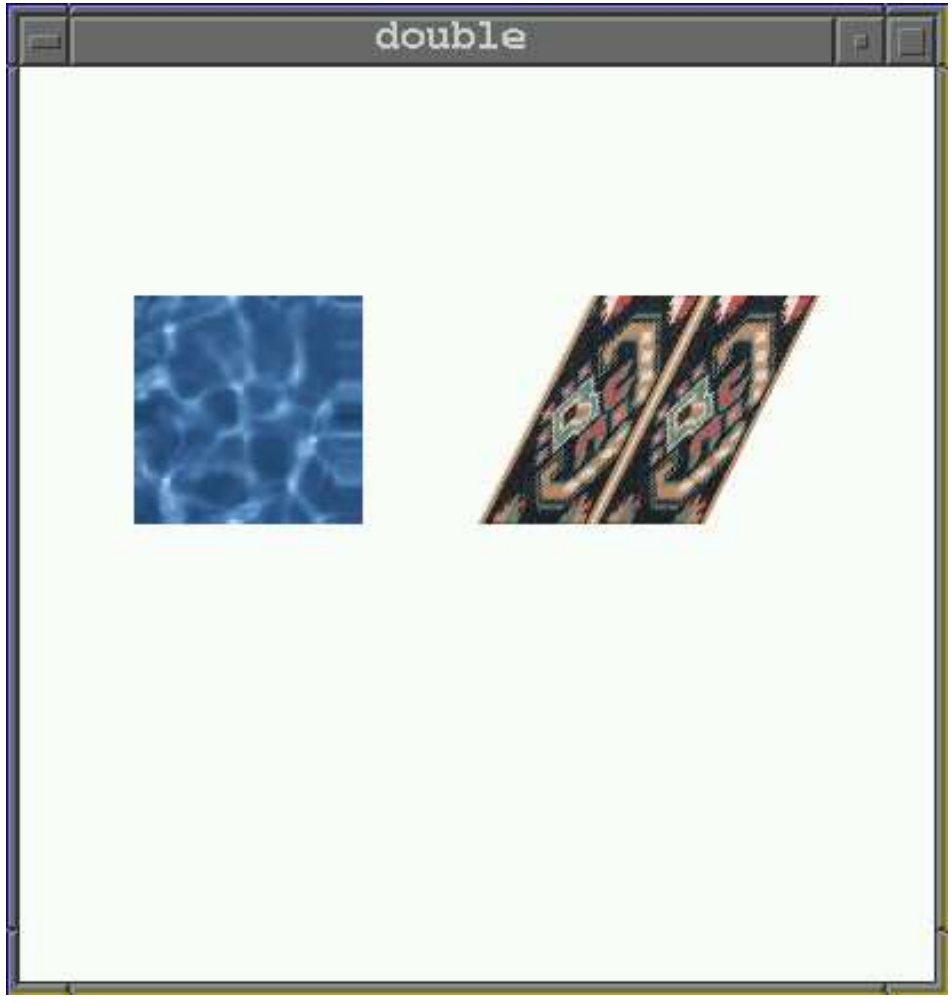



```
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_CLAMP);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);
glEnd();
```

Paramétrage de textures (8)

- Paramétrer la combinaison chromatique :
  - Réalisée par la primitive $glTexEnv$.
  - Mise à jour du tampon chromatique :
    1. Par écrasement ($GL\_REPLACE$).
    2. Par addition (GL_ADD).
    3. Par combinaison/multiplication ($GL\_MODULATE$).

Exemple complet (1)



Exemple complet (2)

```
static GLuint texName[2];
GLubyte* data;
int width, height;

void init(void)
{
   glGenTextures(2, texName);

   data=glmReadPPM("water20.ppm", &width , &height);
   glBindTexture(GL_TEXTURE_2D, texName[0]);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                   GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                   GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
                height, 0, GL_RGB, GL_UNSIGNED_BYTE,
                data);
```

Exemple complet (3)

```
    data=glmReadPPM("carpet12.ppm", &width , &height);
    glBindTexture(GL_TEXTURE_2D, texName[1]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                    GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                    GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
                 height, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 data);

    glEnable(GL_TEXTURE_2D);
}
```

Exemple complet (4)

```
void display(void)
{
   glLoadIdentity ();
   gluLookAt (ex,ey,ez,   0,0,-100.0, 0.0, 1.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);
   glClear (GL_DEPTH_BUFFER_BIT);

   glTranslatef(-1.5,0,0);
   glColor3f(1,1,0);

   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

   glBindTexture(GL_TEXTURE_2D, texName[0]);
   glPushMatrix();
   glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
   glTexCoord2f(0.0, 1.0); glVertex3f(0, 1, 0);
   glTexCoord2f(1.1, 1.0); glVertex3f(1, 1, 0);
   glTexCoord2f(1.1, 0.0); glVertex3f(1, 0, 0);
   glEnd();
```

Exemple complet (5)

```
    glBindTexture(GL_TEXTURE_2D, texName[1]);
    glTranslatef(1.5,0,0);
    glBegin(GL_POLYGON);
    glTexCoord2f(0, 0); glVertex3f(0, 0, 0);
    glTexCoord2f(0, 1); glVertex3f(0.5, 1, 0);
    glTexCoord2f(2, 1); glVertex3f(1.5, 1, 0);
    glTexCoord2f(2, 0); glVertex3f(1, 0, 0);
    glEnd();


    glutSwapBuffers();
}
```

Opérations sur les pixels (2)

- Opérations sur les pixels:

1. Application de textures.
2. Calcul de l'éclairage.

Eclairage (1)

- Une scène éclairée, c'est : des **sources** de lumière et des objets constitués d'une **matière**.
- L'existence d'une lumière n'a de sens que s'il existe une surface qui va l'absorber ou la réflechir.

- **Démarche générale d'approximation de la lumière :**
  - La lumiere est décomposée en quatre parties indépendantes : les lumières ambiantes, diffuses, spéculaires et émissives.
  - OpenGL fait le calcul d'éclairage pour chaque partie compte tenu de la matière des objets. Ces quatre parties sont finalement ajoutées pour déterminer la lumière totale à restituer.
  - Chaque lumière ou matière est définie par les composantes (R,V,B,A).

Eclairage (2)

1. **Lumière ambiante :** lumière qui a été tant dispersée que sa source n'est plus déterminable : elle semble venir de partout.
2. **Lumière diffuse :** lumière qui vient d'une direction particulière. Lumière plus brillante si elle atteint directement une surface que si elle l'effleure. Cette lumière est dispersée de façon égale dans toutes les directions lorsqu'elle touche une surface.
3. **Lumière spéculaire :** lumière qui vient d'une direction particulière et qui rebondit dans une direction donnée. Lumière proche de la brillance. Provoque l'apparition d'un reflet.
4. **Lumière émissive :** Lumière émisse par un objet. Augmente la sensation de brillance d'un objet mais n'éclaire pas vraiment la scène.

Eclairage (3)

- **Structure d'un programme :**

1. Choisir les paramètres généraux d'éclairage (primitive $glLightModel$).
2. Définir les sources lumineuses ainsi que leurs composantes en lumière diffuse, ambiante et spéculaire (primitive $glLight$).
3. Définir les propriétés de matière : réflexion et émission de lumière (primitive $glMaterial$).
4. Lors de la description de scène, définir pour chaque objet :
   - Les lumières éclairant l'objet.
   - La matière qui la compose.
   - Description de la géométrie de l'objet incluant les vecteurs de direction (normales de surfaces).

Eclairage (4)

- **Les normales de surface :**
  - Vecteur pointant dans la **direction perpendiculaire** à une surface. Définit l'orientation du polygone selon la source de lumière (et donc la quantité de lumière reçue).
  - Une normale est définie pour chaque sommet par la primitive $glNormal3f$.
  - A partir des normales des sommets qui définissent la quantité de lumière reçue, les autres sommets du polygone sont extrapolés.
  - Un polygone dans un plan peut avoir un seul vecteur normal ; ce qui n'est pas le cas pour un polygone courbé.
  - Les vecteurs doivent rester normalisés => taille 1 (attention aux $glScale$).

Eclairage (5)

- **Exemple (avec textures) :**

```
glEnable(GL_RESCALE_NORMAL); // pour glScale
...
// Pour combiner l'éclairage avec une texture
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_MODULATE);
...
// Une normale par sommet, on peut se contenter
// d'une seule normale pour ce polygone car il est sur un plan
glBegin(GL_QUADS);
glNormal3f(0.0,0.0,1.0);
glTexCoord2f(0.0, 0.0); glVertex3f(0.0, 0.0, 0.0);
glNormal3f(1.0,0.0,1.0);
glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 0.0, 0.0);
glNormal3f(1.0,1.0,1.0);
glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
glNormal3f(0.0,1.0,1.0);
glTexCoord2f(1.0, 0.0); glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

Eclairage (6)

- **Paramétrage global de l'éclairage (primitive $glLightModelv$):**

  1. Doit on calculer l'éclairage sur les deux faces ? (commande $GL\_LIGHT\_MODEL\_TWO\_SIDE$)
  2. Où se situe le point de vue ? (commande $GL\_LIGHT\_MODEL\_LOCAL\_VIEWER$)
  3. Quelle intensité pour la lumière ambiante générale ? (commande $GL\_LIGHT\_MODEL\_AMBIENT$)

- **Exemple :**

```
GLfloat general[] = { 0.2, 0.2, 0.2, 1.0 };

glEnable(GL_LIGHTING);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, general);
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
```

Eclairage (7)

- **Définir une source de lumière, c'est :**
  - Intensité et couleur de la lumière émise (ambiante, diffuse et spéculaire) décrite composante de couleur par composante de couleur.
  - Position de la lumière (cas $w = 1$, définition d'une lumière directionnelle sinon). Attention aux transformations de modélisation/visualisation (impact sur la matrice de modélisation).
- Plusieurs sources possibles (LIGHT0, LIGHT1, ... LIGHT7) définies indépendamment.
- La lumière totale à restituer est calculée en ajoutant les composantes RVB de toutes les sources de lumière.

Eclairage (8)

- **Primitive** $glLightfv$ **:**
  1. $GL\_AMBIENT$: intensité/couleur de la lumière ambiante.
  2. $GL\_DIFFUSE$: intensité/couleur de la lumière diffuse.
  3. $GL\_SPECULAR$: intensité/couleur de la lumière spéculaire.
  4. $GL\_POSITION$: position de la source lumineuse.

- Exemple :

```
GLfloat light_ambient[] = { 0.4, 0.4, 0.4, 0 };
GLfloat light_diffuse[] = { 0.3, 0.3, 0.3, 0 };
GLfloat light_specular[] = { .9, .9,.9, 0 };
GLfloat light_position[] = { -7.0, 2.0, 5.0, 1.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glEnable(GL_LIGHT0);
```

Eclairage (9)

- **Définir les propriétés de la matière d'un objet :**
  - Comment une matière réagit quand elle est soumise à une lumière ? => lumière absorbée et réfléchie.
  - Quantité/pourcentage de lumière réfléchie (ambiante, diffuse et spéculaire) décrite composante de couleur par composante de couleur.
  - Un objet peut émettre de lui même une certaine intensité lumineuse.

- Ces coefficients sont ajoutés entre toutes les sources de lumière.

Eclairage (10)

- **Quelle est la couleur d'un objet éclairé ?**
  - Avec une lumière (LR, LV, LB) et une matière (MR, MV, MB), on obtient un objet de couleur (LR.MR, LV.MV, LB.MB).
  - Réflectivité ambiante et diffuse sont généralement proches voire identiques. **Définissent la couleur de la matière.**
  - Réflectivité spéculaire est blanche ou grise. La couleur du reflet est la couleur de la lumière.
- **Exemple :**
  - Par une lumière blanche, une balle rouge est une balle qui réflechit tout le rouge et pas de vert ni de bleu.
  - Par une lumière verte, la même balle sera noire.

Eclairage (11)

- Chaque composante exprime **le pourcentage de lumière réfléchie**.

Exemple : (1,0.5,0.5) signifit que 100% du rouge, 50% du bleu et 50% du vert sont réfléchis.

- Primitive $glMaterialfv$:

1. $GL\_AMBIENT$: taux de réflection de la lumière ambiante.
2. $GL\_DIFFUSE$: taux de réflection de la lumière diffuse.
3. $GL\_SPECULAR$: taux de réflection de la lumière spéculaire.
4. $GL\_EMISSION$: intensité de la lumière émise par l'objet.
5. $GL\_SHININESS$: taille du reflet (compris dans [0..100]).

Eclairage (12)

- **Exemple :**

```
GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat mat_diffuse[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat mat_specular[] = { 1, 1, 1, 1 };
GLfloat mat_shininess[] = { 70.0 };
GLfloat mat_emission[] = {0, 0, 0, 0};

glEnable(GL_LIGHTING);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
0lMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
```

Exemple complet (1)

- **Initialisations :**

```
// Propriétés de l'objet (sa matière)
GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat mat_diffuse[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat mat_specular[] = { 1, 1, 1, 1 };
GLfloat mat_shininess[] = { 70.0 };
GLfloat mat_emission[] = {0, 0, 0, 0};

// Propriétés de la source lumineuse
GLfloat light_ambient[] = { 0.4, 0.4, 0.4, 0 };
GLfloat light_diffuse[] = { 0.3, 0.3, 0.3, 0 };
GLfloat light_specular[] = { .9, .9,.9, 0 };
GLfloat light_position[] = { -7.0, 2.0, 5.0, 1.0 };

// Propriétés générales de l'éclairage
GLfloat general_light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
```

Exemple complet (2)

- **Fonction d'affichage :**

```
void display(void)
{
  glLoadIdentity ();
  gluLookAt (0,0,5, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glClear (GL_COLOR_BUFFER_BIT);
  glClear (GL_DEPTH_BUFFER_BIT);

  glEnable(GL_LIGHTING);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, general_light_ambient);
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glEnable(GL_LIGHT0);
```

Exemple complet (3)

- **Fonction d'affichage (suite) :**

```
  glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

  // Sphère jaune faiblement éclairée par une lumiere blanche
  draw_sphere(1  ,1,1,0);

  // Sphère rouge sans éclairage
  glDisable(GL_LIGHTING);
  draw_sphere(1  ,1,0,0);

  glutSwapBuffers();
}
```

Sommaire

1. Introduction.
2. Opérations sur les sommets.
3. Opérations sur les pixels.
4. Ce qu'il faut retenir.

Ce qu'il faut retenir

- **Que sont les opérations sur les sommets ou transformations ?**
  - Une transformation = génération d'une matrice + multiplication avec la matrice active.
  - Transformations importantes : modélisation et visualisation.
  - Comment faire une description hiérarchique d'une scène.

- **Que sont les opérations sur les pixels ?**
  - Texture: incontournable, augmente le réalisme, facilite la description de scène.
  - Modèle d'éclairage : sources, objets et matières. Types de lumière. Couleur, intensité, position.