Transformations de scène avec OpenGl

Frank Singhoff

Bureau C-203

Université de Brest, France

LISyC/EA 3883

singhoff@univ-brest.fr

Sommaire

1. Introduction
2. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. Ce qu'il faut retenir

Introduction (1)

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

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

Introduction (2)

- **Etapes d'une opération/transformation sur sommets (ex : rotation, puis translation) :**

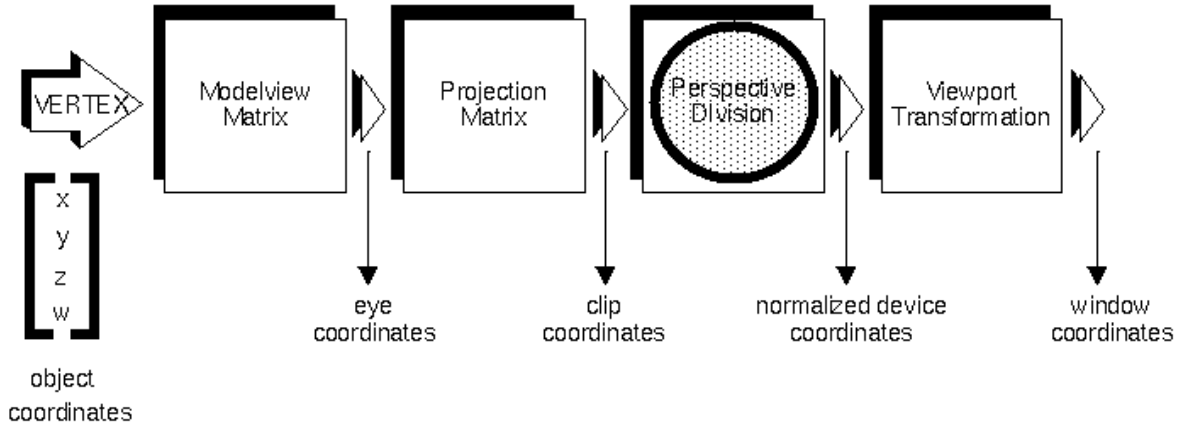
1. Soit un vecteur contenant les coordonnées d'un sommet. On mémorise tous les points du modèle de cette façon.
2. Générer une matrice 4x4 de **transformation**. Cette matrice mémorise les deux déplacements (2 matrices combinées, notion de **matrice active**).
3. Multiplier le vecteur de chaque sommet du modèle par la matrice de transformation. Si v est un sommet du modèle et M, une matrice de transformation, $v' = M.v$ est le sommet transformé.
4. Afficher la scène avec ces nouveaux sommets ... l'objet est déplacé !

Introduction (3)

- **Pourquoi des matrices 4x4 ?**
  - Notion de coordonnées homogènes : soit $v = (x, y, z)$ des coordonnées cartèsiennes, 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 $(x1, y1, z1, 1) -> (x1, y1, z1, 1 * 3)$.

Introduction (4)



- 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.

Sommaire

1. Introduction
2. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. Ce qu'il faut retenir

Transformations de modèle (1)

- Transformation qui consiste à **déplacer un objet** ou **le système de coordonnées de la scène**.

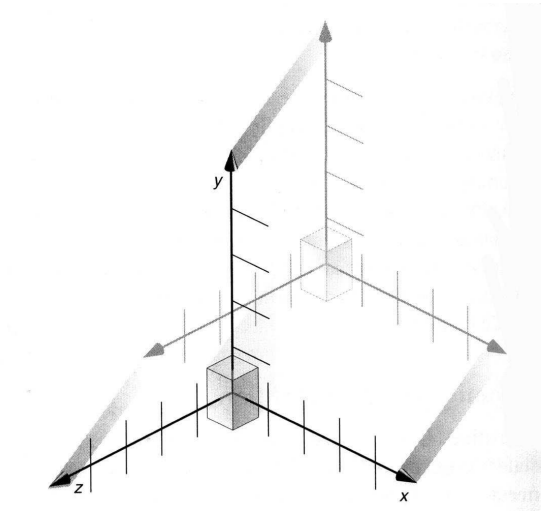
- Primitives/types de déplacement :

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

- Notion de **matrice active** : matrice qui combine toutes les transformations demandées.

Transformations de modèle (2)

- **Translation :** génèrer une matrice de translation puis multiplier la matrice active de modélisation.



- 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

$$x' = x + tx;$$

$$y' = y + ty;$$

$$z' = z + tz;$$

Transformations de modèle (3)

- La primitive OpenGl doit générer une matrice M, tel que :

$$v' = M.v$$

ou encore :

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

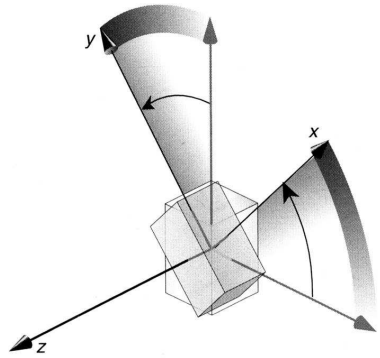
- Primitive :

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

Génère une matrice M qui déplace un objet/système de coordonnées selon le vecteur de translation (tx, ty, tz), puis multiplie la **matrice active** avec M.

Transformations de modèle (4)

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



- Primitive :

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

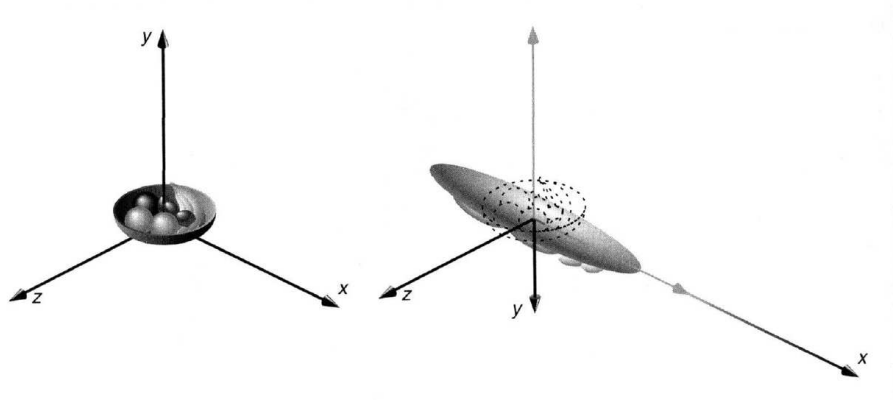
Applique sur un objet/système de coordonnées une rotation d'angle $angle$ sur les axes x, y et/ou z.

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

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

Transformations de modèle (5)

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



- Primitive :

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

Elargit, rétricit ou réfléchit un objet/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.

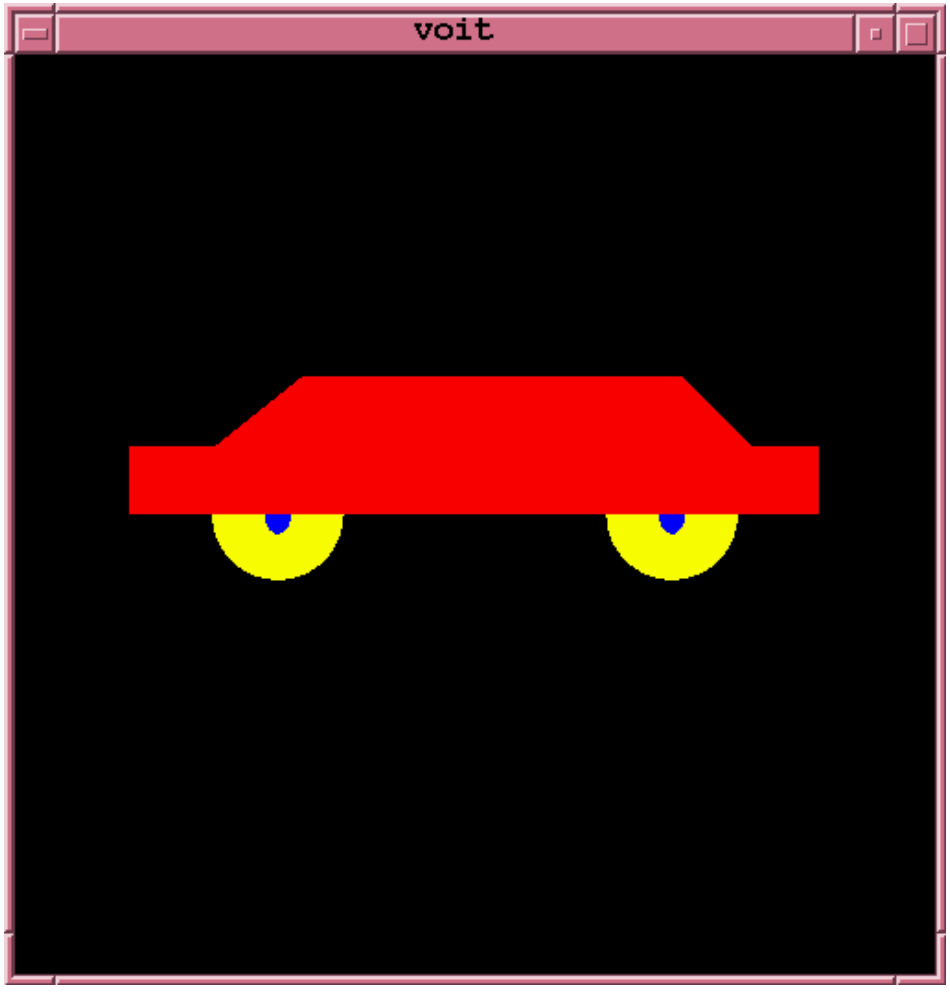
Transformations de modèle (6)

- La matrice active peut être sauvegardée, pour être restaurée plus tard : utilisation d'une pile de matrice.
- Pile de matrices de modélisation/visualisation et pile de matrices de projection.

- Primitives :
  - $glMatrixMode$: choisit la matrice active (de modélisation/visualisation avec $GL\_MODELVIEW$, ou de projection avec $GL\_PROJECTION$).
  - $glLoadIdentity$: initialise la matrice active.
  - $glPushMatrix$: empile la matrice active.
  - $glPopMatrix$: dépile 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 $GL\_MAX\_PROJECTION\_STACK\_DEPTH$).

Sommaire

1. Introduction
2. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. Ce qu'il faut retenir

Transformations de visualisation (1)

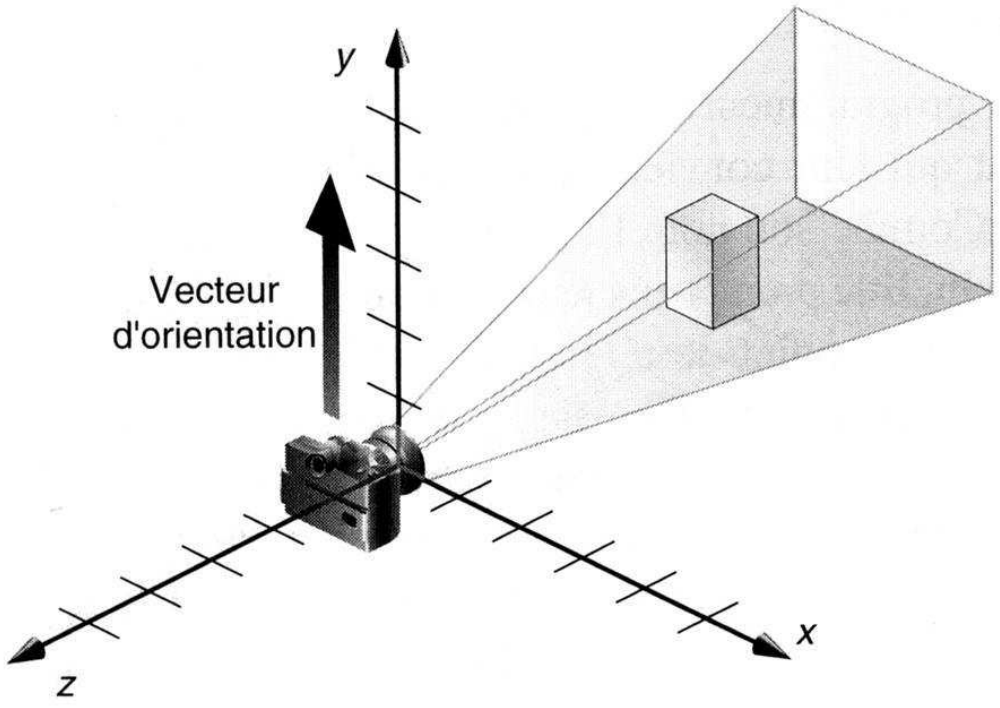
- Transformation qui consiste à définir la position et l'orientation de la caméra.

- 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)

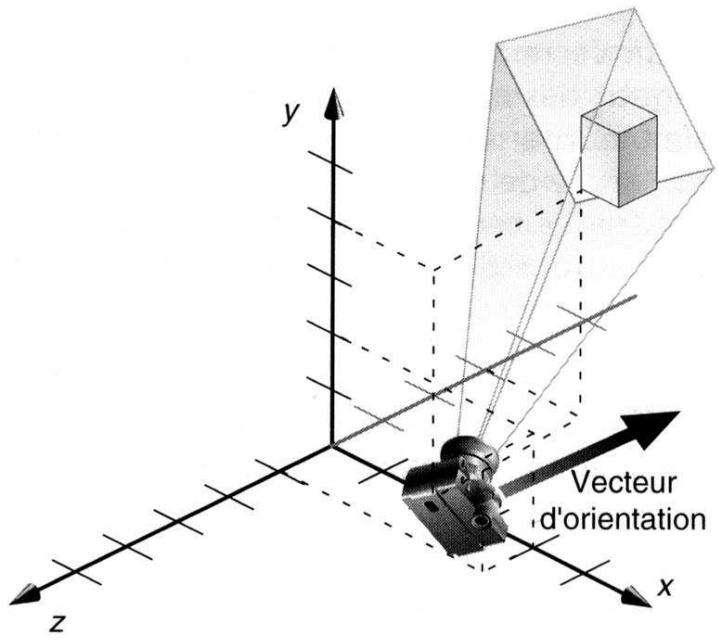


- Position par défaut :

```
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.

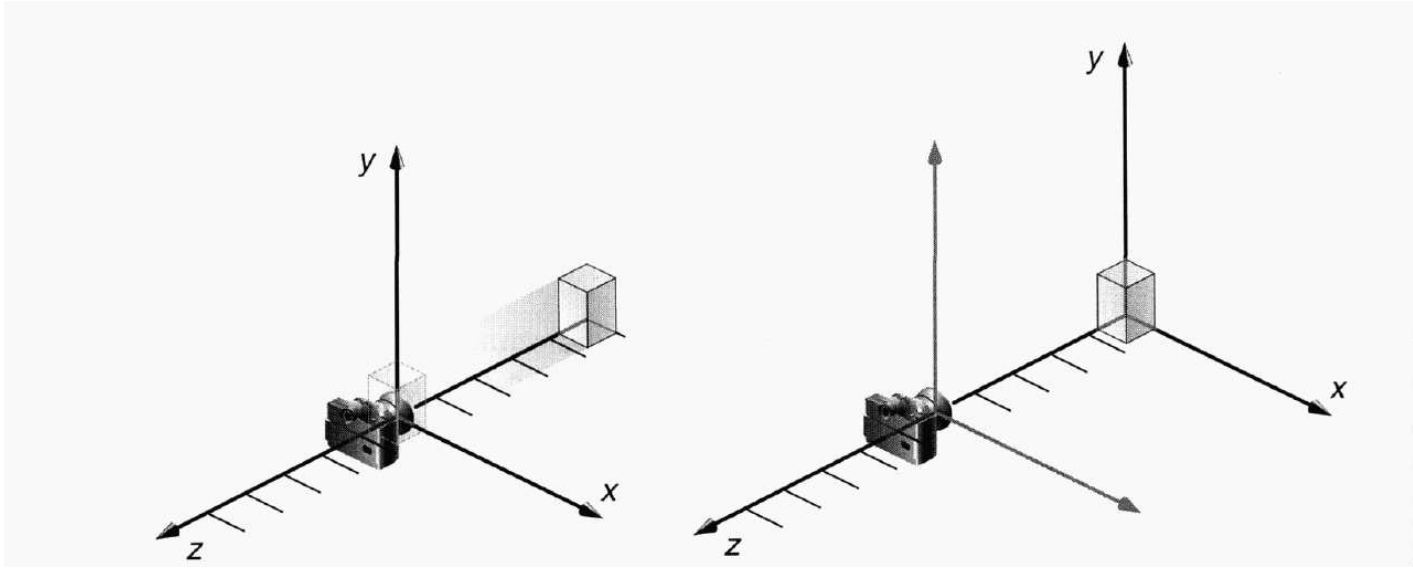
Transformations de visualisation (3)



- 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.

Sommaire

1. Introduction
2. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. Ce qu’il faut retenir

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.

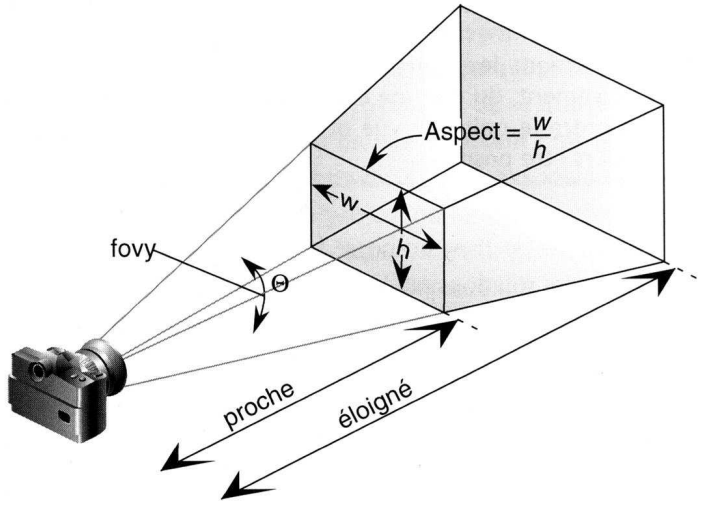
- Utilise une matrice différente : la matrice de projection.

- 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)

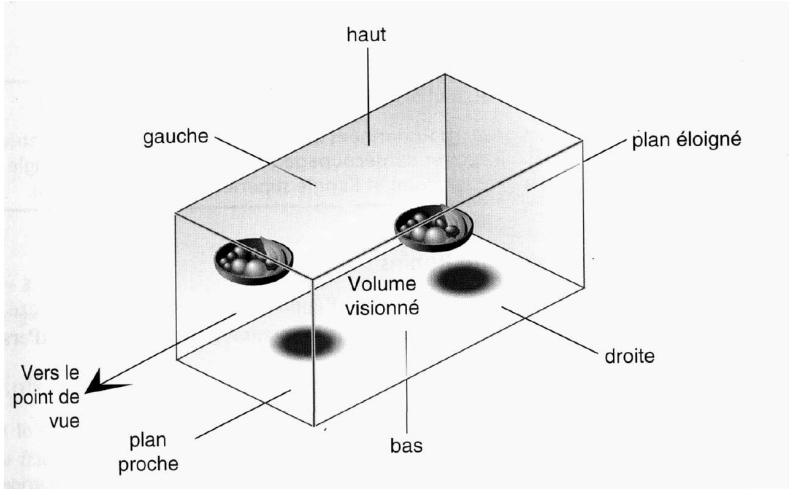


- **Perspective conique :** plus l'objet est éloigné et plus il est petit ; le volume est un frustum (tronc de pyramide).
- Deux primitives selon que le frustum soit symétrique avec un axe aligné sur l'axe des z ($gluPerspective$) ou non ($glFrustum$) :

```
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 : $gluOrtho2D$.

Sommaire

1. Introduction
2. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. Ce qu'il faut retenir

Transformation de cadrage (1)

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

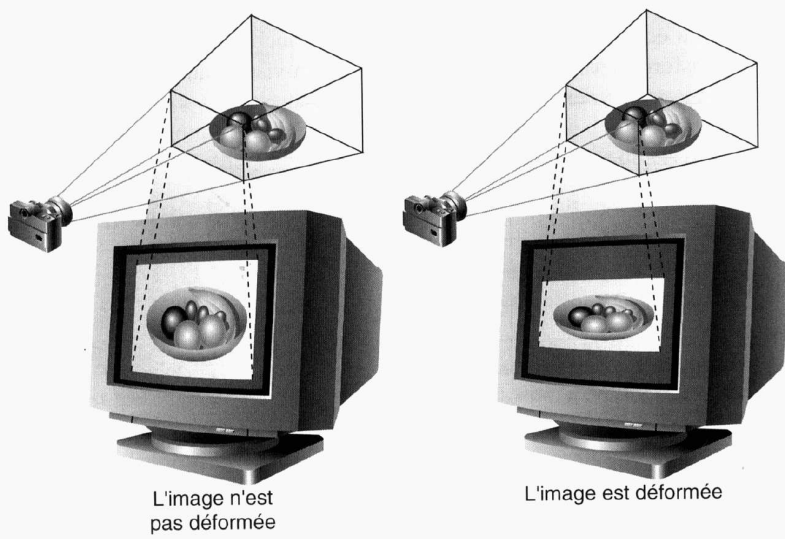
- Primitive :

```
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. Transformations de modèle
3. Transformations de visualisation
4. Transformations de projection
5. Transformations de cadrage
6. 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 (**cf. les primitives** $glRotate$**,** $glTranslate$**,** $gluLookAt$**,** $glFrustum$**,** $gluPerspective$ **excepté** $glViewPort$)
- Transformations importantes : modélisation et visualisation.
- La transformation de visualisation est remplaçable.
- Comment faire une description hiérarchique d'une scène avec les transformations de modèle et la pile de matrice de modélisation/visualisation.