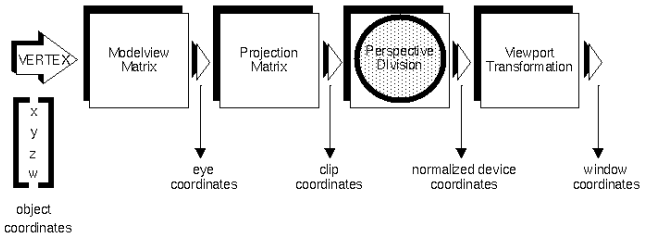
OpenGL : Visualisation et transformation

4 novembre 2009

1 Processus de visualisation

Pipeline de visualisation OpenGL

- Le rôle d'OpenGL est de projeter une scène 3D sur une image 2D : l'écran.
- Pour celà, chaque objet défini dans la scène subit plusieures transformations dans le **pipeline de visualisation**



Processus de visualisation

Quatre transformations utilisées au cours du processus de création d'une image :

1. Transformation de **modélisation** (Model) :
   Permet de créer la scène à afficher par création, placement et orientation des objets qui la compose.
2. Transformation de **visualisation** (View) :
   Permet de fixer la position et l'orientation de la caméra.
3. Transformation de **projection** (Projection) :
   Permet de fixer les caractéristiques optiques de la caméra (type de projection, angle d'ouverture, ...).
4. Transformation d'**affichage** (Viewport) :
   Permet de fixer la taille et la position de l'image sur la fenêtre d'affichage.

Matrice de transformation

- Chaque transformation est définie par une matrice.
- OpenGL compose ces matrices afin d'appliquer une transformation sur chaque objet de la scène.
- Le résultat de cette transformation est l'objet dans les coordonnées de l'image à afficher.
- OpenGL fourni un ensemble de fonctions permettant de manipuler les matrices de transformation.
- On choisi la matrice courante à manipuler avec **glMatrixMode**
- Ensuite on utilise une fonction permettant de définir ou de modifier la matrice courante : glLoadIdentity, glLoadMatrix, glMultMatrix,glTranslate, glRotate, glScale, gluLookAt, etc.

Exemple

```
static float rotx = 10.0f ;
static float roty = 20.0f ;
static float rotz = 30.0f ;

glViewport(0,0,w,h);                          // 4
glMatrixMode(GL_PROJECTION);                  // 3
glLoadIdentity();                             // 3
glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);        // 3
glMatrixMode(GL_MODELVIEW);                   // 2-1
glLoadIdentity();                             // 2
glTranslatef(0.0,0.0,-5.0);                   // 2
glRotatef(rotx,1.0,0.0,0.0);                  // 2
glRotatef(roty,0.0,1.0,0.0);                  // 2
glRotatef(rotz,0.0,0.0,1.0);                  // 2
glScalef(1.0,2.0,3.0);                        // 1
draw_cube(1.0);                               // 1
```

Choix de la transformation

Fonction glMatrixMode

Indique sur quel type de matrice on va appliquer les transformations

```
glMatrixMode(mode)
```

- **mode** : GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE

Pour réaliser un affichage, glMatrixMode est généralement appelé successivement une fois sur chacun des mode de manière à établir les matrices Modelview et Projection

2 Transformation géometrique

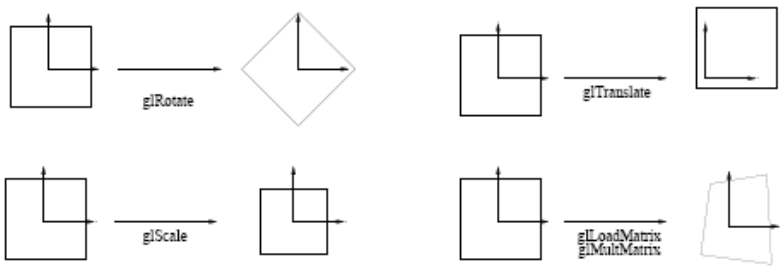
Transformation géometrique

Le positionnement dans la scène se fait à partir de transformations géométriques de bases :

Translation : glTranslate{fz d}(x, y, z) Multiplie la matrice courante par une matrice qui déplace l'objet (ou son repère local) en x, y, z.

Rotation : glRotate{f d}(α, dx, dy, dz) Multiplie la matrice courante par une matrice qui applique une rotation à l'objet (ou son repère local) d'angle α degrés autour de l'axe (dx,dy,dz) passant par l'origine.

Mise à l'échelle : glScale{f d}(sx, sy, sz) Permet de faire une homothétie.



Enchaînement des transformations

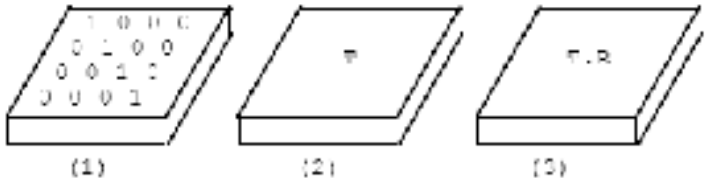
- OpenGL dispose d'une matrice de transformation courante pour la modélisation, matrice qui est rendue active par la fonction : **glMatrixMode(GL_MODELVIEW)**
- Cette matrice de modélisation est appliquée automatiquement à tous les objets qui vont être tracés.

Remarque

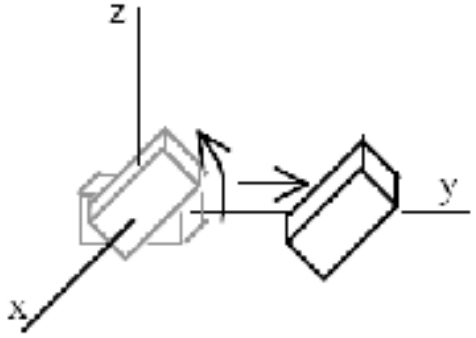
Les opérations effectuées sur un objet doivent être lues dans l'ordre inverse de leur apparition dans le code.

Exemple

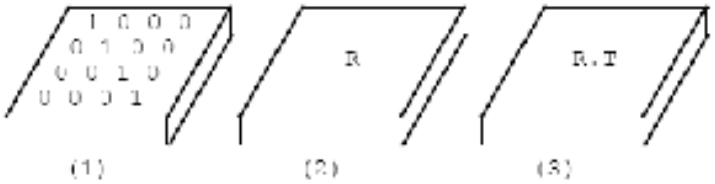
```
/* matrice de
   transformation A */
glLoadIdentity() ; /* 1 */
glTranslatef(0, 5, 0); /* 2 */
glRotatef(45, 1, 0, 0); /* 3 */
/*objet qui subira la
   transformation*/
dessineBoite();
```



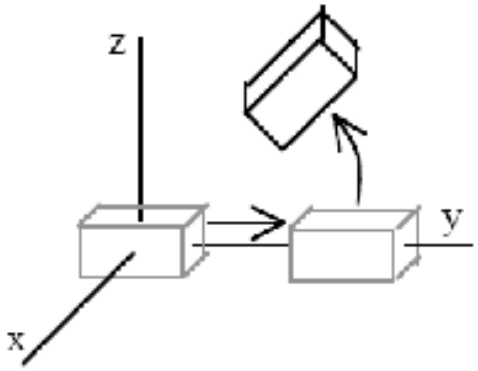
Evolution de la matrice de modélisation (A)




```
/* matrice de
   transformation B */
glLoadIdentity() ; /* 1 */
glRotatef(45, 1, 0, 0); /* 2 */
glTranslatef(0, 5, 0); /* 3 */
/* objet qui subira la
    transformation*/
dessineBoite();
```



Evolution de la matrice de modelisation (B)



3 Transformation spécifique à la visualisation

Transformation de visualisation

La transformation de visualisation permet de fixer la position et l'orientation de la caméra.

Fonction gluLookAt

```
void gluLookAt( GLdouble ex,GLdouble ey,GLdouble ez,
                GLdouble cx,GLdouble cy,GLdouble cz,
                GLdouble upx,GLdouble upy,GLdouble upz
                );
```

Multiplie la matrice courante (idéalement PROJECTION) par une transformation parametré par :

- Position de la caméra (ex,ey,ez)
- Orientation pour qu'elle vise le point (cx,cy,cz)
- La direction du repère courant (upx,upy,upz) (fréquemment le repère global). Typiquement (0.0,1.0,0.0), pour avoir upy dans la direction de l'écran.

4 Transformation de projection

Transformation de projection

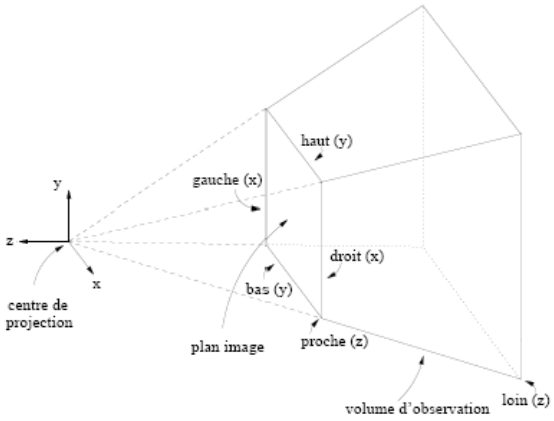
La transformation de projection permet de fixer les caractéristiques optiques de la caméra (type de projection, angle d'ouverture, ...).

Type de projection
OpenGL permet de definir deux types de projection :
- **Perspective**
- **Orthogonal**

Projection perspective

Pour définir une projection de type perspective, il faut définir :
- Un centre de projection,
- Une distance focale (distance du centre de projection au plan de projection)
- Une direction de projection.

Le centre de projection est à l'origine du repère courant et la distance focale est paramétrable.



Fonction glFrustrum

```
void glFrustum( GLdouble gauche, GLdouble droit,
                GLdouble bas, GLdouble haut,
                GLdouble proche, GLdouble loin);
```

Le volume de visualisation est une pyramide tronquée d'origine O, orientée selon l'axe -z. :

Fonction gluPerspective

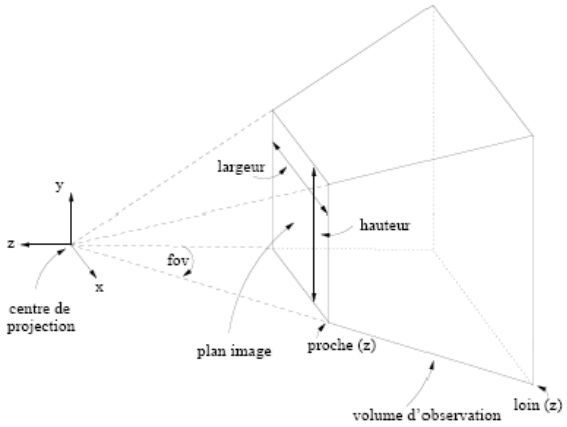
```
void gluPerspective(GLdouble fov, GLdouble rapport,
                    GLdouble proche, GLdouble loin);
```

Multiplie la matrice courante (idéalement PERSPECTIVE) par une transformation parametré par :

- **fov** est l'angle d'ouverture verticale
- **rapport** est le rapport d'échelle entre la largeur et la haueur du volume d'observation
- **proche** et **loin** sont les distances (positives) au plans du volume d'observation

Fonction gluPerspective

- **fov** est l'angle d'ouverture verticale
- **rapport** est le rapport d'échelle entre la largeur et la haueur du volume d'observation
- **proche** et **loin** sont les distances (positives) aux plans du volume d'observation



Projection orthogonal

Fonction glOrtho

```
void glOrtho(   GLdouble g, GLdouble d,
                GLdouble b, GLdouble h,
                GLdouble cmin, GLdouble cmax);
```

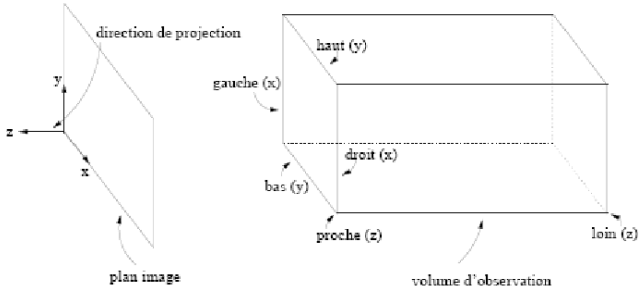
Multiplie la matrice courante par la transformation de projection orthographique selon l'axe -Z.

- Les paramètres **(g,d,b,h,-cmin,-cmax)** définissent le volume de visualisation parallélépipèdique.
- **cmin** et **cmax** peuvent être positifs ou négatifs

Projection orthogonal

Fonction glOrtho

– Les paramètres **(g,d,b,h,-cmin,-cmax)** définisent le volume de visualisation parallélépipèdique.
– **cmin** et **cmax** peuvent être positifs ou négatifs



Fonction glOrtho2D

```
void glOrtho2D(gauche, droit, bas, haut)
```

Projection d'un objet 2D sur un écran.

5 Transformation d'affichage (Viewport)

Transformation d'affichage (Viewport)

Fonction glViewport

```
glViewport(x, y, largeur, hauteur)
```

Cette fonction définit une fenêtre d'affichage à l'intérieur de la fenêtre graphique
– Cette fenêtre d'affichage est positionnée en **(x ; y)**
– A pour largeur *largeur* et hauteur *hauteur*.
– L'intérêt majeur de cette fonction est de pouvoir suivre les évolutions de la fenêtre graphique lorsque celle-ci est redimensionnée par l'utilisateur.

Remarque

Pour éviter des distorsions à l'affichage, il faut conserver le rapport d'échelle du volume d'observation. Ce dernier est défini par le rapport entre la largeur du volume du volume d'observation et sa hauteur ; le rapport entre la largeur de la fenêtre d'affichage et sa hauteur doit donc être équivalent.

6 Autres opérations

Autres opérations

glLoadIdentity() : Charge la matrice courante avec une matice identité
glLoadMatrixf d(m) : Charge la matrice m dans la matrice courante.
glMultMatrixf d(m) : Multiplie la matrice m par la matrice courante

Pile de matrice

glPushMatrix() : Empile la matrice courante dans la pile de matrices.

glPopMatrix() : Dépile la matrice en haut de pile et remplace la matrice courante par celle-ci.