

# Univers Virtuels

## OpenGL : Modélisation / Visualisation

*Alexis NEDELEC*

LISYC EA 3883 UBO-ENIB-ENSIETA  
Centre Européen de Réalité Virtuelle  
Ecole Nationale d'Ingénieurs de Brest

*enib ©2007*



# Processus de visualisation

## Univers réel

- 1 visualisation : positionner l'appareil photo
- 2 modélisation : arranger les éléments à photographier
- 3 projection : choisir la focale de l'appareil photo
- 4 cadrage : définir la taille au développement

## Univers virtuel

- 1 visualisation (**View**) : positionner la caméra virtuelle
- 2 modélisation (**Model**) : création d'objets dans une scène
- 3 projection (**Projection**) : réglage de visualisation
- 4 cadrage (**Viewport**) : taille de l'image résultante

# Processus de visualisation en OpenGL

## Etapes de transformation

- 1 placer la caméra virtuelle : `gluLookAt()`
- 2 composer une scène virtuelle :
  - transformer les objets :  
`glTranslatef()`, `glRotatef()`, `glScalef()`...
  - à créer :  
`glBegin().... glEnd()`, `glutWireCube(1)` ...
- 3 choisir une projection :  
`glOrtho()`, `glPerspective()`, `glFrustum()`
- 4 choisir les caractéristiques de l'image : `glViewport()`

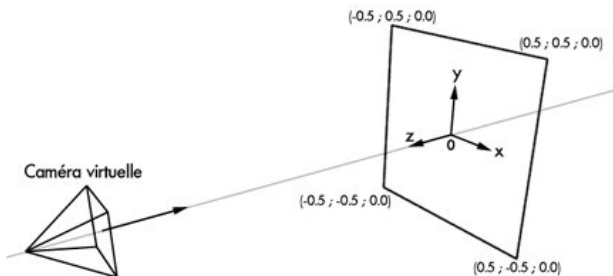
## Matrices de Transformation

- 1 modélisation/visualisation : `glMatrixMode(GL_MODELVIEW)`
- 2 projection : `glMatrixMode(GL_PROJECTION)`

# Caméra virtuelle OpenGL

## Point d'observation par défaut

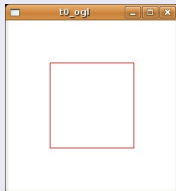
- la caméra est située sur l'axe  $Oz$
- direction de visée : origine du repère
- verticale de la caméra : axe  $Oy$  du repère



# Création de scène OpenGL

## Exemple de transformation

```
void display()  
{  
    glClear(GL_COLOR_BUFFER_BIT);  
    glLoadIdentity();  
    glColor3f(1.0,0.0,0.0);  
    glutWireCube(1);  
    glFlush();  
}
```



# Coordonnées et matrice homogène

## 16 coefficients de matrice

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

## Composition de transformations

- $[P]$  : point de l'espace,  $[P]^T$  transposée de  $[P]$
  - $[M]$  : matrice homogène,  $[M]^T$  : transposée de  $[M]$
- $[P']$  : résultat des transformations successives  $i$  ( $1 \leq i \leq n$ ),
- $[P'] = [M_n] \dots [M_i] \dots [M_1][P]$
  - $[P'] = [P]^T [M_1]^T \dots [M_i]^T \dots [M_n]^T$

# Matrices de transformations OpenGL

## Activation de matrices : `glMatrixMode()`

- `GL_MODELVIEW` : matrice de transformation-visualisation
- `GL_PROJECTION` : matrice de projection
- `GL_TEXTURE` : matrice de texture

`GL_MODELVIEW` : matrice active par défaut

## Manipulation de matrices

- `glLoadIdentity()` : initialisation de matrice
- `glLoadMatrix()` : chargement de matrice (16 coefficients)

# Transformation-Visualisation OpenGL

## Enchaînement de transformation

- 1 `glTranslatef(float dx, float dy, float dz) :`  
 $[M_a] = [M_a][M_t]$ , translation  $(dx, dy, dz)$
- 2 `glRotatef(float theta, float x, float y, float z) :`  
 $[M_a] = [M_a][M_\theta]$ , rotation de  $\theta$  autour de l'axe  $(0, x, y, z)$
- 3 `glScalef(float hx, float hy, float hz) :`  
 $[M_a] = [M_a][M_h]$ , homothétie, mise à l'échelle  $(hx, hy, hz)$

## Exemple d'enchaînement

Translation (axe Oy) suivi d'une rotation ( $45^\circ$ , axe Oz)

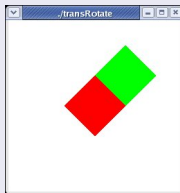
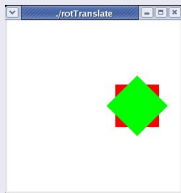
```
glLoadIdentity();  
glRotatef(45.0, 0.0, 0.0, 1.0);  
glTranslatef(0.0, 1.0, 0.0);
```

# Transformation-Visualisation OpenGL

## Exemple de transformation

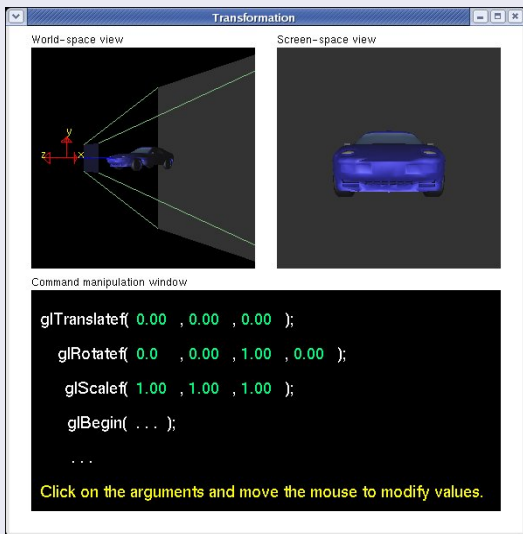
```
glLoadIdentity();  
glTranslatef(0.5,0.0,0.0);  
glColor3f(1.0,0.0,0.0);  
glRectf(-0.25, -0.25, 0.25, 0.25);  
glRotatef(45.0,0.0,0.0,1.0);  
glColor3f(0.0,1.0,0.0);  
glRectf(-0.25, -0.25, 0.25, 0.25);
```

## Visualisation de transformation ?



# Visualisation OpenGL

## Didacticiel Nate Robbins



The screenshot shows a window titled "Transformation" with three main sections:

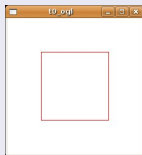
- World-space view:** A 3D scene showing a blue car on a black plane. A camera frustum is visible, defined by a grey rectangle and green lines. A 3D coordinate system with red, green, and blue axes is shown on the left.
- Screen-space view:** A 2D projection of the blue car from the world-space view.
- Command manipulation window:** A black area containing OpenGL commands with numerical arguments highlighted in green. The commands are:

```
glTranslatef( 0.00 , 0.00 , 0.00 );  
glRotatef( 0.0 , 0.00 , 1.00 , 0.00 );  
glScalef( 1.00 , 1.00 , 1.00 );  
glBegin( . . . );  
...  
Click on the arguments and move the mouse to modify values.
```

# Visualisation OpenGL

## Visualisation par défaut

- caméra positionnée sur l'axe  $Oz$  (centre de la fenêtre)
- dirigée vers l'origine du repère (vers l'intérieur,  $z < 0$ )
- projection orthogonale (pas de point de fuite)



## Matrice de projection et cadrage

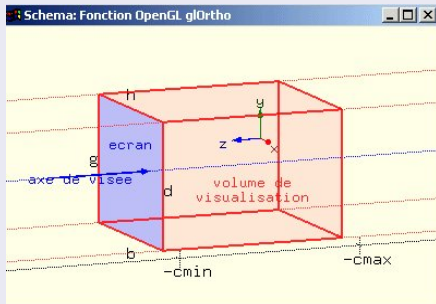
- perspective cavalière : `glOrtho()`, `glOrtho2D()` ...
- perspective conique : `glPerspective()`, `glFrustum()`
- dimensionnement de la photo : `glViewport()`

# Types de projection

## Perspective cavalière

```
glOrtho(GLdouble left, GLdouble right,  
        GLdouble bottom, GLdouble up,  
        GLdouble near, GLdouble far)
```

## Volume de visualisation

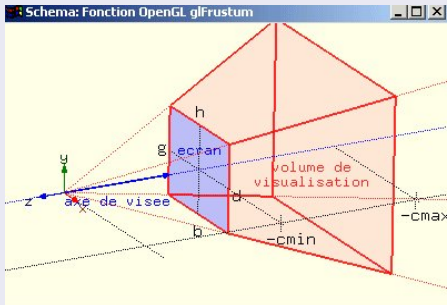


# Types de projection

## Perspective conique

```
glFrustum(GLdouble left, GLdouble right,  
          GLdouble bottom, GLdouble up,  
          GLdouble near, GLdouble far)
```

## Volume de visualisation

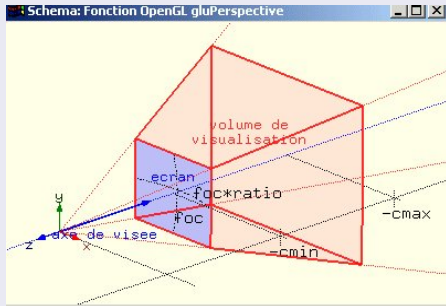


# Types de projection

## Perspective conique symétrique

```
gluPerspective(GLdouble fovy,  
              GLdouble ratio_wh,  
              GLdouble near, GLdouble far)
```

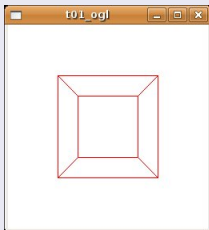
## Volume de visualisation



# Matrice de projection

Recadrage : `glutReshapeFunc(reshape)`

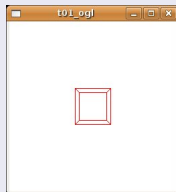
```
void reshape(int width, int height) {  
    glViewport(0,0, (GLsizei) width, (GLsizei) height);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);  
    glMatrixMode(GL_MODELVIEW);  
}
```



# Matrice de Visualisation

## Affichage : glutDisplayFunc(display)

```
void display() {  
    glClearColor(GL_COLOR_BUFFER_BIT);  
    glLoadIdentity();  
    gluLookAt (0.0,0.0,4.0,0.0,0.0,0.0,0.0,1.0,0.0);  
    glColor3f(1.0,0.0,0.0);  
    glutWireCube (1);  
    glutSwapBuffers ();  
}
```



# Modélisation versus Visualisation

## Approche modélisation

- création d'un objet
- transformations (ex : translations  $(1, 0, 0)$  suivi de  $(0, 0, -3)$ )

## Approche visualisation

- création d'un objet
- positionnement de la caméra (ex : en  $(-1, 0, 3)$ )

## Positionnement de caméra

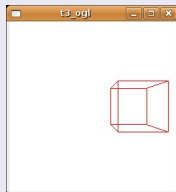
`gluLookAt(xpos, ypos, zpos, xdir, ydir, zdir, hx, hy, hz)`

- position dans le repère : `xpos, ypos, zpos`
- direction (point visé) : `xdir, ydir, zdir`
- axe vertical : `hx, hy, hz`

# Modélisation versus Visualisation

## Approche modélisation

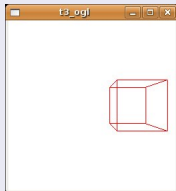
```
void display() {  
    ...  
    glTranslated(0,0,-3);  
    glTranslated(1,0,0);  
    glColor3f(1.0,0.0,0.0);  
    glutWireCube(1);  
    ...  
}
```



# Modélisation versus Visualisation

## Approche visualisation

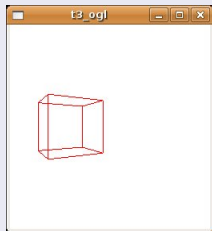
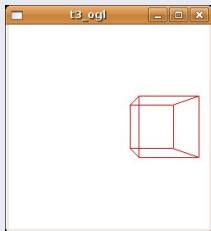
```
void display() {  
    ...  
    gluLookAt(-1.0,0.0,3.0,-1.0,0.0,0.0,0.0,1.0,0.0);  
    glColor3f(1.0,0.0,0.0);  
    glutWireCube(1);  
    ...  
}
```



# Modélisation versus Visualisation

Quelle visualisation ?

```
void display() {  
    ...  
    gluLookAt(1.0,0.0,-3.0,1.0,0.0,0.0,0.0,1.0,0.0);  
    ...  
}
```



# Création de scène

## Plusieurs objets à dessiner

Placer un objet en  $(1, 0, 1)$  et l'autre en  $(3, 0, 0)$

```
glLoadIdentity();  
glTranslatef(1.0,0.0,1.0);  
dessineObjet();  
glTranslatef(3.0,0.0,0.0);  
dessineObjet();
```

## Problème de la matrice active

- accumulation des transformations
- deuxième objet placé en  $(4, 0, 1)$

Solution : `glLoadIdentity()` après le dessin du premier objet

# Déplacement d'observateur

## Position d'observation : gluLookAt()

```
glLoadIdentity();  
gluLookAt(10.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);  
glTranslatef(1.0,0.0,1.0);  
dessineObjet();  
glLoadIdentity();  
glTranslatef(3.0,0.0,0.0);  
dessineObjet();
```

## Problème de la matrice active

- réinitialisation après le premier dessin
- appel `gluLookAt()` après chaque dessin ....

Solution : piles de matrices OpenGL

# Pile de matrices : visualisation de scènes

## Matrice active : Empiler-Dépiler

- `glPushMatrix()` : copie de la matrice active dans la pile .
- `glPopMatrix()` : enlever la matrice du sommet de la pile et la copier dans la matrice active

## Matrice active : Affichage de scène

```
glLoadIdentity();  
gluLookAt(10.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);  
glPushMatrix();  
    glTranslatef(1.0,0.0,1.0);  
    dessineObjet();  
glPopMatrix();  
    glTranslatef(3.0,0.0,0.0);  
    dessineObjet();
```

# Pile de matrices : visualisation de scènes

## Dessine-moi une roue

```
void dessine_roue_et_boulons(void) {
    dessine_roue();
    for (int i=0;i<5;i++) {
        glPushMatrix();
        glRotatef(degre*i, 0.0, 0.0, 1.0);
        glTranslatef(xboulon, 0.0, 0.0);
        dessine_boulon();
        glPopMatrix();
    }
}
```

# Pile de matrices : visualisation de scènes

## Dessine-moi une voiture

```
void dessine_voiture(void) {  
    dessine_carrosserie();  
    glPushMatrix();  
        glTranslatef(x1, 0.0, z1);  
        dessine_roue_et_boulons();  
    glPopMatrix();  
    glPushMatrix();  
        glTranslatef(x1, 0.0, -z1);  
        dessine_roue_et_boulons();  
    ...  
}
```

# Graphe de scènes : définition

## Définition

- Graphe Acyclique Orienté (DAG)
- noeuds et liens parent-enfant

## Principe de base

- les feuilles sont les objets à dessiner
- noeud intermédiaire : groupe ou transformation
- chaque noeud n'a qu'un seul parent
- transformation courante : composition des transformations sur le chemin menant de la racine au noeud courant
- coordonnées d'un objet relatives à la transformation courante

# Graphe de scènes : parcours d'arbre

## Parcours d'un arbre

En chaque noeud

- on mémorise la matrice de transformation courante
- on transforme localement
- on dessine chacun des noeuds fils
- on restaure la matrice de transformation

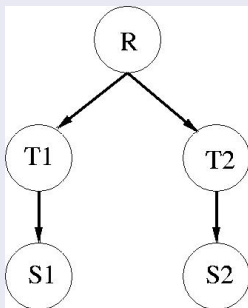
## Intérêts

- héritage de propriété (rotation roue-boulon)
- facilité de manipulation (déplacement de l'ensemble)
- partage d'objets (une seule roue, 4 transformations)

# Grappe de scènes : construction

## Construction

```
glRotate(90,10,0);  
glPushMatrix();  
glTranslate(-2,0,0);  
afficherSphere1();  
glPopMatrix();  
glPushMatrix();  
glTranslate(2,0,0);  
afficherSphere2();  
glPopMatrix();
```



## Exemples courants

- voiture, système solaire, humanoïdes ...

# Animation de scènes : principe

## Projection cinématographique

```
void projecteur(void) {  
    int pellicule = 1000000;  
    for (i=0;i<pellicule;i++) {  
        vider_la_fenetre();  
        dessiner(i);  
        attendre_un_24eme_de_seconde();  
    }  
}
```

## Problème

- temps nécessaire au dessin, effacement
- dessin en retard d'objets, “aspects fantomatiques”

# Animation de scènes : “Double-buffering”

## Solution : Projecteur double-tampon

Il n’y a plus de pellicule mais deux cadres

- un tampon pour afficher
- un tampon pour dessiner

## Animation double-tampon

```
initialiser_mode_double_tampons();  
for (i=0;i<1000000;i++) {  
    vider_la_fenetre();  
    dessiner(i);  
    echange_les_tampons();  
}
```

# Animation de scènes : OpenGL

## Affichage en OpenGL

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0,1.0,1.0);
    glRectf(-0.5,-0.5,0.5,0.5);
    glutSwapBuffers();
}
```

# Animation de scènes : OpenGL

## Définition de l'animation

```
void displayEvent(void)
{
    spin = spin + 0.5;
    if (spin > 360.0)
        spin = spin -360.0;
    glutPostRedisplay();
}
```

## Gestion de l'animation

```
int main(int argc, char **argv) {
    ...
    glutIdleFunc(displayEvent);
    ...
}
```

# Liaison Événement-Action

## Types d'événements

- affichage, clavier, souris
- divers périphériques (tablettes graphiques, spaceballs ...)
- modification de la configuration de la fenêtre
- indépendant des interactions (idle)

## Boucle d'événements

```
int main(int argc, char **argv) {  
    initGLUT(argc, argv);  
    initGL();  
    glutDisplayFunc(display);  
    glutMainLoop();  
    return(0);  
}
```

# Fonctions de rappels (callbacks)

## Implémentation du comportement

```
void keyboard(unsigned char key,int x,int y) {
    switch (key) {
        case 'p': /* affichage du carre plein */
            glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
            glutPostRedisplay();
            break;
        case 'f': /* affichage en mode fil de fer */
            glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
            ...
    }
}

int main(int argc,char **argv) {
    ...
    glutKeyboardFunc(keyboard);
    ...
}
```

# Bibliographie

## Livres

- **M. Woo, J. Neider, T. Davis, D. Schreiner :**  
“OPengl 2.0 Guide Officiel”  
Collection Campus Press (2006)
- **Samuel R. Buss :**  
“3D Computer Graphics :  
A mathematical introduction with OpenGL”  
Collection Cambridge University Press (2003)
- **B. Péroche, D. Bechmann :**  
“Informatique graphique et rendu”  
Collection Hermès, Lavoisier (2007)
- **R. Malgouires :**  
“Algorithmes pour la synthèse d’images et l’animation 3D”  
Éditions Dunod (2002)

# Bibliographie

## Adresses “au Net”

- [www.opengl.org](http://www.opengl.org) : le site officiel
- [www.xmission.com/~nate/opengl.html](http://www.xmission.com/~nate/opengl.html) :  
demos OpenGL de Nate Robbins
- [www.linuxgraphic.org/section3d/openGL/didact.html](http://www.linuxgraphic.org/section3d/openGL/didact.html) :  
tutoriaux de Xavier Michelon
- [helios.univ-reims.fr/Labos/LERI/membre/bittar/](http://helios.univ-reims.fr/Labos/LERI/membre/bittar/) :  
formation OpenGL 2007/2008 par Eric Bittar
- <http://nehe.gamedev.net> : tutoriaux OpenGL
- <http://local.wasp.uwa.edu.au/~pbourke> :  
la 3D en long et en large (et en hauteur)
- [www.developpez.com](http://www.developpez.com) : entre autre de la 2D/3D