Univers Virtuels

OpenGL : Modélisation / Visualisation

Alexis NEDELEC

Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Processus de visualisation

Univers réel

1. Positionner l'appareil photo
2. Arranger les éléments d'une scène à photographier
3. Choisir la focale de l'appareil photo
4. Choisir la taille de la photographie au développement

Univers virtuel

1. modélisation (`Model`) : création de scène
2. visualisation (`View`) : position de caméra
3. projection (`Projection`) : réglage de visualisation
4. affichage (`Viewport`) : taille de l'image résultante

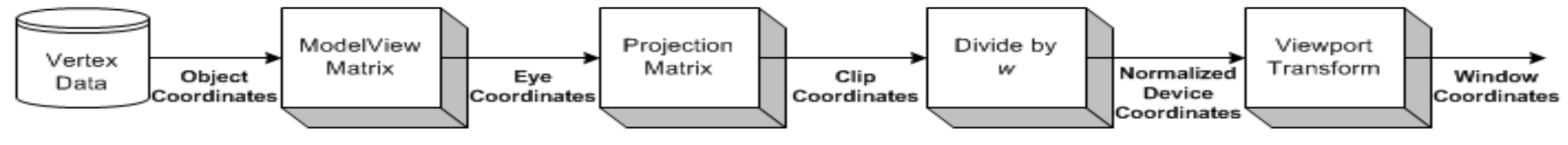
Processus de visualisation en OpenGL

Univers virtuel : étapes 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(), glFrustrum()`
4. choisir les caractéristiques de l'image : `glViewport()`

Processus de visualisation en OpenGL

Pipeline graphique



Matrices de transformations

1. modélisation-visualisation : `glMatrixMode(GL_MODELVIEW)`
2. projection : `glMatrixMode(GL_PROJECTION)`
3. fenêtrage : `glViewport(x,y,w,h)`
4. volume à visualiser : `glDepthRange(near,far)`

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

Modélisation-Visualisation : `glMatrixMode(GL_MODELVIEW)`

$$\begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} = [ModelView] \begin{bmatrix} x_{obs} \\ y_{obs} \\ z_{obs} \\ w_{obs} \end{bmatrix} = [View].[Model] \begin{bmatrix} x_{obs} \\ y_{obs} \\ z_{obs} \\ w_{obs} \end{bmatrix}$$

Projection : `glMatrixMode(GL_PROJECTION)`

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = [Projection] \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix}$$

Matrices de transformations

Projection : `glMatrixMode(GL_PROJECTION)`

$$\begin{bmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{bmatrix} = \begin{bmatrix} x_{clip}/w_{clip} \\ x_{clip}/w_{clip} \\ x_{clip}/w_{clip} \end{bmatrix}$$

NDC : Normalized Device Coordinates

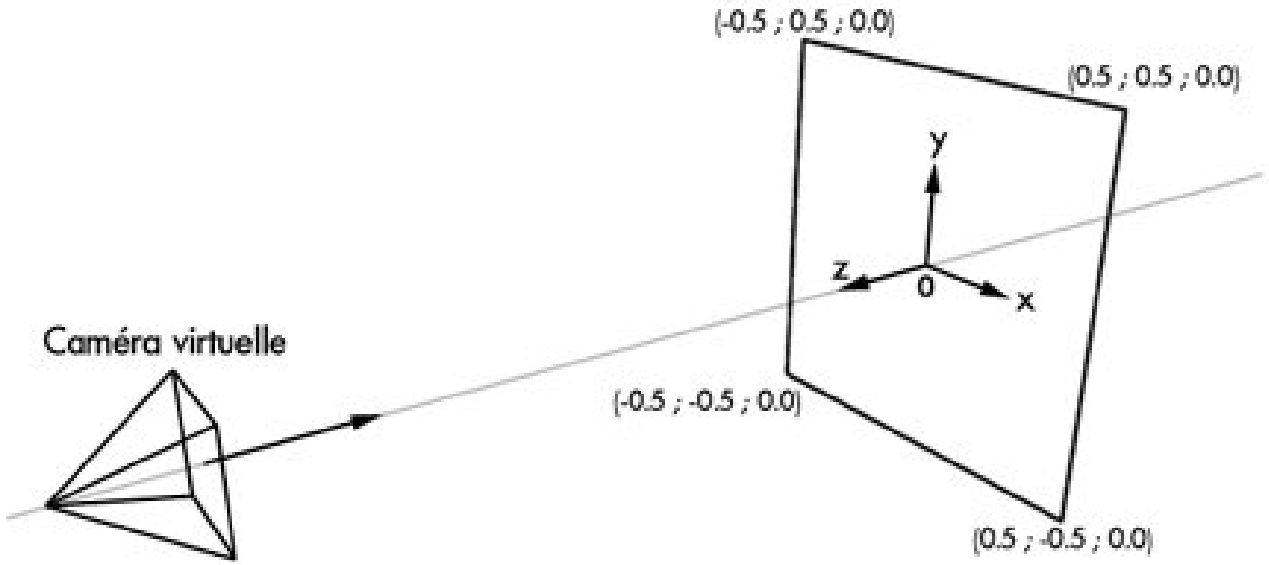
Fenêtrage (x,y,w,h) et profondeur (f,n)

$$\begin{bmatrix} x_{win} \\ y_{win} \\ z_{win} \end{bmatrix} = \begin{bmatrix} \frac{w}{2}.x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2}.y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2}.z_{ndc} + \frac{f+n}{2} \end{bmatrix}$$

Caméra Virtuelle OpenGL

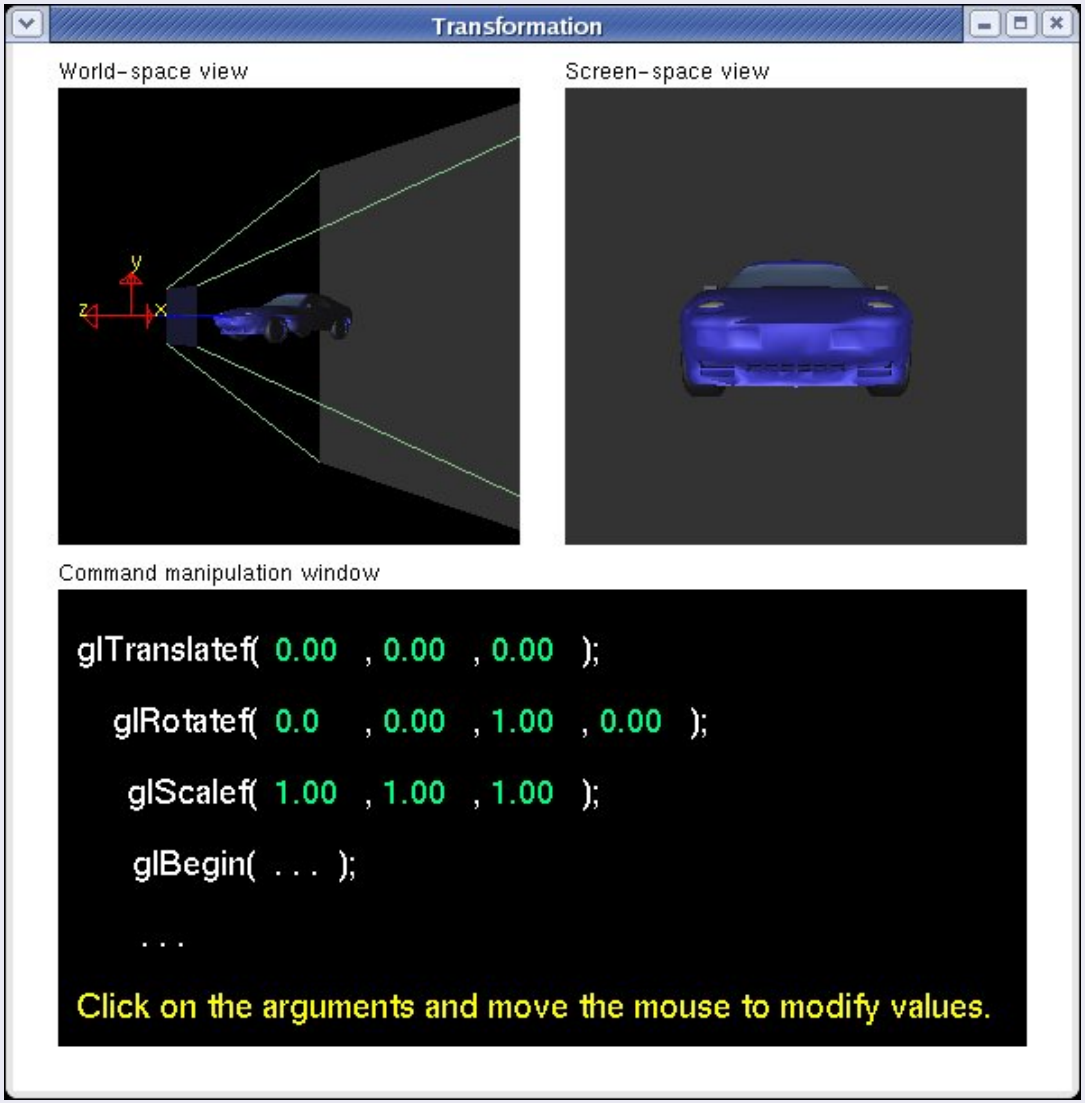
Point d'observation par défaut

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



Transformations OpenGL

Didacticiel Nate Robbins



Matrices de transformations

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)

Matrices de transformations

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 θ 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^o, axe Oz)

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

Coordonnées et matrice homogène

glLoadIdentity()

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

glTranslatef(dx,dy,dz)

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordonnées et matrice homogène

glRotatef(θ,1,0,0)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

glRotatef(θ,0,1,0)

$$\begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & & 0 \\ -sin\theta & & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordonnées et matrice homogène

glRotatef(θ,0,0,1)

$$\begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation θ autour d'un vecteur unitaire $\overrightarrow{u} = (u_x, u_y, u_z)$

$$R = \begin{bmatrix} u_x^2 + (1-u_x^2)c & u_x u_y(1-c) - u_z s & u_x u_z(1-c) + u_y s \\ u_x u_y(1-c) + u_z s & u_y^2 + (1-u_y^2)c & u_y u_z(1-c) - u_x s \\ u_x u_z(1-c) - u_y s & u_y u_z(1-c) + u_x s & u_z^2 + (1-u_z^2)c \end{bmatrix}$$

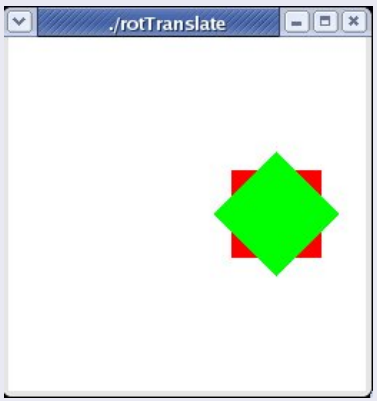
$c = cos\theta, s = sin\theta$

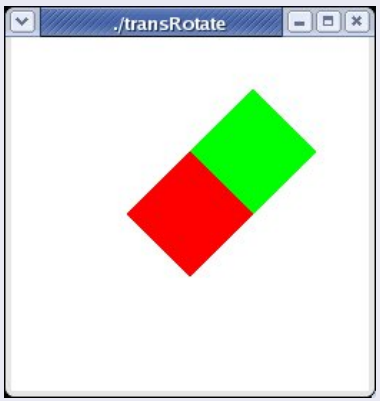
Matrices de transformations

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 de transformation

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)

Exemple : Observation de scène

```
glLoadIdentity();
glTranslated(0,0,-5);
glTranslated(1,0,0);
glBegin(GL_QUADS);
...
glEnd();
```

Positionnement de caméra

Interprétation de modélisation

- création d'un quadrilatère
- transformations : translations $(1, 0, 0)$ suivi de $(0, 0, -5)$

Interprétation de visualisation

- création d'un quadrilatère
- positionnement de la caméra en $(-1, 0, 5)$

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

Caméra : position,direction de visée et axe vertical

```
glLoadIdentity();
gluLookAt(-1,0,5,0,0,0,0,1,0);
...
```

Projection et cadrage

Matrice de projection et zone d'affichage

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

Recadrage : `glutReshapeFunc(reshape)`

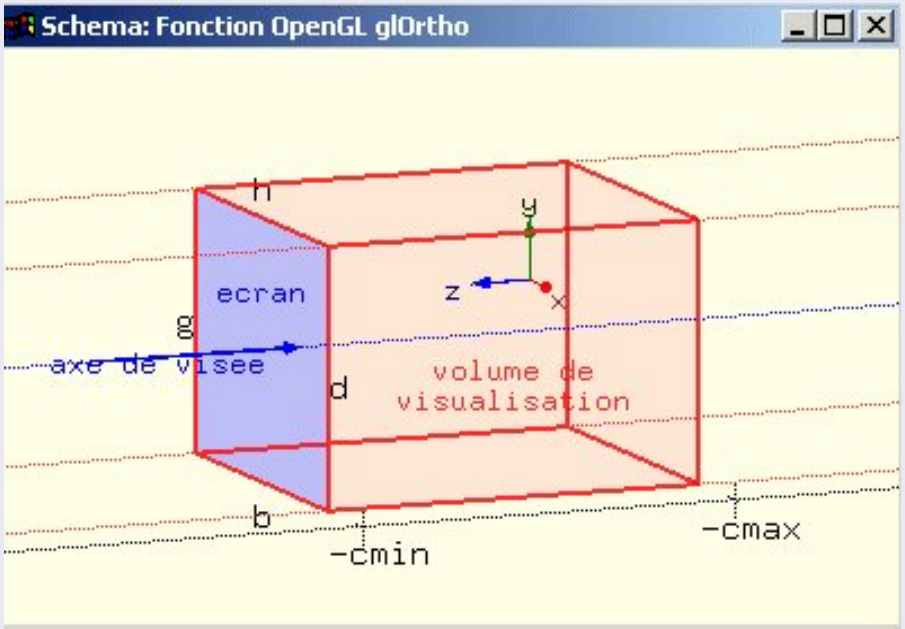
```
void reshape(int width, int height) {
  glViewport(0,0, (GLsizei) width, (GLsizei) height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
```

Types de projection

glOrtho()

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

Matrice de projection et zone d'affichage

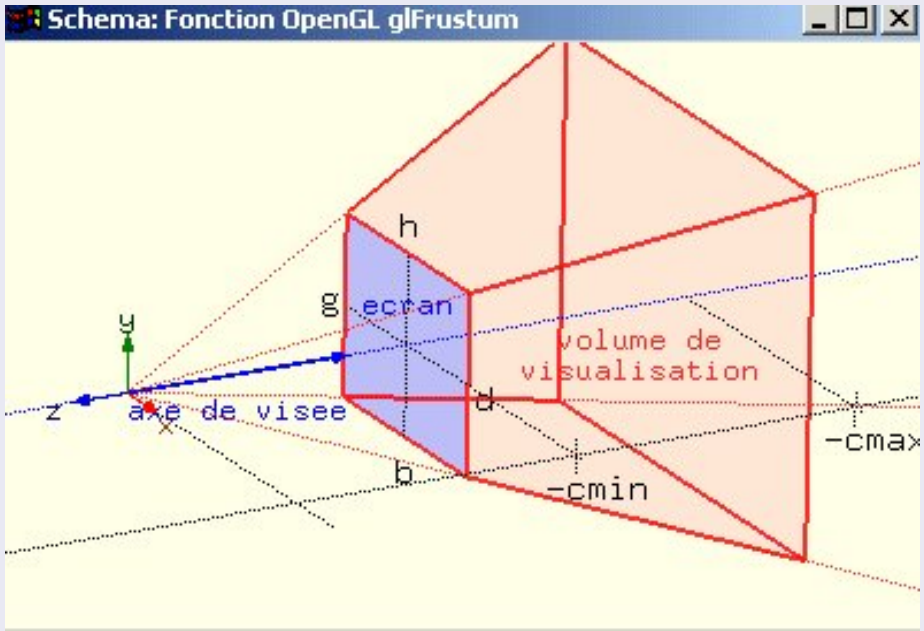


Types de projection

glFrustum()

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

Matrice de projection et zone d'affichage

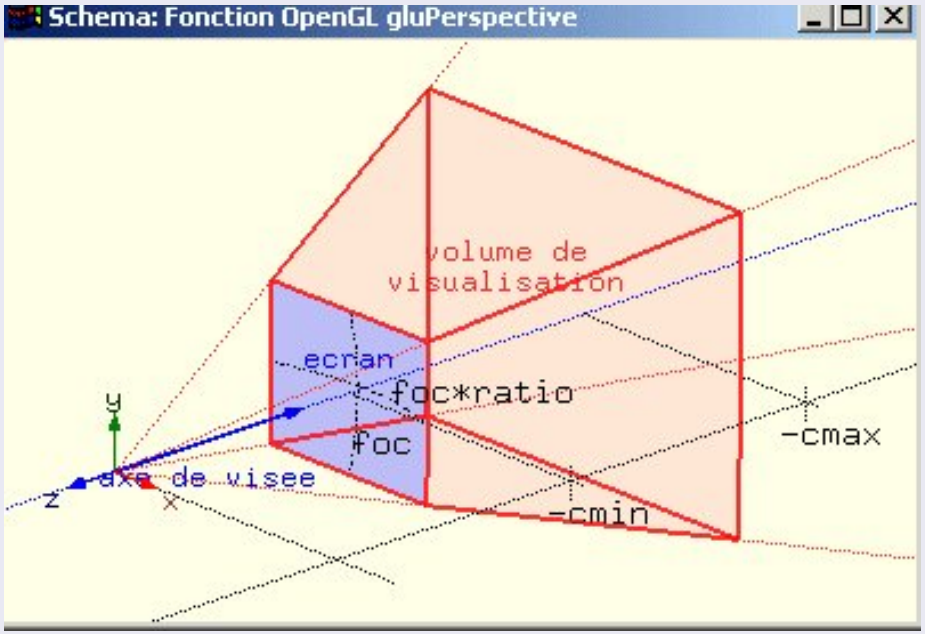


Types de projection

glPerspective()

```
gluPerspective(GLdouble foc,
               GLdouble ratio,
               GLdouble near, GLdouble far)
```

Matrice de projection et zone d'affichage



Création de scène

Plusieurs objets à dessiner

Placer un objet en $(1, 0, 1)$ et l'autre en $(5, 0, 0)$

```
glLoadIdentity();
glTranslatef(1.0,0.0,1.0);
dessineObjet();
glTranslatef(5.0,0.0,0.0);
dessineObjet();
```

Problème de la matrice active

- accumulation des transformations
- deuxième objet placé en $(6, 0, 1)$

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(5.0,0.0,0.0);
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(1.0,0.0,2.0);
  dessineObjet();
glPopMatrix();
  glTranslatef(5.0,0.0,0.0);
  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

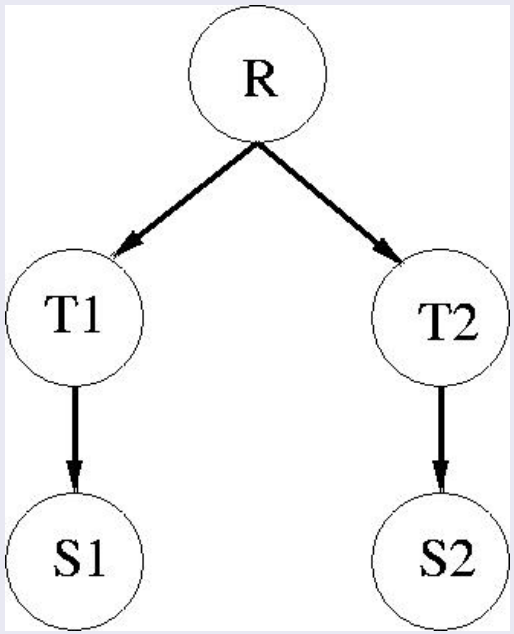
Intérêts

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

Graphe 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 ...

PyGame : sceneGraph.py

Initialisation

```
import math
import sys
import pygame,sys,time
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *

class ExitException(Exception):
    def __init__(self,*kwargs):
        Exception.__init__(self,*kwargs)
```

PyGame : sceneGraph.py

Primitives

```
def Point():
    glBegin(GL_POINTS)
    glVertex(0,0,0)
    glEnd()

def Triangle():
    glBegin(GL_TRIANGLES)
    glVertex(0,0.8,0)
    glVertex(0.2,-0.2,0)
    glVertex(-0.2,-0.2,0)
    glEnd()
```

PyGame : sceneGraph.py

Primitives

```
def Quad():
    glBegin(GL_QUADS)
    glVertex( 0.5, 0.5)
    glVertex( 0.5,-0.5)
    glVertex(-0.5,-0.5)
    glVertex(-0.5, 0.5)
    glEnd()
```

PyGame : sceneGraph.py

La Scène

```
class SceneNode:
    def __init__(self,children = None):
        if children:
            self.children = children
        else:
            self.children = []
    def draw(self):
        for x in self.children:
            x.draw()
```

PyGame : sceneGraph.py

Noeud Primitive

```
class ThingNode(SceneNode):
    def __init__(self,func):
        SceneNode.__init__(self)
        self.func = func
    def draw(self):
        glColor(0,0,0)
        glPushMatrix()
        self.func()
        glPopMatrix()
```

PyGame : sceneGraph.py

Noeud d' "Echelle"

```
class ScaleNode(SceneNode):
    def __init__(self,factor,children = None):
        SceneNode.__init__(self,children)
        self.factor = factor
    def draw(self):
        glPushMatrix()
        if callable(self.factor):
            glScale(*self.factor())
        else:
            glScale(*self.factor)
        SceneNode.draw(self)
        glPopMatrix()
```

PyGame : sceneGraph.py

Noeud de Translation

```
class TranslationNode(SceneNode):
    def __init__(self,offset,children = None):
        SceneNode.__init__(self,children)
        self.offset = offset
    def draw(self):
        glPushMatrix()
        if callable(self.offset):
            glTranslate(*self.offset())
        else:
            glTranslate(*self.offset)
        SceneNode.draw(self)
        glPopMatrix()
```

PyGame : sceneGraph.py

Noeud de Rotation

```
class RotationNode(SceneNode):
    def __init__(self,angle,axe,children = None):
        SceneNode.__init__(self,children)
        self.angle,self.xRot,...=angle,axe[0],...
    def draw(self):
        glPushMatrix()
        if callable(self.angle):
            glRotate(self.angle(),
                     self.xRot,self.yRot,self.zRot)
        else:
            glRotate(self.angle,
                     self.xRot,self.yRot,self.zRot)
        SceneNode.draw(self)
        glPopMatrix()
```

PyGame : sceneGraph.py

Programme de test

```
def main():
    dx,dy,dz= 0.0,0.0,0.0
    pygame.init()
    size = width,height = 480,480
    screen = pygame.display.set_mode(size,
                    pygame.OPENGL | pygame.DOUBLEBUF)
    glLoadIdentity()
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(-10,10,-10,10)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glClearColor(1,1,1,0)
#    ... next slide
```

PyGame : sceneGraph.py

Programme de test

```
    root = SceneNode(
        [TranslationNode((5,2,0),
                         [RotationNode(\
    lambda :pygame.time.get_ticks()/10%360,
                             (1.0,0.0,0.0),
                             [ThingNode(Triangle)]),
                          TranslationNode((-5,-2,0),\
                                          [ThingNode(Quad)])
                          ])
         ])
#    ... next slide
```

PyGame : sceneGraph.py

Programme de test

```
    def render_loop():
        glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT)
        glLoadIdentity()
        glTranslate(dx,dy,dz)
        root.draw()
        glFlush()
        pygame.display.flip()
#   ... next slide
```

PyGame : sceneGraph.py

Programme de test

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            raise ExitException()
        elif event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                raise ExitException()
            elif event.key == K_UP :
                    dy=dy-0.5
            # K_DOWN (dy=dy+0.5) ... and so on
            else :
                pass
    render_loop()
pygame.quit()
```

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 Evénement-Action

Types d'événements

- affichage, clavier, souris
- divers périphériques (tablettes graphiques, spaceballs ...)
- modification de la configuration de la fenêtre
- indépendent 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 comportrement

```
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);
  ...
```

Références bibliographique

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)

Références Internet

Adresses “au Net”

- `www.opengl.org` : le site officiel
- `www.xmission.com/~nate/opengl.html` : Nate Robbins
- `www.linuxgraphic.org/section3d/openGL/didact.html` : les tutoriaux de Xavier Michelon
- `http://ironalbatross.net/wiki/index.php5?title=Python_Simple_SceneGraph` : Graphe de Scène en Pygame
- `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` : entre autre de la 2D/3D