Univers Virtuels

OpenGL : Modélisation / Visualisation

Alexis NEDELEC

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

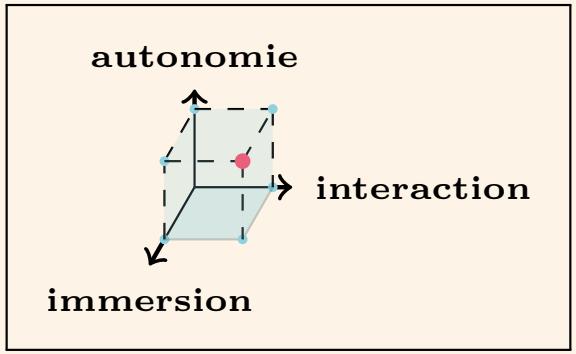
Réalité Virtuelle

De l'immersion et de l'interaction avec un peu ... d'appréhension

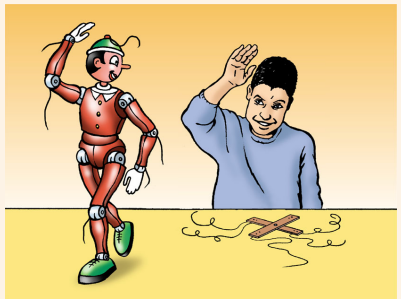
De l'immersion et de l'autonomie avec un peu ... d'humour

Réalité Virtuelle

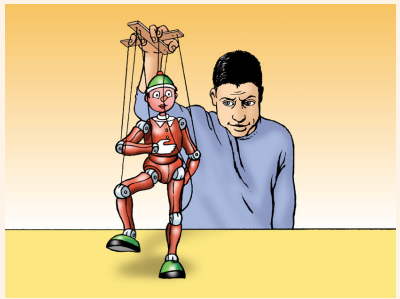
De l'immersion, interaction et autonomie avec un peu de ... CERV



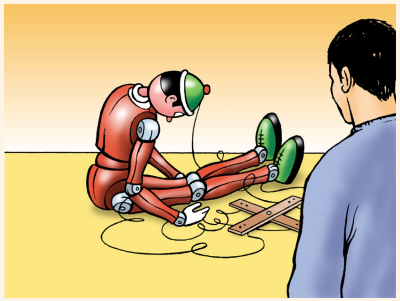
autonomie



interaction



immersion



Réalité Virtuelle

Des environnements virtuels avec un peu de ... philosophie

"La **philosophie** est écrite dans cet immense livre qui se tient toujours ouvert devant nos yeux, je veux dire **l'Univers**, mais on ne peut le comprendre si l'on ne s'applique d'abord à en **comprendre la langue** et à connaître les **caractères** avec lesquels il est écrit. Il est écrit dans la **langue mathématique** et ses caractères sont des **triangles, des cercles et autres figures géométriques**, sans le moyen desquels il est humainement impossible d'en comprendre un mot. Sans eux, c'est une errance vaine dans un labyrinthe obscur."

Galilée, Il Saggiatore (L'Essayeur), 1623

Réalité Virtuelle

Des environnements virtuels avec un peu de ... philosophie

Les cinq éléments de Platon :

1. le feu : **tétra**èdre, constitué de **quatre** faces (triangles équilatéraux) identiques
2. la terre : **hexa**èdre, constitué de **six** faces (carrés) identiques
3. l'air : **octa**èdre, constitué de **huit** faces (triangles équilatéraux) identiques
4. l'univers : **dodéca**èdre, constitué de **douze** faces (pentagones) identiques
5. l'eau : **icosa**èdre, constitué de **vingt** faces (triangles équilatéraux) identiques

`http://therese.eveilleau.pagesperso-orange.fr/pages/truc_mat/indexF.htm`

Environnements virtuels

Les cinq éléments

Polyèdres réguliers

- toutes les faces (polygones) sont identiques et régulières
- tous les sommets sont identiques (même degré, nombre d'arêtes reliant chaque sommet)

Polygone régulier :

- équilatéral (tous les côtés sont de même longueur)
- équiangle (tous les angles sont égaux)

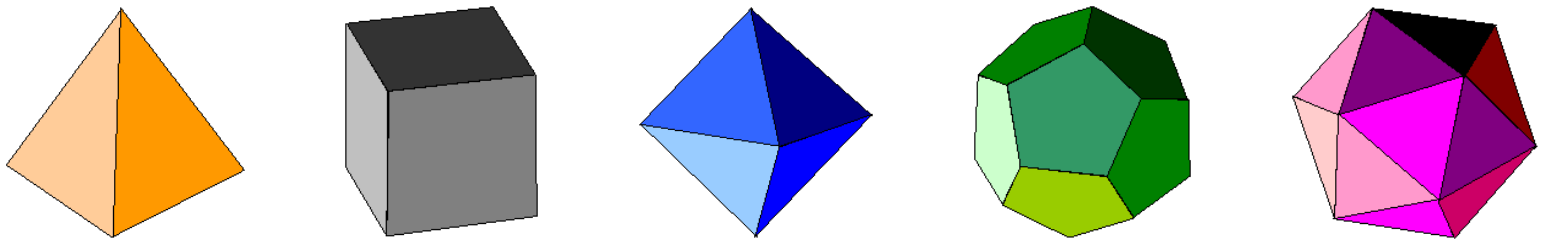
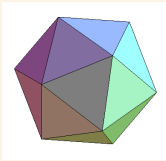
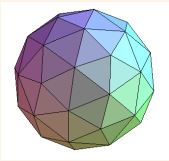
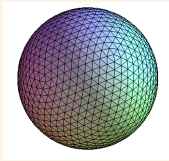


Figure: Polyèdres réguliers

Environnements virtuels

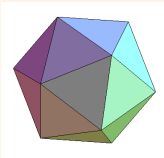
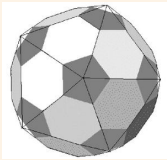
Des objets à leur représentation 3D

Icosaèdre subdivisé : représentation 3D

Icosaèdre subdivisé : Objet réel (géodes)

Icosaèdre tronqué : représentation 3D

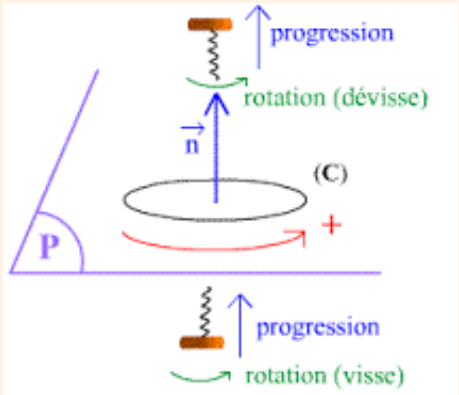



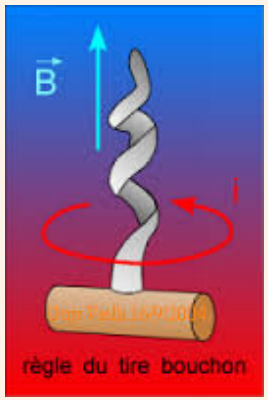
Icosaèdre tronqué : Objet réel (ballons)

Environnements virtuels

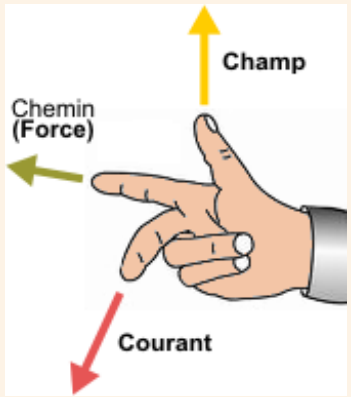
Des objets à leur représentation 3D

Polygone : notion de normale



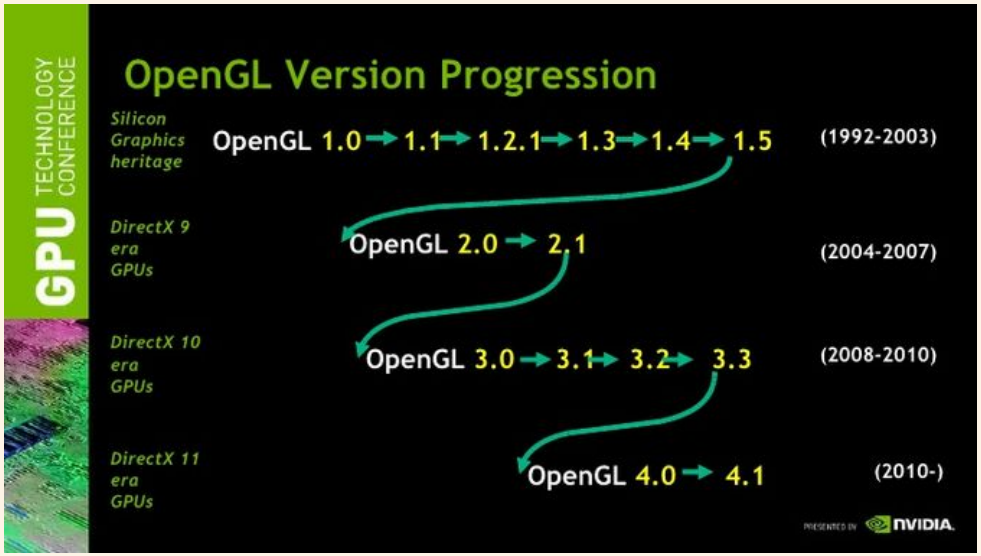


Normales dans "la vie de tous les jours"



OpenGL

Représentation d'objets 3D



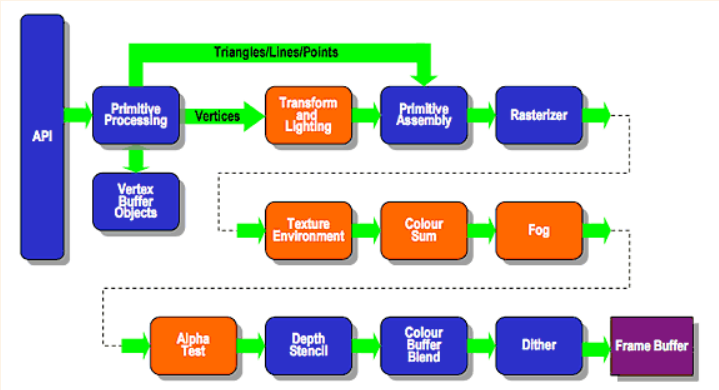
- www.opengl.org/sdk/docs
- www.opengl-tutorial.org/fr/beginners-tutorials
- raphaello.univ-fcomte.fr/IG/Programme2016-2017.htm
- developer.nvidia.com/transitioning-opengl-vulkan

Pipeline graphique

Des objets à leur représentation sur écran

- transformations géométriques : traitement des sommets
- mélange pixels/textures : calcul de pixels par facette d'objets
- rasterisation : affichage des pixels à l'écran

Pipeline graphique fixe



alexandre-laurent.developpez.com/tutoriels/OpenGL/OpenGL-GLSL/?page=page_1

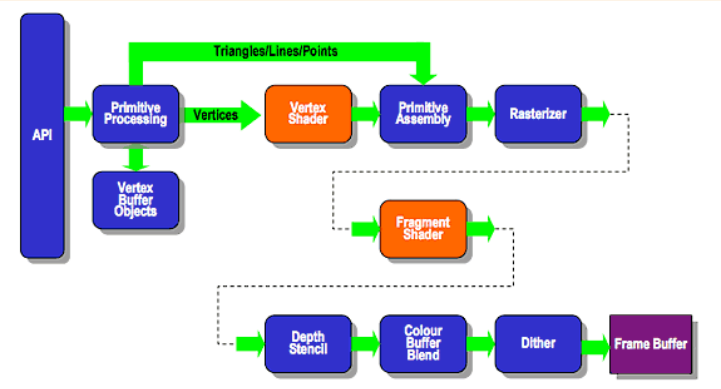
Pipeline graphique

Des objets à leur représentation sur écran

Evolution pour les cartes graphique programmables

- Vertex shader : code remplaçant le pipeline géométrique
- Fragment shader : code remplaçant mélange pixels/textures

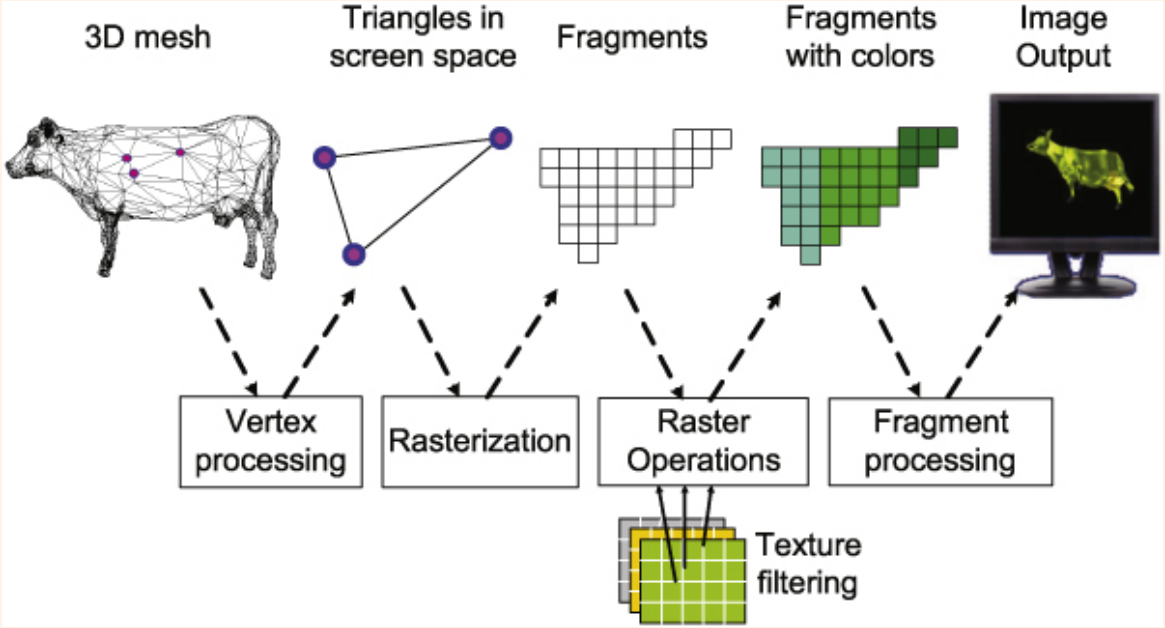
Pipeline graphique programmable



igm.univ-mlv.fr/~vnozick/teaching/slides/ensg/01%20Pipeline_graphique.pdf

Pipeline graphique

Des objets à leur représentation sur écran



Représentation d'environnement 3D

Dans la réalité avec un appareil photo

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

Dans un environnement virtuel avec OpenGL

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

Représentation d'environnement 3D

Dans un environnement virtuel avec OpenGL

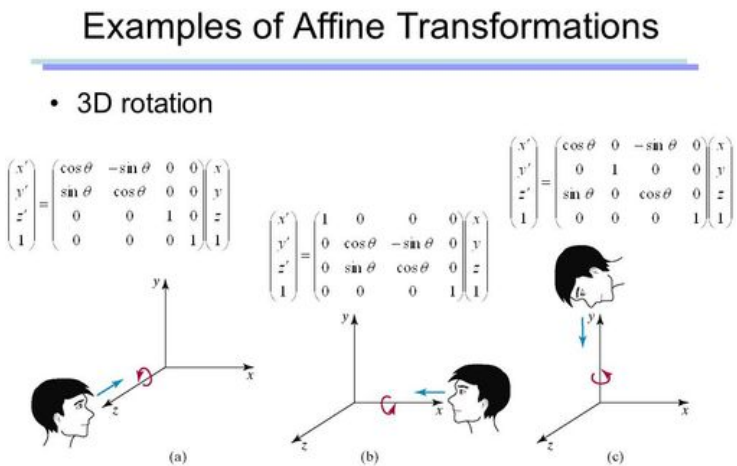
1. placer la caméra virtuelle : `gluLookAt()`
2. composer une scène virtuelle :
   - transformer les objets :
     `glTranslatef()`, `glRotatef()`, `glScalef()`...
   - à créer :
     `glBegin()`.... `glEnd()`, `glutWireCube(1)` ...
3. choisir une projection :
   `glOrtho()`, `glPerspective()`, `glFrustum()`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

Représentation d'environnement 3D

Transformations affines

- translations,rotations,homothéties
- coordonnées et matrices homogènes

Frédéric Legrand : `www.f-legrand.fr/scidoc/docimg/graphie/geometrie/affine/affine.html`



`https://slideplayer.com/slide/9396372/`

Représentation d'environnement 3D

Coordonnées et matrice homogène

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

Composition de transformations

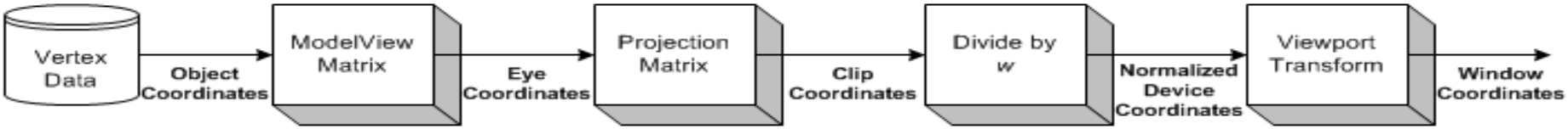
- $[P]$: point de l'espace, $[P]^T$ transposée de $[P]$
- $[M]$: matrice homogène, $[M]^T$: transposée de $[M]$

$[P']$: résultat des transformations successives i $(1 \leq i \leq n)$,

- $[P'] = [M_n] \dots [M_i] \dots [M_1][P]$
- $[P'] = [P]^T[M_1]^T \dots [M_i]^T \dots [M_n]^T$

Pipeline graphique

Transformations successives



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)`

Pipeline graphique

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)`

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

Pipeline graphique

Projection : `glMatrixMode(GL_PROJECTION)`

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

NDC : Normalized Device Coordinates

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

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

Matrices 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)

Modélisation-visualisation

glLoadIdentity()

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

glTranslatef(tx,ty,tz)

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

Modélisation-visualisation

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)

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

Modélisation-visualisation

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 $\overrightarrow{u} = (u_x, u_y, u_z)$

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

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

Matrice active

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)

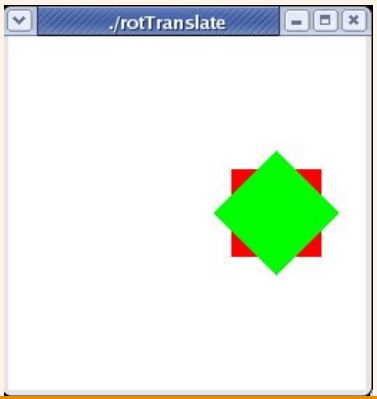
```
glLoadIdentity();
glRotatef(45.0,0.0,0.0,1.0);
glTranslatef(0.0,1.0,0.0);
```

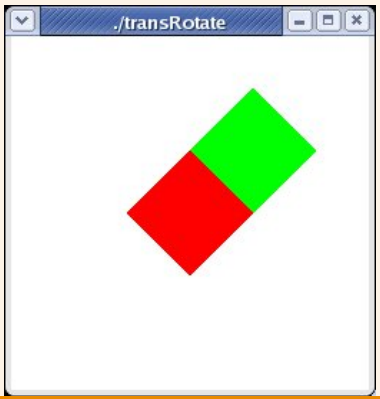
Matrice active

Enchaînement de transformations

```
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 ?

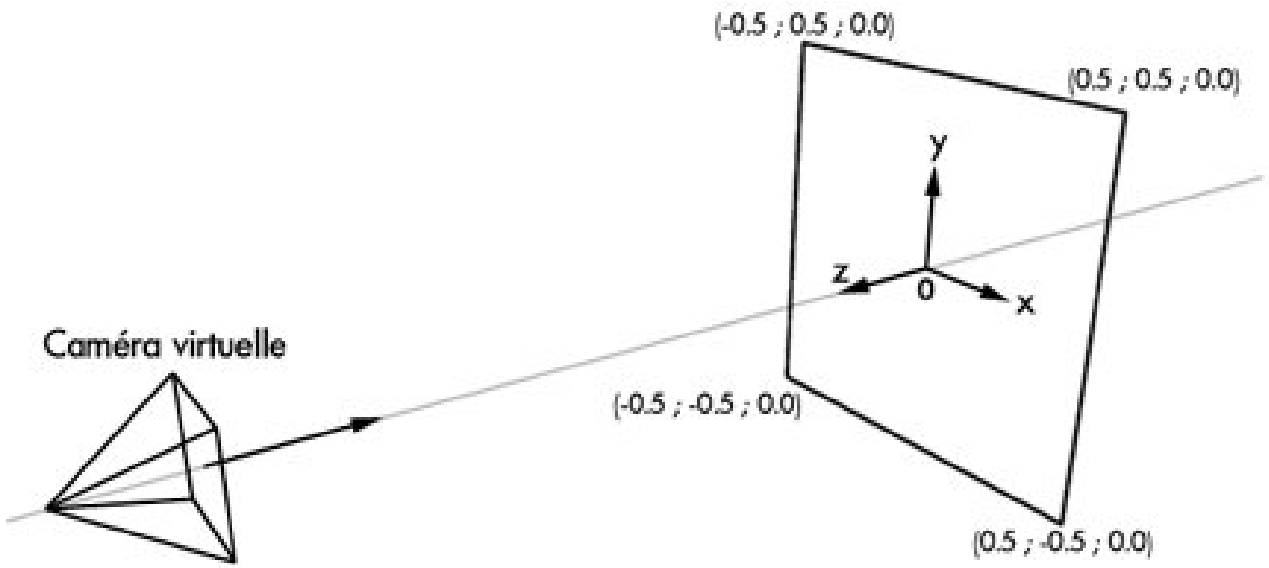




Caméra Virtuelle

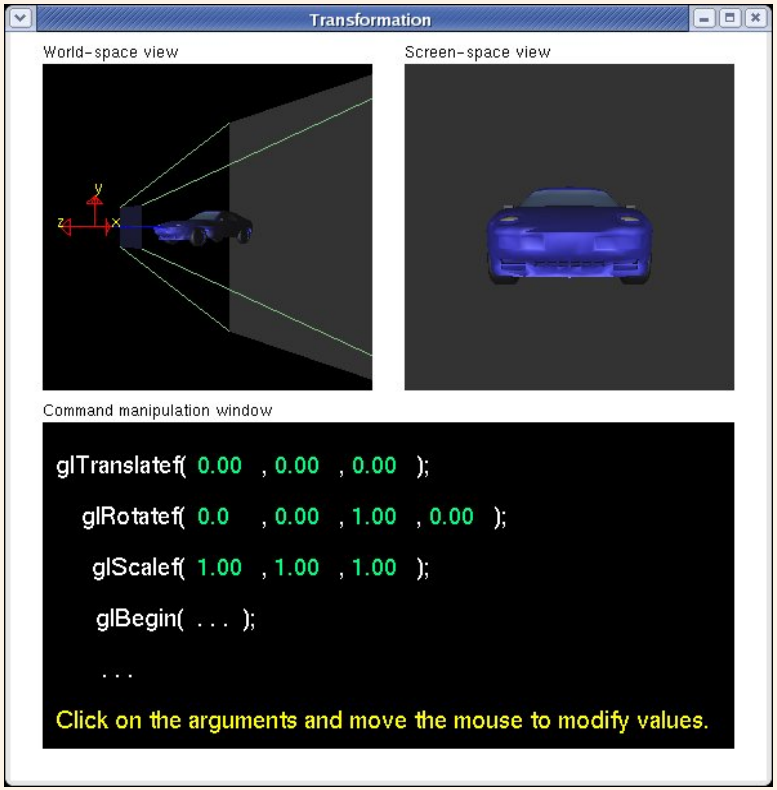
Point d'observation par défaut

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



Caméra Virtuelle

Didacticiel Nate Robbins



Caméra Virtuelle

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();
```

Caméra Virtuelle

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

Volume de visualisation et fenêtrage

- perspective conique : `glFrustum()`, `gluPerspective()`
- perspective orthogonale : `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(-10.0, 10.0, -10.0, 10.0, 1.0, 5.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
```

Volume de visualisation

Projection orthogonale

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

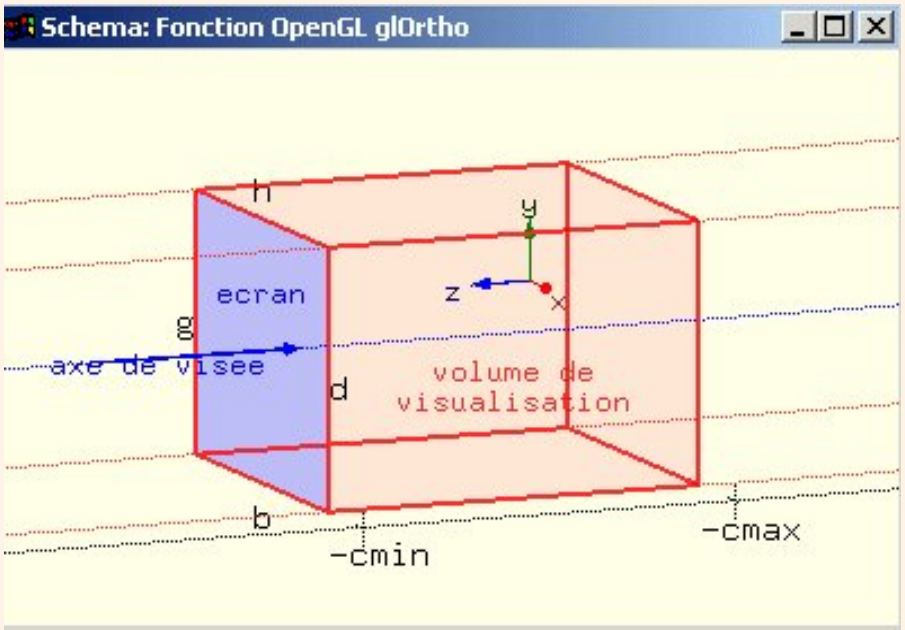
Matrice de projection

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & tx \\ 0 & \frac{2}{top-bottom} & 0 & ty \\ 0 & 0 & \frac{-2}{far-near} & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$tx = \frac{right+left}{right-left}, ty = \frac{top+bottom}{top-bottom}, tz = \frac{far+near}{far-near}$$

Volume de visualisation

Projection orthogonale



http://raphaello.univ-fcomte.fr/IG

Volume de visualisation

Projection en perspective

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

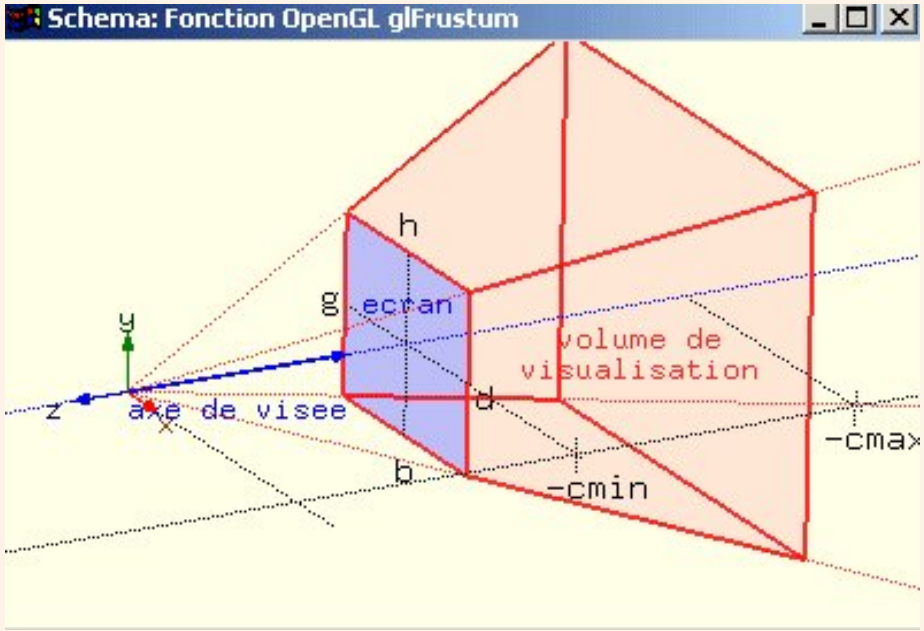
Matrice de projection

$$\begin{bmatrix} \frac{2.near}{right-left} & 0 & A & 0 \\ 0 & \frac{2.near}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$A = \frac{right+left}{right-left}, B = \frac{top+bottom}{top-bottom}, C = -\frac{far+near}{far-near}, D = -\frac{2.near.far}{far-near}$$

Volume de visualisation

Projection en perspective



Volume de visualisation

Projection en perspective

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

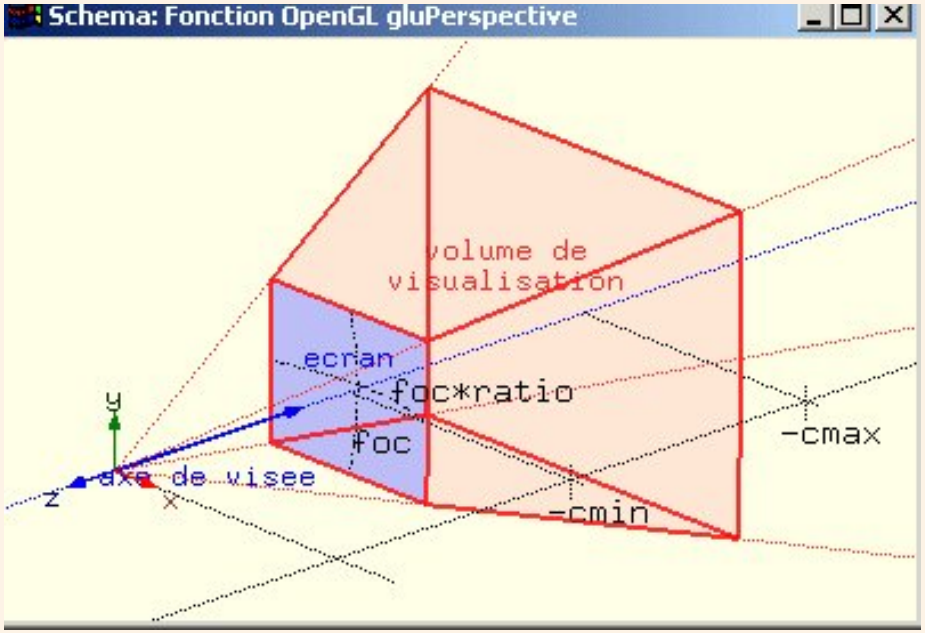
Matrice de projection

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2.near.far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$f = cotangent(\frac{fovy}{2})$

Volume de visualisation

Projection en perspective



Animation

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 fantômatiques"

Animation

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();
}
```

"Double-buffering"

Affichage de scène

```
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();
}
```

"Double-buffering"

Modification de la scène

```
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);
...
}
```

Interaction

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 "callback"

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);
  ...
```

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

Création de scène

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

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

Dessine-moi une roue

```
void roue(double dimension,int boulons) {
  cylindre(dimension);
  float angle = 360.0/boulons;
  for (int i=0;i<boulons;i++) {
    glPushMatrix();
      glRotatef(angle*i, 0.0, 0.0, 1.0);
      glTranslatef(0.75*(dimension/2), 0.0, 0.0);
      cylindre(0.20*dimension);
    glPopMatrix();
  }
}
```

Pile de matrices

Dessine-moi une voiture

```
void voiture(double dimension) {
  carosserie(dimension)
  float x1=0.75*dimension;
  float z1=0.5*dimension;
  glPushMatrix();
    glTranslatef(x1, 0.0, z1);
    roue(0.20*dimension,5);
  glPopMatrix();
  glPushMatrix();
    glTranslatef(x1, 0.0, -z1);
    roue(0.20*dimension,5);
  ...
}
```

PyOpenGL

Modules python

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
...
```

Programme de test

```
if __name__ == "__main__" :
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH)
    glutInitWindowSize(500,500)
    glutInitWindowPosition(100,100)
    glutCreateWindow(sys.argv[0])
    glClearColor(1.0,1.0,1.0,1.0)
```

Gestion des callbaks

Programme de test

```
    glutDisplayFunc(display)
    glutReshapeFunc(reshape)
    glutKeyboardFunc(on_normal_key_action)
    glutMouseFunc(on_mouse_action)
    glutSpecialFunc(on_special_key_action)
#    glutIdleFunc(animation_step)
    glutMainLoop()
```

Programmation de l'application

- affichage de scène : `display()`
- redimensionnement : `reshape()`
- interaction : `on_..._action()`
- animation : `animation_step()`

Gestion de scène

Affichage de scène

```
def display() :
  glClear(GL_COLOR_BUFFER_BIT)
  glLoadIdentity()
  position=[0.0,0.0,1.0]
  direction=[0.0,0.0,0.0]
  viewup=[0.0,0.0,0.0]
  gluLookAt(position[0],position[1],position[2],
            direction[0],direction[1],direction[2],
            viewup[0],viewup[1],viewup[2])
  glColor3ub(255,0,0)
#  roue(1,5)
  voiture(1)
  glutSwapBuffers()
```

Gestion de scène

Création de scène

```
def voiture(dimension) :
  glPushMatrix()
  glScalef(2,1,1)
  glutWireCube(dimension)
  glPopMatrix()
  glTranslatef(0,-0.5,0)
  pos_x=0.75*dimension
  pos_z=0.5*dimension
  glPushMatrix()
  glTranslatef(pos_x,0.0,pos_z)
  roue(0.20*dimension,5)
  glPopMatrix()
  ....
```

Gestion de scène

Création de scène

```
def roue(dimension,boulons) :
  glutWireCube(dimension)
  angle = 360.0/boulons;
  for i in range(boulons) :
      glPushMatrix();
      glRotatef(angle*i,0.0,0.0,1.0)
      glTranslatef(0.70*(dimension/2.0),0.0,0.0)
      glutWireCube(0.20*dimension)
      glPopMatrix()
```

Gestion de scène

Redimensionnement de scène

```
def reshape(w,h) :
  foc,ratio=60.0,w*1.0/h
  near,far=0.1,10.0
  glViewport(0,0,w,h)
  glMatrixMode (GL_PROJECTION)
  glLoadIdentity()
  gluPerspective(foc,ratio,near,far)
  glMatrixMode(GL_MODELVIEW)
```

Gestion des interactions

Interaction clavier

```
def on_normal_key_action(key,x,y) :
    if  key=='a' :
        glutIdleFunc(None)
    elif  key=='A' :
        glutIdleFunc(animation_step)
    elif  key=='h' :
        print("# Help : documentation -- #\n")
        print("a/A : stop/start animation\n")
        ...
    elif  ...  :
        ...
    else :
        ...
    glutPostRedisplay()
```

Gestion des interactions

Interaction clavier

```
def on_special_key_action(key, x, y) :
    if  key==GLUT_KEY_UP :
            print("key up")
    elif  key==GLUT_KEY_DOWN :
            print("key down")
    elif key== GLUT_KEY_LEFT :
            print("key left")
    elif  key== GLUT_KEY_RIGHT :
            print("key right")
    elif  ...  :
        ...
    else :
        ...
    glutPostRedisplay()
```

Gestion des interactions

Interaction souris

```
def on_mouse_action(button, state, x, y) :
    if  button==GLUT_LEFT_BUTTON :
        if state==GLUT_DOWN :
            print("left button down")
    elif  button==GLUT_MIDDLE_BUTTON :
        if state==GLUT_UP :
            print("middle button up")
    elif ... :
        ...
    else :
        ...
    glutPostRedisplay()
```

Graphe de scène

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 scène

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

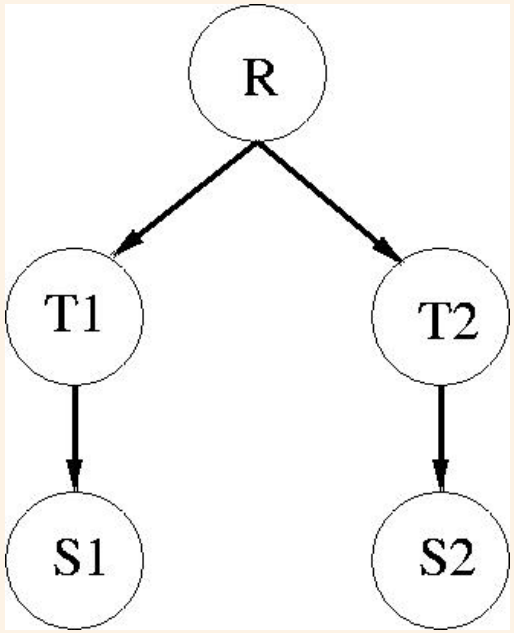
Intérêts

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

Graphe de scène

Exemple

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

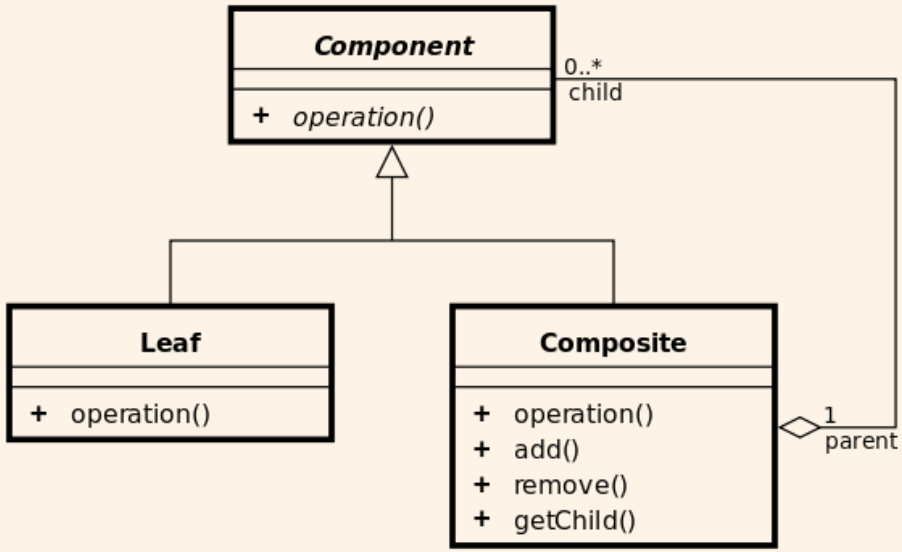


Exemples courants

- voiture, système solaire, humanoides ...

Patron de conception

Modèle `Composite`



Modèle Composite

Module composite.py

```
class Component(object):
    def __init__(self, *args, **kw):
        pass
    def draw(self):
        pass

class Leaf(Component):
    def __init__(self, *args, **kw):
        Component.__init__(self, *args, **kw)
    def draw(self):
        pass
```

Modèle Composite

Composite : aggrégation d'objets

```
class Composite(Component):
    def __init__(self, *args, **kw):
        Component.__init__(self, *args, **kw)
        self.children={}
    def __repr__(self):
        return "<Composite('{}')>".format(self.children)
```

Modèle Composite

Composite : opération (draw())

```
def draw(self):
    for child in self.children.values():
        child.draw()
def add(self,name,child) :
    self.children[name]=child
def remove(self,name):
    del self.children[name]
```

Modèle Composite

Composite : recherche d'objets

```
def get_child(self,name) :
    result=None
    if name in self.children.keys() :
        result=self.children[name]
    else :
        for child in self.children.values():
            if hasattr(child,"get_child") :
                result=child.get_child(name)
                break
    return result
```

Nœuds de Transformation

`scene_graph.py` : code en annexe p.85

Translation (Composite Component)

```
class TranslationNode(Composite):
    def __init__(self,offset,children=None):
        Composite.__init__(self,children)
        self.offset = offset
    def __repr__(self):
        return "<TranslationNode('{}')>".\
               format(self.offset)
    def set_offset(self,offset) :
        self.offset=offset
    def get_offset(self) :
        return self.offset
```

Nœuds de Transformation

Translation (Composite Component)

```
def draw(self):
    glPushMatrix()
    if callable(self.offset):
        glTranslate(*self.offset())
    else:
        glTranslate(*self.offset)
    for child in self.children.values():
        child.draw()
    glPopMatrix()
```

Même principe pour tout Nœud `Composite Component` (cf. p.85)

Primitives d'affichage

Le Point (Leaf Component)

```
class Point(Leaf) :
    def __init__(self,point):
        Leaf.__init__(self)
        self.point=copy.copy(point)
    def __repr__(self):
        return "<Point('{} {} {}')>".format(
          self.point[0],\
          self.point[1],\
          self.point[2]
            )
```

Primitives d'affichage

Le Point (Leaf Component)

```
def set_point(self,point) :
    self.point=copy.copy(point)
def get_point(self) :
    return self.point
def draw():
    glBegin(GL_POINTS)
    x,y,z=self.point
    glVertex(x,y,z)
    glEnd()
```

Primitives d'affichage

La Sphere (Leaf Component)

```
class Sphere(Leaf) :
    def __init__(self,radius,slices,stacks):
        Leaf.__init__(self)
        self.radius=radius
        self.slices=slices
        self.stacks=stacks
```

Primitives d'affichage

La Sphere (`Leaf Component`)

```
    def __repr__(self):
        return "<Sphere({},{},{})>".format(\
                self.radius, self.slices,self.stacks
               )
    def draw(self):
        glutWireSphere(self.radius,\
                       self.slices,\
                       self.stacks)
```

Même principe pour tout Nœud `Leaf Component` (cf. p.85)

Création de scène

La Scene : manipulation du `Model`

```
class Scene :
    def __init__(self,model) :
        self.model=model
        self.frame_rate=20
        self.camera_position=[0.0,0.0,5.0]
        self.camera_direction=[0.0,0.0,0.0]
        self.camera_viewup=[0.0,1.0,0.0]
        self.translation=[0.0,0.0,0.0]
```

Animation de scène

La Scene : animation du `Model`

```
    def animation_step(self):
        """Update animated parameters."""
        rotation=self.model.get_child("TN3-RN2")
        if rotation :
            if rotation.get_angle()>=360.0:
                rotation.set_angle(0.0)
            else :
                rotation.set_angle(rotation.get_angle()
                                   +2)
            sleep(1/float(self.frame_rate))
        glutPostRedisplay()
```

Visualisation de scène

La Scene : volume de visualisation

```
def reshape_scene(self,width,height):
    w,h=glutGet(GLUT_WINDOW_WIDTH),\
        glutGet(GLUT_WINDOW_HEIGHT)
    glViewport(0,0,w,h)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60,w/h,0.1,10)
```

Visualisation de scène

La Scene : affichage de scène

```
    def display_model(self) :
        """Glut display function."""
        glClearColor(0.0,0.0,0.0,0.0);
#        glClearColor(1.0,1.0,1.0,1.0);
        glClear( GL_COLOR_BUFFER_BIT |
                 GL_DEPTH_BUFFER_BIT )
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_COLOR_MATERIAL);
```

Visualisation de scène

La Scene : affichage de scène

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
pos_x,pos_y,pos_z=self.camera_position
dir_x,dir_y,dir_z=self.camera_direction
up_x,up_y,up_z=self.camera_viewup
gluLookAt(self.pos_x,self.pos_y,self.pos_z,
          self.dir_x,self.dir_y,self.dir_z,
          self.up_x,self.up_y,self.up_z)
tx,ty,tz=self.translation
glTranslatef(tx,ty,tz)
self.model.draw()
glutSwapBuffers()
```

Interaction avec la scène

La Scene : interaction clavier

```
def on_normal_key_action(self,key, x, y) :
    if  key == 'a' :
       glutIdleFunc(None)
    elif  key == 'A' :
       glutIdleFunc(self.animation_step)
    elif  key == 'x' :
      self.translation[0]+=0.1
    elif  key == 'X' :
      self.translation[0]-=0.1
    ...
    glutPostRedisplay()
```

Interaction avec la scène

La Scene : interaction clavier

```
def on_special_key_action(self,key, x, y) :
    if key==GLUT_KEY_UP :
        self.camera_position[1]+=0.1
    elif key==GLUT_KEY_DOWN :
        self.camera_position[1]-=0.1
    ...
    glutPostRedisplay()
```

Interaction avec la scène

La Scene : interaction souris

```
    def on_mouse_action(self, button, state, x, y):
        # print(button) left:0, middle:1, right:2
        # middle-far:3,middle-near:4
        if rotation :
            if rotation.get_angle()>=360.0:
                rotation.set_angle(0.0)
            angle_step=5.0
            if button==0:
                else :
                    rotation.set_angle(
                                rotation.get_angle() \
                                 +angle_step )
        ...
        glutPostRedisplay()
```

Programme d'application

Le Modele (Composite Component)

```
class Model(Composite) :
    def __init__(self,children=None):
        Composite.__init__(self,children)
```

Initialisation OpenGL

```
if __name__ == "__main__" :
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH \
                        | GLUT_DOUBLE)
    glutInitWindowSize(1200,1000)
    glutInitWindowPosition(100,100)
    glutCreateWindow(sys.argv[0])
```

Programme d'application

Création du graphe de scène

```
model=Model()
p1=Point((0.5,0.5,0.0))
p2=Point((0.5,-0.5,0.0))
p3=Point((-0.5,-0.5,0.0))
p4=Point((-0.5,0.5,0.0))
quad=Quadrilatere(p1,p2,p3,p4)
model.add("QL1",quad)
rotate=RotationNode(0,(0,0,1))
model.add("RN1",rotate)
translate=TranslationNode((0.0,-2.0,0.0))
rotate.add("RN1-TN1",translate)
quad=Sphere(0.5,10,20)
translate.add("RN1-TN1-QL2",quad)
```

Programme d'application

Création du graphe de scène

```
translate=TranslationNode((1.0,1.0,0.0))
model.add("TN2",translate)
p1=Point((0.0,0.8,0.0))
p2=Point((0.2,-0.2,0.0))
p3=Point((-0.2,-0.2,0.0))
triangle=Triangle(p1,p2,p3)
translate.add("TN2-TL1",triangle)
```

Programme d'application

Création du graphe de scène

```
translate=TranslationNode((-1.0,1.0,0.0))
model.add("TN3",translate)
rotate=RotationNode(0,(0,0,1))
translate.add("TN3-RN2",rotate)
triangle=Triangle(p1,p2,p3)
rotate.add("TN3-RN2-TL2",triangle)
```

Programme d'application

Gestion des callbacks

```
scene=Scene3D(model)
glutReshapeFunc(scene.reshape_scene)
glutDisplayFunc(scene.display_model)
glutIdleFunc(scene.animation_step)
glutKeyboardFunc(scene.on_normal_key_action)
glutSpecialFunc(scene.on_special_key_action);
glutMouseFunc(scene.on_mouse_action)
glutMainLoop()
```

Conclusion

A retenir

- OpenGL 2.1, OpenGL 4.5, SL, ES, ES Next, Vulkan...
- Matrices homogènes
- pipeline graphique
- Modélisation/visualisation
- interaction, animation (callbacks)
- pile de matrices, graphe de scènes
- ....

Annexe : Graphe de scène

Module scene_graph.py

```
from sys import argv, exit
from time import sleep
import copy,math
from composite import *

try:
    from OpenGL.GLUT import *
    from OpenGL.GL import *
    from OpenGL.GLU import *
except:
    print ("Error: PyOpenGL not installed properly !!")
    sys.exit()
```

Annexe : Graphe de scène

Nœud de Translation

```
class TranslationNode(Composite):
    def __init__(self,offset,children=None):
        Composite.__init__(self,children)
        self.offset = offset
    def __repr__(self):
        return "<TranslationNode('{}')>".\
               format(self.offset)
    def set_offset(self,offset) :
        self.offset=offset
    def get_offset(self) :
        return self.offset
```

Annexe : Graphe de scène

Nœud de Translation

```
def draw(self):
    glPushMatrix()
    if callable(self.offset):
        glTranslate(*self.offset())
    else:
        glTranslate(*self.offset)
    for child in self.children.values():
        child.draw()
    glPopMatrix()
```

Annexe : Graphe de scène

Nœud de Rotation

```
class RotationNode(Composite):
    def __init__(self,angle,axe,children=None):
        Composite.__init__(self,children)
        self.angle=angle
        self.axe=copy.copy(axe)
    def __repr__(self):
        return "<RotationNode(\
                  'angle:{},\
                   axe:{} {} {}')>".format(
                                       self.angle,
                                       self.axe[0],
                                       self.axe[1],
                                       self.axe[2]
                                   )
```

Annexe : Graphe de scène

Nœud de Rotation

```
    def set_angle(self,angle) :
        self.angle=angle
    def get_angle(self) :
        return self.angle
    def set_axe(self,axe) :
        self.axe=copy.copy(axe)
    def get_axe(self) :
        return self.axe
```

Annexe : Graphe de scène

Nœud de Rotation

```
def draw(self):
    glPushMatrix()
    rx,ry,rz=self.axe
    if callable(self.angle):
        glRotate(self.angle(),rx,ry,rz)
    else:
        glRotate(self.angle,rx,ry,rz)
    for child in self.children.values():
        child.draw()
    glPopMatrix()
```

Annexe : Graphe de scène

Nœud de mise à l'échelle

```
class ScaleNode(Composite):
    def __init__(self,factor,children=None):
        Composite.__init__(self,children)
        self.factor = factor
    def __repr__(self):
       return "<ScaleNode('{}')>".format(self.factor)
    def set_factor(self,factor) :
        self.factor=factor
    def get_factor(self) :
        return self.factor
```

Annexe : Graphe de scène

Nœud de mise à l'échelle

```
    def draw(self):
        glPushMatrix()
        if callable(self.factor):
            glScale(*self.factor())
        else:
            glScale(*self.factor)
        for child in self.children.values() :
            child.draw()
        glPopMatrix()
```

Primitives d'affichage

Le Point

```
class Point(Leaf) :
    def __init__(self,point):
        Leaf.__init__(self)
        self.point=copy.copy(point)
    def __repr__(self):
        return "<Point('{} {} {}')>".format(
          self.point[0],\
          self.point[1],\
          self.point[2]
            )
```

Primitives d'affichage

Le Point

```
def set_point(self,point) :
    self.point=copy.copy(point)
def get_point(self) :
    return self.point

def draw():
    glBegin(GL_POINTS)
    x,y,z=self.point
    glVertex(x,y,z)
    glEnd()
```

Primitives d'affichage

Le Triangle

```
class Triangle(Leaf) :
    def __init__(self,p1,p2,p3):
        Leaf.__init__(self)
        self.p1,self.p2,self.p3=p1,p2,p3
    def __repr__(self):
        return "<Triangle('{}, {}, {}')>".format(\
                    self.p1,self.p2,self.p3
            )
    def set_points(self,p1,p2,p3) :
        self.p1,self.p2,self.p3=p1,p2,p3
    def get_points(self) :
        return self.p1,self.p2,self.p3
```

Primitives d'affichage

Le Triangle

```
def draw(self):
    glBegin(GL_TRIANGLES)
    x1,y1,z1=self.p1.get_point()
    x2,y2,z2=self.p2.get_point()
    x3,y3,z3=self.p3.get_point()
    glVertex(x1,y1,z1)
    glVertex(x2,y2,z2)
    glVertex(x3,y3,z3)
    glEnd()
```

Primitives d'affichage

Le Quadrilatere

```
class Quadrilatere(Leaf) :
    def __init__(self,p1,p2,p3,p4):
        Leaf.__init__(self)
        self.p1,self.p2,self.p3,self.p4=p1,p2,p3,p4
    def __repr__(self):
        return "<Quadrilatere('{}, {}, {},{}')>".format(\
                    self.p1,self.p2,self.p3,self.p4
                )
    def set_points(self,p1,p2,p3,p4) :
        self.p1,self.p2,self.p3,self.p4=p1,p2,p3,p4
    def get_points(self) :
        return self.p1,self.p2,self.p3,self.p4
```

Primitives d'affichage

Le Quadrilatere

```
def draw(self):
    glBegin(GL_QUADS)
    x1,y1,z1=self.p1.get_point()
    x2,y2,z2=self.p2.get_point()
    x3,y3,z3=self.p3.get_point()
    x4,y4,z4=self.p4.get_point()
    glVertex(x1,y1,z1)
    glVertex(x2,y2,z2)
    glVertex(x3,y3,z3)
    glVertex(x4,y4,z4)
    glEnd()
```

Primitives d'affichage

Le Cube

```
class Cube(Leaf) :
    def __init__(self,size):
        Leaf.__init__(self)
        self.size=size
    def __repr__(self):
        return "<Cube({})>".format(self.size)
    def draw(self):
        glutWireCube(self.size)
```

Primitives d'affichage

La Sphere

```
class Sphere(Leaf) :
    def __init__(self,radius,slices,stacks):
        Leaf.__init__(self)
        self.radius=radius
        self.slices=slices
        self.stacks=stacks
    def __repr__(self):
        return "<Sphere({},{},{})>".format(\
                self.radius, self.slices,self.stacks
               )
    def draw(self):
        glutWireSphere(self.radius,\
                       self.slices,\
                       self.stacks)
```

Primitives d'affichage

Le Cylindre

```
class Cylindre(Leaf) :
    def __init__(self,base,height,slices,stacks):
        Leaf.__init__(self)
        self.base=base
        self.height=height
        self.slices=slices
        self.stacks=stacks
```

Primitives d'affichage

Le Cylindre

```
def __repr__(self):
    return "<Cylindre {},{},{})>".format(\
          self.base,self.height,\
          self.slices,self.stacks)
def draw(self):
    glutWireCone(self.base,self.height,\
                 self.slices,self.stacks)
```

Références Internet

Adresses "au Net"

- www.opengl.org
- www.opengl.org/sdk/docs
- http://www.openglsuperbible.com
- glprogramming.com/red
- iel.ucdavis.edu/projects/chopengl/demos.html
- www.opengl-tutorial.org/fr/beginners-tutorials
- raphaello.univ-fcomte.fr/IG/Programme2016-2017.htm
- alexandre-laurent.developpez.com/tutoriels/OpenGL/OpenGL-GLSL/?page=page_1

Références Internet

Adresses “au Net”

- igm.univ-mlv.fr/~vnozick/teaching/slides/ensg/01%20Pipeline_graphique.pdf
- www.codeproject.com/Articles/771225/Learning-Modern-OpenGL
- ironalbatross.net/wiki/index.php5?title=Python_Simple_SceneGraph
- nehe.gamedev.net
- paulbourke.net
- www.developpez.com
- Ressources documentaires de l’ENIB