

## Univers Virtuels

### OpenGL : Modélisation / Visualisation

*Alexis NÉDÉLEC*

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

*enib ©2022*



# Réalité Virtuelle

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

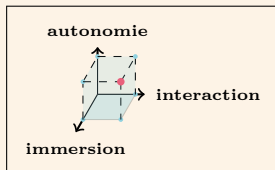


De l'immersion et de l'autonomie avec un peu ... d'humour



# Réalité Virtuelle

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



## La métaphore de Pinocchio [Tisseau : In virtuo]

**immersion**



**interaction**



**autonomie**



# Environnements virtuels

## 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

# Environnements virtuels

## Des environnements virtuels avec un peu de ... philosophie

Les cinq éléments de Platon :

- ❶ le feu : **tétraèdre**, constitué de **quatre** faces (triangles équilatéraux) identiques
- ❷ la terre : **hexaèdre**, constitué de **six** faces (carrés) identiques
- ❸ l'air : **octaèdre**, constitué de **huit** faces (triangles équilatéraux) identiques
- ❹ l'univers : **dodécaèdre**, constitué de **douze** faces (pentagones) identiques
- ❺ l'eau : **icosaèdre**, constitué de **vingt** faces (triangles équilatéraux) identiques

[http://therese.eveilleau.pagesperso-orange.fr/pages/truc\\_mat/indexF.htm](http://therese.eveilleau.pagesperso-orange.fr/pages/truc_mat/indexF.htm)

# Environnements virtuels

## Polyèdres réguliers : les cinq éléments

- 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)

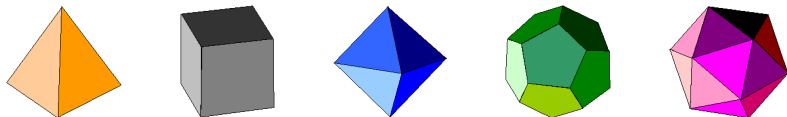


Figure: Polyèdres réguliers

# Environnements virtuels

## Des objets physiques à leur représentation 3D (polyèdres)

Icosaèdre subdivisé : Objet réel (géodes)



Icosaèdre subdivisé : représentation 3D



Icosaèdre tronqué : Objet réel (ballons)



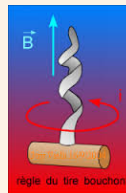
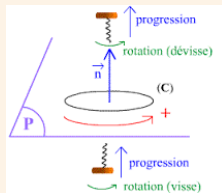
Icosaèdre tronqué : représentation 3D



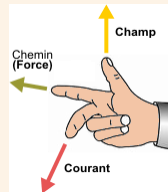
# Environnements virtuels

## Des objets physiques à leur représentation 3D (polyèdres)

### Polygone : notion de normale



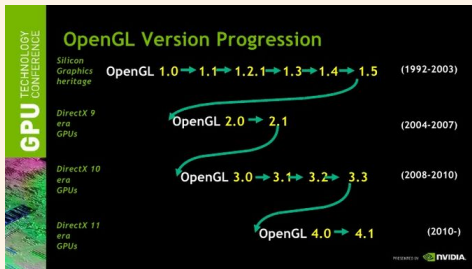
### Normales dans "la vie de tous les jours"





# OpenGL : Historique

## Des objets 3D (polyèdres) à leur représentation sur écran



- OpenGL : documentation
- OpenGL : tutoriaux
- OpenGL : cours de Nicolas JANEY
- d'OpenGL à Vulkan

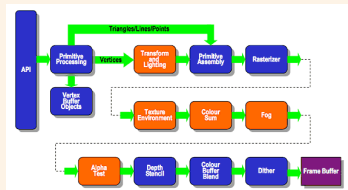
# OpenGL : Pipeline graphique

## Des objets 3D (polyèdres) 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

Cf. Annexes code visualisation de triangle (p.89)

## Pipeline graphique fixe



Alexandre Laurent : Les shaders dans OpenGL

# OpenGL : Pipeline graphique

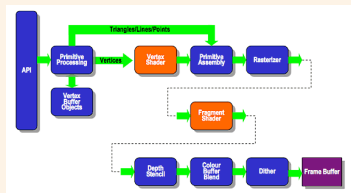
## Des objets 3D (polyèdres) à leur représentation sur écran

Evolution pour les cartes graphique programmables

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

Cf. Annexes code (Parwiz Forogh) visualisation de triangle (p.92)

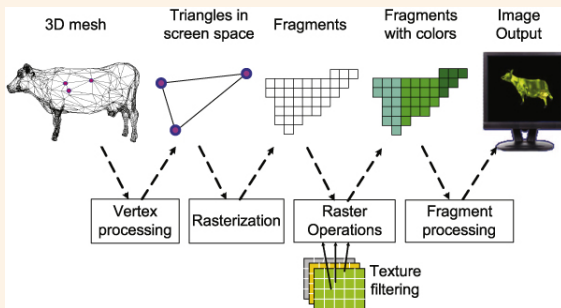
## Pipeline graphique programmable



Alexandre Laurent : Les shaders dans OpenGL

# OpenGL : Pipeline graphique

## Des objets 3D (polyèdres) à leur représentation sur écran



Vincent Nozick : cours sur le pipeline graphique

# Création d'environnement 3D

## Dans la réalité avec un appareil photo

- ➊ Positionner l'appareil photo
- ➋ Arranger les éléments d'une scène à photographier
- ➌ Choisir la focale de l'appareil photo
- ➍ Choisir la taille de la photographie au développement

## Dans un environnement virtuel avec OpenGL

- ➊ modélisation (**Model**) : création de scène
- ➋ visualisation (**View**) : position de caméra
- ➌ projection (**Projection**) : réglage de visualisation
- ➍ affichage (**Viewport**) : taille de l'image résultante

# Création d'environnement 3D

## Dans un environnement virtuel avec OpenGL

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

# Création d'environnement 3D

## Transformations affines

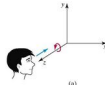
- translations, rotations, homothéties
- coordonnées et matrices homogènes

Frédéric Legrand : Transformations affines

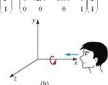
## Exemple : Matrices de rotation autour des trois axes

Examples of Affine Transformations


- 3D rotation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$


(a)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$


(b)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$


(c)

Jehee Lee : Geometric Transformations

# Création d'environnement 3D

## Coordonnées et matrice homogène

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

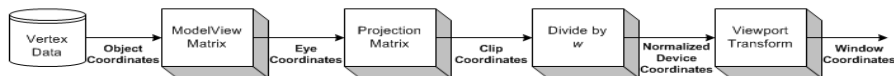
## Composition de transformations

- $[P]$  : 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



Song Ho Ahn : OpenGL Transformation

## 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)

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

# 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( $\theta$ ,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( $\theta$ ,0,1,0)`

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

# Modélisation-visualisation

`glRotatef( $\theta, 0, 0, 1$ )`

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

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

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

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

# Matrice active

## Enchaînement de transformation

- ❶ `glTranslatef(float dx, float dy, float dz) :`  
 $[M_a] = [M_a][M_t]$ , translation  $(dx, dy, dz)$
- ❷ `glRotatef(float theta, float x, float y, float z) :`  
 $[M_a] = [M_a][M_\theta]$ , rotation de  $\theta$  autour de l'axe  $(0, x, y, z)$
- ❸ `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);
```

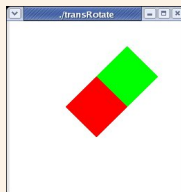
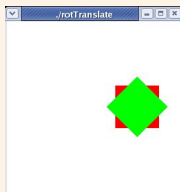


# Matrice active

## Enchaînement de transformations

```
glLoadIdentity();  
glTranslatef(0.5,0.0,0.0);  
glColor3f(1.0,0.0,0.0);           // red square  
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);           // green square  
glRectf(-0.25, -0.25, 0.25, 0.25);
```

## Visualisation de transformation ?



# Application OpenGL

## Création de scène

```
void create_scene()
{
    glTranslatef(0.5,0.0,0.0);
    glColor3f(1.0,0.0,0.0);           // red square
    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);           // green square
    glRectf(-0.25, -0.25, 0.25, 0.25);

}
```

# Application OpenGL

## Fonction d'affichage

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT); // init buffer
    glClearColor(1.0,1.0,1.0,1.0); // white color
    glLoadIdentity();             // init Modelview matrix
    create_scene();
    glFlush();                     // asks to display scene
}
```

## Affichage OpenGL

```
void glut_event()
{
    glutDisplayFunc(display); // to Display scene
}
```

# Application OpenGL

## Point d'entrée de l'application

```
#include <stdlib.h>
#include <GL/glut.h>

void glut_init(int argc, char **argv);
void glut_event(void);

int main(int argc, char **argv)
{
    glut_init(argc, argv);
    glut_event();
    glutMainLoop();
    return 0;
}
```

# Application OpenGL

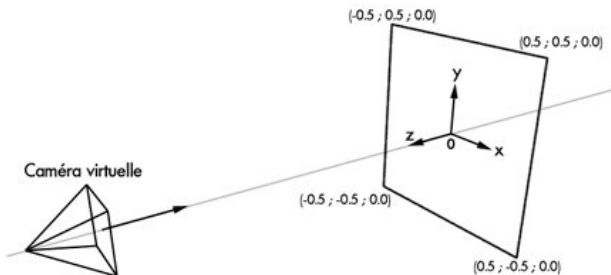
## Initialisation OpenGL utility toolkit (**glut...**)

```
void glut_init(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowPosition(200, 200);
    glutInitWindowSize(250, 250);
    glutCreateWindow(argv[0]);
}
```

# Caméra Virtuelle

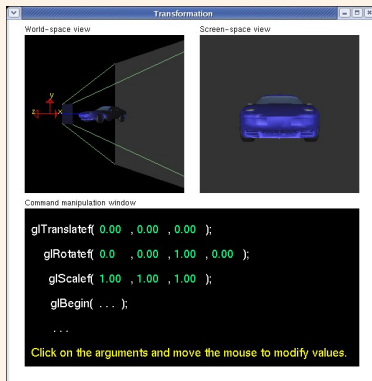
## Point d'observation par défaut

- 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);  
}
```

# Volume de visualisation

## Projection orthogonale

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

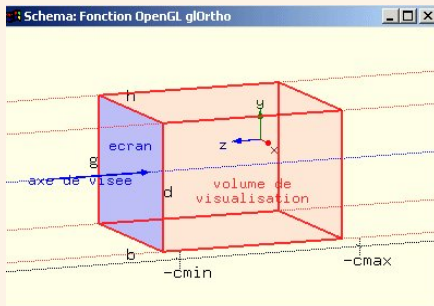
## Matrice de projection

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

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

# Volume de visualisation

## Projection orthogonale



OpenGL : cours de Nicolas JANEY

# Volume de visualisation

## Projection en perspective

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

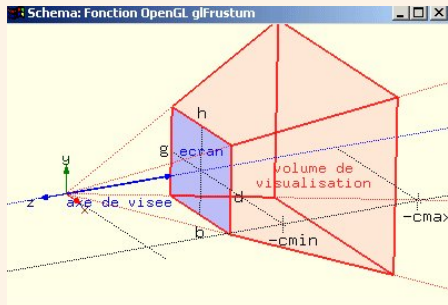
## Matrice de projection

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

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

# Volume de visualisation

## Projection en perspective



# Volume de visualisation

## Projection en perspective

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

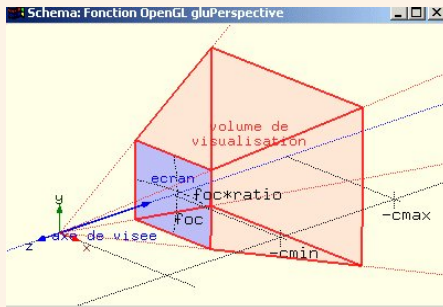
## Matrice de projection

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

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

# 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 fantomatiques”

# 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 create_scene()
{
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0,1.0,1.0);
    glRectf(-0.5,-0.5,0.5,0.5);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    create_scene();
    glutSwapBuffers();
}
```

# "Double-buffering"

## Modification de la scène

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

## Gestion de l'animation

```
void glut_event()  
{  
    ...  
    glutMouseFunc(on_mouse_action);  
    ...  
}
```

# "Double-buffering"

## Gestion de l'animation

```
void on_mouse_action(int button,int state,int x,int y)
{
    switch(button) {
        case GLUT_LEFT_BUTTON :
            if (state == GLUT_DOWN)
                glutIdleFunc(animate);
            break;
        case GLUT_MIDDLE_BUTTON :
            if (state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        // case ... :
    }
```

# Interaction

## 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) {  
    ...  
    glut_event(); // glut...Func(...);  
    ....  
    glutMainLoop();  
    return(0);  
}
```

# 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(0.0,0.0,10.0,0.0,0.0,0.0,0.0,1.0,0.0);  
glTranslatef(1.0,0.0,1.0);  
dessineObjet();  
glLoadIdentity();  
glTranslatef(5.0,0.0,0.0);  
dessineObjet();
```

## Problème de la matrice active

- `glLoadIdentity()` : réinitialisation de la matrice

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

```
glPushMatrix();  
    glTranslatef(1.0,0.0,1.0);  
    dessineObjet();  
glPopMatrix();  
glPushMatrix();  
    glTranslatef(5.0,0.0,0.0);  
    dessineObjet();  
glPopMatrix();
```

# 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 callbacks

## 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

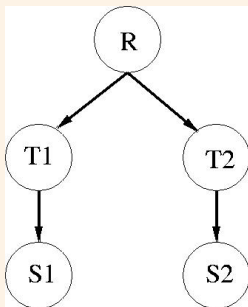
## Intérêts

- 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();
```

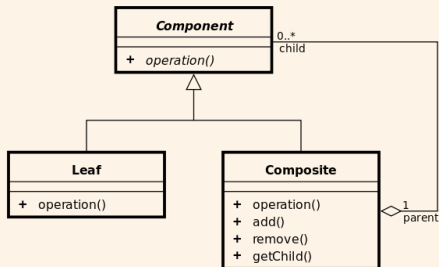


## Exemples courants

- voiture, système solaire, humanoïdes ...

# 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.98

## Nœud de Translation (Composite Component)

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

# Nœuds de Transformation

## Nœud de 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.98)

# 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.98)

# Création de scène

## Le Modele (Composite Component)

```
from composite import *
from scene_graph import *

class Model(Composite) :
    def __init__(self,children=None):
        Composite.__init__(self,children)
        self.animation_angle=0.0
    def get_animation_angle(self) :
        return self.animation_angle
    def set_animation_angle(self,angle) :
        self.animation_angle=angle
```

# 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]  
        self.rotation_node=None
```

# Animation de scène

## La Scene : animation du Model

```
def animation_step(self):  
    """Update animated parameters."""  
    self.rotation_node=self.model.get_child("RN1")  
    # self.rotation_node=self.model.get_child("TN3-RN2")  
    if self.rotation_node :  
        if self.rotation_node.get_angle()>=360.0:  
            self.rotation_node.set_angle(0.0)  
        else :  
            self.rotation_node.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 == b"a" :  
        glutIdleFunc(None)  
    elif key == b"A" :  
        glutIdleFunc(self.animation_step)  
    elif key == b"x" :  
        self.translation[0] += 0.1  
    elif key == b"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 self.rotation_node :  
        if self.rotation_node.get_angle()>=360.0:  
            self.rotation_node.set_angle(0.0)  
        angle_step=5.0  
        if button==0:  
            self.rotation_node.set_angle(\  
                self.rotation_node.get_angle()+angle_step\  
            )  
        elif button==1:  
            ...  
        glutPostRedisplay()
```

# Programme d'application

## 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()
leaf=Quadrilatere()
model.add("Quad1",leaf)

rotate=RotationNode(0,(0,0,1))
model.add("RN1",rotate)
translate=TranslationNode((0.0,-2.0,0.0))
rotate.add("RN1-TN1",translate)
leaf=Cube(0.5)
# leaf=Sphere(0.5,10,20)
# leaf=Torus(0.25,0.5,10,20)
# leaf=Cone(0.5,1.0,10,20)
translate.add("RN1-TN1-Cube",leaf)
```

# Programme d'application

## Création du graphe de scène

```
translate=TranslationNode((1.0,1.0,0.0))  
model.add("TN2",translate)  
leaf=Triangle()  
translate.add("TN2-Triangle1",leaf)  
translate=TranslationNode((-1.0,1.0,0.0))  
model.add("TN3",translate)  
rotate=RotationNode(0,(0,0,1))  
translate.add("TN3-RN2",rotate)  
leaf=Triangle()  
rotate.add("TN3-RN2-Triangle2",leaf)
```

# 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 (évolution 2.1, 3.3, 4.6, Vulkan...)
- Matrices homogènes
- pipeline graphique
- Modélisation/visualisation
- interaction, animation (callbacks)
- pile de matrices, graphe de scènes
- ....



# PyOpenGL : programme de base

Création d'un triangle sans shaders

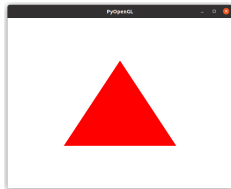
## Importation modules python

```
# -*- coding: utf-8 -*-  
from sys import argv, exit  
from time import sleep  
import math  
try:  
    from OpenGL.GL import *  
    from OpenGL.GLU import *  
    from OpenGL.GLUT import *  
except:  
    print ("Error: PyOpenGL not installed properly !!")  
    sys.exit()
```

# PyOpenGL : programme de base

## Création de triangle

```
def init(size=0.5) :  
    glBegin(GL_TRIANGLES)  
    # glBegin(GL_POLYGON)  
    glColor3f(1.0,0.0,0.0)  
    glVertex2f(-size,-size)  
    glVertex2f(size,-size)  
    glVertex2f(0,size)  
    glEnd()
```



# PyOpenGL : programme de base

## Fonction d'affichage de la scène

```
def display() :  
    glClearColor(1.0,1.0,1.0, 1)  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)  
    init()  
    glFlush()
```

## Programme de tests

```
if __name__ == "__main__" :  
    glutInit(sys.argv)  
    glutInitWindowSize(640, 480)  
    glutCreateWindow("PyOpenGL")  
    glutDisplayFunc(display)  
    glutMainLoop()
```

# PyOpenGL : shaders

Création d'un triangle avec shaders (Parwiz Forogh)

## Importation modules python

```
# -*- coding: utf-8 -*-  
from sys import argv, exit  
from time import sleep  
import numpy as np  
try:  
    from OpenGL.GL import *  
# from OpenGL.GL import shaders  
    from OpenGL.GLU import *  
    from OpenGL.GLUT import *  
    import math  
except:  
    print ("Error: PyOpenGL not installed properly !!")  
    sys.exit()
```

# PyOpenGL : shaders

## Définition des shaders

```
VERTEX_SHADER = """
#version 330
    in vec4 position;
    void main() {
        gl_Position = position;
    }
"""

FRAGMENT_SHADER = """
#version 330
    void main() {
        gl_FragColor =
        vec4(1.0f, 0.0f,0.0f,1.0f);
    }
"""
```

# PyOpenGL : shaders

## Compilation des shaders, création du triangle

```
def init():
    vertexshader=shaders.compileShader(VERTEX_SHADER,\
                                       GL_VERTEX_SHADER)
    fragmentshader=shaders.compileShader(FRAGMENT_SHADER,\
                                       GL_FRAGMENT_SHADER)
    shaderProgram=shaders.compileProgram(vertexshader,\
                                       fragmentshader)

    triangles=[-0.5,-0.5,0.0,
               0.5,-0.5,0.0,
               0.0,0.5,0.0]
    triangles = np.array(triangles,dtype=np.float32)
```

# PyOpenGL : shaders

## Vertex Buffer Object (VBO)

```
VBO = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER,VBO)
glBufferData(GL_ARRAY_BUFFER,triangles.nbytes,\
             triangles, GL_STATIC_DRAW)
position = glGetAttribLocation(shaderProgram,\
                                'position')
glVertexAttribPointer(position,3,\
                      GL_FLOAT,GL_FALSE,0,None)
glEnableVertexAttribArray(position)
```

# PyOpenGL : shaders

## Fonction d'affichage de la scène

```
def display():  
    global shaderProgram  
    glClearColor(1.0,1.0,1.0,1)  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)  
    glUseProgram(shaderProgram)  
    glDrawArrays(GL_TRIANGLES,0,3)  
    glUseProgram(0)  
    glutSwapBuffers()
```



# PyOpenGL : shaders

## Programme de tests

```
if __name__ == '__main__':  
    glutInit(sys.argv)  
    glutInitWindowSize(640, 480)  
    glutCreateWindow("Modern pyOpenGL")  
    init()  
    glutDisplayFunc(display)  
    glutMainLoop()
```

# PyOpenGL : 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()
```

# PyOpenGL : Graphe de scène

## Noeud de Translation

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

# PyOpenGL : Graphe de scène

## Noeud 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()
```

# PyOpenGL : 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(\n
            'angle:{},\n
            axe:{} {} {}')>".format(
                self.angle,
                self.axe[0],
                self.axe[1],
                self.axe[2]
            )
```

# PyOpenGL : 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
```

# PyOpenGL : 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()
```

# PyOpenGL : 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
```



# PyOpenGL : 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()
```

# PyOpenGL : Graphe de scène

## Primitives: Le Point

```
class Point(Leaf) :  
    def __init__(self):  
        Leaf.__init__(self)  
    def draw():  
        glBegin(GL_POINTS)  
        glVertex(0,0,0)  
        glEnd()
```

# PyOpenGL : Graphe de scène

## Primitives: Le Triangle

```
class Triangle(Leaf) :  
    def __init__(self):  
        Leaf.__init__(self)  
    def __repr__(self):  
        return "<Triangle()>"  
    def draw(self):  
        glBegin(GL_TRIANGLES)  
        glVertex(0,0.8,0)  
        glVertex(0.2,-0.2,0)  
        glVertex(-0.2,-0.2,0)  
        glEnd()
```

# PyOpenGL : Graphe de scène

## Primitives: Le Quadrilatere

```
class Quadrilatere(Leaf) :  
    def __init__(self):  
        Leaf.__init__(self)  
    def __repr__(self):  
        return "<Quadrilatere()>"  
    def draw(self):  
        glBegin(GL_QUADS)  
        glVertex( 0.5, 0.5)  
        glVertex( 0.5,-0.5)  
        glVertex(-0.5,-0.5)  
        glVertex(-0.5, 0.5)  
        glEnd()
```

# PyOpenGL : Graphe de scène

## Primitives: 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)
```

# PyOpenGL : Graphe de scène

## Primitives: 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)
```

# PyOpenGL : Graphe de scène

## Primitives: 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
```

# PyOpenGL : Graphe de scène

## Primitives: 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 bibliographique

## Livres

- **Graham Sellers et. al.**  
OpenGL Superbible: Comprehensive Tutorial and Reference  
(7th Edition),
- **Samuel R. Buss:**  
“3D Computer Graphics :  
A mathematical introduction with OpenGL”  
Collection Cambridge University Press (2003)

# Références Internet

## Adresses "au Net"

- [www.opengl.org](http://www.opengl.org)
- [www.opengl.org/sdk/docs](http://www.opengl.org/sdk/docs)
- <http://www.openglsuperbible.com>
- [www.glprogramming.com/red](http://www.glprogramming.com/red)
- [www.opengl-tutorial.org/fr/beginners-tutorials](http://www.opengl-tutorial.org/fr/beginners-tutorials)
- [www.codeproject.com/Articles/771225/Learning-Modern-OpenGL](http://www.codeproject.com/Articles/771225/Learning-Modern-OpenGL)
- <http://www.songho.ca/opengl>

# Références Internet

## Adresses “au Net”

- <http://raphaello.univ-fcomte.fr/IG>
- [http://igm.univ-mlv.fr/~vnozick/teaching/slides/ensg/01%20Pipeline\\_graphique.pdf](http://igm.univ-mlv.fr/~vnozick/teaching/slides/ensg/01%20Pipeline_graphique.pdf)
- [www.f-legrand.fr/scidoc/docimg/graphie/geometrie/affine/affine.html](http://www.f-legrand.fr/scidoc/docimg/graphie/geometrie/affine/affine.html)
- <http://www.nehe.gamedev.net>
- <http://www.paulbourke.net>
- <http://wiki.enib.fr/crd> : Ressources documentaires de l'ENIB