Maillage 2D 3D

Dr Wissal DRIRA

wissaldrira@yahoo.com

Plan du cours

I. Introduction

II. La bibliothèque graphique GLUT

III. Gestion des évènements du clavier et de la souris

IV. Visualisation et caméra

V. Texture

VI. Eclairage et matériaux

Introduction

- Pour tout rendu infographique et surtout pour tout rendu 3D, il est indispensable de savoir placer la position du regard et sa direction. Car même si nous construisons un monde en 3 dimensions, la restitution sur un écran est une projection de ce monde 3D en 2 dimensions.
- Il existe quatre opérations pour convertir les coordonnées d'un objet en 3D en coordonnées 2D pour l'écran, qui sont
  - **l'affichage**
  - **la projection**
  - **la visualisation**
  - **la modélisation**
- L'ensemble de ces transformations de la visualisation peut être comparé à un appareil photo ou à une caméra, vous décidez de placer votre caméra à une position ***(transformation de visualisation),*** vous construisez votre décor et le déplacez ***(transformation de modélisation),*** et vous ajuster le zoom pour définir quel volume vous prenez en photo ***(transformation de projection)***.

Introduction

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

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

Introduction

- Les transformations ***de visualisation*** et de ***modélisation*** n'en forment qu'une pour OpenGL (transformation modelview). Cette transformation fait partie de l'environnement OpenGL.
- *La* ***transformation de projection*** existe en tant que telle dans OpenGL, et fait elle aussi partie de l'environnement OpenGL.
- Chacune de ces deux transformations peut être modifiée indépendamment de l'autre ce qui permet d'obtenir une indépendance des scènes modélisées vis à vis des caractéristiques de la "caméra" de visualisation.
- La transformation ***d'affichage*** est elle aussi paramétrable en OpenGL

Choix de la transformation OpenGL de travail

void glMatrixMode(GLenum mode);

- ***mode*** : désigne la matrice que l'on souhaite activer. Il peut prendre comme valeur *GL_MODELVIEW, GL_PROJECTION, ou GL_TEXTURE*.
- Pour réaliser un affichage, *glMatrixMode* est généralement appelé successivement une fois sur chacun des deux paramètres de manière à établir les matrices *MODELVIEW* et *PROJECTION*. Ces appels sont habituellement réalisés au sein de la fonction *reshape*.
- Pour effectuer des transformations sur les objets de la scène (dans la fonction *Display*), il faut modifier la matrice de transformation-visualisation (*GL_MODELVIEW*). Par défaut, la matrice de modélisation-visualisation est la matrice active, ce qui explique pourquoi nous n'avons pas eu besoin d'utiliser *glMatrixMode()* avant.

Transformations géométriques

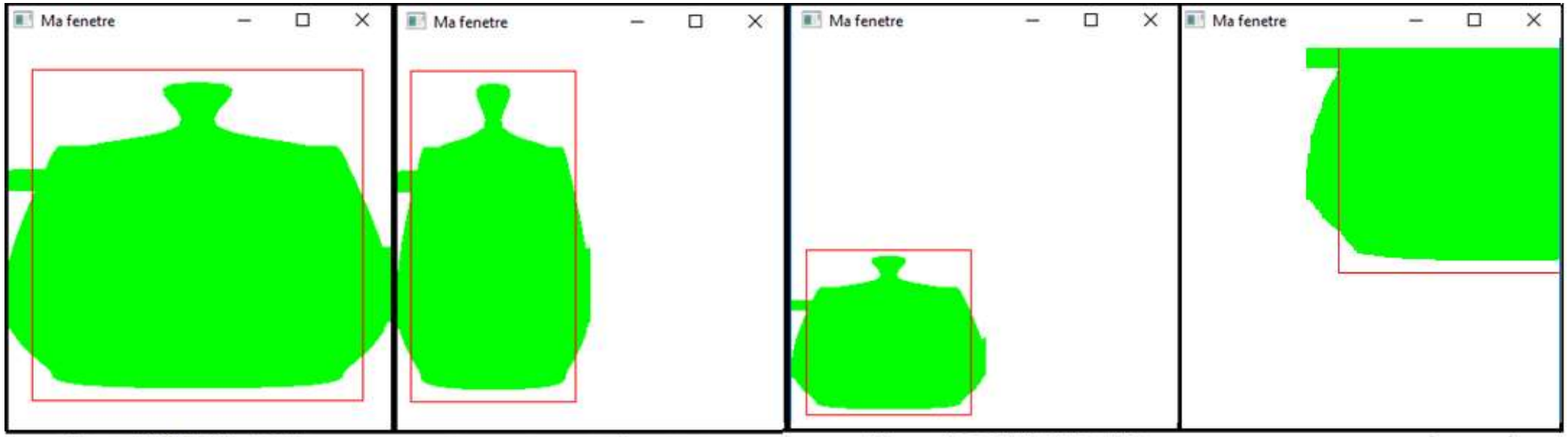
- **void glLoadIdentity(void):** Affecte la transformation courante avec la transformation identité.
- **void glTranslate{f d}(TYPE x,TYPE y,TYPE z) :** Compose la transformation courante par la translation de vecteur (x,y,z). Très utilisé en modélisation.
- **void glRotate{f d} (TYPE α,TYPE dx,TYPE dy,TYPE dz)** : Compose la transformation courante par la rotation d'angle **α** degrés autour de l'axe (dx,dy,dz) passant par l'origine. Très utilisé en modélisation.
- **void glScale{f d} (TYPE a,TYPE b,TYPE c)** : opère un changement d'échelle sur 3 axes: Les coordonnées en x sont multipliées par a, en y par b et en z par c (a, b et c doivent être différent de 0) . Très utilisé en modélisation.

Transformations d'Affichage

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height) ;

glViewport permet de définir la fenêtre avec:

- *x, y* : coordonnées du le coin inférieur gauche du rectangle de la fenêtre de visualisation, en pixels. La valeur initiale est (0,0).
- *width, height* :la largeur et la hauteur de la fenêtre. Lorsqu'un contexte GL est d'abord attaché à une fenêtre, largeur et hauteur sont définies selon les dimensions de cette fenêtre.



Example : *glViewport*


```
#include <GL/glut.h>
#include <GL/glu.h>
void Display(void) {
glClear(GL_COLOR_BUFFER_BIT);
glColor3d(0,1,0);
glutSolidTeapot(1);
glColor3d(1,0,0);
glutWireCube(1.7);
glFlush(); }
```

```
int main(int argc, char *argv[])  {
glutInit(&argc, argv);
glutCreateWindow("Ma fenetre");
glutDisplayFunc(Display);
glutReshapeFunc(Resize);
glClearColor(1,1,1,0);
glutMainLoop();
return 0; }
```

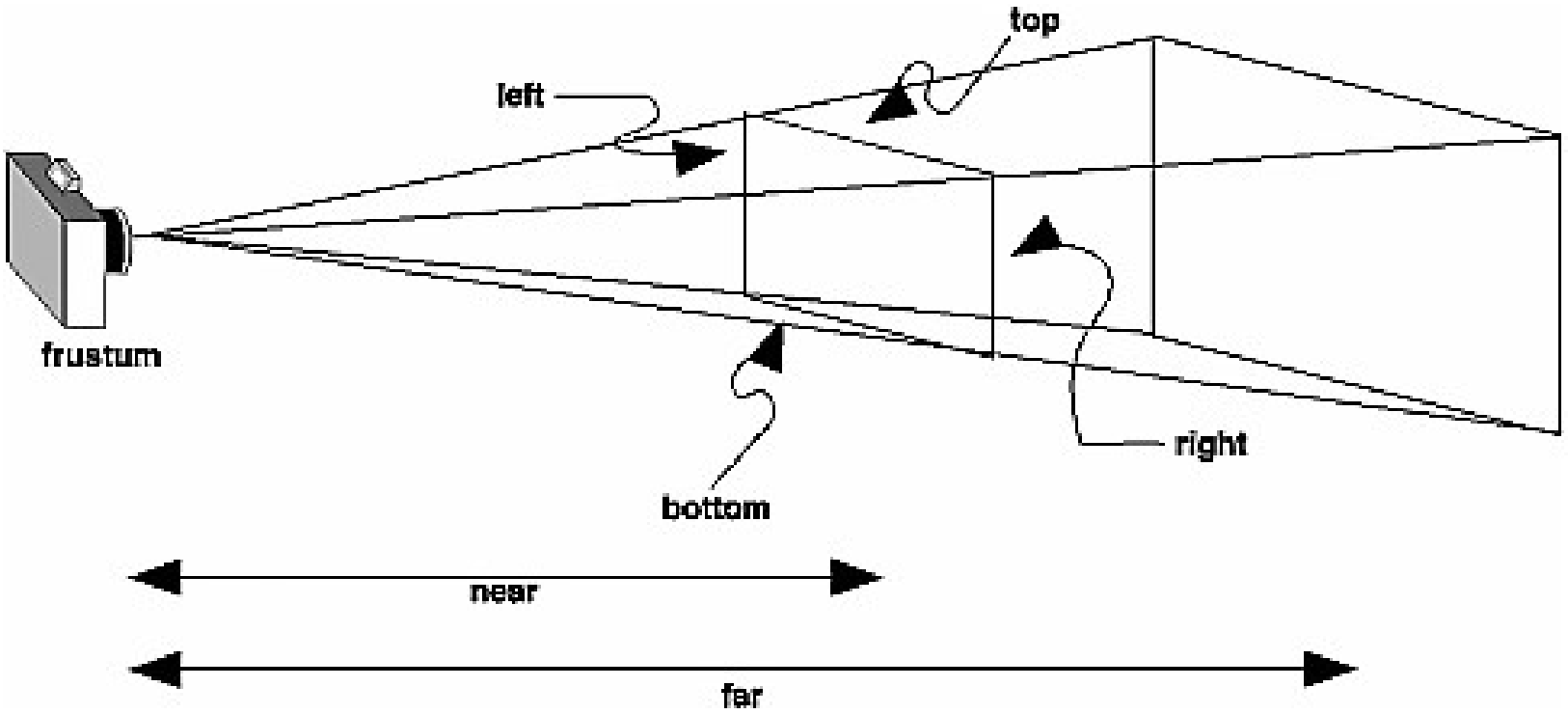
```
void Resize(int Width, int Height)      //Redimensionner correctement la scène
                                        //lorsqu'on agrandit la fenêtre d'éxécution
 { if (Height==0) Height=1;  // Empêche une division par zéro si la fenêtre est trop petite

glViewport(0, 0, Width, Height); //Réinitialiser la fenêtre d'affichage actuelle et la transformation de la perspective

// Essayer aussi
// glViewport(0, 0, Width/2, Height/2);
// glViewport(0, 0, Width/2, Height);
// glViewport(100, 100, Width, Height);
}
```

Transformation spécifique à la projection(1)

W DRIRA



void glFrustum (GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

Transformation spécifique à la projection(1)

W DRIRA

void glFrustum (GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

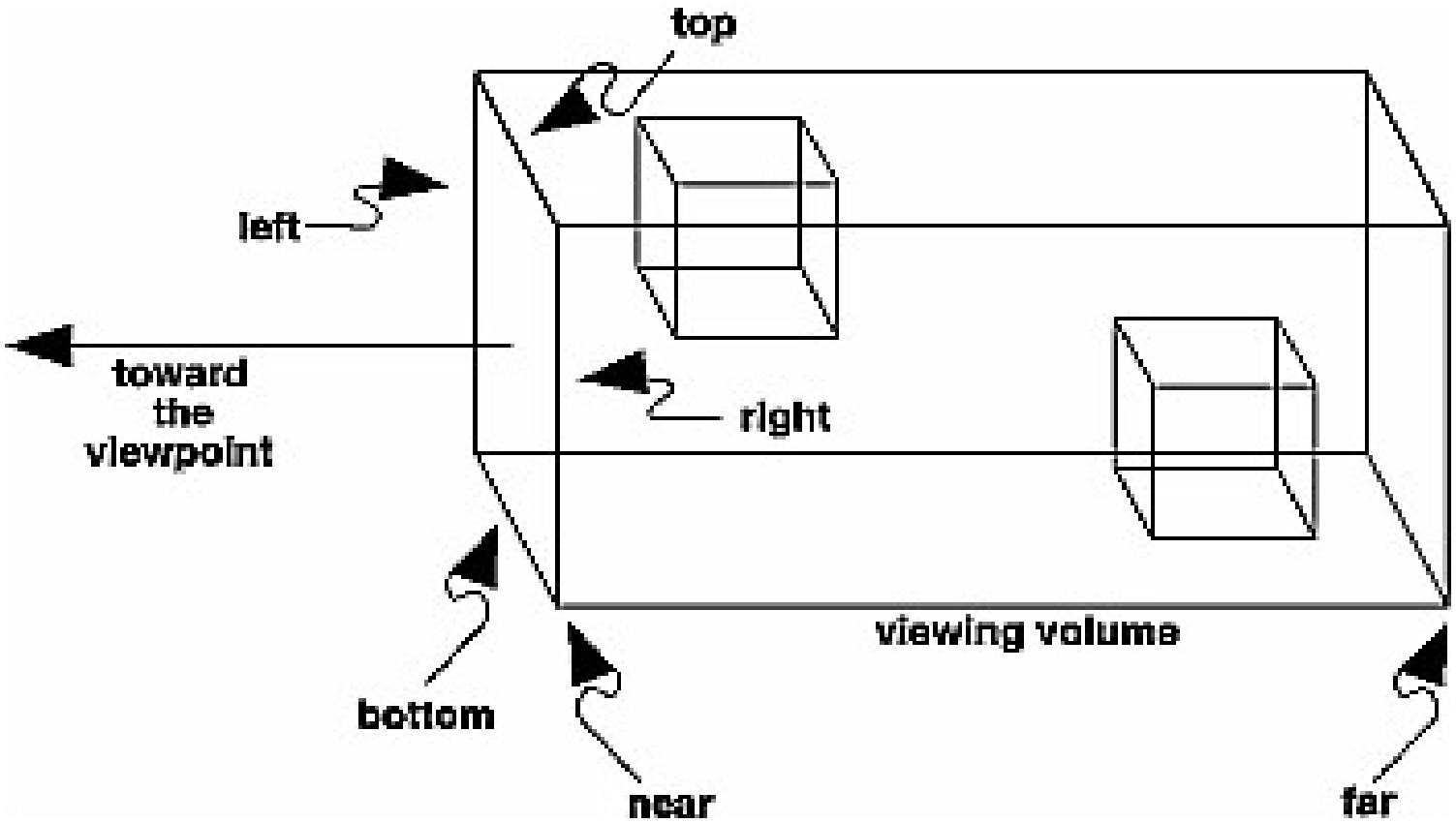
- Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O (position de la caméra virtuelle de visualisation), orientée selon l'axe -z (orientation de la caméra virtuelle de visualisation)
- *left / right :* Les coordonnée à droite/gauche du plan vertical du clipping
- *bottom / Top:* Les coordonnée haut/bas du plan horizontal du clipping
- *near / far:* les distances entre l'origine et les plans de clipping en z. Etant des distances, *near* et *far* doivent avoir une valeur positive. Ces valeurs doivent aussi respecter *near* < *far* . Ces valeurs étant comptées selon l'axe -z, seuls les objets placés en z négatif (entre - *far* et - *near*) peuvent être visibles.

Example : *glFrustum*


```
#include <GL/glut.h>
#include <GL/glu.h>
void Display(void) {   glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();
        glTranslated(0,0,-2); // Tester z= -1 et z= -1.25
        glColor3d(1,0,0);
        glutWireCube(1.5);
                glPopMatrix();
                glPushMatrix();
                glTranslated(0,0,-5);
                glColor3d(0,1,0);
                glutSolidTeapot(1);
                glPopMatrix();
glFlush();}
void Resize(int Width, int Height){
  if (Height==0) Height=1;
  glViewport(0, 0, Width, Height);
   glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-2,2,-1,1,2,6); }
```

```
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutCreateWindow("Ma fenetre");
glutDisplayFunc(Display);
glutReshapeFunc(Resize);
glClearColor(1,1,1,0);
glutMainLoop();
return 0;
}
```

Transformation spécifique à la projection(2)



void glOrtho (GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

Transformation spécifique à la projection(2)

void glOrtho(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

- Compose la transformation courante par la transformation de projection orthographique selon l'axe -z et définie par le volume de visualisation parallélépipèdique (*left,right,bottom,top,near,far*).

- *Near* et *far* peuvent être positifs ou négatifs mais doivent respecter *near <far*

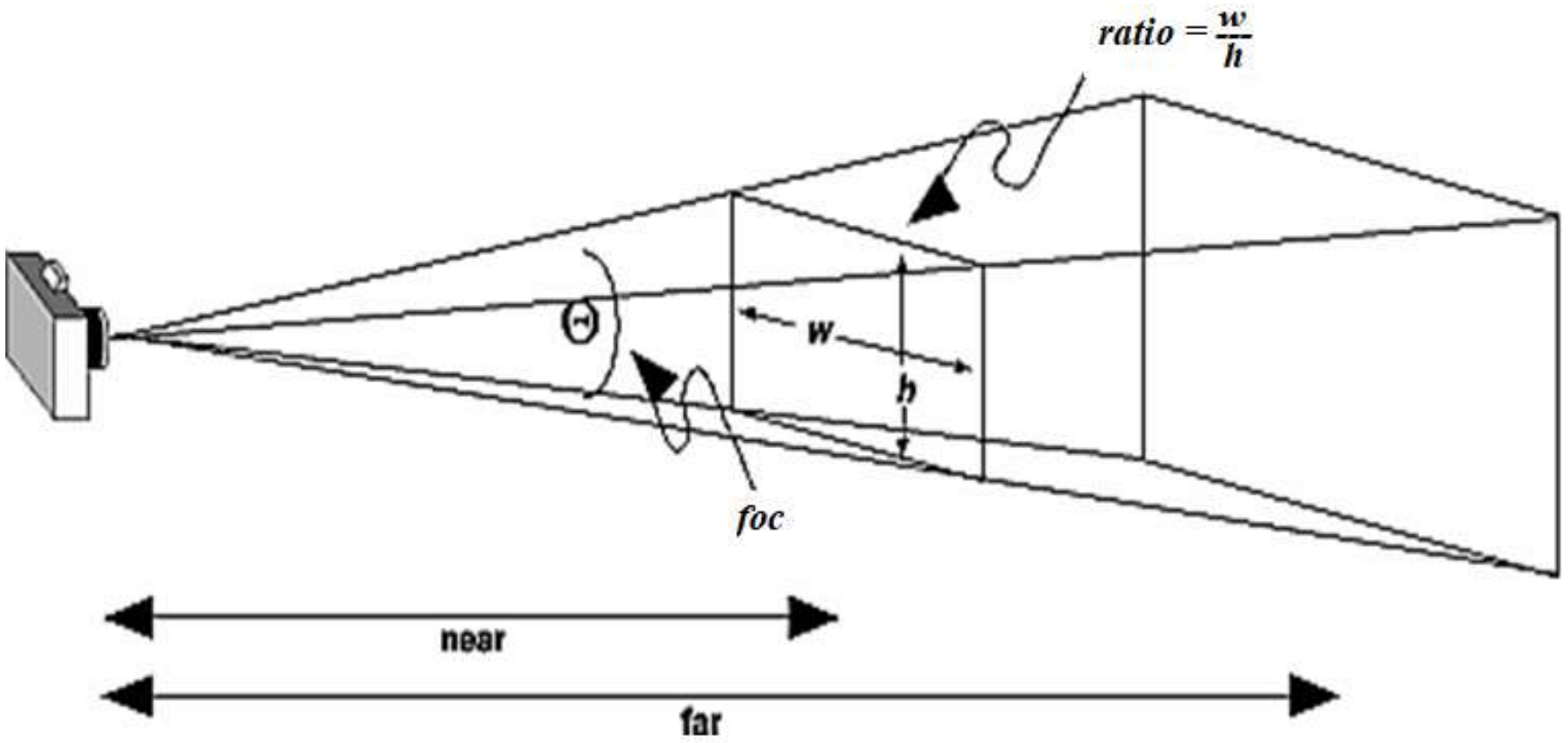
Example : *glOrtho*


```
#include <GL/glut.h>
#include <GL/glu.h>
void Display(void) {   glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();
        glTranslated(0,0,-2);
        glColor3d(1,0,0);
        glutWireCube(1.5);
                glPopMatrix();
                glPushMatrix();
                glTranslated(0,0,-5);
                glColor3d(0,1,0);
                glutSolidTeapot(1);
                glPopMatrix();
glFlush();}
void Resize(int Width, int Height){
  if (Height==0) Height=1;
  glViewport(0, 0, Width, Height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-2,2,-1,1,2,6);}
// Tester aussi  glOrtho(-5,5,-10,10,1,6); glOrtho(-10,10,-10,10,1,6);
// Tester aussi glOrtho(-5,5,-5,5,1,6); en ajoutant pour le cube glRotated(45,1,1,0);
```

```
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutCreateWindow("Ma fenetre");
glutDisplayFunc(Display);
glutReshapeFunc(Resize);
glClearColor(1,1,1,0);
glutMainLoop();
return 0;
}
```

Transformation spécifique à la projection(3)

W DRIRA



void gluPerspective(GLdouble foc, GLdouble ratio,
GLdouble near, GLdouble far);

Transformation spécifique à la projection(3)

W DRIRA

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

- Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O, orientée selon l'axe -z, possédant l'angle *foc* comme angle d'ouverture verticale, *ratio* (rapport largeur/hauteur) avec (*foc* ratio)* comme angle d'ouverture horizontale, et les plans de clipping proche et éloignés *near* et *far* en z = -*near* et z = -*far*. Etant des distances, *near* et *far* doivent avoir une valeur positive et respecter *near* < *far*
- *Exemple : gluPerspective(40,1.5,50,100)* ➔ configure une caméra de visualisation en perspective placée à l'origine et orientée selon l'axe -z
  - avec un angle d'ouverture verticale de 40°,
  - un angle d'ouverture horizontale de 40°*1.5 = 60°,
  - un plan de clipping qui élimine tous les objets ou morceaux d'objet situés en z > -50 et en z < -100.

Ces valeurs sont considérées dans le repère courant au moment de l'appel de fonction.

Example : *gluPerspective*


```
#include <GL/glut.h>
#include <GL/glu.h>
void Display(void) {   glClear(GL_COLOR_BUFFER_BIT);
        glPushMatrix();
        glTranslated(0,0,-2);
        glColor3d(1,0,0);
        glutWireCube(1.5);
                glPopMatrix();
                glPushMatrix();
                glTranslated(0,0,-5);
                glColor3d(0,1,0);
                glutSolidTeapot(1);
                glPopMatrix();
glFlush();}
void Resize(int Width, int Height){
  if (Height==0) Height=1;
  glViewport(0, 0, Width, Height);
   glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(90,Width/Height,1,10); }
  // Tester aussi // gluPerspective(40,Width/Height,1,10); gluPerspective(90,0.5,1,10);
  //gluPerspective(90,Width/Height,2.75,10);   gluPerspective(90,Width/Height,3,10);
```

```
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutCreateWindow("Ma fenetre");
glutDisplayFunc(Display);
glutReshapeFunc(Resize);
glClearColor(1,1,1,0);
glutMainLoop();
return 0;
}
```

Transformation spécifique à la visualisation (view)

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

- Compose la transformation courante (généralement MODELVIEW) par la transformation donnant un point de vue depuis *(ex,ey,ez)* avec une direction de visualisation passant par *(cx,cy,cz)*.
- *(upx,upy,upz)* indique quelle est la direction du repère courant (fréquemment le repère global) qui devient la direction y (0.0, 1.0, 0.0) dans le repère écran (‘direction du haut’).
- gluLookAt est une fonction de la bibliothèque GLU.

Transformation spécifique à la visualisation (view)

- Exemple:

gluLookAt(10.0,15.0,10.0,3.0,5.0,-2.0,0.0,1.0,0.0)

place la caméra en position (10.0,15.0,10.0), l'oriente pour qu'elle vise le point (3.0,5.0,-2.0) et visualisera la direction (0.0,1.0,0.0) de telle manière qu'elle apparaisse verticale dans la fenêtre de visualisation. Ces valeurs sont considérées dans le repère global.

- Dans la pratique, gluLookAt place et oriente la scène devant la caméra de telle manière que la caméra semble placée et orientée selon les paramètres d'entête.
  ➔ gluLookAt doit être exécuté en mode MODELVIEW . (donc après la définition de la caméra en mode GL_PROJECTION) et doit être placé avant la création de la scène

Example : *gluLookAt*


```
#include <GL/glut.h>
#include <GL/glu.h>
void Display(void) {
glClear(GL_COLOR_BUFFER_BIT);
glColor3d(0,1,0);
glutSolidTeapot(1);
glColor3d(1,0,0);
glutWireCube(1.7);
glFlush();}
void Resize(int Width, int Height){
  if (Height==0) Height=1;
  glViewport(0, 0, Width, Height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(5,5,10,0,0,0,0,1,0);
}
```

```
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutCreateWindow("Ma fenetre");
glutDisplayFunc(Display);
glutReshapeFunc(Resize);
glClearColor(1,1,1,0);
glutMainLoop();
return 0;
}
```

Example : *gluLookAt*