OpenGL 3.0

Chapitre 1

Espaces du modèle, Espace du monde
Modes matriciels, Caméra, Fenêtre de vue

exemple: gl-test01.cpp

C.Turrier
15 décembre 2020

Sommaire

1) Espace du modèle (model space)

On appelle **modèle**, un objet 3D particulier, un cube par exemple. Chaque objet 3D possède un système de coordonnées qui lui est attaché appelé espace du modèle (**model space**).

Les coordonnées des sommets d'un objet s décrites dans l'espace du modèle (exemple glutWireCube(100.0);) ou directement décrites dans l'espace de la scène 3D (espace du monde).
Les valeurs des coordonnées et des variables de grandeurs (diamètre, coté...) utilisées sont exprimées en unités relatives (les unes par rapport aux autres).

Exemple 1

```
glutWireCube(1.0);
```

Lorsqu'on utilise cette instruction, les coordonnées des sommets sont automatiquement décrites dans l'espace du modèle et le centre du cube est positionné par défaut en (0,0,0) dans l'espace du monde

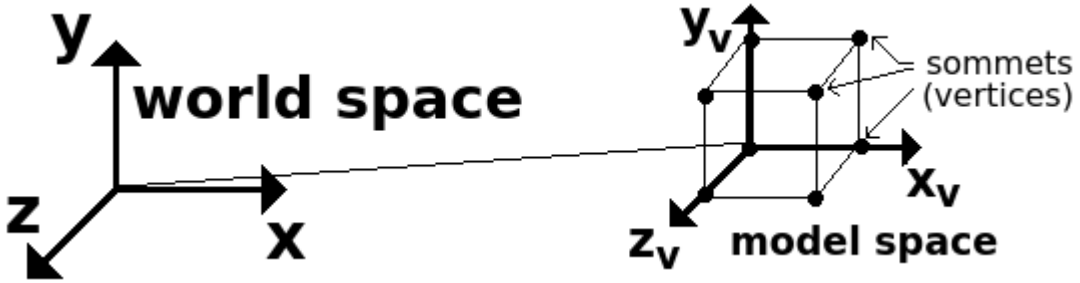
Exemple 2

Lorsqu'on utilise les instructions suivantes, les coordonnées des sommets sont directement décrits dans l'espace de la scène 3D (espace du monde).

```
glBegin(GL_LINES);
glBegin(GL_TRIANGLES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(10.0, 10.0, 0.0);
    glVertex3f(0.0, 10.0, 0.0);
glEnd();
```

2) Espace du monde (world space)

On appelle espace du monde (**world space**) l'espace dans lequel se trouvent les objets 3D constituant la scène 3D. À un instant donné, la position d'un objet 3D dans l'espace du monde dépend de sa position initiale dans cet espace et des tranformations qui lui ont été ensuite appliquées (translation glTranslatef ou rotation glRotatef) .



3) Modes matriciels de OpenGL

Lorsqu'on utilise OpenGL, on peut à tout moment se placer dans le mode matriciel de notre choix. On distingue les deux modes matriciels suivants : le mode matriciel **glMatrixMode(GL_MODELVIEW)** et le mode matriciel **glMatrixMode(GL_PROJECTION)** .

3.1) Mode matriciel glMatrixMode(GL_MODELVIEW)

Le mode matriciel glMatrixMode(GL_MODELVIEW) signifie que les transformations qui vont suivre vont être des **t**ranslations (**glTranslatef**) et/ou des rotations (**glRotatef**).

Ces transformations vont s'appliquer au objets 3D concernés , c'est à dire aux objets décrits dans la fonction utilisateur **glutDisplayFunc**(affiche), par rapport aux axes de l'espace du monde (cf chapitre2)

3.2) Mode matriciel glMatrixMode(GL_PROJECTION)

Le mode matriciel **glMatrixMode(GL_PROJECTION)** signifie que l'instruction qui va suivre va être **glOrtho**, **glFrustum** ou **gluPerspective** et va déterminer la façon dont les objets 3D de la scène vont être projetés à l'écran.

L'instruction **glOrtho()** conduit à ce que les plans proches et éloignés des objets 3D, si ils ont la même taille, apparaitront également à l'écran avec la même taille

L'instruction **glFrustum()** conduit à ce que les plans éloignés des objets 3D, même si ils ont la même taille, que les plans proches, apparaitront à l'écran avec une taille plus petite que les plans proches. Ceci crée un effet visuel de profondeur (pespective) qui augmente le réalisme de la scène 3D.

L'instruction **gluPerspective()** conduit, avec des paramètres différents, à un effet de perspective pratiquement identique à celui obtenu avec **glFrustum()**.

mode projection glOrtho

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.5, 1.5, -1.5, 1.5, -1.5, 1.5);//glOrtho(left, right, bottom, top, near, far)
```

Seuls les objets 3D qui se trouvent dans l'espace cubique (clipping planes) définit par **glOrtho(left, right, bottom, top, near, far)**, dans l'espace du monde , seront ici pris en compte par OpenGL dans la projection à l'écran. On peut remarquer que les signes et **near** et **far** doivent être inversés, dans l'instruction **glOrtho**, par rapport à leur valeur réelle, dans l'espace du monde.
En mode de projection glOrtho, la notion de caméra n'existe pas ; la projection de la scène 3D à l'écran s'effectue directement, de façon orthogonale (sans déformation) selon l'axe z.

mode projection glFrustum

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.5, 1.5, -1.5, 1.5, 1.5, 150.0);  //glFrustum(left, right, bottom, top, near, far)
gluLookAt(0.0, 0.0, -3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//caméra

gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
```

Dans glFrustum, les valeurs de near et de far doivent être obligatoirement positives. Cela conduit à ce que le signe de **near** doit être inversé, dans l'instruction **glFrustum**, par rapport à sa valeur réelle, dans l'espace du monde. De plus, la valeur réelle de near (donc -near) doit être supérieure eyeZ (ici, -1.5>-3.0) pour qu'une partie de la scène se sorte pas du champs de la caméra.

4) Caméra

Dans le mode projection **glFrustum**, la notion de caméra existe.

Le champs de la caméra est un cône parallèlépipédique définit par les instructions :

```
glFrustum(left, right, bottom, top, near, far)
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
```

Seule la partie de la scène 3D qui entre dans le champs de la caméra est projetée à l'écran

par exemple, l'instruction gluLookAt(0.0, 0.0, -3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); fixe :

- la position de la caméra dans l'espace du monde (0.0, 0.0, -3.0)
- le point de l'espace du monde pointé par la caméra (0.0, 0.0, 0.0)
- l'axe vertical (y) de l'espace du monde (0.0, 1.0, 0.0)

5) Fenêtre de vue (viewport)

La fenêtre de vue (viewport) est le rectangle 2D, à l'écran, dans lequel sont projetés les éléments de la scène 3D, après l'opération glutSwapBuffers (bascule du contenu du buffer arrière ou back buffer situé en mémoire dans le buffer avant ou front buffer situé à l'écran)
En général, les dimensions de la fenêtre de vue sont fixées de façon à correspondre aux dimensions de la fenêtre OpenGl du programme.

6) Exemple

L’exemple suivant affiche, en mode ortho, frustum ou perspective :
- un cube de coté 1.4 centré à l’origine (0,0,0) de l’espace du monde ;
- un triangle de coté 1 placé dans le plan z=0.

```
/*****************************************************************
* gl-test01.cpp  Test de glOrtho, glFrustum et gluPerspective
* avec un cube et un triangle filaires (wireframe)
* compilation :
* g++ gl-test01.cpp -o test -lglut -lGLU -lGL
* gcc gl-test01.cpp -o test -lglut -lGLU -lGL
* exécution :
* ./test o  (pour projection ortho)
* ./test f  (pour projection frustrum)
* ./test p  (pour projection perspective)
* ****************************************************************/
#include <GL/freeglut.h>
#include <string.h>
int ortho = 1;
int frustum = 0;
int perspective = 0;

static void affiche(void) {
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();

if (ortho==1) {}
//si glFrustrum on place la caméra en z=-3 pour voir le cube centré à l'origine en entier
if (frustum==1) {gluLookAt(0.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);}
//if gluPerspective on translate le cube et le triangle de z=-2 pour les voir en entier
if (perspective==1){ glTranslatef(0.0, 0.0, -2.0);}

glLineWidth(2.0);
//cube couleur blanc, rouge ou vert
```

```
if (ortho==1){glColor3f(1.0f, 1.0f, 1.0f);}
if (frustum==1){glColor3f(1.0f, 0.0f, 0.0f);}
if (perspective==1){glColor3f(0.0f, 1.0f, 0.0f);}
glutWireCube(1.4);

//triangle blanc
GLfloat x0=0;GLfloat y0=0; GLfloat z0=0;
GLfloat x1=x0+1.0; GLfloat y1=y0+1.0;
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_LINES);
  glVertex3f(x0, y0, z0); glVertex3f(x1, y1, z0);
  glVertex3f(x1, y1, z0); glVertex3f(x1, y0, z0);
  glVertex3f(x1, y0, z0);glVertex3f(x0, y0, z0);
glEnd();
glFlush();
}

static void dimensionne(int w, int h) {
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (ortho==1) {glOrtho(-1.5, 1.5, -1.5, 1.5, -20.0, 1.5);}
if (perspective==1) { gluPerspective (90.0, 1.0, 1.5, 20.0);}
if (frustum==1) {glFrustum(-1.5, 1.5, -1.5, 1.5, 1.5, 20.0);}
glMatrixMode(GL_MODELVIEW);
}

int main( int argc, char *argv[] )
{
glutInit(&argc, argv);
if (strcmp(argv[1],"f")==0) { frustum=1;}
if (strcmp(argv[1],"p")==0) { perspective=1;}
if (strcmp(argv[1],"o")==0) { ortho=1;}

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(300, 300);
glutInitWindowPosition(100, 100);
```

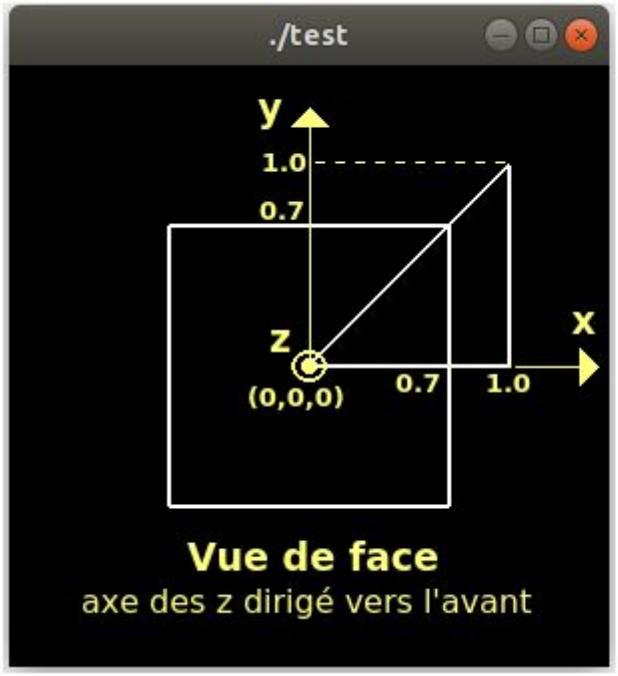
```
glutCreateWindow(argv[0]);
glClearColor(0.0, 0.0, 0.0, 0.0);
glShadeModel(GL_FLAT);
glutDisplayFunc(affiche);
glutReshapeFunc(dimensionne);
glutMainLoop();
return 1;
}
```

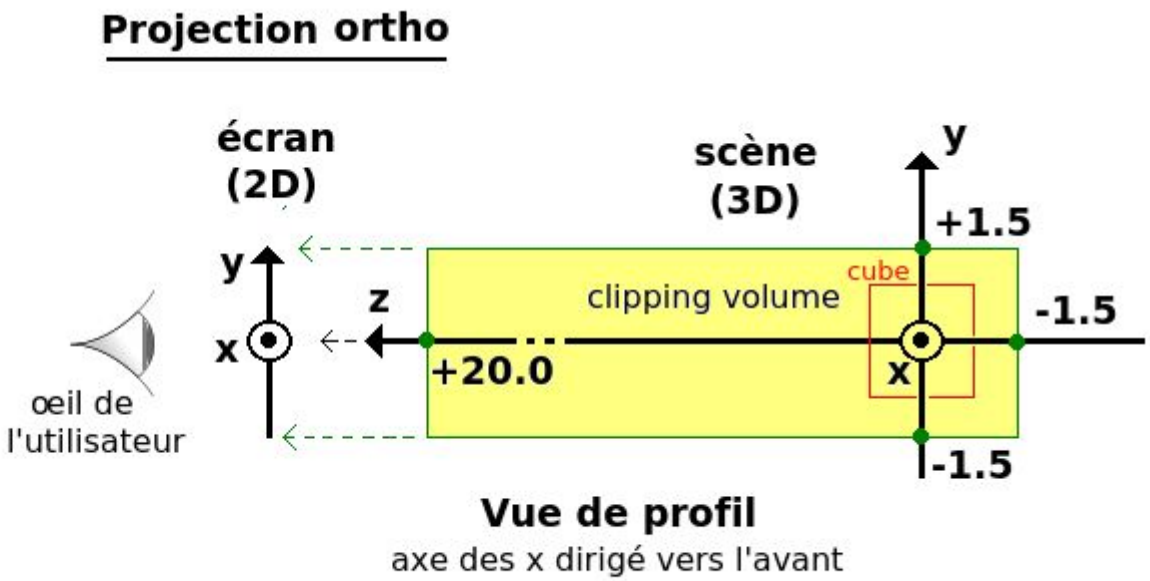
Le lancement du programme avec la commande :
./test o conduit à une projection de la scène 3D de type **glOrtho**
./test f conduit à une projection de la scène 3D de type **glFrustum**
./test p conduit à une projection de la scène 3D de type **gluPerspective**

./test o

Projection glOrtho()

```
glOrtho(left, right, bottom, top, near, far)
glOrtho(-1.5, 1.5, -1.5, 1.5, -20.0, 1.5);
```



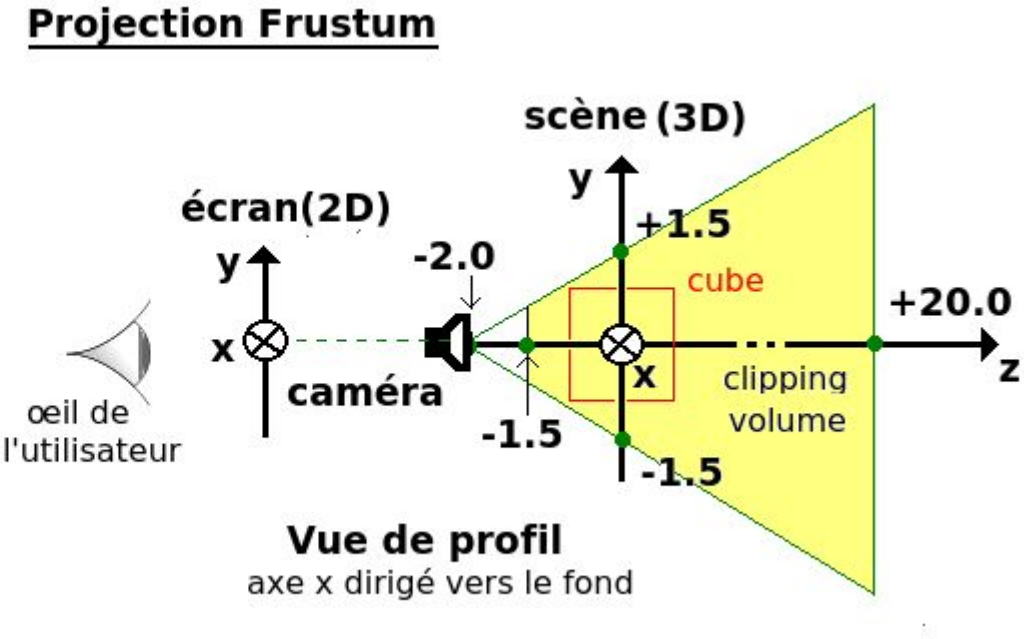


En projection glOrtho(), les signes de near=-20 et far= 1.5, écrits dans l'instruction, sont inversés par rapport à la réalité de la scène 3D (cf schéma de droite)

./test f

Projection glFrustum

```
glFrustum(left, right, bottom, top, near, far)
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
glFrustum(-1.5, 1.5, -1.5, 1.5, 1.5, 20.0)
gluLookAt(0.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

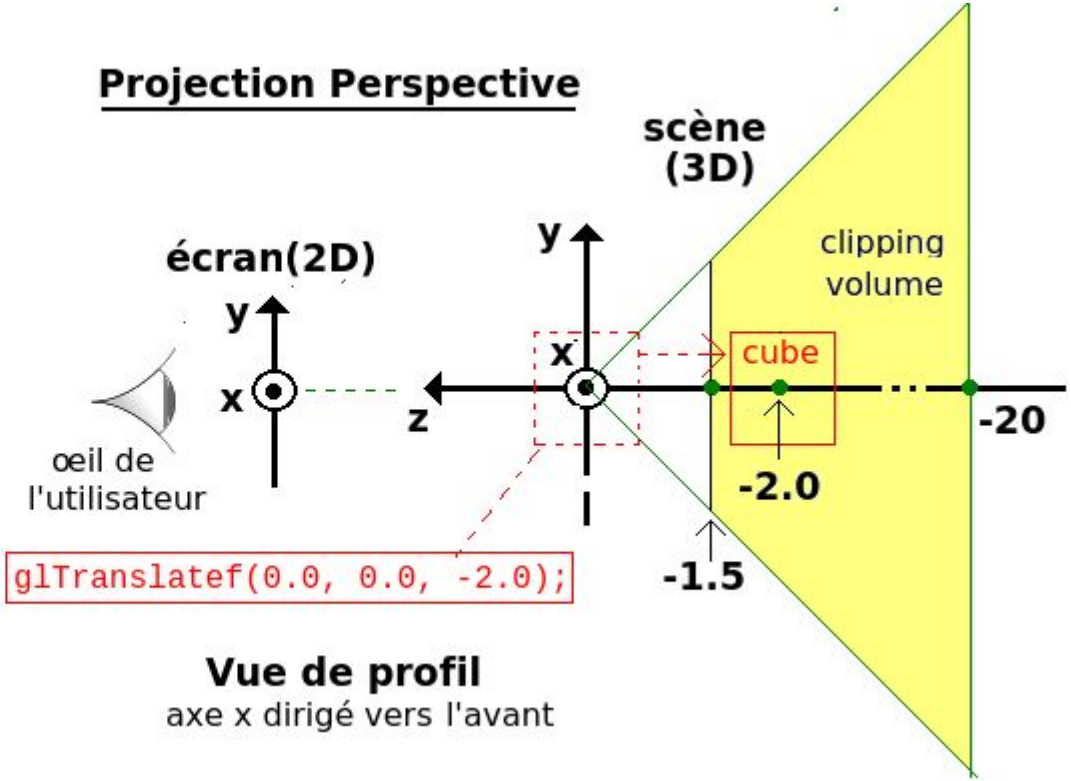


En projection glFrustum(), les valeurs de near et de far doivent être obligatoirement positives. Cela conduit à ce que le signe de **near** doit être inversé, dans l'instruction **glFrustum**, par rapport à sa valeur réelle, dans l'espace du monde. La caméra doit être placée à une distance eyeZ sur l'axe des Z telle que la valeur réelle de near (donc -near) soit supérieure à eyeZ (ici, -1.5>-2.0), ceci pour qu'une partie de la scène se sorte pas du champs de la caméra.

./test p

Projection gluPerspective()

```
gluPerspective(fovy, aspect, zNear, zFar);
gluPerspective (90.0, 1.0, 1.5, 20.0);
glTranslatef(0.0, 0.0, -2.0);
```



En projection gluPerspective(), lLe point de vue de la scène (équivalent de la caméra) est placé par défaut en (0,0,0) c'est pourquoi il faut translater le cube de -2.0 pour qu'il puisse être vu (clipping volume)

Commentaires

● #include <GL/freeglut.h>
pour pouvoir utiliser OpenGL

● #include <GL/freeglut.h>
pour pouvoir utiliser la fonction strcmp() de comparaison de deux chaines de caractères

● int ortho = 1;
● int frustum = 0;
● int perspective = 0;
variables globales associées au mode de projection choisi.

● void affiche(void)
Fonction utilisateur qui est appelée automatiquement (à travers la fonction glutDisplayFunc) lors du lancement du programme ou lors d'un appel de la fonction glutPostRedisplay();

● glClear(GL_COLOR_BUFFER_BIT);
La fonction glClear() met à zéro les buffers utilisés par OpenGL
Le paramètre GL_COLOR_BUFFER_BIT indique que les buffers courants sont activés pour le dessin des couleurs ;

● glLoadIdentity();
La fonction glLoadIdentity () est appelée, avant d'effectuer une transformation (glTranslate, glRotate…), lorsqu'on souhaite réinitialiser la matrice de transformation à son état d'origine (matrice identité).
On fait cet appel lorsqu'on souhaite qu'une transformation soit effectuée à l'aide de la matrice de transformation dans son état d'origine (et non pas l'aide de la matrice de transformation dans son état courant) .

- if (ortho==1) {} //on ne fait rien
- //si glFrustrum on place la caméra en z=-3 pour voir le cube centré à l'origine en entier
- if (frustum==1) {gluLookAt(0.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);}
- //if gluPerspective on translate le cube et le triangle de z=-2 pour les voir en entier
- if (perspective==1){ glTranslatef(0.0, 0.0, -2.0);}

- glLineWidth(2.0);

on fixe l'épaisseur des lignes formant le triangle et le cube à la valeur 2

- //cube couleur blanc, rouge ou vert
- if (ortho==1){glColor3f(1.0f, 1.0f, 1.0f);}
- if (frustum==1){glColor3f(1.0f, 0.0f, 0.0f);}
- if (perspective==1){glColor3f(0.0f, 1.0f, 0.0f);}

On trace un cube de couleur blanche, rouge ou verte selon que le mode de projection choisi au lancement du programme est ortho, frustum ou perspective

- glutWireCube(1.4);

Dessin d'un cube rouge filaire de coté égal à 1.4, au centre (0,0,0) de l'espace du monde

- //triangle blanc
- GLfloat x0=0;GLfloat y0=0; GLfloat z0=0;
- GLfloat x1=x0+1.0; GLfloat y1=y0+1.0;
- glColor3f(1.0, 1.0, 1.0);
- glBegin(GL_LINES);
  - glVertex3f(x0, y0, z0); glVertex3f(x1, y1, z0);
  - glVertex3f(x1, y1, z0); glVertex3f(x1, y0, z0);
  - glVertex3f(x1, y0, z0);glVertex3f(x0, y0, z0);
- glEnd();

Dessin, dans le buffer de dessin, d'un triangle blanc filaire de coté égal à 1, entre les points (0,0,0), (1,1,0) et (1 0,0) de l'espace du monde

● glFlush();
Bascule du contenu du buffer de dessin à l'écran (mode simple buffer)

● void dimensionne(int w, int h)
fonction utilisateur qui est appelée automatiquement (à travers la fonction glutReshapeFunc) lors du lancement du programme ou lors d'un redimensionnement de la fenêtre du progamme.

● glViewport(0, 0, w, h);
La fonction glViewport(x, y, w, h) définie la fenêtre de vue (viewport) à l'écran.

- x, y : coin inférieur gauche du rectangle de la fenêtre de vue, en pixels (la valeur initiale est (0,0)) ;
- w,h : largeur et hauteur de la fenêtre de vue. Lorsqu'un contexte GL est d'abord attaché à une fenêtre, la largeur et la hauteur sont définies sur les dimensions de cette fenêtre.

● glMatrixMode(GL_PROJECTION);
Indique que les opérations matricielles suivantes vont s'appliquer à la pile de matrices de projection.
Pour information :
glMatrixMode spécifie à quelle type de matrice s'appliquent les opérations qui vont suivre
GL_MODELVIEW
Applique les opérations de matrice suivantes à la pile de matrices modelview.
GL_PROJECTION
Applique les opérations matricielles suivantes à la pile de matrices de projection.
GL_TEXTURE
Applique les opérations de matrice suivantes à la pile de matrices de texture.
GL_COLOR
Applique les opérations matricielles suivantes à la pile de matrices couleur.

● glLoadIdentity();
Réinitialise la matrice de transformation à son état d'origine (matrice identité)

● if (ortho==1) {glOrtho(-1.5, 1.5, -1.5, 1.5, -20.0, 1.5);}
● if (perspective==1) { gluPerspective (90.0, 1.0, 1.5, 20.0);}
● if (frustum==1) {glFrustum(-1.5, 1.5, -1.5, 1.5, 1.5, 20.0);}

On paramètre le mode de projection choisi

● glMatrixMode(GL_MODELVIEW);
Passage en mode matriciel modelview, pour la suite. Ceci signifie que les opérations matricielles suivantes vont s'appliquer à la pile de matrices de vue.

● int main(int argc, char *argv[])
Programme principal

Initialisations

● glutInit(&argc, argv);
Intialisation de OpenGL

● if (strcmp(argv[1],"f")==0) { frustum=1;}
● if (strcmp(argv[1],"p")==0) { perspective=1;}
● if (strcmp(argv[1],"o")==0) { ortho=1;}
Selon la variable passée en argument lors du lancement du programme, on initialise à 1 la variable correspondante

● glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
Initialisation du mode graphique RGB, avec buffer simple

● glutInitWindowSize(300, 300);
● glutInitWindowPosition(100, 100);
● glutCreateWindow(argv[0]);
● glClearColor(0.0, 0.0, 0.0, 0.0);
Création d'une fenêtre de programme de 300x300 pixels, de fond noir et dont le coin haut gauche est placé en (100,100) sur l'écran

● glShadeModel(GL_FLAT);
Initialisation d’un mode d’ombrage plat.

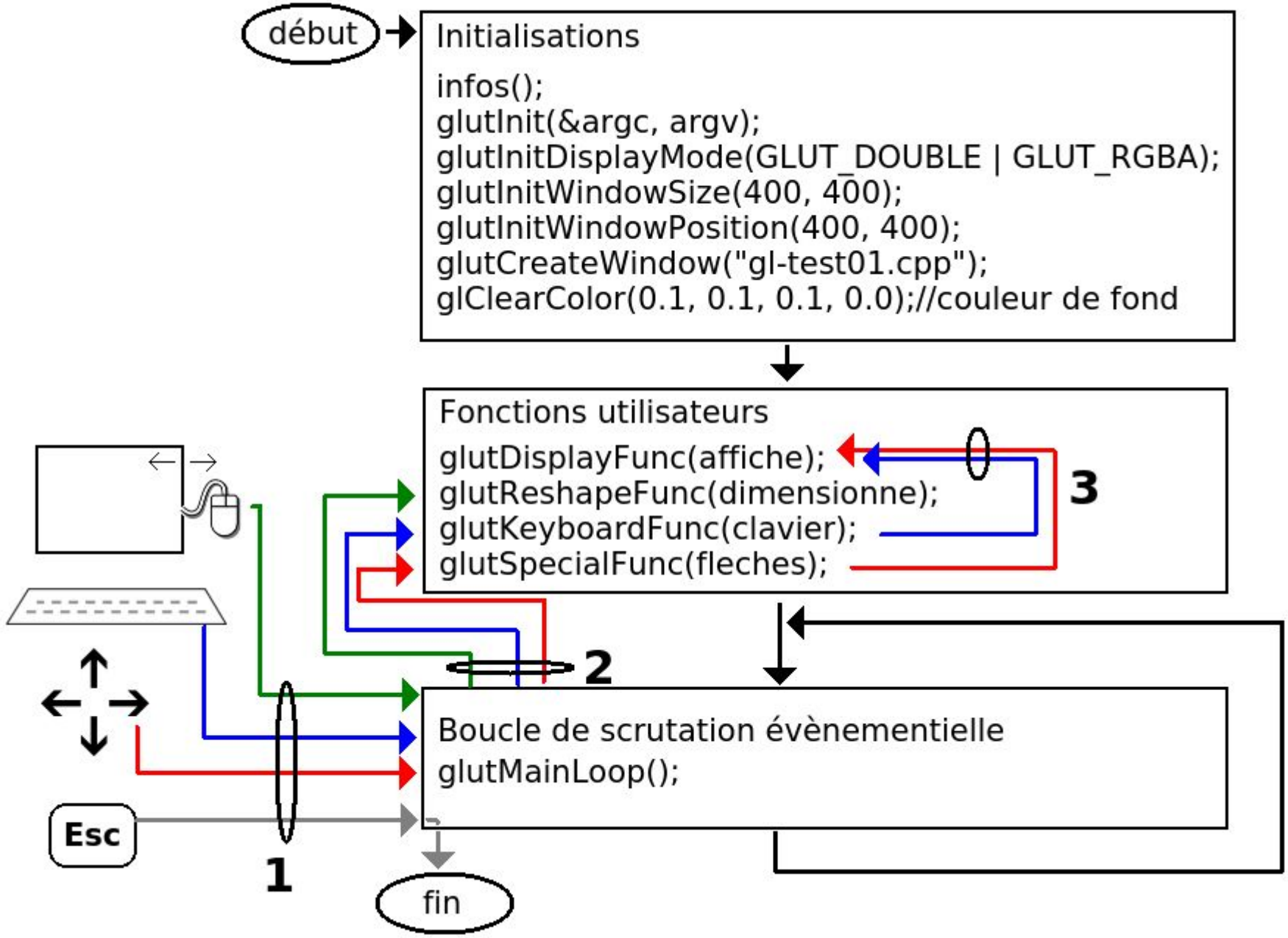
Fonctions utilisateurs

glutDisplayFunc(affiche);
glutReshapeFunc(dimensionne);

Boucle de scrutation évènementielle
glutMainLoop();

Fin du programme
return 1;
}

D’une façon plus générale, le squelette d’un programme OpenGL est illustré par le schéma suivant..


début
Initialisations
infos();
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
glutInitWindowSize(400, 400);
glutInitWindowPosition(400, 400);
glutCreateWindow("gl-test01.cpp");
glClearColor(0.1, 0.1, 0.1, 0.0);//couleur de fond
Fonctions utilisateurs
glutDisplayFunc(affiche);
glutReshapeFunc(dimensionne);
glutKeyboardFunc(clavier);
glutSpecialFunc(fleches);
3
2
Boucle de scrutation évènementielle
glutMainLoop();
Esc
1
fin