Introduction :

Glut, l'ensemble de fonctions pour OpenGL propose un mécanisme simple et efficace pour créer des fenêtres OpenGL sous l'environnement X−Window. Cependant, si vous souhaitez développer une application nécessitant une interface utilisateur digne de ce nom, il vous faudra vous tourner vers une autre API, sans doute plus complexe, mais plus adaptée à vos besoins.

Sous les environnements Unix, les créations et la gestion d'interface graphique (en anglais GUI pour Graphical User Interface) ont longtemps été l'apanage de Motif. Avec l'essor de Linux, deux autres bibliothèques ont réussi à percer et dominent à l'heure actuelle le marché. La première, nommée QT, constitue la brique de base de l'environnement KDE. Cette bibliothèque étant écrite en C++, nous ne l'aborderons pas et nous lui préfèrerons GTK+, créée pour servir de base à GIMP (GTK signifie d'ailleurs Gimp ToolKit).

Pour ce didacticiel, vous aurez besoin d'une bonne connaissance de la programmation GTK. Si ce n'est pas votre cas, ne désespérez pas. David Odin a publié un excellent livre sur le sujet, intitulé « Programmation Linux avec GTK+ » [10].

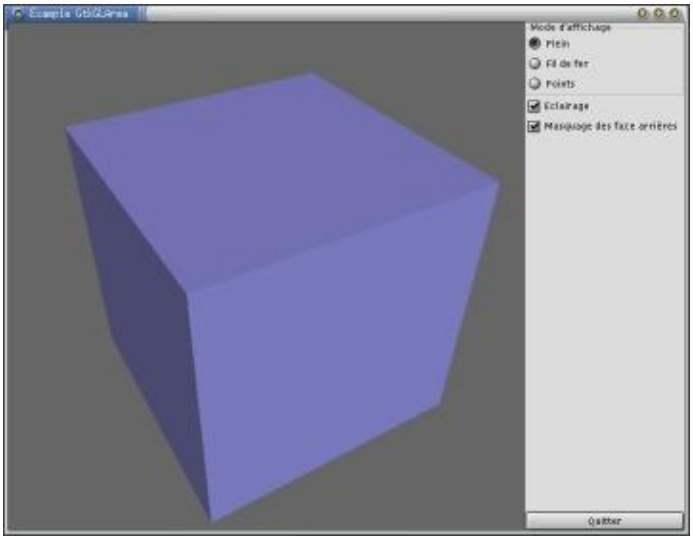

Figure 1 : Un programme GtkGLArea

GTK et OpenGL

Par défaut, GTK ne propose pas de widget OpenGL. Mais pour palier ce manque, un généreux développeur nommé

Janne Löf a créé GTKGLArea, qui propose un nouveau widget dérivé de GtkDrawingArea permettant d'insérer des fenêtres OpenGL dans vos interfaces GTK. Vous trouverez GTKGLArea à l'adresse indiquée en référence [9]. Notez que vous aurez besoin de GTK en version 1.2 minimum.

A l'attaque

Pour illustrer ce didacticiel, je vous propose de créer une application GTK toute simple affichant un cube en 3D (figure 1). Je sais, je vous avais promis il y a quelques temps de ne plus vous reparler de cube. Si je reviens sur ma promesse, c'est tout simplement pour garder un code source concis et ne pas avoir à utiliser les primitives géométriques standard de Glut. Comme d'habitude, vous pourrez faire tourner le cube avec la souris (bouton gauche) et zoomer (bouton droit). Pour le changement du mode d'affichage, nous utiliserons des boutons GTK en lieu et place de nos habituels raccourcis clavier.

Si vous jetez un coup d'œil au code source de notre programme, vous vous rendrez compte que l'architecture globale est finalement peu différente de ce que nous avions avec glut. On se retrouve avec un système événementiel où un certain nombre de fonctions de rappel sont associées aux différents événements pouvant survenir (création de la fenêtre, clic de souris, appui sur un bouton…).

Initialisation

La phase d'initialisation d'OpenGL permet, je vous le rappelle, de définir les paramètres de base nécessaires au bon fonctionnement de l'application OpenGL : paramètres de perspective, d'éclairage, de couleur… Lorsque nous utilisions Glut, nous avions l'habitude d'appeler depuis la fonction main() une procédure InitOpenGL(). Avec GTKGLArea, il va falloir agir différemment. La raison est simple : pour initialiser OpenGL, il est nécessaire de disposer d'une fenêtre (au sens X−Window du terme) à laquelle sera attaché le contexte OpenGL. Dans le cas de Glut, cette fenêtre est créée avant l'appel à InitOpenGL par la fonction glutCreateWindow(). Mais avec GTK, la fenêtre n'est effectivement créée que lors de l'appel à gtk_widget_show().

La solution pour remédier à ce problème est simple : il suffit d'attacher la fonction d'initialisation d'OpenGL au signal 'realize' qui sera émis lors de la création de la fenêtre.

Mise en place de la fenetre OpenGL dans l'interface GTK

La création de l'interface graphique de notre programme est faite par creeInterface(). Nous ne nous intéresserons qu'à la partie concernant le widget GTKGLArea :

```
if(gdk_gl_query() == FALSE) {
  fprintf(stderr,"Impossible d'utiliser OpenGL\n");
  exit(1);
}

glarea = gtk_gl_area_new(listeAttributs);
gtk_widget_set_events(GTK_WIDGET(glarea),
                      GDK_EXPOSURE_MASK|
                      GDK_BUTTON_PRESS_MASK|
                      GDK_BUTTON_RELEASE_MASK|
                      GDK_POINTER_MOTION_MASK|
                      GDK_POINTER_MOTION_HINT_MASK);
gtk_widget_set_usize(GTK_WIDGET(glarea),300,300);
gtk_box_pack_start(GTK_BOX(boiteh),glarea,TRUE,TRUE,0);
gtk_signal_connect (GTK_OBJECT(glarea), "realize",
                    GTK_SIGNAL_FUNC(initGlarea), NULL);
gtk_signal_connect (GTK_OBJECT(glarea), "expose_event",
                    GTK_SIGNAL_FUNC(affichage), NULL);
gtk_signal_connect (GTK_OBJECT(glarea), "configure_event",
                    GTK_SIGNAL_FUNC(redimGlarea), NULL);
gtk_signal_connect (GTK_OBJECT(glarea), "motion_notify_event",
                    GTK_SIGNAL_FUNC(mouvementSouris), NULL);
gtk_signal_connect (GTK_OBJECT(glarea), "map_event",
                      GTK_SIGNAL_FUNC(rappelMap), NULL);
```

Le premier appel, gdk_gl−query(), permet de tester la possibilité d'utiliser OpenGL. Ensuite, le widget GTKGLArea se crée par un simple appel à :

```
gtk_gl_area_new(int ListeAttrib);
```

Le paramètre ListeAttrib est un tableau d'entiers terminé par zéro (où la constante GDK_GL_NONE) qui détermine le type de fenêtre que l'on souhaite créer, à la manière de ce que nous faisions avec glutInitDisplayMode(). Chaque appel à gtk_gl_area_new() entraîne la création d'un nouveau contexte, ce qui signifie que si vous créez plusieurs fenêtres, il vous faudra redéfinir les paramètres OpenGL pour chacune d'entre elles, notamment, les listes d'affichage. Bien sûr, ce système est coûteux en mémoire, aussi il existe une variante de gtk_gl_area qui permet de partager un même contexte entre plusieurs widgets : gtk_gl_area_share_new(). Nous n'aborderons pas cette fonction ici. Les personnes intéressées se réfèrerons au fichier de documentation gtkglarea.txt fourni avec le code source de l'application.

Avec gtk_gl_area_set_events(), on définit les signaux auxquels doit réagir notre widget, à savoir l'exposition (nécessité de redessiner le widget), l'appui et le relâchement d'un bouton de souris, les mouvements du pointeur. On définit ensuite la taille désirée pour notre widget avec gtk_widget_set_usize(), on insère judicieusement le widget dans l'interface avec gtk_box_pack_start, et enfin, on connecte les différents événements aux fonctions de rappel adéquates par des appels successifs à gtk_signal_connect().

Les fonctions de rappel

Si vous observez le début de chacune des fonctions de rappel, vous remarquerez qu'elles contiennent toutes un appel a gtk_gl_area_make_current(), qui prend comme paramètre le widget OpenGL. Cet appel nécessaire permet de rendre actif le contexte OpenGL de la fenêtre. N'oubliez pas que vous avez la possibilité de travailler avec plusieurs contextes, d'où l'utilité de cette fonction. gtk_gl_area_make_current() renvoie true si l'activation s'est bien déroulée et FALSE en cas de pépin, ce qui ne devrait pas arriver si votre application est correctement construite.

Les fonctions de rappel liées au widget OpenGL se terminent en général par un appel à gtk_widget_queue_draw(), qui émet un signal expose forçant le widget à se redessiner pour prendre en compte les modifications induites par la fonction de rappel.

Nous avons déjà parlé de la fonction de rappel d'initialisation initGLArea(). Elle est fortement similaire à ce que nous avions avec Glut. La fonction affichage() associée à l'événement 'expose' présente deux subtilités :

− Elle commence par un test. Les événements sont stockés dans une pile avant d'être traités par le gestionnaire GTK. Il se peut qu'à un moment donné, plusieurs événements 'expose' pour le même widget soit présents dans la pile. Il est inutile de traiter chacun de ces événements. Un seul réaffichage suffit. Sachant que le membre 'count' de la structure 'evenement' passée en paramètre par le gestionnaire GTK contient le nombre d'événements de type 'expose' restant dans la pile, on ne redessine le widget que si count=0.

− Elle se termine par gtk_gl_swap_buffers() qui, comme vous l'aurez deviné, échange les deux tampons d'image utilisés puisque nous avons activé le double buffering en créant notre fenêtre.

La gestion de la souris est prise en charge par la fonction de rappel mouvementSouris. Lors de l'appel, c'est−à−dire en cas de mouvement de la souris au−dessus du widget OpenGL, la fonction reçoit une structure événement contenant des informations sur l'état de la souris au moment de l'événement. Je vous invite à vous référer au livre de David Odin si vous ne vous rappelez plus les subtilités du motion hint.

La portion de code ci−dessous permet d'extraire de la structure 'evenement' les données de position de la souris (x,y) et d'état des boutons (etat). Le mécanisme de zoom et de rotation est identique à celui que nous utilisions avec Glut.

J'attire votre attention sur un dernier point. Pour calculer les rotations et le zoom de notre objet, nous utilisons les variables xprec et yprec qui servent à stocker les précédentes positions de la souris afin de pouvoir calculer le déplacement de la souris relativement à sa dernière position. Il faut nous assurer que ces variables sont correctement initialisées sous peine de voir notre cube subir une rotation ou un zoom aléatoire lors du premier clic dans la fenêtre. Pour cela, on connecte à la fonction rappelMap() le signal 'map_event' qui est émis lorsque la fenêtre est affichée à l'écran.

Conclusion

Nous voilà arrivés au terme de cette présentation de GTKGLArea. Avec la pratique, vous vous rendrez compte que le duo GTK+OpenGL vous permettra de créer des applications 3D d'aspect très professionnel, possédant une interface graphique efficace et agréable.

Références :

OpenGL 1.2	Woo, Neider, Davis et Shreiner − Campus Press Référence La traduction française de la dernière édition du livre de référence en matière de programmation OpenGL
Introduction à l'Infographie	Foley, Van Dam, Feiner et Hughes − Vuibert La bible de l'informatique graphique.
Programmation Linux avec GTK+	David Odin – Editions Eyrolles. Un excellent ouvrage en francais sur la programmation GTK
www.opengl.org	Le site officiel d'OpenGL. Tout y est : présentation, documents de spécification, liens vers des didacticiels, bibliographie
www.mesa3d.org	Le site de Mesa, l'implémentation libre d'OpenGL la plus utilisée sous Linux
reality.sgi.com/mjk/glut3	La page de glut. Vous y trouverez le manuel de référence glut
http://www.linuxgraphic.org/section3d/openGL/index.html	La section OpenGL du site Linuxgraphic.org. Un tout nouveau forum attend vos questions.
http://www.gtk.org	Le site de gtk
http://www.student.oulu.fi/~jlof/gtkglarea/	Le site de GtkGLArea

Code Source

```
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <gtkgl/gtkglarea.h>
#include <GL/gl.h>
#include <GL/glu.h>

#define DISTANCE_INIT 6.0
#define DISTANCE_MIN 4.5
#define DISTANCE_MAX 25.0

GtkWidget *fenetre,*glarea,*boiteh,*boitev,*boitev2,*bouton,*frame;
GSList *groupe;
int listeAttributs[] = {
  GDK_GL_RGBA,
  GDK_GL_DOUBLEBUFFER,
  GDK_GL_DEPTH_SIZE, 1,
  GDK_GL_NONE
};
int theta=-35,phi=20,xprec,yprec;
float distance=DISTANCE_INIT;

/* Parametres d'éclairage */
GLfloat L0pos[]={ 0.0,2.0,1.0};
GLfloat L0dif[]={ 0.3,0.3,0.8};
GLfloat L1pos[]={ 2.0,2.0,2.0};
GLfloat L1dif[]={ 0.5,0.5,0.5};
GLfloat Mspec[]={0.5,0.5,0.5};
GLfloat Mshiny=50;
```

```
/* Prototypes des fonctions */
void creeInterface();
void affichage(GtkWidget *widget,GdkEventExpose *evenement);
void initGlarea(GtkWidget* widget);
void redimGlarea(GtkWidget* widget, GdkEventConfigure* evenement);
void mouvementSouris(GtkWidget* widget, GdkEventMotion* evenement);
void rappelMap(GtkWidget* widget, GdkEventAny* evenement);
void modeAffichage(GtkWidget* widget, char *chaine);
void basculeEclairage(GtkWidget* widget,gpointer donnee);
void basculeFacesArrieres(GtkWidget *widget,gpointer donnee);

int main(int argc,char **argv)
{
  gtk_init(
  creeInterface();


  /* affichage de l'interface */
  gtk_widget_show_all(fenetre);

  gtk_main();
  return 0;
}



/* Fonction de creation de l'interface graphique */
void creeInterface()
{
  /* la fenetre principale */
  fenetre=gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(fenetre),"Example GtkGLArea");
  gtk_signal_connect(GTK_OBJECT(fenetre),"destroy",GTK_SIGNAL_FUNC(gtk_main_quit),NULL);
  gtk_signal_connect(GTK_OBJECT(fenetre),"delete_event",GTK_SIGNAL_FUNC(gtk_main_quit),NULL);

  /* un boite verticale */
  boiteh=gtk_hbox_new(FALSE,0);
  gtk_container_add(GTK_CONTAINER(fenetre),boiteh);


  /* une glarea */
  if(gdk_gl_query() == FALSE) {
    fprintf(stderr,"Impossible d'utiliser OpenGL\n");
    exit(1);
  }
  glarea = gtk_gl_area_new(listeAttributs);
  gtk_widget_set_events(GTK_WIDGET(glarea),
                        GDK_EXPOSURE_MASK|
                        GDK_BUTTON_PRESS_MASK|
                        GDK_BUTTON_RELEASE_MASK|
                        GDK_POINTER_MOTION_MASK|
                        GDK_POINTER_MOTION_HINT_MASK);
  gtk_widget_set_usize(GTK_WIDGET(glarea),300,300);
  gtk_box_pack_start(GTK_BOX(boiteh),glarea,TRUE,TRUE,0);
  gtk_signal_connect (GTK_OBJECT(glarea), "realize",
                      GTK_SIGNAL_FUNC(initGlarea), NULL);
  gtk_signal_connect (GTK_OBJECT(glarea), "expose_event",
                      GTK_SIGNAL_FUNC(affichage), NULL);
  gtk_signal_connect (GTK_OBJECT(glarea), "configure_event",
                      GTK_SIGNAL_FUNC(redimGlarea), NULL);
  gtk_signal_connect (GTK_OBJECT(glarea), "motion_notify_event",
                      GTK_SIGNAL_FUNC(mouvementSouris), NULL);
```

```
  gtk_signal_connect (GTK_OBJECT(glarea), "map_event",
                      GTK_SIGNAL_FUNC(rappelMap), NULL);

  /* une boite verticale */
  boitev=gtk_vbox_new(FALSE,0);
  gtk_box_pack_start(GTK_BOX(boiteh),boitev,FALSE,FALSE,0);

  /* une frame */
  frame=gtk_frame_new("Mode d'affichage");
  gtk_box_pack_start(GTK_BOX(boitev),frame,FALSE,FALSE,0);

  /* un boite verticale pour la frame */
  boitev2=gtk_vbox_new(FALSE,0);
  gtk_container_add(GTK_CONTAINER(frame),boitev2);

  /* un groupe de bouton radio */
  bouton=gtk_radio_button_new_with_label(NULL,"Plein");
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(bouton),TRUE);
  gtk_box_pack_start(GTK_BOX(boitev2),bouton,FALSE,FALSE,0);
  gtk_signal_connect(GTK_OBJECT(bouton),"toggled",GTK_SIGNAL_FUNC(modeAffichage),"a");

  groupe=gtk_radio_button_group(GTK_RADIO_BUTTON(bouton));
  bouton=gtk_radio_button_new_with_label(groupe,"Fil de fer");
  gtk_box_pack_start(GTK_BOX(boitev2),bouton,FALSE,FALSE,0);
  gtk_signal_connect(GTK_OBJECT(bouton),"toggled",GTK_SIGNAL_FUNC(modeAffichage),"b");


  groupe=gtk_radio_button_group(GTK_RADIO_BUTTON(bouton));
  bouton=gtk_radio_button_new_with_label(groupe,"Points");
  gtk_box_pack_start(GTK_BOX(boitev2),bouton,FALSE,FALSE,0);
  gtk_signal_connect(GTK_OBJECT(bouton),"toggled",GTK_SIGNAL_FUNC(modeAffichage),"c");


  /* un bouton pour bascule pour l'éclairage */
  bouton=gtk_check_button_new_with_label("Eclairage");
  gtk_box_pack_start(GTK_BOX(boitev),bouton,FALSE,FALSE,0);
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(bouton),TRUE);
  gtk_signal_connect(GTK_OBJECT(bouton),"toggled",GTK_SIGNAL_FUNC(basculeEclairage),NULL);

  /* un bouton pour le masquage des faces arrieres*/
  bouton=gtk_check_button_new_with_label("Masquage des face arrières");
  gtk_box_pack_start(GTK_BOX(boitev),bouton,FALSE,FALSE,0);
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(bouton),TRUE);
  gtk_signal_connect(GTK_OBJECT(bouton),"toggled",GTK_SIGNAL_FUNC(basculeFacesArrieres),NULL);

  /* un bouton Quitter */
  bouton=gtk_button_new_with_label("Quitter");
  gtk_box_pack_end(GTK_BOX(boitev),bouton,FALSE,FALSE,0);
  gtk_signal_connect(GTK_OBJECT(bouton),"clicked",GTK_SIGNAL_FUNC(gtk_main_quit),NULL);

}



/* Fonction de rappel pour l'affichage */
void affichage(GtkWidget *widget,GdkEventExpose *evenement)
{
  if (evenement->count > 0)
    return;

  if (gtk_gl_area_make_current(GTK_GL_AREA(widget))) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
```

```
    gluLookAt(0.0,0.0,distance,0.0,0.0,0.0,0.0,1.0,0.0);
    glRotatef(phi,1.0,0.0,0.0);
    glRotatef(theta,0.0,1.0,0.0);

    glBegin(GL_POLYGON);
      glNormal3d(0,0,1); glColor3d(1,0,0);
      glVertex3d(-1,1,1); glVertex3f(-1,-1,1); glVertex3f(1,-1,1); glVertex3f(1,1,1);
    glEnd();
    glBegin(GL_POLYGON);
      glNormal3d(1,0,0); glColor3d(0,1,0);
      glVertex3d(1,1,1); glVertex3f(1,-1,1); glVertex3f(1,-1,-1); glVertex3f(1,1,-1);
    glEnd();

    glBegin(GL_POLYGON);
      glNormal3d(0,0,-1); glColor3d(0,0,1);
      glVertex3d(1,1,-1); glVertex3f(1,-1,-1); glVertex3f(-1,-1,-1); glVertex3f(-1,1,-1);
    glEnd();

    glBegin(GL_POLYGON);
      glNormal3d(-1,0,0); glColor3d(1,1,0);
      glVertex3d(-1,1,-1); glVertex3f(-1,-1,-1); glVertex3f(-1,-1,1); glVertex3f(-1,1,1);
    glEnd();

    glBegin(GL_POLYGON);
      glNormal3d(0,1,0); glColor3d(1,0,1);
      glVertex3d(-1,1,-1); glVertex3f(-1,1,1); glVertex3f(1,1,1); glVertex3f(1,1,-1);
    glEnd();

    glBegin(GL_POLYGON);
      glNormal3d(0,-1,0); glColor3d(0,1,1);
      glVertex3d(-1,-1,1); glVertex3f(-1,-1,-1); glVertex3f(1,-1,-1); glVertex3f(1,-1,1);
    glEnd();

    gtk_gl_area_swapbuffers(GTK_GL_AREA(widget));
  }
}


/* fonction d'initialisation d'OpenGL*/
void initGlarea(GtkWidget* widget)
{
  if (gtk_gl_area_make_current(GTK_GL_AREA(widget))) {
    glClearColor(0.4,0.4,0.4,1.0);
    glColor3d(1,0,0);
    glPointSize(4.0);
    glLineWidth(2.0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    /* Mise en place de la perspective */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,1.0,0.1,DISTANCE_MAX+2);
    glMatrixMode(GL_MODELVIEW);

    /* Parametrage de l'éclairage */
    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,L0dif);
    glLightfv(GL_LIGHT0,GL_SPECULAR,L0dif);
    glLightfv(GL_LIGHT1,GL_DIFFUSE,L1dif);
```

```
    glLightfv(GL_LIGHT1,GL_SPECULAR,L1dif);

    /* Paramétrage du matériau */
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,Mspec);
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Mshiny);
  }
}



/* Fonction de rappel pour le redimensionnement du widget OpenGL  */
void redimGlarea(GtkWidget* widget, GdkEventConfigure* evenement)
{
  int l = widget->allocation.width;
  int h = widget->allocation.height;
  if (gtk_gl_area_make_current(GTK_GL_AREA(widget))) {
    if (l<h)
      glViewport(0,(h-l)/2,l,l);
    else
      glViewport((l-h)/2,0,h,h);
  }
}



/* fonction de rappel pour les mouvements de la souris */
void mouvementSouris(GtkWidget* widget, GdkEventMotion* evenement)
{
  int x,y;
  GdkModifierType etat;
  if (evenement->is_hint)
    gdk_window_get_pointer(evenement->window,
  else {
    x=evenement->x;
    y=evenement->y;
    etat=evenement->state;
  }

  /* bouton gauche = rotation */
  if (etat GDK_BUTTON1_MASK){
    theta+=x-xprec;
    phi+=y-yprec;
    gtk_widget_queue_draw(widget);
  }

  /* bouton droit = Zoom */
  if (etat GDK_BUTTON3_MASK){
    distance+=((float)(y-yprec))/20.0;
    if (distance<DISTANCE_MIN)
      distance=DISTANCE_MIN;
    if (distance>DISTANCE_MAX)
      distance=DISTANCE_MAX;
    gtk_widget_queue_draw(widget);
  }
  xprec=x;yprec=y;
}



/* Fonction de rappel pour le mapping du widget */
void rappelMap(GtkWidget* widget, GdkEventAny* evenement)
{
  int x,y;
```

```
  gdk_window_get_pointer(glarea->window,
  xprec=x;
  yprec=y;
}

/* Fonction de bascule du mode d'affichage */
void modeAffichage(GtkWidget* widget, char *chaine)
{
  if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget)))
    if (gtk_gl_area_make_current(GTK_GL_AREA(glarea))) {
      switch (chaine[0]) {
      case 'a':
        glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
        break;
      case 'b':
        glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
        break;
      case 'c':
        glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
        break;
      }
      gtk_widget_queue_draw(GTK_WIDGET(glarea));
    }
}



/* Fonction de bascule du mode d'eclairage */
void basculeEclairage(GtkWidget* widget,gpointer donnee)
{
  if (gtk_gl_area_make_current(GTK_GL_AREA(glarea))) {
    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget)))
      glEnable(GL_LIGHTING);
    else
      glDisable(GL_LIGHTING);
    gtk_widget_queue_draw(GTK_WIDGET(glarea));
  }
}



/* Fonction de bacule pour le masquage des faces arrieres */
void basculeFacesArrieres(GtkWidget *widget,gpointer donnee)
{
  if (gtk_gl_area_make_current(GTK_GL_AREA(glarea))) {
    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget)))
      glEnable(GL_CULL_FACE);
    else
      glDisable(GL_CULL_FACE);
    gtk_widget_queue_draw(GTK_WIDGET(glarea));
  }
}
```