```
/*
 * OpenGL TP3 : Transparence d'objet, Omgrage porte sur un plan
 * Les initialisations peuvent etre effectuees dans setLight(),
 * L'ombre est a faire avant l'affichage de l'objet
 */
#include <GL/gl.h>
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>

void setLight(void);

/* Constant for the menu */
enum moves
{
  F_NONE,
  F_LIGHT_FIXE,
  F_LIGHT_MOVE,
  F_TABLE_FIXE,
  F_TABLE_MOVE,
  F_VIEW_FIXE,
  F_VIEW_MOVE,
  F_AXE,
  F_TRANSPARENCE,
  F_COLOR_MATERIAL
};


static int displayAxe = GL_TRUE;
static int displayTransparence = GL_TRUE;
static int displayColorMat = GL_FALSE;
enum moves displayFace=GL_BACK;

static int moveViewing = GL_FALSE;
static int moveTable = GL_TRUE;
static int moveLight = GL_FALSE;

static int width, height;         /* Dimension de la fenetre */
static int curx, cury;            /* Position de la souris   */

static GLfloat rotxTable = 0.0;   /* Rotation autour de x */
static GLfloat rotyTable = 0.0;   /* Rotation autour de y */

static GLfloat rotxLight = 0.0;   /* Rotation autour de x */
static GLfloat rotyLight = 0.0;   /* Rotation autour de y */
```



```
static GLfloat rotxViewing = 0.0; /* Rotation autour de x */
static GLfloat rotyViewing = 0.0; /* Rotation autour de y */

static int prey = -1;
static GLfloat zoomFactor = 1;

static GLfloat couleur_plateau[] = {0.2, 0.4, 0.15, 0.5};
static GLfloat couleur_pied[] = {0.3, 0.3, 0.3, 1.0};

/* Quelques couleurs materielles */
GLfloat matZero[4] =  {0.00, 0.00, 0.00, 1.00};
GLfloat matOne[4]  =  {1.00, 1.00, 1.00, 1.00};
GLfloat matRed[4] =  {1.00, 0.00, 0.00, 1.00};
GLfloat matGreen[4]  =  {0.00, 1.00, 0.00, 1.00};
GLfloat matBlue[4]  =  {0.00, 0.00, 1.00, 1.00};

GLfloat shadowMat[4][4];  /* Matrice de projection d'ombre */
GLfloat plane[4]={0.0, 1.0, 0.0, 0.6}; /* Equation du sol  */

/*  GL_SMOOTH is actually the default shading model.  */
void init (void)
{
    glMatrixMode(GL_PROJECTION);/* Definition de matrice de
        projection */
    glLoadIdentity();
    glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 10.0);

    glMatrixMode(GL_MODELVIEW); /* Changement de pile de
        matrices OpenGL */
    glLoadIdentity();

  /* Les deux instructions suivantes produisent un effet
     identique si elles sont placÈes au début de display */
 // setLight(); /* Definition de la source lumineuse 0 */
 // gluLookAt(0.0, 2.0+zoomFactor, 2.0+zoomFactor, 0.0, 0.0,
     0.0, 0.0, 1.0, 0.0); /* Positionnement de camera */

    glClearColor (0.90, 0.90, 0.90, 1.0); /* Couleur de fond en
        noir   */
    glShadeModel (GL_SMOOTH);       /* Model d'ombrage (Gouraud
        ) */

    glFrontFace (GL_CCW);           /* Activation l'elimination
        de faces arrieres */
    glEnable (GL_CULL_FACE);
```

```
    glPolygonMode (GL_BACK, GL_LINE);/* Mode d'affichage des
        faces */
    glPolygonMode (GL_FRONT, GL_FILL);

    glEnable (GL_DEPTH_TEST);       /* Activation de Z-buffer
        */
    glDepthFunc(GL_LEQUAL);

    /* Activer position de camera locale. Elle est placee a l
        'infini par defaut */
    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
}


/* ------------------------------------------------------------ */
/* Modelisation d'un rectangle de longueur "width", de      */
/* largeur , "height" et de couleur "color". Les sommets    */
/* sont ordonnes dans le sens CCW                           */
/* ------------------------------------------------------------ */
void drawRectangle(float width, float height, GLfloat *color)
{
    GLfloat demiw, demih;

    demiw = width/2.;
    demih = height/2.;

    glBegin (GL_POLYGON);
        glColor4fv(color);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(-demiw, -demih, 0.0);

        glColor4fv(color);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(demiw, -demih, 0.0);

        glColor4fv(color);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(demiw, demih, 0.0);

        glColor4fv(color);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(-demiw, demih, 0.0);
    glEnd ();
}
```



```
/* ------------------------------------------------------------ */
/* Modelisation d'une parallelepipede a partir de           */
/* drawRectangle, largueur, hauteur et profondeur de        */
/* la parallelepipede.                                      */
/* ------------------------------------------------------------ */
void drawParallelepipede(float width, float height, float depth
    , GLfloat *color)
{
    /* Face avant */
    glPushMatrix();
        glTranslatef(0.0, 0.0, depth/2.);
        drawRectangle(width, height, color);
    glPopMatrix();

    /* Face arriere */
    glPushMatrix();
        glTranslatef(0.0, 0.0, -depth/2.);
        glRotatef(180.0, 1.0, 0.0, 0.0);        /* pour l'ordre
            des sommets (en CCW) */
        drawRectangle(width, height, color);
    glPopMatrix();

    /* Faces haut/bas */
    glPushMatrix();
        glRotatef(90.0, 1.0, 0.0, 0.0);
        /* Face haut */
        glPushMatrix();
            glTranslatef(0.0, 0.0, height/2.);
            drawRectangle(width, depth, color);
        glPopMatrix();

        /* Face bas */
        glPushMatrix();
            glTranslatef(0.0, 0.0, -height/2.);
            glRotatef(180.0, 1.0, 0.0, 0.0);    /* pour l'ordre
                des sommets (en CCW) */
            drawRectangle(width, depth, color);
        glPopMatrix();
    glPopMatrix();

    /*Faces droite/gauche */
    glPushMatrix();
        glRotatef(90.0, 0.0, 1.0, 0.0);
        /* Face droite */
        glPushMatrix();
```

```
            glTranslatef(0.0, 0.0, width/2.);
            drawRectangle(depth, height, color);
        glPopMatrix();

        /* Face gauche */
        glPushMatrix();
            glTranslatef(0.0, 0.0, -width/2.);
            glRotatef(180.0, 0.0, 1.0, 0.0);    /* pour l'ordre
                des sommets (en CCW) */
            drawRectangle(depth, height, color);
        glPopMatrix();
    glPopMatrix();
}


/* ---------------------------------------------------------- */
/* Modelisation d'une table:un plateau, un cadre et 4 pieds */
/* ---------------------------------------------------------- */
void drawTable(float plateau_w, float plateau_h, float
    plateau_d,
               float pied_w, float pied_h, float pied_d,
               float *coul_plateau, float *coul_pied)
{

    GLfloat matAmbT[4]  =  {0.20, 0.20, 0.20, 0.80};
    /*  GLfloat matDiff[4] =  {0.70, 0.70, 0.56, 1.00};*/
    GLfloat matDiffT[4] =  {0.70, 0.00, 0.00, 0.50};
    GLfloat matSpecT[4] =  {0.50, 0.50, 0.50, 0.50};
    GLfloat matShine   = 20.00;

    /* cadres */
    glPushMatrix();
        glTranslatef(0.0, 0.0, plateau_d/2.0-pied_d/2.0);
        drawParallelepipede(plateau_w, plateau_h/2., pied_d,
            coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(0.0, 0.0, -plateau_d/2.0+pied_d/2.0);
        drawParallelepipede(plateau_w, plateau_h/2., pied_d,
            coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(plateau_w/2.0-pied_w/2.0, 0.0, 0.0);
        drawParallelepipede(pied_w, plateau_h/2., plateau_d,
```



```
            coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(-plateau_w/2.0+pied_w/2.0, 0.0, 0.0);
        drawParallelepipede(pied_w, plateau_h/2., plateau_d,
            coul_pied);
    glPopMatrix();

    /* Pieds */
    glPushMatrix();
        glTranslatef(-plateau_w/2.0+pied_w/2.0, -pied_h/2.0, -
            plateau_d/2.0+pied_d/2.0);
        drawParallelepipede(pied_w, pied_h, pied_d, coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(-plateau_w/2.0+pied_w/2.0, -pied_h/2.0,
            plateau_d/2.0-pied_d/2.0);
        drawParallelepipede(pied_w, pied_h, pied_d, coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(plateau_w/2.0-pied_w/2.0, -pied_h/2.0, -
            plateau_d/2.0+pied_d/2.0);
        drawParallelepipede(pied_w, pied_h, pied_d, coul_pied);
    glPopMatrix();

    glPushMatrix();
        glTranslatef(plateau_w/2.0-pied_w/2.0, -pied_h/2.0,
            plateau_d/2.0-pied_d/2.0);
        drawParallelepipede(pied_w, pied_h, pied_d, coul_pied);
    glPopMatrix();

    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbT);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffT);
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecT);
    glMaterialf (GL_FRONT, GL_SHININESS, matShine);


    /* Plateau */
    glPushMatrix();
        glTranslatef(0.0, plateau_h/2., 0.0);
        drawParallelepipede(plateau_w+plateau_h, plateau_h/2.,
            plateau_d+plateau_h, coul_plateau);
    glPopMatrix();
```

```
}

/* ------------------------------------------------------------ */
/*                Affichage des axes de WCS                     */
/* ------------------------------------------------------------ */
void drawAxes(float lx, float ly, float lz)
{
  glMaterialfv(GL_FRONT, GL_AMBIENT, matOne);
  glMaterialfv(GL_FRONT, GL_SPECULAR, matZero);
  glMaterialf (GL_FRONT, GL_SHININESS, 0.0);

  glMaterialfv(GL_FRONT, GL_EMISSION, matZero);

  glBegin(GL_LINES);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matRed);
    glVertex3f(0, 0, 0);
    glVertex3f(lx, 0, 0);

    glMaterialfv(GL_FRONT, GL_DIFFUSE, matGreen);
    glVertex3f(0, 0, 0);
    glVertex3f(0, ly, 0);

    glMaterialfv(GL_FRONT, GL_DIFFUSE, matBlue);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, lz);
  glEnd();
}


/* ------------------------------------------------------------ */
/* Setup the shadow matrix                                      */
/* ------------------------------------------------------------ */
void ShadowMatrix(GLfloat shadowMat[4][4],
                  GLfloat plane[4], GLfloat lightpos[4])
{
    GLfloat dot;

    dot = plane[0] * lightpos[0] +
          plane[1] * lightpos[1] +
          plane[2] * lightpos[2] +
          plane[3] * lightpos[3];

    shadowMat[0][0] = dot - lightpos[0] * plane[0];
    shadowMat[1][0] = 0.f - lightpos[0] * plane[1];
    shadowMat[2][0] = 0.f - lightpos[0] * plane[2];
```



```
    shadowMat[3][0] = 0.f - lightpos[0] * plane[3];

    shadowMat[0][1] = 0.f - lightpos[1] * plane[0];
    shadowMat[1][1] = dot - lightpos[1] * plane[1];
    shadowMat[2][1] = 0.f - lightpos[1] * plane[2];
    shadowMat[3][1] = 0.f - lightpos[1] * plane[3];

    shadowMat[0][2] = 0.f - lightpos[2] * plane[0];
    shadowMat[1][2] = 0.f - lightpos[2] * plane[1];
    shadowMat[2][2] = dot - lightpos[2] * plane[2];
    shadowMat[3][2] = 0.f - lightpos[2] * plane[3];

    shadowMat[0][3] = 0.f - lightpos[3] * plane[0];
    shadowMat[1][3] = 0.f - lightpos[3] * plane[1];
    shadowMat[2][3] = 0.f - lightpos[3] * plane[2];
    shadowMat[3][3] = dot - lightpos[3] * plane[3];
}

/* ------------------------------------------------------------ */
/* Initialisation de l'ombrage avec utilisation du stencil      */
/* buffer pour limiter l'affichage de l'ombre au plan de        */
/* projection                                                   */
/* ------------------------------------------------------------ */
void InitCastShadows()
{
    glClear(GL_STENCIL_BUFFER_BIT);
    glEnable(GL_STENCIL_TEST);
    glStencilFunc(GL_ALWAYS, 1, 1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
}


/* ------------------------------------------------------------ */
/* Affichage de la projection de la table pour créer            */
/* l'ombre sur le sol                                           */
/* ------------------------------------------------------------ */
void EndCastShadows(GLfloat matrix[4][4])
{
    GLfloat colorOmbre_plateau[] = {0.f, 0.f, 0.f, 0.5f};
    GLfloat colorOmbre_pied[] = {0.f, 0.f, 0.f, 0.85f};

    glStencilFunc(GL_EQUAL, 1, 1);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);

    glPushMatrix();
```

```
            glMultMatrixf((GLfloat *) matrix);

            if (displayTransparence) {
                colorOmbre_plateau[3] = couleur_plateau[3];
            }
            else
                colorOmbre_plateau[3] = 0.8f;

            drawTable(0.8, 0.1, 0.5, 0.04, 0.6, 0.04,
                colorOmbre_plateau, colorOmbre_pied);
        glPopMatrix();

        glEnable(GL_DEPTH_TEST);
        glDisable(GL_STENCIL_TEST);
        glEnable(GL_LIGHTING);
    }


    /* ------------------------------------------------------------ */
    /* Setup the light parameters                                   */
    /* ------------------------------------------------------------ */
    void setLight(void)
    {
        GLfloat light0Pos[4] = {0.50, 1.25, 0.75, 0.00};
        GLfloat light0Amb[4] = {0.40, 0.40, 0.40, 1.00};
        GLfloat light0Diff[4] = {1.00, 1.00, 1.00, 1.00};
        GLfloat light0Spec[4] = {1.00, 1.00, 1.00, 1.00};

        GLfloat light0SpotExp = 0.00;
        GLfloat light0SpotCutoff = 180.00;

        GLfloat light0matAmb[4]  =  {0.20, 0.20, 0.20, 1.00};
        GLfloat light0matDif[4]  =  {0.60, 0.60, 0.60, 1.00};
        GLfloat light0matEmi[4]  =  {0.0, 0.0, 0.0, 1.00};
        GLfloat matZero[4] =  {0.00, 0.00, 0.00, 1.00};

        glEnable(GL_LIGHTING);  /* Activation de model d'eclairage
            */
        glEnable(GL_LIGHT0);    /* Activer la source 0
            */

        /* Definition de proprietes de la source 0 */
        glLightfv(GL_LIGHT0, GL_POSITION, light0Pos);
        glLightfv(GL_LIGHT0, GL_AMBIENT, light0Amb);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light0Diff);
        glLightfv(GL_LIGHT0, GL_SPECULAR, light0Spec);
```



```
        /* Definition de proprietes de la sphere qui represente la
            source 0 */
        glMaterialfv(GL_FRONT, GL_AMBIENT, light0matAmb);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, light0matDif);
        glMaterialfv(GL_FRONT, GL_EMISSION, light0matEmi);
        glMaterialfv(GL_FRONT, GL_SPECULAR, matZero);

        /* Positionnement de la sphere a la meme place que la
            source 0 */
        glPushMatrix();
            glTranslatef(light0Pos[0], light0Pos[1], light0Pos[2]);
            glutSolidSphere(0.05, 16, 16);
        glPopMatrix();

        /* Intialization of shadow application */
        InitCastShadows();
        ShadowMatrix(shadowMat, plane, light0Pos);

        // glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light0SpotDir);
     // glLightf (GL_LIGHT0, GL_SPOT_EXPONENT, light0SpotExp);
     // glLightf (GL_LIGHT0, GL_SPOT_CUTOFF, light0SpotCutoff);
    }



    /* ------------------------------------------------------------ */
    /* Setup the objects material                                   */
    /* ------------------------------------------------------------ */
    void setMaterial()
    {
        GLfloat matAmb[4]  =  {0.20, 0.20, 0.20, 1.00};
        /*  GLfloat matDiff[4] =  {0.70, 0.70, 0.56, 1.00};*/
        GLfloat matDiff[4] =  {0.70, 0.00, 0.00, 1.00};
        GLfloat matSpec[4] =  {0.50, 0.50, 0.50, 1.00};
        GLfloat matShine   = 20.00;

        glMaterialfv(GL_FRONT, GL_AMBIENT, matAmb);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);
        glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
        glMaterialf (GL_FRONT, GL_SHININESS, matShine);

        glMaterialfv(GL_FRONT, GL_EMISSION, matZero);
    }
```

```
/* ------------------------------------------------------------ */
/* Les fonctions glut :  display, reshape, specialkey, menu */
/* ------------------------------------------------------------ */
void display(void)
{
    GLfloat colorSol[]={0.35, 0.05, 0.05, 0.85};

    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if (displayColorMat) {
        glEnable (GL_COLOR_MATERIAL);   /* Activer les couleurs
            des sommets      */
        glColorMaterial (GL_FRONT_AND_BACK,
            GL_AMBIENT_AND_DIFFUSE);
    }
    else
        glDisable(GL_COLOR_MATERIAL);

    glPushMatrix();
        glRotatef (rotxLight, 1.0, 0.0, 0.0);
        glRotatef (rotyLight, 0.0, 1.0, 0.0);
        setLight();
    glPopMatrix ();

    glPushMatrix();
        glRotatef (rotxViewing, 1.0, 0.0, 0.0);
        glRotatef (rotyViewing, 0.0, 1.0, 0.0);
        gluLookAt(0.0, 2.0+zoomFactor ,2.0+zoomFactor, 0.0, 0.0
            , 0.0, 0.0, 1.0, 0.0);
    // glPopMatrix (); /* gluLookAt n'est pas applique s'il n
        'est pas avec drawTable */

        if (displayAxe) drawAxes(0.5, 0.4, 0.4);


        /* Annuler les rotations Viewing pour la table */
        glRotatef (-rotyViewing, 1.0, 0.0, 0.0);
        glRotatef (-rotxViewing, 0.0, 1.0, 0.0);

    // glPushMatrix ();
        glRotatef (rotxTable, 1.0, 0.0, 0.0);
        glRotatef (rotyTable, 0.0, 1.0, 0.0);

        if (displayTransparence) {
            glEnable(GL_BLEND) ;
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```



```
        }

        /* Dessiner le sol */
        glPushMatrix();
            glTranslatef(0.0, -0.60, 0.0);
            glRotatef(-90, 1.0, 0.0, 0.0);
            drawRectangle(2.0, 2.0, colorSol);
        glPopMatrix();

        /* Dessiner l'ombre de la table, a faire avant de
            dessiner le vrai objet */
        EndCastShadows(shadowMat);

        setMaterial();
        drawTable(0.8, 0.1, 0.5, 0.04, 0.6, 0.04,
            couleur_plateau, couleur_pied);

    glPopMatrix();

    glDisable(GL_BLEND);
    glutSwapBuffers ();

}


void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    width = w; height = h;
}


void specialkey (int key, int x, int y)
{
    switch (key) {
    case GLUT_KEY_LEFT :
      if (moveTable) {
        rotyTable -= 5.0;
        if (rotyTable < 0.0) rotyTable += 360.0;
      }
      if (moveLight) {
        rotyLight -= 5.0;
        if (rotyLight < 0.0) rotyLight += 360.0;
      }
      if (moveViewing) {
        rotyViewing -= 5.0;
```

```
      if (rotyViewing < 0.0) rotyViewing += 360.0;
    }
      break;
  case GLUT_KEY_RIGHT :
    if (moveTable) {
      rotyTable += 5.0;
      if (rotyTable > 360.0) rotyTable -= 360.0;
    }
    if (moveLight) {
      rotyLight += 5.0;
      if (rotyLight > 360.0) rotyLight -= 360.0;
    }
    if (moveViewing) {
      rotyViewing += 5.0;
      if (rotyViewing > 360.0) rotyViewing -= 360.0;
    }
      break;
  case GLUT_KEY_UP :
    if (moveTable) {
      rotxTable -= 5.0;
      if (rotxTable < 0.0) rotxTable += 360.0;
    }
    if (moveLight) {
      rotxLight -= 5.0;
      if (rotxLight < 0.0) rotxLight += 360.0;
    }
    if (moveViewing) {
      rotxViewing -= 5.0;
      if (rotxViewing < 0.0) rotxViewing += 360.0;
    }
      break;
  case GLUT_KEY_DOWN :
    if (moveTable) {
      rotxTable += 5.0;
      if (rotxTable > 360.0) rotxTable -= 360.0;
    }
    if (moveLight) {
      rotxLight += 5.0;
      if (rotxLight > 360.0) rotxLight -= 360.0;
    }
    if (moveViewing) {
      rotxViewing += 5.0;
      if (rotxViewing > 360.0) rotxViewing -= 360.0;
    }
      break;
```



```
      case GLUT_KEY_END :
          exit (0);

      }
      glutPostRedisplay ();
  }

  /* Fonction de traitement du mouvement de la souris */
  void motion(int x, int y)
  {
      if (prey != -1 && abs(y-prey)<10) {
          zoomFactor += (float) (y-prey)*5.0 / width;
          glutPostRedisplay ();
      }
      prey = y;
  }


  void menu(int value)
  {
    switch (value)
    {
    case F_NONE:
      break;

    case F_LIGHT_MOVE:
      moveLight = GL_TRUE;
       break;
    case F_LIGHT_FIXE:
      moveLight = GL_FALSE;
       break;

    case F_TABLE_MOVE:
      moveTable = GL_TRUE;
       break;
    case F_TABLE_FIXE:
      moveTable = GL_FALSE;
       break;

    case F_VIEW_MOVE:
      moveViewing = GL_TRUE;
       break;
    case F_VIEW_FIXE:
      moveViewing = GL_FALSE;
       break;
```

```
      case F_COLOR_MATERIAL:
        displayColorMat = !displayColorMat;
        break;

      case F_TRANSPARENCE:
        displayTransparence = !displayTransparence;
        break;

      case F_AXE:
        displayAxe = !displayAxe;
        break;
      }

      glutPostRedisplay ();
    }



    /*  Main Loop
     *  Open window with initial window size, title bar,
     *  RGBA display mode, and handle input events.
     */
    int main(int argc, char** argv)
    {
        glutInit ( &argc, argv );
        glutInitDisplayMode (GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB |
            GLUT_STENCIL);

        glutInitWindowSize ( 500, 500 );
        glutInitWindowPosition ( 100, 100 );
        glutCreateWindow (argv[0]);

        glutCreateMenu(menu);
        glutAddMenuEntry("Light :       ", F_NONE          );
        glutAddMenuEntry("   Move       ", F_LIGHT_MOVE      );
        glutAddMenuEntry("   Fixe       ", F_LIGHT_FIXE      );
        glutAddMenuEntry("              ", F_NONE          );

        glutAddMenuEntry("Table :       ", F_NONE          );
        glutAddMenuEntry("   Move       ", F_TABLE_MOVE      );
        glutAddMenuEntry("   Fixe       ", F_TABLE_FIXE      );
        glutAddMenuEntry("              ", F_NONE          );

        glutAddMenuEntry("Viewing :     ", F_NONE          );
        glutAddMenuEntry("   Move       ", F_VIEW_MOVE      );
        glutAddMenuEntry("   Fixe       ", F_VIEW_FIXE      );
```



```
        glutAddMenuEntry("              ", F_NONE          );

        glutAddMenuEntry("Color Material :", F_NONE          );
        glutAddMenuEntry("   Toggle ColorMat", F_COLOR_MATERIAL );
        glutAddMenuEntry("              ", F_NONE          );

        glutAddMenuEntry("Transparence :", F_NONE          );
        glutAddMenuEntry("   Toggle Transparence", F_TRANSPARENCE )
            ;
        glutAddMenuEntry("              ", F_NONE          );

        glutAddMenuEntry("Axes :        ", F_NONE          );
        glutAddMenuEntry("   Toggle Axes", F_AXE      );
        glutAttachMenu(GLUT_RIGHT_BUTTON);

        init();
        glutDisplayFunc (display);
        glutReshapeFunc (reshape);
        glutSpecialFunc (specialkey);
        glutMotionFunc(motion);
        glutMainLoop();

        return 0;
    }
```