Introduction :

Le sujet principal de cet didacticiel est une fois de plus le placage de texture. Nous aborderons aujourd'hui les textures procédurales, l'animation, et l'éclairage de textures. En plus de cela, nous nous intéresserons à la simulation de brouillard, à la lecture d'images TIFF et nous reviendrons sur un sujet que nous avons effleuré dans un précédent tutoriel : les normales.

Pour illustrer ces nombreuses notions, je vous propose un programme exemple relativement gros, qui reprend la plupart des notions que nous avons étudiées jusqu'à présent. Il s'agit d'un… cube tournant ! Je vous promets que c'est la dernière fois. Dès le prochain didacticiel, on change de sujet. Le programme qui nous intéresse aujourd'hui vous permet zoomer sur le cube, de l'éclairer et de le noyer dans du brouillard. Par ailleurs, deux textures différentes sont appliquées sur les faces du cube, et une de ces textures est une texture procédurale animée !

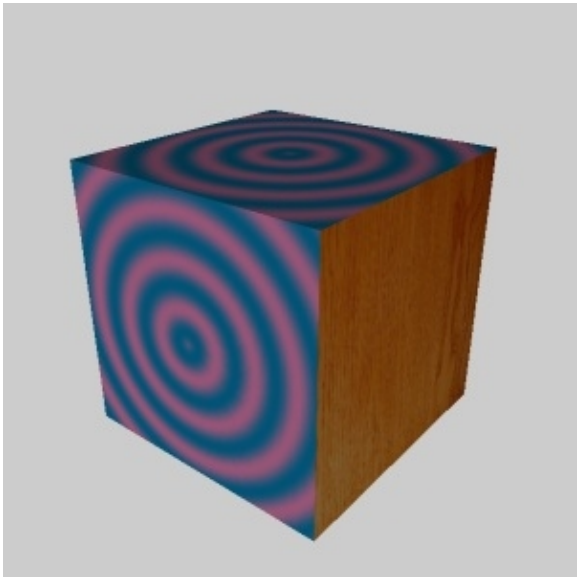

Figure 1 : Le programme exemple

Utilisation de plusieurs textures

La dernière fois, nous avons plaqué sur chacune des faces du cube la même image. Aujourd'hui, nous allons plaquer une image de bois sur 3 faces du cube, et une texture procédurale sur les 3 autres faces du cube. Il nous faut pour cela mettre en œuvre le mécanisme d'objets de texture qu'OpenGL met à notre disposition.

Lorsqu'on souhaite travailler avec plusieurs textures, il faut leur assigner un identifiant, c'est−à−dire un nombre entier. OpenGL dispose d'un système interne de gestion des identifiants de texture ('texture ID' en anglais). Lorsque vous voulez créer une texture, il vous faut demander à OpenGL de vous attribuer un identifiant en faisant un appel à la fonction

```
Void glGenTextures(GLsizei nombre,Gluint *idtextures)
```

‘nombre’ indique le nombre de textures que vous souhaiter créer, et ‘idtextures’ est un pointeur vers un tableau d’entiers qui contiendra au retour de la fonction les ‘nombres’ identifiants de texture que vous avez demandés.

Par la suite, on retrouve encore le mécanisme de machine à état d’OpenGL : vous définissez une texture active, et toutes les opérations sur les textures qui seront faites par la suite seront appliquées à la texture active. La fonction que vous permet de définir la texture active est glBindTexture(), dont le prototype est le suivant :

```
void glBindTexture(GLenum cible, Gluint idtexture)
```

‘cible’ correspond au type de texture utilisé (GL_TEXTURE_1D, GL_TEXTURE_2D ou GL_TEXTURE_3D). Ce paramètre est nécessaire car l’allocation d’espace pour la texture se fait lors du premier appel à glBindTexture() pour un numéro de texture donné, et non lors de l’attribution de l’identifiant. Le paramètre ‘idtexture’ désigne l’identifiant de la texture qu’on souhaite rendre active.

L’utilisation des textures dans un programme OpenGL se fait donc en général de la façon suivante :

− Lors de l’initialisation, on réserve les identifiants de texture et on crée les textures :

```
glGenTextures(2,IdTex);
glBindTexture(GL_TEXTURE_2D,IdTex[0]) ;
/* Création de la première texture (avec glTexParameteri(), glTexImage2D...) */
...
glBindTexture(GL_TEXTURE_2D,IdTex[1]) ;
/* Création de la deuxième texture */
...
```

− Au cours de la fonction d’affichage, on change éventuellement la texture active avant la description de chaque face :

```
void affichage()
{
 ...
 glBindTexture(GL_TEXTURE_2D,IdTex[0]);
 glBegin(GL_POLYGON);
 /* description de la face 1 */
 glEnd();

 glBindTexture(GL_TEXTURE_2D,IdTex[1]);
 glBegin(GL_POLYGON);
 /* description de la face 2 */
 glEnd();
 ...
 }
```

Chargement d'image TIFF

La dernière fois, nous avons chargé une image JPEG grâce à la bibliothèque JPEG. Aujourd’hui, nous allons placer dans notre première texture une image lue dans un fichier TIFF, avec la bibliothèque libtiff (référence [8]). Le format TIFF (Tag Image File Format) est un format de fichier sans perte de qualité et qui offre la possibilité de stocker une couche alpha pour la transparence des images. Je ne vais pas détailler les prototypes des fonctions utilisées pour la lecture du fichier TIFF, vous verrez qu’on trouve bon nombre de similitudes avec la bibliothèque JPEG. Voici la portion de code qui permet de lire une image TIFF :

```
TIFF* tif = TIFFOpen(fichier, "r");
if (tif) {
  TIFFGetField(tif, TIFFTAG_IMAGEWIDTH, &l);
  TIFFGetField(tif, TIFFTAG_IMAGELENGTH, &h);
  npixels = l * h;
  raster = (uint32*) _TIFFmalloc(npixels * sizeof (uint32));
  if (raster != NULL) {
    /* lecture de l'image */
    if (TIFFReadRGBAImage(tif, l, h, raster, 1)) {
      /* transfert de l'image vers le tableau 'image' */
      for (i=0;i<256;i++)
        for (j=0;j<256;j++) {
          image[i][j][0]=((unsigned char *)raster)[i*256*4+j*4+0];
          image[i][j][1]=((unsigned char *)raster)[i*256*4+j*4+1];
```

```
          image[i][j][2]=((unsigned char *)raster)[i*256*4+j*4+2];
        }
    }
  else {
    printf("erreur de chargement du fichier %s\n",fichier);
    exit(0);
  }
  _TIFFfree(raster);
}
TIFFClose(tif);
```

Dans un premier temps, on ouvre le fichier image avec TIFFOpen(). Si l'ouverture se passe bien, on récupère la taille de l'image avec la fonction TIFFGetField(), puis on alloue un tableau mono−dimensionnel de la taille de l'image avec TIFFmalloc(). Un pixel d'image correspond à un quadruplet RGBA, stocké sur 4 octets, ce qui correspond à un entier (type uint32). Si l'allocation se passe correctement, on peut lancer la lecture de l'image grâce à la fonction TIFFReadRGBAImage(). Ici, il n'y a pas besoin de faire un balayage comme nous l'avons fait avec la libjpeg. Au final, l'image est placée dans le tableau 'raster'. Il ne reste plus qu'à réorganiser les données de 'raster' dans un tableau à 3 dimensions nommé 'image', compatible avec la fonction glTexImage2D(). Pour faire les choses dans les règles, on libère les données contenues dans 'raster' et dont nous n'avons plus besoin avec un appel à _TIFFfree(), puis on ferme le fichier avec TIFFClose().

Création d'une texture procédurale

Nous avons vu le au cours du précédent tutoriel le principe des textures procédurales : il consiste à utiliser une fonction mathématique au lieu d'une image pour habiller un objet 3D. En guise de seconde texture pour notre cube, nous allons utiliser une texture procédurale basée sur la fonction mathématique cosinus. Dans un moteur 3D, le gros avantage des textures procédurales et qu'elles prennent peu de place en mémoire, contrairement à une image. Avec OpenGL nous n'allons pas tirer profit de cet avantage puisque nous devons passer par un tableau pour spécifier une texture avec glTexImage2D() : nous remplissons un tableau 2D avec les valeurs d'une fonction mathématique. Dans le programme exemple, le chargement de la texture procédurale se fait dans chargeTextureProc().

```
  for (i=0;i<256;i++)
    for (j=0;j<256;j++) {
      a=fonctionTexture(i,j);
      image[i][j][0]=a;
      image[i][j][1]=128;
      image[i][j][2]=128;
    }
```

De la même manière que pour une texture image, on parcourt le tableau à deux dimensions. Pour chaque cellule, on calcule la valeur 'a' de la fonction mathématique (nommée fonctionTexture()) pour le point considéré. Cette fonction renvoie une valeur comprise entre 0 et 255, puis on affecte une couleur dépendante de 'a' aux composantes rouge verte et bleue du point :

```
int fonctionTexture(int x,int y)
{
  float dx=(128.0−(float)x)/255.0*40.0;
  float dy=(128.0−(float)y)/255.0*40.0;
  float a=cos(sqrt(dx*dx+dy*dy)+decalage);
  return (int)((a+1.0)/2.0*255);
}
```

Nous utilisons aujourd'hui comme fonction procédurale un cosinus qui nous permet d'obtenir des cercles concentriques. Je ne rentre pas dans les détails mathématiques. Ceux d'entre−vous qui sont spécialistes n'auront aucun mal à décortiquer cette fonction.

Animation de la texture procedurale

Nous souhaitons animer la texture que nous venons de créer en faisant bouger les cercles concentriques générés par la fonction. On utilise pour cela une variable 'décalage' introduite dans le cosinus de la fonction procédurale, et la fonction de rappel d'oisiveté ('idle'). Cette fonction dont nous avons déjà parlé est appelée chaque fois que le gestionnaire d'événements n'a rien à faire (pas d'événement clavier, souris,… à traiter). La définition d'une fonction de

rappel d'oisiveté se fait grâce à :

```
void glutIdleFunc(void (*func)(void))
```

Dans la fonction de rappel, nous allons incrémenter la valeur du terme 'décalage'. La fonction cosinus est périodique et sa période vaut 2 Pi, c'est−à−dire que cos(x+2Pi)=cos(x), et chaque fois que décalage devient supérieur à 2 Pi, on décrémente sa valeur de 2 Pi, ce qui permet d'éviter les problèmes de dépassement de capacité :

```
void inactif()
{
  /* increment du décalage */
  decalage+=0.1;
  if (decalage>2*PI)
    decalage−=2*PI;
  /* rechargement de la texture */
  chargeTextureProc(IdTex[1]);
  glutPostRedisplay();
}
```

Ce décalage va se traduire à l'écran par un léger déplacement des cercles constituant la texture. La répétition de ce décalage chaque fois que le gestionnaire de déplacement n'a rien à faire va produire l'effet de mouvement sur la texture.

Eclairage des textures

L'éclairage des textures se fait de la même manière que pour les objets non texturés, nous y avons déjà consacré un didacticiel. La seule différence est que l'utilisation de textures a pour effet d'atténuer fortement l'éclairage spéculaire (les tâches lumineuses donnant un aspect brillant aux objets). Pour remédier à ce problème, OpenGL permet de séparer le calcul de l'éclairage spéculaire grâce à l'appel suivant :

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SEPARATE_SPECULAR_COLOR);
```

Pour désactiver cette fonctionnalité, remplacez GL_SEPARATE_SPECULAR_COLOR par GL_SINGLE_COLOR. Dans le programme exemple, la touche 's' permet de basculer entre ces deux modes de calcul de l'éclairage.

Brouillard

Il est relativement facile de créer un effet de brouillard avec OpenGL. En effet, le rendu utilise la notion de profondeur (distance d'un objet à l'observateur) pour le rendu. En 'mixant' la couleur d'un objet avec une couleur 'du brouillard' en fonction de la distance (plus l'objet est éloigné, plus la couleur de fond est prépondérante), on crée un joli brouillard.

L'activation et la désactivation du brouillard se fait respectivement avec glEnable(GL_FOG) et glDisable(GL_FOG). La modification des paramètres du brouillard se fait avec la fonction

```
glFogf(GLenum nomparam,Glfloat valeur)
```

'paramètre' est le nom du paramètre à modifier, et 'valeur' est évidemment la nouvelle valeur du paramètre. Il existe plusieurs modes de brouillard, correspondant à différentes fonctions de mélange de la couleur de l'objet et de la couleur du brouillard. Je ne vais pas détailler tous les modes aujourd'hui (vous trouverez tout ça dans le livre [1]), je vous livre simplement les paramètres utilisés pour le programme exemple. Il s'agit d'un brouillard de type exponentiel :

```
glFogi(GL_FOG_MODE,GL_EXP); /* Réglage du type de brouillard */
glFogfv(GL_FOG_COLOR,couleurAP); /* Couleur du brouillard */
glFogf(GL_FOG_START,0); /* distance de début du brouillard  */
glFogf(GL_FOG_END,15); /* distance de fin du brouillard */
glFogf(GL_FOG_DENSITY,0.35); /* Densité du brouillard */
```

Normales

Nous avons déjà parlé des normales au cours du tutoriel concernant l'éclairage, mais nous avons en fait détourné le problème en utilisant comme objets 3D des primitives glut dont les normales étaient déjà mises en place. Aujourd'hui nous allons voir en détail comment définir une normale.

Les algorithmes d'éclairage ont besoin de connaître l'orientation des faces d'un objet. Pour cela, ils mettent en jeu la notion de normale, un vecteur perpendiculaire au polygone considéré (figure 2). Bien sûr, en connaissant les points constituant un polygone, il est possible de calculer un vecteur normal à ce polygone, mais ce calcul est coûteux, et sauf convention implicite dans l'ordre des sommets, il ne permet pas de déterminer automatiquement les faces avant et arrière du polygone. C'est pourquoi il nous faut spécifier les vecteurs normaux.

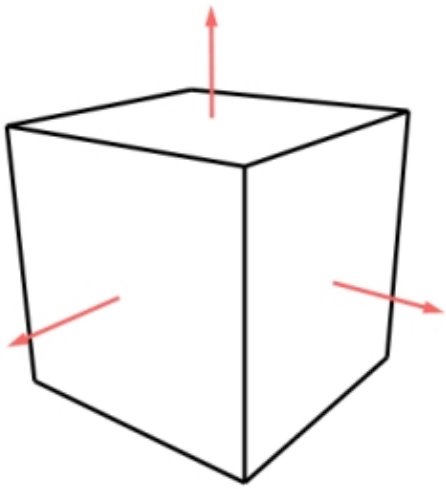

Figure 2 : un cube et les normales de ses faces

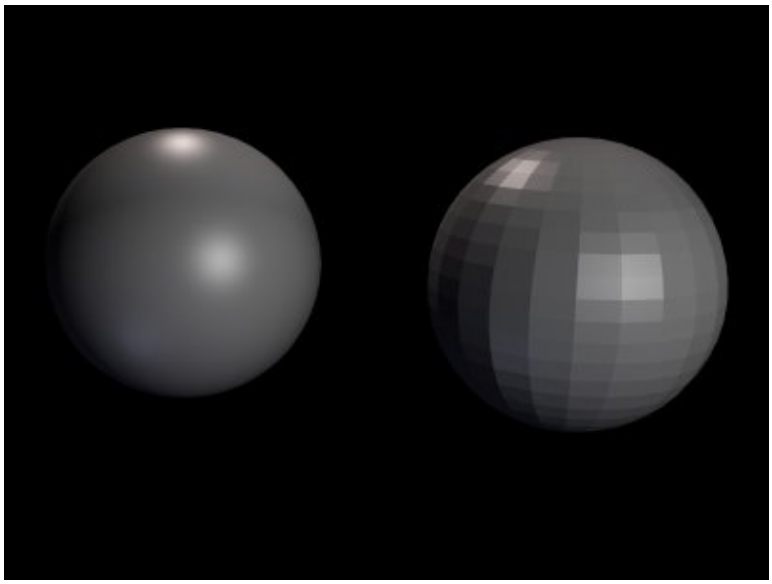

Figure 3 : deux sphères, avec et sans lissage des faces

En OpenGL, on n'associe pas les normales aux polygones mais aux sommets, car cela permet de créer des effets de lissage sur les surfaces courbes (figure 3). Nous aurons l'occasion d'appliquer le lissage la prochaine fois. La spécification des normales se fait comme d'habitude grâce à un système de normale active définie avec la fonction

```
glNormal3f(float x,float y,float z);
```

Dans notre cas, chacun des sommets d'une face se voit affecter la même normale, et donc la définition d'une face de notre cube se fait grâce à une portion de code du type :

```
glBegin(GL_POLYGON);
 glNormal3f(0.0,0.0,1.0);
 glTexCoord2f(0.0,1.0);   glVertex3f(−0.5, 0.5, 0.5);
 glTexCoord2f(0.0,0.0);   glVertex3f(−0.5,−0.5, 0.5);
 glTexCoord2f(1.0,0.0);   glVertex3f( 0.5,−0.5, 0.5);
 glTexCoord2f(1.0,1.0);   glVertex3f( 0.5, 0.5, 0.5);
glEnd();
```

Attention, même si à priori les vecteurs (0,0,1) et (0,0,2) pointent dans la même direction, il est important pour l'algorithme d'éclairage que les vecteurs normaux aient une longueur de 1.

Zoom

Dernière nouveauté, le programme permet de zoomer en maintenant le bouton droit de la souris enfoncé en déplaçant la souris de haut en bas. Le principe d'implémentation de ce système est le même que pour la rotation avec le bouton gauche : une variable 'distance' permet de stocker la distance de l'utilisateur au centre de la scène. Grâce aux fonctions de rappel liées à la souris et ses mouvements, on met à jour 'distance' en fonction des mouvements de l'utilisateur. La valeur de 'distance' est utilisée dans la fonction d'affichage pour le positionnement de l'utilisateur avec gluLookAt().

Conclusion

Le programme utilise les bibliothèques TIFF et mathématiques (pour le cosinus et la racine carré (sqrt())). Il faut donc spécifier au compilateur qu'il doit lier ces deux bibliothèques en rajoutant '−lm –ltiff' à la commande de compilation.

Ce didacticiel est le dernier qui traitera exclusivement de l'apprentissage d'OpenGL. Dès le prochain tutoriel, nous nous intéresserons à des sujets plus pratiques dans lesquels OpenGL n'interviendra plus que comme un outil. Bien sûr, nous n'avons pas étudié toutes les possibilités qu'offre l'API et les futurs didacticiels seront l'occasion d'introduire des concepts plus avancés de la programmation OpenGL.

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
Eclairage et rendu numériques	Jeremy Birn − Campus Press. Orienté pratique, cet ouvrage vous apprendra à créer des rendus de qualité.
Introduction à l'Infographie	Foley, Van Dam, Feiner et Hughes − Vuibert La bible de l'informatique graphique.
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.libtiff.org	Le site web de la bibliothèque libtiff

Code source :

```
/**************/
/* Texture2.c */
/**************/

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <tiffio.h>
#include <string.h>
#include <math.h>

#define PI 3.14159265

/* declaration des variables */
float distance=2.5; /* distance de l'observateur a l'origine */
int anglex=30,angley=20,xprec,yprec; /* stockage des déplacements de souris */
GLbitfield masqueClear; /* masque pour l'utilisation de glClear() */
GLfloat couleurAP[]={0.8,0.8,0.8,1.0}; /* couleur de fond */
int IdTex[2]; /* tableau d'Id pour les 2 textures */
float decalage=0; /* décalage de la texture procedurale pour l'animation */

/* Parametres de lumière */
GLfloat L0pos[]={ 0.0,2.0,-1.0};
GLfloat L0dif[]={ 1.0,0.6,0.6};
GLfloat L1pos[]={ 2.0,2.0,2.0};
GLfloat L1dif[]={ 0.0,0.5,1.0};
GLfloat Mspec[]={0.5,0.5,0.5};
GLfloat Mshiny=50;

/* déclaration des indicateurs booleens */
unsigned char b_gauche=0; /* le bouton gauche de la souris est il presse ? */
unsigned char b_droit=0; /* le bouton gauche de la souris est il presse ? */
unsigned char prof=1; /* Tampon de profondeur */
unsigned char brouillard=0; /* brouillard */
unsigned char eclairage=1; /* eclairage */
unsigned char separe=1; /* séparation de la composante spéculaire pour
                           l'éclairage des textures */

/* prototypes des fonctions */
void init();
void affichage();
void clavier(unsigned char key,int x,int y);
void souris(int button, int state,int x,int y);
void mouvSouris(int x,int y);
void redim(int w,int h);
void inactif();
void chargeTextureTiff(char *fichier,int numtex);
void chargeTextureProc(int numtex);
int fonctionTexture(int x,int y);



/*********************************/
/* int main(int argc,char **argv) */
/*********************************/
/* fonction principale            */
/*********************************/

int main(int argc,char **argv)
{
  /* initialisation de glut */
```

```
  glutInit(
  glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
  glutInitWindowSize(400,400);
  glutCreateWindow(argv[0]);

  /* Chargement des textures */
  glGenTextures(2,IdTex);
  chargeTextureTiff("texture.tif",IdTex[0]);
  chargeTextureProc(IdTex[1]);

  /* initialisation d'OpenGL */
  init();

  /* mise en place des fonctions de rappel glut */
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);
  glutMouseFunc(souris);
  glutMotionFunc(mouvSouris);
  glutReshapeFunc(redim);
  glutIdleFunc(inactif);

  /* Boucle principale */
  glutMainLoop();
  return 0;
}



/***************************/
/* void init()             */
/***************************/
/* Initialisation d'OpenGL */
/***************************/

void init()
{
  /* Parametres de base */
  glClearColor(couleurAP[0],couleurAP[1],couleurAP[2],couleurAP[3]);
  glColor3f(1.0,1.0,1.0);
  glShadeModel(GL_SMOOTH);

  /* Parametres de perspective */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,1,0.1,16.0);
  glMatrixMode(GL_MODELVIEW);


  /* Paramètres d'eclaraiage */
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
  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);
  if (eclairage)
    glEnable(GL_LIGHTING);
  else
    glDisable(GL_LIGHTING);
  if (separe)
    glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SEPARATE_SPECULAR_COLOR);
  else
    glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SINGLE_COLOR);

  /* Paramétres du matériau */
  glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,Mspec);
  glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Mshiny);
```

```
  /* Mise en place des textures */
   glEnable(GL_TEXTURE_2D);

  /* mise en place du tampon de profondeur */
  if (prof) {
    masqueClear=GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT;
    glEnable(GL_DEPTH_TEST);
  }
  else {
    masqueClear=GL_COLOR_BUFFER_BIT;
    glDisable(GL_DEPTH_TEST);
  }

  /* Mise en place du brouillard */
  glFogi(GL_FOG_MODE,GL_EXP);
  glFogfv(GL_FOG_COLOR,couleurAP);
  glFogf(GL_FOG_START,0);
  glFogf(GL_FOG_END,15);
  glFogf(GL_FOG_DENSITY,0.35);
  if (brouillard)
    glEnable(GL_FOG);
  else
    glDisable(GL_FOG);
}



/*************************************/
/* void affichage()                  */
/*************************************/
/* fonction de rappel pour l'affichage */
/*************************************/

void affichage()
{
  glClear(masqueClear);

  /* Positionnement de l'observateur (ou de l'objet) */
  glLoadIdentity();
  gluLookAt(0.0,0.0,distance,0.0,0.0,0.0,0.0,1.0,0.0);
  glRotatef(angley,1.0,0.0,0.0);
  glRotatef(anglex,0.0,1.0,0.0);

  /* Description de l'obet */
  glBindTexture(GL_TEXTURE_2D,IdTex[0]);
  glBegin(GL_POLYGON);
  glNormal3f(0.0,0.0,1.0);
  glTexCoord2f(0.0,1.0);   glVertex3f(-0.5, 0.5, 0.5);
  glTexCoord2f(0.0,0.0);   glVertex3f(-0.5,-0.5, 0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f( 0.5,-0.5, 0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f( 0.5, 0.5, 0.5);
  glEnd();


  glBindTexture(GL_TEXTURE_2D,IdTex[1]);
  glBegin(GL_POLYGON);
  glNormal3f(1.0,0.0,0.0);
  glTexCoord2f(0.0,1.0);   glVertex3f( 0.5, 0.5, 0.5);
  glTexCoord2f(0.0,0.0);   glVertex3f( 0.5,-0.5, 0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f( 0.5,-0.5,-0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f( 0.5, 0.5,-0.5);
  glEnd();

  glBindTexture(GL_TEXTURE_2D,IdTex[0]);
  glBegin(GL_POLYGON);
  glNormal3f(0.0,0.0,-1.0);
  glTexCoord2f(0.0,1.0);   glVertex3f( 0.5, 0.5,-0.5);
```

```
  glTexCoord2f(0.0,0.0);   glVertex3f( 0.5,-0.5,-0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f(-0.5,-0.5,-0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f(-0.5, 0.5,-0.5);
  glEnd();

  glBindTexture(GL_TEXTURE_2D,IdTex[1]);
  glBegin(GL_POLYGON);
  glNormal3f(-1.0,0.0,0.0);
  glTexCoord2f(0.0,1.0);   glVertex3f(-0.5, 0.5,-0.5);
  glTexCoord2f(0.0,0.0);   glVertex3f(-0.5,-0.5,-0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f(-0.5,-0.5, 0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f(-0.5, 0.5, 0.5);
  glEnd();

  glBindTexture(GL_TEXTURE_2D,IdTex[0]);
  glBegin(GL_POLYGON);
  glNormal3f(0.0,1.0,0.0);
  glTexCoord2f(0.0,1.0);   glVertex3f(-0.5, 0.5,-0.5);
  glTexCoord2f(0.0,0.0);   glVertex3f(-0.5, 0.5, 0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f( 0.5, 0.5, 0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f( 0.5, 0.5,-0.5);
  glEnd();

  glBindTexture(GL_TEXTURE_2D,IdTex[1]);
  glBegin(GL_POLYGON);
  glNormal3f(0.0,-1.0,0.0);
  glTexCoord2f(0.0,0.0);   glVertex3f(-0.5,-0.5,-0.5);
  glTexCoord2f(0.0,1.0);   glVertex3f(-0.5,-0.5, 0.5);
  glTexCoord2f(1.0,1.0);   glVertex3f( 0.5,-0.5, 0.5);
  glTexCoord2f(1.0,0.0);   glVertex3f( 0.5,-0.5,-0.5);
  glEnd();

  /*  echange de tampon (double buffering)*/
  glutSwapBuffers();
}



/************************************************/
/* void clavier(unsigned char touche,int x,int y) */
/************************************************/
/* fonction de rappel clavier                    */
/************************************************/

void clavier(unsigned char touche,int x,int y)
{
  switch (touche) {
  case 'p': /* bascule tampon de profondeur */
    prof=1-prof;
    if (prof) {
      masqueClear=GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT;
      glEnable(GL_DEPTH_TEST);
    }
    else {
      masqueClear=GL_COLOR_BUFFER_BIT;
      glDisable(GL_DEPTH_TEST);
    }
    glutPostRedisplay();
    break;

  case 'b': /* bascule brouillard */
    brouillard=1-brouillard;
    if (brouillard)
      glEnable(GL_FOG);
    else
      glDisable(GL_FOG);
    glutPostRedisplay();
    break;
```

```
  case 'e' : /* bascule eclairage */
    eclairage=1-eclairage;
    if (eclairage)
      glEnable(GL_LIGHTING);
    else
      glDisable(GL_LIGHTING);
    glutPostRedisplay();
    break;

  case 's' : /* bascule séparation eclairage speculaire */
    separe=1-separe;
    if (separe)
      glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SEPARATE_SPECULAR_COLOR);
    else
      glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SINGLE_COLOR);
    glutPostRedisplay();
    break;

  case 27: /* touche escape pour quitter */
    exit(0);
  default :
  }
}



/*********************************************************/
/* void souris(int bouton, int etat,int x,int y)         */
/*********************************************************/
/* fonction de rappel pour l'appui sur un bouton de souris */
/*********************************************************/

void souris(int bouton, int etat,int x,int y)
{
  if (bouton == GLUT_LEFT_BUTTON &etat == GLUT_DOWN) {
    b_gauche = 1;
    xprec = x;
    yprec=y;
  }
  if (bouton == GLUT_LEFT_BUTTON &etat == GLUT_UP)
    b_gauche=0;

  if (bouton == GLUT_RIGHT_BUTTON &etat == GLUT_DOWN) {
    b_droit = 1;
    yprec=y;
  }
  if (bouton == GLUT_RIGHT_BUTTON &etat == GLUT_UP)
    b_droit=0;
}



/***************************************************/
/* void mouvSouris(int x,int y)                    */
/***************************************************/
/* fonction de rappel pour les mouvement de souris */
/***************************************************/

void mouvSouris(int x,int y)
{
  /* si le bouton gauche est presse */
  if (b_gauche) {
    anglex=anglex+(x-xprec);
    angley=angley+(y-yprec);
    glutPostRedisplay();
    xprec=x;
    yprec=y;
```

```
  }

  /* si le bouton gauche est presse */
  if (b_droit) {
    distance+=((float)(y-yprec))/10.0;
    if (distance<1.0)
      distance=1.0;
    if (distance>15.0)
      distance=15.0;
    glutPostRedisplay();
    yprec=y;
  }
}



/*****************************************************************/
/* void redim(int l,int h)                                       */
/*****************************************************************/
/* fonction de rappel pour les redimensionnements de la fenetre */
/*****************************************************************/

void redim(int l,int h)
{
  if (l<h)
    glViewport(0,(h-l)/2,l,l);
  else
    glViewport((l-h)/2,0,h,h);
}



/*************************************************/
/* void inactif                                  */
/*************************************************/
/* fonction de rappel pour pour l'inactivité (idle) */
/*************************************************/

void inactif()
{
  /* increment du décalage */
  decalage+=0.1;
  if (decalage>2*PI)
    decalage-=2*PI;
  /* rechargement de la texture */
  chargeTextureProc(IdTex[1]);
  glutPostRedisplay();

}



/*************************************************/
/* void chargeTextureTiff(char *fichier,int numtex) */
/*************************************************/
/* chargement de l'image tif 'fichier' et placement */
/* dans la texture de numero 'numtex'               */
/*************************************************/

void chargeTextureTiff(char *fichier,int numtex)
{
  unsigned char image[256][256][3];
  uint32 l, h;
  int i,j;
  size_t npixels;
  uint32* raster;

  /* chargement de l'image TIF */
```

```
  TIFF* tif = TIFFOpen(fichier, "r");
  if (tif) {
    TIFFGetField(tif, TIFFTAG_IMAGEWIDTH,
    TIFFGetField(tif, TIFFTAG_IMAGELENGTH,
    npixels = l * h;
    raster = (uint32*) _TIFFmalloc(npixels * sizeof (uint32));
    if (raster != NULL) {
      /* lecture de l'image */
      if (TIFFReadRGBAImage(tif, l, h, raster, 1)) {
        /* transfert de l'image vers le tableau 'image' */
        for (i=0;i<256;i++)
          for (j=0;j<256;j++) {
            image[i][j][0]=((unsigned char *)raster)[i*256*4+j*4+0];
            image[i][j][1]=((unsigned char *)raster)[i*256*4+j*4+1];
            image[i][j][2]=((unsigned char *)raster)[i*256*4+j*4+2];
          }
      }
      else {
        printf("erreur de chargement du fichier %s\n",fichier);
        exit(0);
      }
      _TIFFfree(raster);
    }
    TIFFClose(tif);

    /*      paramétrage de la texture */
    glBindTexture(GL_TEXTURE_2D,numtex);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,256,256,0,
                 GL_RGB,GL_UNSIGNED_BYTE,image);
  }

}

/*************************************/
/* void chargeTextureProc(int numtex) */
/*************************************/
/* Création de la texture procedurale */
/* de numero 'numtex'                 */
/*************************************/

void chargeTextureProc(int numtex)
{
  unsigned char image[256][256][3];
  int i,j;
  int a;

  /* calcule de l'image */
  for (i=0;i<256;i++)
    for (j=0;j<256;j++) {
      a=fonctionTexture(i,j);
      image[i][j][0]=a;
      image[i][j][1]=128;
      image[i][j][2]=128;
    }

  /* Paramètrage de la texture */
  glBindTexture(GL_TEXTURE_2D,numtex);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,256,256,0,
               GL_RGB,GL_UNSIGNED_BYTE,image);
}
```

```
/*****************************************************/
/* int fonctionTexture(int x,int y)                  */
/*****************************************************/
/* Calcule et renvoie la valeur de la fonction       */
/* utilisee pour la texture procedurale au point x,y */
/*****************************************************/

int fonctionTexture(int x,int y)
{
  float dx=(128.0-(float)x)/255.0*40.0;
  float dy=(128.0-(float)y)/255.0*40.0;
  float a=cos(sqrt(dx*dx+dy*dy)+decalage);
  return (int)((a+1.0)/2.0*255);

}
```