Introduction :

Après avoir vu la derniere fois qu'il était relativement simple de générer un terrain à partir d'une image en niveaux de gris, nous allons essayer aujourd'hui de donner à ce terrain un aspect visuel relativement réaliste.

Le didacticiel d'aujourd'hui sera découpé en deux parties distinctes. Dans un premier temps, nous allons revenir sur les aspects théoriques et les algorithmes qu'OpenGL met en oeuvre pour l'éclairage des scènes. Ensuite, nous verrons comment implémenter dans notre générateur de terrain un éclairage correct.

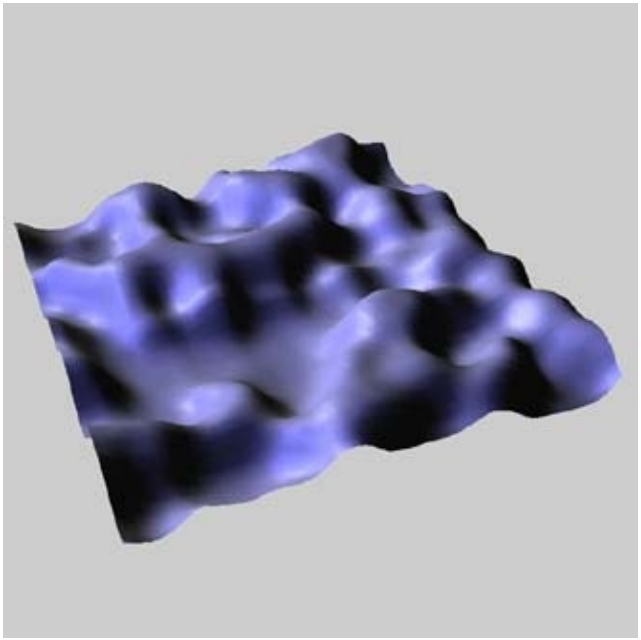

Figure 1 : Un terrain correctement éclairé

Les algorithmes de remplissage (shading)

Nous avons vu que lorsque vous décrivez un polygone avec OpenGL, vous avez la possibilité de spécifier la méthode utilisée pour colorer la primitive grâce à la commande glShadeModel() qui peut prendre comme valeur GL_FLAT (mode à plat) ou GL_SMOOTH (mode dégradé) :

− Si vous utilisez le mode à plat GL_FLAT, le polygone sera rempli de façon uniforme. Même si vous affectez une couleur différente à chaque sommet, le polygone sera rempli avec la couleur du premier sommet du polygone. Ce mode de dessin peut vous sembler désuet, mais il est utile si vous dessinez une scène en fil de fer ou si vous souhaitez créer un effet 'cartoon' en superposant un rendu à plat et un rendu fil de fer.

− Avec le mode dégradé GL_FLAT, le remplissage du polygone se fait par interpolation des couleurs des sommets (algorithme de Gouraud). Derrière ces termes techniques se cache en fait un principe simple : le remplissage du polygone se fait par un dégradé de couleurs.

L'eclairage des objets

Lorsque nous avons étudié la question de l'éclairage, nous avons vu qu'éclairer une scène consistait à modifier la couleur des objets en fonction des paramètres d'éclairage et de matériau. Pour être plus précis, l'éclairage modifie la couleur des sommets constituant les objets. L'algorithme de shading propage ensuite ces modifications de couleur à l'ensemble des objets. L'avantage de ce principe est qu'il est relativement peu coûteux. Cependant, il est source d'erreur.

Prenons un cas simple, illustré sur les figures 2 et 3, qui représentent un plan éclairé par un spot : sur la figure 2, le plan est constitué d'un unique polygone. Sur la figure 3, il est représenté par un maillage de polygones. Compte tenu des explications que je viens de vous fournir concernant l'éclairage et le shading, vous devriez avoir deviné l'explication de ce phénomène : dans le cas de la figure 2, l'éclairage n'est calculé qu'aux quatre coins du plan et ce dernier est rempli avec l'algorithme de dégradé. Comme la tache spéculaire créée par le spot ne 'touche' aucun sommets, elle n'apparaît pas lors du rendu de la scène. Bien entendu, ce problème n'apparaît avec le maillage de la figure 3 puisque la tâche spéculaire éclaire certains vertice du maillage.

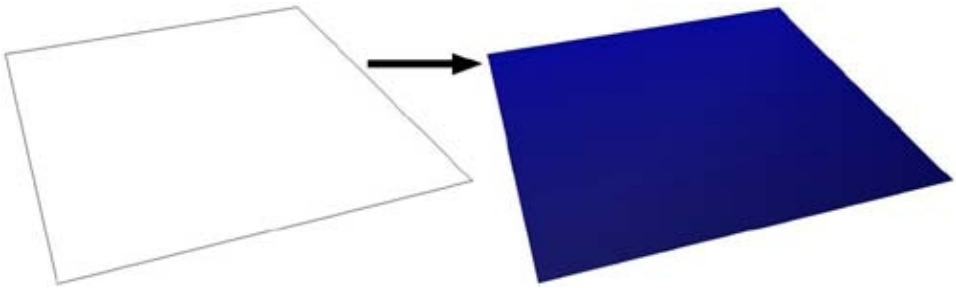

Figure 2 : Eclairage d'un plan constitué d'un unique polygone

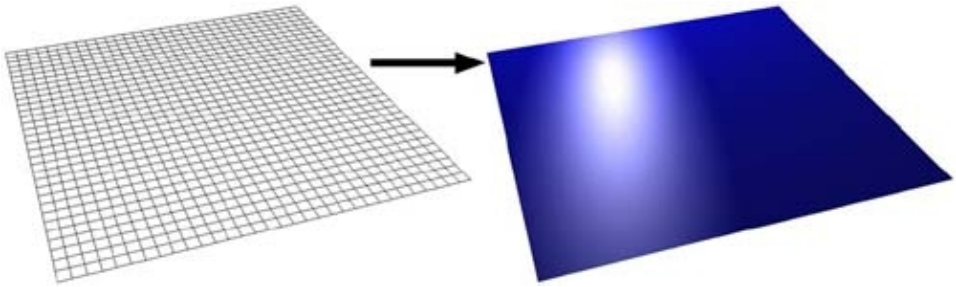

Figure 3 : Eclairage d'un plan défini par un maillage

Les normales

Nous avons déjà parlé à plusieurs reprises des normales. Nous avons vu qu'un vecteur normal est affecté à chaque sommet par un système de variables d'état : on définit le vecteur normal courant grâce à glNormal(), et lors de la description d'un polygone, les sommets se voient affectés comme normale le vecteur normal courant. Pour obtenir un rendu correct, le vecteur normal doit être perpendiculaire à la surface de l'objet, et sa longueur doit valoir 1.

Jusqu'à présent, nous avons toujours évité les problèmes liés aux normales, soit en utilisant des fonctions glut qui se chargent elles−mêmes de générer des normales correctes, soit en travaillant avec des objets 3D pour lesquels les normales sont évidentes (notre bon vieux cube par exemple). Aujourd'hui, il en va tout autrement : les normales de

notre terrain ne sont pas simples à calculer !

Calcul des normales

Il est en général difficile de calculer la normale d'un objet en un point donné. Le problème vient du fait que calculer un vecteur normal à un point n'a pas de sens, on calcule un vecteur normal à une surface en un point donné. Et nous n'avons pas une description parfaite de la surface, nous n'avons qu'une approximation polygonale. En fait, cela nous arrange : nos polygones sont plans (nous nous en sommes assurés la derniere fois en utilisant le triangle comme primitive de base), et donc, chaque point d'un polygone a le même vecteur normal ! Donc finalement, pour calculer la normale en un sommet de la surface, il suffit de calculer une normale pour chacun des polygones auquel appartient le sommet, puis de faire la moyenne de tous ces vecteurs pour obtenir notre normale tant désirée !!!

Reste maintenant à savoir comment calculer un vecteur normal à un polygone : rien de plus simple avec le produit vectoriel.

Le produit vectoriel

Le produit vectoriel est un opérateur mathématique relativement utile en infographie, tout comme son compère le produit scalaire. Le principe du produit vectoriel est simple : le produit vectoriel de deux vecteurs u et v non colinéaires renvoie un vecteur perpendiculaire au plan défini par les vecteurs u et v. Si u et v ont pour composantes respectives (x1,y1,z1) et (x2,y2,z2), alors le produit vectoriel de u et v, noté u^v a pour composantes : (y1*z2−y2*z1 , z1*x2−z2*x1 , x1*y2−x2*y1). Le produit vectoriel possède certaines particularités : une des plus importantes pour nous est que u^v = − (v^u). Il faut donc faire attention a l'ordre dans lequel vous effectuez le produit scalaire, au risque de vous retrouver avec des normales inversées.

Connaissant les sommets d'un polygone, on obtient facilement deux vecteurs appartenant au plan dans lequel est inscrit le polygone : il suffit de prendre 3 sommets P1,P2,P3 et de considérer les vecteurs P1P2 et P1P3. Attention, il faut vous assurer que les deux vecteurs ne sont pas colinéaires et qu'aucun de ces deux vecteurs n'est nul, sinon le produit vectoriel vous renverra un vecteur nul. Dans le cas de notre terrain nous n'auront pas de problème, puisque nous utilisons un maillage régulier. Pour les moins matheux d'entre vous, je me permet de signaler que le vecteur P1 P2 définit par les deux points P1 (x1,y1,z1) et P2(x2,y2,z2) a pour composantes le triplet (x2−x1 , y2−y1 , z2−z1).

Moyennes de normales

Nous savons maintenant calculer un vecteur normal à un polygone. Il ne reste plus qu'à voir comment effectuer une moyenne de normales. A priori, le principe semble simple : il suffit de faire une moyenne composante par composante : ainsi la moyenne de trois vecteurs u,v,w de composantes respectives (x1,y1,z1), (x2,y2,z2) et (x3,y3,z3) vaut ((x1+x2+x3)/3,(y1+y2+y3)/3,(z1+z2+z3)/3). Ce résultat est juste, mais il n'a de sens que si au départ, les trois vecteurs ont la même longueur, ce qui à priori n'est pas le cas. Avant de calculer la moyenne, il convient de diviser chaque composante du vecteur par la longueur de ce dernier de façon à obtenir un vecteur de longueur 1. Un vecteur de longueur 1 est dit 'normé'. Vous prendrez bien soin de distinguer ce terme du terme 'normal', qui signifie perpendiculaire. L'opération qui consiste à diviser les composantes d'un vecteur par sa longueur se nomme normalisation. Un vecteur et le vecteur résultant de sa normalisation sont colinéaires. En notant sqrt() l'opérateur 'racine carrée', la longueur d'un vecteur u de composantes (x,y,z) vaut n=sqrt(x*x+y*y+z*z), et le résultat de la normalisation de u est le vecteur de composantes (x/n,y/n,z/n).

Recapitulatif

Ca y est, nous en avons terminé avec les maths. Voici un petit récapitulatif des opérations nécessaires pour obtenir un vecteur normal à un objet en un sommet :

− Calculer un vecteur normal à chacun des polygones dont le sommet fait partie
− Normer chacun des vecteurs normaux
− Calculer la moyenne des vecteurs normaux normés (Raymond Devos ne ferait pas mieux).

Attention ce n'est pas fini ! Nous avons vu il y a quelques temps de cela que pour que l'éclairage soit correct, il fallait que le vecteur normal soit de longueur 1. Il faut donc normer le résultat de la moyenne.

Implementation

Intéressons nous maintenant aux aspects pratiques de la question et regardons comment implémenter la gestion de l'éclairage dans le programme du precedent didacticiel. Le changement majeur est l'utilisation d'un tableau de type vertex pour calculer le terrain. Le type vertex contient 6 composantes : les cordonnées cartésiennes du sommet (x,y,z) ainsi que les coordonnées de la normale au terrain en ce sommet (nx,ny,nz). Vous remarquerez que le tableau utilisé est mono dimensionnel, car il est plus facile de le déclarer dynamiquement. Le vertex (i,j) du terrain est stocké dans T[i*(largeur du tableau)+j].

Le coeur du code est bien sûr la fonction creeTerrain() qui génère la liste d'affichage du terrain. Dans un premier temps, on alloue de la mémoire pour le tableau de 'vertex' T. Je vous rappelle que le programme permet de faire varier la densité du maillage, ce qui explique l'allocation dynamique de mémoire. On initialise également les normales des vertex à zéro, car nous allons utiliser une méthode incrémentale pour calculer ces normales.

```
T=(vertex *)malloc((nbSubdiv+1)*(nbSubdiv+1)*sizeof(vertex));

/* Initialisation des normales */
for (i=0;i<=nbSubdiv;i++)
  for (j=0;j<=nbSubdiv;j++) {
    T[i*(nbSubdiv+1)+j].nx=0.0;
    T[i*(nbSubdiv+1)+j].ny=0.0;
    T[i*(nbSubdiv+1)+j].nz=0.0;
  }
```

La méthode utilisée pour calculer la position des sommets et celle que nous avons déjà utilisée :

```
for (i=0;i<=nbSubdiv;i++)
  for (j=0;j<=nbSubdiv;j++) {
    T[i*(nbSubdiv+1)+j].x=-1.0+i*pas;
    T[i*(nbSubdiv+1)+j].y=-1.0+j*pas;
    T[i*(nbSubdiv+1)+j].z=elevation(i,j);
  }
```

Ensuite on passe au calcul des normales. Et nous allons nous permettre une petite astuce. Dans la méthode que je vous ai présentée, nous avons vu que pour chaque sommet, il fallait calculer la moyenne des normales de tous les polygones auxquels le sommet appartient. Mais tous les sommets ne font pas partie d'un même nombre de polygones. Aussi nous allons plutôt parcourir le maillage par patch carré (c'est-à-dire un couple de triangles). Pour chaque triangle, on calcule un vecteur normal (et normé !) qu'on ajoute à la composante normale (nx,ny,nz) des sommets qui composent le triangle :

```
for (i=0;i<=nbSubdiv;i++)
  for (j=0;j<=nbSubdiv;j++) {
    T[i*(nbSubdiv+1)+j].x=-1.0+i*pas;
    T[i*(nbSubdiv+1)+j].y=-1.0+j*pas;
    T[i*(nbSubdiv+1)+j].z=elevation(i,j);
  }

for (i=0;i<nbSubdiv;i++)
  for (j=0;j<nbSubdiv;j++) {
    P1=&T[i*(nbSubdiv+1)+j];
    P2=&T[(i+1)*(nbSubdiv+1)+j];
    P3=&T[(i+1)*(nbSubdiv+1)+j+1];
    P4=&T[i*(nbSubdiv+1)+j+1];

    V1.x=P2->x-P1->x; V1.y=P2->y-P1->y; V1.z=P2->z-P1->z;
    V2.x=P3->x-P1->x; V2.y=P3->y-P1->y; V2.z=P3->z-P1->z;
```

```
    V3.x=P4->x-P1->x; V3.y=P4->y-P1->y; V3.z=P4->z-P1->z;

    incx=V2.y*V1.z-V1.y*V2.z;
    incy=V2.z*V1.x-V1.z*V2.x;
    incz=V2.x*V1.y-V1.x*V2.y;
    norme=sqrt(incx*incx+incy*incy+incz*incz);
    incx/=norme; incy/=norme; incz/=norme;
    P1->nx-=incx; P1->ny-=incy; P1->nz-=incz;
    P2->nx-=incx; P2->ny-=incy; P2->nz-=incz;
    P3->nx-=incx; P3->ny-=incy; P3->nz-=incz;

    incx=V3.y*V2.z-V2.y*V3.z;
    incy=V3.z*V2.x-V2.z*V3.x;
    incz=V3.x-V2.y-V2.x*V3.y;
    P1->nx-=incx; P1->ny-=incy; P1->nz-=incz;
    P3->nx-=incx; P3->ny-=incy; P3->nz-=incz;
    P4->nx-=incx; P4->ny-=incy; P4->nz-=incz;
  }
```

Comme je vous l'ai expliqué précédemment, il ne faut pas oublier de normer les vecteurs normaux moyens :

```
for (i=0;i<=nbSubdiv;i++)
  for (j=0;j<=nbSubdiv;j++) {
    P1=&T[i*(nbSubdiv+1)+j];
    norme=sqrt(P1->nx*P1->nx+P1->ny*P1->ny+P1->nz*P1->nz);
    P1->nx/=norme;
    P1->ny/=norme;
    P1->nz/=norme;
  }
```

L'essentiel du travail est fait : il ne reste plus qu'à générer la liste d'affichage terrain en utilisant le tableau que nous venons de créer :

```
glNewList(terrain,GL_COMPILE);
glColor3f(0.0,0.0,0.0);
glLineWidth(1.0);
for (i=0;i<nbSubdiv;i++)
  for (j=0;j<nbSubdiv;j++) {
    glBegin(GL_TRIANGLES);
    /* triangle 1 */
    glEdgeFlag(TRUE);
    drawVertex(i,j,T);
    drawVertex(i+1,j,T);
    if (!areteTransv)
glEdgeFlag(FALSE);
    drawVertex(i+1,j+1,T);
    /*triangle 2 */
    drawVertex(i,j,T);
    if (!areteTransv)
glEdgeFlag(TRUE);
    drawVertex(i+1,j+1,T);
    drawVertex(i,j+1,T);
    glEnd();
  }
glEndList();
```

La fonction drawVertex n'est utilisée que dans un but de lisibilité du code, elle se contente de déclarer un sommet et la normale qui lui est associée :

```
void drawVertex(int i,int j,vertex *T)
{
  glNormal3fv(&(T[i*(nbSubdiv+1)+j].nx));
```

```
  glVertex3fv(&(T[i*(nbSubdiv+1)+j].x));
}
```

Vous noterez que le programme vous propose quelques fonctionnalités nouvelles comme l'affichage des normales et des lampes. Ces fonctionnalités ne présentent aucune difficulté, et je vous laisse donc les découvrir par vous même.

Conclusion

Nous voilà arrivés au terme de ces deux didacticiels concernant le rendu de terrain. Bien entendu, la méthode que nous avons utilisée n'est pas la seule qui existe. D'autres techniques, telles que les fractales, sont plus appropriées dans certaines situations (par exemple si vous souhaitez obtenir un terrain aléatoire). Internet fourmille de documents relatifs à la génération de terrain. Lancez une recherche sur un moteur tel que Google et vous obtiendrez en retour de nombreuses adresses utiles. La prochaine fois, nous aborderons la question de l'intégration d'OpenGL dans une interface GTK avec la bibliothèque GtkGLArea. En effet, la bibliothèque Glut est bien pensée et facile d'emploi, mais il faut reconnaître qu'elle est quand même relativement limitée au niveau de ses possibilités.

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.

Code Source

```
/*************************************************************/
/*                        terrain2.c                         */
/*************************************************************/
/* Generation de terrain correctement illumine a partir      */
/* d'une image JPEG                                          */
/*************************************************************/


#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <jpeglib.h>
#include <jerror.h>
```

```
#include <math.h>

#define NB_SUBDIV_INIT 32
#define NB_SUBDIV_MAX 64
#define ECHELLE_VERT_INIT 0.1
#define ECHELLE_VERT_MAX 1.0
#define ECHELLE_NORMALES 0.1
#define DISTANCE_INIT 4.0
#define DISTANCE_MAX 15.0

/* Definition du type sommet */
typedef struct {
  float x;
  float y;
  float z;
  float nx;
  float ny;
  float nz;
  char nb;
} vertex;

/* Variables globales */

unsigned char image[256][256]; /* l'image du terrain */
unsigned char afficheRepere=TRUE; /* Affichage du repere */
unsigned char faceArriere=FALSE; /* Affichage des faces arrieres de polygones */
unsigned char areteTransv=FALSE; /* Affichage de l'arete transversale */
unsigned char afficheNormales=FALSE; /* Affichage des normales */
unsigned char afficheLampes=TRUE; /* Affichage des lampes */
unsigned char modePlein=TRUE; /* Affichage en mode plein ou fil de fer */
int repere=0,terrain=0,normales=0,lampes=0; /* Identifiants des listes d'affichage */
int nbSubdiv=NB_SUBDIV_INIT; /* Nombre de subdivisions du maillage */
float echelleVert=ECHELLE_VERT_INIT; /* echelle verticale du relief */
char b_gauche=0,b_droit=0; /* bouton de souris presse ? */
int theta=-30,phi=300; /* Position de l'observateur */
int xprec,yprec; /* sauvegarde de la position de la souris */
float distance=DISTANCE_INIT; /* distance de l'observateur a l'origine */

/* 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 init();
void affichage();
void clavier(unsigned char touche,int x,int y);
void souris(int bouton,int etat,int x,int y);
void mouvement(int x,int y);
void redim(int l,int h);
void creeRepere();
void creeTerrain();
void creeNormales(vertex *T);
void creeLampes();
void drawVertex(int i,int j,vertex *T);
float elevation(int i,int j);
void loadJpegImage(char *filename);
```

```
int main(int argc,char **argv)
{
  /* Initialisation de glut */
  glutInit(
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(500,500);
  glutCreateWindow(argv[0]);

  /*Initialisation d'OpenGL */
  init();
  loadJpegImage(argv[1]);

  /* Creation des objets */
  creeRepere();
  creeLampes();
  creeTerrain();

  glutMainLoop();
  return 0;
}



/* Initialisation d'openGL */
void init()
{
  glClearColor(0.8,0.8,0.8,1.0);
  glEnable(GL_DEPTH_TEST);
  if (modePlein)
    glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
  else
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
  glCullFace(GL_BACK);
  if (faceArriere)
    glDisable(GL_CULL_FACE);
  else
    glEnable(GL_CULL_FACE);

  /* Mise en place de la perspective */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,1.0,0.1,20.0);
  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);

  /* Mise en place des fonction de rappel */
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);
```

```
  glutMouseFunc(souris);
  glutMotionFunc(mouvement);
  glutReshapeFunc(redim);
}



/* Fonction de rappel pour l'affichage */
void affichage()
{
  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,0.0,0.0,1.0);
  glLightfv(GL_LIGHT0,GL_POSITION,L0pos);
  glLightfv(GL_LIGHT1,GL_POSITION,L1pos);
  if (afficheLampes)
    glCallList(lampes);
  glCallList(terrain);
  if (afficheNormales)
    glCallList(normales);
  if (afficheRepere)
    glCallList(repere);
  glutSwapBuffers();
}



/* Fonction de rappel pour le clavier */
void clavier(unsigned char touche,int x,int y)
{
  switch (touche) {
  case 27: /* touche 'ESC' pour quitter */
    exit(0);
  case '+': /* augmentation du nombre de subdivisions */
    nbSubdiv++;
    if (nbSubdiv>NB_SUBDIV_MAX)
      nbSubdiv=NB_SUBDIV_MAX;
    creeTerrain();
    glutPostRedisplay();
    break;
  case '-': /* diminution du nombre de subdivisions*/
    nbSubdiv--;
    if (nbSubdiv<1)
      nbSubdiv=1;
    creeTerrain();
    glutPostRedisplay();
    break;
  case 'p': /* augmentation de l'echelle verticale */
    echelleVert+=0.02;
    if (echelleVert>ECHELLE_VERT_MAX)
      echelleVert=ECHELLE_VERT_MAX;
    creeTerrain();
    glutPostRedisplay();
    break;
  case 'o': /* diminution de l'echelle verticale */
    echelleVert-=0.02;
    if (echelleVert<-ECHELLE_VERT_MAX)
      echelleVert=-ECHELLE_VERT_MAX;
    creeTerrain();
    glutPostRedisplay();
    break;
```

```
  case 'f':
    modePlein=1-modePlein;
    if (modePlein)
      glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
    else
      glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    glutPostRedisplay();
    break;
  case 'n': /* Affichage des normales ON/OFF */
    afficheNormales=1-afficheNormales;
    glutPostRedisplay();
    break;

  case 'r': /* Affichage du repere ON/OFF */
    afficheRepere=1-afficheRepere;
    glutPostRedisplay();
    break;
  case 'l' : /* Affichage des lampes ON/OFF */
    afficheLampes=1-afficheLampes;
    glutPostRedisplay();
    break;
  case 'c': /* affichage des faces arrieres ON/OFF */
    faceArriere=1-faceArriere;
  if (faceArriere)
    glDisable(GL_CULL_FACE);
  else
    glEnable(GL_CULL_FACE);
  glutPostRedisplay();
  break;
  case 't': /* affichage des aretes transversales */
    areteTransv=1-areteTransv;
    creeTerrain();
    glutPostRedisplay();
    break;
  }
}



/* fonction de rappel pour l'appui sur les boutons 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;
}



/* Fonction de rappel pour les mouvements de souris */
void mouvement(int x,int y)
```

```
{
  /* si le bouton gauche est presse */
  if (b_gauche) {
    theta+=x-xprec;
    if (theta>=360)
      while (theta>=360)
        theta-=360;
    phi+=y-yprec;
    if (phi<0)
      while (phi<0)
        phi+=360;
    xprec=x;
    yprec=y;
    glutPostRedisplay();
  }

  /* si le bouton droit est presse */
  if (b_droit) {
    distance+=((float)(y-yprec))/50.0;
    if (distance<1.0)
      distance=1.0;
    if (distance>DISTANCE_MAX)
      distance=DISTANCE_MAX;
    glutPostRedisplay();
    yprec=y;
  }
}



/* Fonction de rappel pour le redimensionnement de 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);
}



/* Creation de la liste d'affichage pour le repere */
void creeRepere()
{
   repere=glGenLists(1);
  glNewList(repere,GL_COMPILE);
  glDisable(GL_LIGHTING);
  glLineWidth(2.0);
    glBegin(GL_LINES);
      glColor3f(1.0,0.0,0.0);
      glVertex3f(0.0,0.0,0.0);
      glVertex3f(0.3,0.0,0.0);
      glColor3f(0.0,1.0,0.0);
      glVertex3f(0.0,0.0,0.0);
      glVertex3f(0.0,0.3,0.0);
      glColor3f(0.0,0.0,1.0);
      glVertex3f(0.0,0.0,0.0);
      glVertex3f(0.0,0.0,0.3);
    glEnd();
    glEnable(GL_LIGHTING);
  glEndList();
}
```

```
/* Creation de la liste d'afffichage pour le terrain */
void creeTerrain()
{
  int i,j;
  float pas=2.0/nbSubdiv;
  vertex *T;
  vertex *P1,*P2,*P3,*P4;
  vertex V1,V2,V3;
  float incx,incy,incz,norme;
  T=(vertex *)malloc((nbSubdiv+1)*(nbSubdiv+1)*sizeof(vertex));

  /* Initialisation des normales */
  for (i=0;i<=nbSubdiv;i++)
    for (j=0;j<=nbSubdiv;j++) {
      T[i*(nbSubdiv+1)+j].nx=0.0;
      T[i*(nbSubdiv+1)+j].ny=0.0;
      T[i*(nbSubdiv+1)+j].nz=0.0;
    }
  /* remplissage du tableau Tvertex */
  for (i=0;i<=nbSubdiv;i++)
    for (j=0;j<=nbSubdiv;j++) {
      T[i*(nbSubdiv+1)+j].x=-1.0+i*pas;
      T[i*(nbSubdiv+1)+j].y=-1.0+j*pas;
      T[i*(nbSubdiv+1)+j].z=elevation(i,j);
    }
  for (i=0;i<nbSubdiv;i++)
    for (j=0;j<nbSubdiv;j++) {
      P1=+j];
      P2=v+1)+j];
      P3=v+1)+j+1];
      P4=+j+1];

      V1.x=P2->x-P1->x; V1.y=P2->y-P1->y; V1.z=P2->z-P1->z;
      V2.x=P3->x-P1->x; V2.y=P3->y-P1->y; V2.z=P3->z-P1->z;
      V3.x=P4->x-P1->x; V3.y=P4->y-P1->y; V3.z=P4->z-P1->z;

      incx=V2.y*V1.z-V1.y*V2.z;
      incy=V2.z*V1.x-V1.z*V2.x;
      incz=V2.x*V1.y-V1.x*V2.y;
      norme=sqrt(incx*incx+incy*incy+incz*incz);
      incx/=norme; incy/=norme; incz/=norme;
      P1->nx-=incx; P1->ny-=incy; P1->nz-=incz;
      P2->nx-=incx; P2->ny-=incy; P2->nz-=incz;
      P3->nx-=incx; P3->ny-=incy; P3->nz-=incz;


      incx=V3.y*V2.z-V2.y*V3.z;
      incy=V3.z*V2.x-V2.z*V3.x;
      incz=V3.x-V2.y-V2.x*V3.y;
      P1->nx-=incx; P1->ny-=incy; P1->nz-=incz;
      P3->nx-=incx; P3->ny-=incy; P3->nz-=incz;
      P4->nx-=incx; P4->ny-=incy; P4->nz-=incz;
    }

  /* normalisation des normales */
  for (i=0;i<=nbSubdiv;i++)
    for (j=0;j<=nbSubdiv;j++) {
      P1=+j];
      norme=sqrt(P1->nx*P1->nx+P1->ny*P1->ny+P1->nz*P1->nz);
      P1->nx/=norme;
      P1->ny/=norme;
```

```
      P1->nz/=norme;
    }

  /* Liste pour l'objet terrain */
  if (glIsList(terrain))
    glDeleteLists(terrain,1);
  terrain=glGenLists(1);
  glNewList(terrain,GL_COMPILE);
  glColor3f(0.0,0.0,0.0);
  glLineWidth(1.0);
  for (i=0;i<nbSubdiv;i++)
    for (j=0;j<nbSubdiv;j++) {
      glBegin(GL_TRIANGLES);
      /* triangle 1 */
      glEdgeFlag(TRUE);
      drawVertex(i,j,T);
      drawVertex(i+1,j,T);
      if (!areteTransv)
        glEdgeFlag(FALSE);
      drawVertex(i+1,j+1,T);

      /*triangle 2 */
      drawVertex(i,j,T);
      if (!areteTransv)
        glEdgeFlag(TRUE);
      drawVertex(i+1,j+1,T);
      drawVertex(i,j+1,T);
      glEnd();
    }
  glEndList();

  /* Generation de la liste d'affichage des normales*/
  creeNormales(T);

  /* Liberation de Tvertex */
  free(T);
}


/* Cree la liste d'affichage pour les normales */
void creeNormales(vertex *T)
{
  int i,j;
  vertex *P;
  if (glIsList(normales))
    glDeleteLists(normales,1);
  normales=glGenLists(1);
  glNewList(normales,GL_COMPILE);
  glDisable(GL_LIGHTING);
  glLineWidth(1.0);
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_LINES);
  for (i=0;i<=nbSubdiv;i++)
    for (j=0;j<=nbSubdiv;j++) {
      P=+j];
      glVertex3fv(x);
      glVertex3f(P->x+ECHELLE_NORMALES*P->nx,
                 P->y+ECHELLE_NORMALES*P->ny,
                 P->z+ECHELLE_NORMALES*P->nz);
    }
  glEnd();
  glEnable(GL_LIGHTING);
  glEndList();
```

```
}

/* Creation de la Liste d'affichage pour les lampes */
void creeLampes()
{
  lampes=glGenLists(1);
  glNewList(lampes,GL_COMPILE);
  glDisable(GL_LIGHTING);
  glColor3f(1.0,1.0,1.0);
  glPointSize(6.0);
  glBegin(GL_POINTS);
  glVertex3fv(L0pos);
  glVertex3fv(L1pos);
  glEnd();
  glLineWidth(1.0);
  glBegin(GL_LINES);
  glVertex3fv(L0pos);
  glVertex3f(0.0,0.0,0.0);
  glVertex3fv(L1pos);
  glVertex3f(0.0,0.0,0.0);
  glEnd();
  glEnable(GL_LIGHTING);
  glEndList();
}



/* Affiche le sommet (i,j) du maillage */
void drawVertex(int i,int j,vertex *T)
{
  glNormal3fv()+j].nx));
  glVertex3fv()+j].x));
}



/* Calcul de la hauteur d'un point */
float elevation(int i,int j)
{
  int valeur=image[(int)((float)i/nbSubdiv*255)][(int)((float)j/nbSubdiv*255)];
  return ((float)valeur/128.0−1.0)*echelleVert;
}



/* Chargement d'une image jpeg */
void loadJpegImage(char *filename)
{
  FILE *file;
  struct jpeg_decompress_struct cinfo;
  struct jpeg_error_mgr jerr;
  unsigned char *im=(unsigned char *)image,*ligne;

  cinfo.err = jpeg_std_error(
  jpeg_create_decompress(

  /* On mets en place une image par defaut si filename=NULL*/
  if (filename==NULL){
    filename=(char *)malloc(128);
    strcpy(filename,"terrain.jpg");
  }
```

```
  if (!(file=fopen(filename,"rb"))) {
    fprintf(stderr,"Erreur : impossible d'ouvrir %s\n",filename);
    exit(1);
  }
  jpeg_stdio_src(file);
  jpeg_read_header(TRUE);
  if ((cinfo.image_width!=256) || (cinfo.image_height!=256)) {
    fprintf(stderr,"Erreur : l'image doit etre de taille 256x256\n");
    exit(1);
  }
  if (cinfo.out_color_space!=JCS_GRAYSCALE){
    fprintf(stderr,"Error : l'image doit etre en niveaux de gris\n");
    exit(1);
  }
  jpeg_start_decompress(

  while (cinfo.output_scanline<256){

    ligne=im+256*cinfo.output_scanline;
    jpeg_read_scanlines(
  }

  jpeg_finish_decompress(
  jpeg_destroy_decompress(
}
```