Introduction :

Je vous propose d'aborder aujourd'hui un sujet pratique, à savoir la génération de terrains en 3D. De nombreuses techniques existent pour cela. Certaines sont basées sur les fractals, d'autres sur des fonctions de bruit. Comme nous avons vu dernièrement qu'il était facile d'exploiter des images grâce aux bibliothèques JPEG et TIFF, nous allons utiliser la technique des champs de hauteur – heightfields – qui consiste à calculer le relief du terrain à partir d'une image en niveaux de gris.

Ce sujet nous occupera pendant deux didacticiels. En effet, aujourd'hui, nous nous contenterons d'un terrain en fil de fer qui nous permettra d'aborder certains concepts d'OpenGL que nous n'avons pas encore étudiés : les listes d'affichage, les notions de faces avant et arrière d'un polygone, le masquage d'arêtes et les polygones dégénérés. La prochaine fois, nous tacherons de donner à notre terrain un aspect plus réaliste, ce qui nous contraindra à reconsidérer les délicats problèmes de normales et de coordonnées de texture.

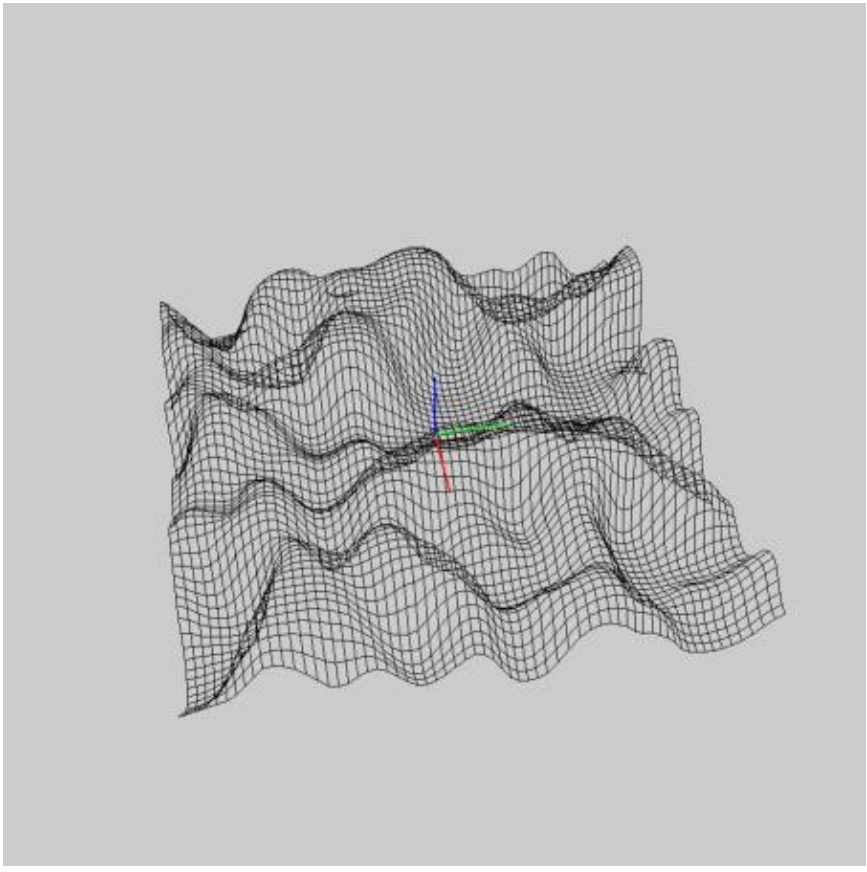

Figure 1: un terrain en OpenGL

Principe des heightfields

Le principe de champs de hauteur que nous allons mettre en place est relativement simple à comprendre. Dans un premier temps, nous allons générer un maillage carré dans le plan XY. Le maillage est illustré sur la figure 2. Ce n'est en fait ni plus ni moins qu'un ensemble de polygones carrés (on utilisera le terme de patch pour distinguer les petits carrés du maillage complet) collés les uns aux autres. L'opération se réalise très facilement avec deux boucles imbriquées. Une fois le maillage créé, il n'y a plus qu'à affecter aux sommets de chaque patch une coordonnée en Z. En fonction de la position du sommet considéré dans le maillage, on établit une correspondance avec un pixel de l'image en niveaux de gris. Si le pixel correspondant est noir, l'élévation du sommet sera minimale, alors que s'il est blanc, la hauteur sera maximale. Comme vous le voyez, ce n'est pas très compliqué.

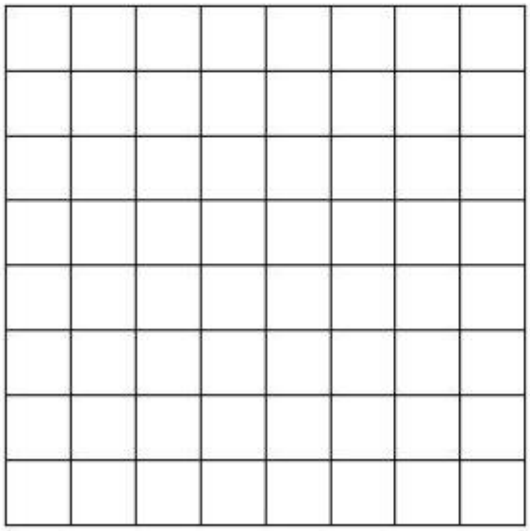

Figure 2 : le maillage vu de dessus

Des polygones qui dégénèrent

En utilisant le principe que je viens d'énoncer, nous allons être confrontés à un léger problème : nous allons créer des polygones dégénérés. OpenGL définit strictement les propriétés que doit respecter un polygone pour obtenir un rendu correct. Les polygones doivent entre autres être convexes (i.e. toute ligne joignant deux points quelconques du polygone doit être entièrement comprise dans le polygone), non auto−intersectants et tous les points doivent être dans un même plan. Or cette dernière condition n'est pas vérifiée lorsque l'on génère le terrain puisque les valeurs d'élévation des sommets sont issues d'une image, et il y a finalement peu de chance pour que les 4 sommets de chaque patch soient dans un même plan. Dans le cas d'affichage en mode fil de fer, la dégénérescence des polygones ne pose pas de problème. En revanche, si on éclaire la scène, le rendu des faces en mode plein sera faux. Autant prendre les devants et afficher quelque chose de correct en mode fil de fer.

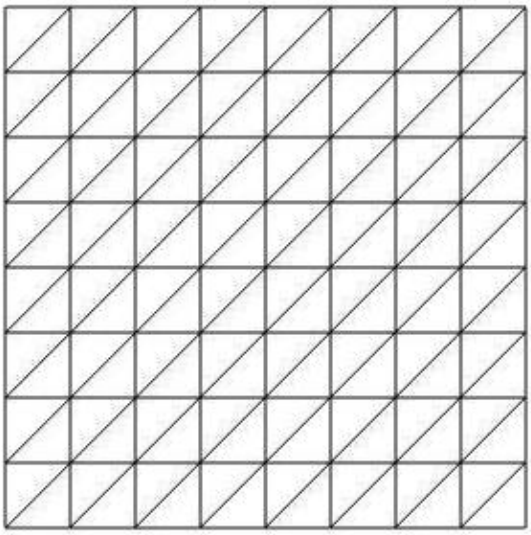

Figure 3 : le maillage avec les arêtes transversales

La solution au problème de dégénérescence est relativement simple : il suffit de décomposer chacun de nos patches en deux triangles. Notre maillage global ressemble donc à la figure 3. Reconnaissez que l'arête transversale est relativement disgracieuse. Nous allons nous en débarrasser en utilisant la fonction glEdgeFlag() d'OpenGL. Le drapeau d'arête (Edge Flag) est une bascule OpenGL qui permet de n'afficher que certaines arêtes d'un polygone. Le prototype de glEdgeFlag est le suivant :

```
void glEdgeFlag(Glboolean valeur);
```

'valeur' peut valoir TRUE ou FALSE. Les arêtes dont le premier sommet est défini lorsque le drapeau est sur TRUE seront affichées. En revanche si le drapeau est sur FALSE, l'arête n'est pas affichée. Bien sur tout ceci n'a d'intérêt que lors d'un affichage en mode fil de fer. L'exemple ci dessous montre comment dessiner un patch carré en utilisant deux patches triangulaires et la fonction glEdgeFlag() :

```
/* Dessin du triangle 1 */
glBegin(GL_POLYGON);
  glEdgeFlag(TRUE);
  glVertex3fv(P1);
  glVertex3fv(P2);
  glEdgeFlag(FALSE);
  glVertex3fv(P3);
glEnd();

/*Dessin du triangle 2 */
glBegin(GL_POLYGON);
  glVertex3fv(P1);
  glEdgeFlag(TRUE);
  glVertex3fv(P3);
  glVertex3fv(P4);
glEnd();
```

Les listes d'affichage

Les listes d'affichage sont une des rares fonctionnalités majeures d'OpenGL que nous n'avons pas encore étudiées. Les listes d'affichage sont des sortes de macros OpenGL, c'est à dire un ensemble prédéfini d'instructions OpenGL qu'on exécute en 'appelant' la liste d'affichage. L'utilité la plus évidente des listes d'affichage est l'enregistrement d'objets récurrents. Si vous devez afficher une salle de conférence, il vous suffit d'enregistrer l'objet chaise dans une liste d'affichage que vous appelez autant de fois que vous avez de chaises dans la salle. Les listes d'affichage vous permettent également de stocker des matériaux et des conditions d'éclairage. Par exemple, vous pouvez créer une liste d'affichage 'métal' contenant les instructions de glMatérial() permettant d'obtenir un rendu métallique. Pour définir un objet métallique, il vous suffit d'appeler la liste d'affichage 'métal' avant de décrire l'objet en question.

Les liste d'affichage sont conçues de façon à optimiser le temps de rendu, et donc, il est souvent plus efficace d'utiliser une liste d'affichage que de décrire les instructions 'manuellement'. Le gain dépend de l'implémentation utilisée. L'utilisation d'une liste d'affichage est relativement simple. Comme pour les textures, il faut réserver un identifiant pour une liste d'affichage. On utilise pour cela la fonction

```
Gluint GlGenLists(GLsizei n);
```

'n' définit le nombre de listes qu'on souhaite créer. La fonction renvoie un bloc de 'n' identifiants. Une fois ceci fait, on peut commencer à créer la liste d'affichage avec

```
void glNewList(Gluint liste,GLenum mode);
```

'liste' doit être un entier retourné par glGenLists(). 'mode' peut prendre les valeurs GL_COMPILE et GL_COMPILE_AND_EXECUTE. Si vous utilisez GL_COMPILE_AND_EXECUTE, les instructions que vous entrez dans la liste sont enregistrées et exécutées immédiatement. Si vous utilisez GL_COMPILE, les instructions ne sont pas exécutées lors de l'enregistrement.

Les instructions OpenGL qui suivent l'appel à glNewList() sont mémorisées dans la liste d'affichage jusqu'à l'instruction qui marque la fin de l'enregistrement :

```
void glEndList();
```

Pour exécuter une liste d'affichage, il vous suffit d'appeler

```
void glCallList(Gluint liste);
```

'liste' est bien évidemment l'identifiant de la liste. Il également possible d'effacer une liste d'affichage avec

```
void glDeleteLists(Gluint liste,GLsizei n);
```

'n' désigne le nombre de liste consécutives à effacer. Il correspond au nombre 'n' de listes d'affichage réservées avec glGenLists(). A titre d'exemple, le bout de code ci dessous crée une liste d'affichage contenant un polygone rouge :

```
int objet;
glGenLists(objet,1);
glNewList(objet,GL_COMPILE);
glColor3f(1.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex3fv(P1);
glVertex3fv(P2);
glVertex3fv(P3);
glVertex3fv(P4);
glEnd();
glEndList();
```

Il existe un certain nombre d'instructions OpenGL qui ne sont pas enregistrables dans des listes d'affichage. Vous les trouverez dans 'la bible d'OpenGL' (ouvrage [1]). Comme nous ne les avons pas utilisées, je ne les citerai pas ici.

Faces avant et arrière

Il y a quelques temps, je vous ai présenté la fonction permettant de définir le mode de remplissage des polygones, et je vous ai laissé entendre qu'il était possible de remplir différemment les faces avant et arrière d'un polygone, ce qui nécessite de pouvoir distinguer une face avant et arrière pour chaque polygone. Par défaut, si le points constituant un polygone sont affichés à l'écran dans le sens inverse des aguilles d'une montre, la face dessinée et la face avant. Il est possible de modifier ce comportement avec

```
glFrontFace(GLenum mode);
```

Si 'mode' vaut GL_CCW, les faces avant sont celles orientées dans le sens contraire des aiguilles d'une montre (CounterClockWise en anglais). Si 'mode' vaut GL_CW, les faces avant sont celle orientées dans le sens des aiguilles d'une montre (ClockWise).

Le masquage des faces arrières (backface culling)

Le masquage de face arrière est une fonctionnalité permettant de ne pas afficher les faces arrières d'un objet. Jusqu'à aujourd'hui, cette fonctionnalité ne présentait pas beaucoup d'intérêt visuel, puisque nous travaillions avec un cube fermé et nous ne voyions aucune face arrière. En réalité, OpenGL permet de masquer les faces avant, arrières ou les deux face (ce qui est relativement inutile, je vous le concède). Les faces à masquer sont définies avec

```
GlCullFace(GLenum mode);
```

GL_FRONT correspond aux faces avants, GL_BACK aux faces arrière et GL_FRONT_AND_BACK aux deux faces. Pour activer le masquage, il suffit ensuite d'appeler glEnable(GL_CULL_FACE)

La pratique

Le code source de notre générateur de terrain reprend bon nombre de routines que nous avons déjà utilisées plusieurs fois (rotation, zoom, redimensionnement) et il utilise toutes les fonctionnalités que nous avons évoquées aujourd'hui. Il utilise deux listes d'affichage : une pour afficher le terrain et une pour afficher le repère XYZ. Le masquage des faces arrière est utilisé (vous avez la possibilité de le désactiver pour voir la différence), et on peut activer ou désactiver le masquage de l'arête transversale des patches.

La routine de lecture d'image JPEG à été légèrement modifiée de façon à permettre de charger une image en niveaux de gris de taille fixe (256x256). Vous avez la possibilité de passer en argument au programme l'image de votre choix. Elle doit simplement avoir un taille de 256x256 et être en niveaux de gris. Si vous ne passez aucune image en argument, le programme charge une image par défaut Toujours concernant la routine de chargement d'image, je tiens à signaler une petite erreur commise au cours des deux derniers didacticiels : dans les routines de chargement de fichier JPEG et TIFF, il n'est pas nécessaire de réorganiser les données lues, puisqu'elles sont déjà bien ordonnées. Il suffit d'étudier l'organisation en mémoire des deux structures pour se rendre compte qu'elles sont en fait identiques. Merci à Laurent Vuibert qui a été le premier à signaler cette bourde.

La création du terrain

Le terrain est de dimension fixe : il s'agit d'un carré dans le plan XY dont les extrémités sont les point (−1,−1),(−1,1),(1,−1),(1,1). On souhaite pouvoir faire varier la densité de notre maillage avec les touches '+' et '−'. Pour représenter la densité de notre maillage, on utilise un constante noté nbSubdiv qui représente le nombre de subdivisions sur chaque axe (voir figure 4). Si nbSudiv=4, le maillage est constitué de 4x4=16 patches. La fonction creeTerrain() prend en charge la définition de la liste d'affichage permettant de générer le terrain :

```
void creeTerrain()
{
  int i,j;
  float pas=2.0/nbSubdiv;
  float P1[3],P2[3],P3[3],P4[3];

  /* 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++) {
      P1[0]=-1.0+i*pas; P1[1]=-1.0+j*pas ; P1[2]=elevation(i,j);
      P2[0]=-1.0+(i+1)*pas; P2[1]=-1.0+j*pas ; P2[2]=elevation(i+1,j);
      P3[0]=-1.0+(i+1)*pas; P3[1]=-1.0+(j+1)*pas ; P3[2]=elevation(i+1,j+1);
      P4[0]=-1.0+i*pas; P4[1]=-1.0+(j+1)*pas ; P4[2]=elevation(i,j+1);

      glBegin(GL_TRIANGLES);
      /* triangle 1 */
      glEdgeFlag(TRUE);
      glVertex3fv(P1);
      glVertex3fv(P2);
```

```
      if (!areteTransv)
        glEdgeFlag(FALSE);
      glVertex3fv(P3);

      /*triangle 2 */
      glVertex3fv(P1);
      if (!areteTransv)
        glEdgeFlag(TRUE);
      glVertex3fv(P3);
      glVertex3fv(P4);
      glEnd();
    }
  glEndList();
}
```

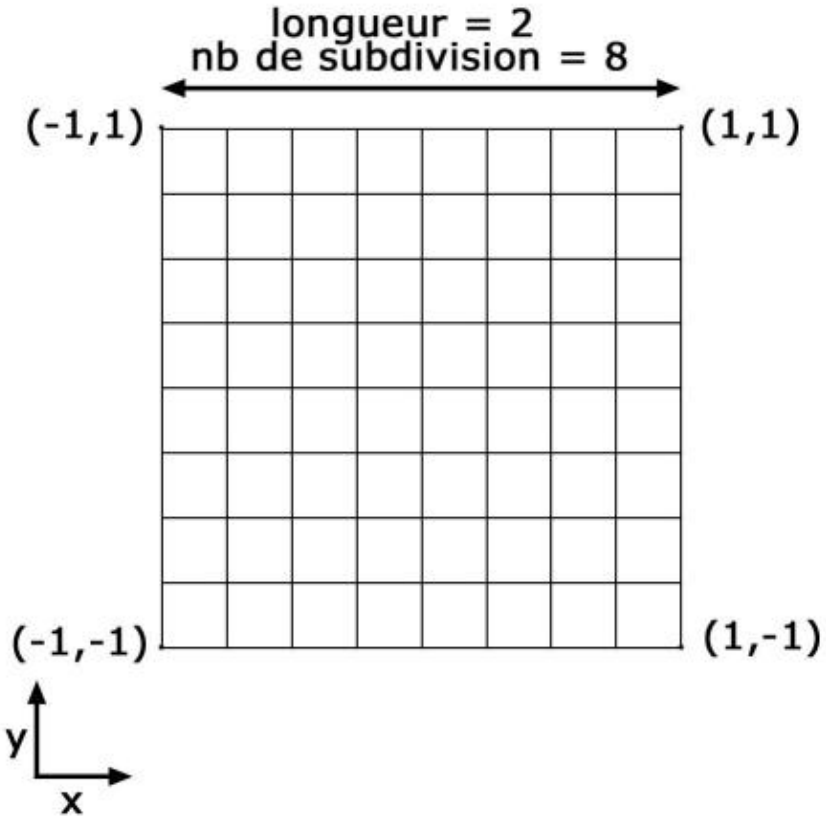

Figure 4: Subdivision du maillage

Le dessin du maillage se fait par une boucle imbriquée avec deux variables i et j. Le cœur de la boucle dessine le patch situé dans la i+1−ième ligne et la j+1−ième colonne. Le sommet supérieur gauche de ce patch se situe au point (−1+i*pas,−1+j*pas,Z) si on considère que 'pas' désigne la largeur d'un patch, c'est à dire 2/nbSubdiv. Les trois autres points du patch se calculent en incrémentant correctement i et j. La coordonnées en Z des sommets et calculée dans la fonction élévation() :

```
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;
}
```

elevation(i,j) renvoie la hauteur du sommet supérieur gauche du patch de la i+1−ième ligne et la j+1−ième colonne. La valeur retournée par élévation est comprise entre –echelleVert (si le pixel est noir) et +echelleVert (si le pixel est blanc). Vous pouvez faire varier l'échelle verticale echelleVert avec les touches 'p' et 'o'.

Bien entendu, nous appliquons le découpage du patch en 2 triangles qui nous permettra d'éviter les problèmes

d'éclairage. Vous avez la possibilité d'activer et de désactiver le masquage de l'arête transversale (en fait, il y en a 2, une par triangle) avec la touche 't'. Vous noterez également que l'argument passé à glBegin() n'est pour une fois pas GL_POLYGON, mais GL_TRIANGLES. En mode GL_TRIANGLES, les sommets décrits sont automatiquement groupés par trois pour créer des triangles. Ceci évite de devoir utiliser une seconde paire d'instructions glBegin()−glEnd() pour créer le second triangle constituant chaque patch.

Conclusion

Ca y est, nous voilà arrivé au bout de ce programme et vous avez enfin un programme OpenGL avec un résultat visuellement intéressant. Il ne reste plus qu'à rendre tout cela plus réaliste, mais il faudra attendre le prochain didacticiel. D'ici là, vous pouvez améliorer ce programme. Une idée d'amélioration consiste à permettre à l'utilisateur d'utiliser une image de taille quelconque. Il faudra alors prendre en compte le problème du rapport hauteur/largeur. Vous pouvez aussi permettre à l'utilisateur de fournir une image en couleur. Si vous vous plongez dans la documentation de la libjpeg, vous verrez qu'il est possible de demander à la bibliothèque de convertir l'image vers un autre format lors de la lecture. Cependant, rien ne vous empêche de faire la conversion vous−même.

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.
ftp://ftp.uu.net/graphics/jpeg/	Vous trouverez ici les sources de la bibliothèque JPEG, ainsi que des documents de référence concernant le format de compression JPEG

Code Source

```
/*************************************************************/
/*                          terrain .c                       */
/*************************************************************/
/* Generation de terrain 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>
```

```
#define NB_SUBDIV_INIT 32
#define NB_SUBDIV_MAX 64
#define ECHELLE_VERT_INIT 0.3
#define ECHELLE_VERT_MAX 1.0
#define DISTANCE_INIT 4.0
#define DISTANCE_MAX 15.0


/* 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 */
int repere,terrain; /* 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 */

/* 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();
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);
  glutInitWindowSize(500,500);
  glutCreateWindow(argv[0]);

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

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

  glutMainLoop();
  return 0;
}



void init()
{
  glClearColor(0.8,0.8,0.8,1.0);
  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);

  /* 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);
  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);
  glCallList(terrain);
  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 'r': /* Affichage du repere ON/OFF */
    afficheRepere=1-afficheRepere;
    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 gauche est presse */
  if (b_droit) {
    distance+=((float)(y-yprec))/10.0;
    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);
  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();
  glEndList();
}



/* Creation de la liste d'afffichage pour le terrain */
void creeTerrain()
{
  int i,j;
  float pas=2.0/nbSubdiv;
  float P1[3],P2[3],P3[3],P4[3];

  /* 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++) {
      P1[0]=-1.0+i*pas; P1[1]=-1.0+j*pas ; P1[2]=elevation(i,j);
      P2[0]=-1.0+(i+1)*pas; P2[1]=-1.0+j*pas ; P2[2]=elevation(i+1,j);
      P3[0]=-1.0+(i+1)*pas; P3[1]=-1.0+(j+1)*pas ; P3[2]=elevation(i+1,j+1);
      P4[0]=-1.0+i*pas; P4[1]=-1.0+(j+1)*pas ; P4[2]=elevation(i,j+1);

      glBegin(GL_TRIANGLES);
      /* triangle 1 */
      glEdgeFlag(TRUE);
      glVertex3fv(P1);
      glVertex3fv(P2);
      if (!areteTransv)
```

```
        glEdgeFlag(FALSE);
      glVertex3fv(P3);

      /*triangle 2 */
      glVertex3fv(P1);
      if (!areteTransv)
      glEdgeFlag(TRUE);
      glVertex3fv(P3);
      glVertex3fv(P4);
      glEnd();
    }
  glEndList();
}



/* 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(
}
```