Introduction :

Nous allons aujourd'hui nous intéresser à un sujet relativement complexe qui nous occupera pendant deux didacticiels : le placage de texture. Bien souvent, la texture en question est une image et nous aborderons donc également le problème du chargement d'images au format JPEG.

Le placage de texture

Le placage de texture est une technique permettant d'accroître le réalisme d'un rendu 3D. Vous en connaissez sans doute le principe : il consiste à coller une image sur un objet 3D à la manière d'une tapisserie. Dans l'exemple que je vous propose aujourd'hui, nous allons reprendre et modifier le cube tournant que nous avons créé dans un précédent tutoriel pour obtenir un cube en bois. Pour cela, nous allons 'coller' une image de bois sur chacune des faces du cube (voir figure1).

Il est intéressant de noter que si dans notre cas l'image utilisée est chargée depuis un fichier au format JPEG, il est tout à fait possible d'utiliser des images calculées à partir de fonctions mathématiques. On parle alors de textures procédurales. Les textures procédurales les plus utilisées sont sans doute les textures de type « nuage » et « bruit de Perlin » (figure 2).

Contrairement à ce qu'on pourrait penser, les textures ne se bornent pas aux images en 2 dimensions. OpenGL permet également d'utiliser des textures 1D, peu intéressantes, et des textures 3D. S'il paraît difficile d'obtenir des images en 3 dimensions (bien que ce soit possible), les textures procédurales 3D sont courantes, et elles possèdent de nombreux avantages par rapport aux textures 2D, notamment en ce qui concernent la continuité des textures aux arêtes. Pour s'en convaincre, il suffit de comparer les figures 1 et 2. L'image collée sur chacune des faces du cube de la figure 1 est la même, et le sens des veinures du bois n'est pas cohérent. En revanche, sur la texture procédurale 3D de la figure 2, la continuité de la texture est assurée sur chacune des faces.

Nous implémenterons dans le prochain didacticiel une texture procédurale. Aujourd'hui, notre tâche va se décomposer en deux parties :

1− Lecture d'une image JPEG
2− Exploitation de l'image chargée par placage sur le cube

Figure 1 : Le programme du mois

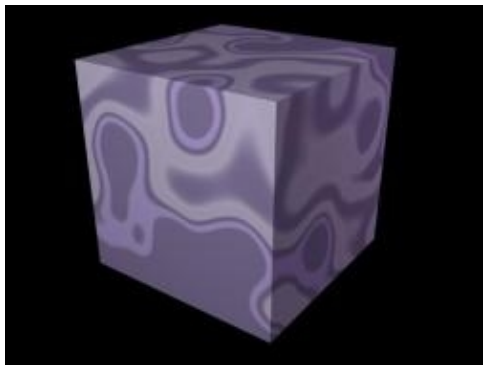

Figure 2 : Une texture procédurale 3D : le bruit de Perlin

Le format JPEG

Le format JPEG (Joint Photographic Experts Group) est sans doute le format de fichier image le plus répandu. Ses principales caractéristiques sont :

- Stockage d'images en mode RGB et niveau de gris (pas de mode couleurs indexées)
- Compression avec perte basée sur une transformée en cosinus discrète
- Qualité de compression paramétrable
- Pas de gestion de la transparence
- Possibilité d'insérer dans l'image des informations supplémentaires par un système de marqueurs

Le JPEG est le format à choisir pour les images complexes (type photographie) dont on veut réduire la taille (pour les transmettre par internet). Attention cependant, la compression avec perte implique une dégradation de l'image, notamment sur les zones à fort contraste.

OpenGL ne prend pas en charge la lecture de fichier au format JPEG, et il serait illusoire de vouloir écrire aujourd'hui une routine adéquate, l'algorithme de compression étant très complexe. Nous allons donc appeler à la rescousse la bibliothèque C JPEG. Cette bibliothèque est livrée avec toutes les distributions standard. Cependant, les fichiers d'entête ne sont pas forcement installés par défaut (par exemple, sur les distributions Mandrake, la bibliothèque se trouve dans l'archive libjpeg.rpm et les fichier d'entête sont dans libjpeg−devel.rpm). Si vous ne parvenez pas à compiler un programme utilisant la bibliothèque JPEG, installez le package devel correspondant, ou bien téléchargez et recompilez la bibliothèque depuis le site FTP cité en référence [8].

La bibliothèque JPEG (libjpeg)

La bibliothèque JPEG permet la lecture et l'écriture de fichiers respectant la spécification du format JPEG. Le mode d'emploi complet de la bibliothèque se trouve dans le fichier libjpeg.doc de la distribution source disponible sur [8]. Contrairement à ce que son extension semble indiquer, il s'agit d'un fichier texte. Si vous parcourez ce fichier, vous vous rendrez compte combien le format JPEG et la bibliothèque sont complexes. Rassurez−vous, nous n'utiliserons aujourd'hui que la partie concernant la décompression de fichier JPEG, et dans le code source de notre exemple, tout se trouve dans la fonction loadJpegImage() :

```
void loadJpegImage(char *fichier)
{
  struct jpeg_decompress_struct cinfo;
  struct jpeg_error_mgr jerr;
  FILE *file;
  unsigned char *ligne;
  int i,j;

  cinfo.err = jpeg_std_error(
  jpeg_create_decompress(
  if ((file=fopen(fichier,"rb"))==NULL)
```

```
    {
      fprintf(stderr,"Erreur : impossible d'ouvrir le fichier texture.jpg\n");
      exit(1);
    }
  jpeg_stdio_src(file);
  jpeg_read_header(TRUE);

  if ((cinfo.image_width!=256)||(cinfo.image_height!=256)) {
    fprintf(stdout,"Erreur : l'image doit etre de taille 256x256\n");
    exit(1);
  }
  if (cinfo.jpeg_color_space==JCS_GRAYSCALE) {
    fprintf(stdout,"Erreur : l'image doit etre de type RGB\n");
    exit(1);
  }

  jpeg_start_decompress(
  ligne=image;
  while (cinfo.output_scanline<cinfo.output_height)
    {
      ligne=image+3*256*cinfo.output_scanline;
      jpeg_read_scanlines(
    }
  jpeg_finish_decompress(
  jpeg_destroy_decompress(

  for (i=0;i<256;i++)
    for (j=0;j<256;j++) {
      texture[i][j][0]=image[i*256*3+j*3];
      texture[i][j][1]=image[i*256*3+j*3+1];
      texture[i][j][2]=image[i*256*3+j*3+2];
    }
}
```

Pour utilisez la bibliothèque JPEG, il faut commencer pas inclure les fichiers d'entête :

```
#include <jpeglib.h>
#include <jerror.h>
```

Le processus de décompression nécessite la mise en oeuvre de 2 structures de données nommées cinfo et jerr. La première va contenir les informations concernant la structure de l'image (ses dimensions, son type, ses paramètres de compression...). On initialise cette structure grâce à la fonction jpeg_create_decompress(). La seconde structure, jerr, va servir au traitement des erreurs qui surviendront éventuellement au cours de la décompression. Par un appel à jpeg_stdio_src(), on indique que les messages d'erreur doivent être envoyés sur la sortie standard. Il convient ensuite d'ouvrir le fichier contenant l'image grâce à un classique fopen(). On prendra soin de vérifier que l'ouverture du fichier s'est bien passée en testant la valeur de retour de fopen(). On indique ensuite que les données concernant l'image à décompresser seront lues depuis le fichier que l'on vient d'ouvrir, avec un appel à jpeg_stdio_src().

A présent, il ne reste plus qu'a lire l'entête du fichier avec jpeg_read_header(). Les informations concernant l'image sont alors placées dans la structure cinfo. On peut donc tester si celles−ci sont conformes à nos souhaits. Nous voulons une image de taille 256x256 en mode RGB. Si les tests sont passés avec succès, on peut commencer la décompression de l'image par un appel à jpeg_start_decompress(). L'image sera décompressée dans un tableau que nous aurons pris soin de déclarer. Sachant que nous souhaitons lire une image de 256x256 et qu'un pixel contient trois composantes (R,V et B) stockées chacune sur un octet, nous avons besoin d'un tableau de 256x256x3 octets (le type correspondant en C est unsigned char qui peut prendre 256 valeurs différentes, de 0 à 255).

```
unsigned char image[256*256*3];
```

Vous remarquerez que nous stockons l'image dans un tableau unidimensionnel La bibliothèque JPEG ne permet pas de décompresser l'image dans un format qu'accepte OpenGL (il lui faut un tableau à trois dimension où la troisième dimension correspond à la composante couleur R,V,B ou A). Nous serons donc obligés de réorganiser le tableau après la fin de la lecture de l'image. La procédure de décompression se fait par un balayage de ligne avec une boucle « tant que » et un pointeur nommé « ligne » qui indique l'adresse à laquelle doivent être placées les données décompressées. La fonction jpeg_read_scanlines() provoque la décompression d'une ligne de l'image. Une fois la décompression de l'image achevée, il ne reste plus qu'à terminer le processus avec jpeg_finish_decompress() et à libérer la structure cinfo avec jpeg_destroy_decompress().

La double boucle imbriquée qui termine la fonction LoadJpegImage() copie les données du tableau image[] dans un nouveau tableau à 3 dimensions qui pourra être exploité comme une texture par OpenGL. Le tableau image[] contient la suite des composantes R,V et B de chacun des pixels, en parcourant l'image de gauche à droite et de haut en bas. Avec des indices démarrant à 0, la composante rouge du pixel de la ligne d'indice 4 et de la colonne d'indice 12 de l'image se trouve dans image[256*4+12], la composante verte de ce même pixel se trouve dans image[256*4+12+1] et sa composante bleue est dans image[256*4+12+2]. Dans le tableau texture, ces mêmes composantes se trouvent respectivement dans texture[4][12][0], texture[4][12][1], texture[4][12][2].

Figure 3 : la texture de bois

Texture et OpenGL

Comme nous l'avons dit en introduction, l'exemple d'aujourd'hui est une reprise du code source du didacticiel sur le cube. Les modifications apportées sont :

- l'utilisation d'une texture
- la mise en place d'une projection perspective
- la suppression des tableaux de stockage des sommets. La description des faces du carré se fera «à la main », afin de faciliter la compréhension du positionnement de la texture.

Utilisation de la texture

L'utilisation du placage de texture faite dans ce programme est quasiment la plus simple qui soit : nous n'utilisons qu'une texture, et nous la plaquons sur des faces carrées. Nous verrons dans le prochain tutoriel que l'opération de positionnement de texture est souvent bien plus compliquée.

La première étape nécessaire au placage de la texture est évidemment ... le chargement de l'image avec la fonction LoadJpegImage(). Nous allons ensuite paramétrer la manière dont OpenGL devra filtrer la texture lors de l'application de celle−ci sur un objet. Lors du rendu, la texture va être déformée par la perspective. A certains endroits elle va être étirée, à d'autre elle va être rétrécie. Le filtrage définit la méthode de calcul final de la texture déformée. OpenGL propose les méthodes « nearest » (la plus rapide) et « linear » (la plus jolie). Afin de vous permettre de comparer les deux méthodes, le programme vous permet de basculer d'une méthode à l'autre grâce aux touches 'n' et 'l'. Le paramétrage de la méthode de placage de texture se fait grâce à la fonction

```
void glTexParameteri(Glenum cible,GLenum nomparam,GL valeur)
```

'cible' définit le type de texture que l'on veut paramétrer (GL_TEXTURE_1D,GL_TEXTURE_2D ou GL_TEXTURE_3D). nomparam désigne le paramètre que l'on souhaite modifier. Nous ne modifierons aujourd'hui que la méthode de filtrage lors d'un étirement (GL_TEXTURE_MAG_FILTER) ou d'un rétrécissement (GL_TEXTURE_MIN_FILTER). 'valeur' correspond à la nouvelle valeur à affecter au paramètre. Dans notre cas, il s'agit de GL_LINEAR ou GL_NEAREST.

L'étape suivante consiste à définir la texture 2D que nous souhaitons utiliser avec :

```
void glTexImage2D(GLenum cible,Glint niveau, Glint formatinterne,GLsizei largeur,
     GLsizei hauteur,GLint bord,GLenum format,GLenum type,const Glvoid *texture)
```

La cible sera pour nous GL_TEXTURE_2D. le paramètre de niveau n'est utile que si vous utilisez le mipmapping (textures multirésolution). Nous lui donnerons la valeur 0. Le format interne de stockage de la texture sera GL_RGB. La

hauteur et la largeur de la texture valent toutes les deux 256. La texture ne possède pas de bord (valeur 0). L'image fournie est en mode RGB et chaque composante est codée sur un caractère non signé, donc on affectera respectivement GL_RGB et GL_UNSIGNED_BYTE aux paramètres de format et de type.

Il est également nécessaire d'activer l'utilisation de la texture2D grâce à l'appel

```
glEnable(GL_TEXTURE_2D)
```

Enfin la dernière étape consiste à positionner la texture sur le polygone. En anglais, on parle de «UV mapping». Le principe est simple : on affecte un système de coordonnées (u,v) à la texture suivant l'illustration de la figure 4. Positionner la texture consiste à affecter à chaque sommet d'un polygone la coordonnée de la texture en ce point. Dans notre cas, puisque les polygones sont des carrés, la tâche est simple, les coordonnées de textures aux sommets des points des faces sont simplement (0,0), (1,0), (1,1) et (0,1). A l'instar de la couleur des sommets, les coordonnées de texture se spécifient lors de la déclaration des polygones entre un glBegin() et un glEnd(). Le principe est toujours le même : lorsqu'un sommet est déclaré avec glVertex(), la coordonnée de texture courante lui est affectée. La modification de la coordonnée de texture courante se faite par un appel à

```
glTexCoord2f(GLfloat u,Glfloat v)
```

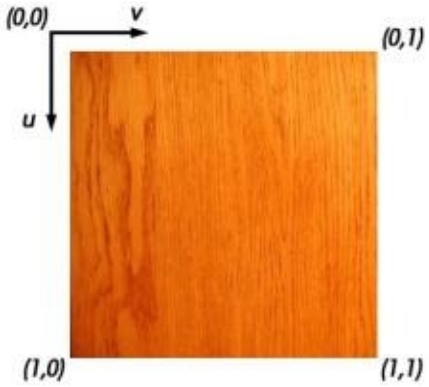

Figure 4 : Coordonnées de texture

Conclusion

La compilation du code source qui suit nécessite que vous indiquiez au compilateur qu'il doit lier le programme à la bibliothèque jpeg, grâce à l'option −ljpeg. Comme d'habitude, je vous invite à modifier ce programme et à visualiser les changement que cela entraîne sur le rendu. La prochaine fois, nous continuerons notre tour d'horizon du placage de texture en compliquant un peu le tout : textures multiples, transparence, éclairage des textures.

Références

[1] 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
[2] Eclairage et rendu numériques	Jeremy Birn − Campus Press. Orienté pratique, cet ouvrage vous apprendra à créer des rendus de qualité.
[3] Introduction à l'Infographie	Foley, Van Dam, Feiner et Hughes − Vuibert La bible de l'informatique graphique.
[4] www.opengl.org	Le site officiel d'OpenGL. Tout y est : présentation, documents de spécification, liens vers des didacticiels, bibliographie
[5] www.mesa3d.org	Le site de Mesa, l'implémentation libre d'OpenGL la plus utilisée sous Linux
[6] reality.sgi.com/mjk/glut3	La page de glut. Vous y trouverez le manuel de référence glut
[7] http://www.linuxgraphic.org/section3d/openGL/index.html	La section OpenGL du site Linuxgraphic.org. Un tout nouveau forum attend vos questions.
[8] 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

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

unsigned char image[256*256*3];
unsigned char texture[256][256][3];
char presse;
int anglex=30,angley=20,x,y,xold,yold;

void affichage();
void clavier(unsigned char touche,int x,int y);
void souris(int boutton, int etat,int x,int y);
void sourismouv(int x,int y);
void redim(int l,int h);
void loadJpegImage(char *fichier);

int main(int argc,char **argv)

{
  /* Chargement de la texture */
  loadJpegImage("texture.jpg");

  /* Creation de la fenetre OpenGL */
  glutInit(
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(200,200);
  glutCreateWindow("Texture JPEG");

  /* Initialisation de l'etat d'OpenGL */
  glClearColor(0.0,0.0,0.0,0.0);
  glShadeModel(GL_FLAT);
```

```
  glEnable(GL_DEPTH_TEST);

  /* Mise en place de la projection perspective */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,1,1.0,5.0);
  glMatrixMode(GL_MODELVIEW);

  /* Parametrage du placage de textures */
  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,texture);
  glEnable(GL_TEXTURE_2D);

  /* Mise en place des fonctions de rappel */
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);
  glutMouseFunc(souris);
  glutMotionFunc(sourismouv);
  glutReshapeFunc(redim);

  /* Entrée dans la boucle principale glut */
  glutMainLoop();
  return 0;
}



void affichage()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();

  gluLookAt(0.0,0.0,2.5,0.0,0.0,0.0,0.0,1.0,0.0);
  glRotatef(angley,1.0,0.0,0.0);
  glRotatef(anglex,0.0,1.0,0.0);

  glBegin(GL_POLYGON);
  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();

  glBegin(GL_POLYGON);
  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();

  glBegin(GL_POLYGON);
  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();

  glBegin(GL_POLYGON);
  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();

  glBegin(GL_POLYGON);
```

```
  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();
  glBegin(GL_POLYGON);
  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();

  glutSwapBuffers();

}



void clavier(unsigned char touche,int x,int y)
{
  switch(touche) {
  case 'l':
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glutPostRedisplay();
    break;
  case 'n':
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    glutPostRedisplay();
  break;

  case 27: /* touche ESC */
    exit(0);
  default:
  }
}

void souris(int bouton, int etat,int x,int y)
{
  if (bouton == GLUT_LEFT_BUTTON &etat == GLUT_DOWN)
  {
    presse = 1;
    xold = x;
    yold=y;
  }
  if (bouton == GLUT_LEFT_BUTTON &etat == GLUT_UP)
    presse=0;
}



void sourismouv(int x,int y)
{
  if (presse)
  {
    anglex=anglex+(x−xold);
    angley=angley+(y−yold);
    glutPostRedisplay();
  }

  xold=x;
  yold=y;
}


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 loadJpegImage(char *fichier)
{
  struct jpeg_decompress_struct cinfo;
  struct jpeg_error_mgr jerr;
  FILE *file;
  unsigned char *ligne;
  int i,j;

  cinfo.err = jpeg_std_error(
  jpeg_create_decompress(
  if ((file=fopen(fichier,"rb"))==NULL)
    {
      fprintf(stderr,"Erreur : impossible d'ouvrir le fichier texture.jpg\n");
      exit(1);
    }
  jpeg_stdio_src(file);
  jpeg_read_header(TRUE);

  if ((cinfo.image_width!=256)||(cinfo.image_height!=256)) {
    fprintf(stdout,"Erreur : l'image doit etre de taille 256x256\n");
    exit(1);
  }
  if (cinfo.jpeg_color_space==JCS_GRAYSCALE) {
    fprintf(stdout,"Erreur : l'image doit etre de type RGB\n");
    exit(1);
  }

  jpeg_start_decompress(
  ligne=image;
  while (cinfo.output_scanline<cinfo.output_height)
    {
      ligne=image+3*256*cinfo.output_scanline;
      jpeg_read_scanlines(
    }
  jpeg_finish_decompress(
  jpeg_destroy_decompress(

  for (i=0;i<256;i++)
    for (j=0;j<256;j++) {
      texture[i][j][0]=image[i*256*3+j*3];
      texture[i][j][1]=image[i*256*3+j*3+1];
      texture[i][j][2]=image[i*256*3+j*3+2];
    }
}
```