3 TP OpenGL - Plaquage de textures

Nicholas Journet - TP OpenGL - IUT - [1]

3.1 Introduction

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. Nous allons reprendre et modifier le cube tournant que nous avons créé dans un précédement pour obtenir un cube avec une photo dessus. Pour cela, nous allons 'coller' une image sur chacune des faces du cube.
OpenGL ne prend pas en charge la lecture de fichier au format JPEG. Nous allons donc appeler à la rescousse la bibliothèque `C JPEG` (cette bibliothèque est livrée avec toutes les distributions standard).
La bibliothèque JPEG permet la lecture et l'écriture de fichiers respectant la spécification du format JPEG. Nous n'utiliserons que la partie concernant la décompression de fichier JPEG, et dans le code source de notre exemple, tout se trouve dans la fonction `loadJpegImage()` qui vous est fournie.
Afin de simplifier ce TP, nous ne manipulerons que des images de 256x256 qu format `JPEG` 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 programme contiendra donc la déclaration suivante :

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

La fonction `LoadJpegImage()` copie les données du tableau `image[]` dans un nouveau tableau `(texture[][][])` 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]`.

3.2 Plaquage d'une texture

Dans cette partie nous n'utilisons qu'une texture, et nous la plaquons sur des faces carrées. La première étape nécessaire au placage de la texture 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). Le paramétrage de la méthode de placage de texture se fait grâce à la fonction

```
1  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.
  La méthode de filtrage lors d'un étirement est `GL_TEXTURE_MAG_FILTER`. Celle correspondant à un rétrécissement est `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`.

1. Ce tp est issu des notes de cours de Xavier Michelon - linuxorg. Toute modification de ce support de cours doit y faire référence

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

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

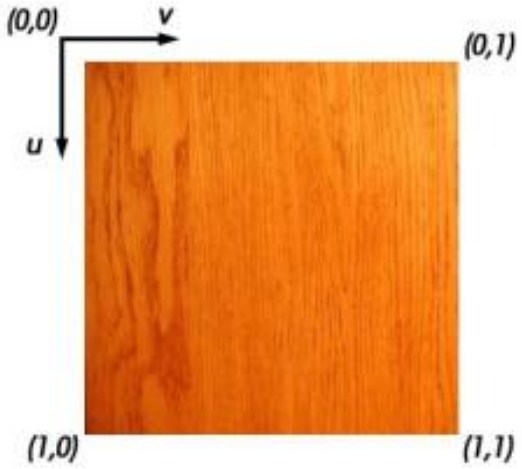
- La cible sera pour nous `GL_TEXTURE_2D`.
- le paramètre de niveau n'est utile que si vous utilisez le mipmapping (textures multirésolution). Nous lui donnerons la valeur 0
- Le format interne de stockage de la texture sera `GL_RGB`.
- La hauteur et la largeur de la texture valent toutes les deux 256
- 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
- `texture` est la texture a appliquée (celle chargée par `LoadJpegImage()`)

.

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

```
1 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 suivante.

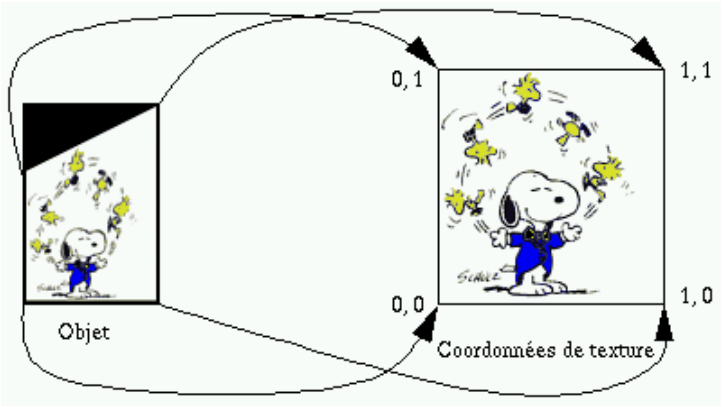


Positionner la texture consiste à affecter à chaque sommet d'un polygone la coordonnée de la texture en ce point. Dans notre cas, puisque les polygones sont des carrés, la tâche est simple, les coordonnées de textures aux sommets des points des faces sont simplement `(0,0)`, `(1,0)`, `(1,1)` et `(0,1)`. 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 :

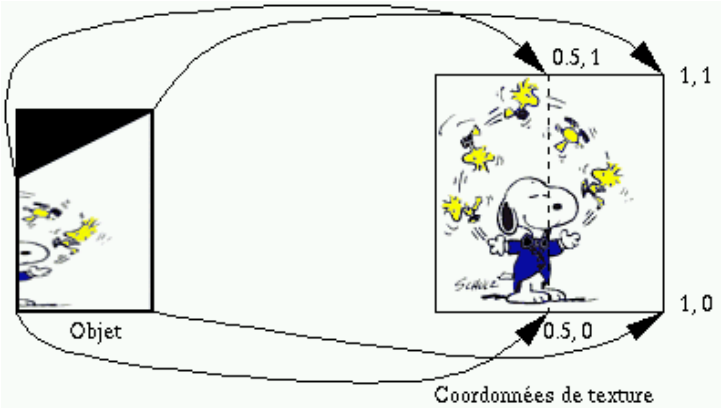
```
1 glTexCoord2f(GLfloat u,Glfloat v)
```

Si l'on désire que toute l'image soit affichée sur un objet de type quadrilatère, on assigne les valeurs de texture `(0.0, 0.0)` `(1.0, 0.0)` `(1.0, 1.0)` et `(0.0, 1.0)` aux coins du quadrilatère [2].

2. Image et texte extraits des notes de cours de E Bittar - univ Reins



Si l'on désire mapper uniquement la moitié droite de l'image sur cet objet, on assigne les valeurs de texture `(0.5, 0.0) (1.0, 0.0) (1.0, 1.0) et (0.5, 1.0)` aux coins du quadrilatère[3].



3.3 Mise en application

Le code correspondant à la fonction `loadJpegImage()` vous est fournie. Les fonctions `redim()` `sourismouv()` `souris()` sont les mêmes que dans les exercices précédents.
On vous donne également le main du programme. Vous remarquerez la déclaration des deux tableaux permettant de stocker l'image et la texture. Vous remarquerez que l'intialisation des parametres permettant le placage de textures correspond à ce que l'on a décrit dans la section précédente.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <GL/glut.h>
4  #include <jpeglib.h>
5  #include <jerror.h>
6
7  unsigned char image[256*256*3];
8  unsigned char texture[256][256][3];
9  char presse;
10 int anglex=30,angley=20,x,y,xold,yold;
11
12 void affichage();
13 void clavier(unsigned char touche,int x,int y);
14 void souris(int boutton, int etat,int x,int y);
15 void sourismouv(int x,int y);
16 void redim(int l,int h);
17 void loadJpegImage(char *fichier);
18
19 int main(int argc,char **argv)
20
```

3. Image et texte extraits des notes de cours de E Bittar - univ Reins

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

  /* Creation de la fenetre OpenGL */
  glutInit(&argc,argv);
  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);

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

Nous allons maintenant écrire la fonction de rappel `clavier()` permettant de changer la méthode de placage de texture.

```
void clavier(unsigned char touche,int x,int y)
{
  switch( _______ ) {
  case 'l'://placage lineaire d'une texture 2D
    //la plaquage d'un etirement sera lineaire
    glTexParameteri( ____________________________________ );
    //la plaquage d'un retrecissement sera lineaire
    glTexParameteri( ____________________________________ );
//on oublie pas de demander a redessiner
      ____________________ ;
    break;
  case 'n'://placage au plus proche voisin d'une texture 2D
    //la plaquage d'un etirement sera realise en fonction du voisin
    glTexParameteri( ____________________________________ );
    //la plaquage d'un retrecissement sera realise en fonction du voisin
    glTexParameteri( ____________________________________ );
     //on oublie pas de demander a redessiner
      ____________________ ;
```

```
19    break;
20
21    case 27: /* touche ESC */
22      exit(0);
23
24    }
25  }
```

Nous allons enfin écrire la fonction `display` dans laquelle nous plaquerons la texture sur le cube.

```
1   void affichage()
2   {
3     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
4
5     glLoadIdentity();
6
7     gluLookAt(0.0,0.0,2.5,0.0,0.0,0.0,0.0,1.0,0.0);
8     glRotatef(angley,1.0,0.0,0.0);
9     glRotatef(anglex,0.0,1.0,0.0);
10
11
12    glTexCoord2f(          );   glVertex3f(-0.5, 0.5, 0.5);
13    glTexCoord2f(          );   glVertex3f(-0.5,-0.5, 0.5);
14    glTexCoord2f(          );   glVertex3f( 0.5,-0.5, 0.5);
15    glTexCoord2f(          );   glVertex3f( 0.5, 0.5, 0.5);
16
17
18
19  //On repete la meme operation pour les 5 autres faces dont les coordonees
      sont:
20  //face 2:
21      ( 0.5, 0.5, 0.5);( 0.5,-0.5, 0.5);( 0.5,-0.5,-0.5);( 0.5, 0.5,-0.5);
22  //face3:
23    ( 0.5, 0.5,-0.5);( 0.5,-0.5,-0.5);(-0.5,-0.5,-0.5);(-0.5, 0.5,-0.5);
24  //face 4:
25  (-0.5, 0.5,-0.5);(-0.5,-0.5,-0.5);(-0.5,-0.5, 0.5);(-0.5, 0.5, 0.5);
26  //face 5:
27  (-0.5, 0.5,-0.5);(-0.5, 0.5, 0.5);( 0.5, 0.5, 0.5);( 0.5, 0.5,-0.5);
28  //face 6:
29    (-0.5,-0.5,-0.5);(-0.5,-0.5, 0.5);( 0.5,-0.5, 0.5);(0.5,-0.5,-0.5);
30
31    glutSwapBuffers();
32
33  }
```

3.4 Plaquage de plusieurs textures

Dans la section précédente nous avons plaqué une texture sur un objet. Dans cette partie nous allons ajouter un objet et lui appliquer une autre texture.
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 faut demander à OpenGL de vous attribuer un identifiant en faisant un appel à la fonction

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

- `nombre` indique le nombre de textures que vous souhaitez créer
- `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 :

```
1  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 :

```
1
2
3
4  /*Lors de l'initialisation, on reserve les identifiants de texture et on
       cree les textures*/
5  glGenTextures(2,IdTex);
6  glBindTexture(GL_TEXTURE_2D,IdTex[0]) ;
7  /* Creation de la premiere texture (avec glTexParameteri(),
       glTexImage2D...) */
8  ...
9  glBindTexture(GL_TEXTURE_2D,IdTex[1]) ;
10 /* Creation de la deuxieme texture */
11 ...
12 //Au cours de la fonction d'affichage, on change eventuellement la texture
       active avant la description de chaque face :
13
14 void affichage()
15 {
16  ...
17  glBindTexture(GL_TEXTURE_2D,IdTex[0]);
18  glBegin(GL_POLYGON);
19  /* description de la face 1 */
20  glEnd();
21
22  glBindTexture(GL_TEXTURE_2D,IdTex[1]);
23  glBegin(GL_POLYGON);
24  /* description de la face 2 */
25  glEnd();
26  ...
27  }
```

Dans la section précédente, nous avons chargé l'image au format `jpeg`. Ici, nous utiliserons une fonction permettant de charger une image `tif` (ne pas oublier d'inclure `tiffio.h`)

```
1  void chargeTextureTiff(char *fichier,int numtex)
```

Tout comme `loadJpegImage`, cette fonction permet de charger l'image dans un tableau adéquat aux fonctions d'OpenGL. La seule différence est que la fin de cette fonction comporte un paramétrage de la texture en fonction du numéro passé en paramètre. Cela permet de gérer plusieurs textures.

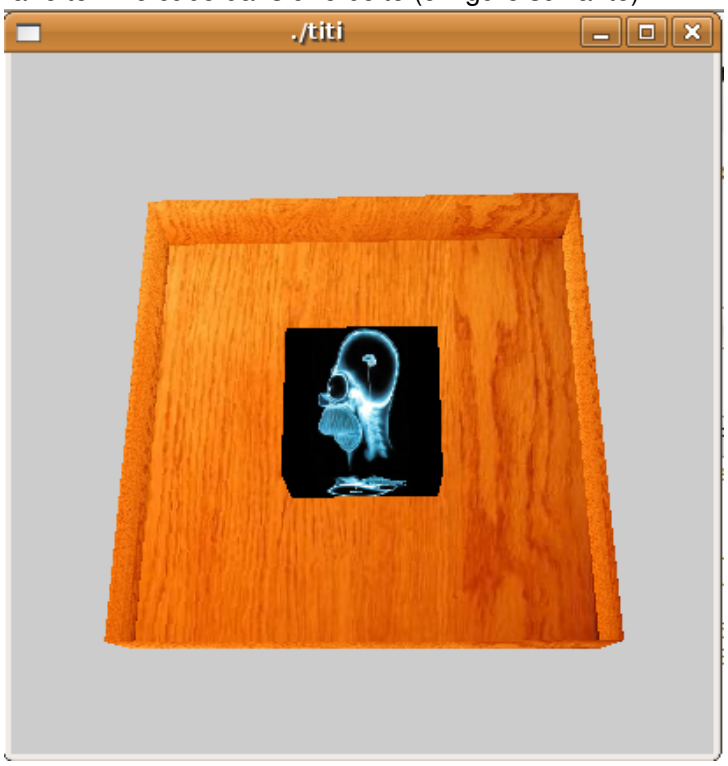
```
1  void chargeTextureTiff(char *fichier,int numtex)
2  {
3    unsigned char image[256][256][3];
4    uint32 l, h;
5    int i,j;
6    size_t npixels;
7    uint32* raster;
8  /*Cf code fourni*/
9    TIFFClose(tif);
```

```
/*parametrage de la texture */

//On associe l'image charge a la texture numero mutex
  glBindTexture(GL_TEXTURE_2D,numtex);
//parametrisation des filtres de taille
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
//definition de la texture 2D
  glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,256,256,0,
     GL_RGB,GL_UNSIGNED_BYTE,image);
}
```

3.5 Mise en application

Nous allons faire en sorte de faire tenir le cube dans une boite (cf figure suivante)



Tout d'abord recopier le `main()` suivant. Rien de bien nouveau, observez simplement que le chargement de texture est bien réalisé dans le main (phase d'initialisation)

```

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <tiffio.h>
#include <string.h>
#include <math.h>
#define PI 3.14159265
/*n'oubliez pas les prototypes des fonctions */
/* declaration des variables */
float distance=5.5; /* distance de l'observateur a l'origine */
int anglex=30,angley=20,xprec,yprec; /* stockage des deplacements 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 */
/* declaration des indicateurs booleens */
```

```
unsigned char b_gauche=0; /* le bouton gauche de la souris est il presse ? */
unsigned char b_droit=0; /* le bouton droit de la souris est il presse ? */
int main(int argc,char **argv)
{
  /* initialisation de glut */
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
  glutInitWindowSize(400,400);
  glutCreateWindow(argv[0]);
  /* Chargement des textures */
  glGenTextures(2,IdTex);
  chargeTextureTiff("texture.tif",IdTex[0]);
  chargeTextureTiff("texturehomer.tif",IdTex[1]);

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

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

  /* mise en place du tampon de profondeur */
    masqueClear=GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT;
    glEnable(GL_DEPTH_TEST);

  /* mise en place des fonctions de rappel glut */
  glutDisplayFunc(affichage);
  glutMouseFunc(souris);
  glutMotionFunc(mouvSouris);
  glutReshapeFunc(redim);
  glutMainLoop();
  return 0;
}
```

Nous allons maintenant compléter la fonction affichage de la question précédente.
Normalement vous avez créé un cube de taille 1 centré en 0 (soit 6 faces). Pour créer la boite, vous avez tout simplement à créer un cube à qui il manque une face et dont la texture est différente du cube.

```
 void affichage()
{
  glClear(masqueClear);
  /* Positionnement de l'observateur (ou de l'objet) */
  glLoadIdentity();
//Vous pouvez changer la valeur de distance dans la fonction de rappel de la souris
  gluLookAt(0.0,0.0,distance,0.0,0.0,0.0,0.0,1.0,0.0);
//idem pour l'angle de vue
  glRotatef(angley,1.0,0.0,0.0);
  glRotatef(anglex,0.0,1.0,0.0);

/* Description de la boite */
  glBindTexture(GL_TEXTURE_2D,IdTex[0]);
```

```
14    glBegin(GL_POLYGON);
15    glTexCoord2f(0.0,1.0);    glVertex3f(-1.5, 1.5, 0.5);
16    glTexCoord2f(0.0,0.0);    glVertex3f(-1.5,-1.5, 0.5);
17    glTexCoord2f(1.0,0.0);    glVertex3f( 1.5,-1.5, 0.5);
18    glTexCoord2f(1.0,1.0);    glVertex3f( 1.5, 1.5, 0.5);
19    glEnd();
20  //COMPLETER  EN CREANT LES AUTRES FACES
21    /* Description de l'obet1 */
22  //Si vous creez le meme cube que dans la question precedente, vous allez
      definir deux faces dont l'une est incluse dans l'autre (celle du cube et
      celle de la boite). Le Z buffer ne sait pas gerer ce cas de figure et
      vous verrez donc apparaitre des clignotements.
23  //A vous de trouver un moyen de remedier a ce probleme.
24    glBindTexture(GL_TEXTURE_2D,IdTex[1]);
25    glBegin(GL_POLYGON);
26    //code identique a celui de la question precedente
27    /*  echange de tampon (double buffering)*/
28    glutSwapBuffers();
29  }
```

3.6 Déplacement du cube

Dans cette section nous allons faire bouger le cube si l'on clique dessus. Le problème est que la souris se déplace dans un plan en 2D alors que nous souhaitons obtenir des coordonnées en 3D lorsque l'on clique.
Heureusement OpenGL nous fournit des fonction qui permettent d'interpoler (en fonction du mode de projection, de la vue...) les coordonnées 3D à partir des coordonnées 2D.
Vous pouvez utiliser directement le bout de code suivant dans une fonction de rappel. Les 3 coordonnées interpolées sont dans les variables `ox,oy,oz`

```
1  GLdouble ox=0.0,oy=0.0,oz=0.0;
2  GLint viewport[4];
3  GLdouble modelview[16],projection[16];
4  GLfloat wx=x,wy,wz;
5  glGetIntegerv(GL_VIEWPORT,viewport);
6  y=viewport[3]-y;
7  wy=y;
8  glGetDoublev(GL_MODELVIEW_MATRIX,modelview);
9  glGetDoublev(GL_PROJECTION_MATRIX,projection);
10 glReadPixels(x,y,1,1,GL_DEPTH_COMPONENT,GL_FLOAT,&wz);
11 gluUnProject(wx,wy,wz,modelview,projection,viewport,&ox,&oy,&oz);
```

Il ne vous reste plus qu'à vérifier que les 3 coordonnées correspondent au cube dessiné et à le déplacer le cas échéant.

```
1  if (ox <0.5 && ox>-0.5 && oy <0.5 && oy>-0.5)
2  decalx++;
```

Dans la fonction d'affichage, il suffit de positionner chaque point en fonction des variables de décalage.

```
1  glVertex3f( 0.5+decalx, 0.5+decaly, 0.5);
```