Introduction

Après notre présentation d'OpenGL, nous passons aujourd'hui à la pratique, avec l'étude d'un exemple classique qui peut être considéré comme le "Hello World" d'OpenGL. Le but est d'aborder les mécanismes standards de la programmation OpenGL en C : création de fenêtre avec glut, gestion des événements, description de la scène 3D.

L'exemple que nous allons étudier aujourd'hui est relativement simple : notre but est d'afficher à l'écran un carré coloré. En guise de petit plus, et afin d'aborder la gestion des événements, nous donnerons à l'utilisateur la possibilité de modifier la méthode d'affichage du carré (carré plein, fil de fer, ou sommets seuls) par l'intermédiaire du clavier. Pour l'instant, nous nous contenterons d'un carré en deux dimensions, car un passage à la troisième dimension pour l'affichage d'un cube nous obligerait à considérer des notions de projection perspective et de placement de la caméra virtuelle qui feront l'objet de plusieurs didacticiels.

Notre scène 3D

Avant de commencer à coder, je vous propose de visualiser sur la figure 1 la scène que devra représenter notre petit programme. Le résultat final est sur la figure 2.

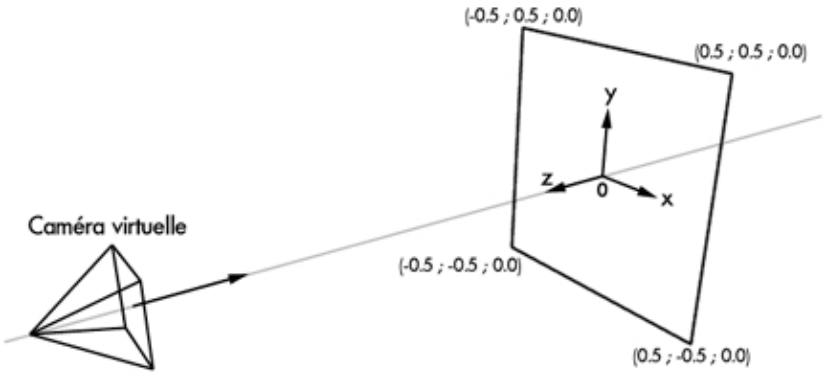

Figure 1 : La scène 3D

Préparatifs :

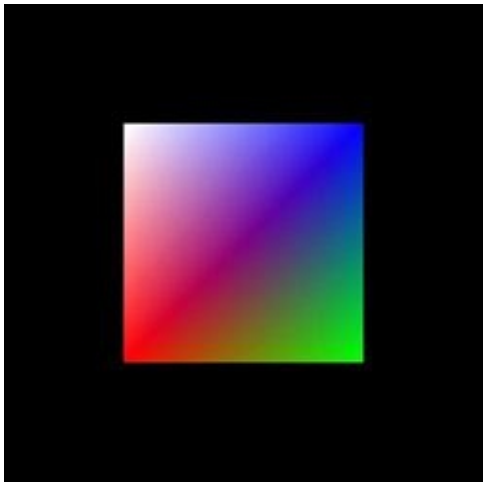

Figure 2 : Le résultat final (en couleur !)

A la manière d'un modeleur 3D, nous utiliserons en OpenGL le concept de caméra virtuelle. Virtuelle parce qu'elle n'est pas à proprement parler un objet de la scène 3D, mais elle nous indique le point de vue de l'observateur, la direction du regard et l'inclinaison de la caméra. Comme nous n'aborderons pas les modifications de point de vue et de perspective dans cet article, le point de vue sera celui qu'OpenGL offre par défaut : la caméra est située sur l'axe Z, pointe vers l'origine du repère (le point O), et la verticale de la caméra correspond à l'axe Y du repère. La vue de la scène se fait grâce à une projection orthogonale (pas de point de fuite pour les objets) qui permet de visualiser dans la fenêtre tous les points dont les coordonnées en X et Y sont comprises entre −1 et +1. En définissant notre carré par les 4 points indiqués sur la figure 1, on obtient la vue représentée sur la figure 2.

Conventions de noms de fonctions

Les fonction OpenGL sont régies par un certain nombre de règles :

- Tous les noms de fonctions OpenGL sont préfixés : les noms de fonction OpenGL commencent par 'gl'. Pour les bibliothèques annexes telles que GLU et GLX, les préfixes respectifs sont 'glu' et 'glX'. En guise d'exercice, je vous laisse déterminer le préfixe pour les fonctions glut ;).
- Certaines commandes OpenGL existent en plusieurs variantes, suivant le nombre et le type de paramètres qu'admet la variante. Ainsi, la fonction glVertex3f() prend trois paramètres ('3') de type Float ('f'), alors que glVertex2i() fonctionne avec 2 entiers. Certaines fonctions possèdent une version suffixée par un 'v'. Les paramètres sont passés à ces fonctions par l'intermédiaire d'un pointeur sur un tableau.
- OpenGL redéfinit des types de données numériques : GLint correspond aux entiers, GLfloat aux flottants, GLbyte aux caractères non signés... Il est préférable d'utiliser ces types de données plutôt que les types standards du C.

A l'attaque !

Votre session emacs est ouverte ? Allons−y ! Première chose à faire : inclure les fichiers d'en−tête OpenGL. Il en existe plusieurs, mais heureusement glut les appelle pour nous. On se contentera donc d'un simple

```
#include <GL/glut.h>
```

La bibliothèque OpenGL est conçue comme une machine à états. Par conséquent, une des premières choses à faire est d'initialiser cette machine. Le nombre de variables d'état accessibles est assez impressionnant. Le mode de dessin, la couleur de fond et le type d'éclairage en sont des exemples. Au cours de cette série d'articles, nous introduirons progressivement les états les plus couramment utilisés. Bien sûr, OpenGL dispose d'un état par défaut, et nous nous contenterons donc dans la phase d'initialisation de régler 2 états : la couleur de fond, et la taille d'un point. Les fonctions à utiliser sont les suivantes:

```
void glClearColor(GLclampf rouge,GLclampf vert,GLclampf bleu,GLclampf alpha);
void glPointSize(GLfloat taille);
```

glClearColor() permet de spécifier la couleur de remplissage utilisée lors d'un effacement de la scène. On peut donc l'assimiler à la couleur du fond. Le mode de spécification de couleur est le classique RGB avec une composante supplémentaire alpha, utilisée pour la gestion de la transparence des objets. Nous ne l'utiliserons pas ici et lui

affecterons systématiquement la valeur 0. Chacune des 4 composantes doit être comprise entre 0 et 1. Si vous avez l'habitude de travailler avec des entiers compris entre 0 et 255, une simple division de chaque composante par 255 vous permettra de trouver la bonne valeur. Ainsi glClearColor(1.0,1.0,1.0,0.0) nous donne un fond blanc, et glClearColor(0.0,0.0,0.0,0.0) un fond noir.

Par défaut, lorsqu'on choisit de représenter les objets simplement par leurs sommets, la taille des sommets à l'écran est de 1 pixel. Afin de les rendre plus visible, nous réglerons la taille des sommets à 2 pixels :

```
glPointSize(2.0);
```

Initialisation de glut et création de la fenêtre :

Comme vous nous l'avons vu le mois dernier, c'est glut qui gère les interactions entre OpenGL et le serveur X. La phase d'initialisation va nous permettre de créer notre fenêtre. Un des atouts de glut est sa grande simplicité. L'initialisation de glut se fait de la manière suivante :

```
glutInit(
glutInitDisplayMode(GLUT_RGB);
glutInitWindowPosition(200,200);
glutInitWindowSize(250,250);
glutCreateWindow("ogl1");
```

Si vous suivez la rubrique "Programmation GTK" de David Odin dans Linux Magazine, vous avez peut−être une idée de ce que fait la fonction glutInit(). Elle analyse les données passées en paramètres au programme et initialise la bibliothèque glut, notamment en établissant la communication avec le serveur X. La fonction glutInitDisplayMode est celle qui vous paraîtra sans doute la plus obscure. Elle permet de régler les paramètres liés à l'affichage : type d'image (palette indexée ou RVB), utilisation d'un tampon de profondeur (plus connu sous le nom de Z−buffer ou depth buffer), utilisation du double buffering... Aujourd'hui un simple tampon d'image en mode RVB défini par la constante GLUT_RGB nous suffira.

Afin de satisfaire les plus anglophobes, je vais quand même préciser que les fonctions glutInitWindowPosition() et glutInitWindowSize() permettent respectivement de définir la position du coin supérieur gauche de la fenêtre OpenGL par rapport au coin supérieur gauche de l'écran, et de spécifier la largeur et la hauteur de la fenêtre.

La fenêtre est créée grâce à l'appel à glutCreateWindow() et son nom doit être transmis sous forme de chaîne de caractères. Il s'agit du nom qui apparaîtra notamment dans la décoration de la fenêtre, gérée par le window manager. Si la fenêtre est bel et bien créée, elle ne sera affichée à l'écran que lors de l'entrée dans la boucle de gestion des événements.

Mise en place des fonctions de rappel :

Le système de gestion des événements offert par glut est relativement similaire à celui de GTK. Des fonctions sont associées aux différents types d'événements envoyés par le serveur X, puis une boucle d'attente est lancée. A chaque fois qu'un événement est émis par le serveur X, la fonction de rappel associée à l'événement est appelée. Vous devez associer au moins une fonction de rappel, la fonction d'affichage, qui est celle dans laquelle vous devez décrire votre scène 3D. Vous pouvez associer des fonctions de rappel aux événements liés :

- au clavier
- à la souris
- à des périphériques d'entrée répandus en infographie (spaceballs, boîtes de potentiomètres, tablettes graphiques...)
- à la modification de la configuration de la fenêtre

Il existe par ailleurs deux fonctions de rappel spéciales qui permettent d'enregistrer des fonctions qui seront appelées à intervalles de temps réguliers ou pendant les temps d'oisiveté (idle), c'est−à−dire lorsque le gestionnaire n'a aucun événement à traiter. Ces deux fonctions sont extrêmement pratiques pour la création de scènes animées.

Dans notre programme, en plus de la fonction d'affichage obligatoire, nous allons mettre en place une fonction de rappel pour le clavier. Les deux appels suivants permettent de spécifier quelles seront les fonctions respectivement associées à l'affichage et au clavier :

```
glutDisplayFunc(affichage);
glutKeyboardFunc(clavier);
```

Après cela, il ne nous reste plus qu'à lancer la boucle d'attente des événements :

```
glutMainLoop();
```

La fonction d'affichage :

La fonction d'affichage constitue le coeur du programme. C'est ici que nous allons décrire la scène. Pour être plus exact, la fonction d'affichage contient la procédure à appliquer pour redessiner notre scène dans la fenêtre, en commençant par un remplissage de la fenêtre avec la couleur de fond que nous avons défini dans la phase d'initialisation. On utilise pour cela

```
void glClear(GLbitfield masque);
```

OpenGL travaille avec plusieurs zone tampons en plus de la zone d'image (tampon de profondeur, d'accumulation...). Le paramètre masque permet de spécifier les tampons que l'on souhaite effacer. Nous n'utilisons aujourd'hui que le tampon d'image. Notre masque vaudra GL_COLOR_BUFFER_BIT.

Il nous faut ensuite décrire la scène 3D. La méthode de représentation d'un scène d'OpenGL est la plus classique qui soit :

- une scène est constituée d'objets.
- Un objet est défini par un ensemble de polygones.
- Un polygone est un ensemble de points de l'espace reliés entre eux par des arêtes.

Si vous êtes un de fan de Quake 3, vous savez sans doute que cette méthode de description n'est pas la seule : OpenGL autorise l'utilisation de courbes de Bézier et autres NURBS pour la description des objets. Oublions ceci pour l'instant. Pour spécifier notre scène 3D, nous allons devoir énumérer la liste des polygones. La méthode est la suivante : on indique le début de la description du polygone, on explicite chaque point du polygone, puis on indique la fin de l'énumération. A chaque déclaration de points, on peut modifier certaines variables d'état, comme la couleur active. Notre scène ne comportant qu'un polygone, la description sera vite faite.

```
glBegin(GL_POLYGON);
glColor3f(1.0,0.0,0.0);
glVertex2f(−0.5,−0.5);
glColor3f(0.0,1.0,0.0);
glVertex2f(0.5,−0.5);
glColor3f(0.0,0.0,1.0);
glVertex2f(0.5,0.5);
glColor3f(1.0,1.0,1.0);
glVertex2f(−0.5,0.5);
glEnd();
```

Chaque description de polygone commence par un appel à

```
void glBegin(GLenum mode);
```

Le paramètre mode auquel nous donnons la valeur GL_POLYGON indique la technique utilisée pour relier par des arêtes les différents points du polygone. En mode GL_POLYGON, chaque sommet et relié à son prédécesseur, et pour fermer le polygone, le dernier point est relié au premier.

La primitive de spécification d'un sommet du polygone est glVertex(). comme nous travaillons dans le plan d'équation z=0, nous utiliserons la variante à deux arguments de type GLfloat :

```
void glVertex2f(GLfloat x,GLfloat y);
```

Bien entendu, x et y sont les coordonnées cartésiennes du sommet. La commande glColor3f() modifie la couleur dite active. Son prototype est :

```
void glColor3f(GLfloat    r, GLfloat v,GLfloat b);
```

Chaque sommet d'un polygone possède une couleur, qui est la couleur active lors de la création du sommet avec glVertex(). En associant à chaque point de notre carré une couleur différente, et compte tenu du mode de remplissage de polygone proposé par défaut, notre carré sera rempli en interpolant les couleurs des sommets. En termes simples, on obtient un dégradé.

La fin de la description d'un polygone est marquée par un simple glEnd().Dans le cadre d'un cube, il faudrait réitérer 5 fois l'opération de description de polygone. Pour nous simplifier la tâche, OpenGL dispose d'un certain nombre des fonctionnalités comme des liste d'affichage.

Pour terminer notre fonction d'affichage, un glFlush() permet de s'assurer que toutes les commandes ont bien été transmises au serveur.

La fonction de rappel du clavier :

Nous avons enregistré précédemment une fonction nommée clavier() grâce à la fonction glutKeyboardFunc(). La dernière étape de la création de notre programme va consister à écrire cette fonction. Son prototype est défini par glut :

```
void clavier(unsigned char touche,int x,int y);
```

Lorsque l'utilisateur appuiera sur une touche du clavier, la boucle de gestion effectuera un appel à clavier(). La variable touche contiendra le caractère correspondant à la touche pressée, x et y indiqueront la position de la souris lors de la frappe sur le clavier. x et y sont données relativement à la fenêtre OpenGL. Dans la plupart des cas, la fonction de rappel pour les événements de type clavier est constituée d'une structure switch dont la clause de test porte sur le caractère passé en paramètre. Voici notre fonction clavier :

```
void clavier(unsigned char touche,int x,int y)
{
  switch (touche)
    {
    case 'p': /* affichage du carre plein */
      glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
      glutPostRedisplay();
      break;
    case 'f': /* affichage en mode fil de fer */
      glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
      glutPostRedisplay();
      break;
    case 's' : /* Affichage en mode sommets seuls */
      glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
      glutPostRedisplay();
      break;
    case 'q' : /*la touche 'q' permet de quitter le programme */
      exit(0);
    }
}
```

La touche 'q' est associée à la fonction exit() permettant de quitter le programme. Cette méthode n'est pas très 'propre', et nous l'améliorerons le mois prochain. Les touches 'p','f' et 's' vont permettre à l'utilisateur de choisir le mode de tracé des polygones. Pour cela, il suffit de modifier l'état de la machine OpenGL, par un appel à la fonction

```
void glPolygonMode(GLenum face,GLenum mode);
```

chaque polygone possède une face avant et une face arrière définie par l'ordre de saisie des sommets du polygone. Il est possible d'associer un mode de dessin différent pour la face avant et arrière d'un polygone. Le paramètre face indique la face dont on veut changer le mode de dessin :

- GL_FRONT : face avant
- GL_BACK : face arrière
- GL_FRONT_AND_BACK : les deux faces

Le paramètre mode définit le mode de dessin souhaité pour la ou les faces choisies :

- GL_FILL : le polygone est entièrement rempli.

- GL_LINE : seuls les contours du polygone sont dessinés (mode fil de fer)
- GL_POINT : seuls les sommets du polygone sont affichés.

En changeant la valeur d'une variable d'état, nous modifions le processus de rendu de notre image, il est donc nécessaire d'effectuer une demande de réaffichage de la scène en appelant glutPostRedisplay(), dont l'effet et de générer un événement qui va amener la boucle de gestion des événements à lancer la fonction d'affichage.

Conclusion :

Nous voilà arrivés au terme de cet article. Le code source complet du programme est fourni. Voici la commande nécessaire à la compilation.

```
gcc ogl1.c -o ogl  -L/usr/X11R6/lib -lGL -lGLU -lglut -lX11 -lXmu -lXi -lm
```

Le mois prochain, nous corrigerons quelques petits défauts du programme, notamment le phénomène de distorsion du carré lors d'un redimensionnement de la fenêtre OpenGL. Nous aborderons également les techniques de base de l'animation. D'ici là, amusez vous bien !

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
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

Code source:

```
/*********************************************************/
/*                      ogl1.c                           */
/*********************************************************/
/* Premiers pas avec OpenGL.                             */
/* Objectif : afficher a l'ecran un carre en couleur     */
/*********************************************************/

/* inclusion des fichiers d'en-tete Glut */

#include <GL/glut.h>

void affichage();
void clavier(unsigned char touche,int x,int y);

int main(int argc,char **argv)
{

  /* initialisation de glut et creation
     de la fenetre */
  glutInit(
  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowPosition(200,200);
  glutInitWindowSize(250,250);
  glutCreateWindow("ogl1");

  /* Initialisation d'OpenGL */
  glClearColor(0.0,0.0,0.0,0.0);
```

```
  glColor3f(1.0,1.0,1.0);
  glPointSize(2.0);
  /* enregistrement des fonctions de rappel */
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);


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

void affichage()
{
  /* effacement de l'image avec la couleur de fond */
  glClear(GL_COLOR_BUFFER_BIT);

  /* Dessin du polygone */
  glBegin(GL_POLYGON);
  glColor3f(1.0,0.0,0.0);
  glVertex2f(−0.5,−0.5);
  glColor3f(0.0,1.0,0.0);
  glVertex2f(0.5,−0.5);
  glColor3f(0.0,0.0,1.0);
  glVertex2f(0.5,0.5);
  glColor3f(1.0,1.0,1.0);
  glVertex2f(−0.5,0.5);
  glEnd();

  /* on force l'affichage du resultat */
  glFlush();
}

void clavier(unsigned char touche,int x,int y)
{
  switch (touche)
    {
    case 'p': /* affichage du carre plein */
      glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
      glutPostRedisplay();
      break;
    case 'f': /* affichage en mode fil de fer */
      glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
      glutPostRedisplay();
      break;
    case 's' : /* Affichage en mode sommets seuls */
      glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
      glutPostRedisplay();
      break;
    case 'q' : /*la touche 'q' permet de quitter le programme */
      exit(0);
    }
}
```