Les primitives géométriques d'OpenGl

Département/IUP informatique

Frank Singhoff

C-208

singhoff@univ-brest.fr

Sommaire

1. Introduction.
2. Gestion de la couleur
3. Primitives géométriques
4. Gestion des tampons
5. La GLUT
6. Ce qu'il faut retenir

Partie 1

Introduction

Introduction (1)

- OpenGl, c'est :
  - C'est une bibliothèque graphique 2D/3D avec une API "à état"'.
  - Conçue en 1989 (GL) par Sillicon Graphics, puis portée sur d'autres architectures en 1993 (OpenGL).
  - Avec OpenGL, on peut produire des images synthétiques sophistiquées (reflets, ombres) en temps réel si on dispose de ressources matérielles adéquates.
  - Existe sur de nombreuses architectures (NT, Unix, OS2, Amiga, etc...)
  - Nombreuses implémentations (Exemple sous NT : SGI OpenGL, 3dfx OpenGL, Mesa, Microsoft OpenGL...)

Introduction (2)

- Quelles librairies avec OpenGL ?

1. La bibliothèque OpenGL à proprement dit.
2. Des bibliothèques complémentaires :
   - L'OpenGL Utility Library (GLU), surcouche à OpenGL, fournit des fonctions plus évoluées pour la gestion des caméras ou la création de modèle avec une haute définition.
   - L'OpenGL Extension Library pour le système graphique X11 (GLX) : interopérabilité avec le protocole X11.
   - La GL Utility Toolkit (GLUT). Fournit une interface simple pour le dialogue avec le système d'exploitation.

Introduction (3)

- Types de données :

Suffixe	OpenGl	C
b	GLbyte	signed char
i	GLint	int
f	GLfloat	float
d	GLdouble	double

- Signatures de l'API :

`[xxx]service[n][y](a0, a1, a2, ..., an);`

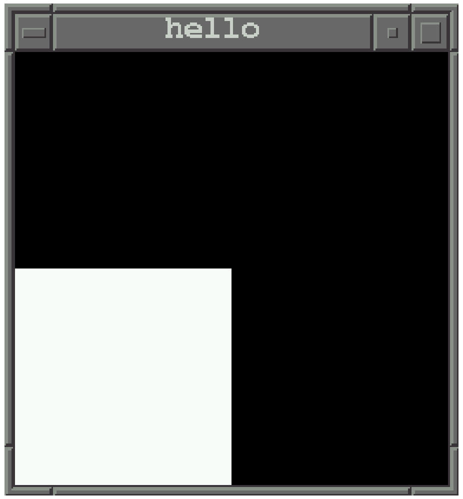
xxx pour la bibliothèque et *y* pour le suffixe.

Exemples :

```
glVertex2i(1, 3); // sommet (1,3,0)
glVertex3f(1.0, 3.0, 0.0);
```

- Manipulation des variables états : *glEnable*, *glDisable*, *glGetFloatv*, *glIntegerv*, *glBooleanv*, ...

Dessiner notre première scène



- Dessiner cette scène, c'est :

1. Vider l'écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré.

Partie 2

Gestion de la couleur

Gestion de la couleur (1)

- Chaque couleur est codée par 4 composantes qui définissent l'intensité en R(rouge), V(vert), B(bleue), A(alpha).
- Deux modes de mémorisation de la couleur des pixels :

1. Mode RVBA : le buffer mémorise directement pour chaque pixel la couleur sous la forme des composantes RVBA.
2. Mode index : le buffer mémorise pour chaque pixel un index dans une table de couleur RVBA.

- Quel mode choisir ? bitplans disponibles, nombre de couleurs nécessaires, taille tampons chromatiques, services OpenGL à utiliser.

Gestion de la couleur (2)

- *glColor* modifie la variable d'état définissant la couleur active dans le mode RVBA.
- Les valeurs de chaque composante sont comprises entre 0 et 1.
- Quelques exemples de valeurs significatives :

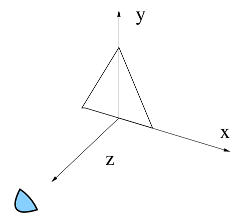
```
glColor3f(0.0, 0.0, 0.0);   // noir
glColor3f(0.0, 0.0, 1.0);   // bleu
glColor3f(0.0, 1.0, 0.0);   // vert
glColor3f(1.0, 0.0, 0.0);   // rouge
glColor3f(0.7, 0.7, 0.7);   // gris clair
glColor3f(0.3, 0.3, 0.3);   // gris fonce
glColor3f(1.0, 1.0, 1.0);   // blanc
glColor3f(1.0, 1.0, 0.0);   // jaune
glColor3f(1.0, 0.6, 0.6);   // rose
```

Partie 3

Primitives géométriques

Primitives géométriques (1)

- Comment mémoriser un volume ? un objet 3D ?

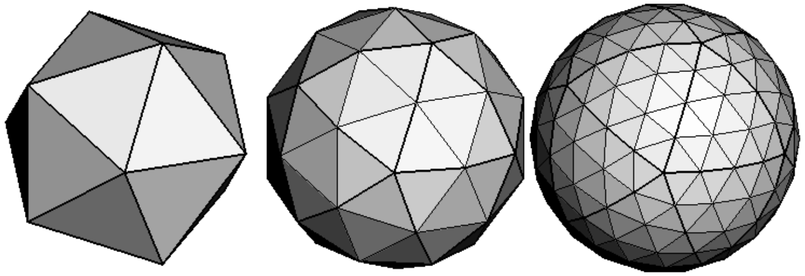


Coordonnées = (x,y,z)

- Chaque sommet est défini par un triplé (x,y,z).
- Un objet est approximé par une série (finie) de sommets calculés ou non. Les sommets peuvent être soit :
  - Des polygones (polyèdres).
  - Des courbes ou surfaces (ex : Bezier, splines).
- Nécessaire compromis entre efficacité et qualité de l'image (méthode d'approximation).

Primitives géométriques (2)

- Exemple d'approximation d'une sphère :



- Nombre de sommets et éventuellement temps de calcul de ces sommets.
- Temps de calcul de transformation (ex : translation) ou d'effet graphique (ex : lumière).
- ...

UE Multimédias et animation

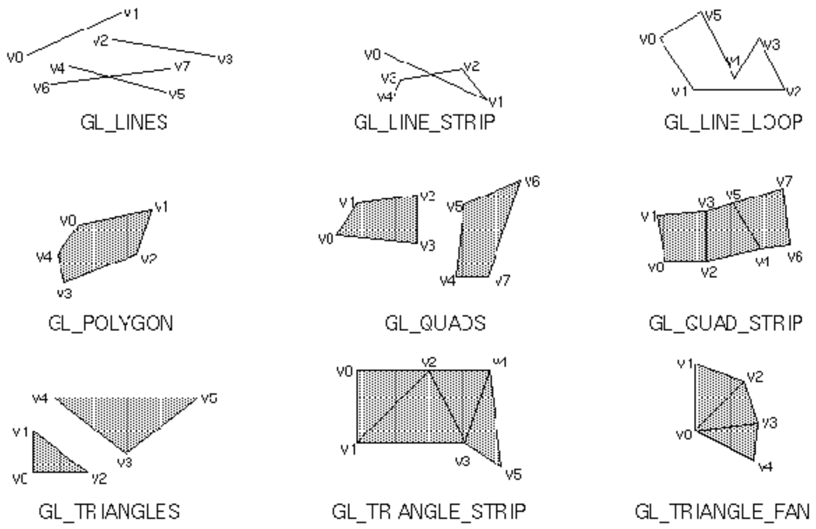
Primitives géométriques (3)

- Comment définit on un polygone :

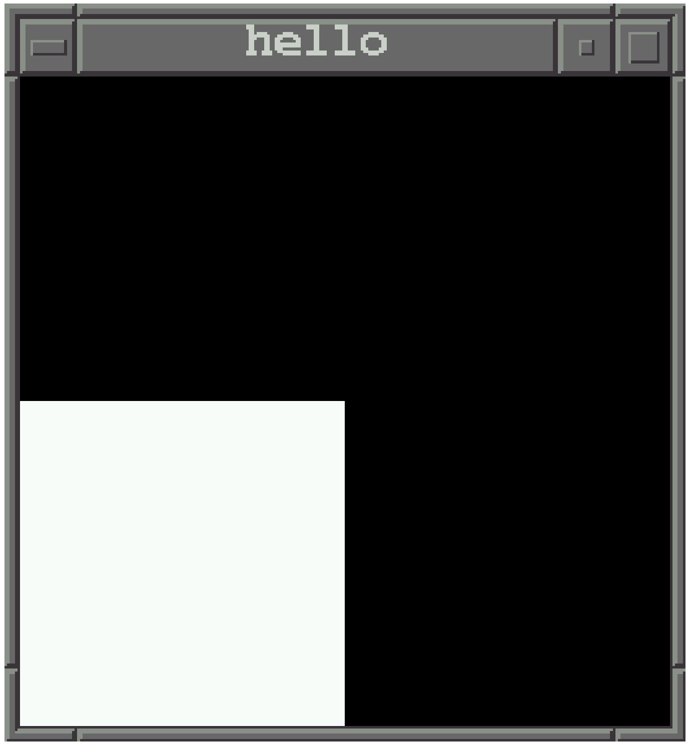
1. Par une liste ordonnée de sommets. Un sommet est défini par la fonction $glVertex*$. Une liste de sommets est définie par les fonctions $glBegin$ et $glEnd$.
2. Par une méthode de connection de ces sommets (ou primitives).

- Primitives disponibles : *GL_POINTS*, *GL_LINES*, *GL_LINE_STRIP*, *GL_TRIANGLES*, *GL_TRIANGLES_STRIP*, *GL_TRIANGLE_FAN*, *GL_QUADS*, *GL_QUADS_STRIP*, *GL_POLYGON*.

Primitives géométriques (4)



Primitives géométriques (5)



Primitives géométriques (6)

- Exemple de fonction d'affichage d'une scène :

```
/* Vide le tampon chromatique */
   glClear (GL_COLOR_BUFFER_BIT);

/* Dessine le polygone */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();
```


Partie 4

Gestion des tampons

Gestion des tampons (1)

- OpenGl dispose des tampons :

– chromatiques ($GL\_COLOR\_BUFFER\_BIT$).
– de profondeur ($GL\_DEPTH\_BUFFER\_BIT$).
– d'accumulation ($GL\_ACCUM\_BUFFER\_BIT$).
– stencil ($GL\_STENCIL\_BUFFER\_BIT$).
– auxiliaires (GL_AUXi).
en avant (GL_FRONT) et en arrière (GL_BACK).

- Principaux services :

1. Vider un tampon : $glClear$.
2. Sélectionner la valeur pour vider un tampon : $glClearColor$, $glClearDepth$, $glclearStencil$, $glClearAccum$.
3. Copier/lire/écrire : $glDrawBuffer$, $glReadBuffer$, $glCopyPixels$, ...
4. Certains traitements graphiques (ou tests) tels que le lissage, stencil, profondeur, ...

UE Multimédias et animation

Gestion des tampons (2)

- Exemple du test de profondeur :

– Le test de profondeur permet de ne pas traiter/afficher les polygones cachés par d'autres (opération de clipping).
– Il n'est pas activé par défaut. Activation par :

$$glEnable(GL\_DEPTH\_TEST)$$

– Lors de la restitution, le tampon $GL\_DEPTH\_BUFFER\_BITS$ est rempli conformément à la position des différents objets. Entre deux restitutions, ce tampon doit donc être vidé par :

$$glClear(GL\_DEPTH\_BUFFER\_BITS)$$

Partie 5

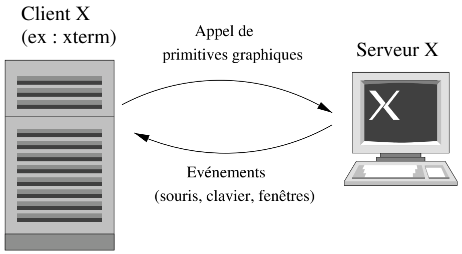
La bibliothèque GLUT

La bibliothèque GLUT

- La GLUT offre des services annexes à l'animation 3D ... mais des services nécessaires :

– Interaction avec le système de multi-fenêtrage (X11).
– Interaction avec les périphériques d'entrées (souris/claviers).
– Boucle d'événements et outils d'animation.
– Gestion de menus simplifiés.
– Quelques objets 3D simplifiés.

Interaction avec X11



- Programmation événementielle : *callbacks* + boucle d'attente des événements.
- Principales primitives OpenGl associées à X11 :
– *glutCreateWindow* : création de fenêtre.
– *glutInitWindowSize*, *glutInitWindowPosition* : positionnement de variables d'état OpenGL.
– Callbacks de traitements des événements X11 (clavier, souris, événements *Map* et *Resize*).

Boucle d'événements et d'animation (1)

- Services nécessaires pour tous types d'animations :
– *glutMainLoop* : boucle d'attente des événements.
– *gluInitDisplayMode*() : définit le mode de fonctionnement de la bibliothèque (simple/double buffering, codage des couleurs, ...).
– *glutDisplayFunc* : associe un callback aux demandes d'affichage.
– *glutPostRedisplay*() : provoque une restitution (et donc un ré-affichage).
- L'animation peut être implantée de diverses manières.

1. Animation avec tampon simple. *glFlush* pour forcer la terminaison de la restitution.
2. Animation avec tampon double. *glutSwapBuffers* pour forcer la terminaison de la restitution et échanger les tampons.

Boucle d'événements et d'animation (2)

• Généralement, les restitutions sont déclenchées sur occurrences d'événements et exécution de leur callback respectif. Pour ce faire, les *callbacks* invoquent *glutPostRedisplay* pour provoquer la restitution.

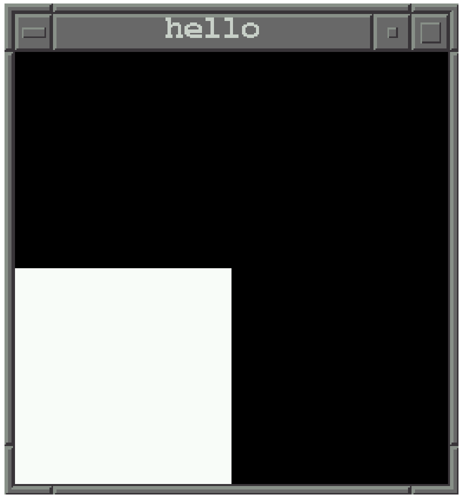
• Exemple de callbacks :

1. Restitution sur clavier/souris : $glutMouseFunc$ ou $glutKeyboardFunc$ enregistrent les callbacks qui vont traiter les événements produits lors de l'utilisation de la souris ou du clavier.

2. Restitution sur timer : $glutTimer$ enregistre le callback associé au timer. Le callback est invoqué à chaque fois que le compteur du timer atteint zéro.

3. Restitution par tâche de fond : $glutIdle$ enregistre un callback invoqué lorsque plus aucun événement n'est en attente.

Boucle d'événements et d'animation (3)



Boucle d'événements et d'animation (4)

- Il s'agit ici de *simple buffering*, remplacer $glFlush$ par $glutSwapBuffers$ pour du *double buffering*.

```
#include <stdlib.h>
#include <GL/glut.h>
/*
  Dessine la scène, peut être invoqué a
  différents moments pour différentes raisons
*/
void display(void)
{
/* Vider le tampon */
   glClear (GL_COLOR_BUFFER_BIT);

/* Dessiner le polygone */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();

/* Forcer le rendu maintenant */
   glFlush ();
}
```


Boucle d'événements et d'animation (5)

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

/*
  Initialisation de la scène (variables d'état OpenGl)
*/
void init (void)
{
/* Choisir la couleur d'effacement */
   glClearColor (0.0, 0.0, 0.0, 0.0);

/* Selectionner la matrice de projection */
   glMatrixMode(GL_PROJECTION);

/* Initilialise la matrice de projection */
   glLoadIdentity();

/* Clipping de cadrage */
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

Boucle d'événements et d'animation (6)

- Restitution déclenchée par le clavier.

```
/* Callback clavier */
static void key(unsigned char k, int x, int y) {
    ...
  glutPostRedisplay();
}

int main(int argc, char** argv) {
/* Initialisation de la GLUT */
   glutInit(&argc, argv);

/* Choix du mode d'affichage */
/* GLUT_SIMPLE pour simple buffering,
/* GLUT_DOUBLE pour double buffering */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

/* Création fenêtre et initialisation de la scène */
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();

/* Connexion callbacks puis boucle d'événements */
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```


UE Multimédias et animation

Boucle d'événements et d'animation (7)

- Restitution déclenchée par timer :

```
/* Callback timer */
static void mon_timer(int x)
{
    ...
  glutPostRedisplay();
}

int main(int argc, char** argv)
{
/* Initialisation de la GLUT */
   glutInit(&argc, argv);

/* Choix du mode d'affichage */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

/* Création fenêtre et initialisation de la scène */
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();

/* Connexion callbacks puis boucle d'événements */
   glutDisplayFunc(display);
   glutTimerFunc(40, my_timer, 1);
   glutMainLoop();
}
```

Boucle d'événements et d'animation (8)

- Il ne doit pas y avoir d'instruction de restitution hors du callback d'affichage. Appel à *glutPostRedisplay* pour forcer la restitution.
- Le callback d'affichage doit être idempotent (ne doit pas modifier l'état d'OpenGl).
- Ne pas activer de callback inutilement (performances).
- Attention aux calculs redondants (performances).

UE Multimédias et animation

Objets 3D de la GLUT

- Sphère pleine ou fil-de-fer :

```
void glutSolidSphere(GLdouble radius,
GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius,
GLint slices, GLint stacks);
```

- Cube plein ou fil-de-fer :

```
void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);
```

- Autres objets : Cône, Tore, Tétrahedron, Dodécahedron, Icosahedron, Tasse à thé.

Ce qu'il faut retenir

- Comment décrire un objet par un ensemble de polygones ?
- Comment manipuler des couleurs ?
- Programmation événementielle et animation.