Travaux dirigés d'infographie n°2

Cours de synthèse d'images I

—IMAC première année—

Algorithme de dessin : objets canoniques

Lors de cette scéance, nous étudierons la notion d'objets canoniques.

▸ Exercice 1. *Tracés de droites*

Pour tracer une droite en **openGL**, *on utilise les instructions suivantes :*

```
/* Dessin de segments de droites */
glBegin(GL_LINES);
  glVertex2f(deb_x, deb_y); /* Le point de depart */
  glVertex2f(fin_x, fin_y); /* Le point d'arrivee */
  /* On peut ajouter d'autres couples de points et
     utiliser des tests et des boucles entre
     le glBegin() et le glEnd() */
glEnd();
```

L'instruction `glBegin(GL_LINE_STRIP)` *permet de dessiner une ligne brisée et une ligne qui "boucle" sur elle même peut être affichée avec* `glBegin(GL_LINE_LOOP)`.

Écrire une application **glut** *qui dessine des lignes brisées dont les vertex sont détérminés par des clic successifs dans la fenêtre (la ligne doit être redessinée lorsque la fenêtre est raffraichie). Pour stocquer les points, vous utiliserez un tableau de* `point2f`*:*

```
typedef struct{
  float x;
  float y;
}point2f;
```

Lorsque l'on clique sur le bouton de droite, le point cliqué devient le dernier de la ligne courante qui boucle sur elle même.

▸ Exercice 2. *Carré et cercle canoniques*

Écrire les fonctions `dessine_carre()` *et* `dessine_cercle()` *qui dessinent, respectivement, un carré de coté 1 et un cercle de rayon 1, centrés sur l'origine. Le cercle est déssiné*

en utilisant `DISCRETISATION_CERCLE` *segments de droite.* `DISCRETISATION_CERCLE` *est une constante du pré-compilateur.*

Affichez ces deux objets dans votre fenêtre **glut**.

▸ Exercice 3. *Manipuler les objets canoniques*

En l'état, les objets canoniques de la question précédente ont un intérêt limité. Toutefois **openGL** *va nous permettre de les "transformer" à notre guise. Les opérations de transformations d'*__openGL__ *sont liés à la matrice* `GL_MODELVIEW` *(quelle sont ses dimensions pour un objet à deux dimensions ? à trois dimensions ?). On utilise l'instruction* `glMatrixMode(GL_MODELVIEW)` *pour signifier à* **openGL** *que les opérations matricielles ultérieures se feront sur la matrice de transformation. Il y a bien sûr d'autres matrices, et en particulier, la matrice de projection* `GL_PROJECTION`.
Commençons par charger la matrice identité dans la matrice GL_MODELVIEW à l'aide de la fonction `glLoadIdentity()` *(fixe la matrice courante à la matrice identité). Une translation s'effectue à l'aide de la fonction* `glTranslatef(x,y,z)`. *Une rotation dans le plan* $(x0y)$ *à l'aide de* `glRotatef(alpha,0.0,0.0,1.0)` *où* `alpha` *est l'angle de rotation en degrés.*

Pour appliquer les transformations, **openGL** *multiplie simplement les coordonnées des vertex par la matrice GL_MODELVIEW.*

- *En utilisant la fonction* `dessine_cercle()`*, non modifiée, afficher un cercle centré en* $(1, 2)$.
- *En utilisant la fonction* `dessine_carre()`*, non modifiée, afficher le carré canonique ayant subi une rotation de* $45°$ *puis une translation de 1 unité sur l'axe des* x.
- *En utilisant la fonction* `dessine_carre()`*, non modifiée, afficher le carré canonique ayant subi une translation de 1 unité sur l'axe des* x *puis une rotation de* $45°$.
- *Allons nous utiliser une structure de données pour représenter les objets canoniques ?*
- *Implanter les comportements suivants dans votre programme :*
  - *En cliquant dans la fenêtre, on déplace le centre du carré sous le curseur.*
  - *Avec le bouton droit maintenu enfoncé et en déplaçant la souris, on effectue une rotation du carré autour de son centre.*
  - *Le cercle est lancé dans une direction quelconque et "rebondit" indéfiniment sur les bords.*