OpenGL - TD 02

Dessin de base :
Objets canoniques et transformations

Lors de cette séance, nous aborderons l'un des concepts les plus simples de la modélisation en 2D (et en 3D) : La construction d'objets canoniques.

Exercice 01 – Après les points ...

Lors du précédent TP, vous avez appris à dessiner des points à l'aide de ces instructions :

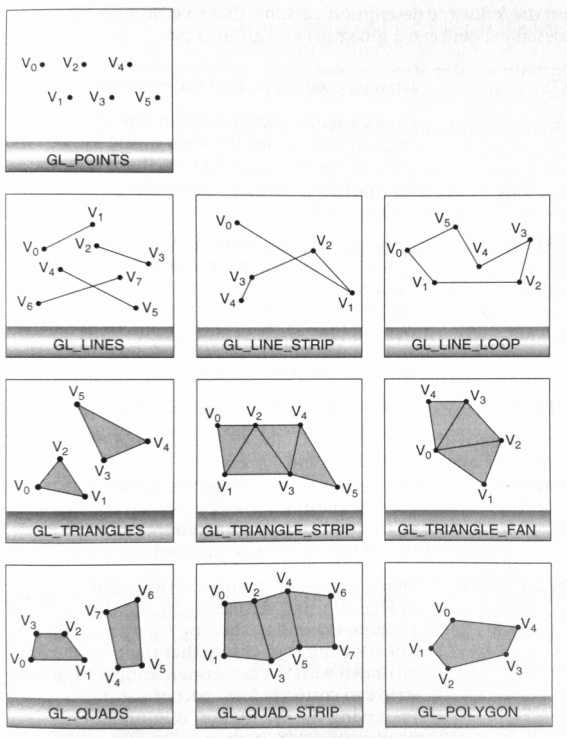
```
glBegin(GL_POINTS);
    glVertex2f(x, y);
glEnd();
```

Entre `glBegin()` et `glEnd()`, il est possible d'utiliser :
- `glColor3f` (ou autres variantes de `glColor`)
- `glVertex2f` (ou autres variantes de `glVertex`)
- tout ce que vous avez appris en C (boucle, tests, etc)

Les autres fonctions OpenGL ou SDL ne sont pas acceptées.

L'ensemble des **vertices** (sommets en français) que vous créez entre ces deux instructions forme la liste des sommets qui formeront la primitive à afficher.
Par exemple, si vous utilisez `GL_POINTS` en paramètre de `glBegin`, alors chaque appel à `glVertex` dessinera un unique point.

Comme vous avez pu le voir dans le TD 01, il existe d'autres valeurs que `GL_POINTS`.
Voici la liste des primitives :
- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN
- GL_QUADS
- GL_QUAD_STRIP
- GL_POLYGON



Note :
Dans le cas où il y aurait un nombre insuffisant de vertex, la primitive demandée ne sera pas dessinée.

A faire :

01. Pour vous échauffer, faites une application qui dessine une ligne brisée par clics successifs dans la fenêtre. Il sera nécessaire de stocker tous les points et d'afficher entièrement la ligne à chaque dessin de la fenêtre.
Vous pouvez utiliser un tableau, ou mieux une liste de structures de points. (c.f. TD 01 ex 06)

02. Faites en sorte que lorsqu'on clique avec le bouton droit de la souris, le point cliqué devienne le dernier de la ligne, qui doit alors boucler sur elle même.
(i.e. le dernier point est relié au premier)

Exercice 02 – Des objets canoniques …

Votre maîtrise des primitives fraîchement acquise, vous allez maintenant pouvoir créer des objets plus complexes. Ces objets sont de taille unitaire, et centrés sur l'origine.

A faire :
Implémentez les fonctions suivantes permettant de dessiner des objets canoniques :

01. `drawOrigin()`

Dessine les axes à l'origine, c'est à a dire :
- Un segment de taille 1 et de couleur rouge sur l'axe horizontal (axe des *x*)
- Un segment de taille 1 et de couleur verte sur l'axe vertical (axe des *y*)

01. `drawSquare()`

Dessine un carré de coté 1, centré sur l'origine.

03. `drawCircle()`

Dessine un cercle de diamètre 1, centré sur l'origine.
Pour dessiner le cercle, vous devrez faire approximer ce dernier à l'aide d'un nombre donné de segments de droite. Ce nombre est à définir en tant que constante dans votre code.

04. Pour finir, réalisez l'affichage de ces objets canoniques dans une fenêtre SDL.

Rappel :
Tout dessin doit se faire entre `glClear` et `SDL_GL_SwapBuffers` .

Optionnel :
05. Ajouter un paramètre `int full` aux fonctions précédentes.
Ce paramètre permettra choisir si l'objet est à dessiner plein ou seulement en contour :
- si `full` vaut 0, afficher le contour
- si `full` vaut 1, remplir le polygone

Exercice 03 – Transformations !

Les deux objets canoniques définis dans l'exercice 02 sont peu intéressants en l'état… Cependant OpenGL à la particularité de pouvoir déplacer et/ou déformer les objets qu'il dessinera par la suite. Cela se fait en modifiant la matrice de transformation d'OpenGL.

Pour modifier cette matrice, il faut utiliser l'instruction suivante :

```
glMatrixMode(GL_MODELVIEW);
```

Cette ligne permet de dire que la matrice courante (celle que l'on va modifier) est la matrice de transformation (il y en a d'autres comme la matrice de projection `GL_PROJECTION`).

Pour demander le chargement de la matrice identité, on utilise :

```
glLoadIdentity();
```

Elle fixe la matrice courante à la matrice identité.

On peut effectuer une translation à l'aide de :

```
glTranslatef(x, y, z);
```

(`x` , `y` et `z` doivent être des flottants)

On peut effectuer une rotation dans le plan ($x0y$) (i.e. autour de l'axe z) à l'aide de :

```
glRotatef(alpha, 0., 0., 1.);
```

(`alpha` est l'angle de rotation en degrés)

On peut également effectuer une mise à l'échelle à l'aide de :

```
glScalef(x, y, z);
```

(`x` , `y` et `z` doivent être des flottants)

Note :
La matrice de transformation `GL_MODELVIEW` s'applique à tous les points que l'on transmet à la carte graphique par la suite (i.e. jusqu'à nouvel ordre).

A faire :

01. Modifiez la taille de votre fenêtre virtuelle pour quelle fasse 8 unités en *x*, 6 unités en *y* et soit toujours centrée en (0, 0), via `gluOrtho2D()`.

02. Dessinez le repère dans votre fenêtre.

03. Modifiez la fonction d'affichage pour afficher un cercle orange centré en (1, 2)
(Bien entendu, vous ne devez pas modifier `drawCircle`)

04. Changez le code d'affichage pour ajouter un carré rouge mais après qu'il ait subi une rotation de 45°, puis une translation sur l'axe des *x*.

05 Permutez la rotation et la rotation pour dessiner un carré violet supplémentaire.
Relevez la différence de positions entre ce carré violet et le carré rouge dessiné en 04.

Question :

06. Expliquez la différence de positions entre les carrés rouge et violet, dessinés en 04 et 05a.
Laquelle des deux approches vous semble la plus adaptée pour dessiner un objet en OpenGL ?

A faire :

07. Faites en sorte qu'en cliquant dans la fenêtre, on déplace le centre d'un carré jaune, situé initialement en (0, 0), à l'emplacement du clic.

Optionnel :

08. Faites en sorte qu'en maintenant le bouton droit de la souris, on puisse effectuer une rotation du carré jaune (par rapport a son centre) en déplaçant le curseur.
(Plus simplement, vous pouvez aussi faire tourner le carré jaune à une vitesse constante en maintenant appuyé un bouton ou une touche du clavier)

09. Faites en sorte qu'un cercle bleu se déplace aléatoirement dans la fenêtre, en permanence.