Initiation à OpenGL et SDL

L'objectif de cette matière est de vous initier à OpenGL (en 2D, cette année).
Ce premier TP va également vous permettre de prendre en main la SDL afin de créer une fenêtre dans laquelle un rendu OpenGL pourra être effectué. Nous verrons également comment intégrer la gestion des événements de type clavier, souris, etc…

Exercice 01 – Késako

Avant de commencer à travailler, il serait bon de savoir quels outils nous utilisons.

Questions :
À l'aide d'internet, définissez rapidement :
01. Qu'est-ce qu'OpenGL ?
02. Qu'est-ce que la SDL ?

Exercice 02 – Votre première fenêtre

Maintenant que vous savez ce que sont OpenGL et SDL, il est temps de s'en servir.
Cet exercice ne vous demandera pas d'écrire de code, il vise à vous montrer la manière dont nous allons travailler, notamment via l'outil de compilation **make**.

L'archive TD01.zip est disponible sur le site du chargé de TD. (A définir)

Le fichier *minimal.c* , situé dans le répertoire *TD01/doc/*, constitue la base de code sur laquelle nous travaillerons avec OpenGL et SDL.
Vous trouverez également le fichier *makefile*, à la racine du répertoire *TD01/* .

Questions :
01. Que signifient les directives `#include` au début de *minimal.c* ?
02. Quelles sont les bibliothèques utilisées pour OpenGL et SDL ?
03. Comment sont elles appelées à la compilation dans le *makefile* ?

Le fichier source pour ce programme est donné dans *TD01/src/ex02/* .
(Il s'agit d'une simple copie de *minimal.c*)

Pour compiler à partir du makefile, ouvrez un terminal dans *TD01/*, et entrez la commande :

```
make ex02
```

Le programme *td01_ex02.out* est crée dans le répertoire *TD01/bin/* .
Pour l'exécuter , entrez la commande (toujours à partir de *TD01/*) :

```
bin/td01_ex02.out
```

Notice – Organisation d'un répertoire de TD

En faisant l'exercice 02, vous avez pu prendre en main la compilation et l'exécution d'un programme utilisant OpenGL et SDL.
Vous aurez sans doute remarqué que les fichiers de travail se trouvent à différents endroits. (Et peut être même remarqué l'apparition d'un répertoire *TD01/obj/*)

Un répertoire de TD est pensé comme un objet indépendant, vous pourrez l'extraire du .zip sur votre machine la où vous le souhaitez, et travailler immédiatement dedans (ou presque).

Le répertoire pour chaque TD sera divisé en trois sous-répertoires :
- *bin/* : les exécutables crées via **make** seront placés ici
- *doc/* : contient divers fichiers de référence, notamment *minimal.c* et l'énoncé du TD
- *src/ :* contient les fichiers sources sur lesquels vous travaillez pour chaque exercice
- *obj/* : contient les fichiers **.o* générés lors de la compilation

Avant de démarrer un nouvel exercice, vous aurez besoin de faire ceci :
- créer un répertoire pour votre exercice dans */src*, comprenant un nouveau fichier source
- mettre a jour le *makefile* du TD, pour qu'il puisse trouver ce fichier et le compiler

A faire :
01. Créez le répertoire *TD1/src/ex03/*

```
mkdir -p src/ex03
```

02. Copiez le fichier *minimal.c,* et renommez le en *td01_ex03.c*
(Une autre option est de repartir du fichier source de l'exercice précédent)

```
cp doc/minimal.c src/ex03/td01_ex03.c
cp src/ex02/td01_ex02.c src/ex03/td01_ex03.c
```

03a. Ouvrez *TD1/makefile*

```
gedit makefile
```

03b. Ajoutez les lignes suivantes, dans la partie `# Fichiers TD 01` :

```
# Fichiers exercice 03
OBJ_TD01_EX03= ex03/td01_ex03.o
EXEC_TD01_EX03= td01_ex03.out
```

Ajoutez les lignes suivantes, dans la partie `# Regles compilation TD 01` :

```
ex03 : $(OBJDIR)$(OBJ_TD01_EX03)
    $(CC) $(CFLAGS) $(OBJDIR)$(OBJ_TD01_EX03) -o $(BINDIR)$(EXEC_TD01_EX03)
$(LDFLAGS)
```

Vous êtes désormais en mesure de compiler les fichiers pour l'exercice 03 et de générer l'exécutable *td01_ex03.out* dans le répertoire *TD01/bin*, en entrant simplement :

```
make ex03
```

Exercice 03 – Domptage de la fenêtre

La fonction `SDL_CreateWindow` permet d'ouvrir une fenêtre et de fixer ses paramètres. Elle à la signature suivante :

```
SDL_Window* SDL_CreateWindow(
      const char* title,
      int x, int y, int w, int h,
      Uint32 flags);
```

Les paramètres `w` et `h` permettent de fixer les dimensions de la fenêtre.

Le paramètre `flags` est un champ de bits permettant d'énumérer divers paramètres à l'aide de l'opérateur `|` de la manière suivante : `param1 | param2 | … | param_n` .

Vous pourrez trouver sur cette page la liste des paramètres pouvant être placés dans `flags` : https://wiki.libsdl.org/SDL_CreateWindow

La fonction `SDL_WM_SetWindowTitle` permet quand à elle de modifier le titre de la fenêtre : https://wiki.libsdl.org/SDL_SetWindowTitle

A faire :
À partir du fichier *minimal.c* et des fonctions ci-dessus :

01. Créez une fenêtre redimensionnable (`SDL_WINDOW_RESIZABLE`) compatible OpenGL, de taille 400x400 et ayant pour titre : TD 01 Ex03 .

Pour pouvoir dessiner une scène dans une fenêtre, il est nécessaire d'indiquer à OpenGL quelle zone de la scène sera visible via cette fenetre.
Lorsqu'on voudra dessiner un point, il faudra fournir à OpenGL les coordonées de ce point devront etre passées dans le repère virtuel de la scène. (qui peut s'exprimer en 3D)
OpenGL ira ensuite convertir ces coordonnées dans un repère 2D lors de l'afficage à l'écran.

Cette étape de conversion entre l'espace virtuel de la scène et l'écran s'appelle la **projection**.
Une fois que ces coordonées ont été obtenues, OpenGL peut travailler pour déterminer quels pixels devront etre coloriés. (Etape connue sous le nom de **rasterization**, à retenir.)

Pour indiquer la taille de l'espace de que vous souhaitez représenter dans une fenetre, nous utiliserons une variable constante statique `GL_VIEW_SIZE` .

```
/* Espace fenetre virtuelle */
static const float GL_VIEW_SIZE = 1.;

// Nous representons ici les points situés en -0.5 et 0.5 en x et en y
```

A faire :

Nous allons également ajouter la fonction `onWindowResized()` pour garantir le bon fonctionnement de la projection tout au long de l'exécution.

02a. Ajoutez la fonction `onWindowResized()` dans votre code :

```
static const float aspectRatio;

void onWindowResized(unsigned int width, unsigned int height)
{
    aspectRatio = width / (float) height;

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if( aspectRatio > 1)
    {
        gluOrtho2D(
        -GL_VIEW_SIZE / 2. * aspectRatio, GL_VIEW_SIZE / 2. * aspectRatio,
        -GL_VIEW_SIZE / 2., GL_VIEW_SIZE / 2.);
    }
    else
    {
        gluOrtho2D(
        -GL_VIEW_SIZE / 2., GL_VIEW_SIZE / 2.,
        -GL_VIEW_SIZE / 2. / aspectRatio, GL_VIEW_SIZE / 2. / aspectRatio);
    }

}
```

Le viewport définit la taille en pixels de la fenêtre réelle.
Puis on passe en mode projection, et après avoir chargé la matrice identité, on recrée la projection orthogonale qui identifie coordonnées virtuelles et coordonnées pixels.

02b. Faites le nécessaire pour que la fonction `reshape()` soit appelée à chaque redimensionnement de la fenêtre. (événement de type `SDL_WINDOWEVENT`)
Les nouvelles dimensions de la fenetre seront dans `e.window.data1` et `e.window.data2` .

https://wiki.libsdl.org/SDL_WindowEvent

02c. Pour finir, pensez aussi à appeler `onWindowResized()` après la création de la fenêtre.

```
onWindowResized(WINDOW_WIDTH, WINDOW_HEIGHT);
```

Exercice 04 – Des événements …

La SDL permet une gestion poussée et très libre des événements utilisateur.
Lorsqu'elle détecte un événement, elle place dans une file interne une structure de type `SDL_Event` décrivant l'événement.
Il est nécessaire de traiter tous les événements reçus à chaque tour de la boucle d'affichage.

La fonction `int SDL_PollEvent(SDL_Event *event)` permet de défiler un événement stocké dans la file interne. Elle renvoie 0 si il n'y a aucun événement a défiler.
Sinon, elle renvoie 1, et remplit `*event` avec les informations concernant l'événement.

La structure `SDL_Event` est détaillée sur la page suivante :
https://wiki.libsdl.org/SDL_Event

Comme on peut le voir en haut de la page, ce n'est pas une structure mais en fait une union.
Le champs `type` indique le type de l'événement reçu.
Il indique par quel champ devra être manipulé l'union par la suite.

Par exemple, si `e.type` vaut `SDL_KEYDOWN` ou `SDL_KEYUP`, on doit accéder au champ `e.key` pour connaître les propriétés de l'événement.
Par contre si `e.type` vaut `SDL_SDL_MOUSEBUTTONUP` il faut accéder à `e.button`.

A faire :
01. Modifiez le programme pour qu'il s'arrête lorsque l'utilisateur appuie sur la touche `Q`.
Vous pourrez ainsi quitter le programme proprement, même en plein écran. (c.f. exercice 03)

02. Faites en sorte que lorsque l'utilisateur clique en position (x, y), la fenêtre soit redessinée avec la couleur (rouge = x mod 255, vert = y mod 255, bleu = 0) au prochain tour de la boucle d'affichage, plus précisément au prochain appel de `glClear(GL_COLOR_BUFFER_BIT)`.

Il faut utiliser la fonction `glClearColor(r, g, b, a)` pour modifier la couleur de remplissage / nettoyage de la fenêtre.
`r`, `g`, `b` et `a` doivent être compris entre 0 et 1, il faut donc diviser les valeurs `r`, `g`, `b` entre 0 et 255 par 255.0, avant de les passer à OpenGL. Pour `a` il suffit de passer 1.

Attention :
Assurez vous que votre division implique au moins un nombre flottant, sans quoi vous effectuerez une division euclidienne qui vous renverra 0, et non la valeur décimale voulue.
Utilisez l'opérateur de cast `()` pour changer le type d'une expression. (ex : `(float) x`)

A faire :
03. Faites de même mais lorsque la souris bouge (événement `SDL_MOUSEMOTION`).
Pour distinguer du précédent événement, faites en sorte que la fenêtre soit redessinée avec la couleur (rouge = (x / largeur_fenetre, vert = 0, bleu = y / hauteur_fenetre).

Note :
Vous pouvez récupérer la largeur et hauteur de la fenêtre via la structure `SDL_Surface`.