Introduction à OpenGL/C++

Malek.Bengougam@gmail.com

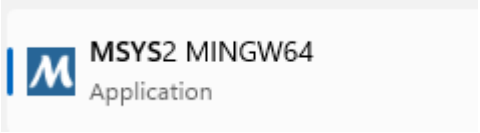
1. Installation d'un environnement de développement

Pour cette première introduction à OpenGL en C++ nous allons devoir installer un environnement de développement ainsi que quelques bibliothèques fondamentales.

Note : si vous êtes sous MacOS ou Linux vous pouvez sauter cette étape.
De même, si vous utilisez un IDE tel que Visual Studio (Code), CLion, etc… et que vous êtes habitué à configurer des projets dans ces environnements vous pouvez également passer cette étape.

1.a. Nous allons utiliser l'environnement MSYS afin de simuler un terminal Unix et ainsi pouvoir compiler en ligne de commande.
Allez sur https://www.msys2.org et téléchargez l'installer commençant par msys2-x86_64_***.exe.

1.b. Après l'installation, exécutez MSYS2 en version MinGW64 (ou alternativement UCRT 64-bit)



A ce stade vous arrivez sur le prompt correspondant à votre répertoire « utilisateur » (représenté par le symbole tilde '~') situé dans le répertoire d'installation de MSYS (par ex. MSys64/home/malek).

1.c. Nous allons ensuite utiliser un gestionnaire de paquet afin de récupérer les outils nécessaires.
Tapez :

```
pacman -Syu
pacman -S base-devel gcc
pacman -S mingw-w64-x86_64-toolchain
g++ -v
```

1.d. Il nous faut maintenant vérifier le bon fonctionnement des compilateurs et la présence de la bibliothèque de base (sous Windows : opengl32.lib).

Tapez : `nano testgl.cpp`
Puis recopiez le code suivant -faites 'CTRL+o' pour sauvegarder, puis 'CTRL+x' pour quitter l'éditeur.

```
#include <GL/gl.h> // remplacez par <OpenGL/gl.h> et <OpenGL/OpenGL.h> sous MacOS
int main(void)
{
    glClearColor(1.f, 1.f, 0.f, 1.f);
    return 1;
}
```

Finalement, tapez :
(Sous Linux, remplacez `-lopengl32` par `-lGL`, et par `-framework OpenGL` sous MacOS)

```
g++ -o testgl testgl.cpp -lopengl32
```

2. installation des bibliothèques nécessaires

Afin de pouvoir utiliser OpenGL, nous devons avoir une surface de dessin (canvas, ou framebuffer). Celui-ci est généralement fourni par une fenêtre du système d'exploitation mais la configuration est assez complexe. Nous allons utiliser quelques bibliothèques afin de nous faciliter le travail. Pour l'heure, dans le cadre de cette introduction, nous allons utiliser GLUT (ou plutôt FreeGLUT) et plus tard nous verrons GLFW.

MSys2 : `pacman -S mingw-w64-x86_64-freeglut` *(ajoutez -ucrt si vous utilisez msys-ucrt64)*
MacOS : `brew install freeglut`
Linux : `apt install freeglut3-dev`

Il faudra désormais lier (link) notre programme avec `-lfreeglut` (`-framework glut` sous MacOS).

```
g++ -o testgl testgl.cpp -lopengl32 -lfreeglut
```

Voici un exemple de programme OpenGL + GLUT qui affiche un triangle après avoir effacé l'écran

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

void Render()
{
    glClearColor(1.f, 1.f, 0.f, 1.f);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.5f);
    glVertex2f(0.5f, -0.5f);
    glVertex2f(-0.5f, -0.5f);
    glEnd();
}

// la 'callback' executee par glutDisplayFunc()
void Display()
{
    Render();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowSize(960,540);
    glutCreateWindow("Triangle");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

2.a. Commentez `glClear(GL_COLOR_BUFFER_BIT);` Que se passe-t-il ?

2.b. Remplacez **GLUT_SINGLE** par **GLUT_DOUBLE**. Quel est le changement ?

2.c. Remplacer maintenant **glFlush**() par **glutSwapBuffers**(). Que constatez-vous maintenant ?

L'option GLUT_DOUBLE va être notre mode d'affichage par défaut. Cette option active le « double-buffer » qui consiste à présenter un buffer (dit « front-buffer ») à l'utilisateur tandis que l'on prépare l'image suivante en dehors de la fenêtre (dans le « back-buffer »). La fonction **glutSwapBuffers**() est définie par GLUT, elle demande à l'OS d'effectuer le basculement entre les buffers front et back.
Ajoutez également glutPostRedisplay() après glutSwapBuffers() afin de forcer le rafraichissement.

Annexe : Compilation en ligne de commande, exemple avec GCC et G++

‘`-c <file>`’ permet de spécifier le fichier source à compiler (les extensions sont généralement .c, .cpp). Cependant cette option ne s’arrête qu’à la génération d’un fichier objet et n’effectue pas le lien entre objet permettant de générer l’exécutable.
On peut modifier le nom du fichier objet à l’aide de ‘`-o <file>`’ (extensions .o ou .obj selon le compilateur). A noter que si l’option ‘-c’ est manquante ‘-o’ donne donc le nom de l’exécutable final.

trivia : lorsque ni ‘-c’ ni ‘-o’ ne sont spécifiés, la génération produit un fichier exécutable nommé ‘a.out’ cf. https://fr.wikipedia.org/wiki/A.out

Attention, les options sont **sensibles à la casse** ! Par exemple ‘`-O`‘ suivi d’un chiffre ou parfois d’une lettre permet d’indiquer le niveau d’optimisation souhaité.

‘`-W<warn>`’ permet de contrôler le niveau des avertissements de compilation, on utilise généralement ‘Wall’ où ‘all’ indique le niveau maximum.

‘`-D`’ permet de définir des macros (équivalent de #define) mais au moment de la compilation. Cela peut être utilise pour produire des options de compilations customisées.

‘`-I<dir>`’ spécifie un chemin vers un répertoire contenant des entêtes (includes, extension .h ou .hpp). Attention il n’y a pas d’espace après le ‘-I’. Chaque chemin doit être spécifié dans une entrée distincte.

‘`-L<dir>`’ spécifie un chemin vers un répertoire contenant des bibliothèques (libraries).

‘`-l <lib>`’ (en minuscule donc) indique le nom de la bibliothèque qui doit être liée au programme (extension .a ou .lib généralement). Si le nom de la bibliothèque commence par ‘lib’ il est alors possible de n’indiquer qu’une partie du nom dans la ligne de commande. Exemple : libpthread.a on spécifie ‘-lpthread’.

Note : pour créer une bibliothèque statique on doit utiliser le linker. Certains compilateurs comme GCC utilisent un outil spécifique : ‘.a’ étant le diminutif de ‘archive’, l’outil se nomme **ar**.

‘`-std=<version>`’ permet d’indiquer la version du C ou C++ à utiliser lors de la compilation.

Note : La plupart de ces options sont disponibles avec plus ou moins la même syntaxe sous d’autres compilateurs comme *clang*, *Intel icc*, *Microsoft C/C++ cl* (à noter que Visual Studio supporte également clang) etc…