Compilation d'un projet OpenGL

mercredi 9 mars 2022 20:05

Dans tous les cas : copiez les fichiers GLShader.h et GLShader.cpp dans un répertoire « common » à la racine de votre répertoire principal.

Comme son nom l'indique, ce répertoire va contenir les fichiers partagés entre plusieurs projets.

1. Compilation sous Linux

1.a en ligne de commande

Il est plus aisé d'installer les packages avec Aptitude :

```
sudo apt-get install libglfw3 libglfw3-dev
sudo apt-get install libglew2.1 libglew2.1-dev
```

Puis pour compiler nos projets :

```
g++ -o OpenGL_101 OpenGL_101.cpp ../common/GLShader.cpp -lglfw -lGLEW -lGL -ldl
```

1.b avec un cmake (sous cLion par ex, ou cmake)

Cf. Chapitre 4

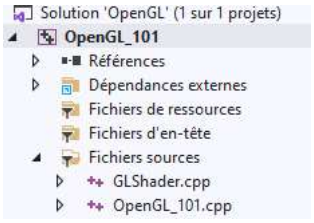
2. Compilation sous Windows

Récupérez les **binaires** précompilés sur les sites https://www.glfw.org/download.html pour GLFW et https://glew.sourceforge.net/index.html pour GLEW.

2.a Visual Studio 20xx

Après avoir créé une première solution, que vous nommerez par ex : OpenGL, et un premier projet de type « C++ » et « Console », que vous nommerez OpenGL_101 par exemple, vous devez ensuite créer un répertoire « common » et « libs » à la racine de la solution (le répertoire « OpenGL »).

Ajoutez également le fichier GLShader.cpp dans la liste des fichiers sources à compiler :



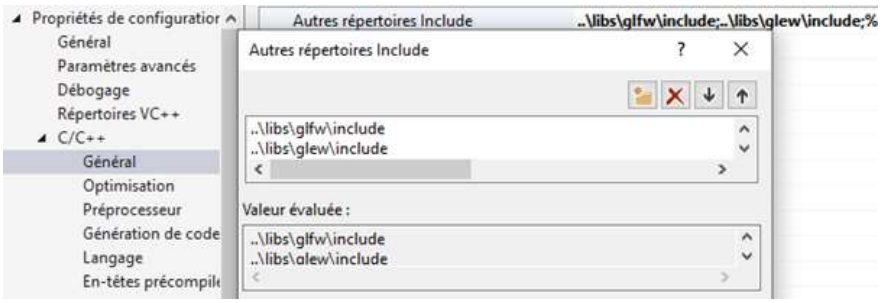
Veillez à bien sélectionner l'architecture « x64 »

Configuration des chemins

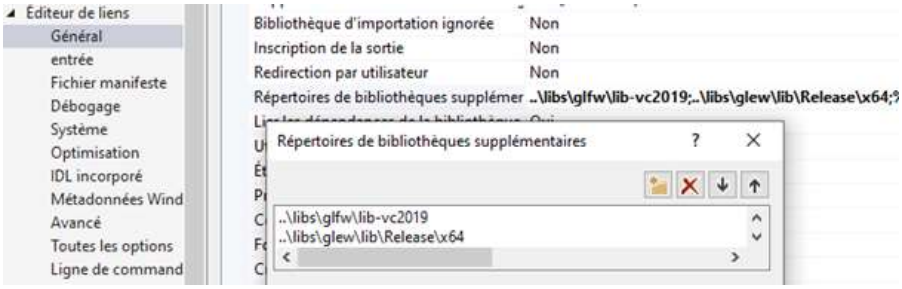
Faites un clic-droit sur le projet, puis sélectionnez « propriété » qui se trouve tout en bas du menu contextuel.

Assurez-vous que la plateforme sélectionnée dans la fenêtre de propriétés est bien « x64 » :

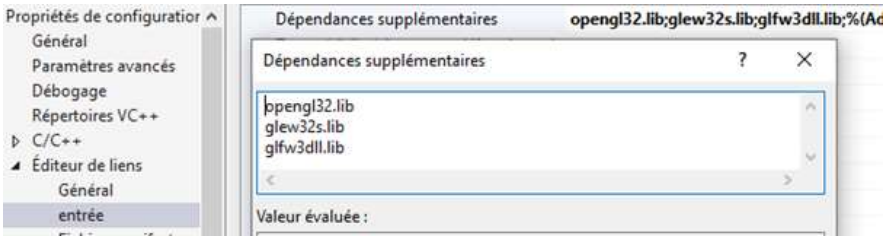
Il nous faut maintenant définir les chemins vers les entêtes (includes) :



Puis vers les bibliothèques (lib) …



… et finalement spécifier le nom des bibliothèques



Une fois l'exécutable créé -il se trouvera à la racine de la solution dans le répertoire x64/Debug- il faudra

copier les DLL de nos bibliothèques (glfw3.dll et glew32.dll) à côté de l'exécutable.

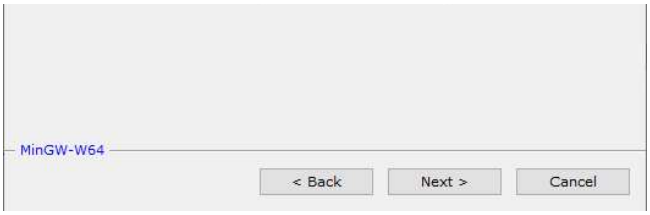
1.b Editeur de texte + Mingw-w64

Mingw-w64 met à disposition un environnement de programmation en ligne de commande inspiré de GNU Linux. La distribution du compilateur est donc gcc/g++.

Télécharger et installer Mingw-w64 depuis sourceforge, voir le lien sur

https://www.mingw-w64.org/downloads/#mingw-builds .

A l'installation, choisir « **x86_64** » pour **l'architecture** et «**posix** » comme bibliothèque de **thread**.



Lancez l'environnement en exécutant le script mingw-w64.bat.

La compilation s'effectue comme sous Linux, cependant il est nécessaire de spécifier les chemins des bibliothèques dans la ligne de commande :

```
 g++ -o OpenGL_101 –I../libs/GLFW/include -I../libs/GLEW/include -
L../libs/GLFW/lib/lib-mingw-w64 –L../libs/GLEW/lib/Release/x64 OpenGL_101.cpp
../common/GLShader.cpp -lglfw3dll -lglew32s -lopengl32
```

3. Compilation sous MacOS

Récupérez les **binaires** précompilés MacOS de GLFW sur https://www.glfw.org/download.html.

Alternativement, vous pouvez installer GLFW3 en utilisant Homebrew depuis la ligne de commande.

MacOS utilise Apple-clang comme compilateur, qui est un alias de gcc/g++.

Il faut noter que ce système d'exploitation est architecturé autour de la notion de **bundle** ou **framework** qui sont plus utilisés que les simples bibliothèques.

MacOS est disponible sur un nombre limité de hardware, Apple expose directement les extensions nécessaires à chaque GPU. Il n'est donc pas nécessaire d'installer GLEW.

Il est cependant nécessaire d'inclure les entêtes suivants avant d'inclure <GLFW/glfw3.h> :

```
#include <OpenGL/gl.h>
#include <OpenGL/OpenGL.h>
```

3.a compilation via le Terminal

```
g++ -o OpenGL_101 OpenGL_101.cpp ../common/GLShader.cpp -lglfw -framework Cocoa -
framework OpenGL -framework IOKit
```

3.b avec cmake

Cf. Chapitre 4

3.c avec xCode

Todo

4 CMake

Il est également possible d'utiliser CMake afin de pouvoir compiler facilement sur de multiples plateformes.

Il est nécessaire d'installer CMake mais il se trouve que CMake est également l'outil de génération par défaut de CLion ou encore Android Studio.

Le cœur de cmake est le fichier CMakeLists.txt qui contient des directives permettant de spécifier le nom de l'exécutable, les chemins des répertoires d'entêtes (headers/includes) ou des bibliothèques (libraries). Voici un exemple de CMakeLists.txt équivalent à ce que l'on a vu précédemment :

```
cmake_minimum_required(VERSION 3.21)
#Le nom de l'executable final ainsi que du projet
project(OpenGL_101)
```

```
#indique le type de compilateur (ici C++) ainsi que la version (C++14)
set(CMAKE_CXX_STANDARD 14)


#directive qui permet de recuperer automatiquement les chemins OpenGL
find_package(OpenGL REQUIRED)
link_directories(${OPENGL_gl_LIBRARY})


#les repertoires des header files (GLEW pas utile sous MacOS)
include_directories(../libs/glfw/include)
include_directories(../libs/glew/include)


#les repertoires des libraries (ici, pas nécessaire sous Linux et MacOS)
link_directories(../libs/glfw/lib-static-ucrt)
link_directories(../libs/glew/lib/Release/x64)

#on specifie l'ensemble des fichiers cpp a compiler et lier ensemble
add_executable(OpenGL_101 OpenGL_101.cpp ../common/GLShader.cpp)


#finalement la liste des bibliotheques a lier (link)
target_link_libraries(OpenGL_101 glfw3 ${OPENGL_gl_LIBRARY} glew32)
```

En fonction de la plateforme, il faut spécifier glfw à la place de glfw3 en link, et omettre glew32 (Il serait préférable d'utiliser find_package() également).

Il est recommandé de créer un sous-répertoire (à partir de celui des fichiers cpp ou CMakeLists.txt) et d'effectuer la génération du projet dans ce sous-répertoire.

Par exemple, en ligne de commande cela donnerait ceci :

```
mkdir build
cd build
cmake ..
```

Alternativement, on peut également spécifier le répertoire de génération avec l'option -B

```
cmake -S . -B build
```

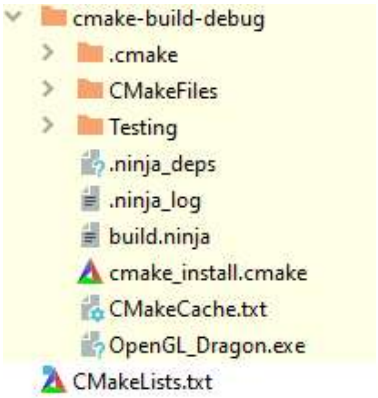
Le générateur de projet peut être spécifié explicitement avec l'option -G parmi les différents compilateurs et environnements de développement, cf.
https://cmake.org/cmake/help/latest/manual/cmake-generators.7.html

Par défaut il s'agit de make, il suffit donc de saisir make à partir du répertoire de build.

```
make
make install
```

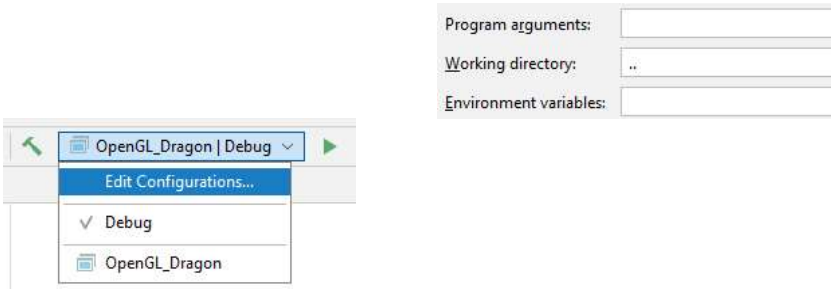
4.a avec CLion

CLion construit également un projet en utilisant une arborescence similaire. Par exemple, le projet « OpenGL_Dragon » en configuration « Debug » produit l'arborescence suivante avec un CMakeLists.txt similaire au précédent :



Notez que « cmake-build-debug » est un sous-répertoire, la racine étant le répertoire du CMakeLists.txt. Remarquez également que l'exécutable final se trouve dans ce sous-répertoire.

Il est préférable de ne pas copier les fichiers de données dans ce sous-répertoire mais plutôt d'indiquer à CLion quel doit être le répertoire courant (*current working directory, CWD*). Pour cela il suffit d'éditer la configuration et d'indiquer « .. », le chemin de référence étant le répertoire de l'exécutable (et donc cmake-debug-build ici).


Program arguments:
Working directory:
..
Environment variables:
OpenGL_Dragon | Debug
Edit Configurations...
Debug
OpenGL_Dragon