OpenGL - TD 01

Initiation au rendu 3D et GLFW

L'objectif de cette première partie des TP est de vous initier au rendu 3D (en cachant toutes les spécificités d'OpenGL que nous verrons l'année prochaine). Ce premier TP va surtout vous permettre de prendre en main GLFW, un système de fenêtrage et d'Interface Homme Machine (**IHM**), 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 (chatGPT), déterminez rapidement :

01. Qu'est-ce qu'OpenGL ?
02. Qu'est-ce que GLFW ?

Exercice 02 – Votre première fenêtre

Maintenant que vous savez ce que sont OpenGL et GLFW, 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 **cmake** et le template de code associé.

Le « template » de code du TD est disponible sur l'espace d'elearning ou sur le site de votre chargé de TD. Il est constitué d'un certain nombre de fichiers que vous reconnaîtrez :

- *README.md* est un fichier classique présentant un projet informatique et, souvent, les instructions pour le compiler, l'installer ou l'utiliser. Vous y trouverez les informations nécessaires de compilation notamment.
- *CMakeLists.txt* est un fichier de configuration pour **cmake**.

Plus important, les dossiers de ce template sont les suivants :

- *assets* : contiendra les données, essentiellement les images que nous utiliserons mais aussi les shaders que nous verrons l'année prochaine
- *bin* : contiendra les exécutables
- *third_party* : contiendra les bibliothèques utiles
- *src* : est un simple dossier de configuration
- *TDXX* : contiendra vos exercices pour chaque TD

Évidemment dans ces TD, nous travaillerons avec OpenGL (qui doit être installé sur votre machine) et GLFW (qui sera compilé directement). Une fois **cmake** exécuté par vscode, il créera un dossier *build*. Sinon vous pouvez faire tout le processus à la main avec les commandes suivantes dans un terminal à l'instar de Mr Robot (c'est évidemment ma version préférée car vous maîtrisez tout le processus) :

```
mkdir build
cd build
cmake ../
make
```

Le **cmake** est configuré pour rechercher dans les répertoires *TD0X* les fichiers cpp commençant par les lettres ‘ex’. Ainsi chaque fichier source *ex*.cpp* (* désigne un ensemble de lettres) créera un exécutable correspondant nommé *TD0X_ex*.cpp* qui sera situé dans le répertoire *bin*. Pour le moment, il n’existe qu’un seul répertoire *TD01* contenant un fichier *ex02.cpp* donc lorsque vous compilerez, vous devriez obtenir l’exécutable *TD01_ex02* créé dans le répertoire *bin/*.

Pour exécuter le code créé, utilisez Vscode ou, directement dans un terminal, entrez la commande (à partir du répertoire source) :

```
bin/TD01_ex02
```

01. Compilez le fichier *ex_02.cpp* (de la manière que vous souhaitez)
02. Lancez l’exécutable correspondant. Pour le moment, vous ne devriez voir qu’une simple fenêtre rectangulaire présentant un fond bordeaux.

Note – Faire d’autres exercices puis d’autres TD

Pour faire les exercices suivants de ce TD il suffira de créer de nouveaux fichiers qui doivent impérativement commencer par *ex*. Ainsi, par exemple, pour réaliser l’exercice 3, vous pouvez copier le fichier *ex02.cpp* en *ex03.cpp*. La compilation générera un exécutable appelé *TD01_ex03*

Lorsque vous changerez de TD, il vous suffira de dupliquer le répertoire *TD01* en *TD02*. Vous pouvez retirer les fichiers .cpp précédent mais il faut absolument conserver le fichier *CMakeLists.txt* . Dans ce nouveau répertoire, tout fichier commençant par ‘*ex*’ générera un exécutable (comme pour le *TD01* donc).

Il est possible de ne compiler que les exercices d’un TD grâce à la commande

```
make TD01
```

C’est disponible également dans les options de build de VSCode (à côté de la roue dentée par exemple).

Exercice 03 – Le domptage de la fenêtre

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

```
GLFWwindow* glfwCreateWindow (int width, int height,
            const char *      title,
            GLFWmonitor *     monitor,
            GLFWwindow *      share);
```

Les paramètres `width` et `height` permettent de fixer les dimensions de la fenêtre. Le second `title` est simplement le titre de la fenêtre. Les deux autres paramètres n'ont pas d'importance pour le moment (monitor sert à créer un fenêtre *fullscreen* et le dernier à partager un contexte avec d'autres fenêtres).

A faire :

01. Faites le nécessaire pour créer l'exercice *ex03.cpp* comme vu dans la partie « Note » précédemment. Puis, grâce la fonction ci-dessus, créez une fenêtre compatible OpenGL, de taille 800x800 et ayant pour titre : « TD 01 Ex 03 ».

Pour pouvoir dessiner une scène 2D ou 3D dans une fenêtre (ou une sous partie de celle-ci), il est nécessaire d'indiquer à OpenGL l'espace des coordonnées du monde virtuel visible depuis cette fenêtre (ou partie de fenêtre). Ainsi, lorsque l'on voudra dessiner un point, il faudra fournir à OpenGL les coordonnées de ce point dans la scène virtuelle, puis OpenGL « convertira » dans le repère virtuel de notre fenêtre (qui lui est en fait fixe).

Cette étape de conversion entre l'espace virtuel de la scène et celui de la fenêtre s'appelle la **projection**. Une fois que ces coordonnées (fenêtre) ont été obtenues, OpenGL peut travailler pour déterminer quels pixels devront être coloriés (une étape connue sous le nom de **rasterization**).

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

```
/* Espace virtuel */
static const float GL_VIEW_SIZE = 1.;
// L'univers 2D visible a une taille de 1.0 en x et en y
```

Évidemment, lorsque la fenêtre est redimensionnée, il faut modifier l'opération de projection évoquée précédemment. Nous allons également donc ajouter une fonction `onWindowResized()` pour garantir le bon fonctionnement de la projection tout au long de l'exécution.

A faire :

02a. Ajouter les lignes vues ci-dessus dans votre code (où à votre avis?). Puis ajoutez la fonction `onWindowResized()` ci dessous également :

```
void onWindowResized(GLFWwindow* /*window*/, int width, int height)
{
      aspectRatio = width / (float) height;
      glViewport(0, 0, width, height);
      if( aspectRatio > 1.0)
```

```
	{
		myEngine.set2DProjection(-GL_VIEW_SIZE * aspectRatio/ 2.,
			GL_VIEW_SIZE * aspectRatio / 2. ,
			-GL_VIEW_SIZE / 2., GL_VIEW_SIZE / 2.);
	}
	else
	{
		myEngine.set2DProjection(-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 (sous) fenêtre où sera dessinée réellement notre scène OpenGL. Puis on va définir, suivant la taille de la fenêtre (`aspectRatio`) les coordonnées de l'univers visible en 2D grâce à une projection orthogonale 2D. La fonction `set2DProjection` permet en effet de définir la projection donc les coordonnées visibles depuis la fenêtre. Étudiez bien cette fonction pour comprendre ce qui se passe en particulier lorsque l'on crée une fenêtre rectangulaire horizontale ou verticale : Finalement quelles sont les coordonnées visibles de l'espace virtuel 2D ?

02b. Pour faire en sorte que cette fonction soit appelée à chaque redimensionnement de fenêtre, nous allons demander à ce que cette fonction soit appelée dans ce cas grâce à l'instruction GLFW `glfwSetWindowSizeCallback` (voir documentation !) où le premier argument représente votre fenêtre, et le second le nom de votre fonction soit `onWindowResized`. Nous verrons dans la question suivante comment ce mécanisme opère dans GLFW. Ajoutez l'appel à la fonction `glfwSetWindowSizeCallback` juste après la création de la fenêtre.

02c. Pour finir, pensez aussi à appeler `onWindowResized()` après la création de la fenêtre et avant la boucle *while* :

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

Note – Le contexte d'une fenêtre (OpenGL)

Seulement pour les curieux.ses. On a parlé (très) brièvement de la notion de contexte (sur le dernier argument de la fonction de création de fenêtre). Lorsqu'une application de type IHM crée une fenêtre, celle-ci doit posséder un certain nombre d'état « graphique » qui sont soit demandés par l'utilisateur (je veux utiliser OpenGL 4.0, je veux de la transparence…) soit imposés par le serveur graphique de la machine. Autrement dit le contexte dépend à la fois de l'application demandée, mais également des possibilités de la machine.

Ainsi vous pourriez peut être demander telle ou telle version d'OpenGL mais si GLFW ne le peut pas, il ne pourra créer le contexte et donc ne pourra créer la fenêtre.

Pour les (encore plus) curieux, vous pouvez voir un certain nombre de ces états (appelés « hint » dans GLFW) à cet endroit. Comme vous le verrez cela va de la version d'OpenGL à si la fenêtre doit être redimmensionnable ou pas. Notez qu'il faut fixer ces « hints » **avant** la création de la fenêtre.

Exercice 04 – Des événements …

La gestion des évènements (utilisateurs ou pas) est un point très important des IHM. Les bibliothèques créant des fenêtres comme GLFW - mais également (en vrac) la SDL 1 et 2, SFML ou encore GLUT – ont globalement **deux types de stratégie** au regard de la gestion des évènements.

La première stratégie, plus flexible et probablement un peu plus efficace, est de gérer l'ensemble des évènements sous forme de liste à traiter à chaque instant. Lorsque l'application – ou plus exactement la bibliothèque – détecte un événement, elle le place dans une file interne avec des informations décrivant l'événement. Ce comportement est utilisé par exemple par les bibliothèque SDL ou SFML. Nous n'aborderons donc pas cette technique ici mais sachez qu'elle nécessite de traiter tous les événements reçus à chaque tour de la boucle d'affichage.

La seconde stratégie est une technique dite de *callback*, beaucoup plus simple à appréhender. Dans cette technique, le programmeur implémente, pour chaque type d'évènement auquel il souhaite réagir, une fonction dite de *callback*. Puis il demande à la bibliothèque de gestion des évènements (ici GLFW) d'appeler cette fonction à chaque fois que l'évènement intervient. Cette fonction sera donc appelée à chaque fois qu'un évènement de ce type sera déclenché par l'utilisateur. Pour chaque type d'évènement, la fonction de *callback* doit avoir une signature spécifique. C'est dans cette signature que l'on retrouvera toutes les informations importantes liées à l'évènement.

Ça paraît flou ? Peut-être, mais vous l'avez déjà fait ! Dans l'exercice 3, nous avons défini une fonction `onWindowResized` avec une signature particulière (notamment deux paramètres qui définissent la nouvelle taille de la fenêtre) et nous avons demander à GLFW de l'appeler à chaque fois que la fenêtre est redimensionnée via la fonction `glfwSetWindowSizeCallback`. Notez que nous gérons également les erreurs GLFW via une fonction de callback (saurez vous deviner laquelle?).

A faire :

01. Créez le nécessaire pour l'exercice 4 comme vu précédemment (dans la partie Notice)

02. Modifiez le programme pour qu'il s'arrête (utilisez `glfwSetWindowShouldClose`) lorsque l'utilisateur appuie sur la touche Q. Pour ce faire, trouver la fonction d'enregistrement de callback (et donc la signature de la fonction de callback) qui vous intéresse dans la documentation sur les évènements.

Note 1 : Préférez ici les *key input* au lieu des *text input*.

Note 2 : La signature propose 4 arguments concernant la touche pressée. Le premier de ceux-ci indique la touche pressée (la liste possible est ici mais attention **elle concerne un clavier US**), le second est un code unique pour la touche dépendant du clavier physique, le troisième indique l'action réalisée sur la touche, alors que le dernier vous indique si certaine touche sont pressées (en même temps que la touche qui a déclenché l'action) comme *Shift*, *Control* ou *Alt*.

Note 3 : Si vous souhaitez avoir directement la 'lettre' de la touche pressée au regard du clavier que vous utilisez, vous pouvez utiliser la fonction `glfwGetKeyName`.

Pour les plus chevroné.es, essayez maintenant de regarder la documentation de GLFW pour mettre votre application en plein écran. L'appui sur la touche Q est alors la seule façon 'propre' de quitter l'application.

03. Faites en sorte que lorsque l'utilisateur clique en position (x, y), la fenêtre soit redessinée avec la couleur (rouge = x mod 256, vert = y mod 256, bleu = 0) au prochain tour de la boucle d'affichage, plus précisément au prochain appel de `glClear(GL_COLOR_BUFFER_BIT)`. Notez que vous pouvez récupérer la position du curseur grâce à `glfwGetCursorPos`, définie à partir du coin supérieur gauche (qui a pour coordonnées 0,0 du coup).

Pour changer la couleur de fond, il faut utiliser la fonction `glClearColor(r, g, b, a)` qui définit (l'état de) la couleur de remplissage de la fenêtre. Ici `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`)

04. Faites de même mais lorsque la souris bouge (trouver la fonction de callback idoïne). 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 fonction `glfwGetWindowSize`.

05. Vous allez maintenant proposer deux modes de visualisation, l'un qui permet de réaliser les effets lumineux de l'exercice précédent, et l'autre qui va proposer de définir la couleur du fond d'écran « manuellement ». Concernant le « mode de visualisation », pour faire simple, vous pouvez employer une variable globale. « Mappez » une touche pour passer d'un mode de visualisation à un autre.

Le second mode de visualisation consiste, là encore, à exploiter les événements claviers. Au départ, vous considérez que la couleur de fond (un triplet R,G,B que vous pouvez, là encore, passer en variable globale) est noire. A l'appui de la touche 'R' la couleur rouge doit augmenter progressivement vers 1. De même pour la couleur verte et bleue pour, respectivement les touches 'G' et 'B'. Les « augmentations » de couleur s'arrêtent dès que la touche correspondante est relâchée.

06. Permettez, que si la touche 'shift' est enfoncée, on descende la couleur au lieu de l'augmenter.