INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Informatique Graphique - TP 1

Démarrage

GRAINDORGE Amance

VERNET Hector

Université de Rennes

I. INTRODUCTION

Le but de ce premier TP est d'installer la pipeline fournie par les encadrants et de commencer à s'en servir. Cette pipeline utilise OpenGL comme API, et SFML pour gérer les interactions avec l'utilisateur (fenêtre, entrées clavier). On ira jusqu'à afficher un cube coloré et lui appliquer des transformations.

II. TUTORIELS

II.1. Tutoriels 1 et 2

Les tutoriels 1 et 2 ont simplement pour but de pouvoir build et exécuter le code fourni. Étant tous les deux sur Arch Linux, cela nous a demandé d'apporter quelques modifications au projet de base.

Nous avons dû passer la version de CMake à 3.28 (la dernière) dans tous les `CMakeLists.txt` du projet. Il a également fallu remplacer toutes les directives ayant pour but de trouver les librairies sur le système par des one-liners comme :

```
find_package(SFML 2.6 REQUIRED COMPONENTS graphics) # Exemple pour SFML
```

Et quelques autres changements de syntaxe que vous pouvez retrouver dans ce commit : https://github.com/pixup1/tpinfographique/commit/719ea31629623d5dd193b0f1222b5710582b9d3c.

Après avoir apporté ces modifications, nous pouvons lancer l'exécutable `practical1` et obtenir une fenêtre vide :

Figure 1 : Fenêtre vide

II.2. Tutoriel 3

On va commencer par afficher une « frame », trois lignes dans l'espace 3d. On écrit notre code dans `sampleProject/practical1.cpp` :

```
// Ajout des shaders defaultVertex et defaultFragment
std::string vShader = "./../../sfmlGraphicsPipeline/shaders/defaultVertex.glsl";
std::string fShader = "./../../sfmlGraphicsPipeline/shaders/defaultFragment.glsl";
ShaderProgramPtr defaultShader = std::make_shared<ShaderProgram>(vShader, fShader);
viewer.addShaderProgram(defaultShader);

// Ajout du Renderable "frame"
FrameRenderablePtr frame = std::make_shared<FrameRenderable>(defaultShader);
viewer.addRenderable(frame);
```

Et voici le résultat :

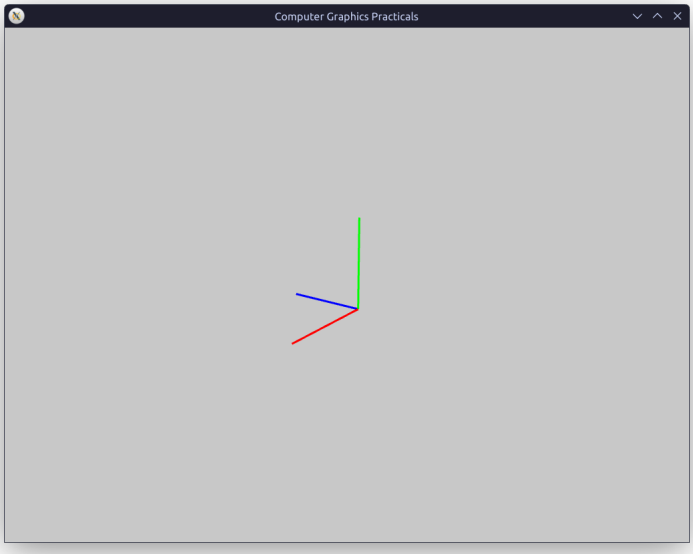

Figure 2 : Lignes

II.3. Tutoriel 4

Dans ce tutoriel, on ajoute un triangle à la scène de la même manière que pour la `frame` :

```
// Ajout des shaders flatVertex et flatFragment
vShader = "./../../sfmlGraphicsPipeline/shaders/flatVertex.glsl";
fShader = "./../../sfmlGraphicsPipeline/shaders/flatFragment.glsl";
ShaderProgramPtr flatShader = std::make_shared<ShaderProgram>(vShader, fShader);
viewer.addShaderProgram(flatShader);

// Ajout du Renderable "cube" (qui sera pour le moment un simple triangle)
CubeRenderablePtr cube = std::make_shared<CubeRenderable>(flatShader);
viewer.addRenderable(cube);
```

On décommente aussi les lignes suivantes :

```
m_positions.push_back( glm::vec3 (-1 ,0 ,0) );
m_positions.push_back( glm::vec3 (1 ,0 ,0) );
m_positions.push_back( glm::vec3 (0 ,1 ,0) );
```

de `sfmlGraphicsPipeline/src/CubeRenderable.cpp` pour ajouter les trois points du triangle. On obtient bien un triangle :

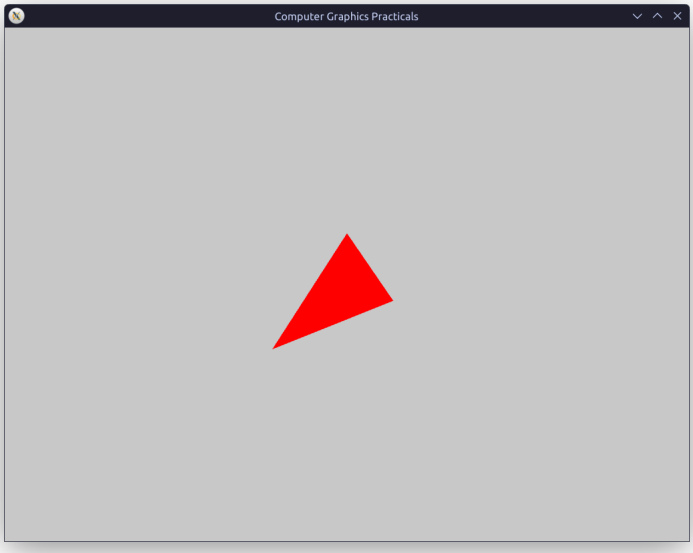

Figure 3 : Triangle

III. EXERCICE 1

IV. EXERCICE 2

V. EXERCICE 3

VI. EXERCICE 4

VII. CONCLUSION