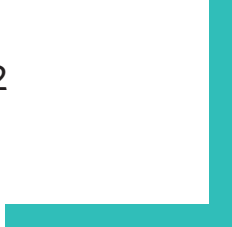




SOLAR SYTEM

OpenGL - Florian Zobèle - IMAC 2



MODE D'EMPLOI

Changer de caméra : touche C

Accélérer : touche S

Avec la caméra de profil : clic gauche ^ pour tourner autour du Soleil

Quitter l'application : Echap

ARCHITECTURE

Pour cette application, j'ai fait le choix d'essayer d'automatiser au maximum les choses afin de ne pas dupliquer de code.

La première étape a été de configurer le projet afin de pouvoir compiler le projet, et ce tout le long du projet. Lors du projet PacMan, la compilation ne fonctionnait pas, ce qui avait fait perdre un temps monstrueux et qui ne m'avais pas permis de coder le jeu.

Ensuite, j'ai fait des classes pour automatiser les VAO et VBO.

Le fichier main est le point d'entrée de l'application. Il est très court et ne fait qu'initialiser l'app et la lancer. Le fichier App va s'occuper de détecter les events et de lancer la boucle de rendus. Les événements sont alors ensuite envoyés jusqu'à un state controller qui permet de changer le state de l'application. Ici, il y a qu'un state jeu mais on pourrait imaginer un state menu par exemple.

Le fichier State.hpp déclare des fonctions communes à tous les states de l'application.

La classe fill PlayState implémente ces fonctions notamment de rendu et d'événements clavier et souris.

Le PlayState possède également une scène, c'est dans celle-ci que seront rendus les planètes et satellites.

La scène s'occupe dans un premier temps d'initialiser les planètes, puis ensuite les satellites. Elle initialise ensuite les différentes caméras. Après avoir mis les planètes et satellites dans des listes, elle fait un rendu de celles-ci.

Chaque planète et chaque satellite dérivent de la classe Object qui possède les éléments communs aux deux éléments comme les matrices, le program avec les shaders, une sphere et une texture.

Avant le début du développement, j'ai longtemps réfléchi à l'organisation des fichiers et comment séparer moteur de rendu et moteur de jeu. Cela m'a pris du temps. J'en suis venu ensuite à faire ce système de state abstraits qui gèrent les événements.

De même pour les caméras, j'ai fait une classe Caméra et mes Trackball caméras étendent de celle-ci. Si j'avais eu besoin d'un FreFly caméra, il aurait été simple de l'implémenter.

La classe planète, tout comme satellite, est très simple. elle se contente seulement d'initialiser la planète et implémente la fonction de rendu. Le rendu d'une planète se passe ainsi. On bouge la planète autour du soleil, on l'anime (elle tourne sur elle-même), puis on la dessine.

Dans toute l'app, j'ai essayé de découper au maximum le code afin de restreindre le code à ce qu'il fait vraiment. pas question de tout mettre dans le main.

Concernant les shaders, ce sont les mêmes que ceux utilisés pendant les TPs.

ORGANISATION

Pour m'aider dans la réalisation de ce projet, j'ai utilisé Github. Etant tout seul, je n'ai pas fait de planning et je codais quand j'en avais l'envie et le temps en me donnant un objectif à chaque session de code.

LES RÉSULTATS

Je suis plutôt satisfait du résultat obtenu même s'il manque des éléments demandés. Au moins, cette fois-ci, le projet compile et ressemble à un système solaire.