Rapport de projet OpenGL

HYPERLOOP

Élèves :

Maylis Teyssendier De La Serve

Hugo Baltz

Maxime Siret

Table des matières

I. Introduction

Dans le cadre du module informatique des étudiants en deuxième année à l'ENSG, nous avons eu à réaliser un projet de programmation avec la bibliothèque graphique OpenGL.

Ce projet s'inscrit dans la continuité des projets des années précédentes et a pour but de visualiser en trois dimensions (3D) une scène représentant un hyperloop en action, c'est-à-dire ici un ou plusieurs hyperloop en mouvement sur des rails (avec un paysage et un décor).

Pour ce faire, nous avons composé notre groupe de projet constitué de trois étudiants : Maylis Teyssendier De La Serve, Hugo Baltz et Maxime Siret.

Le graphe supposé non orienté et l'interface de programmation ont été fournis dans les données du projet.

II. Présentation d'OpenGL

1. Description

La bibliothèque graphique OpenGL contient des primitives graphiques de base et des fonctions permettant de pouvoir mettre en place et gérer la projection voulue.

De plus, nous pouvons disposer d'une version améliorée de la bibliothèque : GLU (OpenGL Utility). Cette version contient des fonctions avancées permettant de gérer les points de vue (de dessus, à l'intérieur, etc…) et dispose de primitives géométriques plus complexes et sophistiquées.

Dans les deux cas (les deux versions), il va falloir y ajouter un contexte de dessin. Nous allons donc avoir besoin d'une couche supplémentaire qui va gérer les interactions avec l'utilisateur et les fenêtres d'affichage : GLUT (OpenGL Utility Toolkit).

Concernant les langages, OpenGL est assez souple, et nous avons décidé de l'utiliser via C++ compilé avec GNU.

2. Projection

Il y a deux types de projections majeures disponibles avec OpenGL : la projection orthométrique (perspective cavalière) ou la projection à point de fuite (comme notre vue).

La première est conçue pour fonctionner correctement avec des représentations d'objets purement géométriques (en 2D). Cependant, elle n'est pas très intuitive en 3D car elle ne correspond pas à notre habitude de vision (cela semble logique).

Ainsi, nous avons utilisé la deuxième projection, la projection à point de fuite qui est notamment abondamment utilisée en jeux vidéo, d'autant plus que la bibliothèque GLU rend son utilisation plus facile par la présence de fonctions associées.

III. Manuel d'installation

- Copier le dossier fourni en veillant à bien laisser l'architecture comme elle est.
- Installer CodeBlocks :

Comment installer CodeBlocks + OpenGL

==========

Sous windows

Il faut:

- Installer CodeBlocks comme d'hab.

-Télécharger glut32-bin.zip sur le site de Nate Robin's
http://user.xmission.com/~nate/glut/glut-3.7.6-bin.zip

- Le dezipper

+ Copier glut.def et glut32.lib dans CodeBlocks/MinGW/lib

+ Copier glut.h dans CodeBlocks/MinGW/include/GL

+ Copier glut32.dll dans CodeBlocks/MinGW/bin

Sous linux

Il suffit de suivre les instructions sur http://singhgurjot.wordpress.com/2012/05/17/how-to-install-openglglut-libraries-in-ubuntu-12-04/

- Afin d'installer les librairies nécessaire au bon fonctionnement, Copier-coller dans « C:\Program Files (x86)\CodeBlocks » le contenu du répertoire « data\a_copier » dans le répertoire MinGW de CodeBlocks.
- Penser à bien ajouter les linker suivant à votre compiler dans Settings/Compiler/Linker Settings dans l'ordre suivant : mingw32, SDL, SDLmain et SDL_image
- Lancer le fichier src\trainGl.cbp dans CodeBlocks.
- Il ne vous reste plus qu'à build et à lancer le programme

IV. Manuel d'utilisation

Il faut tout d'abord lancer l'exécutable «trainGl.exe». Afin de déplacer la caméra il faut utiliser les touches Z, Q, S et D du clavier. Elle est aussi déplaçable en bougeant la souris. Pour changer le mode de vue de la caméra, il suffit d'utiliser la touche B pour changer de train ou repasser en vue d'ensemble et la touche N pour revenir à la vue précédente.

V. Modélisation

Nous avions à notre disposition un graphe (API) contenant des points 3D et des arcs bien entendu. L'API fournie contenait aussi quelques méthodes utilitaires. La base de données est peuplée des classes Sommet, Arc, PointAnnexe et Graphe ainsi que des relations qui les lient. Nous avons interprété graphiquement ces relations et y ajouter nos éléments.

- Un graphe connait ses composantes : list_arc, list_sommet, list_point_annexe.

Nous avons donc décidé de mettre ses composantes en attribut de la classe graphe.

- Un point annexe porte ses coordonnées et il en est de même pour Sommet.

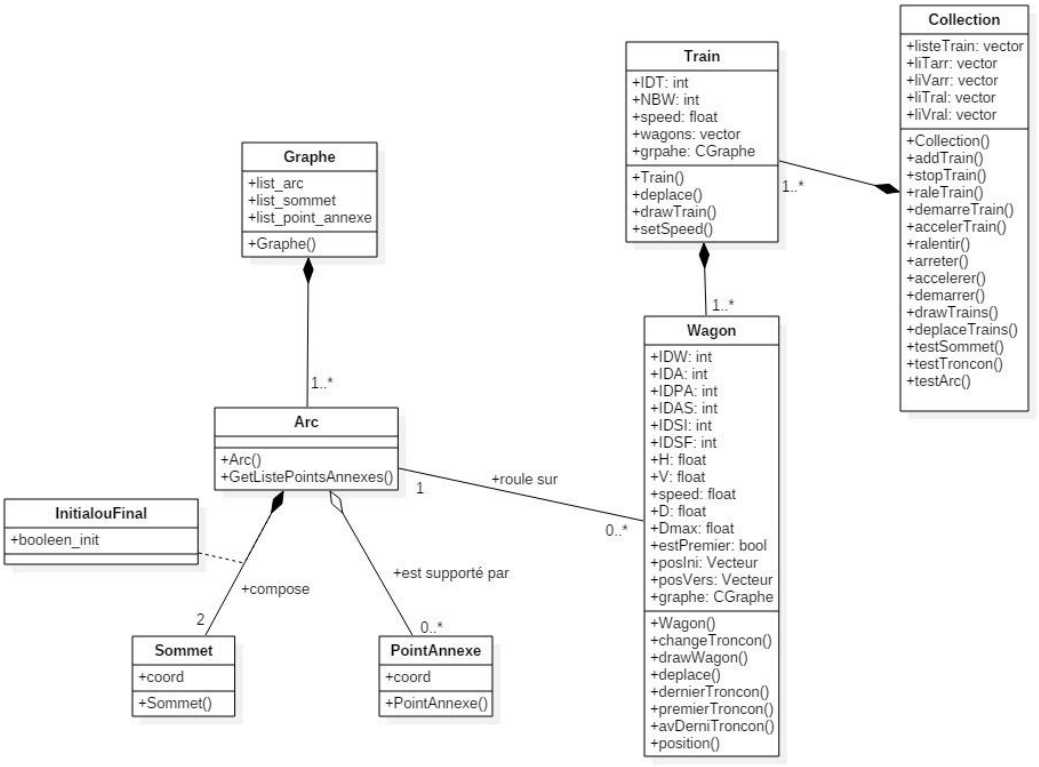
De même nous les avons mis en attributs.

- Un sommet connait ses arcs entrants et sortants, de la même manière tout arc connait ses sommets initial et final.

Grâce à une classe d'association.

- Enfin, un arc connait la liste des points annexes qu'il supporte.

Grâce à une fonction de la classe Arc.



L'hypothèse suivante « on considère que le graphe est non orienté et que les mobiles peuvent circuler dans les deux sens » permet de s'affranchir de la classe d'association « InitialouFinal ».

Nous avons ajouté les classes Collection, Train et Wagon. Un train est composé de wagon et l'ensemble des trains forme la collection. Afin d'éviter les problèmes lors des changements d'arc, chaque wagon est associé à un arc. La collection gère le déplacement des trains (qui gèrent le déplacement de leurs wagons) afin d'éviter les collisions. Elle gère aussi le dessin des trains (qui gèrent le dessin de leurs wagons) pour que tous apparaissent en même temps.

VI. Fonctions du code

#. Représentation tridimensionnelle du graphe comprenant la représentation des sommets (sous forme de sphère) et des arcs (dans une autre couleur)

Graphic::DessinerVoie() appelée par Graphic::Dessiner() dans main.cpp

#. Déplacement des mobiles

Wagon::changeTroncon() appelée par Wagon::deplace() appelée par Train::deplace() appelée par Collection::deplaceTrains() appelée par main::rotate(int) appelée par glutTimerFunc(mSPF,rotate,0) dans main.cpp .

#. Les mobiles sont des trains (ensemble articulé d'au moins deux wagons)

Création d'une classe Train composé de wagons dont on choisit le nombre lors de la création d'un objet train.

#. Gestion des collisions:

L'ensemble de la gestion du mouvement des trains est géré par la classe Collection qui a été créée uniquement dans ce but.

#. Interaction clavier/souris

La classe Camera gère l'interaction clavier/souris.

VII. Sources

OpenGL :

https://www.opengl.org/sdk/docs/man2/

Tuto explicatif en français (bien détaillé) :

http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/opengl2glut.html

Lumières :

http://www.glprogramming.com/red/chapter05.html

Forum d'aide souvent utilisé :

http://stackoverflow.com/search?q=glut

Projet des années précédentes