

Rapport de projet OpenGL

Parc d’attractions

Élève :

Maylis Teyssendier De La Serve

Hugo Baltz

Maxime Siret

#### Table des matières

[I. Introduction 4](#_Toc450493532)

[II. Présentation d’OpenGL 4](#_Toc450493533)

[1. Description 4](#_Toc450493534)

[2. Projection 5](#_Toc450493535)

[III. Manuel d’installation 5](#_Toc450493536)

[IV. Manuel d’utilisation 5](#_Toc450493537)

[V. Modélisation 6](#_Toc450493538)

[VI. Fonctions du code 7](#_Toc450493539)

[VII. Sources 7](#_Toc450493540)

1. 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 « parc d’attractions », c’est-à-dire ici un ou plusieurs trains 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.

1. Présentation d’OpenGL

## 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.

## 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.

1. Manuel d’installation

*Copier le dossier, installer les dll et lancer l’exécutable.*

Voici la liste des tâches à effectuer pour installer le projet :

1) Mettre à jour les pilotes de sa carte graphique (voir site constructeur)

2) Installer Code::Block avec le compilateur MinGW : ici

3) Le programme doit être installée par défaut dans :

C:\Program Files (x86)\CodeBlocks » Copier-coller le contenu du répertoire « data\a\_copier » dans le répertoire MinGW de Code::Blocks :

C:\Program Files (x86)\CodeBlocks\MinGW Cela installe les librairies Glut32, SDL et SDL\_image nécessaires au projet.

4) Vous pouvez ensuite ouvrir le projet (fichier « src\TchouTchou.cbp ») dans Code::Blocks.

1. Manuel d’utilisation

Il faut tout d’abord lancer l’exécutable « .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.

1. 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.*

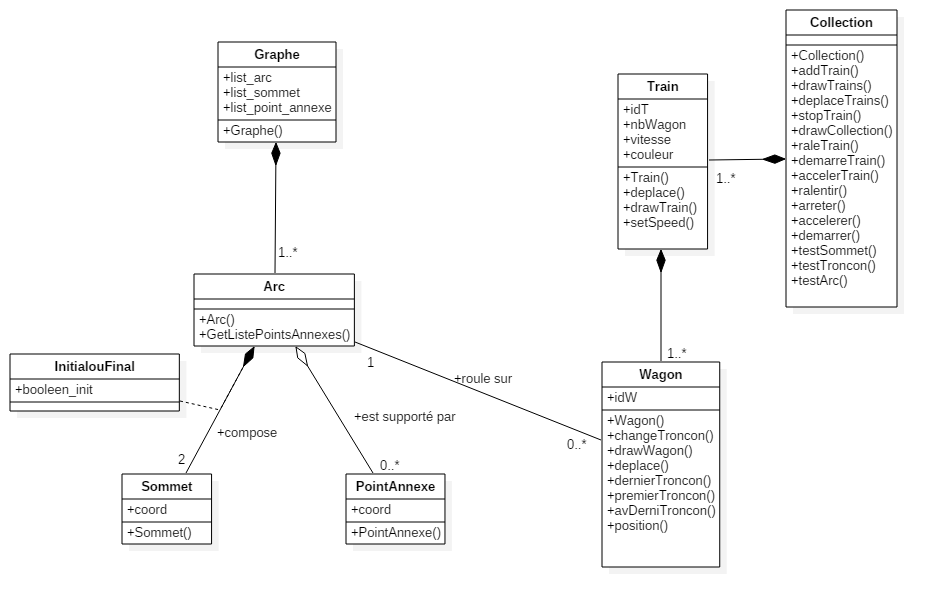
De même nous les avons mis en attributs.

*- 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.

1. Fonctions du code

#. Représentation tridimensionnelle du graphe

Triangulation.

#. Représentation des sommets (sous forme de sphère)

#. Et des arcs (dans une autre couleur)

#. Déplacement des mobiles

#. Les mobiles sont des trains (ensemble articulé d’au moins deux cubes)

#. Gestion des collisions:

- Les mobiles doivent s’attendre lorsqu’ils se dirigent vers le même sommet

- Ils doivent éviter le nez à nez

- Une distance de sécurité doit être respectée lorsque deux mobiles occupent successivement les mêmes arcs.

#. Interaction clavier/souris

1. Sources

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

Projets des années précédentes.