Rapport IGSD :

Introduction :

J'ai réalise le projet seul et j'ai implémenter toutes les fonctionnalités « de base ». Le projet est prévu pour fonctionner sur Linux.

Je n'ai pas modifié le code d'un TP pour arriver au résultats final, j'ai préféré commencer un projet du début afin d'être sur de comprendre l'intégralité du code. Je suis naturellement partie sur de la programmation orienté objet pour plus de facilité.

Mon idée étais de créer une sorte de « moteur de rendu » et ensuite de m'en servir pour réaliser le projet. J'ai donc créé une classe capable de dessiner les données que je lui donne sous forme de VAO.

J'ai rencontré des difficultés dans la création d'un Makefile car c'est un domaine que je ne maîtrisais pas encore correctement. Et avec une trentaine de fichier impossible de faire les commande a la main. J'ai aussi passé beaucoup de temps a fabriquer des « outils » et j'ai manqué de temps pour finir le projet autant que je l'aurais voulu.

J'ai utilisé git et github pour gérer les versions du projet.

Données :

```cpp
#include "csvreader.h"
#include "equipe.h"
#include "match.h"

class Data
{
private:
    std::map<std::string, Equipe> m_listeEquipe;
    std::vector<Match> m_listeMatch;
    std::vector<std::string> m_listeNom;

    Equipe getEquipeFromLine(const std::vector<std::string> &line);
    Match getMatchFromLine(const std::vector<std::string> &line);

public:
    Data();

    const std::map<std::string, Equipe> &getEquipes() const;
    const std::vector<std::string> &getNoms() const;
};
```

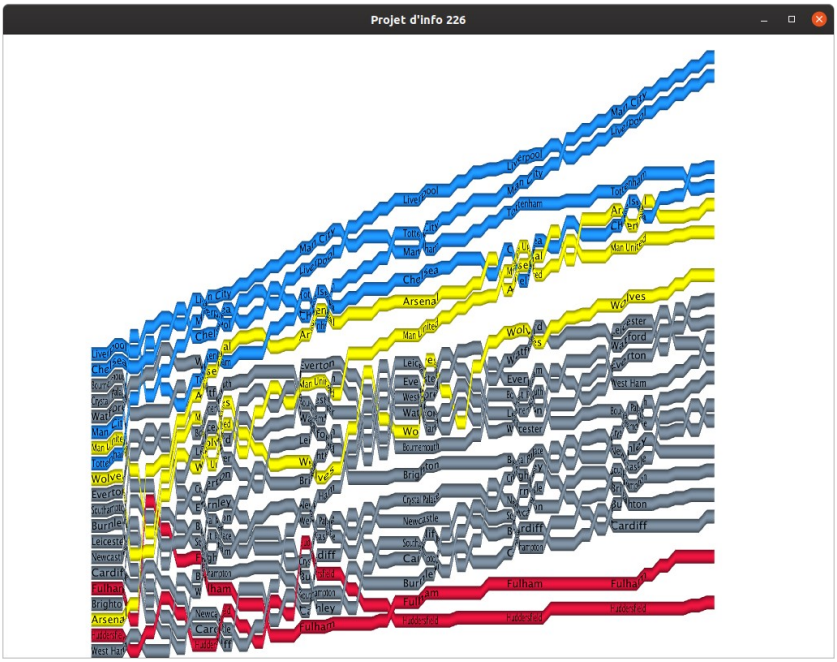
Classe Data

J’ai créé une classe Data pour modéliser les donné. Elle contiens une map d’« Equipe » et une liste de « Match ». Une Équipe contiens le suivi de son classement et de son score au cours des journée ainsi que les index de ses match dans la liste. Un Match contiens le nom des équipes ce qui permet de les retrouver dans la map et les scores.

Le modèle est rempli grâce a la classe « CVSReader » qui lit les CVS et les transformes en tableaux de string en 2 dimensions. Il ne suffit alors que de les parcourir ligne par ligne.

J’ai tous fait pour faciliter la modularité de l’accès des données et la possibilité d’en rajouter pour les parties « aller plus loin ».

Modèles 3D :



Capture d’écran de rendu

J’ai construit la classe « ModelBuilder » qui me permet de remplir un VBO. J’ai construit différentes structures de donnée pour facilité cette tache. Des triangles, quads et lines qui me permettent de faciliter la manipulation des tableaux de floats ainsi que les opération entre eux, notamment l’addition de quads.

Sur chaque lignes on procède segment par segment pour plus de modularité et un bon contrôle des directions, de la courbure de la courbe, des courbes qui se passe les une les autres.

Les lines sont des demis cylindres, une ligne qui monte passe toujours par l’avant et par derrière quand elle descend.

Puisque elle remplie les VBO c’est aussi cette classe qui gère la couleur et la texture. La couleur dépend de la position final de l’équipe et la texture avec le nom de l’équipe appairait sur tous le long de la courbe.

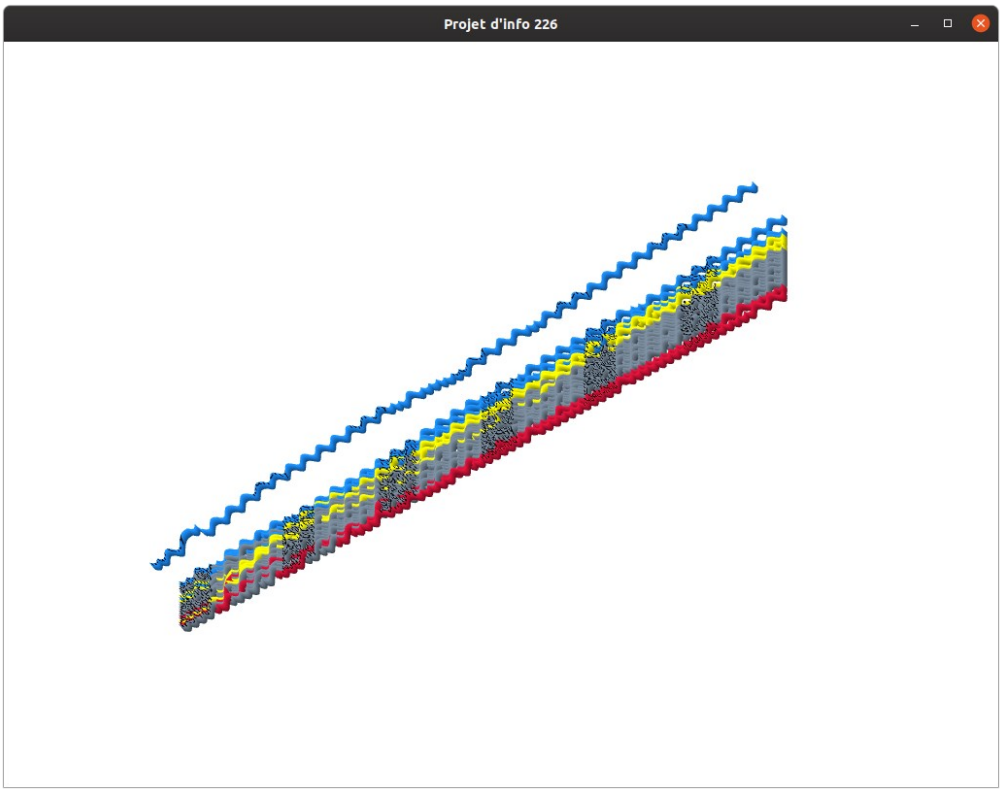
Je n’ai malheureusement pas eu le temps d’implémenter l’effet des coins arrondi.

Couleurs et ombrages :

Zoom sur la capture d’écran

L’« ombrage » est en faite géré au niveau de la couleur. Une fonction sinus permet de créé un effet plus sombre sur les bord.

Interaction :



Autre angle du rendu

Les touche ZQSD permettent de faire tourner le modèle autour de son centre. Je n’ai malheureusement pas réussi a utiliser la fonction de camera a cause d’un bug. Vous aviez vous mèmes tenté de m’aider sans résultats.

Les flèches du haut et du bas permettent de sélectionner une équipe qui sera alors afficher devant toutes les autres (ex : l’image si dessus)